



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY



FACULTAD DE
INGENIERÍA

Actualización y Expansión de PSPcode: Un enfoque integral

Informe de Proyecto de Grado presentado por

Gustavo Samir Audi Rouco, Lía Malvárez Busquets

en cumplimiento parcial de los requerimientos para la graduación de la carrera de
Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Silvana Moreno

Montevideo, 14 de febrero de 2024



Actualización y Expansión de PSPcode: Un enfoque integral
por Gustavo Samir Audi Rouco, Lía Malvárez Busquets tiene licencia [CC](#)
[Atribución 4.0](#).

Agradecimientos

En primer lugar, deseamos expresar nuestro agradecimiento a nuestra tutora Silvana Moreno, quien no solo nos presentó el proyecto con entusiasmo, sino que, gracias a su profundo interés y conocimiento como responsable del curso PF-PSP, siempre estuvo disponible para responder consultas, validar la aplicación y brindarnos orientación fundamental para el éxito de este proyecto.

Extendemos nuestro reconocimiento a la docente Leticia Pérez, quien se involucró de manera colaborativa en el proceso. Su implicación en las fases de prueba y reporte de defectos fue invaluable para garantizar la calidad y fiabilidad de la aplicación.

Asimismo, deseamos expresar nuestro sincero agradecimiento a los estudiantes del curso PF-PSP 2023. Su participación activa como usuarios de PSPcode y la valiosa devolución que nos proporcionaron al final del curso fueron esenciales para el resultado obtenido.

Por último, pero no menos importante, queremos agradecer a nuestras familias y amigos por su constante apoyo, ayuda y comprensión. Su aliento y respaldo fueron indispensables para superar los desafíos y completar este proyecto satisfactoriamente.

Resumen

El presente proyecto se desarrolla en el contexto del curso Principios y Fundamentos del Proceso de Software Personal (PF-PSP) de la Facultad de Ingeniería, Udelar. El objetivo principal de este trabajo incluye la actualización, expansión y mejora de PSPcode. PSPcode es una herramienta desarrollada para la gestión de los proyectos llevados a cabo por los estudiantes del curso, siguiendo la metodología PSP (Personal Software Process).

En el marco de este proyecto, se introdujeron nuevas funcionalidades, tales como sugerencias automáticas de evaluación y la capacidad de corregir proyectos directamente dentro de la aplicación. A su vez, PSPcode se migró a un nuevo servidor de alojamiento, seleccionado cuidadosamente para optimizar los costos y garantizar una mayor estabilidad y escalabilidad a largo plazo.

Tras un proceso de validación en entorno de prueba, PSPcode fue liberada oficialmente para su utilización en el curso PF-PSP de 2023.

El proyecto culminó con una evaluación de la aplicación a través de un cuestionario dirigido a los estudiantes, con el objetivo de recopilar opiniones y experiencias detalladas.

Palabras clave: PSP, PSPcode, Aplicación Web, Desarrollo de Software, Programación, Base de datos, Enseñanza

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Trabajo Realizado	2
1.4. Organización del Documento	3
1.4.1. Capítulo 2: Revisión de Antecedentes	3
1.4.2. Capítulo 3: Gestión de Proyecto	3
1.4.3. Capítulo 4: Actualización y Refactorización de la aplicación	3
1.4.4. Capítulo 5: Implementación de nuevas funcionalidades	3
1.4.5. Capítulo 6: Pruebas y Despliegues	3
1.4.6. Capítulo 7: Conclusiones y Trabajo Futuro	4
2. Revisión de antecedentes	5
2.1. Personal Software Process	5
2.1.1. Introducción	5
2.1.2. Los principios del PSP	5
2.1.3. Versiones y estructura del PSP	6
2.1.4. Fases del PSP	7
2.1.5. Versiones del PSP	7
2.2. Principios y Fundamentos del Proceso de Software Personal	9
2.2.1. Desarrollo del curso	9
2.2.2. Aplicación del PSP	10
2.2.3. Objetivos del curso	11
2.3. Marco teórico técnico	11
2.3.1. Backend	11
2.3.2. Frontend	16
2.4. Herramienta web: PSP code	20
2.4.1. Enumeración de funcionalidades	20
2.5. Estado técnico en el que se heredó PSPcode	22
2.5.1. Aplicación Backend	22
2.5.2. Aplicación Frontend	23
3. Metodología de trabajo	25
3.1. Plan y desarrollo cronológico del proyecto	25
3.1.1. Plan Inicial	26
3.1.2. Desarrollo cronológico del Proyecto	27
3.2. Gestión y Comunicación	28
3.2.1. Comunicación con la Tutora	28
3.2.2. Herramientas de Gestión de Proyecto	28

4. Actualización y Refactorización de la aplicación	30
4.1. Aplicación Backend	30
4.1.1. Ajustes y Desafíos previos al Desarrollo	30
4.1.2. Desarrollo y Actualización del código	31
4.2. Aplicación Frontend	34
4.2.1. Proceso de Actualización de Librerías	35
4.2.2. Optimización de la aplicación	39
4.2.3. Corrección en la configuración de Webpack para un build exitoso	41
4.2.4. Mejoras estéticas y funcionales de la interfaz de usuario	42
5. Implementación de nuevas funcionalidades	43
5.1. Análisis de criterios de corrección automatizables	43
5.1.1. Determinación de criterios automatizables	44
5.1.2. Definición de tareas de desarrollo para las automatizaciones	45
5.2. Implementación de observaciones automáticas	46
5.2.1. Tareas Backend	46
5.2.2. Tareas Frontend	53
5.3. Project Feedback	66
5.3.1. Frontend	67
5.3.2. Backend	74
5.4. Cambios en la plataforma de Admin	77
6. Pruebas y Despliegues	78
6.1. Entorno de Pruebas	78
6.2. Entorno Productivo	78
6.2.1. AWS	81
6.2.2. Digital Ocean	81
6.3. Defectos reportados en etapa de prueba	82
6.4. Defectos reportados en etapa de producción	83
6.4.1. Descripción de errores y soluciones	84
6.4.2. Distribución de errores por fecha	89
6.5. Pruebas de Aceptación de usuario	89
6.5.1. Estructura de la encuesta	90
6.5.2. Resumen de resultados	91
7. Conclusiones y Trabajo Futuro	100
7.1. Conclusiones	100
7.2. Trabajos a futuro	101
Anexo 1 - Revisión de Antecedentes	105
.1. Frontend	105
.1.1. package.json inicial	105
.2. Backend	108
.2.1. Modelo de dominio de la aplicación previa a los cambios	108
.2.2. Modelo de dominio parcial sobre la pestaña de correcciones	109
Anexo 2 - Actualización y Refactorización de la aplicación	110
.3. Frontend	110
.3.1. Wbpack v4 y React 16.8 package.json	110
.3.2. Cambios en la versión de Babel	112
.3.3. Transición detallada de componentes de clase a funcionales	113
.3.4. Eliminación de código muerto	118
.3.5. Versión final del package.json	120

.3.6.	Identificación y propuesta de mejoras en UX/UI	122
.3.7.	Mejoras estéticas y funcionales de la interfaz de usuario	124
.4.	Grading original	137
.4.1.	Criterios escritos y clasificados como se encuentran en la <i>grading</i> original:	137
.5.	Backend	140
.5.1.	Manual de uso del Admin para el flujo de Project Feedback	140
.5.2.	Edición de Fase	143
.5.3.	Base de datos	143
Anexo 3 - Testing		148
.6.	Bugs reportados por las docentes en etapa de Testing (ambiente de staging)	148
.7.	Bugs y problemas encontrados en etapa de Producción	149
Anexo 4 - Despliegues y Monitoreo		150
.8.	Archivo de configuración de Docker	150
.9.	Script de monitoreo de estado de la aplicación	151
.10.	Firewalls configurados en cada uno de los Droplets	152

Capítulo 1

Introducción

1.1. Motivación

Desde el 2012 se dicta el curso Principios y Fundamentos del Proceso de Software Personal (PF-PSP) en la Facultad de Ingeniería, UdelaR. En este curso, los estudiantes deben desarrollar ocho proyectos de programación siguiendo la metodología de trabajo PSP (Personal Software Process), la cuál establece un proceso estricto y disciplinado apuntando a la eficiencia y calidad del desarrollo individual.

Desde 2018, el curso PF-PSP utiliza una herramienta de apoyo llamada PSPcode, construída por Guillermo Kuster y Guillermo Tavidian en el marco de su Proyecto de Grado. Dicha herramienta facilita a los docentes y estudiantes del curso el desarrollo y la gestión de los proyectos que se deben llevar a cabo. La aplicación se construyó de manera exclusiva para las necesidades de este curso.

Durante un lapso de cinco años, la aplicación no recibió mantenimiento, y se encontraba alojada en Heroku (Sección [2.3.1.3](#)) bajo un plan que imponía restricciones significativas, entre ellas, almacenamiento limitado en la base de datos. Este contexto, sumado a la propuesta de las docentes del curso PF-PSP de incorporar nuevas funcionalidades a la aplicación, generó la necesidad de emprender un proyecto integral de actualización y extensión.

1.2. Objetivos

El propósito central del presente proyecto es abordar la situación actual de PSPcode mediante un proceso de actualización, refactorización y expansión. Este enfoque no solo tiene la intención de resolver las limitaciones y restricciones existentes, sino también de introducir nuevas funcionalidades que enriquezcan la experiencia tanto para docentes como para estudiantes. Entre las nuevas características planificadas, se encuentra la capacidad de corregir directamente los proyectos presentados por los estudiantes dentro de la propia aplicación, brindando así una solución integral para el proceso de evaluación.

1.3. Trabajo Realizado

En el transcurso del proyecto, se llevó a cabo una serie de actividades con el propósito de enaltecer y ampliar la aplicación PSPcode. La iniciativa comenzó con una lectura y análisis del informe del Proyecto de Grado que sentó las bases para la versión origen: PSPcode v1 (2017 - 2018). A partir de esta revisión, y el uso de la aplicación a modo de exploración y prueba, se delineó un plan estratégico para abordar áreas de mejora identificadas.

Existían limitantes de espacio en la base de datos de la versión original de la aplicación, previamente identificados por la tutora del proyecto, Silvana Moreno. Esta problemática impediría el uso de PSPcode en 2023, por lo que se priorizó la tarea de investigación en pos de prolongar la vida útil de la aplicación. Como resolución del análisis, se llevó a cabo un cambio de plan en el servidor de Heroku para garantizar una mayor capacidad y eficiencia a futuro.

Simultáneamente, se procedió a la inicialización de dos nuevos repositorios, uno para el Backend, construido en Ruby on Rails (Sección 2.3.1.1), y otro para el Frontend, desarrollado en React (Sección 2.3.2.2). Esta acción estableció la base para futuras actualizaciones y desarrollo.

La fase inicial incluyó la actualización de librerías y la realización de refactorizaciones significativas en ambos repositorios. Estos esfuerzos se concentraron en la reutilización de código, la eliminación de código muerto y la modernización de sintaxis obsoletas, contribuyendo así a un código más eficiente, legible y mantenible. A su vez, se hicieron modificaciones en la interfaz de usuario para hacerla más estética, fluida e intuitiva.

Uno de los logros más destacados fue la ampliación de funcionalidades en la aplicación, como sugerencias automatizadas para facilitar la revisión de proyectos, junto con una nueva pantalla que permitió a los profesores evaluar y corregir proyectos directamente dentro de la aplicación.

La validación de estas nuevas características se llevó a cabo en un entorno de pruebas dedicado (*staging*), con la participación activa de las docentes. Posteriormente, la herramienta se liberó oficialmente en un nuevo entorno de AWS (Sección 2.3.1.6), para su uso en el curso PF-PSP durante el primer semestre de 2023. Los usuarios finales tuvieron la oportunidad de utilizarla, lo que dio inicio a una fase de soporte y resolución de problemas.

Con el objetivo de recopilar experiencias y opiniones, se diseñó y suministró un cuestionario a los estudiantes para obtener su percepción de la nueva versión de la aplicación, dando especial énfasis a las funcionalidades incorporadas. Este análisis proporcionó información valiosa para evaluar el impacto y la eficacia de las mejoras introducidas.

Finalmente, considerando la eficiencia económica y la proyección a largo plazo, se realizó un cambio de ambiente, migrando la aplicación a Digital Ocean (Sección 2.3.1.7), una alternativa que ofrece las mismas características técnicas que AWS a un costo significativamente menor. Este cambio aseguró la estabilidad, escalabilidad y viabilidad económica de la aplicación a largo plazo.

1.4. Organización del Documento

1.4.1. Capítulo 2: Revisión de Antecedentes

En este capítulo se proporciona el contexto necesario para la comprensión de los contenidos del informe. En este caso, se introduce al *Personal Software Process* (Proceso Personal de Software), al curso de la Facultad de Ingeniería “Principios y Fundamentos del Proceso Personal de Software”, y a la herramienta PSPcode original incluyendo el estado técnico que en el que se heredó. A su vez, también se incluye el marco teórico técnico necesario para el entendimiento de las principales tecnologías empleadas en la aplicación.

1.4.2. Capítulo 3: Gestión de Proyecto

En el Capítulo 3 se presenta la metodología de trabajo a lo largo del proyecto, incluyendo descripción cronológica de las actividades realizadas, así como cualquier desviación sucedida con respecto al plan inicial del proyecto. También se describe la forma de comunicación con la tutora y herramientas utilizadas para la gestión interna.

1.4.3. Capítulo 4: Actualización y Refactorización de la aplicación

En este capítulo se describe el proceso de actualización de librerías estableciendo una comparación de las versiones iniciales y finales de los repositorios. A su vez, se enumeran y explican los cambios realizados en el código con el fin de reformarlo haciéndolo más compacto, legible y moderno. El capítulo se divide en dos secciones: Aplicación Backend y Aplicación Frontend.

1.4.4. Capítulo 5: Implementación de nuevas funcionalidades

El Capítulo 5 está dedicado a aquellas funcionalidades que fueron analizadas, diseñadas e implementadas exclusivamente para este proyecto. Describe las etapas del desarrollo de cada *feature* vistas desde la óptica del Frontend y del Backend.

A grandes rázagos, las funcionalidades descriptas son:

- Observaciones (correcciones) automatizadas
- Página de Corrección (o Project Feedback)
- Cambios en la plataforma de Admin aumentando el poder de configuración del docente

1.4.5. Capítulo 6: Pruebas y Despliegues

Este capítulo refiere a la experimentación del proyecto, describiendo las características técnicas de los ambientes de liberación, como también los resultados de pruebas hechas en cada ambiente. En éste capítulo se incluye un registro de los defectos encontrados en la aplicación en cada etapa y cómo fueron resueltos. A su vez, se aborda el análisis del resultado del cuestionario realizado a los usuarios de rol Estudiante como prueba de aceptación.

1.4.6. Capítulo 7: Conclusiones y Trabajo Futuro

Como cierre al cuerpo principal del informe, el Capítulo 7 aborda las conclusiones obtenidas como fruto del trabajo integral del proyecto y la experiencia de desarrollo con la aplicación PSPcode. Se presentan también, posibles puntos de mejora o extensión a este proyecto como trabajo futuro.

Capítulo 2

Revisión de antecedentes

2.1. Personal Software Process

2.1.1. Introducción

El Personal Software Process (PSP) propuesto por Watts S. Humphrey en 1993 [1], es una metodología estructurada para el desarrollo de software a nivel personal, que tiene como objetivo mejorar la calidad, productividad y habilidades de los programadores en su trabajo individual. El proceso proporciona una serie de fases a seguir y un conjunto de técnicas para gestionar y controlar el desarrollo de software a nivel personal.

Lo que motivó al creador a idear esta metodología, fue la siguiente interrogante: “¿Los principios de mejora de procesos, serían efectivos para el desarrollo de software individual?”

Dichos fundamentos han dado resultado en organizaciones y grandes proyectos, pero nunca se plantearon a modo de proceso definido para el desarrollo de software individual. Antes de comenzar activamente a trabajar en la calidad del software y los procesos que la garantizan, Humphrey se desarrolló más de 30 años en el área de la ingeniería de software.

El PSP sienta sus bases en los principios de gestión de proyectos introducidos por Frederick Winslow Taylor hace más de 100 años [2]. Él fue el primero en ver el trabajo manual de las personas como algo analizable y potencialmente mejorable a partir del estudio de datos medibles, para el aumento de eficiencia y eficacia de las tareas.

Además de los principios de Taylor, el PSP también considera la selección, planificación y el orden de tareas. Muestra a los desarrolladores cómo trabajar bajo un proceso definido, medido y planificado para mejorar el desempeño personal.

2.1.2. Los principios del PSP

- Cada ingeniero es diferente; para ser más efectivos, los ingenieros deben planificar su trabajo y estudiarse para basar sus planes en sus propios datos personales.
- Para mejorar consistentemente su desempeño, los ingenieros deben usar procesos medidos y bien definidos.
- Para producir productos de calidad, los ingenieros deben sentirse personalmente responsables de la calidad de sus productos. Los productos excelentes no se producen

por error. Los ingenieros deben esforzarse por hacer un trabajo de calidad.

- Hacer las cosas de forma correcta, siempre es la manera más rápida y económica de hacer un trabajo.
- Cuesta menos encontrar y corregir defectos en etapas tempranas, que finalizando un proceso.
- Es más eficiente prevenir defectos que encontrarlos y corregirlos.

“Para producir productos de calidad de manera constante, los desarrolladores deben planificar, medir y realizar un seguimiento del producto. A su vez, deben enfocarse en la calidad desde el momento cero de un trabajo. Finalmente, deben analizar los resultados de cada proyecto y utilizar estos hallazgos para mejorar sus procesos personales” -Watts S. Humphrey. [1]

2.1.3. Versiones y estructura del PSP

El PSP tiene siete versiones incrementales desarrolladas, todas con su propia estructura y conjunto de datos a registrar y analizar.

A continuación, se presenta el proceso original base del PSP. Este proceso, en todas sus versiones, está estructurado en un conjunto de fases a ejecutarse secuencialmente por el desarrollador, como se puede ver en la figura 2.1.

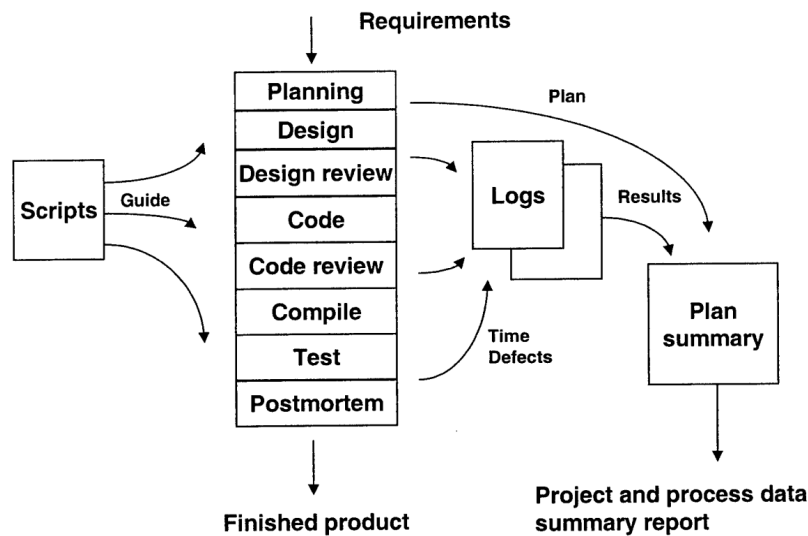


Figura 2.1: Ejemplo basado en el PSP 2, versión en la cual son agregadas las fases de design y code review.

Fuente: “The Personal Software Process: Status and Trends”, Watts S. Humphrey, 2000 [1]

Cabe destacar que esta es la estructura completa del PSP, a efectos del curso PF-PSP la misma puede verse alterada, dichas diferencias serán descritas en la Sección 2.2. Por esta razón, también se especificarán los datos a registrar, y las actividades a realizar en cada fase cuando se describa la aplicación del PSP en el curso.

2.1.4. Fases del PSP

Plan: La fase de planificación se usa para documentar el plan y estimación del producto a construir (ejercicio de programación).

Desarrollo: En esta etapa se realizan todas las actividades de construcción del programa a entregar. Se subdivide en fases: **Diseño, Codificación, Compilación y Pruebas unitarias**. En el PSP 2.0 se incluyen las fases de Revisión de Diseño y Revisión de Código.

PostMortem: se complementa el plan realizado en la primera fase de acuerdo a las mediciones obtenidas tras construir el producto. Es la etapa de conclusiones y contraste entre lo planeado y lo que sucedió.

El desarrollador tiene la posibilidad de reincidir en fases dentro del desarrollo, por ejemplo diseñar o codificar en varias instancias dentro de un mismo proyecto, respetando ciertas reglas, por ejemplo: Cada vez que se codifica, se debe *testear* después. Se ahonda profundamente en estas reglas en la Sección 2.2.

2.1.5. Versiones del PSP

El objetivo del PSP es guiar a los estudiantes o ingenieros en un proceso de evolución y aprendizaje gradual. A medida que avanzan a través de las diferentes versiones del PSP, se espera que adquieran nuevas habilidades y conocimientos, aplicándolos de manera progresiva en su desarrollo de software. Este enfoque incremental les permite desarrollar una base sólida y mejorar continuamente sus prácticas, a medida que avanzan en su camino de dominio del proceso comenzando por el PSP0, para luego aplicar el PSP0.1, luego el PSP1.1, y así sucesivamente.

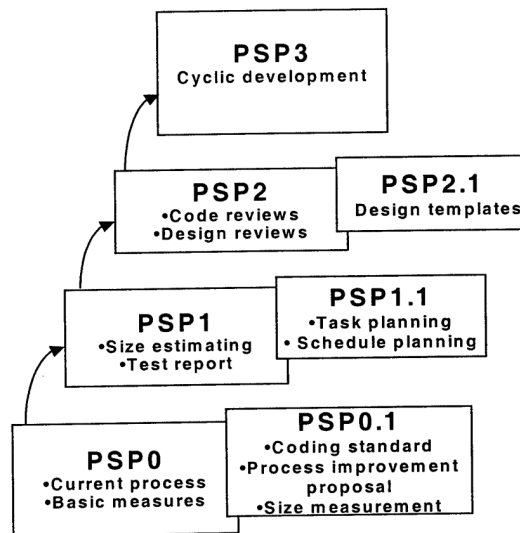


Figura 2.2: Esquema completo de las siete versiones del PSP diseñadas por Humphrey. Cada versión incluye una enumeración concisa de las actividades incorporadas con respecto a la anterior.

Fuente: “The Personal Software Process: Status and Trends”, Watts S. Humphrey, 2000 [1]

2.1.5.1 PSP 0

PSP0 es la versión inicial de PSP y se compone de 5 fases: Planificación, Diseño, Codificación, Compilación, Pruebas Unitarias y Post Mortem. A su vez, se establece una línea base de registro, para llevar a cabo la medición del proyecto en todas las fases: tiempo dedicado a cada fase, defectos inyectados y eliminados en cada fase, y comentarios adicionales.

El PSP0.1 extiende el proceso mediante la adición de un estándar de codificación, y una propuesta de mejora al proceso o PIP (en inglés: Process Improvement Proposal), la cual ayuda al ingeniero a identificar acciones para mejorar su proceso y obtener un mejor desempeño.

2.1.5.2 PSP 1

Como se explicó anteriormente, un desarrollador que comienza a aplicar el PSP lo hace incrementalmente, haciendo uno o dos proyectos bajo cada versión del mismo. En la versión PSP1 (estimación y planificación) el ingeniero registra una estimación en la etapa de *planning*, definiendo cuánto llevará el proyecto en tiempo y líneas de código. Esta estimación estará fuertemente basada en su experiencia con los trabajos previos utilizando PSP.

También se le agrega, a la data registrada de los proyectos, un reporte de casos de prueba y resultados en la etapa de *testing*.

En la versión PSP1.1 se adiciona una planificación más específica y calendarizada de cada fase del proyecto.

2.1.5.3 PSP 2

Para finalizar, en las versiones PSP 2.0 y 2.1, que se enfocan en la gestión de calidad y diseño, se añaden dos fases nuevas (ya vistas en la figura 2.1): Revisión de Diseño y Revisión de Código.

El núcleo de PSP 2.0 es la prevención y corrección de errores. En esta versión del proceso, los ingenieros se capacitan en cómo evaluar y optimizar sus estimaciones, así como en mejorar la calidad de sus productos.

Se utilizan listas de verificación para revisar los diseños y códigos fuente.

Finalmente, PSP 2.1 incorpora técnicas de definición de diseño y análisis rigurosos para minimizar los errores introducidos en el diseño de sus productos.

A modo de conclusión, es importante destacar cómo el PSP está centrado en lograr la mejor calidad de código, utilizando la menor cantidad de recursos (como ser el tiempo), para ello se realizan proyectos y se va mejorando iterativamente el proceso individual del ingeniero, para cumplir el objetivo de generar una estimación acertada, minimizar la cantidad de defectos por etapa, y fundamentalmente conocerse a fondo como desarrollador.

2.2. Principios y Fundamentos del Proceso de Software Personal

A continuación se describirá el curso de la Facultad de Ingeniería para el cual se desarrolló la aplicación PSPcode.

En 2017 (finalizado en 2018) se llevó a cabo el Proyecto de Grado de los estudiantes Guillermo Kuster y Guillermo Tavidian [3], el cual resultó en una aplicación requerida como herramienta para el curso “Principios y Fundamentos del Proceso de Software Personal” (PF-PSP) [4].

El curso PF-PSP es electivo para la carrera de grado de Ingeniería en Computación. Se dicta en semestre impar desde 2012 en la Facultad de Ingeniería. El único año que no se dictó fue 2020, por lo que lleva 11 ediciones hasta la fecha.

El curso tiene una duración de 9 semanas, y los estudiantes reciben 12 créditos para su carrera de grado una vez completado el mismo.

Tiene como materia previa Programación 4, por lo que los estudiantes que cursen PF-PSP se encuentran aproximadamente en 4to año de la carrera de grado. Este es un dato importante para contextualizar la experiencia de los estudiantes, ya que Watts S. Humphrey diseñó PSP originalmente para ingenieros.

El curso se basa, como el PSP en sí mismo, en buscar un proceso que ayude y mejore la calidad del software resultante. En este caso, quienes planean, diseñan y desarrollan los proyectos son estudiantes de la facultad (no aún ingenieros), por lo que este camino se hace más difícil, dada la falta de experiencia y madurez en el rubro.

2.2.1. Desarrollo del curso

El curso tiene el mismo formato cada año: en la primera semana se explica un proceso PSP básico y se presenta la dinámica de los proyectos que se realizan en las siguientes 8 semanas. Cada proyecto tiene como objetivo guiar y ayudar en las tareas de desarrollo de software, para así mejorar individualmente la calidad del producto final y recopilar métricas de producto y proceso [5].

La adaptación del PSP utilizada en el curso consta de las fases: Planificación, Diseño, Codificación, Compilación, Pruebas Unitarias (UT) y Post Mórtem, excluyendo las fases de Design Review y Code Review, propias del PSP original. Los estudiantes cuentan con un conjunto de *scripts* para seguir la metodología. Los *scripts* son una guía que aporta la descripción, y establece las entradas y salidas de los programas a desarrollar.

El trabajo práctico del curso consiste en que cada alumno desarrolle 8 pequeños proyectos siguiendo el modelo de proceso presentado y registrando los datos solicitados en cada fase. Los estudiantes realizan los proyectos de forma individual y secuencial.

Se asigna uno por semana. Antes de iniciar el Proyecto 1, cada alumno deberá elegir el lenguaje de programación que utilizará a lo largo de todo el curso.

Los estudiantes realizan los proyectos como tarea domiciliaria, y tienen un profesor designado, que se encargará de asignarles los distintos proyectos, corregirlos y resolver

dudas.

2.2.2. Aplicación del PSP

El Proyecto 1 se realiza aplicando el PSP0. A partir del Proyecto 2 se utiliza un modelo de proceso adaptado que incorpora tanto las características del PSP0.1, como algunas del PSP1.

En cada fase de un proyecto, los estudiantes deben registrar:

- Nombre de la fase (entre las posibles fases del PSP).
- La fecha y hora de inicio y fin de cada fase.
- Tiempo de interrupción: el tiempo (en minutos), entre el inicio y fin registrado, mientras el estudiante no trabajó activamente.
- Datos sobre los defectos encontrados en cada fase:
 - Fecha y hora de hallazgo
 - Fase en la que fue inyectado
 - Tipo de defecto (entre los posibles, definidos por los docentes del curso)
 - Fecha y hora en la que fue corregido
 - Descripción del defecto
- Comentarios adicionales que se quieran hacer en cada fase.

A partir del Proyecto 2, dado el cambio en la versión del PSP, los estudiantes deben realizar nuevos registros: En la fase de Plan, deben registrar un número estimado de LOCs (líneas de código), y en la fase de Post Mortem, se agrega el valor final de LOCs. Además, en esta última fase, deben completar los campos del PIP (Propuesta de Mejora al Proceso), los cuales son:

- **Descripción del Problema:** espacio para que los estudiantes detallen un problema a resolver con respecto al proceso.
- **Descripción de la Propuesta:** donde pueden proponer soluciones al problema descrito.
- **Notas Adicionales:** sección donde pueden agregar comentarios adicionales con respecto a la propuesta de mejora al proceso (PIP).

En particular, el Proyecto 2 es diferente a los demás, ya que los estudiantes deben construir un software de medición de tamaño, mientras que en los demás proyectos deben programar soluciones matemáticas. Los datos del Proyecto 2 se excluyen del análisis estadístico, ya que tiene mayor dificultad que los demás proyectos.

Hasta el curso de 2017 (inclusive), se utilizaba la herramienta Process Dashboard [6] de SEI (Software Engineer Institute). Dicha herramienta utilizaba el software Microsoft Access, lo que impedía su compatibilidad con Mac OS, esta fue una de las motivaciones para el desarrollo de la aplicación web PSPcode, utilizada como herramienta del curso a partir de 2018.

2.2.3. Objetivos del curso

Además de construir conocimiento y experiencia del PSP en los estudiantes como ya fue mencionado, los docentes tienen objetivos de investigación planteados [7][8][9]. Estos objetivos incluyen conocer el esfuerzo dedicado (tiempo registrado por cada estudiante) al diseño detallado y a la codificación, la calidad del producto de software desarrollado y la calidad del diseño elaborado por los estudiantes, entre otros.

La evidencia fue generada a partir del análisis y documentación de datos de distintas iteraciones del experimento, cada una correspondiente a un grupo de instancias de ejecución del curso divididas de la siguiente manera: 2012-2013-2014, 2015-2016-2017, 2018, 2019-2021.

Desde 2012 hasta hoy, el PF-PSP fue evolucionando acorde a los resultados de los experimentos previos. Para asegurar la calidad de los datos registrados, los estudiantes que ya han participado en una instancia del curso no pueden volver a tomarlo.

2.3. Marco teórico técnico

Esta sección está dedicada a presentar una explicación teórica de las principales tecnologías empleadas en este proyecto. Se abordan de manera concisa y clara las herramientas técnicas fundamentales que sustentan el desarrollo de la aplicación. Esta descripción servirá como base para comprender las decisiones y soluciones adoptadas en el proyecto, brindando al lector una visión clara del entorno tecnológico que subyace en su implementación.

2.3.1. Backend

A continuación se abordarán las tecnologías específicas empleadas en el desarrollo del Backend de la aplicación, así como las herramientas y servicios esenciales que contribuyeron a la construcción de la infraestructura. Este marco teórico sienta bases conceptuales necesarias para comprender la arquitectura y el funcionamiento del sistema.

2.3.1.1 Ruby

Ruby [10], creado por Yukihiro “Matz” Matsumoto, es un lenguaje de programación que integra características de Perl, Smalltalk, Eiffel, Ada y Lisp, enfocado en ser “natural, no simple”. Lanzado en 1995, ha ganado popularidad significativa, destacándose por el *framework* Ruby on Rails [11], un marco de desarrollo web que simplifica y agiliza la creación de aplicaciones.

Matz diseñó Ruby con una sintaxis orientada a objetos, donde todo es un objeto, permitiendo una amplia manipulación del lenguaje. Un ejemplo de su flexibilidad es la posibilidad de redefinir operaciones como la suma:

```
class Numeric
  def sumar(x)
    self.+(x)
  end
end
```

```
y = 5.sumar 6
# y es igual a 11
```

Ruby se distingue por características como bloques flexibles, el concepto de “mixin” para incorporar métodos a clases, y una sintaxis con poca puntuación y convenciones claras para nombrar variables (e.g., **var** para variables locales, **@var** para variables de instancia).

Además, Ruby cuenta con manejo de excepciones, un *garbage collector*, la capacidad de escribir extensiones en C, manejo de hilos y portabilidad. Su extensa biblioteca de gemas, gestionadas por RubyGems [12], incluye el notable *framework* Rails [11].

2.3.1.2 SendGrid

SendGrid [13], fundada en 2009 y adquirida por Twilio [14] en 2019, es una plataforma en la nube que aborda la entrega de correos electrónicos en entornos empresariales. Hasta 2017, recaudó más de 81 millones de dólares y se expandió a varias ciudades, incluyendo una oficina en Londres.

La plataforma ofrece avanzadas capacidades de envío de correo electrónico, como direcciones IP dedicadas y manejo de subusuarios, a través de su API y servidor SMTP. Está diseñada para manejar correos transaccionales, cumpliendo con normas *antispam* y administrando diferentes tipos de correos.

SendGrid mejora la recepción de notificaciones importantes, aportando funcionalidad y seguridad a los sistemas. Además, provee informes de diagnóstico y validación de direcciones de correo electrónico.

Las ventajas de SendGrid incluyen facilidad de integración, seguimiento de usuarios, ahorro de costos al evitar servidores propios, y eficacia en el *email marketing*.

No obstante, presenta desafíos como verificación de correos no infalible, documentación compleja, informes limitados sobre interacción de usuarios e incompatibilidad con aplicaciones móviles.

2.3.1.3 Heroku

Fundada en 2007 y adquirida por Salesforce en 2010, Heroku [15] es una plataforma como servicio (PaaS) que soporta múltiples lenguajes de programación como Ruby, JavaScript, Java, Python, y PHP. Heroku utiliza contenedores denominados Dynos, basados en Linux, para ejecutar aplicaciones, garantizando el aislamiento y la seguridad entre ellos. Además, ofrece acceso a más de 200 complementos para extender las funcionalidades de las aplicaciones.

Los principales productos de Heroku incluyen:

- **Heroku Platform:** Ejecución de aplicaciones en Dynos, soporte para múltiples lenguajes y escalabilidad instantánea.
- **Heroku Postgres:** Servicio de base de datos basado en PostgreSQL, enfocado en seguridad y alta disponibilidad.
- **Heroku Redis:** Gestión de instancias Redis como servicio.

- **Heroku Teams:** Herramientas para colaboración en equipo y control de acceso.
- **Heroku Connect:** Integración entre aplicaciones Heroku y Salesforce.
- **Heroku Enterprise:** Servicios para grandes empresas con controles de acceso y espacios privados.
- **Heroku Elements:** Complementos y *Buildpacks* para el desarrollo y operación de aplicaciones.

En agosto de 2022, Heroku discontinuó sus planes gratuitos, citando razones de fraude y abuso, un cambio relevante para considerar en proyectos futuros.

2.3.1.4 PostgreSQL

PostgreSQL [16], conocido originalmente como POSTGRES, es un sistema de gestión de bases de datos relacional y orientado a objetos, que evolucionó a partir de un proyecto iniciado en 1986 en la Universidad de California, Berkeley, liderado por el Profesor Michael Stonebraker. Financiado por DARPA y NSF, el proyecto se transformó en Postgres95 en 1994 con la incorporación del lenguaje SQL por Andrew Yu y Jolly Chen, y posteriormente renombrado a PostgreSQL en 1996.

Este sistema destaca por su cumplimiento con el estándar SQL y sus funciones avanzadas, como la gestión de consultas complejas, claves foráneas, desencadenadores, vistas actualizables, integridad transaccional y control de concurrencia. Además, PostgreSQL es altamente expansible, permitiendo a los usuarios añadir nuevos tipos de datos, funciones y operadores.

La licencia libre de PostgreSQL facilita su uso, modificación y distribución gratuita para fines privados, comerciales o académicos.

Una herramienta destacada para la administración de PostgreSQL es pgAdmin [17], con una interfaz gráfica intuitiva que permite la gestión eficiente de bases de datos. Incluye funcionalidades para crear, editar, eliminar bases de datos, ejecutar consultas SQL, monitorear el rendimiento, y realizar copias de seguridad y restauración de datos.

2.3.1.5 Docker

Docker [18], un PaaS de código abierto lanzado en 2013, facilita el desarrollo, envío y ejecución de aplicaciones mediante contenedores aislados, asegurando la consistencia y portabilidad. Estos contenedores encapsulan lo necesario para ejecutar una aplicación, siendo útiles en entornos locales, en la nube o híbridos.

Las principales características de Docker incluyen:

- Entrega rápida y consistente de aplicaciones, apoyando entornos estandarizados y flujos de trabajo CI/CD.
- Implementación ágil y escalabilidad, con contenedores portátiles y adaptables a necesidades comerciales.
- Optimización del uso de recursos, siendo una alternativa ligera y eficiente a las máquinas virtuales.

Docker se gestiona a través del *daemon* Dockerd, que administra objetos como imágenes, contenedores, redes y volúmenes, y un cliente para la interacción del usuario.

Docker Desktop facilita la construcción y compartición de aplicaciones contenerizadas, compatible con varios sistemas operativos. Las imágenes de Docker, plantillas para crear contenedores, se almacenan en registros como Docker Hub. Se pueden construir imágenes propias a partir de un Dockerfile, siendo estas ligeras y rápidas.

Los contenedores, instancias ejecutables de imágenes, están aislados y se gestionan a través de la API o CLI de Docker. Se definen por su imagen y configuración, y los cambios no almacenados se pierden al eliminar el contenedor.

2.3.1.6 AWS

Amazon Web Services (AWS) [19], lanzado en 2006 por Amazon, es una plataforma líder en servicios de *cloud computing*, ofreciendo más de 200 servicios integrales de Tecnologías de la Información (IT). AWS surgió aprovechando la capacidad sobrante de servidores de Amazon, dando origen al concepto moderno de *cloud computing*.

El *cloud computing* proporciona recursos informáticos a través de internet según la demanda, con costos basados en el uso y sin necesidad de inversión inicial en infraestructura. AWS elimina la necesidad de anticipar y adquirir servidores, permitiendo despliegue instantáneo de servidores a demanda.

También ofrece una variedad de herramientas y servicios para la creación de entornos de computación remotos, adaptándose a empresas de todos los tamaños con posibilidades de escalar. Algunas herramientas clave de AWS utilizadas en este proyecto incluyen:

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) [20] proporciona capacidad de computación escalable en la nube de AWS. Permite lanzar y configurar servidores virtuales, gestionar seguridad y redes, y administrar almacenamiento. Ofrece características como instancias, imágenes de Amazon Machine (AMI), tipos de instancias, pares de clave, volúmenes de almacenamiento, y varias opciones de ubicación geográfica.

Amazon S3

Amazon Simple Storage Service (Amazon S3) [21] es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos y alto rendimiento. Utilizado para almacenar y proteger datos en una variedad de casos de uso, S3 ofrece tipos de almacenamiento como S3 Standard para datos críticos, S3 Intelligent-Tiering para la gestión eficiente de costos, y S3 Glacier para archivar datos con los costos mas bajos. Los datos se almacenan como objetos dentro de *buckets*, con funciones para gestionar el acceso y versiones de objetos.

Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) [22] es un servicio para gestionar registros de imágenes de contenedores de forma segura y escalable. Admite repositorios privados con permisos de AWS IAM, facilitando la carga, descarga y administración

de imágenes de contenedor. ECR incluye componentes como registro privado, token de autorización, repositorios, políticas sobre repositorios e imágenes de contenedor.

2.3.1.7 Digital Ocean

Fundada en 2003 como *server stack* (conjunto de servidores) por los hermanos Ben y Moisey Uretsky. Digital Ocean [23] evolucionó para ofrecer servicios de alojamiento en la nube y servidores virtuales desde 2012. Con cambios en la dirección ejecutiva, incluyendo a Mark Templeton y luego Yancey Spruill, Digital Ocean se ha establecido como una solución clave de servicios en la nube, con una infraestructura robusta que facilita el alojamiento de aplicaciones y sitios web.

Los usuarios utilizan Droplets, servidores virtuales privados de Digital Ocean, a través de una interfaz web o CLI. La plataforma es conocida por su infraestructura como servicio (IaaS) y es popular entre grandes empresas, ofreciendo personalización en aspectos como centros de datos y regiones geográficas.

Digital Ocean brinda módulos administrativos para monitoreo y gestión de red, incluyendo *firewalls* en la nube, equilibradores de carga y redes privadas virtuales. A cada Droplet se le pueden asignar distintas alertas que notifiquen vía correo electrónico si algún parámetro (uso de CPU, uso de disco, ancho de banda, etc) no se encuentra dentro de sus valores normales.

Las ventajas de Digital Ocean abarcan bajos costos de transferencia, garantía de tiempo de actividad, facilidad de uso, escalabilidad de precios y un enfoque en desarrolladores. Sin embargo, enfrenta desventajas como cobros por instancias apagadas, ubicaciones limitadas de servidores, falta de un nivel gratuito y una gama de productos más limitada en comparación con otros proveedores.

2.3.1.8 Cloudflare

Cloudflare, originado del Project Honey Pot en 2004 por Matthew Prince y Lee Holloway, evolucionó de un sistema de rastreo de *spam* a un servicio integral de seguridad en línea bajo la dirección de Michelle Zatlyn [24]. En 2009, tras ganar una competencia en la Escuela de Negocios de Harvard, Cloudflare obtuvo financiamiento para su desarrollo, reuniendo un equipo de expertos de empresas como Google y PayPal.

Lanzado oficialmente en septiembre de 2010, Cloudflare sorprendió a la comunidad de Project Honey Pot al ofrecer protección contra amenazas y acelerar la carga de sitios web [25]. Hoy en día, ofrece una gama de servicios de seguridad, rendimiento y fiabilidad a millones de clientes a nivel mundial.

Además, Cloudflare actúa como un registrador de nombres de dominio a través de Cloudflare Registrar. Este servicio incluye la gestión y seguridad de dominios, ofreciendo renovaciones sin aumentos de precio y funciones de seguridad avanzadas. Los costos se asocian únicamente con los registros y tarifas de ICANN.

Los registradores de nombres de dominio facilitan el acceso a sitios web mediante alias alfanuméricos, trabajando con registros responsables de los dominios de primer nivel (TLD).

2.3.2. Frontend

A continuación, se incluirá un breve marco teórico de las principales tecnologías y herramientas utilizadas en la aplicación Frontend de PSPcode.

2.3.2.1 JavaScript

JavaScript (1995 por Brendan Eich) [26] es un lenguaje de programación de alto nivel, orientado a objetos, aunque también es posible programar en un estilo funcional. Fue diseñado originalmente para ser utilizado en el lado del cliente (front-end) en aplicaciones web, como es el caso de PSPcode.

JavaScript es un lenguaje interpretado, es decir que el código se ejecuta directamente, sin necesidad de ser compilado previamente. En el caso de JavaScript, el código fuente se interpreta en tiempo real en el navegador web o en tiempo de ejecución en un entorno compatible. Esto permite una mayor flexibilidad y portabilidad en el desarrollo de aplicaciones web.

También es un lenguaje dinámico: no se requiere la declaración previa de tipos de datos, y las variables pueden cambiar de tipo durante la ejecución del programa.

JavaScript ejecuta en un solo hilo por defecto, lo que significa que solo puede procesar una tarea a la vez. Esto puede ser un problema cuando se trabaja con operaciones que potencialmente tardan en completarse, como las peticiones a servidores, ya que la ejecución del resto del código se detendrá hasta que se complete dicha tarea.

Como solución a esta problemática, este lenguaje permite la ejecución asíncrona de código mediante el uso de las llamadas “funciones asíncronas” o “promesas”. Estos métodos permiten que el código continúe ejecutándose mientras se espera a que se complete una tarea de estas características. De esta manera, se mantiene la capacidad de respuesta de la aplicación mientras se realizan tareas en segundo plano.

2.3.2.2 React

React (2013 por Meta, antes Facebook) [27] es una biblioteca muy poderosa de JavaScript y tiene el objetivo principal de construir interfaces de usuario (UI), de manera rápida y eficiente, beneficiando al desarrollador. React se basa en el concepto de “componentes”: piezas de código reutilizables y personalizables, que representan partes de la interfaz de usuario. Por ejemplo: un botón, un formulario, un cuadro de texto, entre muchos otros. Estos componentes se combinan para formar una aplicación completa y constituyen una de las grandes ventajas de React: la modularidad.

Las características que potencian a React son:

- **Rendimiento:** React utiliza un modelo de programación denominado “virtual DOM”, que permite actualizar únicamente los elementos modificados en lugar de toda la página en cada interacción. Esta técnica minimiza la cantidad de cambios necesarios en el DOM real, lo que mejora significativamente la eficiencia y velocidad de la aplicación.

Además, React facilita la gestión de cambios mediante ciclos de vida y estados, lo que contribuye a un rendimiento óptimo y una respuesta más rápida. Esta capaci-

dad para actualizar solo los elementos modificados y ejecutar funciones bajo ciertas circunstancias reduce el número de operaciones innecesarias, mejorando así la eficiencia general de la aplicación.

- **Modularidad:** como ya se mencionó, React se fundamenta en el concepto de componentes, lo que permite crear piezas de código reutilizables. Esto facilita la organización y mantenimiento del código, especialmente en proyectos grandes. Como también, simplifica y acelera mucho el desarrollo en sí mismo, ya que gran parte del código no tiene por qué reescribirse.
- **JSX:** React utiliza JSX (JavaScript XML) como una sintaxis propia para definir la estructura a renderizar de los componentes. JSX combina elementos HTML y JavaScript en un solo archivo (.js o .jsx), simplificando la escritura y comprensión del código al permitir el uso directo de etiquetas HTML dentro de los componentes de React.

Al emplear JSX, se mejora la legibilidad y mantenibilidad del código, ya que la estructura de los componentes refleja la interfaz de usuario final. Durante el proceso de construcción, JSX se compila en código JavaScript estándar con la ayuda de herramientas como Babel [28], que se encarga de convertir el JSX en llamadas a funciones de React para crear y actualizar elementos de la interfaz de usuario.

- **Comunidad y documentación:** React destaca como una de las librerías más utilizadas para la construcción de interfaces de usuario. Según la encuesta anual de Stack Overflow [29] realizada en mayo de 2022, React ha sido la tecnología preferida para este propósito durante los últimos cinco años.

Con más de 19 millones y medio de descargas semanales y 207.000 estrellas en su repositorio de GitHub, React cuenta con una gran comunidad de desarrolladores que contribuyen a su implementación, reportan errores y ofrecen soluciones.

Esta comunidad activa proporciona actualizaciones frecuentes, respuestas rápidas a problemas y una amplia gama de librerías de terceros para simplificar tareas. Además, React cuenta con una documentación exhaustiva y actualizada, junto con numerosos recursos en línea, tutoriales y ejemplos de código para facilitar su aprendizaje y uso.

2.3.2.3 Manejadores de Paquetes

Las aplicaciones modernas de JavaScript utilizan, en su gran mayoría, un *package manager*.

Esta es una herramienta de línea de comandos que se utiliza para instalar, actualizar, configurar y eliminar paquetes de software en un proyecto. Un paquete es un conjunto de archivos que se pueden distribuir e instalar como una unidad en un proyecto de software, los mismos pueden contener una o varias librerías y otros recursos necesarios para su uso. En el ecosistema de JavaScript, los manejadores de paquetes más populares son npm y yarn.

npm (Node Package Manager, por Isaac Z. Schlueter, 2010) [30]: Es el manejador de paquetes más utilizado en el ecosistema de JavaScript. Viene incluido con Node.js, por lo que ya se tiene por defecto cuando se instala dicha tecnología (necesaria para correr cualquier aplicación de React). Permite instalar paquetes desde su repositorio oficial o desde otros repositorios como GitHub. npm también proporciona herramientas para publicar paquetes y compartirlos con la comunidad de desarrolladores.

Yarn (por Facebook, 2016) [31]: Aunque funciona de manera similar a npm, utiliza una arquitectura más moderna y eficiente para manejar las dependencias. Además, Yarn tiene una mejor gestión de caché, y la capacidad de manejar dependencias de forma más segura y predecible. También cuenta con características como la instalación paralela de paquetes y el bloqueo de versiones para evitar conflictos de dependencias. De hecho, yarn fue lanzado con el objetivo de solucionar problemas de seguridad y velocidad que npm tenía en el momento.

Lo cierto es que npm ha evolucionado mucho tras el lanzamiento de Yarn, y la brecha tecnológica entre ellos es cada vez más pequeña, casi nula. Por lo que cualquiera de los dos manejadores funciona muy bien para el propósito descripto.

Funcionamiento de YARN:

Para inicializar Yarn en una aplicación, se recomienda usar el comando `yarn init`, que guía al usuario a través de la línea de comandos para configurar la aplicación con preguntas sobre su nombre, versión, descripción, autor y licencia. Esto crea el archivo `package.json` en la raíz del proyecto con la información básica. También es posible crear este archivo manualmente, pero puede ser propenso a errores.

El archivo `package.json` es fundamental en el funcionamiento de Yarn, ya que define las dependencias y configuraciones del proyecto de JavaScript, incluyendo los atributos como nombre, versión, descripción, dependencias y scripts.

Además, Yarn utiliza archivos como `package-lock.json` y `yarn.lock` para garantizar la consistencia entre desarrolladores y entornos de producción en cuanto a las versiones de las dependencias instaladas. Al ejecutar `yarn install`, Yarn descarga las dependencias especificadas en `package.json`, las guarda en la carpeta `node_modules` y crea o actualiza el archivo `yarn.lock` con las versiones exactas de cada dependencia, el mismo debe ser incluido en el repositorio colaborativo del proyecto para asegurar la consistencia en todos los entornos.

2.3.2.4 Webpack (build tool):

Por otro lado, la aplicación utilizaba (y utiliza) Webpack, una librería de JavaScript creada en 2012 por Tobias Koppers [32], un desarrollador web alemán, como herramienta de construcción (*build tool*) utilizada para compilar, empaquetar y transformar los recursos web, como ser archivos JavaScript, CSS, imágenes o fuentes de texto, en un formato apto y optimizado para su ejecución en un navegador web.

Webpack toma módulos de JavaScript y los organiza en un gráfico de dependencias para luego generar un solo (por defecto, podrían ser varios) archivo JavaScript que pueda ser leído por el navegador. Esto permite una mejor gestión de dependencias y un mejor rendimiento en la carga de la página.

Además de la transformación y empaquetado de archivos, Webpack también permite la inclusión de *plugins* y *loaders* para agregar funcionalidades específicas, como la compresión de código, la optimización de imágenes y el soporte para preprocesadores de CSS. El desarrollador puede agregar, quitar y personalizar estos parámetros desde distintos archivos de configuración `.js` que luego son utilizados por el comando que corre Webpack.

Por ejemplo, en el `package.json` se pueden definir los siguientes scripts:

```
1 {
2   "scripts": {
3     ...
4     "start": "webpack serve --open --config webpack.dev.js"
5     "build": "webpack --config webpack.prod.js",
6     ...
7   }
8 }
```

El comando de `build` se encarga de compilar y empaquetar la aplicación para su uso en producción. El parámetro `--config` especifica el archivo de configuración que se debe utilizar para compilar y empaquetar la aplicación. En este caso, se está utilizando el archivo `webpack.prod.js`, que contiene la configuración necesaria para crear una versión optimizada de la aplicación, que se puede utilizar en un entorno de producción.

El script `start` define un comando que ejecuta el servidor de desarrollo de Webpack, una herramienta que permite visualizar y probar la aplicación mientras se desarrolla. Al ejecutar este comando, debido al parámetro `--open`, se abrirá una nueva pestaña del navegador predeterminado con la URL del servidor de desarrollo (típicamente `localhost:<port.number>`). Cada vez que se realice un cambio en el código fuente de la aplicación, el servidor de desarrollo se encargará de actualizar la página automáticamente.

En este caso, se está utilizando el archivo `webpack.dev.js`, que contiene la configuración necesaria para ejecutar el servidor de desarrollo.

2.3.2.5 Babel

Babel [28] es una herramienta de compilación y transpilación de código JavaScript que permite a los desarrolladores escribir código utilizando funcionalidades de versiones más recientes del lenguaje y luego transformarlo en una versión compatible con navegadores y entornos antiguos. Se utiliza comúnmente junto con Webpack para procesar el código fuente de una aplicación antes de ser empaquetado.

Una de las transformaciones más comunes que realiza Babel es la conversión del código JSX de React a JavaScript estándar. La configuración de Babel se realiza a través del archivo `.babelrc` o en la configuración de Webpack. Babel utiliza *presets* predefinidos para aplicar transformaciones específicas al código, como `@babel/preset-env` para características de JavaScript y `@babel/preset-react` para React.

Además de los *presets*, Babel permite el uso de *plugins* personalizados para aplicar transformaciones específicas al código. Por ejemplo, en PSPcode se utiliza el *plugin react-hot-loader/babel* para habilitar la actualización automática de componentes de React modificados sin recargar toda la página. Esto significa que al tener la aplicación levantada en un servidor local, los cambios realizados en el código fuente de los componentes se

reflejarán de inmediato en el navegador, lo que acelera el ciclo de desarrollo y mejora la productividad.

2.4. Herramienta web: PSP code

En 2018 culminó el Proyecto de Grado de Guillermo Kuster y Guillermo Tavidian [3], el cuál tuvo como resultado PSPcode: una aplicación web utilizada como plataforma de soporte para el curso PF-PSP.

PSPcode fue diseñada para facilitar la interacción y colaboración entre estudiantes y docentes durante el proceso de realización y entrega de programas utilizando la metodología PSP.

PSPcode cuenta con dos roles de usuario: Docente (utilizado por los docentes del curso), y Estudiante (atribuido a los estudiantes del curso).

El sistema se conforma de dos aplicaciones: un sitio de administración, donde los usuarios de tipo Docente pueden configurar parámetros y agregar distintas entidades. Y una aplicación web principal, donde los estudiantes pueden realizar sus ejercicios e interactuar con los docentes.

Ambas aplicaciones web operan con el mismo Backend y base de datos, simplemente tienen distintas funcionalidades y autenticación. Los detalles técnicos y de arquitectura de la versión anterior de PSPcode que afectan directamente a este proyecto se presentarán en la Sección 2.5.

2.4.1. Enumeración de funcionalidades

La plataforma principal permite a los estudiantes desarrollar proyectos, registrando todos los datos (descritos en la Sección 2.2) necesarios en cada fase del proyecto, y cargar un archivo *zip* como entrega final, con los elementos de código que implementan el programa solicitado. Los docentes pueden descargar la entrega y revisar los datos registrados en cada fase, aprobando o pidiendo la re-entrega del proyecto.

Además, la aplicación ofrece un sistema de mensajería que permite a los usuarios comunicarse entre sí.

Por otra parte, al dar su devolución, los docentes tienen la posibilidad de cargar un archivo detallando la corrección; típicamente éste es un documento *pdf* que contiene una planilla respetando siempre el mismo *template*, a la cual a partir de ahora se nombrará como “*grading*”.

A continuación, se enumeran las funcionalidades de la versión original de PSPcode (2018), a la cuál se referirá como “aplicación origen” de ahora en más. El detalle de cada funcionalidad ya existente no pertenece al alcance de este trabajo, por ende, se presentarán las mismas superficialmente para dar contexto de la aplicación origen con la que se comenzó este Proyecto de Grado.

Desde el *Admin*, un usuario de rol Docente puede agregar, eliminar o editar las siguientes entidades:

- Distintos Modelos de Proceso (versiones de PSP).
- Cursos (típicamente se instancia uno por semestre que se dicta PF-PSP).
- Usuarios con rol Docente o Estudiante.
- Proyectos. Configurando las características de los mismos, por ejemplo nombre, el *script*, que PSP deben seguir, etc.
- Las Fases de las que se puede componer cada proyecto una vez completado (Design, Code, Compile, etc.).

Pueden también realizar asignaciones:

- Docentes a los distintos Estudiantes y Cursos.
- Estudiantes a Cursos.
- Proyectos a Cursos y Estudiantes.
- Grupo de Fases a un Modelo de Proceso
- Modelo de Proceso a un Proyecto.

Luego, en la aplicación principal, un usuario de tipo Docente puede:

- Ver un listado de Proyectos.
- Ver un listado de Estudiantes.
- Ver los Proyectos de un Estudiante en particular.
- Ver la información de usuario de un Estudiante.
- Ver el progreso de un Proyecto asignado a un Estudiante.
- Marcar un Proyecto como aprobado o reprobado (necesita re-entrega).
- Enviar al estudiante un archivo externo a modo de *feedback*, cuando se aprueba o desaprueba un proyecto.
- Comunicarse con el estudiante mediante la funcionalidad de mensajería.
- Recibir notificaciones, por ejemplo cuando un Estudiante entrega un Proyecto.

Los usuarios con rol Estudiante pueden:

- Ver y Editar su información de usuario.
- Realizar los Proyectos, agregando y completando distintas Fases del mismo.
- Adjuntar un archivo *zip* a los Proyectos.
- Enviar al Docente los Proyectos terminados.
- Intercambiar mensajes con su Docente asignado.
- Recibir un *feedback* de su Proyecto por parte del Docente.
- Re-entregar una nueva versión del Proyecto en caso de ser necesario.

2.5. Estado técnico en el que se heredó PSPcode

En este capítulo se describe en detalle la situación tecnológica en la que se recibió la aplicación PSPcode y los distintos servicios utilizados.

Se decidió iniciar nuevos repositorios de GitHub en lugar de trabajar sobre los ya existentes. Para ello, a partir de los repositorios originales en formato zip, se crearon dos nuevos: “PSPcode-v2_Frontend” [33] y “PSPcode-v2_Backend” [34], en donde se incorporó todo el trabajo realizado en el Proyecto de Grado de 2018, lo cual fue fundamental como base para el desarrollo del proyecto.

Este proyecto comenzó con un fase de *discovery* donde se buscaron los últimos *commits* realizados en los repositorios originales para inferir en qué rama estaba acumulada la última versión del trabajo, ya que al tratar por primera vez con los repositorios se pudo notar que la última versión del trabajo no se encontraba en la rama *main*, siendo esto lo convencional.

En ambos repositorios (Backend y Frontend), lo último implementado y publicado estaba en una *feature branch* específica. Se tuvo en cuenta que estos últimos *commits* se correspondieran con una versión estable y coincidiera con lo que estaba liberado en producción.

Para comenzar con este trabajo se realizó un *commit* inicial a la rama *main* en los dos nuevos repositorios con las últimas versiones de la aplicación origen.

Además de los repositorios del proyecto anterior, se obtuvo un documento con todas las credenciales necesarias para administrar los diferentes servicios utilizados para el manejo de la aplicación, tales como el correo electrónico personal del curso, el servidor de mails de Sendgrid, Heroku (en donde estaba desplegada la herramienta productiva), Base de datos y la cuenta AWS que contiene el *bucket s3* para el almacenamiento de archivos. Este documento también contiene instrucciones para ejecutar la aplicación tanto de manera local como productiva. Además de esas instrucciones, se detallan los pasos a seguir para poder crear una base de datos nueva y migrar las tablas necesarias para la ejecución de la aplicación.

2.5.1. Aplicación Backend

En primer lugar se presenta en el Anexo .2.1 el modelo de dominio de la aplicación origen, es decir, el estado original de la misma antes de realizar ninguna mejora y/o modificación. El modelo de dominio proporciona una representación visual de las entidades principales y las relaciones entre ellas.

Este modelo fue diseñado para abordar los requisitos iniciales y brindar una visión general de la estructura y funcionalidad de la aplicación en ese momento.

Es importante destacar que el modelo de dominio previo permitió comprender la estructura general de la aplicación y sirvió como punto de partida para realizar las mejoras y modificaciones posteriores.

2.5.1.1 Implementación

El código Backend se implementó en Ruby on Rails, una tecnología con la cual el equipo no estaba familiarizado. Por ende, gran parte del trabajo implicó aprender y experimentar con esta tecnología, lo cual permitió adquirir conocimientos valiosos en cuanto a su uso y sus funcionalidades. Para manipular el código, se utilizó RubyMine, y el IDE de Ruby on Rails de JetBrains.

2.5.1.2 Correos electrónicos

Para facilitar el intercambio de correos electrónicos entre estudiantes y profesores, la aplicación original utiliza el servicio de correo electrónico Sendgrid. Se envían correos que contienen mensajes de bienvenida con las contraseñas de los usuarios recién creados, así como notificaciones cuando un profesor asigna o aprueba un proyecto, entre otros eventos relevantes dentro de la herramienta. Además, cada vez que un profesor o un estudiante envía mensajes dentro de la herramienta, se les informa también a través de correos.

De este servicio se cuenta con el plan gratuito el cual consiste en el intercambio de 100 correos por día. Llegado a este tope, el servidor retorna un error, no pudiendo continuar con el flujo normal de la aplicación.

2.5.1.3 Administración de archivos

Los proyectos dentro de la herramienta, pueden guardar archivos en formato .zip. Para ello, se cuenta con un *bucket* s3 de AWS, el cual está sincronizado con la aplicación y puede ser accedido tanto por los docentes como por los estudiantes para descargar la información de su proyecto.

2.5.1.4 Base de datos

La base de datos consiste en un modelo relacional con tablas en el lenguaje PostgreSQL. Para el desarrollo, se utilizó una base de datos local y el manejo se hizo a través de la aplicación pgAdmin4. La base productiva se creó a partir de Heroku en donde se encontraba desplegada la aplicación.

2.5.1.5 Despliegues

Heroku alojaba la aplicación origen, ya que en su momento este servicio ofrecía un plan gratuito que permitía manejar fácilmente los servidores, configuraciones, escalamiento y administración.

2.5.2. Aplicación Frontend

La aplicación de Frontend fue implementada en el lenguaje JavaScript utilizando la librería React para el desarrollo de la interfaz de usuario. A su vez, se utiliza la herramienta de construcción (*build tool*) Webpack, combinada con la librería Babel como transpilador de código JavaScript.

Las versiones utilizadas por la aplicación PSPcode origen eran las siguientes:

React: 15.5.4

Webpack: 2.5.1

babel-core: 6.24.1

En cuanto a interfaz de usuario, la aplicación origen utiliza componentes de la librería AntD, en su versión `3.0.0-alpha.13`.

Los componentes de React están escritos como Class Components, una sintaxis que no se recomienda desde principios de 2019.

Con respecto a manejadores de paquetes [2.3.2.3](#), en la aplicación origen se habían utilizado dos manejadores distintos: Yarn y npm, ya que poseía en su raíz dos archivos de configuración: `package-lock.json` (propio de npm) y `yarn.lock` (propio de aplicaciones que utilizan yarn). Este es un error que comúnmente se da cuando dos o más desarrolladores intervienen en la aplicación, utilizando cada uno un manejador distinto, y publican en el repositorio su propio archivo de versiones. Lo correcto es que todos los desarrolladores utilicen el mismo manejador, de lo contrario se pueden dar inconsistencias en la administración de los paquetes.

Se decidió continuar por el camino de Yarn, debido a sus ventajas en velocidad y seguridad. A su vez, uno de los integrantes del equipo utiliza dicho manejador de paquetes cotidianamente en su trabajo, por lo que ya se tenía más experiencia con la herramienta.

En el Anexo [.1.1](#) se muestra el listado completo de paquetes utilizados por la aplicación en el momento de su entrega en 2018.

Es importante destacar que las versiones de los paquetes estaban considerablemente desactualizadas, como se detallará en el Capítulo [4](#).

Capítulo 3

Metodología de trabajo

3.1. Plan y desarrollo cronológico del proyecto

El proyecto se comenzó en Marzo de 2022 y los objetivos propuestos fueron:

- Poder realizar las correcciones docentes dentro de PSPcode generando correcciones parciales automáticamente.
- Evaluar migrar PSPcode a otra plataforma
- Permitir crear nuevas fases al proceso así como eliminar fases y modificar tareas y actividades.
- Escribir un cuestionario final para conocer cómo el estudiante percibe la aplicación del curso.
- Investigar la posibilidad de integrar PSPcode con R (herramienta para realizar test estadísticos).

El cronograma propuesto inicialmente para lograr los objetivos fue:

- Mes 1 y 2 (Marzo y Abril): Comprensión del funcionamiento de PSPcode y análisis del código.
- Mes 3 (Mayo): Estudio de la herramienta R y evaluar la integración con PSPcode.
- Mes 4 (Junio) Evaluar diferentes plataformas para *hostear* PSPcode, ya que actualmente se encuentra en Heroku y cuenta con ciertas limitantes.
- Mes 5 al 9 (de Julio hasta Noviembre): Diseño, Implementación y Testing de los cambios propuestos a PSPcode.
- Mes 10 (Diciembre) Informe escrito y presentación oral del trabajo.

Las desviaciones principales entre los objetivos iniciales y los resultados alcanzados fueron:

- Migrar PSPcode a otra plataforma de *hosting* era un objetivo del proyecto que estaba planteado para junio 2022, y se cumplió al liberar la nueva aplicación, como se describe en el Capítulo 6. Sin embargo, la aplicación origen estaba *hosteada* bajo un plan muy limitado (en cuanto a almacenamiento) de Heroku, que resultó ser insuficiente incluso para llevar a cabo el curso de marzo 2022, por esto, investigar

alternativas y aumentar el plan fue una tarea urgente a realizar que se suma a los objetivos siendo la primera en ser ejecutada. Los detalles se describen en la Sección 4.1.1.

- Por otro lado, el código de la aplicación origen, desarrollado en 2017, se encontraba considerablemente obsoleto, lo que impedía el progreso en otras fases del proyecto. Además de los paquetes deprecados, se identificaron prácticas poco recomendadas que requerían refactorizaciones. Por consiguiente, se utilizaron tres meses del calendario del proyecto para llevar a cabo actualizaciones y reescrituras del código detalladas en el Capítulo 4. Esta labor continuó siendo una prioridad a lo largo del desarrollo del proyecto, especialmente en la aplicación Frontend.
- Debido al cambio en el cronograma por esta situación, el objetivo de la investigación de la herramienta R fue despriorizado, y se comenzó con el desarrollo de las nuevas funcionalidades para la aplicación apenas terminadas las refactorizaciones.
- Mientras se daba la etapa de pruebas y validación de las nuevas funcionalidades por parte de las docentes (Diciembre 2022), se realizó una breve investigación de la herramienta R y se tuvo una reunión con la tutora para definir el accionable con respecto a este objetivo. Dado que aún la profesora no tenía del todo definido qué estadísticas debían tomarse, y que el análisis e implementación de esta *feature* llevaría al menos dos meses más, se decide no llevarla a cabo.
- Sin embargo, se suma a los hitos alcanzados en el proyecto, la liberación a producción de la aplicación PSPcode, y su uso durante todo el primer semestre de 2023, completando de esta manera un *testing* y *bug fixing* en producción de mucho valor para la aplicación.
- No solo se escribe, sino que se realiza y analiza, un cuestionario a los estudiantes, cuyo resultado es presentado en la Sección 6.5.

A continuación, se exhiben diagramas con el objetivo de proporcionar una representación visual, clarificando el desarrollo cronológico del proyecto.

A modo de referencia:

- Los encabezados de los meses se encuentran pintados de acuerdo al color de la tarea que toma protagonismo en ese período de tiempo.
- El trabajo constante en segundo plano ejecutado para las tareas “Actualización de librerías y refactorización” y “Escritura del informe” se encuentra representado en la Figura 3.2 con un tono de color más claro que el esfuerzo primario.

3.1.1. Plan Inicial

El plan inicial delineaba un proyecto de 10 meses, que abarcaba desde marzo hasta diciembre de 2022, sin contemplar la experiencia de la aplicación en un ambiente productivo (curso 2023).

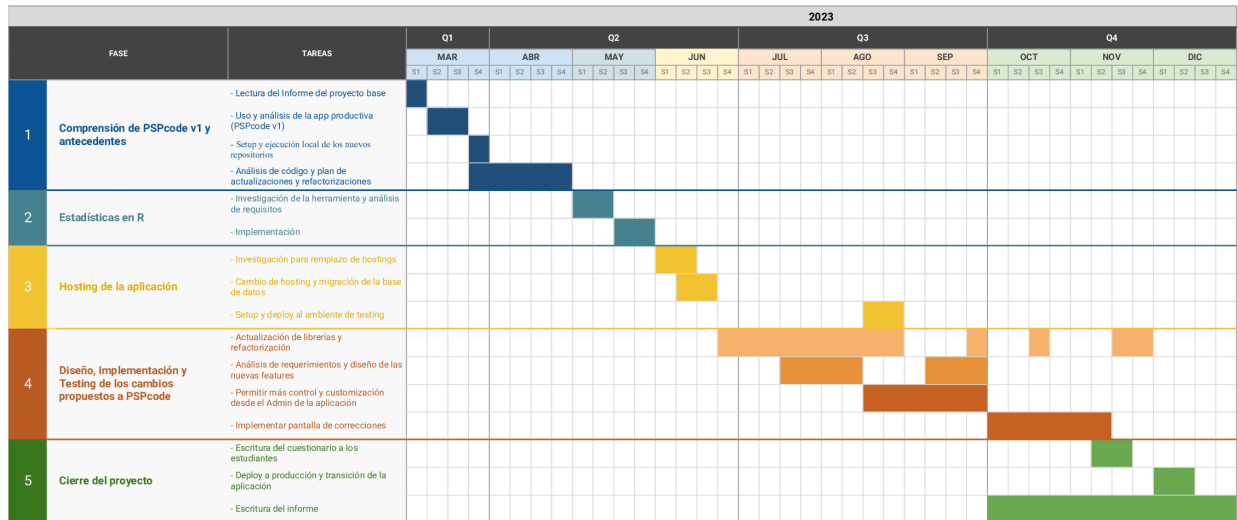


Figura 3.1: Cronograma del plan inicial del proyecto.

3.1.2. Desarrollo cronológico del Proyecto

A continuación se presenta el desarrollo real del proyecto en materia de tiempos. Considerar que la escala fue reducida a la mitad para respetar el espacio disponible.

El proyecto se extendió hasta Noviembre 2023. En cuanto a cambios estructurales puede verse:

- Anticipación del cambio de plan en Heroku.
- La necesidad de dividir las tareas de actualización, refactorización y desarrollo, ya que el primer grupo tomó más esfuerzo del estimado y constituyó una fase en sí misma. En la misma línea, se puede ver la constante iteración de dicha fase a lo largo del proyecto
- La despriorización de tareas referidas a R.
- Se agrega la fase “PSPcode v2 en Producción”, representando las tareas referidas al desempeño de la aplicación en el entorno productivo.

- 35 Pull Requests
- 90 Commits
- 102 Archivos modificados
- 39,299 Líneas agregadas, y 23,156 líneas eliminadas

Métricas del repositorio de Backend:

- 15 Pull Requests
- 60 Commits
- 233 Archivos modificados
- 2,628 líneas agregadas y 1,158 líneas eliminadas

3.2.2.2 Jira

Se utilizó la herramienta Jira para la organización de tareas durante la etapa principal de desarrollo (Abril - Noviembre 2022).

Se definieron tres épicas, y para ellas distintas tareas de Frontend y Backend.

- **“Code Refactor”**: con todas las tareas referidas a la actualización de librerías y refactorización de código. Capítulo 4.
- **“Grading Automation”**: describiendo todos los criterios de la *grading* a automatizar y mostrar en forma de advertencia a los distintos tipos de usuario. Para estas tareas, aparte de la distinción entre Backend y Frontend, se utilizaron *labels* llamadas “professor_view” o “student_view”, indicando si debía ser una corrección a mostrar al estudiante previo a entregar el proyecto, o al profesor, para ayudarlo a corregir un proyecto ya entregado. Sección 5.1.1.
- **“Correction Tab”**: en esta épica se dividieron las tareas a realizar para llevar a cabo la pestaña de correcciones, donde se incluyó toda la implementación de la entidad Project Feedback y sus relaciones. Sección 5.3.

Luego, para la etapa de cambios y corrección de *bugs*, se utilizó la metodología descrita en el Capítulo 6.

Capítulo 4

Actualización y Refactorización de la aplicación

La aplicación origen, desarrollada en 2017, llevaba cinco años sin mantenimiento al comenzar el presente proyecto. La deuda técnica yacente en el código era significativa, con la mayoría de las librerías obsoletas, lo que generaba vulnerabilidades de seguridad y obstaculizaba la mantenibilidad y expansión funcional de PSPcode. Por esto, se dedicó un importante esfuerzo a la actualización y refactorización del código en ambos repositorios: Backend y Frontend.

4.1. Aplicación Backend

4.1.1. Ajustes y Desafíos previos al Desarrollo

Antes de iniciar el desarrollo del Backend de la nueva aplicación, surgió la necesidad de brindar asistencia a la tutora en la solución de diversos problemas relacionados con la aplicación origen, que en ese momento se encontraba en producción. Esta aplicación estaba alojada en Heroku, junto con su base de datos, que había estado experimentando un crecimiento continuo desde aproximadamente el año 2018.

Se detectó que la base de datos había alcanzado la cantidad máxima de filas (*rows*) permitidas en todas sus tablas, llegando a tener 9,332 de 10,000 filas disponibles, como se muestra en la Figura 4.1.

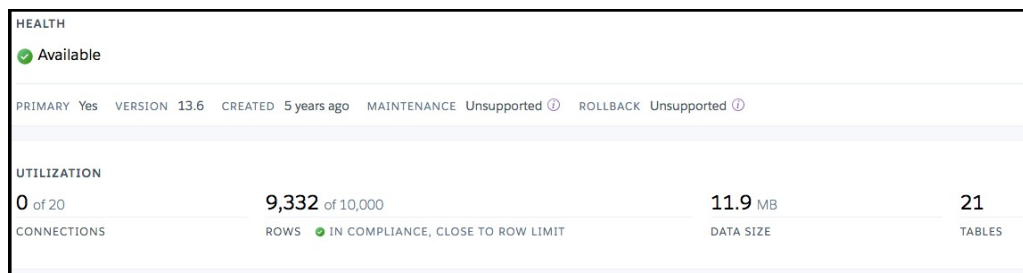


Figura 4.1: Cantidad de columnas totales de la base de datos anterior.

Para abordar la problemática, se dedicó tiempo a investigar sobre Heroku y los planes que ofrecía. Después de una investigación exhaustiva, se llegó a la conclusión de que la

mejor solución era actualizar el plan de la base de datos. Actualmente se estaba utilizando el plan denominado “Mini”, que tiene un costo máximo de U\$S5 al mes y permite un total de 10,000 filas. La decisión fue actualizar al plan “Basic”, similar al plan anterior con la diferencia de que permite 10,000,000 de filas, suficientes para almacenar todos los datos necesarios para el uso de la aplicación en 2022. Se procedió a realizar el cambio de plan, lo que implicó la creación de una nueva base de datos y la posterior migración de los datos de la base anterior.

Una vez que la nueva base de datos estuvo lista y con su límite de filas actualizado, se procedió a la eliminación de la base de datos asociada al plan anterior.

Otro de los problemas que se tuvo que abordar fue el relacionado con el envío de correos electrónicos, dado que se utilizaba Sendgrid, una herramienta desconocida para el equipo en ese momento.

Luego de investigar su funcionamiento y por qué no estaba operando correctamente, se encontró que la cuenta fue desactivada debido a la falta de verificación de su legitimidad durante un largo período de tiempo.

Para solucionar esto, se creó una nueva cuenta en Sendgrid y se vinculó al proyecto. Esto no solo resolvió la suspensión, sino que también permitió que el envío de correos electrónicos volviera a funcionar correctamente, asegurando así que la función de correo de la aplicación estuviera en buen estado. Posteriormente, esta nueva cuenta se utilizó también para la nueva versión de PSPcode.

4.1.2. Desarrollo y Actualización del código

Durante el proceso de desarrollo y actualización del código Backend, se encontraron diversos desafíos que requirieron de un importante esfuerzo por parte del equipo.

4.1.2.1 Actualización de librerías

En primer lugar, se actualizó la versión de Ruby de 2.3.1 a la más nueva disponible en ese momento, la 3.2.1. Esta decisión se tomó principalmente debido a que la versión anterior ya no estaba siendo soportada ni mantenida. La actualización era inminente para asegurar que el sistema funcionara, estuviera protegido y fuera compatible con las tecnologías actuales.

Como parte de este proceso, también se actualizaron ciertas gemas, con especial atención en la versión de *Rails*. Inicialmente, se consideró pasar a la última versión disponible hasta el momento (7.0.4). Sin embargo, a medida que se iban actualizando otras gemas, se presentaron incompatibilidades que requerían ajustes. Esto llevó a la necesidad de bajar la versión de *Rails* en un esfuerzo por mantener la coherencia y asegurar que todas las partes del sistema funcionaran de manera armoniosa, terminando en la versión 6.1.5, que igualmente sigue siendo mantenida.

Lograr la actualización y compatibilidad de todas las gemas demandó una inversión considerable de tiempo y recursos. A su vez, dado que Ruby es un lenguaje de programación interpretado, lo que significa que el código se ejecuta directamente por el intérprete sin requerir un proceso de compilación previo, fue necesario llevar a cabo pruebas de regresión para confirmar que todo funcionaba correctamente, ya que si un flujo en específico no se prueba puede contener errores y no ser detectados.

4.1.2.2 Refactorización

Luego a de obtener un entorno de desarrollo actualizado, se refactorizó gran parte del código. Se utilizó la gema *Rubocop* versión 1.26.1 para formatear y mejorar la calidad del código. Esto no solo mejoró la apariencia del código, sino que también contribuyó a una mayor claridad en la estructura y el flujo de trabajo del proyecto.

A continuación se puntualizan algunos de los cambios que se realizaron como parte del refactor:

- Se han agregado algunas directivas para mejorar la calidad y el correcto funcionamiento del código, como por ejemplo `frozen_string_literal: true`. Esta directiva se utiliza en Ruby para congelar cadenas de caracteres (*strings*) en un archivo. Es decir, cuando se agrega esto al principio de un archivo, se indica que todos los *strings* son inmutables; una vez definidos, no pueden modificarse. Esto ayuda a prevenir cambios no deseados y mejora la integridad.
- Se cambió la forma de escribir la concatenación de *strings*, pasando, por ejemplo, de `first_name + " " + last_name` a `#{first_name} #{last_name}`. Mejorando así la legibilidad y claridad del código, así como su eficiencia.
- Se eliminaron todas las comillas dobles donde no eran necesarias, optando por las comillas simples. Las primeras se utilizan principalmente cuando se necesita interpolación de variables, es decir, para insertar el valor de una variable dentro del *string*.
- Se optó por escribir los arreglos de símbolos de una manera más legible, pasando de `[:símbolo1, :símbolo2]` a `%i[símbolo1, símbolo2]`.
- Se modificó la estructura de control utilizada para decidir cuándo ejecutar ciertas acciones en el código. En lugar de utilizar *if* para entrar en bloques de código basados en condiciones afirmativas, se optó por *unless* para ejecutar bloques de código cuando las condiciones eran negativas o falsas. Este cambio permitió una expresión más directa y clara de la lógica del programa. En lugar de entrar en un bloque de código para realizar una serie de comprobaciones y luego decidir si salir o no, se verifica de inmediato si la condición es falsa y, si lo es, se ejecuta el bloque de código correspondiente. Esto simplificó la estructura y mejoró la legibilidad, al enfocarse en las condiciones negadas y reducir la necesidad de anidación.

4.1.2.3 Base de datos

Se llevaron a cabo diversas modificaciones en la estructura de la base de datos a través de la semilla (o *seed* en inglés): un archivo que incorpora datos iniciales destinados a ser cargados en las tablas de la base de datos, en el Anexo .5.3 se adjunta la semilla utilizada en producción a modo de ejemplo. Se implementó una segmentación de las semillas, dividiéndolas según el entorno: una destinada al entorno de producción (*production*) y otra al entorno de desarrollo (*development*).

Dentro de la información contenida en estas semillas, se encuentran registros de estudiantes, profesores, cursos, criterios, proyectos, fases, y otros elementos relevantes para el sistema. La segmentación por entornos permitió una gestión más eficaz de los datos iniciales y facilitó la adaptación a distintos escenarios de implementación.

A continuación se muestra un ejemplo de carga de una fase:

```
1 Phase.create! name: 'PLAN',  
2   description: 'This phase is to understand the excercise',  
3   order: 1,  
4   first: true,  
5   last: false
```

Para crear cualquier entidad, se debe especificar el nombre de la misma seguido de la palabra `create!` y a continuación todos los atributos que posea dicha entidad, en este caso, se crea la fase de PLAN con su descripción, y la especificación necesaria para indicar que es la primera.

4.1.2.4 Manejo de archivos

PSPcode gestiona archivos, principalmente en formato `.zip`, que contienen el código de los proyectos entregados y otra información relevante.

Para manejar estos archivos, se implementó un proceso específico bajo el marco de este proyecto. Inicialmente, se sube un archivo `.zip` que contiene toda la información del proyecto. Durante las etapas de pruebas locales y en *staging*, se realizó una configuración especial para almacenar los archivos en una ruta específica dentro de la aplicación. Para ello, se creó una carpeta dedicada que contenía los archivos que se iban cargando desde el Backend. Conforme avanzaba el proyecto, esta carpeta iba creciendo en tamaño, ya que se añadían más archivos y se iban actualizando los existentes.

Posteriormente, al llegar a la etapa de producción, se optó por utilizar un *bucket* nuevo (distinto al existente de la aplicación origen) de Amazon Web Services (AWS) para gestionar los archivos. Al utilizar el bucket de AWS, se logró una mayor flexibilidad y eficiencia en el manejo de los archivos en comparación con la configuración local anterior.

4.1.2.5 Variables de entorno

Las variables de entorno en el proyecto se gestionan de manera diferente en entornos locales y productivos. En el caso del desarrollo local, se utilizó un archivo llamado `application.yml` para definir y ajustar estas variables. Sin embargo, cuando se despliega el proyecto en entornos de producción, se adopta una estrategia más robusta y segura.

En producción, las máquinas que alojan el proyecto, conocidas como instancias (*droplets* en DigitalOcean), ejecutan el proyecto a partir de una imagen de Docker generada específicamente para este propósito. La clave aquí es que esta imagen de Docker se ejecuta pasándole el archivo `.env`, que contiene todas las variables de entorno necesarias para el funcionamiento del proyecto.

Este enfoque tiene varias ventajas significativas. En primer lugar, permite modificar las variables de entorno de forma sencilla y segura en entornos productivos sin necesidad de alterar la imagen de Docker que contiene el código fuente del proyecto. Para realizar cambios en las variables de entorno, basta con modificar el archivo `.env` y luego volver a desplegar la imagen de Docker. Esto es crucial para realizar ajustes y correcciones sin

necesidad de volver a crear otra imagen de Docker.

Además, este enfoque mejora la seguridad al ocultar las credenciales y secretos, como las credenciales de acceso al bucket de AWS, la clave secreta de la base de datos y las credenciales de Sendgrid, que de otro modo podrían estar expuestos en el código fuente o en un archivo de configuración visible para cualquiera que tenga acceso al proyecto. Al mantener estas credenciales en un archivo `.env` separado y fuera del alcance directo del proyecto, se reduce el riesgo de exposición accidental o maliciosa.

Se crearon dos nuevas variables de entorno para habilitar o deshabilitar el uso de Sendgrid en el envío de correos electrónicos de bienvenida y notificaciones. La razón de esto fue que, durante el desarrollo y las pruebas, se alcanzó rápidamente el límite de 100 correos por día. Además, una vez que se alcanza este límite, los flujos de trabajo no funcionan correctamente, por lo que la mejor solución fue deshabilitar Sendgrid para estos casos. Estas variables se llaman `USE_SENDGRID_SERVICE` y `USE_SENDGRID_NOTIFICATION`, y por defecto están configuradas en `true`, para permitir los correos especificados anteriormente.

Se dedicó una considerable cantidad de tiempo a dichas tareas, pero es importante destacar que se pudo superar con éxito todos los desafíos que se presentaron y se pudo continuar avanzando con el desarrollo del proyecto. La actualización del código permitió utilizar las últimas funcionalidades y características de Ruby on Rails, lo que mejoró significativamente la calidad y la eficiencia del código. Además, al actualizar las gemas utilizadas en el proyecto, se mejoró la seguridad y se redujo la probabilidad de errores.

4.2. Aplicación Frontend

La aplicación Frontend origen, como se explicó en la Sección 2.5.2, fue desarrollada en el lenguaje JavaScript, utilizando la librería React.

La situación inicial era característica de un software que no se había mantenido en cinco años. Las versiones de las bibliotecas estaban considerablemente desactualizadas.

Comenzando por las más importantes, por ejemplo Webpack (herramienta de construcción o *build tool*): En su versión original, PSPcode empleaba Webpack 2.5, lanzada en mayo de 2017. Uno de los principales desafíos del proyecto consistió en actualizar Webpack a la versión 5, la última lanzada hasta la fecha, mientras se garantizaba el funcionamiento correcto de la aplicación.

Una situación similar se presentó con React, ya que la aplicación origen utilizaba React 15.5.4, lanzado en abril de 2016, y se actualizó a React 17 (marzo 2021).

Para un mejor contexto, cabe destacar que son 75 las librerías instaladas en la aplicación origen. Las mismas debieron ser actualizadas, logrando compatibilidad entre ellas: Al actualizar una librería, las dependencias del proyecto se modifican, por lo tanto se genera la necesidad de actualizar en cadena. Incluso, en varios casos hubo que reemplazar las librerías utilizadas por otras distintas que cumplieran objetivos similares, ya que no existían versiones compatibles de la librería original con la nueva versión sistema.

Por otro lado, muchas librerías modifican su sintaxis y los métodos que exponen entre una versión y la otra, por lo que cada cambio debía hacerse rigurosamente.

4.2.1. Proceso de Actualización de Librerías

Como se mencionó anteriormente, la actualización se focalizó en React y Webpack, siendo estas dos las principales bibliotecas sobre las que se desarrolla, construye y corre la aplicación. A partir de su actualización, modificamos el resto de las librerías instalando la última versión compatible para cada una.

4.2.1.1 Actualización Directa

En una fase inicial de este proceso, se realizó una transición directa de Webpack v2 a v5. Basados en experiencias previas en el desarrollo de software Frontend, se anticipaba que la versión 5 de Webpack no sería compatible con la sintaxis presente en los archivos de configuración de Webpack en ese momento (específicamente, `webpack.config.dev.js` y `webpack.config.prod.js`), y tampoco sería compatible con muchas de las bibliotecas en uso en el proyecto en ese momento.

Sin embargo, igualmente se decidió intentar hacer la actualización de Webpack (de v2 a v5) y modificar las librerías en cadena solucionando cada incompatibilidad de versión que Yarn (el manejador de paquetes) indicaba en la consola.

Se invirtió esfuerzo para que la aplicación funcione utilizando este enfoque, actualizando la versión de cada librería y realizando los cambios necesarios de sintaxis; en algunos casos incluso iterando más de una vez sobre distintas versiones de una misma librería.

Pasar directamente de Webpack v2 a v5 no fue la forma correcta de abordar la actualización. Al ejecutar `yarn add webpack` (instalar la última versión de Webpack), el manejador proporciona un listado de las librerías con problemas de compatibilidad, por ejemplo mensajes como: `warning "> webpack@5.21.0" has incorrect peer dependency "react@15.5.4"` indicando que la nueva versión de Webpack instalada no es compatible con la versión existente de React.

El objetivo era solucionar estos errores para eliminar las incompatibilidades entre versiones, pero se generaron dependencias circulares inconsistentes, que surgían a partir de querer hacer el cambio de manera brusca, saltándose modificaciones que debían realizarse en el proceso, perdiendo información.

Surgieron numerosas complicaciones a raíz de aquellas librerías obsoletas, de las cuales no existen versiones compatibles con Webpack 5. De haber hecho una actualización gradual, se habría obtenido, a tiempo, la información detallada sobre el conjunto de paquetes del proyecto debido al cuál la librería en cuestión deja de funcionar (y no existe una versión más actual) por lo tanto debe ser reemplazada. Al haber hecho un cambio de versión tan precipitado, era difícil rastrear el origen o la razón de muchos de los problemas presentes al intentar correr la aplicación.

A su vez, entre las distintas versiones de Webpack, existen muchos cambios de estructura, sintaxis a la hora de configurar la *build tool*. Por lo que era fundamental realizar migraciones paulatinas de la aplicación, pasando por distintas versiones de esta herramienta.

4.2.1.2 Actualización Gradual

Al tomar consciencia de este error, se optó por el enfoque de la actualización gradual.

Webpack 3

En primer lugar se migró de la versión 2.5.0 a Webpack 3 [35], se hicieron los cambios necesarios:

- Sintaxis y configuración de Webpack.
- Actualización de las otras librerías, buscando versiones compatibles con la nueva versión de Webpack y entre ellas.
- Reemplazo de librerías deprecadas por otras que cumplan la misma función.

Es importante mencionar que estos cambios implican refactorizaciones en el código y la forma de escribir la aplicación y la interfaz de usuario en sí, no solo en archivos de configuración.

Webpack 4 y React 16.8

Luego, se pasó a migrar la aplicación a Webpack 4 [36] y React 16.8 [37]. Este cambio, aparte de los pasos mencionados en la migración anterior, tiene como particularidad la versión de React a la que se llegó.

React 16.8 incorpora los renombrados *Hooks*, estos son métodos implementados en JavaScript y exportados por React, que permiten utilizar las características clásicas de dicha librería, como ser el manejo de estado, los efectos secundarios y el contexto, en componentes declarados como funciones, de esta manera puede evitarse el uso de clases para escribir los componentes de React.

Los componentes funcionales permiten una sintaxis más compacta, logrando un código más limpio. Por ejemplo, en ellos no es necesario el uso de la referencia `this`: En los componentes de clase, es necesario utilizar la palabra reservada `this` como referencia a la instancia de clase actual para obtener cualquier propiedad, estado, o ejecutar una función declarada en la clase (ejemplos de uso: `this.props.someProp`, `this.state.someState` o `this.someMethod()` respectivamente).

Por ello, el salto de React 15 a React 16.8 es crucial y permitiría a futuro realizar una *major refactor* en la aplicación de Frontend, transformando todos los componentes de tipo clase a funcionales. Las versiones exactas de las librerías en este punto del desarrollo se encuentran en el Anexo .3.1.

Haciendo foco en Webpack, para realizar la migración de v3 a v4 se siguieron los pasos yacentes en la documentación oficial de la librería ya citada: Se borraron varios *plugins* ya deprecados, y se cambiaron algunas *keys* del objeto `module.exports` en el archivo de configuración de Webpack para compatibilizar con la nueva versión.

Fueron 45 las librerías modificadas (actualizadas o reemplazadas) en esta iteración. Un caso interesante fue el de Babel. Todo el grupo de dependencias de Babel fue actualizado de su versión 6.x a 7.x [38]. Estos cambios no fueron triviales, ya que Babel deprecó sus librerías en la versión 6.

Hasta su versión 6, Babel era un conjunto de paquetes individuales con nombres como “babel-core”, “babel-loader”, “babel-preset-env”, etc. Sin embargo, este enfoque generaba problemas como fragmentación de los paquetes y dificultades en la gestión de las

dependencias.

Con el objetivo de simplificar la experiencia de uso y mejorar la mantenibilidad del proyecto, los desarrolladores de Babel decidieron adoptar una nueva estructura de paquetes, unificándolos y cambiando el nombre del paquete raíz de “babel” a “@babel”. Esto se hizo para seguir las convenciones de nomenclatura de los paquetes de npm y permitir una mejor organización y agrupación de los paquetes relacionados con Babel, incluyendo importantes mejoras y cambios en la arquitectura de Babel. En el Anexo .3.2 se encuentra el cambio completo de Babel en el `package.json`.

Webpack 5 y React 17

Luego de que la aplicación corrió correctamente para las versiones mencionadas de React, Webpack, y el resto de librerías, se pasó a hacer una tercera iteración de actualizaciones con base en Webpack 5 [39] y React 17.

Como se mencionó previamente, cada iteración conlleva actualizar el resto de las librerías utilizadas en la aplicación, para que todas las versiones resulten compatibles y tan modernas como sea posible. En ciertos casos, incluso es necesario hacer reemplazos de librerías obsoletas por otras que cumplan una función similar, lo que hace largo y tedioso el proceso.

Esta última iteración fue terminada el 29 de mayo de 2022. Cabe destacar que en marzo de ese año se publicó la versión 18 de React. Se leyó la documentación para analizar el *changelog* de la versión, y se decidió utilizar la v17 por varios motivos:

- Se espera que una versión recién publicada de un software no esté libre de *bugs* y problemas, por lo que si se utilizaba la versión 18, iba a ser necesario estar al tanto de noticias y nuevas sub-versiones, lo que significaría seguir realizando actualizaciones de librerías a la aplicación.
- Similar al punto anterior, cuando una versión es tan reciente, la comunidad que la utilizó y reportó problemas y soluciones al respecto, es mucho más chica, por lo que en caso de tener dudas o inconvenientes con el funcionamiento propio de React 18, iba a ser mucho más difícil buscar respuestas.
- Por otro lado, dado que PSPcode es una aplicación heredada, que utiliza librerías existentes desde (al menos) 2017, era muy probable, prácticamente seguro, que muchas de ellas no tuvieran versiones compatibles con React 18 aún (o incluso nunca), por lo que aumentaría la tasa de librerías a reemplazar por completo, incrementando la dificultad de la migración.
- A su vez, por esta misma razón (aplicación heredada), muchas de las ventajas de esta versión de React, como nuevos *hooks* y métodos expuestos por la librería, no iban a ser aprovechados al máximo, ya que para ello debería hacerse, aparte de la actualización de versión, reescrituras completas de los archivos.

La actualización de librerías involucra la aplicación holísticamente. Por ejemplo, un capítulo aparte podría ser el pasaje de AntD “3.0.0-alpha.13” (2017) [40] a AntD: “4.20.0” (última versión existente hasta el momento) [40]:

4.2.1.3 AntD

AntD es una librería de diseño de interfaz de usuario (UI) basada en React. Proporciona una gran cantidad de componentes de UI predefinidos y personalizables, como botones, menús, formularios, tablas, gráficos y otros. Este tipo de librerías se utiliza para construir aplicaciones web de manera más rápida y sin preocuparse por definir cada componente desde cero.

Dado que AntD era un pilar base de la aplicación origen, el cambio de versión implicó una significativa modificación de la sintaxis en prácticamente todos los archivos. Esta actualización fue necesaria para mantener la apariencia y funcionalidad previa, e incluso en casos para lograr que la aplicación Frontend siquiera ejecutara.

A modo de ejemplo, utilizando la versión anterior de AntD, todos los componentes de la librería estaban importados de la siguiente manera:

```
1 const Layout = require('antd/lib/layout');
2 const Tabs = require('antd/lib/tabs');
3 const Breadcrumb = require('antd/lib/breadcrumb');
```

Ahora deben ser importados de esta manera:

```
1 import { Layout, Breadcrumb, Tabs } from 'antd';
```

Vemos de esta forma, cómo la nueva versión de AntD se adapta a la nueva sintaxis de importación y exportación de módulos de JavaScript, introducida en ECMAScript 2015 (también conocido como ES6) como una forma estándar de importar y exportar código en JavaScript. Esta adaptación permite un código más moderno y compatible con las últimas versiones de JavaScript.

Luego, por ejemplo, la forma de importar y usar los íconos de AntD también cambia:

Antes

```
1 import { Icon } from '@ant-design/compatible';
2
3 \\ usage
4 <Icon type="check-circle" />
```

Ahora

```
1 import { CheckCircleOutlined } from '@ant-design/icons';
2
3 \\ usage
4 <CheckCircleOutlined />
```

Para cada ícono usado en la aplicación, hubo que buscar su reemplazo en la nueva versión.

Estos son solo ejemplos de la diversidad de cambios que hubo que hacer en la mayoría de los archivos de la aplicación.

4.2.1.4 Definición de `.nvmrc`

`.nvmrc` es un archivo de configuración utilizado en el contexto del entorno de desarrollo de Node.js (entorno de ejecución de JavaScript 2.3). La sigla “nvmrc” proviene de “Node Version Manager Runtime Configuration” (Configuración de tiempo de ejecución del Administrador de Versiones de Node). Su función principal es especificar la versión de Node.js que se debe utilizar para ejecutar una aplicación Frontend en un proyecto.

Cuando se trabaja en desarrollo Frontend, es común que una aplicación dependa de una versión específica de Node.js y de sus módulos o paquetes asociados. El archivo `.nvmrc` permite definir la versión exacta de Node.js que se requiere para que la aplicación funcione de manera consistente y sin problemas.

Si el proyecto cuenta con el archivo `.nvmrc`, los desarrolladores pueden correr el comando `nvm use` en la raíz del repositorio y automáticamente se seleccionará la versión de Node.js especificada en el archivo `.nvmrc` para instalar las librerías y ejecutar la aplicación.

En PSPcode, se estableció mediante un archivo `.nvmrc` que la versión utilizada de Node.js sería la 14.19.3 (Julio 2022), ya que fue la utilizada para realizar la actualización de las distintas librerías y funcionaba de manera correcta para correr la nueva aplicación. De esta manera se facilita ampliamente la transición a futuros desarrolladores.

4.2.2. Optimización de la aplicación

Luego de obtener una versión funcional y actualizada de la aplicación Frontend, quedaban dos tareas importantes por realizar: El pasaje de todos los componentes de tipo *class* a *functional components*, y la eliminación de archivos y librerías que ya no se estaban utilizando.

Durante todo el proceso de actualización de las librerías, se fue limpiando el código de los archivos modificados por razones migratorias, sin embargo, aún había numerosos componentes declarados en forma de clase, y muchos archivos y librerías que ya no se estaban utilizando.

4.2.2.1 Transición a Functional Components

La transición de componentes de clase a funcionales fue un proceso largo y crucial para la actualización del código Frontend. El mismo comenzó en marzo 2022 con las primeras refactorizaciones de la aplicación, y se continuó a lo largo de todo el proyecto, alternándose con el desarrollo de las nuevas funcionalidades. El 21 de Diciembre de 2022 se terminó de transicionar todos los archivos de *class* a *functional components*, dejando así a la aplicación libre de sintaxis antigua de React.

Globalmente hablando, para cada archivo se lleva a cabo el siguiente proceso:

1. Reemplazar la declaración de clase por una declaración de función, utilizando preferentemente *arrow functions* para una sintaxis más compacta y legible.
2. Eliminar el método `constructor` y manejar los estados con el hook `useState`.
3. Reemplazar los métodos del ciclo de vida por el hook `useEffect`, adaptando su comportamiento según la función que desempeñaban.
4. Corregir la sintaxis de los métodos utilizados en el componente de clase, considerando el uso de `bind` o funciones flecha para evitar problemas con el contexto `this`.

El detalle técnico y ejemplificado de la refactorización se encuentra en el Anexo [.3.3](#).

Esta actualización simplifica y moderniza la estructura del código, mejorando su legibilidad y mantenibilidad.

Es fundamental resaltar que los cambios no deben llevarse a cabo necesariamente en el orden presentado, pero es esencial que todos se completen correctamente antes de ejecutar la aplicación. No se permite ninguna forma de combinación entre la sintaxis propia de los componentes de clase y los componentes funcionales.

Este proceso se realizó para todos los componentes de la aplicación, siendo un total de 31 archivos prácticamente reescritos.

4.2.2.2 Eliminación de Código Muerto

Al finalizar el proceso de refactorización, se realizó una primera iteración de limpieza de librerías inutilizadas y archivos nunca importados. Esto se hizo utilizando el comando `npx unimported` [\[41\]](#).

Al ejecutar el comando `npx unimported` en la raíz del proyecto, la herramienta analiza el proyecto en busca de archivos o módulos que no estén siendo importados en ninguna parte del código. Estos archivos o módulos pueden ser eliminados de forma segura, lo que ayuda a reducir el tamaño y complejidad del proyecto, mejora la organización del código y facilita el mantenimiento a largo plazo.

El comando devuelve una lista de los archivos o paquetes descargados y no utilizados, lo que permite al desarrollador identificar y tomar decisiones informadas sobre qué archivos o paquetes pueden ser eliminados. Esta herramienta es una forma efectiva de mantener un código limpio y optimizado, eliminando cualquier código innecesario u obsoleto.

A partir del resultado de dicho comando, se fueron eliminando librerías y archivos (incluso carpetas enteras) que ya no se utilizaban, teniendo en cuenta todas las modificaciones hechas en la aplicación.

Los archivos y paquetes detectados como inutilizados se fueron eliminando de a uno, probando a correr la aplicación luego de hacer cada cambio.

En el Anexo [.3.4](#) se encuentra el detalle de las librerías y archivos eliminados en las distintas iteraciones.

Se agrega al Anexo [.3.5](#) una versión final del `package.json` luego de todos los cambios realizados.

4.2.3. Corrección en la configuración de Webpack para un build exitoso

El 28 de septiembre de 2022 se realizó la primera liberación al ambiente de prueba. Para lograr esto, hubo que corregir el build de la aplicación de Frontend. La configuración de Webpack era adecuada y funcionaba correctamente para ejecutar la aplicación en modo de desarrollo. Sin embargo, no se había realizado ninguna prueba de *building* hasta el momento.

Se consultó la documentación [42] buscando la configuración recomendada para hacer deploys a ambientes externos utilizando Webpack 5.

En particular, se reemplazaron los dos archivos de configuración existentes:

- `webpack.config.dev.js`
- `webpack.config.prod.js`

por tres nuevos archivos:

- `webpack.common.js`
- `webpack.dev.js`
- `webpack.prod.js`

En este punto se introduce la librería `webpack-merge`: la misma exporta la función `merge` [43] que utiliza Webpack para fusionar lo configurado en diferentes archivos y generar la configuración final que se utilizará durante el proceso de construcción (*build*) de la aplicación, brindando flexibilidad y modularidad al permitir la reutilización de módulos de configuración en diferentes contextos.

El archivo `webpack.common.js` define la mayoría de configuraciones y *plugins* utilizados en ambos ambientes (desarrollo y productivo); y dependiendo el entorno definido en el *script* que se corre al levantar o hacer el *build* de la aplicación, se agregan configuraciones particulares de un ambiente, utilizando el archivo `.dev` o `.prod`.

Primordialmente, en esta instancia se hicieron cambios en el archivo `.prod`, para que el *build* se hiciera correctamente, como modificaciones en la sintaxis, eliminación y reemplazo de *plugins*.

Con respecto al ambiente de desarrollo. Dado que ya estaba funcionando correctamente, los únicos cambios que se hicieron fueron de organización y nomenclatura en los archivos de configuración:

- `webpack.config.dev.js` pasa a ser `webpack.common.js`, donde se encuentra la mayoría de configuraciones necesarias.
- Se agrega el archivo `webpack.dev.js` para incluir las configuraciones exclusivas del ambiente de desarrollo: Los atributos `mode: 'development'` y `devtool: 'inline-source-map'` son removidos de `webpack.common.js`, y colocados en `webpack.dev.js`.

4.2.4. Mejoras estéticas y funcionales de la interfaz de usuario

De forma simultánea a la actualización de librerías y refactorizaciones de código, se realizaron diversas mejoras en la interfaz de usuario (*UI*) de manera iterativa incremental.

Parte de estas oportunidades de mejora fueron presentadas a la tutora en una primera instancia del proyecto. Se encuentra adjunta en el Anexo [.3.6](#) la lista completa de debilidades encontradas relativas al área de Frontend en una primera aproximación a la aplicación origen de PSPcode.

A su vez, más cambios fueron surgiendo mientras se trabajaba en los distintos componentes de la aplicación. En particular, al realizar la actualización de versión de la librería AntD [4.2.1.3](#).

En el Anexo [.3.7](#) se presentan los principales cambios realizados a la *UI* de la aplicación, en pos de mejorar la experiencia de usuario (UX) y la estética de PSPcode.

Capítulo 5

Implementación de nuevas funcionalidades

Uno de los objetivos principales del Proyecto de Grado fue incluir en PSPcode, funcionalidades que permitieran al equipo docente corregir los proyectos, para así, terminar de cerrar el flujo de trabajo dentro de la aplicación, y evitar la carga de archivos externos para dar *feedback*.

Como se describió anteriormente, en la versión origen los estudiantes podían entregar el trabajo realizado y los profesores podían verlo. Sin embargo, a la hora de proveer un *feedback*, las docentes debían generarlo por fuera de la aplicación (típicamente en un documento de texto), y luego subir el archivo para que el estudiante lograra verlo.

En esta versión de la aplicación, se ha logrado proporcionar a los usuarios de rol Docente no solo una funcionalidad que les permite cargar el *feedback* en la aplicación, sino también la inclusión de sugerencias predefinidas. Esto tiene como objetivo hacer que el proceso de corrección sea más cómodo y sencillo para el usuario.

En el Anexo .4 se encuentra la versión completa de la *grading* que las profesoras rellenan y envían a sus estudiantes como forma de devolución a un proyecto.

5.1. Análisis de criterios de corrección automatizables

El proceso comenzó por el análisis de requerimientos junto a la tutora del proyecto y encargada del curso PF-PSP: Silvana Moreno. Su objetivo principal era poder realizar todas las tareas referentes a los proyectos de manera más cómoda desde la aplicación, y automatizar ciertos criterios que eran fundamentalmente numéricos.

Había esencialmente dos requerimientos globales:

1. Recibir sugerencias por parte de la aplicación, con respecto a todos los puntos de la *grading* que no necesiten una visión subjetiva, es decir, que tengan un resultado definido y obtenible a partir de cálculos matemáticos.
2. Contar con una pantalla dedicada a la corrección de un proyecto, donde se pudiese:
 - Hacer comentarios sobre cada punto de la *grading*.

- Definir si un proyecto está aprobado o necesita reentrega.
- Dejar un comentario general del proyecto.

En primer lugar se hizo un análisis de la planilla que Silvana y Leticia utilizaban como guía y *template* de corrección. A partir de la misma, y de numerosos intercambios de ideas, se fue estableciendo cuáles de los criterios de corrección eran posible automatizar.

Se mantuvo un documento compartido con la tutora, y se utilizó un código de colores para definir qué criterios de la planilla podrían ser automatizados, y cuáles no, ya que involucran interpretación y juicio del lenguaje natural, que excede el alcance del proyecto.

En verde se marcaron los criterios para los cuales no había dudas y podían ser calculados de manera programática. En amarillo aquellos de los cuales se tenían dudas sobre a qué referían, por lo tanto no podía definirse de que forma automatizarlos. Y En rojo aquellos que dependían del juicio subjetivo de un docente para su corrección.

Seguidamente, se tuvo un encuentro con Silvana para validar dicha clasificación, y aclarar las dudas.

5.1.1. Determinación de criterios automatizables

Algunos de los criterios existentes en la *grading* no eran utilizados, por lo que directamente se descartaron. De los utilizados:

Los presentes en la categoría “**Programa y Resultado de Test**” no fueron automatizados, ya que refieren específicamente a la escritura y ejecución del programa entregado por el estudiante, por lo que no era información escrita en los formularios de la aplicación, sino, almacenada en base y descargada por las docentes en forma de *zip*.

A continuación se presenta cada sección de la *grading* junto a qué criterios (de los existentes en ellas) fueron automatizados, y cuáles no. Cabe aclarar que, si bien se proporciona una descripción de cada uno de los criterios automatizados, se utilizan los títulos literales yacentes en la planilla.

Se especifican los requerimientos exactos de cada criterio en la sección 5.2.

De la sección “**Logs de Tiempo**” se pudieron automatizar todos los criterios, salvo el último: “Los datos fueron registrados a medida que se realizaba el trabajo.”, ya que era algo imposible de saber, no hay forma, desde la aplicación, de controlar si el estudiante registró los tiempos mientras realmente realizaba las distintas fases.

Se automatizan:

- “Los datos del tiempo se registran en todas las fases del proceso”
- “Las etapas del proceso siguen una secuencia adecuada.”
- “Los tiempos son registrados en los pasos correctos del proceso”
- “Los tiempos de las interrupciones son registrados de forma apropiada”
- “Los datos del tiempo son completos y razonables”

Por otro lado, de los “**Logs de defectos**”. Pudieron implementarse sugerencias para:

- “Todos los defectos tienen la información de registro solicitada”
- “Los defectos fueron inyectados antes de ser removidos”
- “Todos los defectos tienen tiempo de corrección”
- “Los defectos inyectados en compilación y test tienen *fix defect* asociado.”
- “El tipo de defecto es consistente con la fase de inyección”

No se automatizan “Los defectos son descritos adecuadamente.” y “El tipo de defecto es consistente con la descripción.”, ya que dependen de descripciones en lenguaje natural.

Con respecto al “**Formulario PIP**”:

- “El formulario PIP está completo” se automatiza tomando un mínimo de caracteres.
- “Los registros del PIP muestran perspicacia y reflexión” no se automatiza.

En la sección “**Planning Summary**” se automatizan ambos criterios:

- “El tiempo total planificado fue ingresado correctamente”
- “Los tamaños planificado y actual están ingresados correctamente”

En la sección “**Chequeo de Consistencia**” también se automatizan todos los criterios:

- “Los defectos removidos son consistentes con los tiempos en las fases de compilación y test”
- “El total del tiempo de corrección en compilación es menor que el tiempo total dedicado a la compilación”
- “El total de tiempo de corrección en test es menor que el tiempo dedicado al test”

Para finalizar, ningún criterio de la categoría “**General**” es automatizado, ya que son devoluciones generales que da el docente sobre el trabajo del estudiante.

5.1.2. Definición de tareas de desarrollo para las automatizaciones

Durante el análisis de requisitos se relevó que algunos criterios sean advertidos a los estudiantes durante su desarrollo; es decir, que dentro del formulario de fase, se solicite al estudiante realizar cierta corrección antes de entregar. Por ejemplo, que una etapa de un proyecto no pueda durar más de 24 horas, y que ningún campo de registro de tiempo pueda estar vacío.

Cada una de estas tareas fue agregada al tablero de Jira bajo la épica “**Grading Automation**”, donde dividimos los *tickets* en tareas de Backend y tareas de Frontend.

Dado que el sistema debe desplegar un mensaje o alerta en el caso de que un criterio no sea cumplido, **todas** las correcciones automatizadas implican tareas en el área de Frontend. Sin embargo, no todas requieren tareas en el área de Backend: si la corrección debe ser comunicada al estudiante antes de que éste realice la entrega del trabajo, la tarea recae exclusivamente en el Frontend. En dicho caso, los datos a corregir nunca llegan a las instancias de revisión docente, por lo tanto no deben guardarse en la base de datos como parte de un trabajo terminado, dado que la aplicación directamente no debe permitir la entrega del mismo con esos errores.

Por otro lado, los algoritmos para definir el incumplimiento de los criterios que solo son notificados al docente, son implementados en el lado del servidor. Cuando el docente carga el trabajo ya entregado por el estudiante, se le devuelve una advertencia en caso de que no se cumpla algún criterio establecido. En la sección 5.2.2 se presenta el diseño UI de las mismas.

5.2. Implementación de observaciones automáticas

5.2.1. Tareas Backend

Del lado del Backend, solo se implementaron aquellas observaciones que debe ver el profesor, ya que éstas dependen de lógica basada en distintas entidades modificadas en la base de datos.

Para realizar esto, se modificó el JSON de respuesta de la entidad `phase_instance` (la que representa una instancia de fase) ya que contiene toda la información necesaria, como el proyecto asociado, todas sus fases y sus datos de entrada, como el tiempo total registrado en cada una de las fases, el tiempo de inicio y de fin, entre otras. Además se agregaron observaciones en los defectos, por lo que se modificó en este caso el JSON de la entidad `defects`. Se agregó en cada JSON, un atributo nuevo llamado `observations`, el cual es un objeto con distintos atributos correspondientes a cada observación.

Además, este atributo solo es visible en la respuesta si el que genera la solicitud es el docente. Esto implica que sólo los profesores pueden ver estas observaciones. Se decidió hacerlo de esta manera en lugar de ocultar la respuesta del lado del Frontend, ya que si ese atributo fuera enviado siempre, sería visible de manera pública por los estudiantes. Estos podrían inspeccionar el JSON mostrado desde el navegador y obtener ayuda sobre las cosas que se podrían estar haciendo de manera incorrecta.

5.2.1.1 Algoritmos de las observaciones

Implementación

A continuación se explican todos los métodos utilizados para calcular cada una de las siguientes observaciones. Estos métodos se utilizan tanto para las observaciones de las fases, como para calcular observaciones en los criterios que pueden ser automatizados del lado del docente:

- **build_elapsed_time_obs**: este método realiza una serie de comprobaciones relacionadas con el tiempo transcurrido en las etapas que no son `CODE`.

Verifica si la fase (**phase**) está presente y si no es igual a la fase de código (**CODE**). Luego, se verifica si el tiempo total de la fase sin tener en cuenta el tiempo de descanso (**total_without_break_time**) es mayor a un día (**ONE_DAY**).

Si se cumplen todas estas condiciones, el método retorna el mensaje **“The stage lasts more than 24 hours.”** (La etapa dura más de 24 horas).

Este pseudocódigo representa la funcionalidad del código proporcionado de manera simplificada:

```
1 Función build_elapsed_time_obs(phase, total_without_break_time):
2     Si phase está presente Y phase es diferente de CODE Y
3     total_without_break_time es mayor a ONE_DAY Entonces:
4         Retornar "The stage lasts more than 24 hours."
```

Este algoritmo se utiliza para calcular el criterio de la sección **“Log de Tiempo”** cuya descripción es **“Los datos del tiempo son completos y razonables”**.

- **build_fix_time_obs:** este método compara el tiempo de corrección (**total_fix_time**), destinado a la corrección de los defectos detectados en la fase, con el tiempo total de transcurso de la misma sin tener en cuenta el tiempo de interrupción (**total_without_break_time**).

Si el tiempo de corrección es mayor que el tiempo total de la fase, el método retorna el mensaje **“Fix time is greater than stage total time.”** (El tiempo de corrección es mayor que el tiempo total de la etapa).

El siguiente pseudocódigo representa la funcionalidad del código proporcionado de manera simplificada.

```
1 Función build_fix_time_obs(total_fix_time, total_without_break_time):
2     Si total_fix_time es mayor a total_without_break_time Entonces:
3         Retornar "Fix time is greater than stage total time."
4
```

Este algoritmo se utiliza para calcular el criterio de la sección **“Chequeo de Consistencia”** cuya descripción es **“El total del tiempo de corrección en cada fase es menor que el tiempo total dedicado a la misma”**.

- **build_break_time_obs:** este método compara el tiempo de interrupción o de descanso que se tomó en la fase (**interruption_time**) con el tiempo total de la fase sin tener en cuenta el tiempo de corrección (**total_without_fix_time**).

Si el tiempo de interrupción es mayor que el tiempo total de la fase, el método retorna el mensaje **“The interruption time shouldn’t be that long.”** (El tiempo de interrupción no debería ser tan largo).

El siguiente pseudocódigo representa la funcionalidad del código proporcionado de manera simplificada.

```
1 Función build_break_time_obs(interruption_time, total_without_fix_time):
2     Si interruption_time es mayor a total_without_fix_time Entonces:
3         Retornar "The interruption time shouldn't be that long."
4
```

Este algoritmo se utiliza para calcular el criterio de la sección “**Log de Tiempo**” cuya descripción es “**Los tiempos de las interrupciones son registrados de forma apropiada**”.

- **build_plan_time_obs**: este método realiza una serie de comprobaciones relacionadas con el tiempo estimado de la fase de plan.

Verifica si la fase (**phase**) está presente y si es del tipo plan (**PLAN**). Si no se cumplen ambas condiciones, retorna sin hacer nada.

Luego, verifica si el tiempo estimado (**plan_time**) del proyecto no está presente o si es igual a cero (**ZERO**). Si se cumple alguna de estas condiciones, retorna el mensaje “**The estimated time is not defined.**” (El tiempo estimado no está definido).

Finalmente, si el tiempo estimado es mayor que 24 horas (**ONE_DAY**), retorna el mensaje “**The estimated time is greater than 24hrs.**” (El tiempo estimado es mayor a 24 horas).

El siguiente pseudocódigo representa la funcionalidad del código proporcionado de manera simplificada.

```
1 Función build_plan_time_obs(phase, plan_time):
2     Si phase no está presente 0 phase es diferente de PLAN Entonces:
3         Retornar "No action taken."
4
5     Si plan_time no está presente 0 plan_time es igual a ZERO Entonces:
6         Retornar "The estimated time is not defined."
7
8     Si plan_time es mayor a ONE_DAY Entonces:
9         Retornar "The estimated time is greater than 24hrs."
10
```

Este algoritmo se utiliza para calcular el criterio de la sección “**Planning Summary**” cuya descripción es “**El tiempo total planificado fue ingresado correctamente**”.

- **build_empty_loc_obs**: este método verifica si las líneas de código planificadas están definidas en la fase de plan.

Verifica si la fase (**phase**) está presente y si es del tipo plan (**PLAN**). Luego, se verifica si las líneas de código planificadas (**plan_loc**) no están presentes o si son igual a

cero (ZERO).

Si se cumplen todas estas condiciones, el método retorna el mensaje “**Plan LOCs are not defined.**” (Las líneas de código planificadas no están definidas).

A continuación se presenta el pseudocódigo.

```
1 Función build_empty_loc_obs(phase, plan_loc):  
2     Si phase está presente Y phase es igual a PLAN Y (plan_loc no está presente O  
3     plan_loc es igual a ZERO) Entonces:  
4         Retornar "The planned Lines of Code (LOC) are not defined."
```

Este algoritmo se utiliza para calcular el criterio de la sección “**Planning Summary**” cuya descripción es “**Los tamaños planificado y actual están ingresados correctamente.**”

- **build_empty_total_obs:** en este caso, se verifica si el tiempo total del proyecto está definido en la fase de post mortem.

Verifica si la fase (**phase**) está presente y si es igual a post mortem (**POST_MORTEM**). Además, se verifica si el tiempo total (**total**) no está presente o si es igual a cero (**ZERO**).

Si se cumplen todas estas condiciones, el método retorna el mensaje “**Total project time is not defined.**” (El tiempo total del proyecto no está definido).

A continuación se muestra el pseudocódigo.

```
1 Función build_empty_total_obs(phase, total):  
2     Si phase está presente Y phase es igual a POST_MORTEM Y (total no está  
3     presente O total es igual a ZERO) Entonces:  
4         Retornar "Total project time is not defined."
```

Este algoritmo se utiliza en el criterio de la sección “**Planning Summary**” cuya descripción es “**Los tamaños planificado y actual están ingresados correctamente.**”

- **build_time_conflicts_obs:** en este método, se realiza un conjunto de comprobaciones basadas en la lista de todas las instancias de fase (**phase_instances**). Estas comprobaciones detectan conflictos de tiempo entre todas las fases y la fase actual, y generan mensajes de conflicto correspondientes.

En primer lugar, se inicializan las variables **res1** y **res2** como nulas. Luego, se verifica si **phase_instances** está presente. Si no lo está, se retorna [**res1**, **res2**] como una lista vacía.

A continuación, se utiliza un bucle `for each` para iterar sobre cada instancia de fase (`phase_instance_it`) en `phase_instances`. Se realizan diversas comprobaciones utilizando las propiedades y métodos de las instancias de fase.

Si el `id` de la fase de la iteración (`phase_instance_it.id`) es igual al `id` actual de la fase (`id`), se establece la bandera `post_phases` como verdadera, indicando que todas las fases que vienen a continuación en el bucle son posteriores a la actual, y se continúa con la siguiente iteración.

Si se cumple la condición `phase_instance_it.start_time <= end_time`, se establece `first_check` como verdadero, indicando que la fase de esta iteración comienza antes de que termine la fase actual. De manera similar, si se cumple la condición `phase_instance_it.end_time >= start_time`, se establece `second_check` como verdadero, indicando en este caso que la fase de esta iteración finaliza después de que comience la fase actual.

Dentro del bucle, se realizan diferentes verificaciones según el estado de las banderas `post_phases`, `first_check` y `second_check`. Se construyen cadenas de texto (`res1` y `res2`) basadas en el resultado de estas verificaciones y el nombre de las fases correspondientes.

Si `post_phases` es verdadero y `first_check` es verdadero, se agrega al mensaje `res1` la frase “**This phase ends after {nombre de fase de la iteración}**” (Esta fase termina después de {nombre}).

Si `post_phases` es falso y `second_check` es verdadero, se agrega al mensaje `res2` la frase “**This phase starts before {nombre de fase de la iteración}**” (Esta fase comienza antes de {nombre}).

Al final del bucle, se agregan las terminaciones “starts” o “ends” a las cadenas `res1` y `res2` si están presentes.

Por último, se retorna [`res1`, `res2`], que contiene los mensajes generados durante las comprobaciones.

A continuación se muestra el pseudocódigo.

```
1  Función build_time_conflicts_obs(phase_instances):
2      Inicializar res1 y res2 como nulos
3
4      Si phase_instances está presente Entonces:
5          post_phases es falso
6          first_check es falso
7          second_check es falso
8
9      Para cada phase_instance_it en phase_instances Entonces:
10         Si phase_instance_it.id es igual a id Entonces:
11             post_phases es verdadero
12             Continuar
13
14         nombre es phase_instance_it.phase.name
```

```

15     first_check es (phase_instance_it.start_time es menor o igual a
    end_time)
16     second_check es (phase_instance_it.end_time es mayor o igual a
    start_time)
17
18     Si post_phases Entonces:
19         Si first_check Entonces:
20             Si res1 está presente Entonces:
21                 res1 es res1 + ", This phase ends after #{nombre}"
22             Sino:
23                 res1 es "This phase ends after #{nombre}"
24
25         Sino Entonces:
26             Si second_check Entonces:
27                 Si res2 está presente Entonces:
28                     res2 es res2 + ", This phase starts before #{nombre}"
29                 Sino Entonces:
30                     res2 es "This phase starts before #{nombre}"
31
32     Si res1 está presente Entonces:
33         res1 es res1 + ' ends'
34
35     Si res2 está presente Entonces:
36         res2 es res2 + ' starts'
37
38     Retornar [res1, res2]
39

```

Este algoritmo se utiliza en el criterio de la sección “**Log de tiempo**” cuya descripción es “**Los tiempos son registrados en los pasos correctos del proceso**”.

Todos los métodos anteriores se utilizan en las observaciones referidas a las fases. A continuación se describen los métodos relacionados a las observaciones de los defectos:

- **build_discovered_time_fit_obs:** en este caso, el código de este método verifica si el tiempo en que se descubrió el defecto se encuentra dentro del tiempo total de su fase.

Verifica si la instancia de fase (`phase_instance`) está presente, así como si lo están su tiempo de inicio (`start_time`) y tiempo de finalización (`end_time`). También se verifica si el tiempo en que se descubrió el defecto (`discovered_time`) está presente.

Luego, se verifica si el `discovered_time` es menor que el `start_time` o si es mayor que el `end_time` de la fase. Si se cumple alguna de estas condiciones, se retorna el mensaje “**The discovered time is outside the phase’s total time.**” (El tiempo descubierto está fuera del tiempo total de la fase).

Finalmente, si no se cumple ninguna de las condiciones anteriores, se retorna nulo.

A continuación se muestra el pseudocódigo.

```

1     Función build_discovered_time_fit_obs():
2         Si phase_instance está presente
3         Y phase_instance.start_time está presente
4         Y phase_instance.end_time está presente

```

```

5     Y discovered_time está presente
6     Y (discovered_time es menor a phase_instance.start_time 0 discovered_time es
    mayor a phase_instance.end_time) Entonces:
7         Retornar "The discovered time is outside the phase's total time."
8     Sino Entonces:
9         Retornar nulo
10

```

Este algoritmo se utiliza en el criterio de la sección “Chequeo de Consistencia” cuya descripción es “Los defectos removidos son consistentes con los tiempos de la fase.”

- **build_phase_injected_obs:** en este punto, se verifica si la fase donde se inyectó un defecto (`phase_injected`) es anterior a la fase actual en una lista de instancias de fase (`phase_instances`).

Verifica si `phase_instances` está presente. Luego, se inicializan las variables `post_phases` y `has_same_name_before` como falso.

A continuación, se itera sobre cada fase (`phase_instance_it`) en `phase_instances`. Si el identificador (`id`) de `phase_instance_it` es igual al identificador de la `phase_instance` actual, se establece `post_phases` como verdadero, indicando que las fases siguientes en la iteración son posteriores a la actual.

Luego, se verifica si `phase_injected` está presente, si `phase_injected.name` está presente, si `phase_instance_it.phase` está presente y si `phase_instance_it.phase.name` está presente. Además, se verifica si el nombre de `phase_injected` es igual al nombre de `phase_instance_it.phase`, es decir, se encuentra entre todas las fases la que coincide con el nombre de la fase donde se inyectó el defecto.

Si se cumple la condición anterior, `post_phases` es verdadero y `has_same_name_before` es falso, entonces se retorna el mensaje “**The phase where the defect was injected should be prior to this one**” (La fase donde se inyectó el defecto debería ser anterior a esta).

Si `post_phases` es falso, se establece `has_same_name_before` como verdadero.

Finalmente, si no se cumplen las condiciones anteriores, se retorna nulo.

A continuación se presenta el pseudocódigo explicado anteriormente.

```

1     Función build_phase_injected_obs(phase_instances):
2         Si phase_instances está presente Entonces:
3             post_phases es falso
4             has_same_name_before es falso
5
6         Para cada phase_instance_it en phase_instances Entonces:
7             Si phase_instance_it.id es igual a phase_instance.id Entonces:
8                 post_phases es verdadero
9
10            Si phase_injected está presente

```

```

11         Y phase_injected.name está presente
12         Y phase_instance_it.phase está presente
13         Y phase_instance_it.phase.name está presente
14         Y phase_injected.name es igual a phase_instance_it.phase.name
    Entonces:
15         Si post_phases es verdadero
16         Y has_same_name_before es falso Entonces:
17             Retornar 'The phase where the defect was injected should be
prior to this one'
18         Sino Si post_phases es falso Entonces:
19             has_same_name_before es verdadero
20     Sino Entonces:
21         Retornar nulo
22

```

Este algoritmo se utiliza en el criterio de la sección “Log de Defectos” cuya descripción es “Los defectos fueron inyectados antes de ser removidos”.

Modo de uso

Para especificar el algoritmo utilizado en cada criterio, se cuenta con el atributo `algorithm` en el modelo del Criterio, en donde se escribe el nombre del método que implementa el algoritmo sin los *strings* “_build” y “_obs”.

También se debe especificar el tipo de algoritmo que se está utilizando, esto es, si es un algoritmo que resuelve una observación de una fase o de un defecto. Esto se logra con el atributo `algorithm_type` cuyos valores pueden ser 0 o 1, especificando que es del tipo fase o del tipo defecto respectivamente. Además, si se quisiera ejecutar más de un algoritmo, se deben separar los *strings* con la palabra `and`.

Ejemplo, si se quisiera ejecutar el método “`build_elapsed_time_obs`”, se selecciona el algoritmo “`elapsed_time`”. Por otro lado, si se quisieran ejecutar los métodos “`build_empty_loc_obs`” junto con “`build_empty_total_obs`” se debe seleccionar el algoritmo “`empty_loc_and_empty_total`” (Nótese el `and` entre medio de los dos nombres de los métodos).

5.2.2. Tareas Frontend

Por otro lado, se realizó un conjunto de tareas del lado del Frontend, incluyendo: Trabajo de interfaz de usuario (mensajes de advertencia) para explicitar las observaciones obtenidas del servidor y mostrarlas al docente en etapa de corrección. Como también lógica y UI para aquellos errores cometidos por el estudiante que no deben propagarse a etapa de corrección.

A continuación se detalla la implementación para cada criterio, tomando como título, los nombres literales utilizados por los docentes del curso en la actual *grading* de corrección.

5.2.2.1 Observaciones para el Rol Docente

Como se mencionó, estas observaciones corresponden a aquellas advertencias que son presentadas al docente en etapa de corrección. La lógica de las mismas fue implementada

del lado de Backend, y detallada en la sección anterior. En este caso, el trabajo del Frontend es evaluar la información del Backend, y si existe una observación, desplegarla en la pestaña de corrección del proyecto.

El Frontend obtiene del Backend, en la información de cada fase (`activePhase`), un objeto llamado `observations`, el mismo posee el resultado de los distintos algoritmos de chequeo propios de la fase, como se puede ver en la siguiente desestructuración:

```
1  const {
2    break_time,
3    elapsed_time,
4    fix_time,
5    empty_loc,
6    plan_time,
7    empty_total,
8    time_conflict_start,
9    time_conflict_end,
10 } = activePhase.observations || {
11   break_time: null,
12   elapsed_time: null,
13   fix_time: null,
14   empty_loc: null,
15   plan_time: null,
16   empty_total: null,
17   time_conflict_start: null,
18   time_conflict_end: null,
19 };
```

Por otro lado, también hay criterios que, en lugar de ser propios de una fase, son propios de un defecto dentro de la fase. Los mismos son obtenidos del Backend como un objeto:

```
1  defect: {
2    observations: {
3      discovered_time_fit,
4      phase_injected,
5    }
6  }
```

A continuación se presentan las observaciones ordenadas por criterio, como se usan y como se muestran en la interfaz de usuario.

Observaciones Logs de Tiempo

De la categoría de la *grading* “**Log de Tiempo**”, los criterios que fueron automatizados para renderizar una observación del lado del profesor fueron los siguientes:

- **Los datos del tiempo son completos y razonables:** en este caso, cuando el profesor accede a la vista de un proyecto entregado por un estudiante, se exhibirá un mensaje de advertencia en cada fase, excepto en la de Codificación, si la duración de la fase excede las 24 horas (calculada como tiempo de fin menos tiempo de inicio menos tiempo de descanso). Esto se considera incoherente, ya que es un tiempo demasiado largo para completar las tareas de una sola fase.

Se puede ver en la Sección 5.2.1.1 cómo es implementada la lógica del lado del Backend. Del lado del Frontend, se obtiene dentro de la información de la fase en cuestión, el atributo `elapsed_time` del objeto `observations`. Este atributo posee o bien un mensaje de error (que se despliega directamente), o bien `undefined`, caso en el cuál se asume que no hay error sobre el criterio.

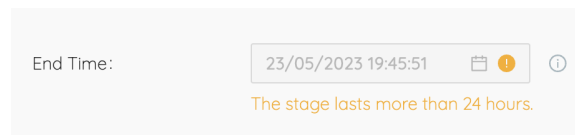


Figura 5.1: En el campo End Time de cada fase, el profesor verá este mensaje si el tiempo total de la fase supera las 24 horas.

- **Los tiempos de las interrupciones son registrados de forma apropiada:** de igual forma que para el criterio anterior, la información se obtiene de un atributo del objeto `observations`. Esta vez, el mensaje de error está dado por el atributo `break_time`. La lógica y resultado del algoritmo de Backend está explicado en la Sección 5.2.1.1.

A continuación podemos ver cómo el Frontend muestra el caso de error:



Figura 5.2: En el campo Interruption Time de cada fase, el profesor verá este mensaje si el tiempo total de la fase supera el tiempo máximo predefinido.

- **Los tiempos son registrados en los pasos correctos del proceso:** esta vez, el mensaje de error es dado por los atributos `time_conflict_start` y/o `time_conflict_end`.

En la siguiente figura se ve un caso donde el Backend envía un mensaje de error si existe **al menos** una fase (en este ejemplo: UNIT TEST) que esté previa en la línea de tiempo a la fase visualizada, pero su fecha de fin sea posterior a la de inicio de la fase abierta.

The screenshot shows a configuration panel for a phase. At the top, there is a dropdown menu labeled 'Phase:' with 'POST MORTEM' selected. Below it is a date and time field labeled 'Start Time:' with the value '21/05/2023 19:45:47'. At the bottom of the panel, a yellow warning box contains the text: 'This phase starts before UNIT TEST ends.' with a close button 'X'.

Figura 5.3: Debajo del tiempo de comienzo de la fase Unit Test, se ve una advertencia indicando el conflicto.

Por otro lado, el caso donde `time_conflict_end` trae un mensaje de error, se da cuando la fase actual tiene una fecha de fin posterior a la de comienzo de alguna fase (en este caso: POST MORTEM) que se encuentre posterior a ella en la línea de tiempo.

The screenshot shows a configuration panel for a phase. At the top, there is a date and time field labeled 'End Time:' with the value '24/05/2023 19:45:19'. Below it is a field labeled 'Int. time:'. At the bottom of the panel, a yellow warning box contains the text: 'This phase ends after POST MORTEM starts.' with a close button 'X'.

Figura 5.4: Debajo del tiempo de fin de la fase Post Mortem, se ve una advertencia indicando el conflicto.

En el caso de que exista más de una fase en conflicto (tanto para adelante como para atrás en la línea del tiempo), el mensaje de error mostrará el nombre de todas las fases involucradas separadas por una coma:

The screenshot shows a configuration panel for a phase. At the top, there is a dropdown menu labeled 'Phase:' with 'CODE' selected. Below it is a date and time field labeled 'Start Time:' with the value '01/02/2023 13:35:37'. At the bottom of the panel, a yellow warning box contains the text: 'This phase starts before PLAN, DESIGN ends.' with a close button 'X'.

Figura 5.5: En este caso, la fase que está siendo visualizada empieza antes de que Plan y Design terminen, mientras en la línea del tiempo se encuentra ubicada antes.

Observaciones Logs de Defectos

- Los defectos fueron inyectados antes de ser removidos:** en este caso, como fue explicado previamente, el mensaje de error (si existe) es obtenido mediante la información propia del defecto, no de la fase.

Este criterio está verificado por el atributo `phase_injected`, que devuelve un mensaje de error en los casos donde el estudiante marque como fase de inyección del defecto una fase que no exista antes de la fase actual en donde está reportando el defecto.

Por ejemplo, si el estudiante reporta un defecto en la fase de CODE, y define que dicho defecto fue inyectado en COMPILE, pero en su *timeline* no existe ninguna fase COMPILE previa a la fase de CODE en la que se encuentra, entonces el sistema mostrará un mensaje de error al profesor, cuando el mismo esté corrigiendo el proyecto.

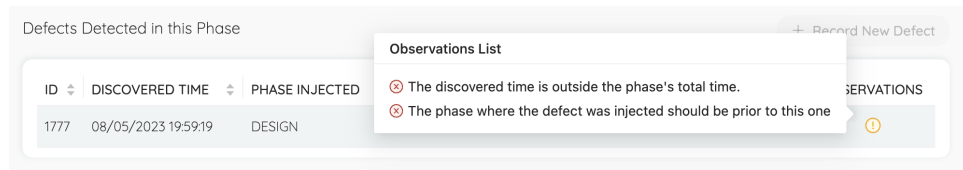


Figura 5.6: En la figura pueden verse dos errores, el segundo corresponde al criterio descrito. El primero al criterio “Los defectos fueron inyectados antes de ser removidos” que será explicado en errores de Chequeo de Consistencia.

Observaciones Planning Summary

- El tiempo total planificado fue ingresado correctamente:** este criterio tiene dos posibles mensajes de error referidos al tiempo planeado para el proyecto: “The estimated time is not defined.” (El tiempo estimado no está definido), y “The estimated time is greater than 24hrs.” (El tiempo estimado es mayor a 24hrs).

Ambos casos son tomados como casos de error, por lo tanto un mensaje es mostrado al profesor en etapa de corrección.

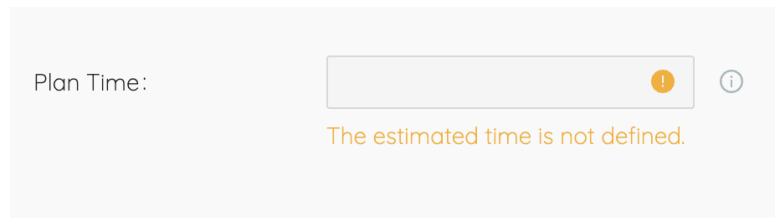


Figura 5.7: En la figura se muestra la advertencia desplegada en el caso donde el tiempo planeado no es definido por el estudiante.

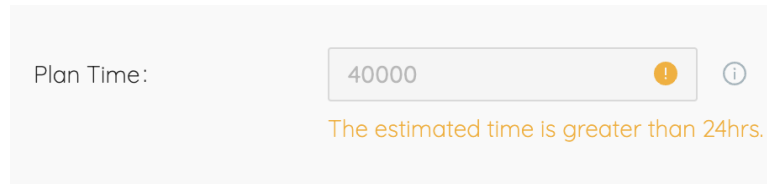


Figura 5.8: En la figura se muestra la advertencia desplegada en el caso donde el tiempo planeado es mayor a 24 horas.

- Los tamaños planificado y actual están ingresados correctamente:** el criterio tiene una semántica muy similar a la anterior. Con respecto a este criterio, parte de él es evaluado en momento de corrección y otra cuando el estudiante está realizando el proyecto. En el caso del campo “Plan LOCs” (líneas de código estimadas para el proyecto), el mensaje de advertencia es generado en el Backend y mostrado en etapa de corrección al docente.

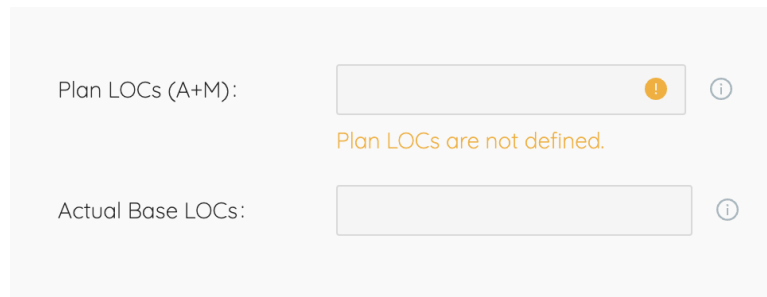


Figura 5.9: En la figura se muestra la advertencia desplegada en el caso donde la cantidad de líneas planeada no fue definida por el estudiante.

Observaciones de Chequeo de Consistencia

- El total del tiempo de corrección en cada fase es menor que el tiempo total dedicado a la misma:** este criterio de la *grading* refiere a que el tiempo total dedicado a la corrección de los defectos de una fase, no puede ser igual o mayor al tiempo total de esta fase. El mensaje de error, si lo hubiera, está dado en el atributo `total_fix_time` del objeto `observations` obtenido del Backend.

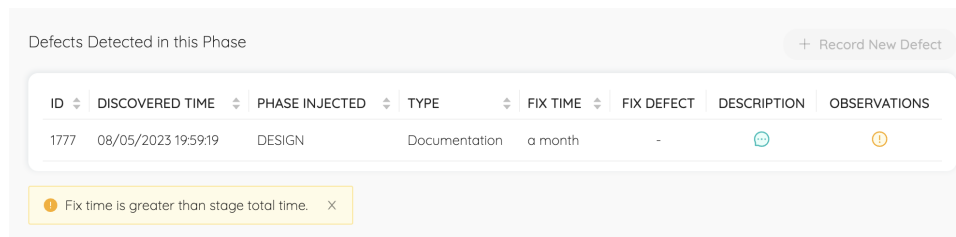


Figura 5.10: Así es mostrado el mensaje de advertencia cuando el tiempo dedicado al arreglo de los defectos, supera el tiempo total de la fase.

- **Los defectos removidos son consistentes con los tiempos en las fases de compilación y test:** el resultado de este criterio, al igual que “**Los defectos fueron inyectados antes de ser removidos**” es traído de Backend en la información propia del defecto. En este caso, en el atributo `discovered_time_fit`, y el error se da cuando la fecha de descubrimiento del defecto definida por el estudiante, no se encuentra entre la fecha de inicio y fin de la fase en cuestión.

El mensaje de error puede verse en la figura 5.6, ya presentada anteriormente para demostrar el manejo del criterio “**Los defectos fueron inyectados antes de ser removidos**”.

5.2.2.2 Observaciones para el Rol Estudiante

Observaciones Log de Tiempo

Se definió que los criterios “**Los datos del tiempo se registran en todas las fases del proceso**” y “**Las etapas del proceso siguen una secuencia adecuada**”, fueran observados en etapa de realización del proyecto, y no se le permitiera al estudiante entregar el mismo con dichos criterios sin cumplir.

- **Los datos del tiempo se registran en todas las fases del proceso:** dicha observación se implementó del lado del Frontend, chequeando que todas las fases agregadas en el proyecto, tengan el tiempo de inicio y de fin completado.

En el componente `ProjectDetailsPage` se define un estado de React llamado `checkboxlist`, que es utilizado para validar los distintos criterios de entrega del proyecto, entre ellos, la completitud de la fecha de inicio y fin.

El tipo de `checkboxlist` es un *array* de objetos JavaScript con la siguiente estructura:

```
1   [{
2     key: string,
3     message: string,
4     valid: boolean,
5   },
6   ...
7   ],
8
```

Cada objeto representa un criterio a cumplir antes de entregar el proyecto. En particular, el criterio “**Los datos del tiempo se registran en todas las fases del proceso**” tiene `all_phase_have_time` como *key* en el arreglo `checkboxlist`.

El valor del estado `checkboxlist` es modificado dentro de un `useEffect` (*hook* de React), que tiene como dependencia aquellos campos del proyecto involucrados en la `checkboxlist`, por ejemplo la fecha de inicio y fin del proyecto.

Cada vez que uno de estos campos cambia, su valor es evaluado y el estado `checkboxlist` se modifica.

Para el caso del criterio en cuestión, se utiliza el siguiente objeto en el arreglo `checkboxlist`:

```
1   {
2     key: 'all_phase_have_time',
3     message: 'All the phases have start and end time',
4     valid: version_data.phases.reduce((acc, x) => acc
5       && !!x.start_time && !!x.end_time, true),
6   },
7
```

Para definir la validez del criterio se utiliza el método `reduce` de Javascript. El mismo se aplica sobre un arreglo, en este caso las fases del proyecto en curso.

`Reduce` recibe dos parámetros: el **callback o función** que se ejecuta en cada elemento del `array` y acumula un valor resultante, y el **valor inicial** (opcional) del acumulador.

La función, a su vez, acepta cuatro argumentos:

- **acumulador**: El valor acumulado por el `reduce` hasta el momento (resultado parcial).
- **elementoActual**: El elemento actual que se está procesando en el arreglo.
- **índice** (opcional): El índice del elemento actual en el arreglo (opcional).
- **array** (opcional): El arreglo original en el que se está aplicando el método.

Si se proporciona el valor inicial del acumulador, el `callback` se ejecutará desde el primer elemento del `array`. Si no se proporciona, se tomará el primer elemento del `array` como valor inicial, y el `callback` se ejecutará a partir el segundo elemento.

La función `callback` se ejecuta en cada elemento del `array`, actualizando continuamente el valor del acumulador, utilizando el elemento actual, según la lógica definida en el `callback`. Al finalizar el recorrido del `array`, el método `reduce()` devuelve el valor acumulado final.

De esta manera, se define si todas las fases presentan fecha de inicio y final.

El estado `checkboxlist` es utilizado en la propiedad `disabled` del botón de *Submit to Professor*. Si se cumple que al menos un booleano `valid` de los objetos de `checkboxlist` es falso, entonces se deshabilita el botón.

A su vez, al hacer *hover* en el botón, se imprime una lista explicitando al estudiante qué criterio no está cumpliendo.

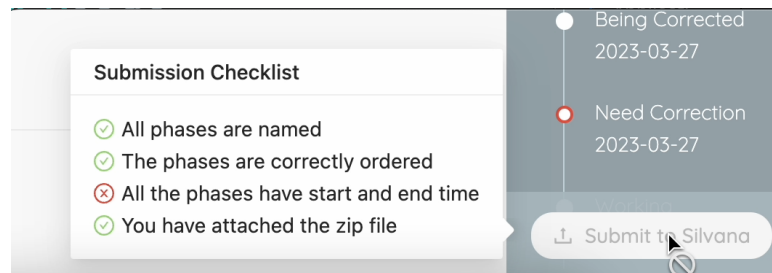


Figura 5.11: En la imagen se ve la lista de criterios a cumplir como estudiante para poder enviar el proyecto.

El resto de los criterios de la lista serán descritos a continuación.

- Las etapas del proceso siguen una secuencia adecuada:** este criterio también es parte de la `checklist` implementada en el componente `ProjectDetailsPage`. En esta oportunidad, se verifica que las etapas obligatorias para el proyecto existan, y a su vez, que cumplan con el orden esperado. Estas reglas se definen de la siguiente manera:

1. La fase de Plan debe ser la primera fase del proyecto.
2. Siempre, una fase de Diseño debe seguir a la de Plan.
3. La fase de Diseño siempre debe ser seguida por una fase de Codificación.
4. La fase de Compile (si existe) siempre va a venir luego de una de Codificación.
5. Siempre debe haber una fase de Test justo antes de Post Mortem.
6. Post Mortem debe ser la última fase.

Al igual que el criterio **“Los datos del tiempo se registran en todas las fases del proceso”**, la validez del orden de las etapas se registra en el estado `checklist`. Esta vez la lógica utilizada para poblar el objeto es la siguiente:

```

1  {
2    key: 'right_order',
3    message: 'The phases are correctly ordered',
4    valid: !version_data.phases.some(({ psp_phase }, index, phases)
5      => (
6        !psp_phase
7        || (!index ? !psp_phase.first : psp_phase.first)
8        || (!phases[index + 1] ? !psp_phase.last : psp_phase.last)
9        || (psp_phase.first && phases[index + 1]?.psp_phase?.name !== '
10       DESIGN') // no hay un design inicial
11       || (psp_phase.last && phases[index - 1]?.psp_phase?.name !== '
12       UNIT TEST') // no hay un test final
13       || (psp_phase.name === 'DESIGN' && ![ 'CODE', 'DESIGN' ].includes
14         (phases[index + 1]?.psp_phase?.name))
15       || (psp_phase.name === 'COMPILE' && phases[index - 1]?.
16         psp_phase?.name !== 'CODE')
17     )),
18  },

```


Se verifica mediante el método `some`, si algún elemento del *array* de fases no cumple con las características detalladas en el *callback*. Estas características corresponden a la lista previamente presentada en lenguaje natural, y se combinan con el operador lógico `or`, ya que basta con que una no se cumpla para dar como inválida la fase, y por ende el proyecto.

Si el atributo `valid` del objeto con *key* `right_order` es falso, un icono de alerta aparece al lado izquierdo del título de la pestaña de fases, y al hacer *hover* con el cursor, el estudiante puede ver las reglas a cumplir para que las fases estén bien ordenadas:

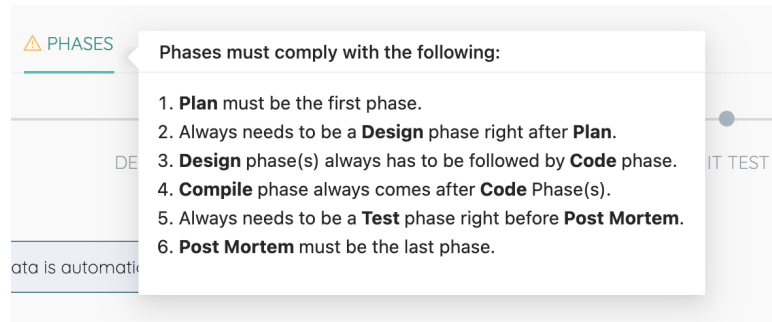


Figura 5.12: Se ve en la imagen el texto que explicita al usuario el orden correcto de las fases en caso de error.

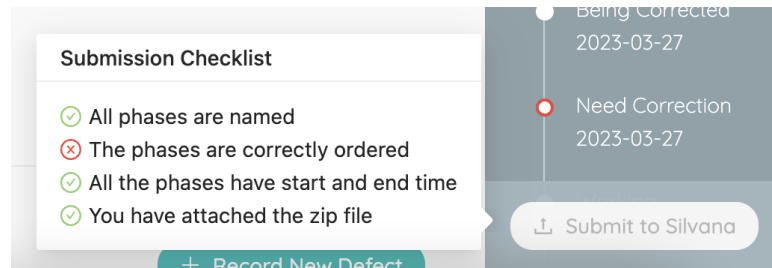


Figura 5.13: Al igual que en el criterio anterior, el botón de *Submit* es deshabilitado y la razón es aclarada al pasar el cursor por el mismo.

Observaciones de Log de defectos

En este caso se trata con dos criterios de la *grading* a la vez, ya que el segundo está incluido en el primero: **Todos los defectos tienen la información de registro solicitada** y **Todos los defectos tienen tiempo de corrección**.

Estos criterios son implementados del lado del Frontend, ya que desde el análisis de requerimientos, Silvana solicitó que el estudiante no pueda registrar un defecto si hay datos faltantes.

La interfaz de usuario deshabilita el botón en tal caso. Los campos requeridos son:

- **Discovered Time:** fecha en la que se descubrió el defecto.

- **Phase Injected:** fase en la que se inyectó el defecto.
- **Defect Type:** tipo de defecto, incluido en los posibles tipos dados (Documentation, Syntax, Build, Package, Assignment, Interface, Checking, Data, Function, System, Environment).
- **Fixed Time:** fecha en la que el defecto es corregido.
- **Description:** una explicación detallando de qué trata el defecto reportado.
- **Fix Defect:** este campo es condicional, solo aparece cuando en al menos una instancia de la fase de inyección, existe algún defecto registrado. El estudiante debe completar este campo si el defecto que está ingresando se provocó por corregir otro defecto durante la fase de inyección. A su vez, si dicha fase es Compile o Unit Testing y las mismas tienen defectos registrados, el campo es obligatorio.

A continuación se muestra el código correspondiente al chequeo de los campos, extraído del componente `DefectForm.js`:

```

1   const isFixDefectReq = ['5', '4'].includes(defectState?.
2   phase_injected?.id)
3   && fixDefectsList.length; // siendo 4 y 5 los IDs de las fases
4   Compile y Testing
5
6   const isSaveDisabled = (!defectState
7   || !defectState?.discovered_time
8   || !defectState?.phase_injected
9   || !defectState?.defect_type
10  || !defectState?.fixed_time
11  || !defectState?.description
12  || (isFixDefectReq && !defectState?.fix_defect)
13  || String(defectState?.description).trim() === ''
14  );

```

Como se explicó previamente, el operador `or` es utilizado para combinar todas las condiciones, de manera que si alguna de ellas se cumple, `isSaveDisabled` tomará el valor verdadero. El valor de `isSaveDisabled` es pasado al botón de *Submit to Professor* mediante la *prop disabled*.

`fixDefectsList` corresponde a la lista de posibles defectos que indujeron el defecto actual. Y como se mencionó anteriormente, el campo es obligatorio cuando se trata de un defecto con fase de inyección Unit Testing o Compile, lo que conduce a otro criterio de la *grading*: **Los defectos inyectados en Compilación y Test tienen “fix defect” asociado.**

Esta automatización se implementó forzando al estudiante a elegir un `fix_defect` en los casos donde el defecto que se está registrando haya sido inyectado en Compile o Unit Testing, y además que esta fase tenga algún defecto registrado.

Se requiere esta información al estudiante ya que se espera que cualquier defecto introducido durante la etapa de compilación o *testing* haya surgido como resultado de

corregir otros defectos previamente existentes. Esto se debe a que, por definición, las tareas de prueba y compilación no deberían introducir nuevos errores en el código.

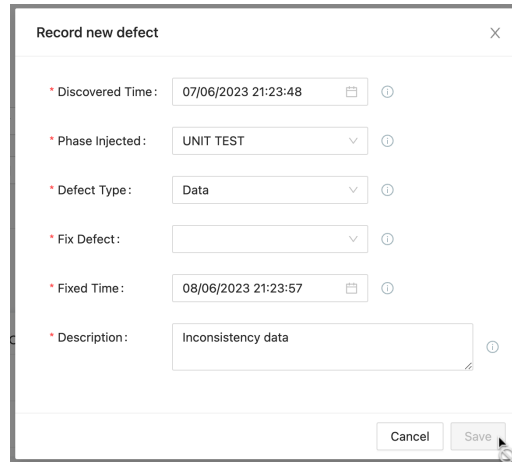


Figura 5.14: El estudiante está registrando un nuevo defecto, que tiene Unit Testing como fase de inyección, y a su vez, dicha fase tiene defectos reportados en ella. Entonces el sistema marca el campo Fix Defect (*dropdown*) como obligatorio, y no permite guardar el nuevo defecto, sin dejar constancia de cuál estaba arreglando (Fix) al introducirlo.

Observaciones Formulario PIP

El formulario PIP de PSPcode consta de 3 campos ya explicados en la Sección 2.2.2. Esta categoría de la *grading* posee dos criterios para corregir a los estudiantes: **El formulario PIP está completo** (se evalúa del lado del estudiante, previo a entregar el proyecto) y **Los registros del PIP muestran perspicacia y reflexión** (no se automatiza por ser subjetivo a quién corrige).

El chequeo para “**El formulario PIP esta completo**” es implementado del lado del Frontend, y el requerimiento inicial fue exigir al estudiante que cada uno de los campos tenga más de 15 caracteres escritos. Luego, en tiempo de pruebas y uso de la aplicación, dicho requerimiento cambió y el campo `other_notes`, que sigue siendo obligatorio, no exige un mínimo de 15 caracteres.

Para este criterio, si el estudiante intenta entregar el proyecto con información faltante en dichos campos, se despliega un modal de advertencia:

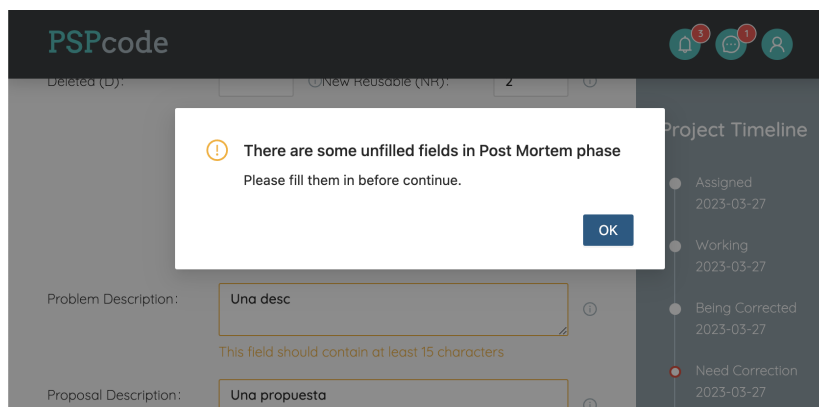


Figura 5.15: En este caso el usuario con rol Estudiante no completó alguno de los tres campos del PIP antes de intentar enviar el proyecto.

Una vez que el usuario presiona el botón *OK* del modal, el sistema lo redirige automáticamente a la fase de Post Mortem, independientemente de su ubicación actual en el proyecto. Es decir, si el usuario no se encuentra en la etapa Post Mortem al intentar entregar el proyecto, o incluso si está visualizando una pestaña diferente de Phases (como ser Summary, Files o Messages), será dirigido directamente a la fase de Post Mortem. Esta fue una decisión de UX tomada con el fin de brindar una indicación clara de la ubicación del error, facilitando el proceso de identificación y resolución del problema.

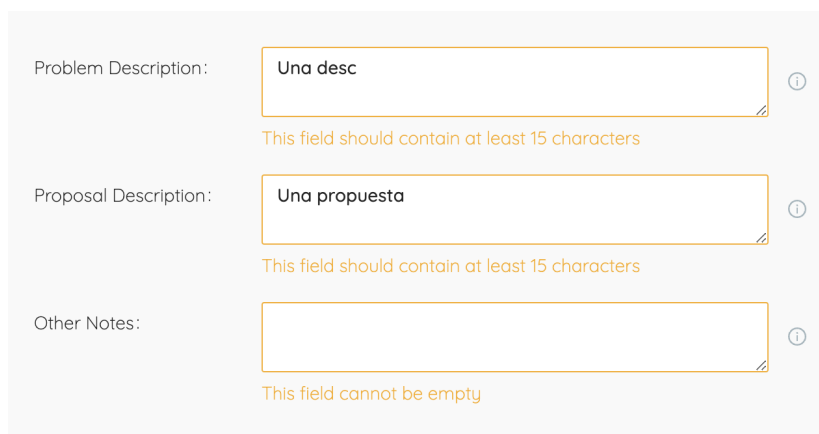


Figura 5.16: Referencias claras a los errores cometidos por el usuario: no llena los campos de PIP.

Cabe destacar que, para una mejor experiencia de usuario, el mensaje de error y recuadro amarillo en los campos, no se ve hasta el momento de entregar. Luego de advertido, cuando el estudiante corrige y cumple los requerimientos de cada campo, la advertencia en amarillo se borra, y puede entregar correctamente.

El código para determinar el error en los campos es el siguiente:

```

1  const isFieldInvalid = (fieldName) => (
2    hasSubmitted && (
3      !activePhase[fieldName]?.length

```

```

4     || (fieldName !== 'pip_notes' && activePhase[fieldName]?.length <
      15)
5   )
6 );

```

5.2.2.3 Observaciones Planning Summary

De esta sección, se tomó acciones del lado del estudiante para el criterio “**Los tamaños planificado y actual están ingresados correctamente**”. Este criterio de la *grading* refiere a que el campo **Plan LOCs (A+M)** de la fase **Plan** debe estar completo. Así como en la fase **Post Mortem**, los campos que miden líneas de código: **New Reusable** y **Total** deben estar completos también.

Sin embargo, Silvana manifestó como requerimiento que solo los campos de **Post Mortem** mencionados sean obligatorios para el estudiante al momento de entregar el proyecto. El campo de **Plan LOCs (A+M)** es chequeado del lado del profesor al corregir el proyecto, detallado en la Sección 5.2.2.1.

Al igual que el Formulario PIP, los campos **New Reusable** y **Total** (cantidad de líneas de código), son exigidos mediante el modal ya presentado, y las advertencias en color amarillo que aparecen al momento de entregar.

Figura 5.17: Luego de que el usuario intenta entregar con al menos uno de estos campos sin rellenar, el sistema lo dirige a la fase de **Post Mortem**, y despliega los mensajes de error presentados en la figura.

5.3. Project Feedback

Más adelante, en pos del mismo objetivo: el uso de PSPcode como una herramienta integral para abordar el curso, se implementó otra funcionalidad completamente nueva a la herramienta, que resultó ser muy práctica para su uso. La misma nace de la necesidad, previamente expuesta, de cerrar el ciclo de cada proyecto internamente en la aplicación.

Esta característica en particular, consiste en la inclusión de una pestaña de corrección en la vista del proyecto, diseñada específicamente para que los profesores puedan dar su devolución y los estudiantes recibirla. La interfaz de usuario de dicha pestaña de corrección se implementó replicando la planilla (*grading*) utilizada por los docentes, que hasta el momento, se editaba externamente, y subía al sistema como un documento de texto.

Los usuarios con rol **Docente** pueden proporcionar comentarios detallados sobre cada criterio predefinido, clasificados en las secciones ya presentadas (**Logs de Tiempo**, **Logs**

de Defectos, etc.), abordando cualquier aspecto del proyecto.

A su vez, se implementaron algoritmos de observaciones para algunos de los criterios, como ya se detalló en la Sección 5.1.2. En caso de presentarse una anomalía en el proyecto entregado por el estudiante, dichas observaciones son incluidas a modo de advertencia en la *grading* a rellenar por el profesor. Cada criterio puede ser aprobado o reprobado de manera individual por el docente. Si todos los criterios de una sección son aprobados, entonces la sección es marcada como aprobada. Finalmente, el profesor aprueba o desaprueba el proyecto en sí mismo.

Una vez que el profesor ha completado la revisión, y entrega las correcciones, se le envía un correo electrónico y mensaje (dentro de PSPcode) a los estudiantes, y ellos pasan a tener acceso (de lectura) a la pestaña de corrección en su proyecto, previamente oculta. Esto les permite ver de manera clara y directa qué aspectos de su trabajo requieren mejoras y cuáles son los elementos que han sido correctamente abordados, sin tener que descargar un archivo ni salir de la aplicación.

Los estudiantes pueden revisar y analizar cada comentario proporcionado por el profesor dentro de PSPcode, lo que les brinda una experiencia de navegación más simple, como también una comprensión más precisa de las áreas que necesitan atención y de las fortalezas que ya han demostrado en el proyecto.

Se definió un documento de contrato entre Frontend y Backend para desarrollar la *feature* de manera consistente. En el mismo se especificó la estructura de los datos que se enviarán de un *end* al otro en pos de guardar y obtener la corrección del proyecto. Dicha estructura será especificada en las siguientes secciones del informe, donde se explicará detalladamente el trabajo realizado en Frontend y Backend.

Tanto las tareas de Backend como de Frontend fueron definidas y documentadas bajo la épica “Correction tab” en el proyecto de Jira (descrito en la Sección 3.2.2.2)

5.3.1. Frontend

La interfaz de usuario se diseñó e implementó de manera estética e intuitiva, tal que el usuario tenga guías de uso, colores de referencia, y animaciones. Se creó casi en su totalidad utilizando AntD, la librería que ya estaba en uso en el Frontend de la aplicación, y fue actualizada como se describió en la Sección 4.2.1.3.

Cada idea del diseño se discutió en reuniones y mensajes con Silvana, y a su vez, éste no se despegaba demasiado de la estética general de la aplicación, y la estructura de la *grading* ya utilizada por ella. Por ende, el riesgo de error en el diseño de la interfaz estaba bastante acotado.

Antes de integrar la *feature* con el Backend, se realizaron pruebas de aceptación enviándole videos *demo* a la tutora con datos *hardcodeados*, y explicando el funcionamiento de los mismos, para validar el diseño.

De dicha instancia surgió sólo un pedido de cambio mínimo de diseño: quitarle el efecto de negrita a los comentarios de cada ítem de la *grading*. Más allá de eso, la profesora se mostró muy conforme con el resultado.

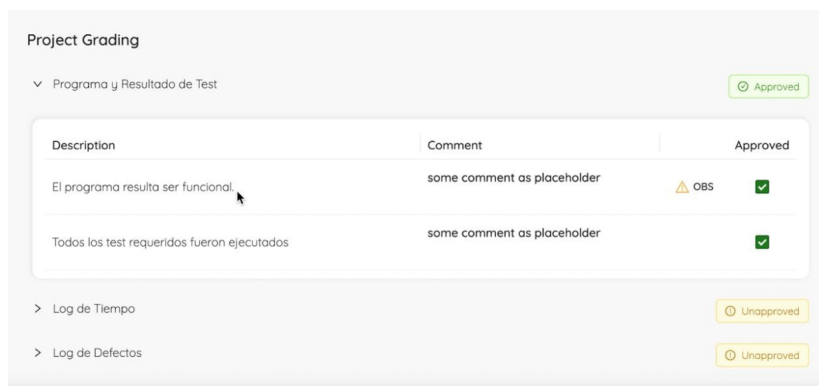


Figura 5.18: En la figura puede verse el texto en negrita (correspondiente al comentario del profesor sobre un criterio), lo cual se solicitó que cambiara a texto normal.

5.3.1.1 Presentación de Interfaz de usuario

En esta sección se presenta detalladamente cada componente y funcionalidad de la interfaz de usuario de la pestaña de **Correction**.

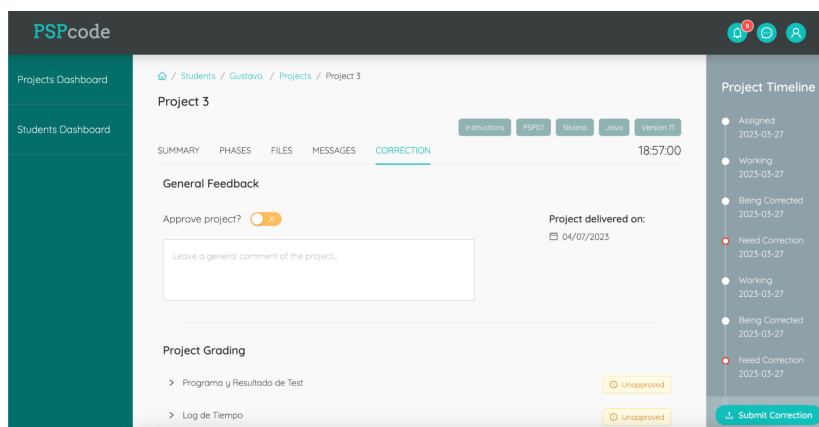


Figura 5.19: Primera aproximación general a la pestaña de correcciones. Vista de un usuario con rol Profesor.

En esta primera captura de la funcionalidad se puede ver la pestaña **Correction** seleccionada, y por debajo el contenido de la misma. En la figura pueden verse los valores por defecto asignados a una corrección, cuando un profesor entra por primera vez a la misma.

Se puede apreciar que: la aprobación general del proyecto no está activada, el comentario general del proyecto es vacío, mostrando el *placeholder* “Leave a general comment of the project”, y todas las secciones de la *grading* listadas debajo, también están en estado de desaprobación. A su vez, todas las secciones se encuentran visualmente colapsadas por defecto, más adelante se detalla la vista de la sección desplegada.

Por otra parte, la pantalla también muestra la fecha en la que el estudiante entregó el proyecto para someterse a la corrección.

En la esquina inferior derecha, se ve el botón *Submit Correction*, mediante el cual el profesor puede enviar la corrección (acción que hace visible dicha pestaña para el estudiante).

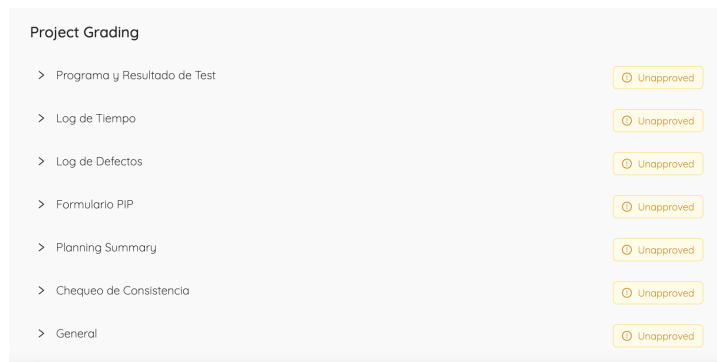


Figura 5.20: Captura de la visualización por defecto de las distintas secciones de la *grading*.

Poniendo foco en las distintas secciones a corregir, como se dijo previamente por defecto, todas se encuentran “Unapproved”. Este estado cambia, una vez que el docente marca todas las casillas de aprobación de los criterios pertenecientes a la sección.



Figura 5.21: Vista desplegada de la sección Planning Summary.

Esta sección posee dos criterios a corregir. Se puede apreciar en cada uno:

- La **descripción** del criterio, que fue tomada directo de la *grading* original y los docentes pueden editar desde la aplicación del Admin, como se detalla en la Sección 5.4.
- Un **comentario** que el docente puede dejar opcionalmente para el estudiante.
- La **casilla de chequeo de aprobación**, donde el docente marca (o no) si el criterio cumple con las expectativas.
- Una columna donde un botón **OBS** se hace visible, sólo si:
 1. el criterio está ligado a una de las automatizaciones descritas en la Sección 5.1.2
 2. si dicha automatización devuelve que el criterio no está satisfecho.

Detallando más profundamente en el botón *OBS*: cuando es presionado, el mismo despliega una nueva fila con color de fondo amarillo claro indicando advertencia, como se puede ver en la Figura 5.21.

En dicha fila, se lista la/s fases que están incumpliendo el criterio, respetando el orden de la *timeline* del proyecto. Los nombres de las fases son *clickables*, llevando al detalle de la fase donde se encuentra el error, así el profesor puede llegar de manera fácil a la raíz del problema, para dejar un comentario acorde si desaprueba el criterio. Además, el estudiante puede ver fácilmente el origen de ese comentario y/o desaprobación a la hora de revisar la corrección de su proyecto.

En la siguiente imagen vemos tres criterios de la sección “**Log de Tiempo**”, el primero y el tercero poseen observaciones automatizadas, mientras el segundo y el último no presentan ninguna observación.



Figura 5.22: Vista desplegada de la sección Log de Tiempo.

Se puede apreciar que al desplegar la fila de observaciones del criterio “**Los tiempos son registrados en los pasos correctos del proceso**”, se detalla en qué fases el criterio no se está cumpliendo.

Este criterio evalúa si la fecha de inicio y de fin de cada fase respeta el orden de las mismas en la *timeline*, sin superposiciones. En este caso existen seis fases que tienen este problema.

Particularmente el nombre de las fases Design y Code, se repite, ya que el estudiante decidió iterar sobre estas etapas. Igualmente, como las fases de la observación son presentadas en el mismo orden de la *timeline* del proyecto, es sencillo identificar cuál es cual. A su vez, al presionar cada una, el estudiante o docente accederá a la fase correcta, ya que cada una de ellas está identificada por id y no por nombre.

Si todos los criterios de una sección son aprobados, la etiqueta principal de la sección pasa de “Unapproved” a “Approved”, a su vez, en la captura se puede ver también el *tooltip* que aparece cuando el cursor pasa por dicha etiqueta.

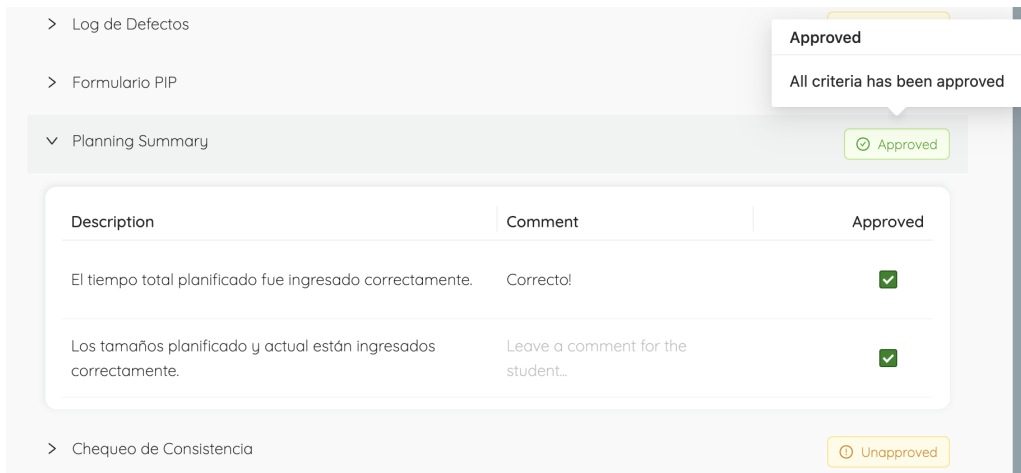


Figura 5.23: Vista de la sección Planning Summary con todos los criterios aprobados por el docente.

La profesora puede agregar un comentario general en el campo de texto principal, y aprobar el proyecto presionando en el *switcher* como se ve en la siguiente captura:

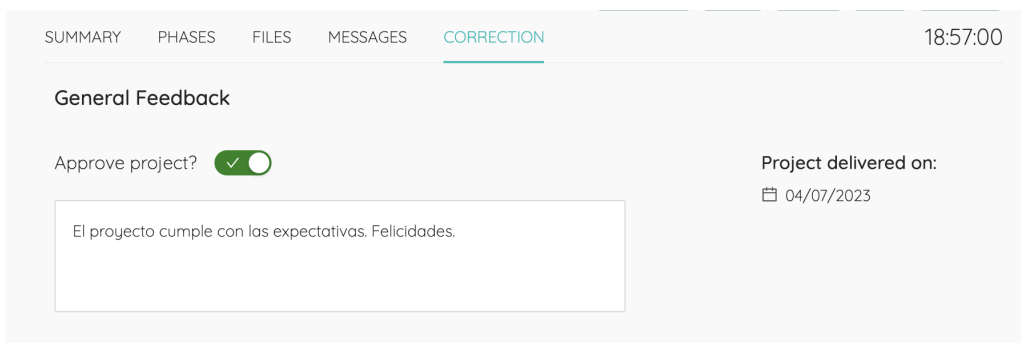


Figura 5.24: Parte superior de la pestaña Correction donde se ve el comentario y la aprobación general.

Si al presionar el *switcher* alguno de los criterios no está individualmente aprobado, la siguiente notificación aparece en la parte superior de la pantalla, en forma de advertencia:

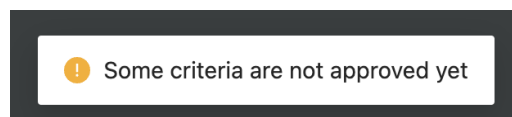


Figura 5.25

Cada cambio realizado en esta pantalla es automáticamente guardado en base, esperando un mínimo de un segundo entre cada *request*, es decir, el POST al Backend no será realizado más de una vez por segundo. Los detalles de la *request* serán descritos más adelante.

A su vez, la siguiente notificación aparece en pantalla cuando un cambio es guardado, con una frecuencia máxima de una vez cada dos segundos:

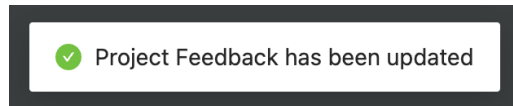


Figura 5.26

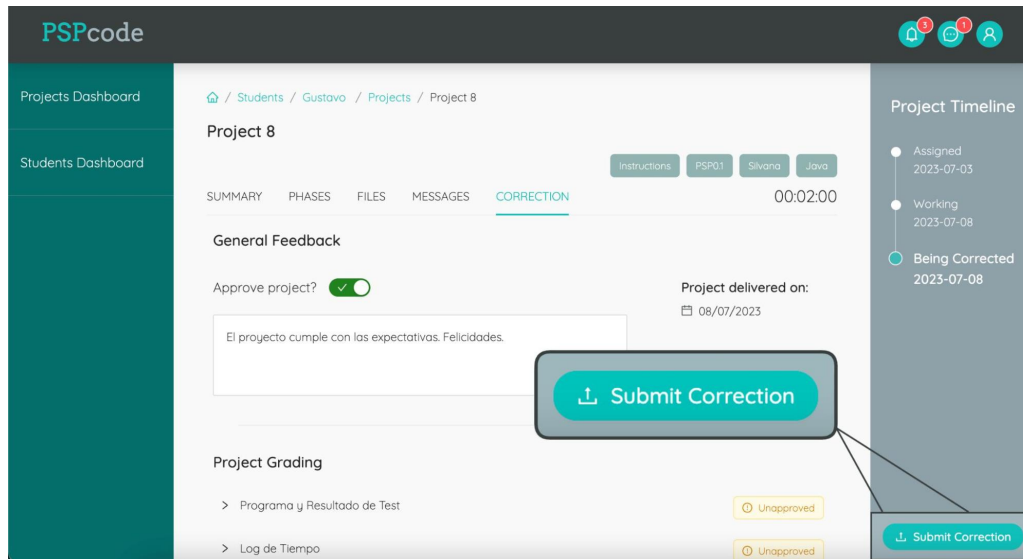


Figura 5.27: Vista en detalle del boton Submit Correction.

Luego, cuando el profesor presiona el botón *Submit Feedback*, un modal de confirmación es presentado al usuario. A su vez, en el caso de que se cumpla alguna circunstancia considerada incoherente, una serie de advertencias es presentada al usuario en dicho modal:

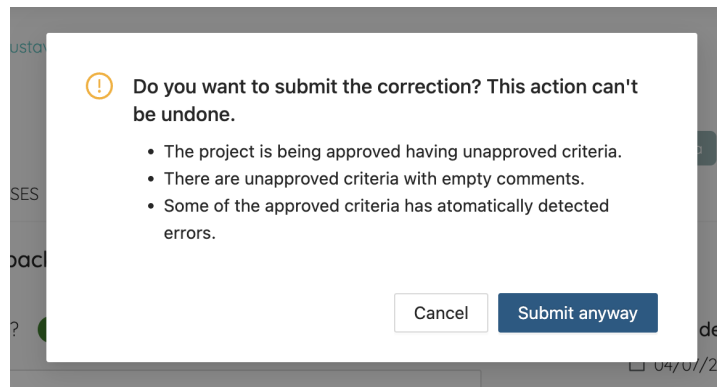


Figura 5.28

Circunstancias advertidas:

- **The project is being approved having unapproved criteria:** esta advertencia es similar a la notificación previamente presentada, se da cuando el profesor quiere enviar el proyecto como aprobado pero alguno de los criterios individuales no lo está.

- **There are unapproved criteria with empty comments:** aparece cuando el profesor deja criterios sin aprobar, pero no agrega comentarios para explicar al estudiante la razón de la reprobación.
- **Some of the approved criteria has automatically detected errors:** esta advertencia se da cuando alguno de los criterios ligados a algoritmos automatizados, presentan OBS (observaciones), pero sin embargo el profesor decide aprobarlos de todas formas.

Las tres circunstancias presentadas se pueden ver en el modal de ejemplo, ya que se tomó la captura en una corrección que cumple estas características desfavorables. Por otro lado, existen otras dos posibles advertencias:

- **All the criteria is approved but the project is not:** esta es prácticamente opuesta a la primera presentada, se muestra cuando todos los criterios individuales están aprobados pero el profesor (posiblemente por error) deja el proyecto en sí mismo sin aprobar.
- **The general comment is empty:** sencillamente, esta advertencia se presenta cuando el docente quiere enviar una corrección con el comentario general vacío, ya que se considera una mala práctica para el estudiante.

Cabe destacar que estas advertencias fueron ideadas por quien suscribe. Dada la forma en la que se implementó la *feature* de Project Feedback, para una mejor experiencia de usuario, se decidió advertir sobre estos comportamientos, para minimizar el error del docente, ya que la acción de enviar un *feedback* y generar una nueva versión del proyecto, no puede ser deshecha.

Al presionar *Submit Anyway*, la información del mismo es guardada, por si acaso el segundo de espera no pasó aún y el último cambio no se guardó automáticamente. A su vez, se agrega la `reviewed_date` (fecha actual) a la información del proyecto, lo que indica al Backend que debe actualizar el estado del proyecto desde `being_corrected` a `approved` o `need_correction`, y por ende la pestaña de corrección pasa a ser visible para el estudiante. El atributo `reviewed_date` tiene como semántica la fecha de revisión del proyecto, es decir, la fecha en la que el profesor envía su corrección.

```

1  static project_feedback_update(userid, projectid, versionid,
2  newProjectfeedback) ${
3  return api.put(`/users/${userid}/assigned_projects/${projectid}/
   project_deliveries/${versionid}/project_feedback`,
   newProjectfeedback);
   }$

```

Como se puede ver, en la request de *update* del Project Feedback se envía al Backend la siguiente información: id del usuario, id del proyecto, id de la versión corregida del proyecto, y el objeto JavaScript con toda la información del Project Feedback generada y editada desde el Frontend, sumándole la fecha de revisión.

Con respecto al objeto JavaScript para representar Project Feedback, el mismo se definió con la siguiente estructura:

```

1  project_feedback: {
2    approved: boolean,
3    comment: string, // general comment
4    delivered_date: string,
5    reviewed_date?: string,
6    grouped_corrections: [{
7      section_name: string,
8      corrections: [{
9        id: number,
10       approved: boolean,
11       comment: string,
12       obs_phases: [{index: number, name: string}], // fases que no
           cumplen el algoritmo ligado a este criterio. El index corresponde al
           lugar de la fase en la timeline.
13     }],
14   }],
15 }},

```

5.3.2. Backend

La implementación de la característica de la pestaña de correcciones involucró la creación de nuevas nuevas entidades en la base de datos para manejar las correcciones de los proyectos. La entidad existente `ProjectDelivery` se modificó para agregar una relación con la nueva entidad `ProjectFeedback`.

Las nuevas entidades están relacionadas de la siguiente manera: `ProjectFeedback` pertenece a `ProjectDelivery`, `Correction` pertenece a `Criterion` y a `ProjectFeedback`, y `Criterion` pertenece a `Section`. Estas relaciones permiten asociar los comentarios y correcciones con las entregas de los proyectos, los criterios evaluados y las secciones correspondientes. En la sección [.2.2](#) se muestra el modelo de dominio relacionado a estas características.

Detalles de entidades y relaciones:

- **ProjectFeedback:** representa los comentarios y correcciones proporcionados por los profesores para una entrega de proyecto específica. Tiene los siguientes atributos: `comment` (comentario proporcionado por el profesor), `approved` (indica si el proyecto fue aprobado o no), `delivered_date` (fecha de entrega del proyecto) y `reviewed_date` (fecha en que se revisó el proyecto). Esta entidad está relacionada con `ProjectDelivery`, lo que significa que un `ProjectFeedback` pertenece a un `ProjectDelivery`. Además, un `ProjectFeedback` puede tener múltiples correcciones, por lo que tiene una relación de **uno a muchos** con la entidad `Correction`.
- **Correction:** representa una corrección específica realizada por el profesor. Tiene los atributos `approved` (indica si la corrección fue aprobada o no) y `comment` (comentario proporcionado por el profesor sobre la corrección). Esta entidad está relacionada con `Criterion` (criterio) y `ProjectFeedback`. Una `Correction` pertenece a un `Criterion` y a un `ProjectFeedback`.
- **Criterion:** representa un criterio o aspecto específico que se evalúa en un proyecto. Tiene los atributos `description` (descripción del criterio), `only_in_psp01` (indica si

el criterio solo aplica en la fase PSP01), **order** (orden del criterio), **algorithm** (algoritmo o algoritmos relacionados con el método utilizado para calcular la observación relacionada con este criterio) y **algorithm_type** (tipo de algoritmo relacionado con el criterio, se utiliza para identificar entre los algoritmos de fase y los de defectos). Esta entidad está relacionada con **Section** (sección), lo que significa que un **Criterion** pertenece a una **Section**.

- **Section:** representa una sección del proyecto. Tiene el atributo **name** (nombre de la sección). Esta entidad tiene una relación de **uno a muchos** con **Criterion**, lo que significa que una **Section** puede tener múltiples **Criterion** (criterios).

Además de las entidades mencionadas, se agregó una nueva ruta de acceso en la API para obtener el **ProjectFeedback** de una versión específica del proyecto (**ProjectDelivery**).

La tabla de la entidad **Correction** experimenta un crecimiento considerable, acumulando un registro por cada combinación de **ProjectFeedback** y **Criterion**. Con el objetivo de abordar potenciales desafíos de eficiencia, se consideró implementar el patrón de diseño Eager Loading. Este patrón implica realizar una consulta única para recopilar todos los datos requeridos y almacenarlos en memoria. Sin embargo, en este caso podría llevar a un uso excesivo de memoria, dadas las dimensiones extensas de la tabla de **corrections**.

En respuesta a esta problemática, se optó por almacenar en memoria solo el **ProjectFeedback** específico que se necesita, seguido de una consulta adicional para recuperar las **Corrections** relacionadas. Esta estrategia evita la necesidad de almacenar los resultados de **Corrections** en memoria, y dado que la tabla de **Correction** está correctamente indexada y la consulta se realiza exclusivamente para un **ProjectFeedback** específico, la gran cantidad de registros en la tabla resulta en un problema menor.

También se realizaron modificaciones en las notificaciones para que, al aprobar o denegar un proyecto, se genere una notificación que se envía al estudiante a través del correo electrónico y como un mensaje dentro de la aplicación. Además, se realizaron cambios para que se habilite automáticamente la pestaña de correcciones en el panel superior de la aplicación, lo que permite al estudiante acceder fácilmente a ella. A través de esta pestaña, el estudiante puede ver las correcciones realizadas por el profesor y los comentarios asociados. Esta mejora asegura que los estudiantes sean notificados de inmediato sobre el estado de su proyecto y puedan consultar rápidamente las correcciones y sugerencias proporcionadas por el profesor.

El JSON de respuesta del **ProjectFeedback**, que se envía hacia el Frontend contiene los siguientes atributos:

- **id:** es el ID del proyecto.
- **approved:** indica si el proyecto ha sido aprobado o no.
- **comment:** es un comentario relacionado con el proyecto.
- **delivered_date:** es la fecha en que se entregó el proyecto.
- **reviewed_date:** es la fecha en que se revisó el proyecto.
- **grouped_corrections:** se explica a continuación.

El método `group_corrections` se encarga de la implementación del atributo `grouped_corrections`. Este método agrupa las correcciones del proyecto por sección y luego crea objetos `CorrectionsGroupDto` que contienen la información de cada grupo de correcciones. Cada objeto `CorrectionsGroupDto` tiene un nombre de sección, una lista de correcciones y una lista de instancias de fases.

La agrupación se realiza utilizando el nombre de la sección de cada corrección. Luego, se mapea cada grupo de correcciones para crear objetos `CorrectionDto` que contienen información detallada de cada corrección, como su ID, descripción, comentario, estado de aprobación y observaciones de fases.

El método `group_corrections` utiliza una serie de métodos auxiliares para obtener las observaciones de fases correspondientes a cada corrección, basándose en los algoritmos asociados a cada una de ellas (los mismos utilizados para el cálculo de observaciones explicado en la Sección 5.2.1) y las instancias de fases del proyecto.

Para obtener las observaciones que se dan en cada criterio de una corrección, se utiliza el método `get_obs_phases`. Este recibe dos parámetros: `Correction` (corrección) y `phase_instances` (instancias de fases). A continuación se muestra un simple pseudocódigo del método en cuestión.

```
1 Función get_obs_phases(correction, phase_instances):
2   phase_obs es []
3   algorithm es correction.criterion.algorithm
4
5   Si algorithm no está presente:
6     Retornar phase_obs
7
8   Si phase_instances está presente:
9     algorithms es algorithm.split('_and_') /* Dividir el algoritmo si hay varios
10      juntos */
11
12   Para cada a en algorithms:
13     algorithm_method_name es "build_{a}_obs" /* Nombre del método del algoritmo */
14
15     Para cada phase_instance, index en phase_instances:
16       result es null /* Resultado del algoritmo */
17
18       Si correction.criterion.algorithm_type == 'phase':
19         result es ejecución del algoritmo de esta fase
20
21       Sino si phase_instance.defects está presente:
22         Para cada defect en phase_instance.defects:
23           result es ejecución del algoritmo de este defecto
24
25           Si result está presente:
26             Detener el bucle
27
28       Si result está presente y la fase está presente y el nombre está presente:
29         phase_obs.agregar({ index: index, name: phase_instance.phase.name })
30
31   Retornar phase_obs
```

5.4. Cambios en la plataforma de Admin

Un objetivo del presente proyecto era dar un mayor control de la aplicación a los usuarios con rol Docente, pudiendo agregar y editar fases, cursos y procesos del sistema. Mediante la modificación del archivo `rails_admin.rb` se agregaron nuevas entidades configurables a la plataforma del Admin. Las entidades agregadas al menú fueron: Courses, Phases and Processes, permitiendo a los docentes agregar, quitar o modificar instancias de dichas entidades.

De esta forma los docentes del curso podrían definir un nuevo proceso, por ejemplo PSP2.0, configurando las características deseadas, y asociando las fases que considere necesarias, como podrían ser Code o Design Review, asimilándose al PSP2.0 original.

Se adjunta en el Anexo .5.2 la pantalla de edición de fase a modo de ejemplo.

A su vez, la implementación del nuevo flujo de correcciones implicó la inclusión de nuevas entidades a la aplicación y por consiguiente al Admin. Se agregan al menú de dicha plataforma Corrections, Criteria, Project Feedbacks y Sections para poder visualizar y editar el contenido de las correcciones, los *feedbacks* de los proyectos, los criterios y las secciones. En el Anexo .5.1.1 se ejemplifica el caso de uso de edición de un criterio.

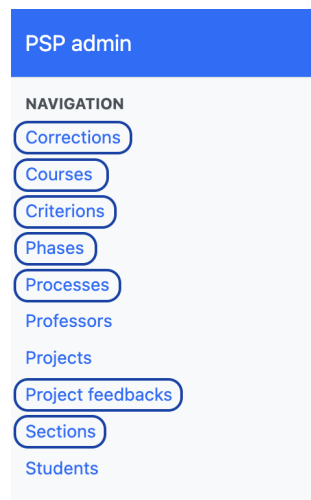


Figura 5.29: Captura de la plataforma Admin, donde se encuentran encuadradas aquellas entidades sumadas al menú en la nueva versión de PSPcode.

Capítulo 6

Pruebas y Despliegues

Para simplificar el despliegue y la administración de la aplicación, se optó por utilizar Docker una plataforma de contenedores que garantiza un entorno consistente y reproducible. Se configuró un archivo **docker-compose** (se adjuntan los archivos correspondientes a producción en el Anexo .8) para definir los contenedores necesarios y sus dependencias.

Entre las cosas que se especifican en el archivo **docker-compose**, se encuentra la ruta de la imagen que debe levantar cada servicio. En uno de los archivos se encuentra la ruta de la imagen del Backend y la del Frontend. Para alojar estas imágenes, se crearon dos *Amazon Elastic Container Registry* que consisten en repositorios públicos de AWS. Uno llamado **pdp-code-backend** para las imágenes del Backend y otro llamado **psp-code-frontend** para las imágenes del Frontend.

6.1. Entorno de Pruebas

Previo al lanzamiento de las nuevas funcionalidades y modificaciones desarrolladas en el proyecto, resultó esencial someterlas a una evaluación metódica. En este sentido, se implementó un entorno de pruebas denominado entorno de “*staging*” o “*testing*” en un servidor personal. Este entorno sirvió como un espacio seguro para validar la aplicación antes de su implementación en el entorno de producción real.

El objetivo principal de este ambiente fue permitir a los desarrolladores y docentes interactuar con el Frontend de la aplicación, probando todas las funcionalidades antes de liberarlas a un entorno de producción accesible para el curso y los estudiantes.

La identificación temprana de errores en este entorno no solo facilitó correcciones sino que también permitió mejoras adicionales basadas en el *feedback* recopilado durante las pruebas, resultando en una aplicación más alineada con los requisitos de las docentes.

6.2. Entorno Productivo

En primer lugar, se adquirió el dominio **pspcode.com** a través de CloudFlare, que también proporcionó servicios DNS para la conexión con PSPcode. Además del servicio DNS, con CloudFlare se crearon certificados de seguridad para poder tener conexiones

seguras a partir del protocolo HTTPS.

Se utilizaron dos servicios de *Cloud Computing* para alojar la aplicación: AWS inicialmente y luego Digital Ocean. La lógica de ambas plataformas fue similar, creando dos instancias que ejecutan el sistema operativo *Linux*; una para alojar una nueva base de datos de PostgreSQL y un *Proxy*, y otra para el Backend y el Frontend.

La base de datos actualmente está alojada de manera independiente, sin estar vinculada a un plan específico de base de datos como servicio, a diferencia de lo que ocurría anteriormente con Heroku. El espacio disponible para esta base de datos está directamente ligado al espacio en disco del Droplet. Al finalizar un semestre del curso, se observó que se habían utilizado aproximadamente 15MB en la base de datos. Considerando que aún hay 3GB disponibles en el Droplet, esto sugiere que, manteniendo un uso constante similar, la capacidad de la base de datos podría ser suficiente para aproximadamente 200 años.

En Cloudflare, se configuraron los registros DNS necesarios para acceder a la IP pública de la instancia que contiene el Proxy. Este es Nginx Proxy Manager, el cual se encarga de redirigir a la segunda instancia, la cual aloja el Backend y el Frontend, además de agregar conexiones seguras de HTTPS. Se detalla a continuación la tabla de registros DNS y la tabla de mapeo de direcciones del Proxy Manager:

Registros DNS del servidor de Cloudflare		
Nombre	Tipo	Contenido
pspcode.com	A	146.190.62.95
admin.pspcode.com	CNAME	pspcode.com
svc.pspcode.com	CNAME	pspcode.com
www.pspcode.com	CNAME	pspcode.com

El registro de tipo A (*address*) se utiliza para asociar una dirección a un nombre de dominio, en este caso se mapea el dominio pspcode.com a la dirección IP 146.190.62.95 que corresponde a la dirección IP pública del Droplet 1 de Digital Ocean (análogo a una instancia) en donde se aloja el Proxy Manager. Luego los otros registros son de tipo CNAME, que simplemente asocian las direcciones posibles de acceso a la aplicación (alias), con el nombre de dominio canónico o principal (el correspondiente al registro tipo A), es decir, todas las direcciones registradas están asociadas a la IP pública del Droplet 1.

La dirección IP 10.124.0.2 corresponde a la IP privada del Droplet 2, que contiene desplegado en el puerto 3000 el Backend y en el puerto 8080 el Frontend. Las peticiones que llegan al Proxy (Droplet 1) son mapeadas según la dirección hacia la IP del Droplet 2 con su correspondiente puerto.

Se presenta a continuación un diagrama de flujo que ilustra el proceso de una solicitud de un usuario a la página del Frontend de la aplicación, en donde se describe la infraestructura actual con Digital Ocean, asumiendo que es la primera vez que se ejecuta y no se conoce la dirección solicitada. El servidor DNS de CloudFlare responde con la dirección IP de **pspcode.com**, es decir del Droplet 1. Luego, la petición llega al Proxy,

Mapeo de direcciones Nginx		
Proxy Host	Destino	Detalle
admin.pspcode.com	http://10.124.0.2:3000	Backend
svc.pspcode.com	http://10.124.0.2:3000	Backend
www.pspcode.com	http://10.124.0.2:8080	Frontend

que redirige a partir del mapeo descrito anteriormente, a la dirección del Droplet 2 y el puerto 8080. Esta responde finalmente a la solicitud del usuario, mostrando la página de *login* de la aplicación. Un flujo análogo ocurre con las solicitudes al Backend.

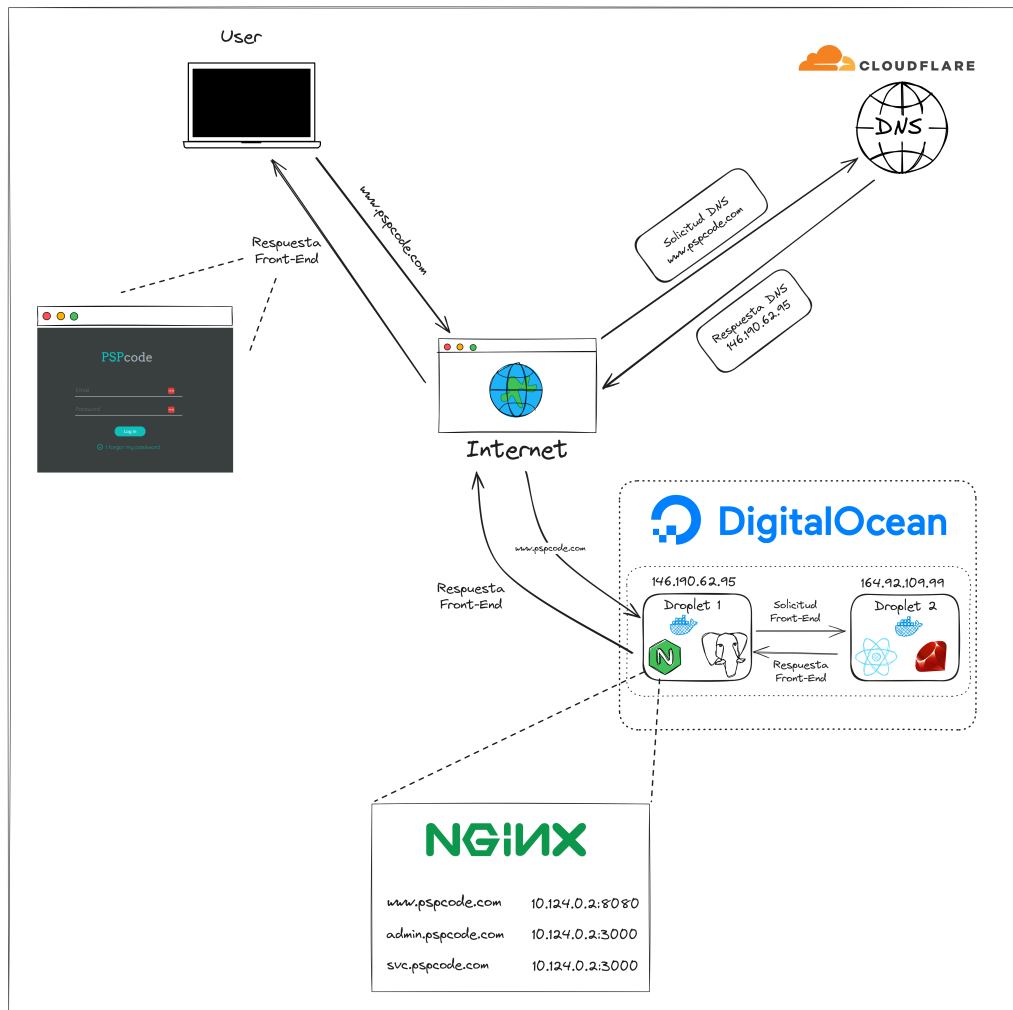


Figura 6.1: Ejemplo petición al Frontend de PSP.

6.2.1. AWS

En la implementación inicial, se desplegaron dos instancias *EC2* del tipo *t2.micro*, la cual cuenta con los siguientes recursos: 1 GiB de memoria RAM, procesador Intel Xeon Scalable de hasta 3,3 GHz y almacenamiento de 8 GiB. Además, se creó un Bucket S3 para gestionar los archivos de la aplicación.

Durante el curso de marzo de 2023, se enfrentaron desafíos como instancias que se caían debido al consumo máximo de la CPU. Para mejorar la detección de estos problemas, se configuraron alertas que notificaban cuando la CPU superaba un umbral definido, permitiendo intervenciones antes de que la aplicación experimentara fallos. También se establecieron eventos automáticos para reiniciar las instancias y, por ende, los contenedores de Docker en respuesta a estas alertas.

Las instancias tienen un costo de U\$S12 por mes, pero existe un plan gratuito de 12 meses (una sola instancia), por lo que el primer año se gastarían sólo U\$S12 al mes por la instancia restante. Posteriormente, el costo se duplicaría. En cuanto al Bucket S3, cuenta con un plan gratuito de 5GB, que no se ha agotado, y luego cobraría U\$S0.023 por GB por mes para los primeros 50TB. Es decir, el Bucket terminaría siendo despreciable y el gasto total estaría en las dos instancias.

6.2.2. Digital Ocean

Al concluir el curso, debido a consideraciones de costos, se optó por migrar a Digital Ocean. La decisión de postergar la migración hasta la finalización del curso se tomó para evitar interferencias en el funcionamiento. Se configuraron dos Droplets con las siguientes especificaciones: 512MB de RAM y 10GB de almacenamiento; sin embargo, no se dispone de información específica sobre el tipo de procesador utilizado. Al notar que la memoria del Droplet 2 resultaba insuficiente, se determinó incrementar su capacidad a 1GB, conllevando un aumento en los costos asociados.

El Droplet 2 que aloja el Backend y Frontend tiene un costo de U\$S6, y el Droplet 1 un costo de U\$S4 por mes. Además se agregó un Backup cada media hora de la instancia que contiene la base de datos, ante cualquier problema que pueda generarse. Este tiene un costo de U\$S0.8 al mes. En total se tendría un gasto aproximado de U\$S10,8 por mes, significativamente menor que el estimado con AWS (alrededor de U\$S30).

Al igual que en AWS, se cargaron alertas que notifican por correo, para saber si el *admin* (Backend) no responde por un tiempo total de 5 minutos, o si la CPU de cualquiera de los dos Droplets está por encima del 70 % de uso por un tiempo máximo de 5 minutos.

A diferencia de AWS, Digital Ocean no permite establecer eventos automáticos ante alertas, por lo que se implementó un *job* de Linux dentro de la instancia que se encarga ejecutar un *script* cada 15 minutos el cual realiza un *ping* hacia la IP de la aplicación con un máximo de 5 reintentos. Si no hay respuesta, se reinicia el contenedor de Docker, asegurando la continuidad de la aplicación sin necesidad de reiniciar toda la instancia. Para abrir la configuración de *jobs*, se ejecuta el comando `crontab -e`, en donde se encuentra en este caso, el horario y el *script* que se quiere ejecutar. El *job* tiene la siguiente configuración:

```
0,15,30,45 * * * * cd / && ./check_and_restart.sh
```

Esto significa que se ejecuta el *script* `check_and_restart.sh` (adjunto en Anexo .9) a toda hora, todos los días, en los minutos 0, 15, 30 y 45. Es decir, cada 15 minutos.

Además, se enfrentaron desafíos adicionales, como instancias que se caían durante la madrugada debido a un tráfico externo desconocido, posiblemente peticiones intencionadas por algún *bot*. Para abordar esto, se creó un Firewall para cada Droplet (accediendo a <https://cloud.digitalocean.com/networking/firewalls/>) de Digital Ocean, en donde se crearon reglas específicas para permitir sólo conexiones deseadas. La configuración y explicación de cada uno de los Firewall creados en los Droplets se encuentra en el Anexo .10.

6.3. Defectos reportados en etapa de prueba

Como se describió previamente en este capítulo, hubo dos etapas de uso de la aplicación: una se dio en un ambiente de *testing*, alojada en un servidor personal, la otra fue en el transcurso del curso 2023, donde la herramienta fue utilizada integralmente, reemplazando PSPcode original.

En ambiente de prueba, quienes testearon la aplicación fueron las docentes del curso: Silvana Moreno (también tutora de este proyecto), y Leticia Pérez. El objetivo de esta instancia era encontrar y arreglar la mayor cantidad de *bugs*, para prevenir los mismos al momento de salir a producción. Esta fase duró desde diciembre hasta principios de marzo.

La metodología que se usó en esta instancia fue un documento compartido de *Google Sheets* donde las profesoras fueron detallando problemas encontrados en la aplicación, seguidos (en la mayoría de los casos) por una clasificación de prioridad: Media o Alta.

A continuación se presentan estos errores o peticiones de cambio. Cabe destacar que la siguiente lista fue redactada por los autores del informe para mejor entendimiento del lector, la tabla con el texto sin alterar escrito por las docentes para reportar los hallazgos se encuentra en Anexo .6.

- Como docente no es visible la entrega (zip adjunto) del estudiante.
- En la tabla de Proyectos del sistema, el texto que indica de quién es el proyecto, por ejemplo: Projects “ofStudent3” se ve todo sin separar.
- El nombre de la fase Post Mortem estaba mal escrito en el sistema: la palabra “Mortem” finalizaba con una “n” en vez de “m”.
- No exigir un mínimo de 15 caracteres en el campo *Other Notes* de los campos PIP de la fase Post Mortem. Sí solicitar al menos un carácter.
- Al momento de mandar el Ejercicio 2, se debería solicitar al estudiante el número de líneas de código del Ejercicio 1.
- En PSP0, es decir en el Ejercicio 1, durante la etapa de Plan, se debe tener el campo *Plan Time* para completar. Actualmente ese campo existe a partir del Ejercicio 2 (con PSP0.1).
- Los ejercicios no deberían comenzar si el ejercicio anterior no fue corregido y aprobado. El estudiante no debería tener la posibilidad de empezar un ejercicio hasta que el docente no se lo asigne

- Desde un usuario con rol Docente, al visualizar la lista de estudiantes, se espera un filtrado por defecto de aquellos estudiantes asignados al docente activo. En caso de querer ver todos los estudiantes, el usuario puede remover el filtro, o seleccionar otro docente.
- El servicio de *mailing* (Sendgrid 2.3.1.2) no está funcionando. Las docentes no están recibiendo los *e-mails* correctamente.

Se tuvieron instancias de intercambio con las docentes para terminar de comprender los distintos problemas y comportamientos esperados, y se realizaron *tickets* en el tablero de Jira para solucionar los mismos.

Todos los cambios solicitados y errores encontrados en esta etapa pudieron ser abordados antes del final de Marzo 2023, para cumplir con la fecha de liberación esperada y lograr utilizar la nueva herramienta PSPcode en el curso 2023.

Varios de ellos fueron ocasionados por errores en la semilla (*seed*) de datos utilizada para este entorno de prueba, y no con el desarrollo del código en sí mismo. En la etapa de Análisis de requisitos no se tuvo en cuenta y no se le otorgó la debida importancia a la consistencia y semántica de los datos utilizados para probar la aplicación, por ejemplo:

- Que cada Proyecto de asociara con su respectiva versión de PSP. Sucedió que al crear la *seed*, se generaron un grupo de proyectos de prueba, llamados *Proyecto <nro del 1 al 8>*, y se le asignó a cada uno una versión del PSP aleatoria entre la 0 y la 0.1.
- A los estudiantes se les asignó desde la base de datos más de un proyecto a la vez, en lugar de asignar únicamente el Proyecto 1 y luego dejar que los usuarios con rol Docente hagan las siguientes asignaciones conforme aprobaban los proyectos.
- Había parámetros booleanos como ser `has_plan_time` mal asignados (como `false` en este caso) al definir las instancias de versiones de PSP.
- Algunos problemas de tipeo, como pudo ser la escritura de “Morten” en lugar de “Mortem”.

Por otro lado, hubo cambios menores que hacer desde el Frontend, como ser la falta de espacio en el texto “*ofStudent3*” o la solicitud de cambio en el filtro por defecto de la lista de estudiantes.

Luego, el error del archivo *.zip* fue solucionado realizando una reparación desde el Backend, ya que lo que sucedía era que la ruta donde se almacenaba el archivo no estaba siendo definida correctamente.

El problema con el servicio de *mailing* fue abordado en la Sección ??, este comportamiento se daba ya que la cuenta que la aplicación origen estaba usando había sido suspendida.

6.4. Defectos reportados en etapa de producción

El 17 de marzo de 2023, luego de que los problemas encontrados en etapa de prueba fueran solucionados, se procedió a liberar la aplicación al ambiente productivo descrito en la Sección 6.2.1.

Durante el transcurso del curso PF-PSP (Marzo - Julio 2023) se dieron numerosas instancias de reporte y solución de errores durante el uso productivo de la aplicación. Todos los errores reportados, fueron atendidos y solucionados en menos de 24 horas. A su vez, ninguno de estos errores impidió a los docentes o estudiantes el cumplimiento de las tareas del curso.

El equipo de desarrollo recibió todos estos reportes en tiempo real vía Whatsapp y mantuvo en una tabla de Google Sheets [.7](#), donde se documentó para cada error:

- La fecha en la que fue reportado.
- Quién hizo el reporte: Leticia, Silvana, o un estudiante.
- Descripción detallada del problema encontrado.
- Tipo de problema, pudiendo ser: confusión en los requerimientos, *bug* en la UI, errores en los datos o en la configuración del *deploy*, o errores de lógica (algoritmos).
- Área o aplicación en la que se debe solucionar el problema: Frontend, Backend, o Fullstack (ambas)
- Descripción de el/los arreglos necesarios para solucionar el problema.
- Fecha en la que el problema fue solucionado completamente. Con respecto a este campo, existen casos donde se destrabó a los usuarios para que no tuvieran inconvenientes en el uso de la aplicación, pero la solución final y permanente del problema se dio horas o días más tarde.

6.4.1. Descripción de errores y soluciones

A continuación se describen los inconvenientes y las soluciones encontradas:

6.4.1.1 B01 - Los cuadros de texto no admiten espacios

La aplicación PSPcode no permite colocar espacios en los cuadros de texto de “Comentarios” dentro del formulario de cada fase de un proyecto.

Este *bug* fue reportado el 20/03/2023 por un estudiante, Silvana hizo efectivo el reporte hacia el equipo mediante Whatsapp.

Este defecto se da debido a un error de interfaz de usuario, puramente desde el lado del Frontend. El arreglo se hizo el mismo día.

Sucedía que la herramienta, para guardar la información del proyecto en el Backend, comparaba la última versión guardada con la actual, si había habido cambios, ejecutaba la request. En particular, para comparar la información del campo “Comentarios”, se le hacía un `trim()` (quitaba los espacios) al *string*, para evitar hacer una llamada al Backend si lo único que habían cambiado eran espacios en ese campo.

Este algoritmo se daba así desde la aplicación origen (heredada), el problema surgió al hacer el refactor. Se hicieron cambios en el código, y el `trim()` ya no solo afectaba la comparación para llamar al Backend, sino también, evitaba que se permitieran espacios en el *string* original mostrado al usuario.

Esto se corrigió eliminando el `trim()` para comparar los *strings*, ya que no se deberían tomar como iguales en ningún caso.

6.4.1.2 B02 - Los campos PIP están siendo requeridos para PSP0

En el Proyecto 1, que se rige bajo PSP0, aparece en la aplicación una advertencia sobre falta de información en los campos de PIP a la hora de corregir. Es decir, en la pestaña de Corrección de un proyecto, a los usuarios de tipo docente se les muestra una advertencia sugiriendo que el estudiante no rellenó los campos del PIP, siendo que para PSP0 estos campos no son solicitados.

Este error fue reportado también el 20/03/2023 por Silvana Moreno.

En este caso el error fue del área Backend, los algoritmos que verificaban la existencia de información para los campos PIP no se debían correr si el Proyecto era PSP0.

El problema surgía desde la semilla de datos, ya que los criterios de corrección (explicados en la Sección 5.1.1), pueden ser, o no, mostrados exclusivamente a partir del PSP0.1.

Este era el caso del criterio “El formulario PIP está completo”, sólo debía tenerse en cuenta para PSP0.1. Se corrigió este defecto, el mismo día que fue reportado, agregando el valor *true* en la base de datos, para la columna `only_in_psp01` de dicho criterio.

6.4.1.3 B03 - Advertencia errónea para el criterio “Los defectos fueron inyectados antes de ser removidos”

Una vez más, Silvana obtuvo una advertencia inesperada corrigiendo a un estudiante. La aplicación muestra en la pestaña de Correcciones que el estudiante removió un defecto en una etapa previa a ser inyectado (ver el error ejemplificado en 5.2.2.1).

El error fue reportado y solucionado el 23/03/2023. En este caso se requirieron modificaciones en ambas aplicaciones: Backend y Frontend.

Un usuario con rol estudiante, al reportar un defecto en una etapa del proyecto, debe seleccionar en qué otra etapa dicho defecto fue inyectado. Como ya se vio en la descripción del sistema, el estudiante puede haber iterado en las distintas etapas a lo largo de la *timeline*, y por ende, tener varias etapas con el mismo nombre en su proyecto.

El problema se daba ya que el *dropdown* para seleccionar la etapa de inyección mostraba todas las etapas del proyecto sin distinguir (por ejemplo) una etapa de nombre Compile, de otra etapa con el mismo nombre.

Sin embargo, internamente dichas fases tenían identidad diferente, siendo distintas instancias de fase en lo que a la base de datos respecta. Por lo tanto, el usuario seleccionaba una de ellas (probablemente al azar), y si el ID de dicha fase coincidía con una fase posterior en la *timeline* a la que el usuario se encontraba editando, el error se dispararía: un defecto no puede removerse en una fase previa a la de inyección.

El *fix* consistió en mostrar al usuario una sola ocurrencia de cada nombre de fase en dicho selector. Siempre y cuando la fase exista en la línea de tiempo del proyecto que se

está editando. Luego, en etapa de corrección, el sistema únicamente advierte al profesor si no se encuentra una fase anterior a la actual con el nombre seleccionado por el usuario en el proyecto.

6.4.1.4 B04 - La aplicación no permite a los estudiantes entregar el proyecto

Este error fue reportado el 24/03, y fue el problema más crítico que se dio en etapa de producción. Si bien se solucionó desde la base de datos al otro día de ser reportado, y no afectó los tiempos del curso, encontrar la fuente del problema llevó cuatro días.

Este error se dio en el Proyecto 1 para algunos estudiantes, es importante destacar este dato, ya que éste es un error más relacionado la interacción del estudiante con la aplicación que a un caso borde o situación aleatoria.

La situación se daba cuando el estudiante presionaba “Submit Project” el sistema desplegaba un error (manejado) y el proyecto no se podría enviar al profesor. Para solucionar el problema temporalmente, se realizó un cambio de estado de los proyectos corruptos desde la base de datos, así se podían destrabar dichos ejercicios y ser corregidos.

El inconveniente principal con este error era la dificultad para reproducirlo, se intentó de muchas maneras y no se lograba llegar al error que los estudiantes veían en pantalla. Luego de numerosas revisiones de código, se llegó a la conclusión de que el problema se daba ya que en la acción del botón *Submit Project*, el Frontend realizaba dos *requests* al Backend. Por alguna razón, la primera estaba siendo completada correctamente, y la segunda no, lo que llevaba a las inconsistencias encontradas en la base de datos.

Se descubrió que la razón por la que las llamadas al Backend se veían interrumpidas era que el botón no se deshabilitaba mientras la solicitud se estaba procesando, por lo tanto, los usuarios que realizaban doble *click* generaban una doble *request* que rompía la aplicación. A partir de ahí, los datos de ese proyecto quedaban corruptos, y ya no se podía entregar.

Lo que se hizo fue, además de corregir los proyectos ya afectados, deshabilitar el botón mientras la solicitud se encontraba en proceso. A su vez, se revisó el resto de botones de la aplicación para prevenir un error similar en otros módulos.

6.4.1.5 B05 - El archivo .zip entregado por el estudiante no es visible en reentregas

Este problema fue reportado por Silvana Cuando un proyecto no es aprobado y el estudiante comienza una nueva versión, se genera un duplicado de la última disponible, así el usuario puede trabajar partiendo de la base a corregir.

Sucedía que, todos los datos se copiaban correctamente, excepto el archivo *.zip* entregado por el usuario. Si el estudiante reemplazaba el archivo, todo funcionaba correctamente, pero había casos donde la razón de la desaprobación no estaba relacionada con dicho archivo, por lo tanto el estudiante no lo modificaba. En esos casos, no se podía acceder al archivo *.zip*.

Se corrigió el código de Backend para realizar un correcto duplicado del archivo existente en la versión anterior del proyecto. A su vez, se corrigieron los archivos ya corruptos en el Bucket S3, reemplazándolos por los correctos. Todo fue hecho el día en el que el bug fue reportado.

6.4.1.6 B06 - La pestaña “Corrections” se oculta cuando el estudiante comienza una nueva versión

Este es un error claro de confusión en los requerimientos. Fue reportado por Silvana también el 25/03.

Se espera que la pestaña de correcciones siempre sea visible luego de que el profesor envía la devolución. Sin embargo, el sistema sólo mostraba la pestaña de correcciones cuando el proyecto estaba en estado `correction_pending`, `need_correction`, o `approved`, es decir, cuando el proyecto era entregado por el usuario y aún estaba sin corregir, o cuando la última versión iniciada ya había sido corregida. Sin embargo, no era tenido en cuenta el caso donde el usuario luego de pasar por dichos estados comenzaba una nueva versión del proyecto, estado: `working`.

El comportamiento se solucionó, ese mismo día, cambiando la condición de renderizado de la pestaña de correcciones:

- Para el usuario con rol Profesor, Siempre que ya exista un `ProjectFeedback` asociado, la pestaña se muestra con la información del último *feedback* (haya sido entregado al estudiante o aún no), a su vez, la información del formulario es editable si solo si el proyecto está en etapa de corrección (estado `correction_pending`).
- Para el usuario con rol Estudiante, la pestaña es visible siempre y cuando exista una corrección (`ProjectFeedback`) asociada a ese proyecto y ya enviada por el profesor. En la pestaña siempre se mostrará la información de la última versión corregida del proyecto.

6.4.1.7 B07 - El servidor de AWS se calló por primera vez

El 06/04 Silvana reportó que la aplicación se encontraba inaccesible.

Se revisaron las métricas de AWS y efectivamente la aplicación estaba caída. Se reinició para solucionar el problema, y se tomaron como medidas preventivas la configuración de alertas vía e-mail y el reiniciado automático de la aplicación ante situaciones como esta.

No se volvió a recibir un reporte de la interrupción del servicio por parte de los usuarios.

6.4.1.8 B08 - La aplicación despliega un error al intentar adjuntar un .zip

Este error fue reportado por un estudiante el 20/04. Cuando un estudiante intentaba subir un archivo .zip como entrega adjunta de un proyecto, el sistema daba error y no permitía la acción.

En este caso, afortunadamente no hubo problemas a la hora de reproducir el *bug*. La razón por la que ya no se podían subir archivos a la aplicación era que las maquinas EC2

de AWS [2.3.1.6](#) habían alcanzado su capacidad máxima.

Esto sucedió ya que en cada liberación de la aplicación se guarda una nueva imagen de la misma en la máquina. Por falta de experiencia con la herramienta, no se consideró borrar las imágenes previas (y ya en desuso) de la aplicación. Se llenaron las 8 gigas de material inoperante. Se eliminaron las imágenes y la aplicación volvió a funcionar correctamente.

La resolución de este problema fue llevada a cabo el mismo día en el que fue reportado.

6.4.1.9 B09 - Los nuevos mensajes con un estudiante en particular no se muestran

El 02/05 Leticia reportó que había un estudiante con el cual no estaba pudiendo comunicarse mediante el *chat* de la aplicación.

Ella manifestó que estaba recibiendo las notificaciones de los mensajes, pero al ingresar a la pestaña de mensajes solo veía conversaciones antiguas con dicho estudiante.

Este problema se dio ya que en los mensajes venían de forma paginada desde el Backend, y en la aplicación no se estaba usando dicha paginación correctamente: únicamente se estaba mostrando la primera página (la más antigua). En el código de la aplicación origen, la paginación tampoco estaba correctamente utilizada. Simplemente, se mostraba únicamente la última página en lugar de la primera, entonces los usuarios nunca notaron el problema.

A su vez, Esto no fue descubierto en etapa de prueba ya que nunca se superó el límite de mensajes por página.

Se solucionó el problema (al otro día de ser reportado) cargando todos los mensajes en una página. Las conversaciones se dividen por estudiante y por proyecto, por lo que no se espera que la cantidad de mensajes sea tal para que el tiempo de carga pueda molestar a los usuarios.

6.4.1.10 B10 - Los mensajes de un estudiante en particular se ven del lado derecho de la pantalla y en azul

El 19/05 Leticia reportó no poder ver los mensajes que le enviaba un estudiante en particular, pero sí veía los que ella le enviaba a él.

Al igual que el anterior, fue un problema difícil de reproducir, únicamente se pudo llegar a la fuente del mismo haciendo pruebas desde el usuario de Leticia.

Al inspeccionar la conversación de Leticia con este estudiante, le llegó a la conclusión de que los mensajes de el estudiante sí estaban siendo obtenidos y mostrados, pero del mismo lado y con el mismo color que los de Leticia, por lo que ella no estaba pudiendo identificarlos.

En la base de datos, los profesores y estudiantes están representados en dos tablas independientes, por lo que se puede dar que algunos de ellos compartan ID. Este era el

caso borde de Leticia y el estudiante en cuestión, ambos se identifican con el número 2.

El componente de mensajes evaluaba si un mensajes tenía como remitente al usuario loggeado comparando los IDs, lo cual representaba un problema en este caso. El *fix* se realizó el mismo día en el que el error fue reportado, y consistió en evaluar (además del ID) el rol del usuario, discriminando entre Estudiante y Profesor.

6.4.2. Distribución de errores por fecha

A continuación se presenta gráficamente la distribución de los errores según su fecha.

Distribución de Errores por Fecha de Detección

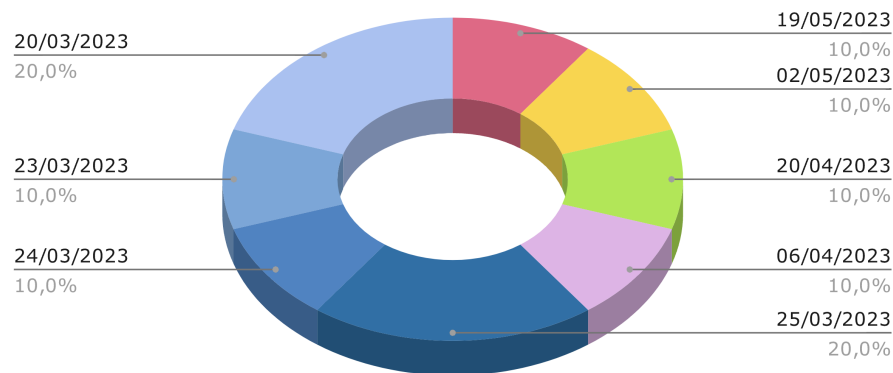


Figura 6.2

Puede verse en los días representados de color azul, cómo en Marzo se reportó un 60% de los errores, mientras el otro 40% se encuentra disperso entre los meses de Abril y Mayo. Esta distribución tiene sentido ya que el uso de la aplicación a lo largo de los proyectos es, en gran parte, reiterativo, y la mayoría de las características son probadas desde el Proyecto 1.

A su vez puede concluirse que la cantidad y criticidad de los errores encontrados en etapa de producción fue baja.

6.5. Pruebas de Aceptación de usuario

En el marco del proceso de validación y mejora de la aplicación web PSPcode, se realizaron pruebas de aceptación de usuario como parte esencial del mismo.

Un aspecto de estas pruebas de aceptación de usuario fue la participación de Silvana Moreno, que además de tutora de tesis, tuvo un rol de usuaria de la aplicación. Esta cercanía y comunicación constante permitió un intercambio de comentarios directos y reportes de errores, como se vio en las secciones anteriores, lo que enriqueció significativamente la evaluación de la aplicación. Sus comentarios y observaciones fueron valiosos para el proceso de mejora del sistema.

A lo largo del desarrollo, la tutora expresó en varias ocasiones su satisfacción con el rendimiento y la utilidad de la aplicación, destacando la efectiva resolución del problema inicial que impactaba a las docentes del curso: la incapacidad de corregir los proyectos dentro de la aplicación. Este logro contribuyó en gran medida a fortalecer la confianza en la calidad del producto.

Por otro lado, en el caso de los estudiantes, la interacción con el equipo de desarrollo se dio a través de las profesoras, sin mantener una comunicación directa. Para evaluar su percepción y experiencia con la aplicación, se realizó una encuesta a través de Google Forms. Esta estrategia permitió recopilar de manera sistemática y anónima la opinión de 16 de los 34 estudiantes que completaron el curso en 2023.

La encuesta se diseñó con el objetivo de recopilar información detallada sobre diversos aspectos de la aplicación, como la facilidad de uso, la satisfacción estética, el nivel de utilidad para el curso, y la identificación de posibles problemas o deficiencias. Fue enviada a los estudiantes el día 17 de Julio de 2023, una vez finalizado el curso.

Esta sección se centra en la presentación de los resultados principales de las encuestas realizadas a los usuarios con rol de estudiantes, quienes jugaron un papel fundamental en la evaluación de la aplicación.

6.5.1. Estructura de la encuesta

El formulario presentado a los estudiantes del curso consiste de diez preguntas. El objetivo fue recopilar una basta cantidad de información y opiniones, sin hacer la encuesta demasiado extensa.

Las preguntas fueron las siguientes:

1. “A niveles generales, ¿cómo fue tu experiencia utilizando la aplicación?”
2. “¿Cómo encontraste el tiempo de respuesta o carga de la aplicación?”
3. “Una vez explicado el modo de uso de la aplicación ¿Cuántas veces tuviste que consultar a la profesora cómo utilizar una funcionalidad?”
4. “En el caso de haber tenido dudas con respecto al uso de la aplicación, ¿recuerdas cuáles fueron, o con qué sección de la aplicación estaban relacionadas?”
5. “¿Experimentaste algún problema con la aplicación que momentáneamente te impidiera completar una tarea del curso?”
6. “En el caso de haber experimentado contratiempos de este tipo, ¿puedes describir cuáles fueron, y cómo lograste finalizar el flujo? Incluyendo el tiempo aproximado que demoraron en ser resueltos”
7. “Con respecto a la pestaña de Correcciones ¿Con qué afirmaciones te sientes identificado/a?”
8. “¿Qué podrías destacar positivamente de la aplicación?”
9. “¿Tu experiencia con la app. PSPcode, afecta (positiva o negativamente) la impresión que te llevas sobre el curso?”
10. Como última pregunta, se dio la oportunidad de aportar cualquier otra retroalimentación que no se haya preguntado específicamente: “¿Tienes algún otro comentario o feedback para la aplicación?”

6.5.2. Resumen de resultados

6.5.2.1 Pregunta 1

“A niveles generales, ¿cómo fue tu experiencia utilizando la aplicación?”

Esta pregunta tiene como objetivo evaluar las sensaciones globales que deja la aplicación en los usuarios. Las opciones de respuesta consiste en una escala numérica del 1 al 5, siendo 1: Mala o Incomoda, y 5: Excelente.

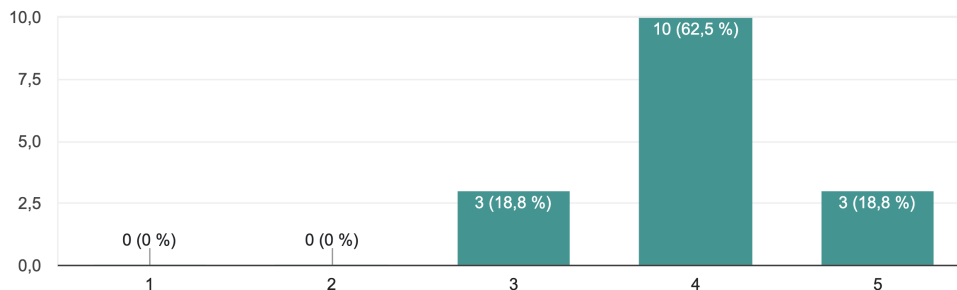


Figura 6.3: Se presentan gráficamente los resultados de la Pregunta 1

De un total de 16 estudiantes participantes en la encuesta, 3 expresaron que su experiencia fue de nivel “Excelente” (5 en la escala de evaluación). En cambio, 10 de los encuestados calificaron su experiencia con un nivel “Bueno” (4 en la escala), lo que denota una evaluación positiva en la amplia mayoría de los casos. Por otro lado, 3 estudiantes evaluaron su experiencia como “Promedio” (nivel 3 en la escala), lo que sugiere una calificación intermedia en el menor de los casos.

6.5.2.2 Pregunta 2

“¿Cómo encontraste el tiempo de respuesta o carga de la aplicación?”

Esta pregunta también tiene una respuesta de opción numérica, siendo 1: “Muy Lento” y 5: “Rápido (nunca sentí tener que esperar)”

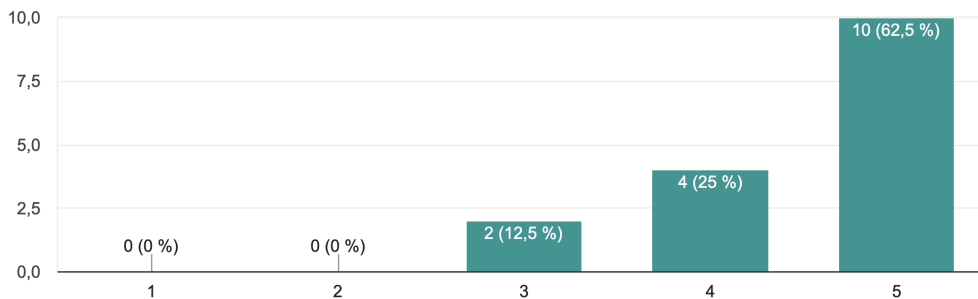


Figura 6.4: Se presentan gráficamente los resultados de la Pregunta 2

En cuanto a la pregunta que evaluaba la velocidad del servicio, se observan resultados prometedores. Específicamente, 2 encuestados calificaron la velocidad como nivel 3, lo que representa una percepción de “Normal”, y 4 personas la evaluaron como nivel 4, lo que indica una apreciación de “Rápido”. Además, 10 participantes otorgaron la máxima calificación de nivel 5, indicando que nunca tuvieron que esperar, lo que refleja una percepción extremadamente positiva de la rapidez del servicio. Estos resultados subrayan una tendencia general positiva en la percepción de la velocidad del servicio por parte de la mayoría de los encuestados.

6.5.2.3 Pregunta 3

“Una vez explicado el modo de uso de la aplicación ¿Cuántas veces tuviste que consultar a la profesora cómo utilizar una funcionalidad?”

esta es una pregunta de múltiple opción, y las posibles opciones de respuesta eran: Nunca, Una vez, Dos o tres veces, Cuatro o cinco veces, Más de cinco veces.

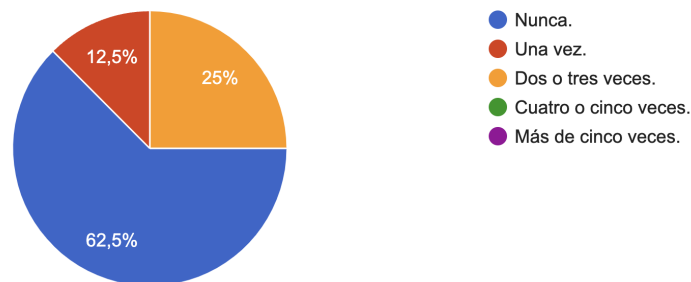


Figura 6.5: Se presentan gráficamente los resultados de la Pregunta 3

En relación a la pregunta sobre la frecuencia de consultas que necesitaron hacer, 4 encuestados indicaron que tuvieron que consultar entre “Dos o tres veces” (opción intermedia en la escala). Por otro lado, 2 personas mencionaron que solo necesitaron hacerlo “Una vez”, y la mayoría, es decir, 10 participantes, expresaron que nunca tuvieron que recurrir a la profesora para aclarar el uso de las funcionalidades de la aplicación. Estos resultados resaltan una tendencia positiva en la comprensión y manejo de la aplicación por parte de la mayoría de los encuestados, con solo un pequeño grupo que necesitó consultas adicionales en el proceso.

6.5.2.4 Pregunta 4

“En el caso de haber tenido dudas con respecto al uso de la aplicación, ¿recuerdas cuáles fueron, o con qué sección de la aplicación estaban relacionadas?”

Esta es una pregunta de respuesta abierta, su objetivo es recabar debilidades de la interfaz gráfica con respecto a la intuitividad.

Se obtuvieron 4 respuestas describiendo distintas confusiones atravesadas por los estudiantes. Dos de ellas no tienen ninguna relación con las áreas de la aplicación agregadas

o modificadas en este Proyecto de Grado. Las otras dos respuestas describen confusiones relacionadas a los mensajes de error del lado del estudiante, agregados en esta nueva versión de PSPcode.

Un estudiante manifiesta no saber que información colocar en uno de los cuadros de PIP (*Other Notes*), y el mismo es obligatorio para la entrega del Proyecto. Sin embargo, se considera fuera del alcance de la aplicación el explicar cuál debe ser el contenido en cada campo del formulario de PSP.

Otro estudiante manifiesta no haber tenido claro que la etapa Diseño del PSP es obligatoria para entregar el proyecto, y la aplicación no permite hacer la entrega sin ella.

Si bien, el sistema no tiene la obligación de brindar esta información, ya que es un tema dado en el curso, en la Sección 5.2.2.2, donde se explica cada uno de los controles pre-entrega implementados, se incluye la descripción del criterio “Las etapas del proceso siguen una secuencia adecuada”.

Cuando dicho criterio no es cumplido por el estudiante, un símbolo de advertencia es agregado en el título de la pestaña “Phases”, que despliega un *tooltip* de advertencia, que explicita al usuario cuales son las etapas obligatorias del proyecto, incluyendo la de Diseño.

Es importante reconocer que la elección de la metodología de advertencia puede haber sido inapropiada, ya que en el caso de este estudiante, no resultó en una comunicación efectiva del motivo por el cual no fue posible completar la entrega del proyecto.

6.5.2.5 Pregunta 5

“¿Experimentaste algún problema con la aplicación que momentáneamente te impidiera completar una tarea del curso?”

Similar a las dos preguntas anteriores, el objetivo con esta es entender con mas especificidad la experiencia del usuario en materia de problemas o impedimentos que hayan tenido.

Las opciones de respuesta para esta pregunta eran: “No, siempre pude completar los flujos”, “Una vez”, “Dos o tres veces”, “Más de tres veces”.

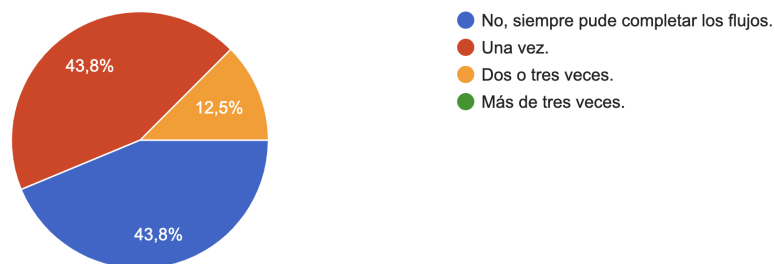


Figura 6.6: Se presentan gráficamente los resultados de la Pregunta 4

Se observa una distribución variada de las respuestas. Siete encuestados mencionaron que experimentaron problemas “Una vez”, dos personas informaron que esto ocurrió “Dos o tres veces” y siete personas afirmaron que nunca enfrentaron tal situación.

De estos resultados se puede observar que si bien tenemos una mayoría de estudiantes (nueve por sobre los dieciséis que respondieron la encuesta) que experimentaron al menos un problema con la aplicación, dichos *bugs* fueron todos resueltos en menos de 24 horas, como se presentó en la sección anterior.

A su vez, la salida a producción de la aplicación fue apresurada con respecto al nivel de *testing* que se hizo, por ende consideramos estos números, un buen resultado.

6.5.2.6 Pregunta 6

“En el caso de haber experimentado contratiempos de este tipo, ¿puedes describir cuáles fueron, y cómo lograste finalizar el flujo? Incluyendo el tiempo aproximado que demoraron en ser resueltos”

Esta, al igual que la Pregunta 4, es de respuesta abierta, para obtener más detalles de los inconvenientes que enfrentaron los estudiantes, y contrastarlos con aquellos que ya fueron resueltos.

Los nueve estudiantes respondieron esta pregunta. Se presentarán a continuación las respuestas más relevantes a efectos de generar un plan de acción.

La mayoría de los problemas, como se esperaba, se tratan de aquellos ya presentados en la sección anterior.

Por ejemplo:

- “En la primera tarea, en algunos campos no se podían poner espacios. terminé escribiendo todo junto (si mal no recuerdo). no me hizo perder mucho tiempo, después para la segunda tarea ya estaba solucionado.” error presentado en la Sección [6.4.1.1](#).
- “Al intentar enviar una tarea para su corrección, la herramienta se “tranco” y no dejaba enviarla dando un error. Demoro uno o dos días en arreglarse.” error presentado en la Sección [6.4.1.4](#)
- “La pagina no cargaba, a la media hora entre y ya andaba” error presentado en la Sección [6.4.1.7](#)

Luego, una respuesta interesante fue la siguiente: “Me paso que una vez estuve horas no entendiendo porque no me dejaba entregar y era porque había dejado una fase sin nombre y no me daba cuenta cuál era (era el primer puntito de todos, el resto en adelante estaba bien), Demore pila en darme cuenta, porque nada me avisaba que era esa la fase que estaba sin nombre.”

La inquietud de este usuario refiere al chequeo que se hace previo a que el estudiante entregue un proyecto. Esta validación es manifestada al usuario de la siguiente forma:

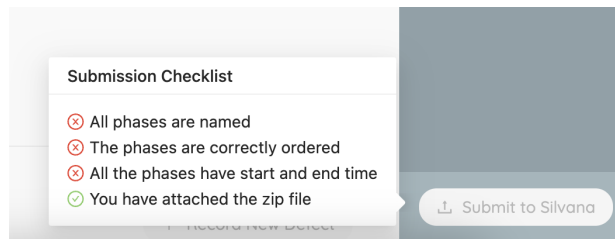


Figura 6.7: El botón *Submit Project* se encuentra deshabilitado, y una de las razones es que hay al menos una fase sin nombre.

Y según lo que este estudiante explica, la primera fase de su *timeline* era la que no tenía nombre, viéndose de esta manera:

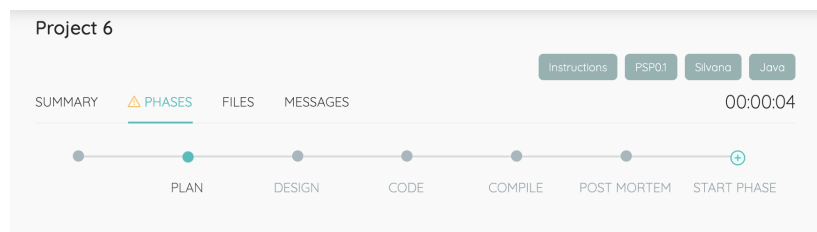


Figura 6.8: Se puede ver la *timeline* con la primera fase (el primer punto) sin nombre.

Es perfectamente entendible la inquietud del estudiante. Tomando como trabajo a futuro hacer un cambio en el diseño de esta advertencia, habría que indicar de alguna forma cuál es la fase sin nombrar.

6.5.2.7 Pregunta 7

“Con respecto a la pestaña de Correcciones ¿Con qué afirmaciones te sientes identificado/a?”

Esta pregunta está acompañada de una captura de pantalla, para dejar claro a los estudiantes a que pestaña se está haciendo referencia. Las opciones de respuesta eran:

- Encontré el diseño estético y agradable.
- La pantalla me pareció fácil de usar.
- Me agradó tener la corrección y comentarios de las docentes dentro de la misma aplicación.
- Encontré la pantalla difícil de entender.
- Siento que no era completa, ya que más de una vez tuve que consultar aspectos de la corrección por afuera con la profesora.
- Tuve problemas de estabilidad o respuesta relacionados a esta funcionalidad.
- Encontré incoherencias o errores en datos desplegados en la pantalla.
- Otro, cuál?

Pudiéndose seleccionar más de una opción.

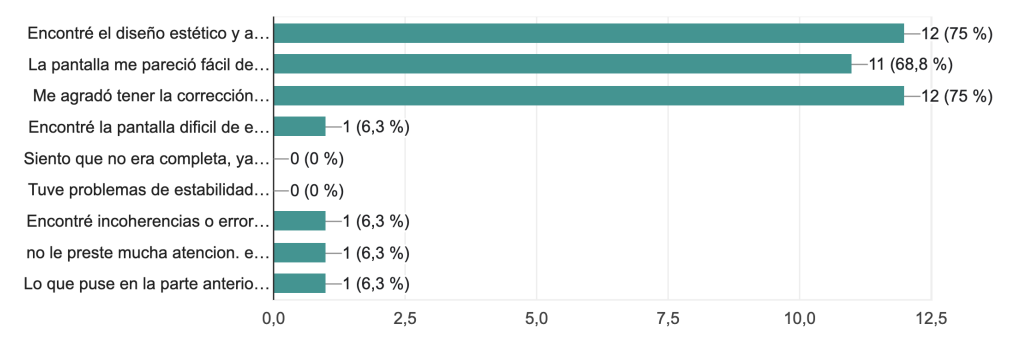


Figura 6.9: Se presentan gráficamente los resultados de la Pregunta 7.

En particular, los resultados de esta pregunta son sumamente positivos, doce de los dieciséis estudiantes votaron “Encontré el diseño estético y agradable.” y “Me agradó tener la corrección y comentarios de las docentes dentro de la misma aplicación.”, y once se sintieron identificados con “La pantalla me pareció fácil de usar.”. A su vez, no hubo ningún voto para “Siento que no era completa, ya que más de una vez tuve que consultar aspectos de la corrección por afuera con la profesora.” o “Tuve problemas de estabilidad o respuesta relacionados a esta funcionalidad.”.

únicamente hubo tres respuestas negativas o constructivas que manifestaban lo siguiente:

- Encontré incoherencias o errores en datos desplegados en la pantalla.
- Encontré la pantalla difícil de entender.
- Otro: “Lo que puse en la parte anterior, el botón *Instruction* está muy camuflado, no me daba cuenta que se podía hacer click, simplemente pensé que eran etiquetas.”. Este estudiante refiere a un comentario que él mismo hizo con respecto a la inactividad de la *UI* en la Pregunta 4. El comentario refiere a las siguientes etiquetas:

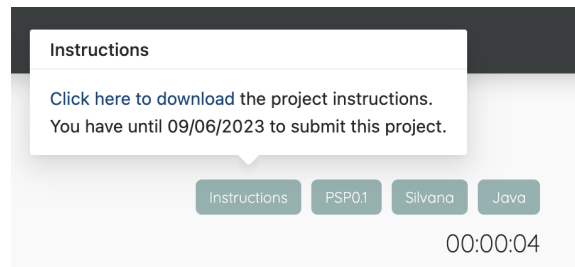


Figura 6.10: Estas etiquetas de información se encuentran en la parte superior de la vista de un proyecto.

Al hacer *hover* por la etiqueta de instrucciones, se hace explícito el link para descargar la letra del proyecto. Sin embargo es entendible la crítica del estudiante, ya que las otras *tags* de información no tienen una funcionalidad, e identificar que “Instructions” sí, puede ser confuso.

Cabe destacar que dichas etiquetas ya existían de la misma forma en la versión origen de PSPcode. Esta es la razón por la que el comentario no fue destacado en la Pregunta 4 6.5.2.4.

6.5.2.8 Pregunta 8

“¿Qué podrías destacar positivamente de la aplicación?”

La octava pregunta es de carácter general, y se pretende inferir si los estudiantes identifican positivamente las características añadidas en el marco de este Proyecto de Grado.

Similar a la pregunta anterior, en esta se les ofrece a los encuestados una serie de afirmaciones para que los mismos se identifiquen, las opciones fueron las siguientes:

- Nada.
- El diseño es estético y agradable.
- El uso de la aplicación es intuitivo.
- Los mensajes de error y advertencias eran claros y facilitaban el uso.
- La aplicación responde rápidamente en cada acción.
- Noté que la aplicación es estable, no me trajo problemas a lo largo del curso.
- Es cómodo hacer los ejercicios, comunicarme con los docentes, y ver las correcciones dentro de una misma plataforma.

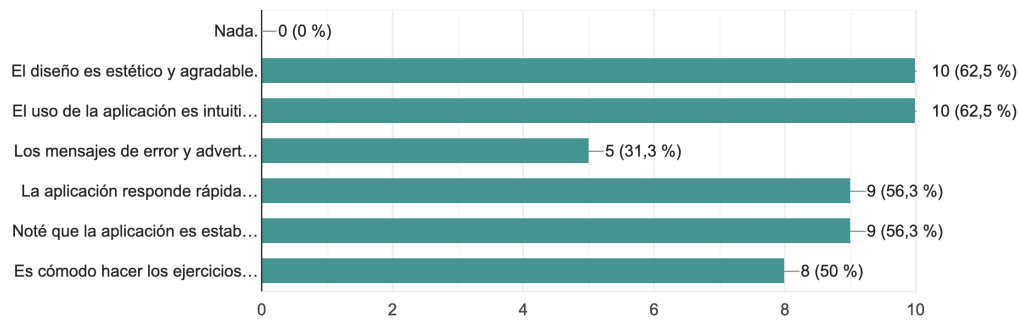


Figura 6.11: Se presentan gráficamente los resultados de la Pregunta 8.

Una vez más, los resultados son sumamente positivos. Ningún estudiante manifestó no encontrar nada positivo en la aplicación. A su vez, la selección del resto de opciones fue alta, superando el 50 % de los encuestados en la mayoría los casos.

Luego, como punto a mejorar se puede destacar que únicamente cinco de dieciséis estudiantes seleccionaron la afirmación: “Los mensajes de error y advertencias eran claros y facilitaban el uso.”. Este resultado no resulta sorprendente, ya que a lo largo de la encuesta, se ha evidenciado una tendencia consistente en la dificultad del usuario para identificar la ubicación del error que está cometiendo, en términos de la interfaz de usuario.

6.5.2.9 Pregunta 9

“¿Tu experiencia con la app. PSPcode, afecta (positiva o negativamente) la impresión que te llevas sobre el curso?”

La pregunta 9 tiene como objetivo reflejar la impresión que deja PSPcode en el usuario a lo largo del semestre, y cómo esta experiencia impacta en su percepción global del curso.

La pregunta presenta opciones con distintas afirmaciones, el encuestado solo puede seleccionar una:

- Si, dejaría de recomendar el curso por la aplicación.
- Si, no dejaría de recomendar el curso, pero la aplicación me dificultó su realización.
- No, la aplicación PSPcode no influyó en mi experiencia con el curso.
- Si, utilizar esta aplicación dedicada me facilitó la realización del curso.
- Si, la experiencia que tuve con la aplicación fue muy buena y me motiva a recomendar el curso.

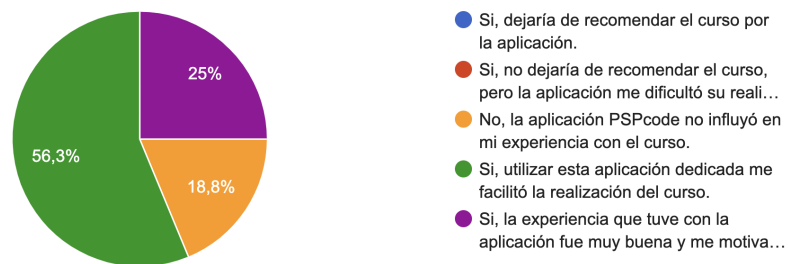


Figura 6.12: Se presentan gráficamente los resultados de la Pregunta 9.

Es muy positivo destacar que no hubo estudiantes que consideren mala su experiencia con PSPcode. Por otro lado, únicamente tres estudiantes consideran que su experiencia con la aplicación fue neutral, y no varía su impresión del curso.

Luego, más del 78% (trece estudiantes) de los encuestados manifiestan haber tenido una experiencia positiva. La buena experiencia con la aplicación motiva a un cuarto de los estudiantes a recomendar el curso. Por otro lado, más de la mitad de los estudiantes creen que la aplicación facilitó la realización del curso.

6.5.2.10 Pregunta 10

“¿Tienes algún otro comentario o feedback para la aplicación?”

Para finalizar se da la oportunidad a los encuestados de aportar algún otro comentario.

Se obtuvieron diez respuestas. Entre ellas, comentarios positivos y comentarios constructivos.

Hay dos respuestas que refieren a temas anteriormente abordados en este análisis. Uno enfatiza en hacer más llamativa la etiqueta de “*Instructions*” explicitando en mayor medida que se trata de un botón para acceder a la letra del proyecto. El otro vuelve a mencionar que le gustaría ver más específicas las razones por las cuales está deshabilitado el botón *Submit Project*. Si bien se muestra una *checklist*, es cierto que no está explícita en ningún lado cuál es la fase que no cumple con los requisitos.

Hubo dos comentarios manifestando que en el caso de colocar un cero (0) en un campo numérico, el mismo desaparecía una vez que se recargaba la página. Esto se daba ya que el código de Frontend solo colocaba en el campo el valor obtenido de Backend si el mismo se considera `true` en un contexto booleano, de lo contrario no poblaba el campo, 0 es un valor que devuelve `false` en una evaluación booleana, entonces el campo se mostraba vacío. El problema fue arreglado, permitiendo mostrar el 0.

Hubo un comentario explicando la incomodidad que tuvo con respecto al problema con la pestaña de *Corrections* oculta [6.4.1.6](#), el mismo fue reportado por parte de las profesoras y solucionado en la aplicación.

Luego, también se obtuvo *feedback* positivo. Por ejemplo “La app ayudó a organizar cada proyecto, estaría bueno que se pueda usar para más materias.” y “Nada más que agregar que muy buena la aplicación. felicitaciones y el mejor de los éxitos en el futuro.”.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

El curso Principios y Fundamentos del PSP de la Facultad de Ingeniería, lleva once años instruyendo a estudiantes avanzados en el Proceso Personal de Software de Humphrey. Esta tarea supone numerosos desafíos para el cuerpo docente. El Proyecto de Grado de 2018 fue un punto de inflexión para la operativa del curso. Al incluir la herramienta PSPcode, parte de las problemáticas que docentes y estudiantes enfrentaban hasta entonces, fueron mitigadas.

El presente proyecto comenzó motivado por dos objetivos principales: la extensión de PSPcode mediante la integración de un módulo de correcciones para ofrecer a los usuarios una experiencia completa dentro de la aplicación, y la migración de la plataforma a otro entorno de alojamiento para evitar las restricciones existentes en el anterior.

El módulo de correcciones se completó exitosamente. A los usuarios con rol de Estudiante se les ofrecen sugerencias mientras realizan sus proyectos, con el objetivo de evitar confusiones en el uso de la aplicación y garantizar entregas libres de errores, como puede ser un campo vacío. También se agregaron numerosas observaciones automáticas en la pantalla de revisión de un proyecto, para simplificar el trabajo de los usuarios de rol Docente. Para finalizar, como característica principal del módulo, se ha incluido una pantalla dedicada a la corrección de proyectos, que incluye todos los campos que las docentes solían completar al utilizar la planilla del documento de texto. De esta manera, el ciclo completo de trabajo de PF-PSP puede realizarse sin salir de PSPcode.

Se llevó a cabo la transición a un nuevo plan de alojamiento. En una primera fase, se hizo un cambio de plan dentro de Heroku para hacer frente al curso de 2022. Posteriormente, se exploraron y evaluaron diversas alternativas de *hosting*, decantando en Digital Ocean. El esfuerzo dedicado a la configuración de ambientes y despliegues, fue fundamental para lograr la mayor escalabilidad, robustez y *performance*, al menor precio. Al mismo tiempo, permitió que el equipo aprendiera y se capacitara fuertemente en el área.

Por otro lado, mediante modificaciones realizadas en la plataforma Admin, se le brinda a los docentes mayor control de las entidades, para tener una experiencia personalizada, pudiendo configurar y adaptar el curso a nuevos procesos PSP o fases del mismo.

Al inicio del proyecto, la deuda técnica propia de una aplicación que no recibió mantenimiento durante cinco años, llevó a que el equipo se centrara en la actualización y

refactorización del código existente. La relevancia dada a esta tarea, permitió culminar el proyecto con una aplicación mantenible y moderna, habilitando el desarrollo futuro de la misma. Sin embargo, el tiempo invertido en el refinamiento del código, relegó la integración con el módulo de estadísticas R a un segundo plano, sin llevar a cabo su implementación.

Por otro lado, se llevaron a cabo importantes mejoras en la interfaz de PSPcode, enfocadas en hacerla más intuitiva, fluida y visualmente agradable. Durante este proceso, se mantuvo la esencia y la paleta de colores original de la página, conservando así la identidad visual característica de PSPcode. Dado que los usuarios finales de la aplicación (de rol de Estudiante) varían año tras año, no fue posible realizar una validación comparativa de la experiencia del usuario en este grupo específico. Sin embargo, la aplicación fue validada con las docentes y se llevó a cabo una encuesta entre los estudiantes de 2023 para recopilar sus opiniones.

En relación a ello y adicionando al alcance planteado al inicio del proyecto, la nueva aplicación fue utilizada como herramienta oficial del curso 2023. Este hito amplió la duración estimada del proyecto, pero aportó gran valor al cierre del mismo. Gracias a su uso en producción, se lograron depurar problemas que no habían surgido en período de pruebas. Además, se concluyó el proyecto encuestando a los estudiantes de PF-PSP.

El *feedback* obtenido de los usuarios fue mayoritariamente positivo, mostrándose conformes con el desempeño de la aplicación, y destacando su utilidad para el desarrollo del curso. En particular, el módulo de correcciones fue ampliamente aceptado, recibiendo pocas sugerencias de mejora. Por otro lado, se obtuvo de la encuesta, parte del trabajo futuro considerado como primordial para potenciar la aplicación.

Luego del trabajo realizado, PSPcode experimentó una transformación significativa que impacta tanto a docentes como a estudiantes. Se logró cumplir el objetivo principal del proyecto: facilitar el trabajo operativo de las docentes con una aplicación más práctica y completa. Las mejoras introducidas en la aplicación han hecho que la tarea de evaluar y supervisar los proyectos de los estudiantes sea mucho más sencilla. A su vez, gracias a los cambios en funcionalidad y usabilidad de la aplicación, los estudiantes podrán tener una mejor experiencia de usuario, representando una evolución en el proceso educativo de PF-PSP.

7.2. Trabajos a futuro

En cuanto a las próximas etapas del proyecto, resulta esencial llevar a cabo un mantenimiento periódico, que incluya la actualización de las bibliotecas y un análisis exhaustivo del código al menos una vez cada dos años. Este enfoque prevendría situaciones tediosas, como la necesidad realizar una actualización y refactorización completa de un código que ha carecido de mantenimiento durante un período prolongado, como se dio en los últimos cinco años.

Por otro lado, se recomienda realizar un cuestionario a los estudiantes cada año, similar al planteado en esta oportunidad. Será crucial para recabar opiniones que guíen las fases de desarrollo subsiguientes y aseguren que la aplicación PSPcode se mantenga alineada con los objetivos generales de docentes y estudiantes. Como principal *feedback* obtenido en la encuesta, y no abordado aún, se encuentra:

- La funcionalidad de los *tags* informativos en la vista del proyecto fue confusa para

algunos usuarios, hubo estudiantes que tuvieron problemas para acceder a las instrucciones del proyecto desde allí. Se aconseja cambiar el diseño o la ubicación del botón de instrucciones, para que se denote su funcionalidad.

- Indicar los errores de manera más específica, por ejemplo cuando un estudiante no puede entregar el proyecto porque una de las fases carece de información obligatoria. Convendría indicar exactamente qué fase está fallando.
- Se considera necesario enfatizar en la explicación sobre el orden correcto de las fases, y cuáles de ellas son obligatorias. El *tooltip* agregado al título de la pestaña de fases no resultó suficiente en varios casos.

Cabe destacar que se buscó mejorar la intuitividad y usabilidad de la aplicación, pero no se pretende que PSPcode sea autoexplicativa con respecto a conceptos y teoría abarcada en los contenidos del curso PF-PSP.

Como ya se mencionó, uno de los objetivos de las docentes de PF-PSP aún pendiente, es la integración de la herramienta estadística R para potenciar las capacidades analíticas de la aplicación. Dado que el curso tiene fines estadísticos sobre sus resultados, sería importante agregar este módulo en un futuro.

Bibliografía

- [1] Watts S. Humphrey. The personal software process (psp). Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 2000.
- [2] Watts S. Humphrey. The personal software process: Status and trends, December 2000. The Software Engineering Institute.
- [3] Guillermo Kuster and Guillermo Tavidian. *Herramienta de soporte a la aplicación del Personal Software Process (PSP)*. Thesis, Facultad de Ingeniería - Udelar, 2018.
- [4] Eva Fing. Principios y fundamentos del proceso personal de software, 2023. <https://eva.fing.edu.uy/course/view.php?id=633>.
- [5] Silvana Moreno. *The Practice of Software Detailed Design of Graduating Students: A family of experiments*. PhD thesis, Facultad de Ingeniería - Udelar, Agosto 2022.
- [6] Process dashboard, 2023. <https://www.processdash.com/>.
- [7] Omar S. Gómez, Antonio A. Aguilera, Gerzon E. Gómez, and Raúl A. Aguilar. Estudio del proceso software personal (psp) en un entorno académico. *ReCIBE. Revista electrónica de Computación, Informática, Biomédica y Electrónica*, 2014.
- [8] Vanessa Casella, Silvana Moreno, Martin Solari, and Diego Vallespir. Representation of software design using templates: impact on software quality and effort. *Journal of Software Engineering Research and Development*, 2021.
- [9] Silvana Moreno, Diego Vallespir, and Martin Solari. An experiment on how graduating students represent software designs. In *XXV Ibero-American Conference on Software Engineering (CIBSE 2022)*, 2022.
- [10] Help and documentation for the ruby programming language, 2023. <https://ruby-doc.org/>.
- [11] Ruby on rails guides, 2023. <https://guides.rubyonrails.org/>.
- [12] Rubygems documentation, 2023. <https://rubygems.org/>.
- [13] Sendgrid documentation, 2023. <https://docs.sendgrid.com/es-mx/>.
- [14] Twilio documentation, 2023. <https://www.twilio.com/>.
- [15] Heroku documentation, 2023. <https://devcenter.heroku.com/categories/reference>.
- [16] PostgreSQL documentation, 2023. <https://www.postgresql.org/docs/14/intro-what-is.html>.
- [17] pgadmin documentation, 2023. <https://www.pgadmin.org/>.
- [18] Docker overview, 2023. <https://docs.docker.com/get-started/overview/>.
- [19] Amazon web services documentation, 2023. <https://docs.aws.amazon.com/>.

- [20] Amazon elastic compute cloud documentation, 2023. https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html.
- [21] Amazon simple storage service documentation, 2023. https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html.
- [22] Amazon elastic container registry documentation, 2023. https://docs.aws.amazon.com/es_es/AmazonECR/latest/userguide/what-is-ecr.html.
- [23] Digital ocean documentation, 2023. <https://docs.digitalocean.com/>.
- [24] Cloudflare history, 2023. <https://www.cloudflare.com/es-es/our-story/>.
- [25] Cloudflare documentation, 2023. <https://developers.cloudflare.com/>.
- [26] Netscape Communications Corporation. Javascript documentation, 2023. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [27] Meta. React js documentation, 2023. <https://react.dev/learn>.
- [28] Sebastian McKenzie. Babel documentation, 2023. <https://babeljs.io/docs/>.
- [29] Stack overflow survey, 2022. <https://survey.stackoverflow.co/2022/#overview>.
- [30] Isaac Z. Schlueter. npm (node package manager) documentation, 2023. <https://docs.npmjs.com/>.
- [31] Meta. Yarn documentation, 2023. <https://yarnpkg.com/getting-started>.
- [32] Tobias Koppers. Webpack documentation, 2023. <https://webpack.js.org/concepts/>.
- [33] Pspcode frontend github repository. https://github.com/liaMalvarez/PSPCode-v2_frontend.git.
- [34] Pspcode backend github repository. https://github.com/GustavoAudi/PSPCode-v2_backend.git.
- [35] Tobias Koppers. Webpack: To v3 from v1 or v2, 2023. <https://webpack.js.org/migrate/3/>.
- [36] Tobias Koppers. Webpack: To v4 from v3, 2023. <https://webpack.js.org/migrate/4/>.
- [37] Meta. React v16.8 documentation, 2023. <https://legacy.reactjs.org/blog/2019/02/06/react-v16.8.0.html>.
- [38] Babel migration from v6 to v7, 2023. <https://babeljs.io/docs/v7-migration>.
- [39] Tobias Koppers. Webpack: To v5 from v4, 2023. <https://webpack.js.org/migrate/5/>.
- [40] Ant design, migration v3 to v4, 2023. <https://4x.ant.design/docs/react/migration-v4>.
- [41] npx unimported command documentation, 2023. <https://www.npmjs.com/package/unimported>.
- [42] Webpack v5 production setup, 2023. <https://webpack.js.org/guides/production/#setup>.
- [43] npm webpack-merge library documentation, 2023. <https://www.npmjs.com/package/webpack-merge>.

Anexo 1 - Revisión de Antecedentes

.1. Frontend

.1.1. package.json inicial

```
1 {
2   "name": "react_redux_base",
3   "version": "0.1.0",
4   "description": "",
5   "engines": {
6     "npm": ">4",
7     "node": ">5"
8   },
9   "scripts": {
10    "preinstall": "node tools/nodeVersionCheck.js",
11    "start-message": "babel-node tools/startMessage.js",
12    "prestart": "npm-run-all --parallel start-message remove-dist",
13    "start": "npm-run-all --parallel open:src lint:watch",
14    "open:src": "babel-node tools/srcServer.js",
15    "open:dist": "babel-node tools/distServer.js",
16    "lint:code": "esw webpack.config.* src tools --color",
17    "lint:style": "stylelint \"src/styles/*.scss\"",
18    "lint": "npm run lint:code && npm run lint:style",
19    "lint:watch": "npm run lint:style && npm run lint:code -- --watch",
20    "clean-dist": "npm run remove-dist && mkdir dist",
21    "remove-dist": "rimraf ./dist",
22    "prebuild": "npm run clean-dist && npm run lint && npm run test",
23    "build": "babel-node tools/build.js && npm run open:dist",
24    "just-build": "babel-node tools/build.js",
25    "test": "jest",
26    "test:CI": "node --harmony_proxies node_modules/jest/bin/jest",
27    "test:cover": "npm run test -- --coverage",
28    "test:cover:CI": "npm run test:cover && cat ./coverage/lcov.info |
29    node_modules/coveralls/bin/coveralls.js",
30    "test:watch": "npm run test -- --watch",
31    "open:cover": "npm run test:cover && open coverage/lcov-report/index.html",
32    "deploy:staging": "npm run just-build && babel-node tools/deployS3
33    staging",
34    "deploy:production": "npm run just-build && babel-node tools/deployS3
35    production",
36    "deploy:gh": "npm run just-build && gh-pages -d dist",
37    "analyze-bundle": "babel-node ./tools/analyzeBundle.js"
38  },
39 }
```

```

36 "author": "Guillermo y Guillermo",
37 "license": "MIT",
38 "homepage": "https://taavidian.github.io/psp-react",
39 "dependencies": {
40   "antd": "3.0.0-alpha.13",
41   "immutable": "3.8.1",
42   "isomorphic-fetch": "2.2.1",
43   "lodash": "4.17.4",
44   "moment": "2.18.1",
45   "prop-types": "15.5.10",
46   "react": "15.5.4",
47   "react-dom": "15.5.4",
48   "react-ga": "2.4.1",
49   "react-moment": "0.6.5",
50   "react-redux": "5.0.4",
51   "react-router": "3.0.5",
52   "react-router-redux": "4.0.8",
53   "react-router-scroll": "0.4.2",
54   "redux": "3.6.0",
55   "redux-form": "6.7.0",
56   "redux-logger": "3.0.1",
57   "redux-react-session": "2.0.0",
58   "redux-thunk": "2.2.0",
59   "validate.js": "0.11.1"
60 },
61 "devDependencies": {
62   "autoprefixer": "6.7.3",
63   "babel-cli": "6.24.1",
64   "babel-core": "6.24.1",
65   "babel-eslint": "7.1.1",
66   "babel-jest": "18.0.0",
67   "babel-loader": "7.0.0",
68   "babel-plugin-transform-react-constant-elements": "6.23.0",
69   "babel-plugin-transform-react-remove-prop-types": "0.3.2",
70   "babel-polyfill": "6.23.0",
71   "babel-preset-env": "1.5.1",
72   "babel-preset-react": "6.24.1",
73   "babel-preset-stage-1": "6.24.1",
74   "babel-register": "6.24.1",
75   "chalk": "1.1.3",
76   "connect-history-api-fallback": "1.3.0",
77   "coveralls": "2.11.16",
78   "css-loader": "0.28.1",
79   "enzyme": "2.8.2",
80   "eslint": "3.19.0",
81   "eslint-config-airbnb": "15.0.1",
82   "eslint-plugin-import": "2.2.0",
83   "eslint-plugin-jsx-a11y": "5.0.3",
84   "eslint-plugin-react": "7.0.1",
85   "eslint-watch": "3.1.0",
86   "express": "4.15.2",
87   "extract-text-webpack-plugin": "2.1.0",
88   "file-loader": "0.11.1",
89   "gh-pages": "1.1.0",
90   "html-webpack-plugin": "2.28.0",
91   "identity-obj-proxy": "3.0.0",

```

```

92     "jest": "20.0.1",
93     "json-loader": "0.5.4",
94     "nock": "9.0.13",
95     "node-sass": "4.5.3",
96     "npm-run-all": "4.0.2",
97     "opn": "5.0.0",
98     "postcss-loader": "1.3.3",
99     "prompt": "1.0.0",
100    "react-hot-loader": "3.0.0-beta.7",
101    "react-test-renderer": "15.5.4",
102    "redux-immutable-state-invariant": "2.0.0",
103    "redux-mock-store": "1.2.1",
104    "replace": "0.3.0",
105    "rimraf": "2.5.4",
106    "sass-loader": "6.0.5",
107    "style-loader": "0.17.0",
108    "stylelint": "7.10.1",
109    "stylelint-config-standard": "16.0.0",
110    "stylelint-order": "0.2.2",
111    "url-loader": "0.5.8",
112    "webpack": "2.5.1",
113    "webpack-bundle-analyzer": "2.6.0",
114    "webpack-dev-middleware": "1.10.2",
115    "webpack-hot-middleware": "2.18.0",
116    "webpack-md5-hash": "0.0.5"
117  },
118  "jest": {
119    "moduleNameMapper": {
120      "\\.(css|scss)$": "identity-obj-proxy",
121      "\\.(gif|ttf|eot|svg|woff|woff2|ico)$": "<rootDir>/tools/fileMock.js"
122    },
123    "globals": {
124      "config": {
125        "API_URL": "localhost:3000"
126      }
127    }
128  },
129  "keywords": [],
130  "repository": {
131    "type": "git",
132    "url": ""
133  }
134 }

```

.2. Backend

.2.1. Modelo de dominio de la aplicación previa a los cambios

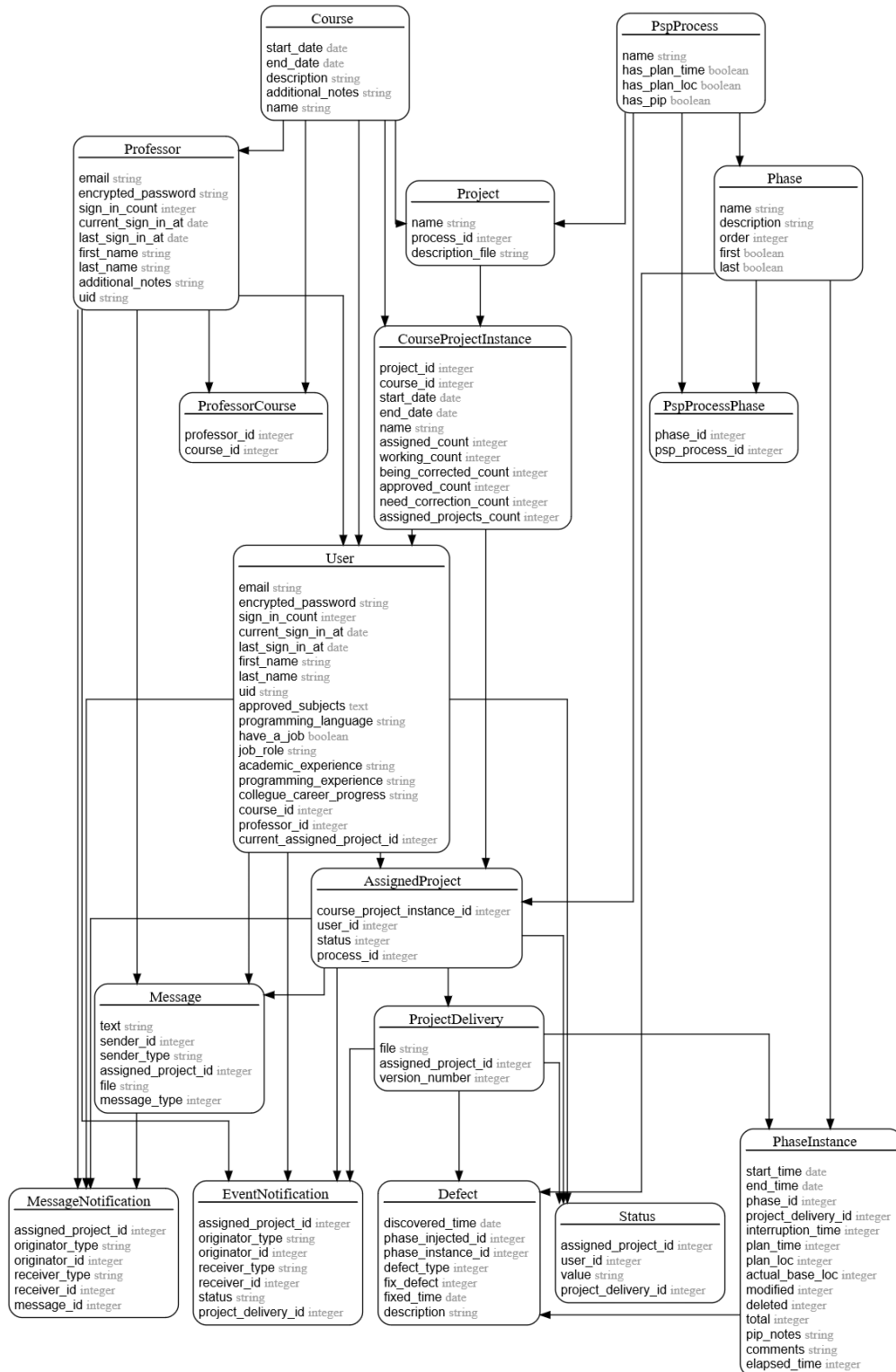


Figura 1: Modelo de dominio del sistema resultante del Proyecto de Grado finalizado en 2018.

.2.2. Modelo de dominio parcial sobre la pestaña de correcciones

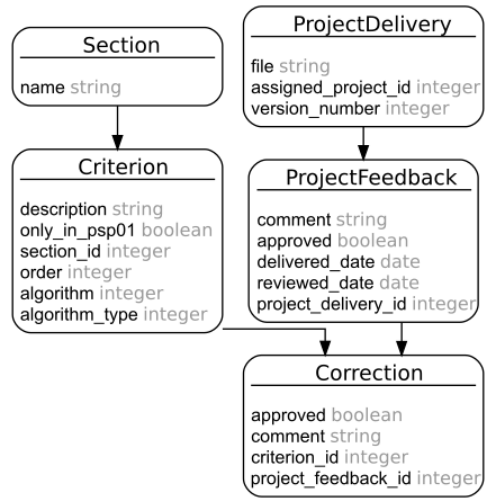


Figura 2: Parte del modelo de dominio relacionado a la pestaña de correcciones.

Anexo 2 - Actualización y Refactorización de la aplicación

.3. Frontend

.3.1. Wbpack v4 y React 16.8 package.json

```
1 {
2   "name": "react_redux_base",
3   "version": "0.1.0",
4   "description": "",
5   "engines": {
6     "npm": ">4",
7     "node": ">5"
8   },
9   "scripts": {
10    "preinstall": "node tools/nodeVersionCheck.js",
11    "start-message": "babel-node tools/startMessage.js",
12    "start": "webpack-dev-server --open",
13    "open:src": "babel-node tools/srcServer.js",
14    "open:dist": "babel-node tools/distServer.js",
15    "clean-dist": "npm run remove-dist && mkdir dist",
16    "remove-dist": "rimraf ./dist",
17    "prebuild": "npm run clean-dist && npm run lint && npm run test",
18    "build": "babel-node tools/build.js && npm run open:dist",
19    "just-build": "babel-node tools/build.js",
20    "test": "jest",
21    "test:CI": "node --harmony_proxies node_modules/jest/bin/jest",
22    "test:cover": "npm run test -- --coverage",
23    "test:cover:CI": "npm run test:cover && cat ./coverage/lcov.info |
24    node_modules/coveralls/bin/coveralls.js",
25    "test:watch": "npm run test -- --watch",
26    "open:cover": "npm run test:cover && open coverage/lcov-report/index.html",
27    "deploy:staging": "npm run just-build && babel-node tools/deployS3 staging",
28    "deploy:production": "npm run just-build && babel-node tools/deployS3
29    production",
30    "deploy:gh": "npm run just-build && gh-pages -d dist",
31    "analyze-bundle": "babel-node ./tools/analyzeBundle.js"
32  },
33  "author": "Guillermo y Guillermo",
34  "license": "MIT",
35  "homepage": "https://tavidian.github.io/psp-react",
36  "dependencies": {
37    "@babel/node": "7.16.8",
```

```

36   "antd": "3.0.0-alpha.13",
37   "immutable": "3.8.1",
38   "isomorphic-fetch": "2.2.1",
39   "lodash": "4.17.4",
40   "moment": "2.24.0",
41   "postcss": "7.0.0",
42   "prop-types": "15.7.2",
43   "react": "16.8.4",
44   "react-dom": "16.8.4",
45   "react-ga": "2.4.1",
46   "react-moment": "1.0.0",
47   "react-redux": "5.0.4",
48   "react-router": "6",
49   "react-router-dom": "6",
50   "react-router-redux": "4.0.8",
51   "redux": "3.6.0",
52   "redux-form": "7.0.0",
53   "redux-logger": "3.0.1",
54   "redux-react-session": "2.6.1",
55   "redux-thunk": "2.2.0",
56   "remove-trailing-separator": "^1.0.1",
57   "validate.js": "0.11.1",
58   "webpack-cli": "2.0.14",
59   "webpack-dev-server": "3.0.0"
60 },
61 "devDependencies": {
62   "@babel/cli": "7.17.6",
63   "@babel/core": "7.17.9",
64   "@babel/eslint-parser": "7.17.0",
65   "@babel/plugin-transform-react-constant-elements": "7.17.6",
66   "@babel/polyfill": "7.12.1",
67   "@babel/preset-env": "7.16.11",
68   "@babel/preset-react": "7.16.7",
69   "@babel/preset-stage-1": "7.8.3",
70   "@babel/register": "7.17.7",
71   "autoprefixer": "6.7.3",
72   "babel-jest": "27.5.1",
73   "babel-loader": "8.2.5",
74   "babel-plugin-transform-react-remove-prop-types": "0.4.24",
75   "chalk": "1.1.3",
76   "connect-history-api-fallback": "1.3.0",
77   "coveralls": "2.11.16",
78   "css-loader": "0.28.1",
79   "enzyme": "3.0.0",
80   "eslint": "8.14.0",
81   "eslint-config-airbnb": "19.0.4",
82   "eslint-plugin-import": "2.26.0",
83   "eslint-plugin-jsx-a11y": "6.5.1",
84   "eslint-plugin-react": "7.29.4",
85   "eslint-watch": "8.0.0",
86   "express": "4.15.2",
87   "file-loader": "0.11.1",
88   "gh-pages": "1.1.0",
89   "html-webpack-plugin": "4.0.0",
90   "identity-obj-proxy": "3.0.0",
91   "jest": "20.0.1",

```

```

92     "json-loader": "0.5.4",
93     "mini-css-extract-plugin": "1.0.0",
94     "nock": "9.0.13",
95     "node-sass": "5.0.0",
96     "npm-run-all": "4.0.2",
97     "opn": "5.0.0",
98     "postcss-loader": "4.0.0",
99     "prompt": "1.0.0",
100    "react-hot-loader": "3.0.0-beta.7",
101    "react-test-renderer": "16.0.0",
102    "redux-immutable-state-invariant": "2.0.0",
103    "redux-mock-store": "1.2.1",
104    "replace": "0.3.0",
105    "rimraf": "2.5.4",
106    "sass-loader": "7.0.0",
107    "style-loader": "0.17.0",
108    "stylelint": "7.10.1",
109    "stylelint-config-standard": "16.0.0",
110    "stylelint-order": "0.2.2",
111    "url-loader": "0.5.8",
112    "webpack": "4.0.0",
113    "webpack-bundle-analyzer": "2.6.0",
114    "webpack-dev-middleware": "3.0.0",
115    "webpack-hot-middleware": "2.25.1",
116    "webpack-md5-hash": "0.0.5"
117  },
118  "jest": {
119    "moduleNameMapper": {
120      "\\.(css|scss)$": "identity-obj-proxy",
121      "\\.(gif|ttf|eot|svg|woff|woff2|ico)$": "<rootDir>/tools/fileMock.js"
122    },
123    "globals": {
124      "config": {
125        "API_URL": "localhost:3000"
126      }
127    }
128  },
129  "keywords": [],
130  "repository": {
131    "type": "git",
132    "url": ""
133  }
134 }

```

.3.2. Cambios en la versión de Babel

Al actualizar Babel de la versión 6.x.x a la 7.x.x, el *package.json* varía desde:

```

1 "devDependencies": {
2   ...
3   "babel-cli": "6.24.1",
4   "babel-core": "6.24.1",
5   "babel-eslint": "7.1.1",
6   "babel-jest": "18.0.0",

```

```

7   "babel-loader": "7.0.0",
8   "babel-plugin-transform-react-constant-elements": "6.23.0",
9   "babel-plugin-transform-react-remove-prop-types": "0.3.2",
10  "babel-polyfill": "6.23.0",
11  "babel-preset-env": "1.5.1",
12  "babel-preset-react": "6.24.1",
13  "babel-preset-stage-1": "6.24.1",
14  "babel-register": "6.24.1",
15  ...
16 }

```

hacia:

```

1  "dependencies": {
2    ...
3    "@babel/node": "7.16.8",
4    ...
5  }
6  "devDependencies": {
7    ...
8    "@babel/cli": "7.17.6",
9    "@babel/core": "7.17.9",
10   "@babel/eslint-parser": "7.17.0",
11   "@babel/plugin-transform-react-constant-elements": "7.17.6",
12   "@babel/polyfill": "7.12.1",
13   "@babel/preset-env": "7.16.11",
14   "@babel/preset-react": "7.16.7",
15   "@babel/preset-stage-1": "7.8.3",
16   "@babel/register": "7.17.7",
17   ...
18 }

```

.3.3. Transición detallada de componentes de clase a funcionales

1. Sustituir la sintaxis de declaración de clase por una declaración de función. Se decide utilizar *arrow functions* para la declaración de funciones ya que permiten una sintaxis más compacta y legible en comparación con las funciones regulares. Se elimina la palabra reservada *functions*, y hay casos donde puede hacerse un **return** implícito, lo que hace que el código sea más breve y fácil de leer.

Ejemplo dado con una función que retorna la suma de sus dos parámetros:

```

1  const myArrowSum = (a, b) => a + b; // arrow function
2
3  function myFuncSum(a, b) { // common function
4    return a + b;
5  }

```

De esta manera, los componentes pasan de declararse como clases:

```

1 class MyComponent extends React.Component {
2   ...
3   render() {
4     ...
5     return (
6       JSX syntax
7     )
8   }
9 }

```

A declararse como *arrow functions*:

```

1 const MyComponent = (props) => {
2   ...
3   return ( // implicit return los casos donde no hay logica previa
4     JSX syntax
5   )
6 }

```

Hay casos donde los componentes son sencillos y no necesitan manejar estados, declarar funciones dentro de su cuerpo, u otro tipo de lógica fuera del `return`, por ende, al usar *functional components* puede aplicarse el `return` implícito ejemplificado anteriormente, lo que dejaría el componente funcional aún más compacto en comparación con el de clase.

Se puede notar también como el método `render` ya no es necesario en componentes funcionales, como sí lo es en componentes de clase para escribir y retornar el código `jsx`. El método `render` se dispara cuando el componente debe ser actualizado, por ejemplo cuando una propiedad que el componente recibe es modificada.

Otra diferencia es que para declarar un componente de React en forma de clase, debe usarse la clase padre `Component` provista por React, esto es necesario para contar con los métodos propios de dicha clase y así manejar el ciclo de vida, estado y propiedades del componente, como son: `render()`, `componentDidMount()`, `componentDidUpdate()`, `setState()`, entre otros.

2. Se elimina el método `constructor` propio de las clases, ya que en los componentes funcionales no es necesario. Las declaraciones y llamados a funciones que hay en el constructor son incluidos en el *functional component* como se explica en los siguientes puntos.
3. Se reemplaza el manejo de los estados de la clase por los estados generados por el Hook `useState`.
 - a) Tomando de ejemplo un componente de clase que declara un estado llamado “myState” en su constructor, así sería el pasaje a *functional component*:

```

1   class MyComponent extends React.Component {
2     constructor(props) {
3       super(props); // recibe las props inherentes de React.
         Component
4
5
6       this.state = {
7         myState: 'initial value',
8       };
9     }
10
11    // resultado como componente funcional
12
13    const MyComponent = (props) => {
14      const [myState, setState] = React.useState('initial value');
15    }

```

El hook `useState` devuelve un *array* que se puede desestructurar (como se ve en el ejemplo) en dos ítems: el primero guarda el valor del estado como una variable inmutable, y el segundo es una función que permite modificar este valor, no puede ser modificado de otra manera.

- b) A continuación reemplazamos todas las actualizaciones que se le hacen al estado con la nueva sintaxis. En *class components*, estas modificaciones de estado se hacen accediendo a la instancia de clase mediante el `.this` y luego al método `setState` que expone la clase; mientras que en *functional components* se utiliza la función devuelta por el hook `setState` descrita anteriormente.

```

1   this.setState({myState: 'new state value'}) // sintaxis anterior
2
3   setMyState('new state value') // nueva sintaxis usando el hook

```

- c) Finalizando, reemplazamos también todas las lecturas del valor del estado:

```

1   this.state.myState; // sintaxis de lectura anterior
2
3
4   myState // sintaxis de lectura actual

```

4. Similarmente, se reemplazan las funciones que manejan el ciclo de vida en un componente de clase, por el *hook* de `useEffect` utilizando el siguiente criterio:

- a) **componentDidMount**: Este método del ciclo de vida se ejecuta una vez que el componente ha sido montado en el DOM. Usando `useEffect`, se puede lograr el mismo comportamiento dejando vacío el arreglo de dependencias. Esto asegura que el efecto se ejecute solo una vez después del primer renderizado.

El mismo proceso se realiza para reemplazar las funciones llamadas y código escrito en el `constructor`.

- b) **componentDidUpdate**: Este método del ciclo de vida se ejecuta cada vez que el componente se actualiza y se vuelve a renderizar, esto sucede cuando una propiedad o un estado cambia. El método `componentDidUpdate` recibe como parámetros el valor de las propiedades y los estados en el ciclo de vida anterior, por lo que es frecuente discriminar la lógica dentro del método (utilizando una estructura de control condicional, como `if/else`) dependiendo qué estado o propiedad se vio afectada al dispararse el método. Con `useEffect`, se puede lograr un comportamiento similar especificando las dependencias relevantes en el arreglo de dependencias. Esto hará que el efecto (la lógica declarada dentro de su *callback*) se ejecute cada vez que alguna de las dependencias cambie. Es usual reemplazar `componentDidUpdate` por más de un `useEffect`, ya que las dependencias especificadas para el *hook* actúan de manera similar a las condiciones de disparo en una estructura de control condicional.

- c) **componentWillUnmount**: Este método del ciclo de vida se ejecuta justo antes de que el componente sea desmontado y eliminado del DOM, generalmente en él se hacen tareas de limpieza, como cancelar suscripciones o eliminar *event listeners*. Se puede lograr un comportamiento similar con un `useEffect`, devolviendo (en su `return`) una función de limpieza del efecto. Esta función se ejecutará antes de que el componente sea desmontado, lo que permite reemplazar el `componentWillUnmount`.

5. Finalmente, debemos corregir la sintaxis de los métodos utilizados en el componente de clase. En un *class component*, los métodos se definen como funciones dentro de la clase utilizando la sintaxis de JavaScript siguiendo las reglas de la programación orientada a objetos. Por ejemplo:

```
1 class MyComponent extends React.Component {
2   ...
3   someDeclaredMethod() {
4     ...
5   }
6 }
```

Mientras que en un componente funcional, debe declararse una nueva función dentro de él, para obtener un método ejecutable. Esto puede hacerse mediante las dos sintaxis de funciones presentadas al inicio de la sección: utilizando la palabra reservada `function` o como función flecha. En este proyecto se opta por la utilización de funciones flechas, los motivos fueron explicitados anteriormente.

El código con la nueva sintaxis se ve así:

```
1 const MyComponent = (props) => {
2   const someDeclaredMethod = () => {
3     ...
4   }
5 }
```

Luego, también debe modificarse a la llamada a la función:

En un *class component*, los métodos se llaman utilizando la sintaxis: `this.nombreDelMetodo()`. Esto se debe a que los métodos están asociados a la instancia de la clase. A su vez, en una clase es necesario hacer un *bind* si el método va a ser ejecutado dentro del código `jsx`. Es especialmente necesario cuando dentro del método declarado, se utiliza *this*. para acceder, por ejemplo, a una prop o un estado. El *bind* se utiliza para asegurar que el contexto *this* usado dentro de ese método, se refiera correctamente a la instancia del componente, en lugar de apuntar a *undefined* o a otro objeto. Hay dos formas de “bindear” un método:

- En el constructor:

```
1 class MyComponent extends React.Component {
2   constructor(props) {
3     super(props);
4     this.handleClick = this.handleClick.bind(this);
5   }
6
7   handleClick() {
8     // Cuerpo del metodo handleClick
9   }
10
11  render() {
12    return (
13      <button onClick={this.handleClick}>Click me</button>
14    );
15  }
16 }
```

- En la llamada al método:

```
1 class MyComponent extends React.Component {
2   handleClick() {
3     // Cuerpo del metodo handleClick
4   }
5
6   render() {
7     return (
8       <button onClick={this.handleClick.bind(this)}>Click me</
9       button>
10    );
11  }
12 }
```


Sin embargo, el uso del *bind* puede evitarse, si los métodos de la clase son declarados como funciones flecha ya que éstas heredan automáticamente el contexto (*this*) del componente padre, lo que evita la necesidad de hacer el enlace manualmente. Esta es otra importante ventaja de las *arrow functions*, que no se había mencionado previamente ya que la diferencia entre *class* y *functional components*, y el concepto de *bind* no habían sido introducidos aún.

```
1 class MyComponent extends React.Component {
2   handleClick = () => {
3     // Código del método handleClick
4   }
5
6   render() {
7     return (
8       <button onClick={this.handleClick}>Click me</button>
9     );
10  }
11 }
```

De esta manera, se concluye la descripción de los cambios necesarios para transicionar de un componente de clase a uno funcional. Se puede ver como el segundo tipo es notoriamente más compacto y sencillo de manejar.

3.4. Eliminación de código muerto

El 21 de Diciembre de 2022 se realizó una primera iteración del proceso de limpieza de código, donde las librerías eliminadas fueron:

```
1 "@ant-design/compatible": "1.1.0",
2 "@babel/node": "7.16.8",
3 "react-ga": "2.4.1",
4 "react-loader-spinner": "6.0.0-0",
5 "react-moment": "1.0.0",
6 "remove-trailing-separator": "^1.0.1",
```

Y los archivos eliminados fueron:

```
1 src/actions/signUpActions.js
2 src/components/App.js
3 src/components/user/SignUpForm.js
4 src/containers/SessionSignUpPage.js
5 src/utils/mockdata.js
6 src/webpack-public-path.js
```

Luego de agregarse más features y hacerle más cambios a la aplicación, el 15 de mayo de 2023 se hizo otra iteración de la limpieza:

Donde, además de eliminar los archivos y librerías no utilizadas, también se borraron del `package.json`, muchos *scripts* que, o bien nunca fueron utilizados, ya que se agregaron en una primera instancia por el *boilerplate* de creación de la aplicación origen, o bien eran utilizados por los anteriores dueños, ya que los procesos de *running* y *deployment* se hacían de manera distinta en ese entonces.

Package.json scripts

Antes:

```
1 "scripts": {
2   "postinstall": "npx patch-package",
3   "start-message": "babel-node tools/startMessage.js",
4   "start": "webpack serve --open --config webpack.dev.js",
5   "open:src": "babel-node tools/srcServer.js",
6   "open:dist": "babel-node tools/distServer.js",
7   "clean-dist": "npm run remove-dist && mkdir dist",
8   "remove-dist": "rimraf ./dist",
9   "prebuild": "npm run clean-dist",
10  "build": "webpack --config webpack.prod.js",
11  "just-build": "babel-node tools/build.js",
12  "test": "jest",
13  "test:CI": "node --harmony_proxies node_modules/jest/bin/jest",
14  "test:cover": "npm run test -- --coverage",
15  "test:cover:CI": "npm run test:cover && cat ./coverage/lcov.info |
16  node_modules/coveralls/bin/coveralls.js",
17  "test:watch": "npm run test -- --watch",
18  "open:cover": "npm run test:cover && open coverage/lcov-report/index.html",
19  "deploy:staging": "npm run just-build && babel-node tools/deployS3 staging",
20  "deploy:production": "npm run just-build && babel-node tools/deployS3
21  production",
22  "deploy:gh": "npm run just-build && gh-pages -d dist",
23  "analyze-bundle": "babel-node ./tools/analyzeBundle.js"
24 }
```

Después:

```
1 "scripts": {
2   "postinstall": "npx patch-package",
3   "start": "webpack serve --open --config webpack.dev.js",
4   "clean-dist": "yarn run remove-dist && mkdir dist",
5   "remove-dist": "rimraf ./dist",
6   "prebuild": "yarn run clean-dist",
7   "build": "webpack --config webpack.prod.js"
8 }
```

Librerías eliminadas en esta iteración:

```
1 "babel-jest": "27.5.1",
2 "jest": "20.0.1",
```

Cabe aclarar que Jest es un *framework* de pruebas en JavaScript, utilizado para realizar pruebas unitarias en aplicaciones y proyectos de desarrollo de software.

La aplicación origen solo incluía un archivo de *spec* (prueba) y además, este archivo tenía una extensión “.ignore” en su nombre, lo que implica que nunca se ejecutaba. Por lo tanto, queda claro que las pruebas de Frontend no habían sido implementadas ni mantenidas.

En el caso de este Proyecto de Grado, agregar *tests* a la aplicación de Frontend tampoco era un objetivo del alcance, por lo que todo lo relacionado a *testing* fue removido.

Archivos eliminados en esta iteración de limpieza:

```
1 src/containers/SessionLoginPage.spec.js.ignore
2 tools/analyzeBundle.js
3 tools/build.js
4 tools/chalkConfig.js
5 tools/deployS3.js
6 tools/distServer.js
7 tools/fileMock.js
8 tools/nodeVersionCheck.js
9 tools/srcServer.js
10 tools/startMessage.js
```

En particular, toda la carpeta `tools/` fue eliminada, ya que eran archivos ejecutados al correr los antiguos comandos de `start` y `build` configurado por los dueños anteriores. Los comandos actuales no necesitan utilizar dichas configuraciones.

.3.5. Versión final del package.json

```
1 {
2   "name": "psp-code",
3   "version": "2.0",
4   "description": "Frontend app used as a tool for the 'Principios y Fundamentos
5     del Proceso Personal de Software' Udelar's course",
6   "engines": {
7     "npm": ">=14.19.3",
8     "node": ">=12.22.6"
9   },
10  "scripts": {
11    "postinstall": "npx patch-package",
12    "start": "webpack serve --open --config webpack.dev.js",
13    "clean-dist": "yarn run remove-dist && mkdir dist",
14    "remove-dist": "rimraf ./dist",
15    "prebuild": "yarn run clean-dist",
16    "build": "webpack --config webpack.prod.js"
17  },
18  "author": "Gustavo Audi y Lia Malvarez",
19  "license": "MIT",
20  "homepage": "https://www.pspcode.com/",
21  "dependencies": {
22    "@ant-design/icons": "4.7.0",
23    "antd": "4.20.0",
24    "classnames": "2.3.1",
25    "date-fns": "2.29.3",
26    "dotenv-webpack": "7.1.0",
27    "history": "5.3.0",
28    "html-webpack-plugin": "5.5.0",
```

```

28     "immutable": "3.8.1",
29     "isomorphic-fetch": "2.2.1",
30     "lodash": "4.17.4",
31     "moment": "2.24.0",
32     "prop-types": "15.7.2",
33     "react": "17.0.0",
34     "react-dom": "17.0.0",
35     "react-redux": "7.2.0",
36     "react-router": "6",
37     "react-router-dom": "6",
38     "react-router-redux": "4.0.8",
39     "redux": "3.6.0",
40     "redux-form": "8.1.0",
41     "redux-logger": "3.0.1",
42     "redux-react-session": "2.6.1",
43     "redux-thunk": "2.2.0",
44     "validate.js": "0.11.1",
45     "webpack-cli": "4.9.2",
46     "webpack-dev-server": "4.9.0",
47     "webpack-merge": "5.8.0"
48   },
49   "devDependencies": {
50     "@babel/cli": "7.17.6",
51     "@babel/core": "7.17.9",
52     "@babel/eslint-parser": "7.17.0",
53     "@babel/plugin-transform-react-constant-elements": "7.17.6",
54     "@babel/polyfill": "7.12.1",
55     "@babel/preset-env": "7.16.11",
56     "@babel/preset-react": "7.16.7",
57     "@babel/preset-stage-1": "7.8.3",
58     "@babel/register": "7.17.7",
59     "autoprefixer": "6.7.3",
60     "babel-loader": "8.2.5",
61     "babel-plugin-transform-react-remove-prop-types": "0.4.24",
62     "chalk": "1.1.3",
63     "connect-history-api-fallback": "1.3.0",
64     "coveralls": "2.11.16",
65     "css-loader": "0.28.1",
66     "enzyme": "3.0.0",
67     "eslint": "8.14.0",
68     "eslint-config-airbnb": "19.0.4",
69     "eslint-plugin-import": "2.26.0",
70     "eslint-plugin-jsx-ally": "6.5.1",
71     "eslint-plugin-react": "7.29.4",
72     "eslint-watch": "8.0.0",
73     "express": "4.15.2",
74     "file-loader": "5.0.0",
75     "gh-pages": "1.1.0",
76     "identity-obj-proxy": "3.0.0",
77     "json-loader": "0.5.4",
78     "mini-css-extract-plugin": "1.0.0",
79     "nock": "9.0.13",
80     "node-sass": "5.0.0",
81     "npm-run-all": "4.0.2",
82     "opn": "5.0.0",
83     "postcss-loader": "4.0.0",

```

```

84     "prompt": "1.0.0",
85     "react-hot-loader": "3.0.0-beta.7",
86     "react-test-renderer": "16.0.0",
87     "redux-immutable-state-invariant": "2.0.0",
88     "redux-mock-store": "1.2.1",
89     "replace": "0.3.0",
90     "rimraf": "2.5.4",
91     "sass-loader": "7.0.0",
92     "style-loader": "0.17.0",
93     "stylelint": "7.10.1",
94     "stylelint-config-standard": "16.0.0",
95     "stylelint-order": "0.2.2",
96     "url-loader": "0.5.8",
97     "webpack": "5.21.0",
98     "webpack-bundle-analyzer": "2.6.0",
99     "webpack-dev-middleware": "3.0.0",
100    "webpack-hot-middleware": "2.25.1",
101    "webpack-md5-hash": "0.0.5"
102  },
103  "keywords": [],
104  "repository": {
105    "type": "git",
106    "url": ""
107  }
108 }

```

.3.6. Identificación y propuesta de mejoras en UX/UI

A continuación se presenta el contenido de una lista de mejoras para el área de Frontend que fue desarrollada y presentada al inicio del proyecto.

Se encuentran los *bugs* y posibles mejoras organizados por área y clasificados por prioridad.

GENERAL:

ALTA - Cuando cambia de página la aplicación carga TODA la página, pudiendo cargar solo el contenido interno y usar el header y sidebar como layout.

MEDIA - Agregar favicon y logo a la app.

BAJA - NO responsive. Casi inutilizable en el celular.

MEDIA - Todas (o casi) todas las acciones no pueden deshacerse, ejemplo:

- asignar un proyecto
- submit proyecto terminado
- aprobar/reprobar un proyecto

MEDIA - Puede ser interesante dar la posibilidad de ver la aplicación en español.

LOGIN:

ALTA - Se rompe al querer cargar los proyectos cuando no tenía ningún proyecto ni estudiante asignado.

ALTA - Bug al restaurar la contraseña.

MEDIA - Demoró bastante en cargar el login (la primera vez).

MEDIA - Mail enviado a los alumnos al crear una cuenta, no debería ser en español?

VIEW DE WORKING (estudiante):

ALTA - “Instrucciones” (tanto en estudiante como en profesor) abre cualquier cosa (un xml).

ALTA - Los inputs de fecha no dejan editar en el input como texto, dejan poner cualquier letra

ALTA - Selector vacío cuando vas a registrar un error (Fix defect)

ALTA - Bug. Hora que aparece arriba a la derecha: porque se pone “:59” en vez de poner :00

MEDIA - Esos (nombre de profesor etc) tags parecen botones, no deberían.

MEDIA - Si vas a pedir que actualice tu perfil, marcar campos como obligatorios o al menos advertir q no quedó pronto.

MEDIA - El cartel de que la data se guarda automáticamente apenas se ve y no es estético

MEDIA - Entonces hay controles en el registro de defectos? ¿Cuáles son los criterios para limitar los tiempos?

MEDIA - Cuando registras un defecto: pusimos que empezamos el 20 (ayer) pero no nos deja registrar que terminamos de arreglar un defecto hoy. Bug?

BAJA - Mensaje que te advierte que hay que tener el perfil completo dice “profile” y “background” como sinónimos.

BAJA - No deja mantener letra apretada y escribir muchas (minor)

BAJA - Una i de info en la descripción no tiene sentido (debería ser un “ver más” o similar)

BAJA - Se necesitaría un hover cuando hay alguna sección disabled (que diga pq)

Manejo de fases

ALTA - Marcar en qué etapa del proyecto estoy (sidebar).

ALTA - Bug. Si pones que inyectas un defecto en la etapa actual y después cambias la etapa en la que estas se rompe porque no la encuentra.

MEDIA - La barra del tiempo sin fases parece un botón (así muchos)

MEDIA - Tabla del SUMMARY cuando estas Working queda medio llena medio no. (algunos 0 y otros vacío).

BAJA - El borrado de una fase no es estético.

BAJA - cuando se acumulan muchas fases aparece una barra de desplazamiento, no estética (caso poco probable).

BAJA - No deja borrar la primera, pero sin embargo el estado inicial era sin ninguna.

MENSAJERÍA:

MEDIA - No se manda con el apretando enter, debe apretarse el boton con el mouse

MEDIA - No se actualiza en tiempo real - simula un chat pero no lo es

MEDIA - La notificación queda como puesta aunque yo esté en (y haya recargado) la

página de mensajerías.

SUBIDA DE ARCHIVOS:

MEDIA - No se ve qué archivos subiste, lo tenés que descargar para verificar.

MEDIA - Entrando a files aparece un cartel “click or drag” pero no se puede clicar (cuando no hay fase iniciada, se le debería decir al usuario)

BAJA - El nombre autogenerado del zip no es nada mnemotécnico.

VIEW DE PROYECTO (profesor):

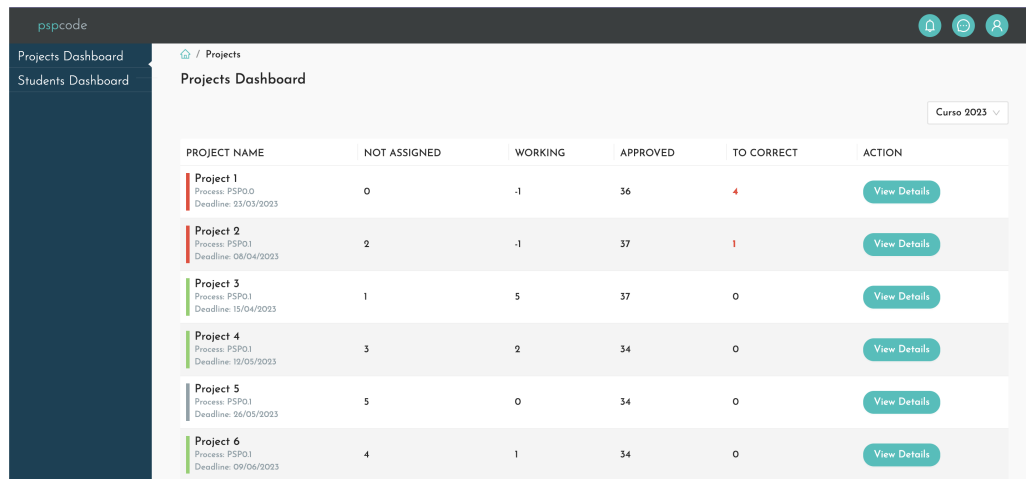
MEDIA - MISMA vista del estudiante: “Phase data is automatically saved” no debería estar.

MEDIA - No debería aparecer “start phase”

MEDIA - Cuando entras aparece la última fase por default en vez de la primera.

.3.7. Mejoras estéticas y funcionales de la interfaz de usuario

.3.7.1 Página de inicio - rol Profesor



PROJECT NAME	NOT ASSIGNED	WORKING	APPROVED	TO CORRECT	ACTION
Project 1 Process: PSP0.0 Deadline: 03/03/2023	0	-1	36	4	View Details
Project 2 Process: PSP01 Deadline: 08/04/2023	2	-1	37	1	View Details
Project 3 Process: PSP01 Deadline: 15/04/2023	1	5	37	0	View Details
Project 4 Process: PSP01 Deadline: 12/05/2023	3	2	34	0	View Details
Project 5 Process: PSP01 Deadline: 06/05/2023	5	0	34	0	View Details
Project 6 Process: PSP01 Deadline: 09/06/2023	4	1	34	0	View Details

Figura 3: Esta es una captura de la *Home Page* de la aplicación origen.

Se pueden notar en esta imagen varios puntos a mejorar desde la perspectiva UI:

- La paleta de colores no está del todo definida. Se utiliza el verde agua en el *logo*, y en la mayoría de los botones. Sin embargo en la barra lateral, en el efecto de *hover* en la tabla, y en los distintos enlaces a lo largo de la aplicación (que no se llegan a apreciar en esta imagen), permanecen en el azul por defecto de la librería AntD.
- Las opciones de la barra lateral se pueden ver super apretadas y sobre el inicio de la misma, pudiendo estar mejor distribuidas

- El encabezado (o *header*) puede verse muy fino para las tendencias actuales de páginas web, haciendo que el *logo* esté muy chico y las opciones sobre la derecha muy apretadas verticalmente.
- Algo similar pasa con la anidación de navegación (o *breadcrumb*) y el título de la página web, se ve todo muy estrecho sobre la parte de arriba de la pantalla.
- Se agregaron sombras sutiles a los distintos contenedores para dar efecto de profundidad y quitar la sensación plana actual.
- Ahora cada fila de la tabla es presionable en cualquier lugar de la misma (no solo en el botón). Si bien el botón de *Action* indica cuál es la acción al presionar la fila, podemos ejecutar dicha acción desde, por ejemplo, el nombre del proyecto. Haciendo la funcionalidad de la tabla más sencilla e intuitiva.
- El *logo* fue levemente modificado, dejando en mayúscula la sigla “PSP” y en minúscula la palabra “code”.
- Se hizo un cambio en la fuente general de la aplicación, la original era demasiado infantil o lúdica, aparte de ser demasiado gruesa en casos donde no corresponde. Con respecto a esta punto, cabe aclarar que a continuación puede haber capturas de la aplicación origen tomadas luego de que parte del cambio de fuente haya sido efectuado, ya que fue uno de los primeros cambios hechos.
- Otra mejora general que se hizo fue cambiar el cursor para todos los elementos presionables (botones, links, entre otros). En la aplicación origen, cuando el cursor pasaba por encima de estos elementos se mostraba como la flecha clásica por defecto, siendo confuso y poco indicativo para el usuario.

PROJECT NAME	NOT ASSIGNED	WORKING	APPROVED	TO CORRECT	ACTION
Project 1 Process: PSP010 Deadline: 23/03/2023	0	-1	36	4	View Details
Project 2 Process: PSP01 Deadline: 06/04/2023	2	-1	37	1	View Details
Project 3 Process: PSP01 Deadline: 15/04/2023	1	5	37	0	View Details
Project 4 Process: PSP01 Deadline: 12/05/2023	3	2	34	0	View Details
Project 5 Process: PSP01 Deadline: 26/05/2023	5	0	34	0	View Details

Figura 4: De esta manera se ve actualmente la página de inicio para un usuario de rol Profesor.

3.7.2 Mejoras de diseño en las tablas de la aplicación

Por otro lado, además de los cambios ya especificados para la *Home Page*, se realizaron numerosas actualizaciones en las tablas de la aplicación. Se modificó el componente para indicar el estado de carga de las tablas.

Se puede ver el antes y después a continuación.

Antes

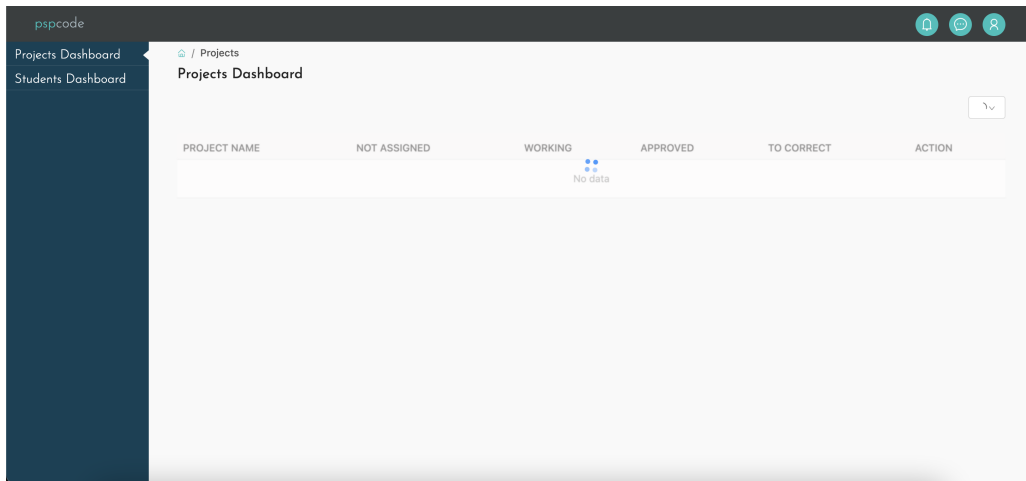


Figura 5: Así se veía la tabla mientras cargaba su información.

Puede verse que el *loader spinner* no está bien centrado verticalmente con la tabla, y la misma tiene una altura menor a la que debería. A su vez, el mensaje “No Data” como parte del componente de carga, es confuso. En el nuevo diseño puede notarse que el *empty state* o estado vacío, se encuentra por detrás del componente de carga.

Después

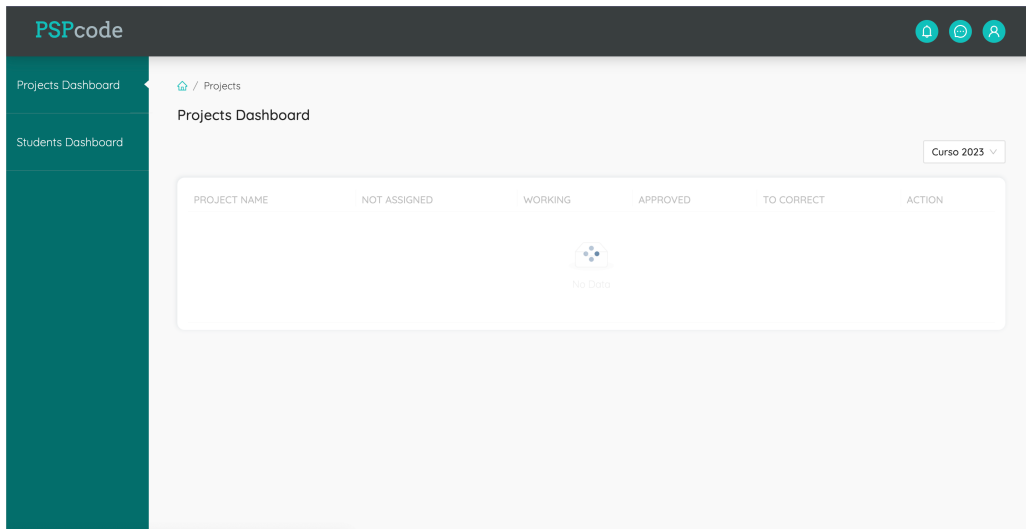


Figura 6: La situación actual de la tabla con estos aspectos solucionados.

También se hicieron cambios en el componente de filtros de la tabla, unificando el color azul Francia y los botones verde agua en un azul más serio y sutil:

Antes

STUDENT NAME	TUTOR	CURRENT PROJECT	STATUS	ACTION
Agustin Ruiz view activity	Silvana	Project 8	Approved	
Romina Sosa view activity	Silvana	Project 8	Approved	
Silvana Freire view activity	Silvana	Project 1	Dued	<button>Poke</button>
Enrique Castro view activity	Silvana	Project 8	Approved	
Darwin Araujo view activity	Silvana	Project 8	Approved	
Micaela Lopez view activity	Silvana	Project 8	Approved	
Victor Regueira view activity	Silvana	Project 4	Dued	<button>Poke</button>
Juan Sandin view activity	Silvana	Project 8	Approved	

Figura 7: Así se veían los filtros de profesores en la tabla de estudiantes.

Después

PROJECT NAME	ASSIGNED DATE	DEADLINE	STATUS
Project 1 Process: PSP00	19/03/2023	23/03/2023	Approved
Project 2 Process: PSP01	24/03/2023	08/04/2023	Being Corrected
Project 3 Process: PSP01	27/03/2023	15/04/2023	Approved
Project 8 Process: PSP01	03/07/2023	07/07/2023	Being Corrected
Project 6 Process: PSP01	08/07/2023	09/06/2023	Working

Figura 8: La situación actual de los filtros de estado de proyecto para los proyectos de un estudiante.

Luego, el ordenamiento (o *sorting*) de las tablas no funcionaba correctamente en la mayoría de las columnas. Puede verse a continuación la diferencia entre el antes y después para la columna de estado de un proyecto.

Antes

STATUS	ACTION
● Approved	
● Approved	
● Dued	Poke
● Approved	
● Approved	

Figura 9: Puede verse el filtro en orden alfabético ascendente activado, sin embargo las filas no están ordenadas de esta forma.

STATUS	ACTION
● Approved	View Project
● Approved	View Project
● Correction Pending	Correct
● Pending	View Projects
● Pending	View Projects

Figura 10: Puede verse el filtro en orden alfabético ascendente activado y funcionando correctamente.

Se visualiza también en la imagen, la inclusión de un botón (*View Project*) para los proyectos en estado *Approved*, ya que previamente no había acción asignada.

.3.7.3 Pantalla My Projects - rol Estudiante

A continuación se muestra el antes y después de la pantalla de inicio para un estudiante: *My Projects*. En esta comparación pueden verse los cambios generales para tablas y de *layout* previamente descritos. En particular un cambio importante en esta pantalla fue la limitación en el ancho de la tabla, ya que para pantallas grandes, al no estar la barra lateral del profesor, la tabla tomaba demasiado ancha, viéndose muy estirada.

Antes

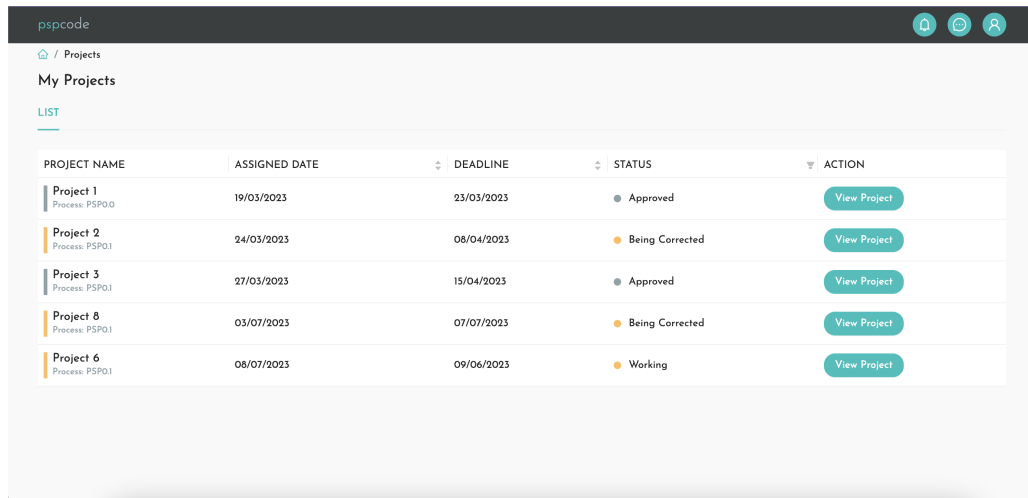


Figura 11: Página My Projects en la aplicación origen.

Después

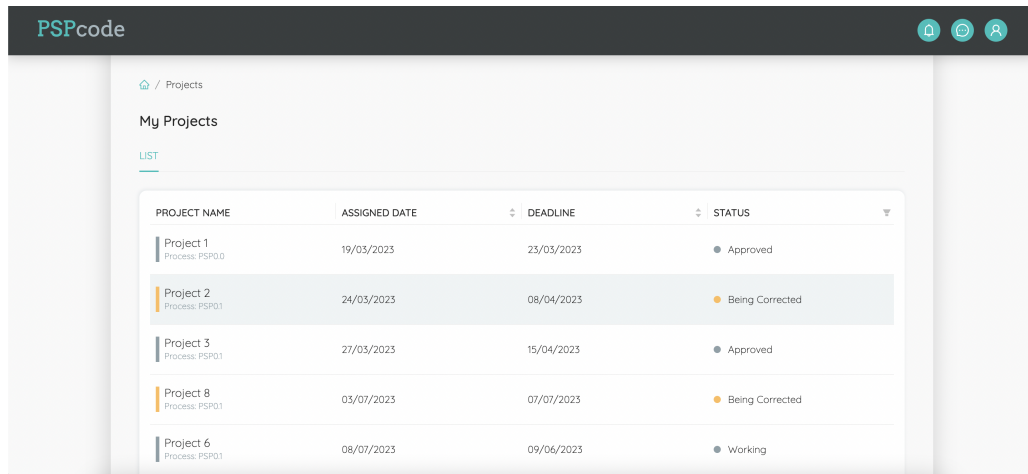


Figura 12: Página My Projects en la nueva aplicación, con el contenido principal reducido a una porción de la pantalla.

.3.7.4 Pantalla en blanco al navegar entre páginas

En la aplicación origen, al cambiar de rutas (navegar entre páginas) se utiliza un *loader* en forma de barra, y el *layout* no es mantenido durante la navegación. Técnicamente, esto quiere decir que el encabezado y la barra lateral no están envolviendo el resto de las páginas, por lo tanto vuelven a cargarse cada vez que la aplicación *rutea*, dando un efecto de parpadeo y posible *crash* de la aplicación al usuario, desconcertándolo, más aún si la carga es lenta.

.3.7.5 Pantalla de mensajes

En esta pantalla de la aplicación se hicieron numerosos cambios. La aplicación ofrece la posibilidad de comunicarse con otro usuario, en el caso de que el usuario *logueado* sea de rol Estudiante, puede comunicarse con su tutor, en el caso de que el usuario *logueado* sea de rol Profesor, puede comunicarse con todos sus estudiantes.

Los mensajes se dividen por versión de proyecto, es decir, cada versión de un nuevo proyecto que se le asigna al estudiante, posee su propio hilo de mensajes con el tutor.

En primer lugar: el componente no diferenciaba su interfaz según el usuario activo. Teniendo en cuenta que el *chat* siempre se da entre usuarios con distinto rol, sucedía que los mensajes del usuario con rol Profesor, siempre se veían en azul y del lado derecho de la pantalla, sin importar si el usuario activo era un estudiante. Esto iba en contra del estándar de las aplicaciones de mensajería, las cuales muestran el texto escrito por el usuario actual del lado derecho, siendo incómodo para los usuarios de rol Estudiante.

Por otro lado, otro punto de mejora muy importante, fue el desplazamiento vertical de la pantalla. En la aplicación origen, este desplazamiento se daba a lo largo de toda la pantalla, perdiendo de vista el encabezado, y el contenido de la barra lateral izquierda (el estado del proyecto).

Pueden verse ambos puntos de cambio mencionados en la siguiente comparación:

Antes

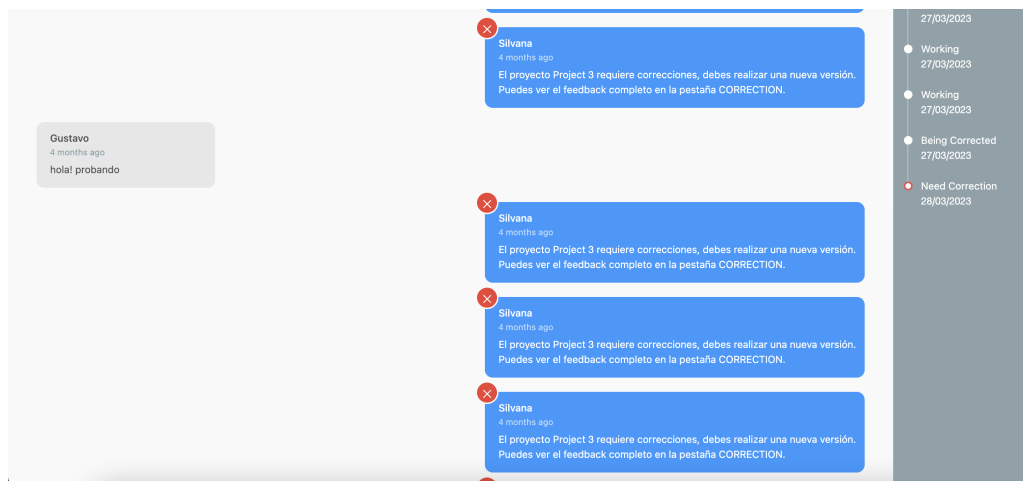


Figura 15: Puede verse el componente de mensajerías, es mostrado de esta forma sin importar si el usuario es Estudiante o Profesor. A su vez, esta es una captura de la pantalla luego de hacer *scroll* hacia abajo

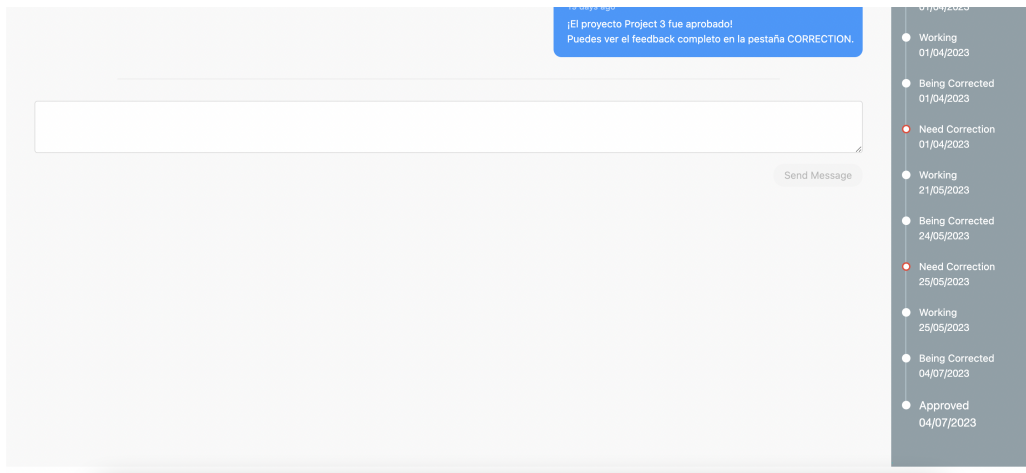


Figura 16: En esta imagen puede verse todo el espacio libre de pantalla provocado por la falta de dos desplazamientos independientes (uno para el chat y otro para la barra de estados de proyecto)

Después

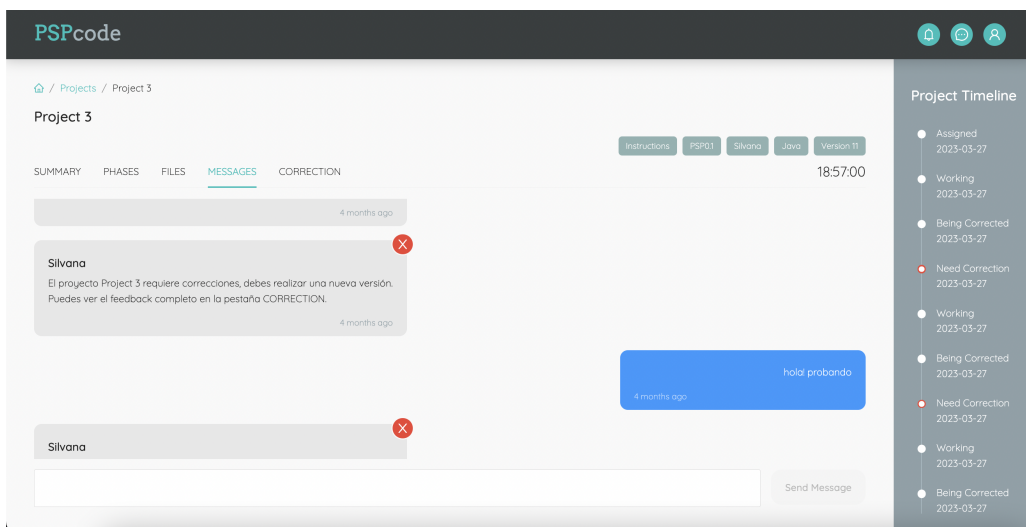


Figura 17: Pantalla de *chat* de la aplicación actual.

En la Figura 17 puede verse que los mensajes del usuario *logueado* (un estudiante en este caso) se muestran del lado derecho, en azul y sin el nombre del usuario. A su vez, el desplazamiento es independiente para el chat y la barra lateral. El desplazamiento de los mensajes es interno a su ventana, el input de escritura y todo el resto del *layout* siempre permanece visible. Cuando un nuevo mensaje es enviado, la ventana se *auto-scrollea* hasta el final con una animación suave, para visualizar los mensajes más actuales.

.3.7.6 Vista de un Proyecto

La vista de un proyecto es la interfaz más compleja de la aplicación, a continuación se describirán muchos de los cambios de diseño efectuados en la misma.

Así se ve el antes y después de la pestaña de fases de un proyecto, vistas desde un usuario Profesor:

Antes

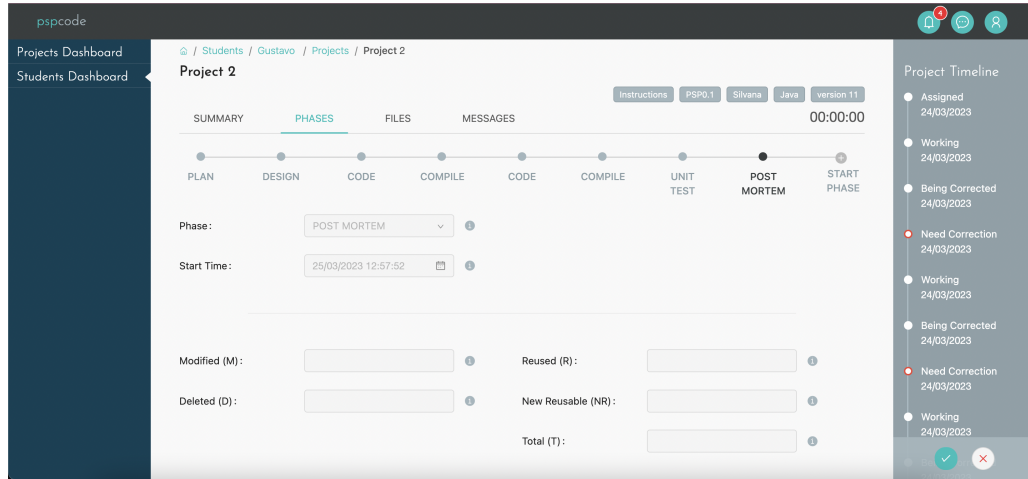


Figura 18: Las fases de un proyecto en la aplicación origen.

Después

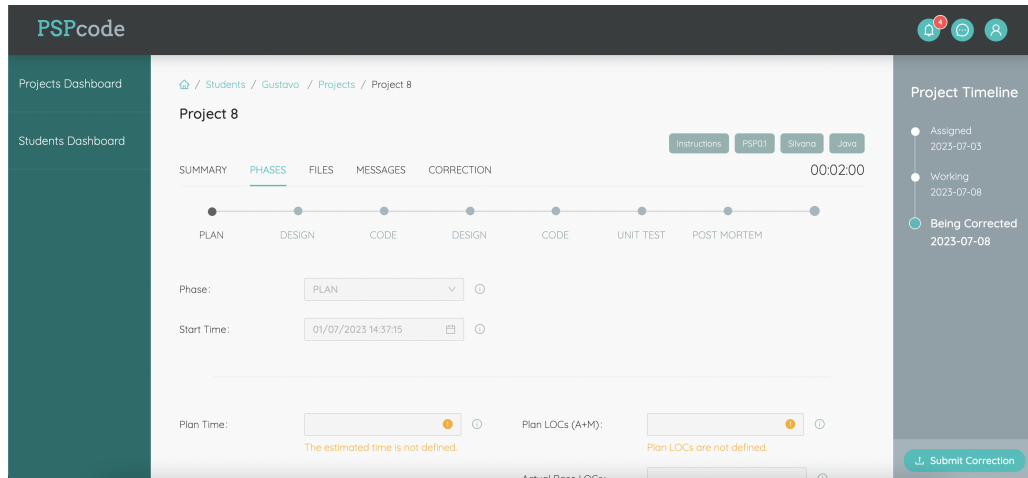


Figura 19: Las fases de un proyecto en la nueva aplicación.

Los cambios son, en su mayoría, estéticos. La manera de utilizar la funcionalidad se mantuvo. Los principales cambios fueron:

- La fuente de texto utilizada, la nueva es más delicada y moderna.
- Los colores utilizados en las *tags* de información (Instructions, PSP0.1, etc.), adecuándose mejor a la paleta de colores del resto de la aplicación.
- Se corrigió el espacio entre el cabezal y el *breadcrumb* (historial de navegación).

- En la versión actual de PSPcode se indica el estado de un proyecto con un punto más grande del que había y de color verde agua en la barra lateral.
- Ya no aparece el símbolo de agregar fase para el profesor, en la versión antigua de PSPcode se seguía viendo dicho símbolo con el botón correspondiente deshabilitado.
- Se eliminaron los botones de aprobar/reprobar proyecto de la barra lateral derecha, ahora hay un boton “Submit Correction” para el profesor, con el cual pueden enviar su corrección detallada. Esta es una nueva *feature* de la aplicación descrita en la Sección 5.3.

Otras correcciones que se hicieron en esta pantalla tienen que ver con el desplazamiento vertical. Al igual que con la pantalla de mensajes, sucedía que si el historial de estados del proyecto era más largo que la información de las fases, el usuario podía ver el contenido de la barra lateral desplazándose hacia abajo y dejando oculto el cabezal y toda la información de la izquierda.

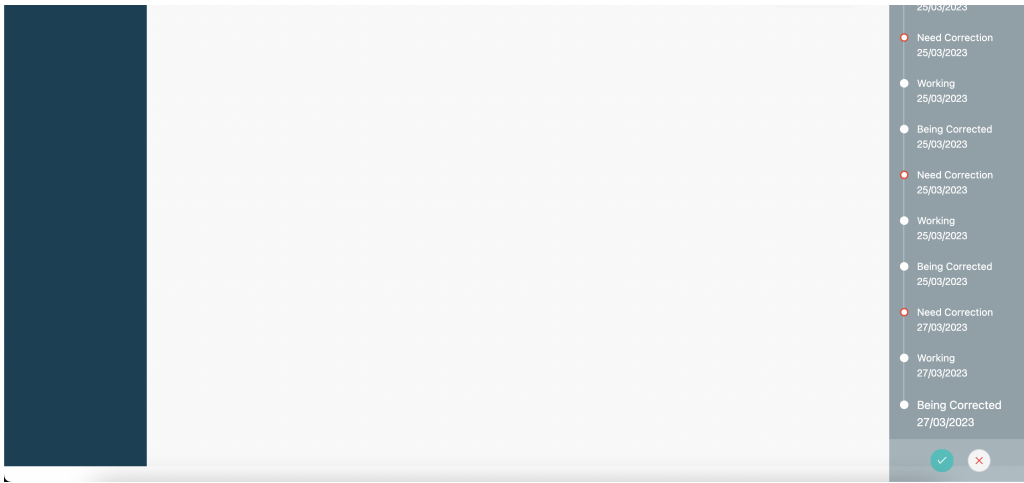


Figura 20: Desplazamiento vertical para la barra lateral derecha en la vista de un proyecto.

Al igual que en la pantalla de mensajes, esto se solucionó independizando el desplazamiento de la barra lateral y de la información del proyecto.

Por otro lado, para la vista del proyecto desde un usuario con rol Estudiante, dos modificaciones destacables fueron:

- En la versión origen, se informaba al usuario que los cambios en un proyecto eran guardados automáticamente mediante un texto en la parte inferior de la pantalla. Esto llevaba a que el estudiante probablemente nunca recibiera el mensaje. Además de ser poco atractivo visualmente. En su lugar, se agregó un recuadro informativo (que el estudiante puede cerrar si le molesta) al inicio de la pantalla de fase.
- En la misma zona inferior, había un mensaje innecesario, y un *link* naranja para efectuar la eliminación de la fase actual. Éste fue reemplazado por un botón con un texto claro *Delete Phase* y un ícono de papelera.
- A su vez, se aumentó el espacio entre este botón y el final de la pantalla, ya que en la aplicación origen estaba todo sobre el borde inferior del navegador, lo que empobrecía la estética.

Antes

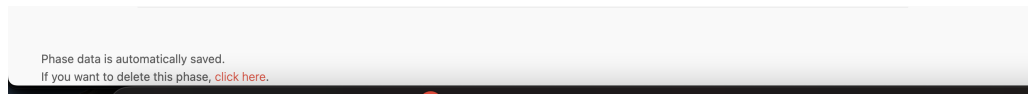


Figura 21: Mensajes explicativos en la aplicación origen, indicando el guardado de información y el borrado de la fase.

Después

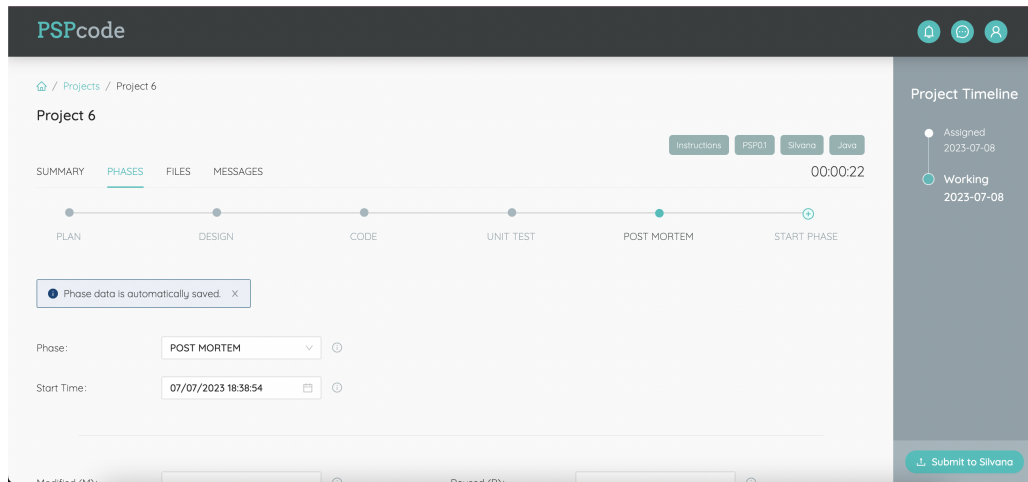


Figura 22: Cartel informativo del modo de guardado de los cambios y vista general de la fase para un Estudiante.

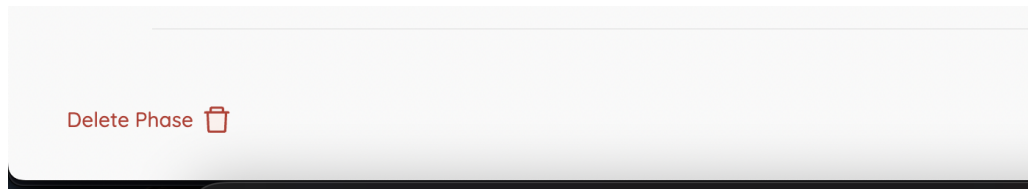


Figura 23: Botón de borrado de fase en la zona inferior de la pantalla.

3.7.7 Notificaciones

En este módulo de la aplicación también hubo arreglos con respecto al desplazamiento. En la versión origen, la ventana de notificaciones (tanto específicas de mensajes como generales) se expandía verticalmente al cargar más, pudiendo tener una altura infinita y hacer muy engorroso el proceso de *scrolllear* a lo largo de toda la pantalla. El cambio que se hizo fue mantener la ventana con una altura fija permitiendo al usuario que deslice las notificaciones dentro de ella.

Antes

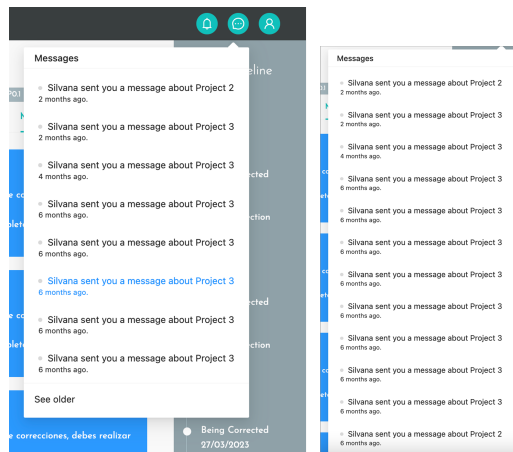


Figura 24: Notificaciones de mensajes antes y después de presionar el botón *See older* en la aplicación origen, la ventana de notificaciones continua luego del alcance de la captura.

Después

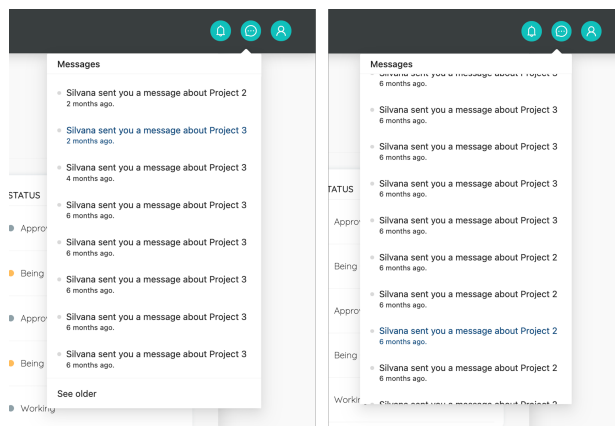


Figura 25: Notificaciones de mensajes antes y después de presionar el botón *See older* en la aplicación actualizada, puede notarse que el alto del contenedor no cambia.

3.7.8 Incorporación de favicon

Alineado con el cambio en el *logo*, se diseñó y agregó un *favicon* a la página, y se le hizo un cambio de nombre a la misma, dando un aspecto más estético y profesional.



Figura 26: Comparación del antes y después de la pestaña original y la resultante luego de este cambio.

De esta manera se concluye la descripción detallada de las modificaciones efectuadas en la interfaz de usuario (UI). Es importante destacar que aunque no se enumeraron explícitamente todos los cambios, se aplicaron ajustes similares a los previamente descritos en todos los módulos de la aplicación.

4. Grading original

A continuación se presenta la *grading* utilizada por las profesoras hasta 2022 inclusive para evaluar a los estudiantes. La misma consiste de un grupo de criterios para los cuales las docentes escriben un comentario a modo de *feedback*.

Sumado a esto, en el documento se incluyen los siguientes datos:

- Nombre del Estudiante
- Nombre del Profesor
- Nombre del Proyecto (ejemplo: “Programa 1”)
- Fecha de inicio del Proyecto
- Fecha de fin del Proyecto
- Tiempo total de interrupciones
- Casilla de aprobación (o reprobación) del Proyecto.

4.1. Criterios escritos y clasificados como se encuentran en la *grading* original:

Programa y Resultado de Test

Criterio	Comentario
El programa resulta ser funcional	-
Todos los test requeridos fueron ejecutados	-
La salida actual es correcta para cada uno de los test	-
El código es compatible con el estándar de codificación	-
Los resultados de los test son consistentes con el estándar de conteo.	-

Cuadro 1: Criterios de la *grading* de evaluación de categoría Programa y Resultado de *Test*.

Log de Tiempo

Criterio	Comentario
Los datos del tiempo se registran en todas las fases del proceso	-
Las etapas del proceso siguen una secuencia adecuada	-
Los tiempos son registrados en los pasos correctos del proceso	-
Los tiempos de las interrupciones son registrados de forma apropiada	-
Los datos del tiempo son completos y razonables	-
Los datos fueron registrados a medida se realizaba el trabajo.	-

Cuadro 2: Criterios de la *grading* de evaluación de categoría *Log* de Defectos.

Log de Defectos

Criterio	Comentario
Todos los defectos tienen la información de registro solicitada	-
Los defectos fueron inyectados antes de ser removidos	-
Todos los defectos tienen tiempo de corrección	-
Los defectos inyectados en compilación y test tienen “fix defect” asociado	-
Los defectos son descritos adecuadamente.	-
El tipo de defecto es consistente con la descripción	-
El tipo de defecto es consistente con la fase de inyección	-
El tipo de defecto es asignado consistentemente	-

Cuadro 3: Criterios de la *grading* de evaluación de categoría *Log de Defectos*.

Formulario PIP

Criterio	Comentario
El formulario PIP está completo	-
Los registros del PIP muestran perspicacia y reflexión	-

Cuadro 4: Criterios de la *grading* de evaluación de categoría Formulario PIP.

Planning Summary

Criterio	Comentario
El tiempo total planificado fue ingresado correctamente	-
Los tamaños planificado y actual están ingresados correctamente	-

Cuadro 5: Criterios de la *grading* de evaluación de categoría *Planning Summary*.

Chequeo de Consistencia

Criterio	Comentario
Los defectos removidos son consistentes con los tiempos en las fases de compilación y test	-
El total del tiempo de corrección en compilación es menor que el tiempo total dedicado a la compilación	-
El total de tiempo de corrección en test es menor que el tiempo dedicado al test	-
Los datos de los defectos y las fases son consistentes con el Log de tiempo	-
Planning summary es consistente con el log de tiempos	-
Planning summary es consistente con el log de defectos	-

Cuadro 6: Criterios de la *grading* de evaluación de categoría Chequeo de Consistencia.

General

Criterio	Comentario
Se siguió el proceso definido	-
Los datos recogidos del proceso son completos, precisos y coherentes	-
El estudiante hizo su propio trabajo	-

Cuadro 7: Criterios de la *grading* de evaluación de categoría General.

.5. Backend

.5.1. Manual de uso del Admin para el flujo de Project Feedback

En primer lugar, se accede al Admin a través de la dirección <https://admin.pspcode.com/> y utilizando las credenciales de un usuario con rol Docente. En este punto se listan todos los modelos que pueden visualizarse y editarse:

Site Administration

Dashboard

Dashboard

Model name	Last created	Records	
Corrections		16935	⌵ + 📄
Courses	8 months ago	1	⌵ + 📄
Criteria		31	⌵ + 📄
Phases		6	⌵ + 📄
Processes	8 months ago	2	⌵ + 📄
Professors	8 months ago	3	⌵ + 📄
Projects	8 months ago	8	⌵ + 📄
Project feedbacks		558	⌵ + 📄
Sections		7	⌵ + 📄
Students	8 months ago	39	⌵ + 📄

Figura 27: Página de inicio del Admin

A continuación se detallan algunos escenarios posibles

5.1.1 Visualizar y modificar Criterios

Se accede al modelo **Criteria**

List of Criteria

Dashboard / Criteria

List + Add new Export Add filter Selected items

Filter Refresh x Export found Criteria

<input type="checkbox"/>	Id	Description	Section	Order	Only in psp01	
<input type="checkbox"/>	31	El estudiante hizo su propio trabajo.	General	3	x	📄 ✎
<input type="checkbox"/>	30	Los datos recogidos del proceso son comp...	General	2	x	📄 ✎
<input type="checkbox"/>	29	Se siguió el proceso definido.	General	1	x	📄 ✎
<input type="checkbox"/>	28	Planning summary es consistente con el lo...	Chequeo de Consistencia	5	x	📄 ✎
<input type="checkbox"/>	27	Planning summary es consistente con el lo...	Chequeo de Consistencia	4	x	📄 ✎
<input type="checkbox"/>	26	Los datos de los defectos y las fases son c...	Chequeo de Consistencia	3	x	📄 ✎
<input type="checkbox"/>	25	El total del tiempo de corrección en cada f...	Chequeo de Consistencia	2	x	📄 ✎
<input type="checkbox"/>	24	Los defectos removidos son consistentes c...	Chequeo de Consistencia	1	x	📄 ✎
<input type="checkbox"/>	23	Los tamaños planificado y actual están ing...	Planning Summary	2	✓	📄 ✎
<input type="checkbox"/>	22	El tiempo total planificado fue ingresado c...	Planning Summary	1	x	📄 ✎
<input type="checkbox"/>	21	Los registros del PID muestran variación	Formulario PID	1	✓	📄 ✎

Figura 28: Lista de Criterios

En esta página se puede ver cada uno de los criterios. Luego se puede, o bien ver la información (botón de **i**) particular de uno en específico, o bien editarla (botón del lápiz). Por ejemplo se muestra el caso donde se visualiza la información del criterio que tiene *id* 31:

Details for Criterion 'Criterion #31'

[Dashboard](#) / [Criteria](#) / [Criterion #31](#)

[Show](#) [Edit](#)

Basic info

Description
El estudiante hizo su propio trabajo.

Section
[General](#)

Order
3

Algorithm
-

Algorithm type
-

Only in psp01

Figura 29: Información del Criterio 31

En esta pantalla se puede ver información de alguno de los campos relevantes, como la descripción, la sección a la que pertenece, si tiene algún algoritmo asociado y si pertenece sólo al PSP0.1.

Luego para editar la información de este mismo criterio, se puede acceder al botón lápiz desde la página de visualización, o desde la lista de criterios, desplegando la siguiente pantalla:

Edit Criterion 'Criterion #31'

[Dashboard](#) / [Criteria](#) / [Criterion #31](#) / Edit

[Show](#) [Edit](#)

Description
Required.

Section [+ Add a new Section](#) [Edit this Section](#)
Required.

Order
Required.

Algorithm
Optional.

Algorithm type
Optional.

Only in psp01

[Save](#) [Save and add another](#) [Save and edit](#) [Cancel](#)

Figura 30: Editar Criterio 31

En esta pantalla se pueden editar los campos nombrados anteriormente. La manera de interactuar es intuitiva y sencilla.

.5.2. Edición de Fase

A continuación se ve el formulario de edición de fase, en este caso se está editando la fase Post Mortem.

The screenshot shows a web interface for editing a phase. The title is 'Edit Phase 'POST MORTEM''. Below the title is a breadcrumb trail: 'Dashboard / Phases / POST.MORTEM / Edit'. There are two tabs: 'Show' (selected) and 'Edit'. The form contains the following fields and controls:

- Name:** A text input field containing 'POST MORTEM'. Below it is the label 'Required.'.
- Description:** A text area containing 'This phase if estimations were correct'. Below it is the label 'Optional.'.
- Order:** A text input field containing '6'. Below it is the text 'Suggested order that phases will be displayed for students'.
- First:** A control with a green checkmark, a red 'x', and a minus sign.
- Last:** A control with a green checkmark, a red 'x', and a minus sign.
- Buttons:** A row of four buttons: 'Save' (blue), 'Save and add another' (light blue), 'Save and edit' (light blue), and 'Cancel' (grey).

Figura 31: Editar fase Post Mortem.

.5.3. Base de datos

A continuación se muestra el archivo semilla utilizado para cargar la base de datos productiva:

```
1 # Clean all data bases
2 Phase.destroy_all
3 Project.destroy_all
4 User.destroy_all
5 Professor.destroy_all
6 Status.destroy_all
7 MessageNotification.destroy_all
8 EventNotification.destroy_all
9 Course.destroy_all
10 CourseProjectInstance.destroy_all
11 AssignedProject.destroy_all
12 Message.destroy_all
13 ProjectDelivery.destroy_all
14 PhaseInstance.destroy_all
15 Defect.destroy_all
16 ProfessorCourse.destroy_all
17 PspProcess.destroy_all
18 Section.destroy_all
19 Criterion.destroy_all
20 ProjectFeedback.destroy_all
```

```

21 Correction.destroy_all
22
23 # Creating phases
24 Phase.create! name: 'PLAN', description: 'This phase is to understand the
      exercise', order: 1,
25           first: true, last: false
26
27 Phase.create! name: 'DESIGN', description: 'This phase is to think how to
      solve the exercise',
28           order: 2, first: false, last: false
29
30 Phase.create! name: 'CODE', description: 'This phase is to start coding',
      order: 3, first: false,
31           last: false
32
33 Phase.create! name: 'COMPILE', description: 'This phase is to compile the
      code (if needed)',
34           order: 4, first: false, last: false
35
36 Phase.create! name: 'UNIT TEST', description: 'This phase is to test the
      code', order: 5,
37           first: false, last: false
38
39 Phase.create! name: 'POST MORTEM', description: 'This phase if
      estimations were correct', order: 6,
40           first: false, last: true
41
42 # Creating Processes
43 PspProcess.create! name: 'PSP0.0', has_plan_time: true, has_plan_loc:
      false, has_pip: false
44
45 PspProcess.create! name: 'PSP0.1', has_plan_time: true, has_plan_loc:
      true, has_pip: true
46
47 # Assign phases to processes
48 PspProcess.all.each do |process|
49   Phase.all.each do |phase|
50     process.phases << phase
51   end
52 end
53
54 # Creating professors
55 Professor.create! first_name: 'admin',
56                 last_name: 'admin',
57                 email: 'admin@example.com',
58                 password: 'admin'
59
60 Professor.create! first_name: 'Silvana',
61                 last_name: 'Moreno',
62                 email: 'silvana@example.com',
63                 password: 'admin'
64

```

```

65 Professor.create! first_name: 'Leticia',
66                   last_name: 'Perez',
67                   email: 'leticia@example.com',
68                   password: 'admin'
69
70 # Creating Sections and Criterions
71 Section.create! name: 'Programa y Resultado de Test'
72 Criterion.create! description: 'El programa resulta ser funcional.',
73                       section: Section.find_by_name('Programa y Resultado de
74                       Test'),
75                       order: 1
76 Criterion.create! description: 'Todos los test requeridos fueron
77                       ejecutados.',
78                       section: Section.find_by_name('Programa y Resultado de
79                       Test'),
80                       order: 2
81 Criterion.create! description: 'La salida actual es correcta para cada
82                       uno de los test.',
83                       section: Section.find_by_name('Programa y Resultado de
84                       Test'),
85                       order: 3, only_in_psp01: true
86 Criterion.create! description: 'Los resultados de los test son
87                       consistentes con el estándar de conteo.',
88                       section: Section.find_by_name('Programa y Resultado de
89                       Test'),
90                       order: 4, only_in_psp01: true
91 Criterion.create! description: 'Los resultados de los test son
92                       consistentes con el estándar de conteo.',
93                       section: Section.find_by_name('Programa y Resultado de
94                       Test'),
95                       order: 5
96
97 Section.create! name: 'Log de Tiempo'
98 Criterion.create! description: 'Los datos del tiempo se registran en
99                       todas las fases del proceso.',
100                      section: Section.find_by_name('Log de Tiempo'),
101                      order: 1
102 Criterion.create! description: 'Las etapas del proceso siguen una
103                       secuencia adecuada.',
104                      section: Section.find_by_name('Log de Tiempo'),
105                      order: 2
106 Criterion.create! description: 'Los tiempos son registrados en los pasos
107                       correctos del proceso.',
108                      section: Section.find_by_name('Log de Tiempo'),
109                      order: 3, algorithm: 5, algorithm_type: 0
110 Criterion.create! description: 'Los tiempos de las interrupciones son
111                       registrados de forma apropiada.',
112                      section: Section.find_by_name('Log de Tiempo'),
113                      order: 4, algorithm: 2, algorithm_type: 0
114 Criterion.create! description: 'Los datos del tiempo son completos y
115                       razonables.',
116                      section: Section.find_by_name('Log de Tiempo'),

```

```

103         order: 5, algorithm: 0, algorithm_type: 0
104 Criterion.create! description: 'Los datos fueron registrados a medida se
      realiza el trabajo.',
105         section: Section.find_by_name('Log de Tiempo'),
106         order: 6
107
108 Section.create! name: 'Log de Defectos'
109 Criterion.create! description: 'Todos los defectos tienen la información
      de registro solicitada.',
110         section: Section.find_by_name('Log de Defectos'),
111         order: 1
112 Criterion.create! description: 'Los defectos fueron inyectados antes de
      ser removidos.',
113         section: Section.find_by_name('Log de Defectos'),
114         order: 2, algorithm: 7, algorithm_type: 1
115 Criterion.create! description: 'Todos los defectos tienen tiempo de
      corrección.',
116         section: Section.find_by_name('Log de Defectos'),
117         order: 3
118 Criterion.create! description: 'Los defectos inyectados en compilación y
      test tienen fix defect asociado.',
119         section: Section.find_by_name('Log de Defectos'),
120         order: 4
121 Criterion.create! description: 'Los defectos son descritos adecuadamente.
      ',
122         section: Section.find_by_name('Log de Defectos'),
123         order: 5
124 Criterion.create! description: 'El tipo de defecto es consistente con la
      descripción.',
125         section: Section.find_by_name('Log de Defectos'),
126         order: 6
127 Criterion.create! description: 'El tipo de defecto es consistente con la
      fase de inyección.',
128         section: Section.find_by_name('Log de Defectos'),
129         order: 7
130 Criterion.create! description: 'El tipo de defecto es asignado
      consistentemente.',
131         section: Section.find_by_name('Log de Defectos'),
132         order: 8
133
134 Section.create! name: 'Formulario PIP'
135 Criterion.create! description: 'El formulario PIP está completo.',
136         section: Section.find_by_name('Formulario PIP'),
137         order: 1, only_in_psp01: true
138 Criterion.create! description: 'Los registros del PIP muestran
      perspicacia y reflexión.',
139         section: Section.find_by_name('Formulario PIP'),
140         order: 2, only_in_psp01: true
141
142 Section.create! name: 'Planning Summary'
143 Criterion.create! description: 'El tiempo total planificado fue ingresado
      correctamente.',

```

```

144         section: Section.find_by_name('Planning Summary'),
145         order: 1, algorithm: 3, algorithm_type: 0
146 Criterion.create! description: 'Los tamaños planificado y actual están
      ingresos correctamente.',
147         section: Section.find_by_name('Planning Summary'),
148         order: 2, algorithm: 4, only_in_psp01: true,
      algorithm_type: 0
149
150 Section.create! name: 'Chequeo de Consistencia'
151 Criterion.create! description: 'Los defectos removidos son consistentes
      con los tiempos en las fases de compilación y test.',
152         section: Section.find_by_name('Chequeo de Consistencia'
      ),
153         order: 1, algorithm: 6, algorithm_type: 1
154 Criterion.create! description: 'El total del tiempo de corrección en cada
      fase es menor que el tiempo total dedicado a la misma.',
155         section: Section.find_by_name('Chequeo de Consistencia'
      ),
156         order: 2, algorithm: 1, algorithm_type: 0
157 Criterion.create! description: 'Los datos de los defectos y las fases son
      consistentes con el log de tiempos.',
158         section: Section.find_by_name('Chequeo de Consistencia'
      ),
159         order: 3
160 Criterion.create! description: 'Planning summary es consistente con el
      log de tiempos.',
161         section: Section.find_by_name('Chequeo de Consistencia'
      ),
162         order: 4
163 Criterion.create! description: 'Planning summary es consistente con el
      log de defectos.',
164         section: Section.find_by_name('Chequeo de Consistencia'
      ),
165         order: 5
166
167 Section.create! name: 'General'
168 Criterion.create! description: 'Se siguió el proceso definido.',
169         section: Section.find_by_name('General'),
170         order: 1
171 Criterion.create! description: 'Los datos recogidos del proceso son
      completos, precisos y coherentes.',
172         section: Section.find_by_name('General'),
173         order: 2
174 Criterion.create! description: 'El estudiante hizo su propio trabajo.',
175         section: Section.find_by_name("General"),
176         order: 3

```

Anexo 3 - Testing

.6. Bugs reportados por las docentes en etapa de Testing (ambiente de staging)

Se presenta a continuación la tabla provista por las docentes con los defectos encontrados en etapa de prueba. En algunos casos los mismos tienen una prioridad establecida.

Defecto	Prioridad
Como docente no veo la entrega (attach) del estudiante	-
Projects ofstudent3 - separar con espacio	-
POST MORTEN deberia ser POST MORTEM	-
No exigiria 15 caracteres en Other notes de los campos de PM	-
Al momento de mandar el ejercicio 2 creo que deberia pedir las locs del ejercicios 1. Puede que eso no esté?	-
En PSP0, es decir en el ejercicio 1 durante la planificacion deben tener el campo PLAN TIME para completar. Hoy no está el campo hasta que pasan al ejercicio 2 (con PSP0.1)	Alta
Los ejercicios no deben arrancar solos, hoy el estudiante tiene la posibilidad de arrancar el 4, 5 sin haber completado el 1. El estudiante no debería tener la posibilidad de arrancar un ejercicios hasta que el docente no se lo asigne	Alta
Al entrar al student dashboard con rol Docente que se filtre por defecto los estudiantes de la docente logueada. Quiero ver solo mis estudiantes y no los de leti. En caso de querer ver los de leti aplico el filtro y veo los de leti	Media
No estamos recibiendo mail. En la herramienta origen recibiamos mails con los estudiantes entregaban y cuando nos enviaban mensajes a través de la plataforma. Ahora si no entro a mirar si me escribieron algo o entregaron no me entero.	Media

Cuadro 8: Criterios de la *grading* de evaluación de categoría General.

.7. Bugs y problemas encontrados en etapa de Producción

Se presenta la tabla utilizada por los desarrolladores en la fase de soporte, para registrar cada problema encontrado (y su solución) en el uso productivo de la aplicación.

Los posibles tipos de bug (que no se aprecian completamente en la imagen) eran:

- UI (interfaz de usuario)
- Devops o Base de datos
- Confusión en los Requisitos
- Algoritmos (mal implementados)

Fecha	Reportado por	Descripción	Tipo	Area	Fix	Fecha de correc.
20/03/2023	Estudiante	Un detalle en la UI de PSPCode no me deja colocar espacios en el text area de comentarios de cada fase, no sé si es por este canal que se reportan bugs o mando esto a algún correo	UI	Frontend	La herramienta le hacía un .trim() (quitaba los espacios) a algunos strings, para compararlos e identificar si había habido cambios, así decidí si hacer una nueva llamada al backend para actualizar la información del proyecto o no. El problema es que al hacer el refector y actualización de código a este componente, se hizo una modificación así que no solo no se llamaba al backend en casos de variar en "espacios", sino que tampoco se modificaba el string original, es decir, los espacios nunca eran agregados al mismo. Esto se corrigió eliminando el .trim() para comparar los strings, ya que no se deberían tomar como iguales.	20/03/2023
20/03/2023	Silvana	En el ejercicio 1 (que es PSP 0) está apareciendo una advertencia sobre falta de información en los campos de PIP, cuando en realidad no debería exigir esto.	Devops o ...	Backend	Teníamos una bandera "only_in_psp01", dicho algoritmo debería haberla tenido seteada en TRUE.	20/03/2023
23/03/2023	Silvana	estoy corrigiendo un estudiante y el grading me da una observación que no entiendo porque está mal. Dice que los defectos de compile se removieron antes de invocarse pero de acuerdo a las fechas no pasa eso y parece estar todo bien	Confusión...	Fullstack	El problema es que al estudiante (en el select) le aparecía más de una fase con el mismo nombre, ya que él había iterado numerosas veces en cada tipo de fases. Entonces al seleccionar una (cuquiera), podía suceder que el sistema tomase una fase que estaba posterior a la fase actual que el estudiante se encontraba editando. Lo que se hizo fue mostrar una fase por tipo en ese selector (de las posibles existentes), a partir de ahí, el sistema mostrará una advertencia al docente únicamente si el estudiante seleccionó una fase que no era anterior a la actual.	23/03/2023
24/03/2023	Estudiante	Varios estudiantes no pudieron entregar su trabajo este día. Silvana nos avisó de la situación. Se trataba de la entrega del primer proyecto (con PSP 0), algunos estudiantes pudieron entregarlo, otros no. El problema era que el estudiante presionaba "Submit Project" y el sistema daba error.	UI	Fullstack	El fix fue deshabilitar el botón una vez presionado y comenzar un loader. Lo que sucedía era que el estudiante daba "doble click" y por lo tanto algunas de las requests se lanzaban repetidas, y otras no se llegaban a lanzar. Lo que generaba que el proceso de envío al backend quedara truncado, y ya no se pudiera rehacer. Nos costó días encontrar el motivo del problema. Mientras tanto, al otro día de reportado, se corrigieron los proyectos corruptos desde la base de datos. Luego, el ... descubrimos la solución final y la hicimos desde frontend.	28/03/2023
25/03/2023	Silvana	Al crear una nueva versión de un proyecto (cuando el mismo requiere reintegrar), toda la información del mismo era duplicada correctamente pero el archivo zip subido por el estudiante no. Por lo tanto, al querer accederlo, no abría. El estudiante podía reemplazar el archivo en esta nueva versión, y funcionaba correctamente. Sin embargo, en muchos de los casos no era este entregable el causante de la reproberación del proyecto, por lo tanto, el estudiante entregaba sin modificar el archivo, y el docente no lograba abrirlo.	Devops o ...	Backend	Se corrigió el código de backend para realizar un correcto duplicado del archivo existente en la versión anterior. A su vez, se corrigieron los archivos ya corruptos existentes en el Bucket.	25/03/2023
25/03/2023	Silvana	Se deja de ver la pestaña "Corrections" cuando el estudiante da click en "continue Project" para comenzar la reintegración. El comportamiento esperado es que la pestaña corrections se mantenga si ya hay una corrección disponible, es decir que siempre sea visible con la información de la última versión corregida.	Confusión...	Frontend	Se corrigió desde el Frontend. El fix consistió en cambiar las condiciones según las cuales, la pestaña era mostrada. Siempre que ya existiera "project_feedback" asociado, la pestaña Corrections se mostraba con la información del último feedback. A su vez, la información del formulario es editable si solo sí, el usuario es de rol profesor y el proyecto está en etapa de corrección (estado correction_pending)	26/03/2023
6/04/2023	Silvana	Se cayó el servidor de AWS por primera vez.	Devops o ...	Backend	Lo reiniciamos y configuramos alertas para que cuando suceda nos avise y se reinicie automáticamente.	6/04/2023
20/04/2023	Estudiante	no estoy pudiendo adjuntar el zip con la entrega. Me sale el siguiente mensaje: "Something went wrong attaching the zip, please try again or contact your professor"	Devops o ...	Backend	Se llenaron las máquinas EC2 de AWS. Esto se dio ya que al hacer tantos deploys de fixes, olvidamos tener en cuenta que debíamos borrar las imágenes anteriores periódicamente. [insertar marco teórico explicando esto]. Se llenaron las 8 gigas de material inútil. Los borramos y todo volvió a funcionar.	20/04/2023
2/05/2023	Leticia	Leticia reportó que había un con el cual no estaba pudiendo comunicarse mediante el chat de la aplicación. Ella manifestó que le llegaba la notificación pero al entrar a la pestaña de mensajes solo veía mensajes viejos con dicho estudiante. También nos dijo que anteriormente se había podido comunicar sin problema.	Algoritmos	Frontend	El problema se dio ya que en los mensajes venían de forma paginada y en la aplicación no se estaba usando dicha paginación correctamente, siempre se estaba obteniendo la primera página. Esto no fue descubierto en etapa de testing ya que nunca se llegó al límite de mensajes por página. A su vez, en el código anterior, la paginación tampoco estaba correctamente utilizada. Simplemente, mostraba por defecto la última página, entonces los usuarios nunca notaron el problema.	4/05/2023
19/05/2023	Leticia	A Leticia le sucedía que no veía los mensajes de una estudiante en particular.	UI	Frontend	El bug se trataba de un caso borde que se daba en la UI de la pantalla de mensajes. El sistema desplegaba los mensajes en azul y a la derecha de la pantalla (es decir, identificados como propio del usuario) si el ID del usuario loggeado coincidía con el ID del usuario remitente de ese mensaje. El problema es que los mensajes siempre se dan entre docentes y estudiantes. Y en base de datos, ambos tipos de usuario están almacenados en dos tablas independientes. Lo que sucedía en este caso es que el ID de Paula, la estudiante con la que sucedía esta problemática, compartía ID (2) con Leticia, la docente que reportó el problema. Para solucionar el bug se pasó a comparar IDs y también rol del usuario, para identificar si los mensajes son propios o del otro lado de la comunicación.	19/05/2023

Figura 32: Tabla de bugs en producción.

Anexo 4 - Despliegues y Monitoreo

.8. Archivo de configuración de Docker

A continuación se presenta el archivo `docker-compose.yml` el cual contiene la configuración necesaria para levantar la aplicación *Backend* y *Frontend*:

```
1 version: "3"
2 services:
3   backend-prod:
4     image: public.ecr.aws/m9q5t8l6/psp-code-backend:0.0.1
5     restart: unless-stopped
6     environment:
7       - SECRET_KEY_BASE=${SECRET_KEY_BASE}
8       - RAILS_ENV=production
9       - SERVER_URL=svc.pspcode.com
10      - PASSWORD_RESET_URL=www.pspcode.com
11      - USE_SENDGRID_SERVICE=true
12      - USE_SENDGRID_NOTIFICATION=true
13      - SENDGRID_USERNAME=${SENDGRID_USERNAME}
14      - SENDGRID_PASSWORD=${SENDGRID_PASSWORD}
15      - BUCKET_ACCESS_KEY_ID=${BUCKET_ACCESS_KEY_ID}
16      - BUCKET_SECRET_ACCESS_KEY=${BUCKET_SECRET_ACCESS_KEY}
17      - S3_BUCKET_NAME=s3-psp-bucket
18      - FROM_EMAIL_ADDRESS=course.psp@gmail.com
19      - HOST_URL=www.pspcode.com
20     ports:
21       - "3000:3000"
22     expose:
23       - 3000
24
25   frontend-prod:
26     image: public.ecr.aws/m9q5t8l6/psp-code-frontend:0.0.1
27     restart: unless-stopped
28     ports:
29       - 8080:80
```

Se definen dos servicios, uno para el *Backend* y otro para el *Frontend*, se especifican los puertos que utilizan la aplicación y las variables de entorno. Estas como ya se describió, están definidas en otro archivo oculto en la ruta del Droplet.

A continuación se presenta el archivo `docker-compose.yml` con la configuración del Droplet que aloja la base de datos y el *Proxy*:

```
1 version: '3'
2 services:
3   nginx-proxy:
4     image: 'jc21/nginx-proxy-manager:latest'
5     restart: unless-stopped
6     ports:
7       - '80:80'
8       - '81:81'
9       - '443:443'
10    volumes:
11      - ./data:/data
12      - ./letsencrypt:/etc/letsencrypt
13   db:
14     image: postgres:14.7-alpine3.17
15     restart: unless-stopped
16     volumes:
17       - ./db:/var/lib/postgresql/data
18     environment:
19       - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
20     ports:
21       - "5444:5433"
22       - "5443:5432"
```

Este archivo de Docker define dos servicios: uno para un *Proxy Nginx* (utilizando la imagen `jc21/nginx-proxy-manager`) y otro para una base de datos PostgreSQL (utilizando la imagen `postgres:14.7-alpine3.17`). Estos servicios están configurados para reiniciarse automáticamente y utilizan volúmenes para persistir datos importantes, como la configuración del *Proxy* y los datos de la base de datos.

.9. Script de monitoreo de estado de la aplicación

```
1 #!/bin/bash
2
3 max_attempts=5
4 host="localhost"
5 port_3000=3000
6 port_8080=8080
7
8 for ((i = 1; i <= max_attempts; i++)); do
9   if (echo >/dev/tcp/$host/$port_3000) &>/dev/null && (echo >/dev/tcp/
10    $host/$port_8080) &>/dev/null; then
11     break
12   else
13     if [ $i -eq $max_attempts ]; then
14       cd ~/ && docker compose down && docker compose up -d
15     fi
16     sleep 1
17   fi
18 done
```

Este *script* intenta conectarse a los puertos 3000 (Backend) y 8080 (Frontend) del *localhost* abriendo una conexión TCP, hasta que ambas conexiones sean exitosas o se alcance el número máximo de intentos. Si se alcanza el límite de intentos, detiene los contenedores Docker definidos en el archivo `docker-compose.yml` en el directorio de inicio y luego los vuelve a iniciar en segundo plano.

Este tipo de *script* es útil para reiniciar servicios o contenedores en caso de problemas de conexión.

.10. Firewalls configurados en cada uno de los Droplets

Inbound Rules

Set the Firewall rules for incoming traffic. Only the specified ports will accept inbound connections. All other traffic will be blocked.

Type	Protocol	Port Range	Sources	
HTTP	TCP	80	All IPv4 All IPv6	More ▾
Custom	TCP	81	All IPv4 All IPv6	More ▾
HTTPS	TCP	443	All IPv4 All IPv6	More ▾
Custom	TCP	5442	All IPv4 All IPv6	More ▾
Custom	TCP	5443	All IPv4 All IPv6	More ▾
New rule ▾				

Outbound Rules

Set the Firewall rules for outbound traffic. Outbound traffic will only be allowed to the specified ports. All other traffic will be blocked.

Type	Protocol	Port Range	Destinations	
ICMP	ICMP		All IPv4 All IPv6	More ▾
All TCP	TCP	All ports	All IPv4 All IPv6	More ▾
All UDP	UDP	All ports	All IPv4 All IPv6	More ▾
New rule ▾				

Figura 33: Firewall Droplet 1. Base de datos y Proxy

Inbound Rules

Set the Firewall rules for incoming traffic. Only the specified ports will accept inbound connections. All other traffic will be blocked.

Type	Protocol	Port Range	Sources	
Custom	TCP	3000	ubuntu-s-1vcpu-512mb-10gb-2	More ▾
Custom	TCP	8080	ubuntu-s-1vcpu-512mb-10gb-2	More ▾

[New rule](#) ▾

Outbound Rules

Set the Firewall rules for outbound traffic. Outbound traffic will only be allowed to the specified ports. All other traffic will be blocked.

Type	Protocol	Port Range	Destinations	
ICMP	ICMP		All IPv4 All IPv6	More ▾
All TCP	TCP	All ports	All IPv4 All IPv6	More ▾
All UDP	UDP	All ports	All IPv4 All IPv6	More ▾

[New rule](#) ▾

Figura 34: Firewall Droplet 2. Backend y Frontend

Lo importante en este caso son las *Inbound Rules*, que representan las conexiones permitidas de entrada al Droplet. Como se puede ver, en la primera figura se aceptan las conexiones TCP de los puertos 5442 y 5443 que corresponden a las conexiones de la base de datos. Hay tres puertos para gestionar las conexiones de Nginx. El puerto 81 está reservado exclusivamente para tareas administrativas. Por otro lado, los puertos 80 y 443 son esenciales para recibir conexiones externas. El puerto 80 se utiliza para el tráfico HTTP, que se emplea cada vez que alguien accede a la web de manera convencional. En cambio, el puerto 443 se asigna al tráfico destinado a navegación segura mediante el protocolo HTTPS. Importante destacar que Nginx realiza automáticamente la redirección de todas las conexiones entrantes desde el puerto 80 hacia el puerto 443. A pesar de esta redirección, es crucial mantener el puerto 80 abierto, ya que, de lo contrario, no sería posible establecer la conexión inicial.

En la segunda figura, se ve como sólo se permiten conexiones entrantes desde el otro Droplet, es decir se bloquean todas las conexiones directas y externas.

Si se quisiera acceder a cualquiera de los Droplet, se tiene que agregar una regla de entrada (haciendo click en *New Rule*) para permitir conexiones de tipo SSH en el puerto por defecto que es el 22. Esto habilita que se pueda acceder al sistema operativo del Droplet a través de la consola presionando el link *Console* que se encuentra en la información de cada uno. Accediendo al Droplet, se pueden realizar despliegues de la aplicación (imágenes de docker que se guardan en los repositorios de AWS), esto se detalla en el archivo *README* del proyecto, tanto para el Backend como para el Frontend.