DISEÑO Y CONSTRUCCIÓN DE UNA HERRAMIENTA PARA CAPACITAR TESTERS

TEST SCHOOL

Marzo 2015 – Octubre 2016
Montevideo, Uruguay

Proyecto de grado
Tutora: Mónica Wodzislawski
Co-tutora: Mariana Travieso, Centro de Ensayos de Software

Integrantes:
Cecilia Barreiro Albarenga
Matías Mansilla Scala
Sebastián Jara Ibarra

ÍNDICE

ÍNDICE **3**

GLOSARIO **8**

1. INTRODUCCIÓN **9**

1.1. PLANTEO Y DEFINICIÓN DEL PROBLEMA 9
1.1.1. Necesidades del CES *9*
1.1.2. Diplomas de la carrera *9*
1.2. OBJETIVOS 10
1.3. RESUMEN 11
1.4. RESULTADOS ESPERADOS 11
1.5. CONCLUSIONES RESUMIDAS 12
1.6. CRONOGRAMA 12
1.6.1. Cronograma inicial *13*
1.6.2. Revisión de cronograma y organización del proyecto *13*

2. REQUISITOS Y FUNCIONALIDADES **15**

2.1. VISIÓN GENERAL 15
2.2. REQUISITOS FUNCIONALES 15
2.3. REQUISITOS NO FUNCIONALES 16
2.4. ESPECIFICACIÓN DE FUNCIONALIDADES DEL SISTEMA 17
2.4.1. Usuario administrador *17*
2.4.1.1. Panel principal de administración *17*
2.4.1.2. Panel de correcciones *19*
2.4.2. Usuario Test School *23*

3. ESTADO DEL ARTE **27**

3.1. INTRODUCCIÓN 27
3.2. CALIDAD DE SOFTWARE 27
3.3. TESTING DE SOFTWARE 28
3.3.1. Verificación y Validación *29*
3.3.2. Error, defecto y falla *30*
3.3.3. Caso de prueba o test *31*
3.3.4. Oráculo *32*
3.4. TIPOS DE TESTING 32
3.4.1. Por opacidad *33*
3.4.1.1. Testing estructural o de caja blanca *33*
3.4.1.2. Testing de caja negra *33*
3.4.2. Por objeto bajo prueba *34*
3.4.2.1. Pruebas unitarias *34*
3.4.2.2. Pruebas de integración *34*
3.4.2.3. Pruebas funcionales y del sistema *34*
3.4.2.4. Pruebas de aceptación *34*
3.4.2.5. Pruebas de regresión *35*
3.5. TESTING FUNCIONAL 35
3.5.1. Descripción *35*
3.5.2. Testing exploratorio *35*
3.5.2.1. Testing basado en sesiones *37*

3.5.3. Testing planificado 39
3.5.4. Técnicas de diseño de casos de prueba 40
3.5.4.1. Clases de equivalencia 40
3.5.4.2. Valores límite 41
3.5.4.3. Combinación por pares 41
3.5.4.4. Máquinas de estado 42
3.5.4.5. Tablas de decisión 42
3.5.4.6. Basadas en casos de uso 43
3.6. PROCESO DE TESTING FUNCIONAL 43
3.6.1. Etapas 43
3.6.1.1. Inventario de pruebas 44
3.6.1.2. Plan de pruebas 44
3.6.1.3. Diseño de las pruebas 45
3.6.1.4. Ejecución de pruebas 45
3.6.1.5. Reporte de Incidentes 45
3.7. MUTACIONES 46
3.8. HERRAMIENTAS INVESTIGADAS 47
3.8.1. Herramientas para inyección de errores 47
3.8.2. Aplicaciones de inspiración e-schooling 48
3.8.3. Elección de aplicación a mutar 48

4. TEST SCHOOL 52

4.1. DESCRIPCIÓN 52
4.2. ESTRUCTURA POR NIVELES 52
4.3. DISEÑO DE INTERFAZ 52
4.3.1. Concepto base 53
4.3.2. Conceptos tomados de otras aplicaciones 53
4.3.2.1. CodeSchool 53
4.3.2.2. Moodle 53
4.3.3. Descripción de la interfaz 54
4.3.3.1. Sitio web Test School 54
4.3.3.2. Panel de administración 56
4.4. VERSIÓN INICIAL 59
4.4.1. Escenarios iniciales 59
4.4.2. Aplicación utilizada 59
4.5. LENGUAJES UTILIZADOS 60
4.5.1. Capa de acceso a datos del sistema 60
4.5.2. Capa de presentación 60
4.6. HERRAMIENTAS UTILIZADAS 60
4.6.1. Repositorios 60
4.6.1.1. Código 60
4.6.1.2. Documentos 61
4.6.2. Manejo de tareas 61
4.6.3. Calidad de código 61
4.6.4. Test unitario 61
4.6.5. Test de integración 61
4.6.6. Test de rendimiento 61
4.7. REQUISITOS QUE CAMBIARON 62
4.7.1. Registro de la sesión 62
4.7.2. Inserción de mutaciones 62

5. ARQUITECTURA 63

5.1. ORGANIZACIÓN 63
5.2. PRESENTACIÓN 63
5.3. USUARIO ADMINISTRADOR TEST SCHOOL 64
5.3.1. Descripción 64
5.3.2. Estilo arquitectónico 64
5.4. USUARIO TESTER TEST SCHOOL 65
5.4.1. Descripción 65
5.4.2. Estilo arquitectónico 65
5.5. USUARIOS MANTIS 66
5.5.1. Administrador 66
5.5.2. Tester 66
5.5.3. Estilo arquitectónico 66
5.6. TECNOLOGÍAS UTILIZADAS 67
5.6.1. Ruby on Rails 67
5.6.2. AngularJS 70
5.6.3. PostgreSQL y MySQL 72
5.7. JUSTIFICACIÓN 72
5.7.1. Usuario tester Test School 72
5.7.2. Usuario administrador Test School 72
5.7.3. Usuarios Mantis 73
5.8. DIAGRAMA DE CLASES 74
5.8.1. Stage 75
5.8.2. Level 75
5.8.3. MutatedApp 77
5.8.4. KnownBug 77
5.8.5. WhitelistMember 79
5.8.6. Session 79
5.8.7. Note 80
5.8.8. TestingTechnique 81
5.8.9. User 81
5.8.10. AdminUser 83
5.8.11. Modelos Mantis 83

6. CASO DE ESTUDIO 84

6.1. OBJETIVO 84
6.2. PLANIFICACIÓN 84
6.2.1. Material 84
6.2.1.1. Manual para usuarios administradores 84
6.2.1.2. Videos para administradores y usuarios 84
6.2.1.3. Encuesta 84
6.3. RESULTADOS DE LA ENCUESTA 86

7. GESTIÓN DE RIESGOS 90

7.1. OBJETIVO 90
7.2. IDENTIFICACIÓN 90
7.3. ANÁLISIS 90
7.3.1. Riesgos identificados 90
7.3.2. Monitoreo 93

8. CONCLUSIONES 97

9. REFERENCIAS BIBLIOGRÁFICAS 100

ANEXOS 103

ANEXO I: CATEGORIZACIÓN DE INCIDENTES 104
ANEXO II: REGISTRO DE PRUEBAS 106
ANEXO III: SERVICIOS DE API 112
ANEXO IV: GESTIÓN DE SEGURIDAD 120
ANEXO V: ESPECIFICACIONES DE ESCENARIOS PARA APLICACIÓN SUPERMARKET 122
ANEXO VI: DEFECTOS INYECTADOS Y TÉCNICAS SUGERIDAS 133
ANEXO VII: ESCENARIOS PRESENTADOS A ESTUDIANTES 146
ANEXO VIII: RANKING DE APLICACIONES EVALUADAS 153
ANEXO IX: HERRAMIENTAS QUE SOPORTAN INYECCIÓN DE DEFECTOS 159
ANEXO X: DESCRIPCIÓN DE MANTISBT 160

ÍNDICE DE FIGURAS

FIGURA 2.1: FUNCIONALIDADES DE UN USUARIO ADMINISTRADOR EN EL PANEL DE ADMINISTRACIÓN....17
FIGURA 2.2: FUNCIONALIDADES DE UN USUARIO ADMINISTRADOR EN EL PANEL DE CORRECCIONES....20
FIGURA 2.3: FUNCIONALIDADES DE UN USUARIO TEST SCHOOL QUE SE RELACIONAN ÚNICAMENTE CON LA BASE DE DATOS POSTGRESQL. 23
FIGURA 2.4: FUNCIONALIDADES DE UN USUARIO TEST SCHOOL QUE SE RELACIONAN CON AMBAS BASES DE DATOS. 23
FIGURA 2.5: FUNCIONALIDAD DE UN USUARIO TEST SCHOOL CON MANTISBT. 24
FIGURA 3.1: CARACTERÍSTICAS DE CALIDAD. 28
FIGURA 3.2: DIAGRAMA "ERROR-DEFECTO-FALLA". 30
FIGURA 3.3: APLICACIÓN DE TÉCNICA DE CAJA NEGRA. EXTRAÍDO DE "SOFTWARE ENGINEERING", 7MA EDICIÓN, CAPÍTULO 23. 34
FIGURA 3.4: EXTRAÍDO DE LAS NOTAS DE TVS [53]. 36
FIGURA 3.5: TESTING PLANIFICADO. 39
FIGURA 3.6: DETALLE DE LAS ETAPAS DEL PROCESO DE TESTING....44
FIGURA 4.1: PÁGINA PRINCIPAL DE TEST SCHOOL 54
FIGURA 4.2: PÁGINA DE LA APLICACIÓN MUTADA 56
FIGURA 4.3: PANEL DE ADMINISTRACIÓN DE TEST SCHOOL 57
FIGURA 4.4: PANEL PRINCIPAL DE CORRECCIONES. 58
FIGURA 4.5: APAREO DE BUGS CONOCIDOS Y BUGS REPORTADOS POR UN USUARIO. 58
FIGURA 5.1: DIAGRAMA QUE REPRESENTA LA ARQUITECTURA DEL SISTEMA Y COMO SE COMUNICAN LOS COMPONENTES ENTRE SÍ...64
FIGURA 5.2: LOS COMPONENTES RESALTADOS SON AQUELLOS ALCANZADOS POR UN ADMINISTRADOR. 65
FIGURA 5.3: LOS COMPONENTES RESALTADOS SON AQUELLOS ALCANZADOS POR UN USUARIO ADMINISTRADOR. 66
FIGURA 5.4: LOS COMPONENTES RESALTADOS SON AQUELLOS ALCANZADOS POR LOS USUARIOS MANTIS, ADMINISTRADORES Y DESARROLLADORES....67
FIGURA 5.5: ARQUITECTURA DE PROYECTO RUBY ON RAILS 68
FIGURA 5.6: DIAGRAMA DE CLASES DE TEST SCHOOL. 74
FIGURA 6.1: RESPUESTAS DE LA ENCUESTA A LOS ESTUDIANTES. 87
FIGURA 6.2: RESPUESTAS DE LA ENCUESTA A ADMINISTRADORES. 88

Glosario

API: También conocida como la interfaz de programación de aplicaciones. Es el conjunto de subrutinas, funciones y procedimientos o métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
Backend: refiere a la capa de acceso a datos del sistema, la cual soporta la lógica de la aplicación.
CRUD: acrónimo en inglés que significa Crear, Leer, Actualizar o Borrar.
DOM: es una API para documentos válidos y bien construidos. Define la estructura válida de documentos y el modo en que se acceden y se manipulan.
DSL: Lenguaje específico de dominio. Es un lenguaje especializado para resolver un dominio específico de problemas. A diferencia de un lenguaje de programación que es realizado para resolver un dominio muy amplio de problemas.
E-Learning ó e-Schooling: se entiende como el proceso de aprender apoyado en o mediado por, la tecnología. Se dice también de la educación virtual que se conoce como "a distancia". El proceso educativo se realiza haciendo uso de canales electrónicos (en especial Internet), utilizando herramientas y aplicaciones digitales, como soporte para enseñar y aprender.
Framework: Define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
Frontend: refiere a la capa de presentación de la aplicación.
Gema: así se les llama a las librerías para el lenguaje Ruby.
Heurísticas: así se conocen el conjunto de técnicas o métodos para resolver un problema.
JSON: JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.
ORM: Mapeo objeto-relacional es la técnica mediante la cual se mapea un sistema de tipos perteneciente a un lenguaje orientado a objetos con los datos de una base relacional.
Pedido HTTP: protocolo de comunicación sin estado para transferencia de información en la web.
Pull request: Petición contra el proyecto base para fusionar contribuciones externas.
Rack: es un conjunto muy común de especificaciones que debe cumplir una aplicación que quiera utilizar este tipo de servidores, utilizada para realizar servidores web en Ruby.
REST: Es un estilo de arquitectura para sistemas distribuidos.
SPA: es una aplicación web ejecutada en una única página. El contenido cambia solo en algunas partes lo que permite reutilizar gran parte de la página. Por ejemplo, el encabezado y el pie de página.
Stack del proyecto: conjunto de componentes que forman parte del proyecto.
Stakeholders: Son todas aquellas personas, grupos y entidades que tienen intereses de cualquier tipo en una empresa y se ven afectados por sus actividades. Son interesados, directos o indirectos, en que la empresa funcione.
SWIFI: Son técnicas de inyección de errores en el software. Existen 2 tipos: aquellas que se agregan en tiempo de compilación y las que se agregan en tiempo de ejecución.
Tecnología de la Información (TI): El término es comúnmente utilizado como sinónimo para los computadores, y las redes de computadoras, pero también abarca otras tecnologías de distribución de información, tales como la televisión y los teléfonos.
Test: Prueba de software diseñada para un sistema en particular y ejecutada sobre éste.
Tester: persona que ejecuta pruebas (tests) sobre un sistema.

Token: firma cifrada que permite a la API identificar al usuario. Se envía en la cabecera de un pedido HTTP.

1. Introducción

1.1. Planteo y definición del problema

1.1.1. Necesidades del CES

El Centro de Ensayos de Software imparte una carrera de Testing en línea con el objetivo de formar profesionales en Testing de Software capaces de planificar, ejecutar, diseñar y gestionar pruebas de productos de software.
Los métodos actuales de evaluación que manejan en estos cursos son manuales y por este motivo se vio la necesidad de contar con algún proceso automatizado de aprendizaje, evaluación y corrección que facilite y agilice la tarea de estudiantes y profesores, proporcionando información útil y confiable sobre dicha evaluación.

Además esta herramienta puede servir como plataforma para mejorar y diversificar la metodología de enseñanza utilizando herramientas TI. La integración de aplicaciones de aprendizaje electrónico (e-learning o e-schooling) se ha impuesto en los últimos años en el dictado de cursos, y existen diversos ejemplos en el mercado, pero no parece haber ninguna que se ajuste exactamente a la realidad del CES.

1.1.2. Diplomas de la carrera

Tester de Software	Tester profesional de Software	Líder de testing de Software
Un tester de software graduado: ● Comprende los conceptos básicos de la disciplina de testing, testing funcional y de performance. ● Conoce la gestión de incidencias, comprende la importancia de la documentación en el testing. ● A partir de lo aprendido puede hacer experiencia en la industria, trabajando. Consideramos que está apto para desempeñarse como tester, ejecutando y comenzando a diseñar pruebas.	Un tester profesional de software graduado: ● Perfeccionó los conocimientos y habilidades adquiridas, su capacidad de diseño y de selección de la mejor estrategia y técnicas en un contexto determinado. ● Es capaz de planificar los recursos disponibles para lograr los objetivos de las pruebas, responsabilizarse por proyectos de testing de mediano porte y proponer mejoras en los procesos involucrados.	Un líder de testing de software graduado: ● Coordina y dirige las actividades de testing de la organización y brega por la mejora permanente de los procesos involucrados. ● Promueve la jerarquización del testing sobre la base de dar visibilidad a la información que aporta y mostrar cómo puede ser utilizada para tomar decisiones. ● Entrena y promueve la capacitación permanente de los integrantes del grupo de testing.

Los testers de software graduados deberían ser capaces de detectar todo tipo de incidentes, por ejemplo:

- Funcionalidad no disponible.
- Implementación incorrecta.
- Salida anormal.
- Manejo de errores.
- Errores de control de flujo.
- Errores de interfaz de usuario.
- Errores de hardware.
- Errores de seguridad.
- Mejora
- No clasificado
- Observación.
- Datos erróneos o errores ortográficos.
- Aspectos de usabilidad.
- Funcionalidades incorrectas.
- Errores en el flujo de un caso de uso.

Para detectar los incidentes deberían aplicar estrategias y técnicas de testing:

- Testing exploratorio: en este caso se requiere definir misiones y reportar sesiones para cada aspecto que se desee probar.
- Clases de equivalencia.
- Máquinas de estado.
- Tablas de decisión.
- Árboles de decisión.
- Combinación por pares.

Además, debería elaborar documentos y registrar notas sobre las pruebas realizadas.

1.2. Objetivos

Los principales objetivos del proyecto son:

- Comprender a fondo las estrategias y técnicas de testing dictadas en los cursos del CES, las realidades a las que se aplican y los tipos de defectos que detectan.
- Brindar al CES una herramienta que asista en la evaluación de los alumnos de cursos fundamentales del diploma de tester de software, lo cual implica:
    - Agilidad de corrección: el sistema debe presentar mutaciones de una aplicación y automatizar la corrección de los defectos reportados.
    - Flexibilidad: el administrador podrá armar niveles de dificultad incluyendo nuevas mutaciones de una aplicación.
    - Efectividad: las versiones defectuosas de la aplicación con las que contará el sistema serán pensadas para evaluar técnicas de testing específicas y los defectos inyectados en ellas serán analizados en detalle para que la evaluación sea coherente y efectiva.
    - El administrador podrá contar con datos estadísticos sobre el desempeño de los estudiantes.

- Permitir que el sistema sea extensible en un futuro, no solo para agregar nuevas aplicaciones, sino también para otras áreas por fuera del testing.

1.3. Resumen

Dado el problema planteado y la motivación expresada por el CES, se construyó una herramienta que asista en el proceso de evaluación de los trabajos de los testers por parte de los profesores de los cursos que allí se brindan.
La principal premisa tenida en cuenta para diseñar y construir la herramienta, fue agilizar el proceso de corrección de los profesores y facilitar el trabajo de los estudiantes, brindándoles una interfaz amigable y motivándolos a progresar dentro del sistema.

El área de aplicación inicial es la enseñanza del testing, por lo cual fue necesario comprender, analizar e investigar varios conceptos fundamentales de la disciplina y herramientas de distintas índoles. De esta manera se pudieron evaluar y mitigar algunos riesgos asociados a requisitos iniciales, para saber si éstos eran viables y en base a eso se construyó una versión inicial del sistema.

Durante la etapa de investigación se estudiaron herramientas con distintos objetivos: por un lado se investigaron aquellas que soportan inyección de defectos, para evaluar riesgos asociados a la generación de mutaciones; y por otro lado, se analizaron algunas aplicaciones de aprendizaje electrónico (o e-learning) para captar algunos aspectos didácticos interesantes y adaptarlos a nuestro sistema.
Otras aplicaciones se investigaron para encontrar una que pudiera ser utilizada como parte de la versión inicial, por lo cual se debía ajustar a las necesidades de los usuarios testers y los integrantes del equipo, ya que fue necesario modificarla.

La aplicación simula de la forma más realista posible, el escenario con el que se encuentran los testers en una rutina usual de su flujo de trabajo. Para esto se les brindan varios escenarios, y para cada uno de ellos, podrán acceder a una aplicación mutada (levantada dentro del propio sistema como un elemento HTML). El objetivo de los estudiantes es reportar todos las incidencias que encuentren sobre dicha mutación, así como también generar un reporte de pruebas que entregan como parte de su trabajo, junto con los reportes ingresados.

Los profesores se encargan de gestionar y manejar usuarios, aplicaciones disponibles, niveles para dichas aplicaciones y las mutaciones asociadas a dichos niveles, así como también los defectos conocidos para cada una de ellas.
Además de gestionar los escenarios y mutaciones que se presentan a los estudiantes, los profesores pueden visualizar toda la información necesaria al momento de la evaluación de un trabajo y cuentan con elementos para realizar una corrección ágil. Para esto podrán ver en forma clara y concisa el trabajo entregado por un estudiante y asociar fácilmente lo que éste reportó y lo que debería haber reportado, basándose en las especificaciones de cada escenario.

1.4. Resultados esperados

En esta sección se presentan los resultados que se esperan obtener del proyecto de grado.

- Elaborar un estado del arte que brinde un marco referencial al proyecto de grado, donde se expongan los tópicos principales y secundarios referentes al objetivo del mismo.
- Implementar un sistema que permita a los administradores contar con varias versiones de una misma aplicación, las cuales manejan distintos niveles de dificultad, ajustándose al nivel de exigencia de cada curso.
- Los administradores podrán acceder a los registros de cada usuario y obtener información relevante para su evaluación.
- Proveer un conjunto de mutaciones de una misma aplicación que conformará la versión inicial del sistema.
- Armar un caso de estudio donde la versión inicial del sistema serán evaluada por un grupo de personas (profesores y estudiantes) cuyas devoluciones serán tomadas en cuenta para las conclusiones.
- El sistema será extensible y podrá soportar nuevas versiones mutadas, tanto de la aplicación utilizada inicialmente, como de cualquier otra que se quiera agregar en el futuro.

1.5. Conclusiones resumidas

Utilizando Ruby on Rails y AngularJS se construyó Test School, una herramienta intuitiva y amigable que motiva a los estudiantes a través de los diferentes niveles de la aplicación.
Las funcionalidades brindadas por el panel de administración permiten a los docentes agilizar y mejorar el proceso de evaluación del trabajo realizado por los estudiantes.
Se integró MantisBT a la plataforma, permitiendo a los usuarios familiarizarse con sus funcionalidades.
Finalmente, se implementó una plataforma altamente escalable y con grandes posibilidades de crecimiento.

1.6. Cronograma

Basándonos en los requisitos iniciales y considerando la información que se manejaba en ese momento, se analizaron extensamente los riesgos, se estimó de forma preliminar el esfuerzo y la fecha de finalización del sistema.
Cabe destacar que dicha valoración fue solo una noción inicial cuya principal función era colaborar en la organización del trabajo y permitir planificar su distribución para completar el trabajo en un período de tiempo razonable.
Junto con las estimaciones se organizó la forma de trabajo que se pensaba llevar a cabo inicialmente.
Al momento de estimar nos resultó más efectivo hacerlo utilizando el juicio de expertos, ya que más allá de la experiencia de los mismos integrantes del grupo, se les consultó a personas con conocimientos en las tecnologías que se utilizaron y éstos colaboraron en la estimación.

1.6.1. Cronograma inicial

A partir de las estimaciones manejadas inicialmente se construyó un cronograma, aunque éste sufrió modificaciones por diversos motivos, siendo el más importante que no se había hecho un análisis profundo de riesgos y éstos se habían subestimado.
A lo largo del proyecto estas fechas fueron revisadas y actualizadas.

Nombre	Fecha inicio	Fecha fin
Comienzo de proyecto	18/3/2015	
Definición del alcance definitivo así como de características de la plataforma - 1 mes	18/3/2015	19/4/2015
Estado del arte en testing de software y mutaciones, capacitación de testers - 1 mes	20/4/2015	18/5/2015
Construcción de la plataforma - 4 meses	18/5/2015	7/9/2015
Caso de estudio - 1 mes	7/9/2015	5/10/2015
Elaboración final del informe - 1 mes	5/10/2015	2/11/2015

Tabla 1: Cronograma inicial.

1.6.2. Revisión de cronograma y organización del proyecto

Una vez avanzado el proyecto se hicieron más claras las tareas que eran necesarios llevar a cabo y el esfuerzo que éstas implicaban. También se había hecho un análisis inicial de riesgos, y aunque se contaba con planes de mitigación para éstos, aún reflejaban que era factible algún atraso. Por este motivo se realizó un nuevo cronograma, esta vez en mayor detalle, con las fechas de inicio y finalización que se esperaba para cada tarea para así tener una visión más realista hacia la completitud del proyecto.

Se organizó el proyecto siguiendo un proceso iterativo incremental donde en cada iteración se agregarían nuevas funcionalidades partiendo por las más importantes o riesgosas, ya que era necesario mitigar ciertos riesgos correspondientes a algunas de ellas. Esto permitiría manejar versiones a lo largo del proyecto, mostrando incrementos del producto al finalizar cada iteración.

Aunque esta era la idea original, algunas funcionalidades que debieron ser abordadas con mayor prioridad consumieron más tiempo que el esperado y eso conllevó a algunos cambios en la organización de tareas, siendo necesario volver a priorizarlas en algunos casos.
En la tabla 2 se pueden ver las tareas distribuidas en iteraciones, junto con los documentos que se pensaban realizar en paralelo.

Finalmente y dado que no se pudo cumplir el nuevo cronograma planteado, se planteó el mes de julio de 2016 como fecha estimada de finalización del proyecto.

Nombre	Fecha de inicio	Fecha de fin
Documentos	29/04/15	9/07/15
Estado del arte	15/05/15	9/07/15
Riesgos y planes de mitigación	29/04/15	23/06/15
Cronograma y estimaciones	29/05/15	25/06/15
Eleccion de aplicaciones	15/05/15	2/07/15
Descripcio del problema y motivacion	15/05/15	2/07/15
Especificación de requisitos	8/05/15	18/06/15
Diseño de la aplicación	19/06/15	9/07/15
Alcance del proyecto	15/05/15	9/07/15
Buscar aplicaciones a mutar	29/05/15	22/06/15
Mitigacion de riesgos	29/05/15	30/06/15
Construir aplicación Angular JS	29/05/15	29/05/15
Prototipo para cada aplicación a mutar	29/05/15	29/05/15
Grabar sesiones	29/05/15	29/05/15
Ejecución de pruebas	29/05/15	30/06/15
Iteración 1	1/07/15	20/08/15
Panel administrador	1/07/15	4/08/15
Página principal	1/07/15	4/08/15
Ejecución pruebas integración	5/08/15	20/08/15
Iteración 2	21/08/15	30/08/16
Refinar diseño de página	21/08/15	25/08/15
Corrección bugs iteración 1	21/08/15	30/08/16
Modificaciones del feedback	1/09/15	7/09/15
Nuevas funcionalidades	8/09/15	16/10/15
Ejecución de pruebas unitarias y de integración	19/10/15	22/10/15
Iteración 3	23/10/15	27/11/15
Corrección bugs iteración 2Modificaciones del feedback	23/10/15	27/10/15
Nuevas funcionalidades	28/10/15	20/11/15
Pruebas finales	20/11/15	27/11/15
Caso de estudio	1/12/15	18/12/15
Elaboración final del informe	1/01/16	26/02/16

Tabla 2: Cronograma revisado

2. Requisitos y funcionalidades

2.1. Visión general

El sistema cuenta con un módulo de administración, que permite manejar mutaciones de aplicaciones para que el tester ejecute pruebas, detecte incidentes y los reporte.
También debe ser capaz de registrar la actividad del tester durante su sesión de entrenamiento, presentando al docente los resultados obtenidos. De esta manera el docente podrá evaluar de manera sencilla el trabajo de los estudiantes, siendo este el principal objetivo.
A su vez, otro de los objetivos es entrenar testers para que desarrollen las habilidades necesarias a partir de los conocimientos teóricos adquiridos.
El sistema intenta simular la experiencia de testear un software de la forma más realista posible, para lo cual se toma una mutación de una aplicación, o simplemente una aplicación a probar, y se le asignan un conjunto de defectos conocidos que tendrán asociada una descripción, severidad, reproducibilidad y prioridad, entre otros. Una vez que la mutación está disponible en una URL y se definieron los defectos conocidos, ésta quedará disponible para que el tester comience su instancia de entrenamiento.
El sistema se dividirá a grandes rasgos en los siguientes componentes:

- Un panel de administración: donde los administradores podrán agregar mutaciones de aplicaciones al sistema, manejar usuarios y acceder a las sesiones enviadas por éstos para corregirlas.
- Un portal web: donde acceden los usuarios (testers), prueban y detectan los errores inyectados en las aplicaciones disponibles.
- Portal web MantisBT: donde los usuarios (testers) reportan los defectos encontrados en las aplicaciones mutadas que prueban en la aplicación web.

2.2. Requisitos funcionales

A continuación se presentan los requisitos funcionales del sistema.

Administrador

1. Inicio de sesión: Un usuario administrador podrá ingresar al panel de administración.
2. Cierre de sesión: Un usuario administrador podrá salir del panel de administración.
3. Crear un usuario administrador: Un usuario administrador podrá crear otro desde el panel de administración.
4. Borrar un usuario administrador: Un usuario administrador podrá ser borrado del sistema desde el panel de administración. Siempre quedará al menos un usuario administrador en el sistema.
5. Habilitar un usuario tester: En el panel de administración se podrá habilitar a un usuario tester a que se registre en la aplicación, ingresando su correo.
6. Borrar un usuario tester: Desde el panel de administración se podrá borrar un usuario tester existente en la base de datos. Dicho usuario ya no podrá acceder a la aplicación, aunque sus registros (reportes de errores, grabación de sesiones) no serán borrados.

7. Gestionar versiones mutadas de una aplicación: Un usuario administrador podrá tener acceso a las distintas versiones de las aplicaciones mutadas desde el panel de administración.
8. Gestionar niveles de dificultad: El administrador podrá organizar los niveles que tendrá el sistema, agregando a ellos las aplicaciones modificadas que considere.
9. Gestionar defectos conocidos: El usuario administrador podrá agregar, editar o eliminar defectos conocidos asociados un nivel.
10. Acceder a los resultados obtenidos por un usuario tester: El administrador puede acceder a todas las sesiones, corregidas o no de un usuario tester.
11. Evaluar el trabajo de los estudiantes: El usuario administrador podrá evaluar el trabajo de los estudiantes relacionando los defectos reportados por los estudiantes con los defectos conocidos de la aplicación mutada, para luego asignarle una nota y devolución a la sesión una vez corregida.

Usuario (tester)

1. Registro de usuario: Un usuario podrá registrarse en el sistema ingresando correo electrónico, nombre de usuario y contraseña.
2. El usuario debe haber sido ingresado previamente al sistema y estar habilitado para acceder al mismo.
3. Inicio de sesión: Un usuario tester podrá acceder a la aplicación a través de la página principal.
4. Cierre de sesión: Un usuario tester podrá cerrar su sesión en la aplicación.
5. Acceder a una aplicación mutada: Un usuario podrá acceder a una aplicación mutada y ejecutar las pruebas que ha diseñado para poder encontrar los defectos de dicho mutante.
6. Reportar un incidente: Un usuario podrá reportar incidentes sobre bugs detectados en las distintas aplicaciones modificadas, utilizando para ello la herramienta Mantis.
7. Para registrarlos, se abre una instancia de Mantis en una nueva pantalla y el usuario ingresa allí los datos correspondientes.
8. Ver los incidentes que ha reportado: Un usuario podrá acceder a su panel y ver allí la evaluación de las sesiones que ha finalizado para cada nivel.

2.3. Requisitos no funcionales

1. Adecuación de la interfaz: La interfaz del sistema deberá funcionar correctamente tanto para la web como móvil.
2. Requerimientos de adecuación al entorno: El sistema deberá funcionar correctamente en los navegadores Mozilla Firefox, Google Chrome, Apple Safari y Microsoft Internet Explorer 10 (en este navegador se pueden presentar algunas diferencias frente a los otros en la interfaz de usuario)
3. Seguridad: El sistema debe garantizar aspectos de seguridad básicos en cada una de las sesiones de usuario, tanto tester como el administrador.
4. Tiempos de respuesta: el tiempo de respuesta para una petición del usuario deberá ser inferior a 0.5 segundos.
5. Carga de usuarios: El sistema deberá soportar una carga de al menos 5 usuarios concurrentes, lo cual implica las siguientes tareas

6. Aplicación a mutar: Encontrar una aplicación para generar diferentes mutaciones basándonos en algunos de los defectos descritos en la tabla que se muestra a continuación. Estas mutaciones pueden ser generadas de manera manual o utilizando una herramienta que genere mutaciones de manera automática.
7. Generación de mutaciones: Generar de manera manual mutaciones de la aplicación elegida la para la versión que fue implantada en los servidores del cliente.
8. Utilización de MantisBT: Se mantendrá el uso de la herramienta de reporte de defectos que utiliza el CES (Mantis).

2.4. Especificación de funcionalidades del sistema

En esta sección presentaremos los casos de uso desde el punto de vista de cada uno de los actores, indicando cómo impacta cada acción en los componentes del sistema. Con esto se tendrá una mejor visión de las responsabilidades de cada uno, así como de las distintas entidades que existen y las funcionalidades de cada una.

2.4.1. Usuario administrador

Aquí se presentan las acciones del usuario administrador, el cual interactúa con el panel de administración y el panel de correcciones. Éstas involucran al componente Ruby on Rails y las bases de datos PostgreSQL y MySQL.

2.4.1.1. Panel principal de administración

En la figura 2.1 se muestran las acciones del administrador en la página principal del panel de administración para luego profundizar cada una de ellas.

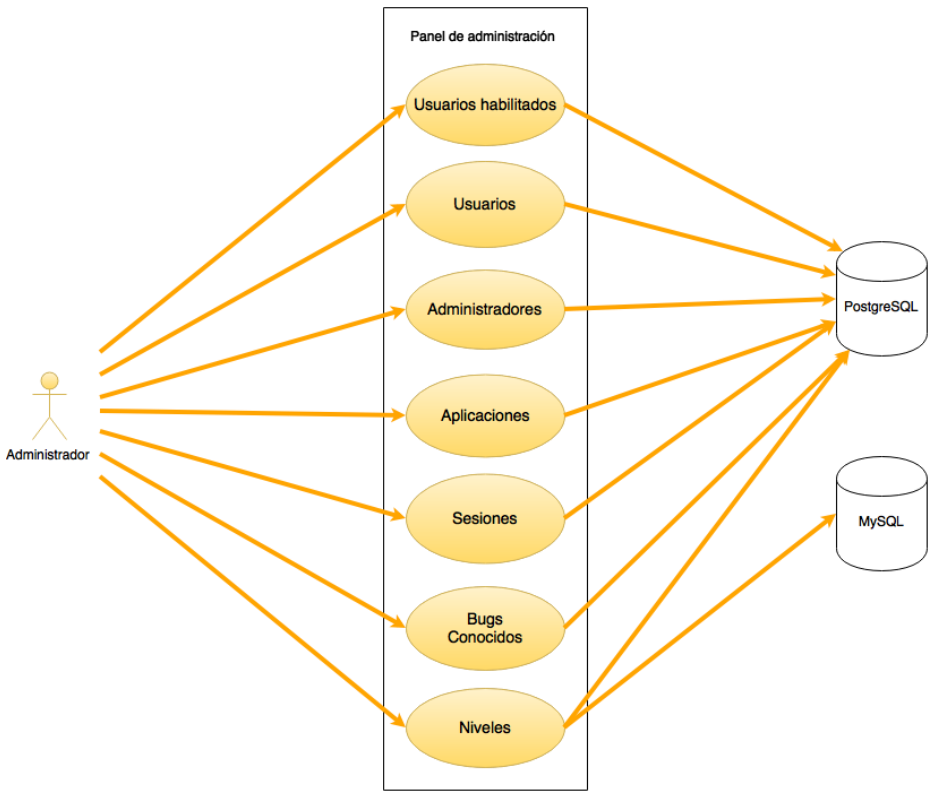


Figura 2.1: Funcionalidades de un usuario administrador en el panel de administración.

El panel de administración forma parte del proyecto Ruby on Rails y el motor de base de datos encargado de manejar los datos de este proyecto es PostgreSQL; aunque éste también se relaciona con la base de datos MySQL para poder asociar y obtener datos relacionados al proyecto Mantis. En aquellas acciones que solo se establezca comunicación con la base de datos PostgreSQL no se hará ninguna mención al respecto por ser éste el caso habitual. Se hará hincapié en los casos en que es necesario comunicarse con MySQL para explicar el detalle de esta interacción.

Administradores

Un administrador maneja otros administradores. Tiene disponible las siguientes acciones:

- Ver y filtrar la lista completa de usuarios administradores.
- Ver y editar la información de un administrador.
- Añadir un nuevo administrador y borrar uno existente.

Un usuario administrador no puede registrarse al sistema, sino que debe ser creado por otro administrador, por lo tanto al momento de crear uno nuevo se debe indicar su contraseña y confirmación de contraseña.

Usuarios habilitados

Los usuarios habilitados, también conocidos como la lista de invitados, son aquellos correos que están habilitados a registrarse al sitio como usuario Test School.

Un administrador puede:

- Ver y filtrar la lista completa de usuarios habilitados.
- Agregar y editar usuarios habilitados.

Un correo puede ser agregado de dos maneras:

- Individualmente. Para ello se le presenta un formulario con un campo para el correo.
- Múltiples. Se provee de la opción de subir un CSV con la lista de correos habilitados. Los mismos tienen que estar separados por una coma (‘,’).

Usuarios

Un usuario es un estudiante de un curso del CES, el cual es referenciado como usuario tester Test School. Cuando el estudiante se registra en el sistema pasa a ser un usuario del mismo.

Un administrador puede:

- Ver y filtrar la lista de usuarios.
- Visualizar toda la información de un usuario, esto es, su perfil y las sesiones que ha realizado.
- Eliminar un usuario. Esto implica borrar todas sus sesiones, todos sus accesos a los niveles, sus notas, sus bugs reportados y finalmente, su usuario en Mantis.

Debido a que cuando se elimina un usuario, se eliminan todos los datos relacionados con el del lado de Mantis, es que este caso de uso tiene acceso a ambas bases de datos.

Vale destacar que un administrador no puede editar información de un usuario, por ser ésta propia del estudiante.

Aplicaciones

Se entiende por aplicación a un conjunto de niveles. Un administrador puede gestionar las aplicaciones disponibles en el sistema de la siguiente manera:

- Ver todas las aplicaciones disponibles y filtrarlas.
- Crear o borrar una aplicación. En el caso de que elimine una aplicación, se destruyen todos los niveles asociados a él, y por tanto, todas las sesiones de los usuarios asociadas a dichos niveles.
- Ver y editar los detalles de una aplicación.

Niveles

El concepto de nivel se asocia a instancias que deben aprobar los estudiantes, donde se hace foco en determinadas funcionalidades de la aplicación, no en el programa completo.
Está compuesto por un escenario o realidad y asociado a un conjunto de tipos de defectos así como también a una o varias mutaciones.
Un usuario administrador puede ver la lista de niveles, filtrarlos por algún criterio, editar un nivel existente, eliminar y añadir uno nuevo.
Es importante destacar que al momento de crearse un nuevo nivel también se crea un nuevo proyecto Mantis en la base de datos MySQL, lo cual facilita la tarea del administrador.
El identificador del proyecto Mantis es guardado en el nivel de la base de datos PostgreSQL para poder así vincular los mismos conceptos de diferentes proyectos almacenados en diferentes bases de datos. Esto es de gran utilidad en el panel de correcciones donde es necesario saber a qué nivel pertenecen los incidentes reportados por un usuario en un proyecto Mantis.

Sesiones

Una sesión es una instancia que se crea cuando el usuario Test School inicia un nivel. La misma puede tener distintos estados: activa, completada y corregida.
En la página principal del panel de administración el administrador simplemente podrá visualizar las mismas y filtrarlas por estado.
En el panel de correcciones, un administrador podrá realizar correcciones sobre las mismas. Ésto se explica en la siguiente sección.

Bugs conocidos

El administrador podrá gestionar bugs conocidos, los cuales pertenecen a una mutación en particular. Estos son una parte clave dado que cuando se corrige una sesión, se comparan los defectos reportados por el estudiante con los ya conocidos.

Entre las acciones disponibles en esta sección se encuentran:

- Ver y editar bugs conocidos.
- Filtrar los bugs conocidos disponibles en el sistema
- Añadir uno nuevo o eliminar uno existente.

2.4.1.2. Panel de correcciones

En la figura 2.2 se muestran cada una de las acciones del administrador en el panel de correcciones.

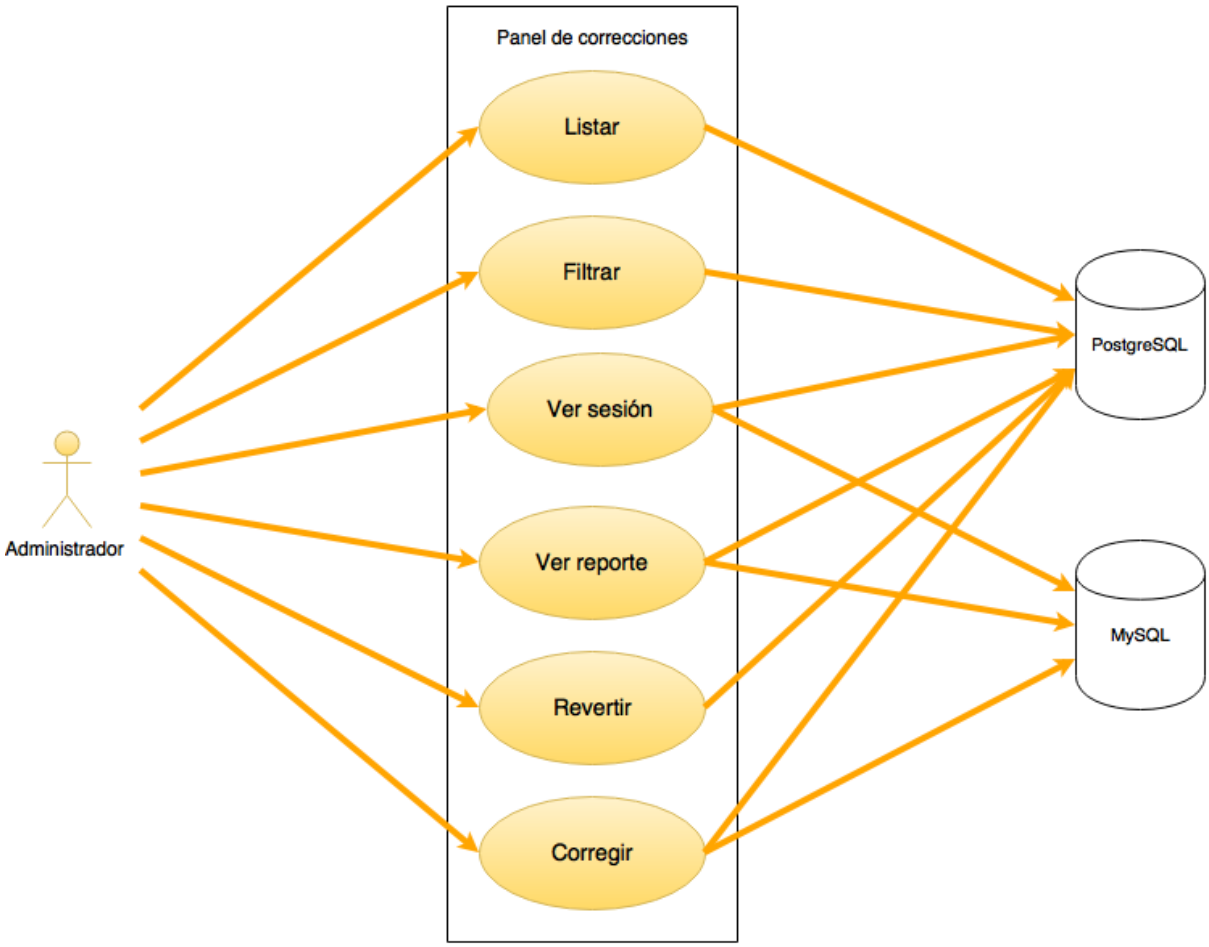


Figura 2.2: Funcionalidades de un usuario administrador en el panel de correcciones.

El panel de correcciones también pertenece al panel de administración, por lo tanto es parte del proyecto Ruby on Rails. Siendo que la principal base de datos utilizada en cada caso de uso es PostgreSQL, solo se hará mención a la base de datos en aquellos casos donde también se usa MySQL.

Listar

El administrador que ingresa al sitio de correcciones se encontrará con una lista de sesiones para corregir. Por defecto, dicha lista corresponderá a sesiones para niveles que el usuario aún no ha aprobado.
En el caso de uso "Filtrar" se detalla en profundidad el tipo de sesiones que pueden aparecer en este listado.

Filtrar

El administrador filtra sesiones que ve en el listado de acuerdo a distintas condiciones:

- Año en que el usuario realizó el curso: dado que la herramienta desarrollada tiene como finalidad instruir a usuarios en cursos que se dicten a futuro, se pensó conveniente agregar un filtro que ayude a reducir el número de sesiones a través del tiempo.
- Aplicación y nivel: si se conoce el nivel para el cual se quiere corregir una sesión, se puede utilizar este filtro para reducir la cantidad de sesiones desplegadas.
- Usuario Test School: se puede filtrar por el o los usuarios para los cuales se quiere corregir una sesión. El filtro es múltiple, esto quiere decir que más de un usuario puede ser seleccionado al mismo tiempo.

- Estado de la sesión: para este filtro existen cuatro posibles opciones que corresponden a una característica en particular de las sesiones. Éstas son:
    - Aprobadas: listará aquellas sesiones que ya han sido corregidas y cuya nota sea igual a mayor a la nota mínima de aprobación (al momento de escribir esto dicha nota corresponde con el valor 7).
    - No aprobadas: corresponden a sesiones que no han sido aprobadas (nota menor a 7).
    - Sin calificar: corresponde a aquellas sesiones para niveles que el usuario aún no ha aprobado. Cabe destacar que una vez corregida la sesión, el usuario queda habilitado a cursar nuevamente el nivel. Este estado es el valor por defecto de este filtro y se debe a que aunque los administradores tienen la opción de corregir sesiones de niveles que hayan sido previamente aprobados por el usuario, esta no será su prioridad. Es una sesión que corresponde a un nivel que el alumno ya ha cursado y ya recibió una devolución por la misma.
    - Sin calificar pero con alguna sesión corregida previamente: utilizado para aquellas sesiones sin calificar pero para un nivel que ya fue aprobado.

Ver Sesión

Esta sección permite ver información detallada correspondiente a una sesión finalizada, la cual se distribuye de la siguiente manera:

- En la parte izquierda de la misma se muestran los bugs conocidos de la mutación para la cual se realizó la sesión divididos por categoría
- A la derecha se muestran todos los bugs reportados por el usuario para esta sesión, también divididos por categoría.
- En la parte superior se muestra información estadística sobre cuántos bugs conocidos tiene dicha mutación y cuántos fueron reportados.

La principal funcionalidad de esta pantalla es que el profesor pueda relacionar fácilmente los bugs conocidos con los bugs reportados por el estudiante. Esto puede ser realizado de dos maneras, según la preferencia de quien corrija:

- Utilizando la funcionalidad de drag and drop: arrastrando un bug reportado hacia un bug conocido, asociando ambos automáticamente.
- Asociando el número identificador de un bug conocido al correspondiente bug reportado mediante una lista desplegable.

En esta pantalla también se cuenta con un botón revertir cuya acción será descrita más adelante.
Para desplegar la información ya mencionada en la pantalla se accede a ambas bases de datos: PostgreSQL para obtener los bugs conocidos de una mutación; MySQL para obtener los bugs reportados por el estudiante.

Ver Reporte

Una vez asociados los bugs reportados con los conocidos, se podrá generar un reporte el cual es un resumen de la información disponible sobre la sesión. Dicho reporte se despliega mediante el botón ¨Calificar¨.

El principal objetivo de esta sección es facilitar el proceso de corrección de una sesión, mostrando toda la información necesaria para esta instancia. Esta sección se divide en dos partes:

- La primera, meramente informativa, muestra toda la información recolectada en la sesión.
- La segunda sección dispone de un formulario para otorgar una devolución y evaluar mediante una nota.

En la primera parte (sección superior) se muestran los siguientes datos:

- Duración de la sesión: tiempo desde que el estudiante inicia la sesión hasta que la misma fue enviada para ser corregida.
- Registro de pruebas: el usuario puede subir el registro de pruebas que utilizó para testear la mutación y adjuntarlo previo a finalizar la sesión. Este registro puede ser descargado desde esta vista y analizado por los profesores para asignar una nota final.
- Bugs conocidos que fueron reportados: se refiere a la cantidad de bugs conocidos que fueron reportados por los testers, los cuales fueron relacionados en la pantalla anterior.
- Lista de bugs conocidos: se muestra cada uno de los bugs conocidos con un tick o una cruz a su derecha dependiendo de si fue reportado o no por el estudiante. También se muestra el puntaje correspondiente a este bug conocido.

En la segunda parte (sección inferior) se distribuye con:

- Entrada de texto para ingresar devolución: donde el profesor tiene la posibilidad de escribir una devolución sobre el trabajo del estudiante
- Calificación sugerida: el sistema realiza un cálculo de acuerdo a la cantidad de bugs conocidos y se sugiere una nota del 1 al 12. Cabe destacar que al momento de ingresar un bug conocido, el administrador le asigna un valor del 1 al 10 según su importancia. Para calcular la nota sugerida se pondera el valor de cada uno de los bugs conocidos. Recordar que uno de los objetivos principales de la aplicación es facilitar la tarea de corrección del profesor.
- Lista desplegable para elegir la nota final: el profesor debe elegir la nota final del estudiante para la sesión en cuestión a partir de una lista desplegable con valores del 1 al 12. Si se presiona el cuadro con la nota sugerida, el valor marcado en la lista desplegable se va a actualizar automáticamente con este valor.

Corregir

Desde la vista del reporte de una sesión, se puede finalizar la corrección seleccionando “Corregir”, lo cual desencadena varias acciones:

- Se envía un correo al estudiante notificándolo que su sesión fue corregida, el cual tiene un link a la sección de calificaciones del estudiante.
- Se lo habilita a realizar nuevamente el nivel al cual pertenece la sesión, independientemente de haber aprobado o reprobado el nivel.
- En caso de haber aprobado la sesión se habilitan los niveles hijos del nivel aprobado. Recordar que los niveles tienen una jerarquía en forma de grafo.

Revertir

Esta funcionalidad surgió a partir de la inquietud de los profesores que expresaron que es común el caso en que los estudiantes solicitan que su sesión sea reabierta una vez que ya fue entregada para corrección. Esto los habilita a entregar nuevamente su trabajo, siendo este último el que será corregido por los profesores.

Para contemplar esta funcionalidad, existe un botón “Revertir” en la vista de la sesión (en la parte inferior derecha) que sólo aparece visible mientras el profesor pueda corregir la sesión. Esta acción básicamente revierte una sesión finalizada por un estudiante y permite que el mismo pueda continuar probando y entregar nuevamente cuando lo considere apropiado (dentro de los límites que dispongan los profesores).

2.4.2. Usuario Test School

Figura 2.3: Funcionalidades de un usuario Test School que se relacionan únicamente con la base de datos PostgreSQL.

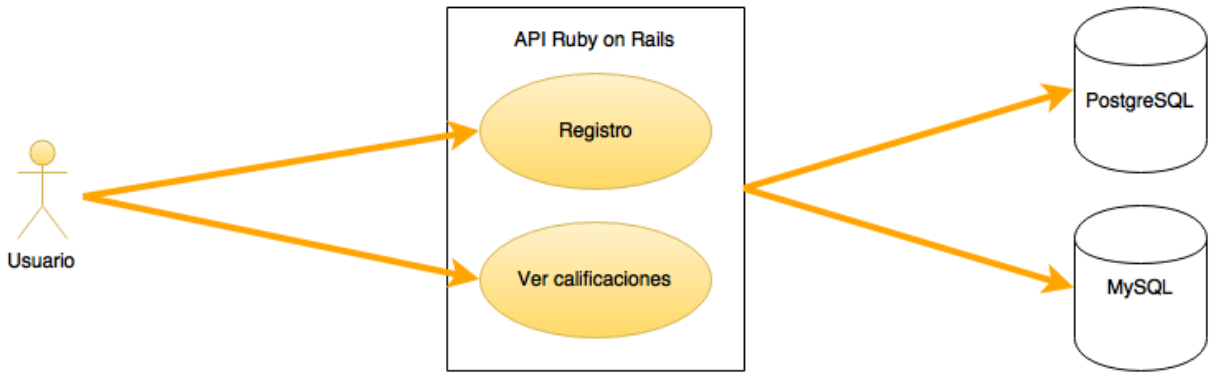


Figura 2.4: Funcionalidades de un usuario Test School que se relacionan con ambas bases de datos.

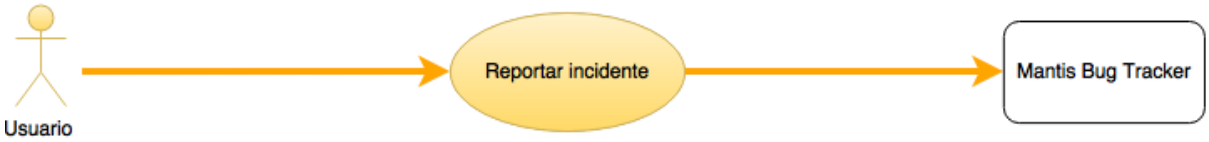


Figura 2.5: Funcionalidad de un usuario Test School con MantisBT.

Registro

El usuario podrá registrarse en el sitio siempre y cuando su email se encuentre dentro de la lista de invitados. Dicha validación se realiza en el backend del sistema y de no resultar exitosa, se le notificará al usuario que no puede registrarse, quedando éste invalidado.
El usuario estará habilitado a ingresar su email, nombre real, nombre de usuario y contraseña. Si el registro es exitoso se procederá a crear un usuario en Mantis, por lo que la API se conectará con este otro proyecto, utilizando los mismos datos para crearlo.
El usuario quedará registrado en ambas bases de datos.

Recuperación de contraseña

El usuario envía su correo al backend donde en primera instancia se verifica que el mismo exista para tomar acción. De no ser así se despliega un mensaje de error notificando que ese correo no existe en el sistema.
Un mensaje es enviado al correo del usuario con los pasos a seguir para cambiar su contraseña. Un link lo redirigirá a la pantalla para ingresar su nueva contraseña.

Inicio de sesión

En esta pantalla el usuario ingresa sus credenciales (correo y contraseña) para autenticarse en la API y poder acceder a los distintos cursos y niveles disponibles.
Podrá además acceder a datos personales como información de su usuario y sus calificaciones, los cuales serán explicados en los casos de uso correspondientes.
Aclaración: este inicio de sesión es independiente del usuario Mantis, por lo cual no se inicia sesión automáticamente en esa plataforma.

Cerrar sesión

El usuario termina su sesión y consecuentemente deja de autenticarse contra la API, siendo redirigido a la pantalla de inicio de sesión. Su token es destruido en la base de datos por lo que ya no es válido y no podrá acceder a pantallas como las de los niveles, su perfil o sus calificaciones.
Es de destacar que la sesión en Mantis no es cerrada ya que se trata de aplicaciones distintas y sesiones independientes.

Ver y editar perfil

En esta pantalla el usuario tendrá a disposición la información que ingresó al momento de registrarse. Allí verá su correo, nombre y nombre de usuario y tendrá la posibilidad de editarlos. También podrá agregar o editar una foto de perfil para que los profesores puedan identificarlo fácilmente.

Ver calificaciones

En esta pantalla se muestran las calificaciones correspondientes a las correcciones que el usuario recibió para cada uno de los niveles, de la cuales se detalla su nota y devolución.
Para facilitar la navegabilidad del usuario, las calificaciones se agrupan por aplicación y a su vez dentro de cada una de ellas, se pueden desplegar u ocultar los niveles corregidos junto con la correspondiente calificación otorgada.
Por defecto, la primera aplicación y el primer nivel se despliegan para que el usuario pueda observar inmediatamente esa información.

Listar aplicaciones y niveles

Una vez que el usuario se ha autenticado, se lo dirige a una pantalla donde se muestran todos los niveles agrupados según la aplicación a la que pertenecen.
Los niveles presentarán distintos estados según si el usuario ya los ha cursado, si lo está haciendo o si todavía no ha comenzado una sesión.
Si un usuario no está habilitado a cursar un nivel, el mismo se mostrará deshabilitado (opaco). Por el contrario, si el nivel está habilitado el usuario podrá ingresar a la pantalla de detalles del mismo. Aquí se presentan 3 posibilidades:

- Que el nivel haya sido desbloqueado al aprobar un nivel anterior y que no tenga sesión iniciada, en cual caso tendrá un indicador de "Nuevo".
- Que el nivel haya sido desbloqueado al aprobar un nivel anterior, pero que el usuario tenga una sesión no aprobada para dicho nivel. El mismo estado tendrán aquellos niveles iniciales de cada aplicación.
- Que el usuario tenga una sesión aprobada para dicho nivel, en cual caso verá una barra de estado en color verde y con una marca de aprobación.

Ver nivel

Una vez que un usuario elige un nivel e ingresa se lo dirige a la pantalla en detalle de dicho nivel, donde se presenta al usuario una descripción de la realidad y el objetivo del nivel.
Esto incluye: qué se busca testear dentro de la aplicación y las posibles técnicas de testing que se espera que aplique.
Si el usuario no tiene sesión o su última sesión ya fue corregida para este nivel se habilita un botón para iniciar una nueva. En caso que tenga una sesión activa podrá retomarla. Finalmente, si el usuario tiene una sesión pendiente de corrección no tendrá ninguna opción posible; simplemente se despliega un texto indicando que debe esperar a que su sesión sea corregida por alguno de los profesores.

Comenzar nivel

Como ya se mencionó en el caso de uso anterior, cuando un usuario no tiene una sesión (o su última sesión finalizada) podrá iniciar una nueva. Aquí se realiza una petición a la API, la cual tiene como resultado la creación de una nueva sesión para dicho nivel con estado activa. Cuando se crea, el usuario es direccionado hacia una página donde podrá observar la aplicación a testear, la cual se encuentra embebida en un elemento iframe de HTML.
En esta pantalla además se presentan opciones al usuario las cuales serán detalladas en los casos de uso que presentan a continuación.

Reanudar nivel

Como se mencionó en el caso de uso "Ver nivel", el usuario tendrá la posibilidad de reanudar su nivel. Esto le permite cerrar la pestaña del explorador donde se encuentra o cambiar de computadora por ejemplo, sin perder el progreso que tenía en dicha sesión.

Subir registro de pruebas

Una vez que el usuario se encuentra en la pantalla donde se le presenta la aplicación a testear, tendrá como opción subir un registro de las pruebas que realizó. Para ello tendrá un botón en la parte inferior izquierda (color amarillo), el cual desplegará un mensaje que le brinda la opción de subir un archivo (si son muchos los podrá comprimir dentro de un archivo .zip, u otro archivo que considere conveniente).
Si el administrador lo considera pertinente, podrá subir un archivo de referencia para dichas pruebas el cual podrá ser descargado por el usuario desde un link otorgado en dicho mensaje.

Crear, ver y borrar notas

Continuando en la misma pantalla, el usuario tendrá la posibilidad de crear, editar y eliminar notas que funcionan como recordatorios en caso que quiera continuar su sesión más tarde.
Las notas se encuentran en un panel lateral que puede ser desplegado mediante un click en la flecha que se encuentra en el lado derecho de la pantalla.
Estos recordatorios son propios de la sesión y no del nivel, por lo que si el usuario desea comenzar una nueva sesión en el futuro no verá estos comentarios.
Así mismo, los profesores no están habilitados a verlos, ya que no se consideran parte de la corrección, sino simplemente una ayuda para el usuario.

Finalizar nivel

Dentro de la pantalla principal donde se corre la aplicación que se testea, existe un último caso de uso que le permite al usuario finalizar la sesión y enviarla para que sea corregida una vez que considera que ha completado y reportado los suficientes errores (o todos ellos). Para esto cuenta con un botón en la parte inferior izquierda de la pantalla.
Si el nivel cuenta con un registro de pruebas habilitado para descargar y utilizar como guía y el usuario no ha subido el suyo todavía, se procederá a notificarle que le lo haga. De todas maneras se le permitirá entregar dicha sesión, teniendo en cuenta que su nota final será afectada negativamente.
El usuario es redirigido a la pantalla de los niveles donde quedará a la espera de la corrección. Si accede al mismo nivel, verá como no tiene acción para empezar o reanudar una sesión, sino que verá un mensaje que le informará que cuenta con un sesión que está pendiente de corrección.

3. Estado del arte

3.1. Introducción

Este capítulo corresponde al marco referencial del proyecto de grado donde se expone los tópicos principales y secundarios referentes al objetivo del mismo. Abarca los conceptos necesarios para comprender los temas desarrollados en el informe y la investigación para saber si existe alguna herramienta en el mercado que satisfaga los requisitos.
El capítulo se distribuye de la siguiente manera:
En primer lugar se explica el concepto de calidad de software, el cual es referenciado a lo largo del documento porque está íntimamente relacionado con el testing de software.
En la sección 3.3 se profundiza en la definición de "Testing de software", presentando los puntos de vista de diversos autores y se introducen algunos conceptos básicos.
En la sección 3.4 se plantean algunas clasificaciones de pruebas que existen en la literatura, haciendo énfasis en la de Whittaker [1], que fue utilizada para este proyecto de grado.
En la sección 3.5 se describen las estrategias del testing funcional: exploratorio y planificado, junto técnicas de diseño de casos de prueba. Se continúa (sección 3.6) con la presentación del proceso de testing que aplica el CES, explicando en mayor detalle aquellas actividades que deberán realizar los estudiantes al momento utilizar la plataforma.
En la sección 3.7 se hace una breve introducción a la noción de mutación y cómo se utilizan para evaluar la calidad de los casos de prueba, en particular los propuestos por los estudiantes.
Finalmente, (sección 3.8) se hace referencia a la investigación de mercado, la literatura disponible que fue revisada y analizada, así como las herramientas encontradas, de las cuales se brinda una breve descripción.

3.2. Calidad de Software

La calidad del producto, junto con la calidad del proceso, es uno de los aspectos más importantes actualmente en el desarrollo de software.
La familia de normas ISO/IEC 25000 (basadas en ISO/IEC 9126 y en ISO/IEC 14598) proporciona una guía para el uso de la nueva serie de estándares internacionales llamada Requisitos y Evaluación de Calidad de Productos de Software (SQuaRE).

En particular existen dos normas dentro de la División de Modelo de Calidad donde se presentan modelos de calidad detallados incluyendo características para calidad interna, externa y en uso del producto software. Actualmente esta división se encuentra formada por:

- ISO/IEC 25010 - System and software quality models: describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.
- ISO/IEC 25012 - Data Quality model: define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran almacenados de manera estructurada y forman parte de un Sistema de Información.

Según el estándar ISO/IEC 25010 [2] la calidad un producto de software se puede interpretar como el

grado en que dicho producto satisface los requisitos de sus usuarios aportando para esto un determinado valor.

Figura 3.1: Características de calidad.

El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las siguientes ocho características de calidad (ver figura 3.1):

- **Adecuación Funcional**: Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.
- **Eficiencia de desempeño**: Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
- **Compatibilidad**: Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.
- **Usabilidad**: Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.
- **Fiabilidad**: Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.
- **Seguridad**: Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.
- **Mantenibilidad**: Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.
- **Portabilidad**: Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

3.3. Testing de software

Para definir el concepto de testing de software se analizarán varios enfoques dados por distintos autores, para observar los puntos que tienen en común y las diferencias que presentan.

Según C. Kaner [3], el testing es una investigación técnica y empírica orientada a proporcionar información sobre la calidad de un producto de software para un actor o usuario.

En esta definición resalta el aspecto empírico del proceso de testing, ya que se basa en la experimentación, en donde se le da información sobre la calidad de un producto o servicio a las personas interesadas. Los testers son los encargados de comunicar la información obtenida de forma

adecuada, para que las personas interesadas en los resultados tomen las decisiones correspondientes respecto al software.
Es necesario establecer los objetivos del testing para determinar cuál será el enfoque que se le dará. Si el objetivo del testing es encontrar la mayor cantidad de errores posibles, entonces seguramente el testing se enfoque en las áreas más complejas del software o las que han tenido más defectos. Si el objetivo es dar seguridad a los usuarios, entonces seguramente se enfoque el testing en los escenarios más usados por los clientes.
Para Kaner, las pruebas no se reducen a ejecutar el software, sino que el tester investiga el software empíricamente y desde un punto de vista técnico. Típicamente, por ejemplo, el tester revisará la especificación al comenzar a diseñar pruebas de sistema. Si encuentra incongruencias en la especificación deberá determinar si se trata de un fallo, lo que requiere revisar otra información o contactar a ciertas personas: el tester debe investigar para tener toda la información relevante respecto al software.

De acuerdo a la definición dada por el SWEBOK, el testing de software consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto finito de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado [4] y [5].
Según Myers [6], el testing es el proceso de ejecutar un programa con la intención de encontrar errores.
Este autor destaca la importancia de agregar algún valor al producto cuando se lo prueba. Agregar valor significa elevar la calidad o credibilidad del programa, lo cual implica, entre otras cosas, encontrar y remover errores. Por lo tanto, no se prueba un programa para asegurar que funciona correctamente, sino que hay que partir de la asunción de que el programa contiene errores, y es necesario probarlo para encontrar la mayor cantidad posible.

Analizando las definiciones presentadas, se puede inferir que el testing es una actividad cognitiva y no mecánica ni repetitiva que involucra varias funciones mentales como el lenguaje, la imaginación, percepción, entre otros.

A partir de la información obtenida del testing se pueden tomar decisiones. Las decisiones pueden ser desde cuándo liberar un producto a producción, conociendo los riesgos que esto implica, hasta cómo mejorar las diferentes áreas dentro de la empresa. En definitiva, el testing es un agente de cambio, lo importante es interpretar la información obtenida para que todos los actores puedan actuar en forma oportuna donde sea necesario.

3.3.1. Verificación y Validación

Según la definición de IEEE [7] la verificación consiste en el proceso de evaluar un sistema o componente para determinar si el producto construido durante una fase de desarrollo satisface las condiciones especificadas al comienzo de dicha fase.
En otras palabras, la verificación es una actividad desarrollada por ingenieros teniendo en cuenta un modelo del programa y el programa en sí.

Según la definición de IEEE [7], la validación consiste en el proceso de corroborar que el programa satisface las expectativas del usuario. Por este motivo, la evaluación la debe realizar el usuario teniendo en cuenta lo que él espera del programa y el programa en sí.

En general en la bibliografía, la prueba es tomada como una parte del proceso de Verificación y Validación. Según el enfoque de SWEBOK [4], se define que la salida observada de la ejecución del programa puede ser comparada mediante:

- Prueba para la verificación: El comportamiento observado es revisado contra las especificaciones.
- Prueba para la validación: El comportamiento observado es revisado contra las expectativas del usuario

3.3.2. Error, defecto y falla

Error

El error (error) es la ocurrencia de la condición inválida o valor incorrecto en el sistema. La interacción entre el defecto y un estímulo es lo que produce el error [8]. Un error es la parte del estado del sistema, que es susceptible de ocasionar un fallo. Un error es entonces la manifestación de un defecto en el sistema.

Existen dos estados posibles de un error, latente o detectado. Un error es latente mientras que no ha sido reconocido como tal. Puede ser detectado por mecanismos de detección de errores que analizan el estado del sistema, o por los efectos del error sobre el sistema.

Defecto

Un defecto o bug es una anomalía física (defecto, imperfección) en el software, en el hardware o en los datos que tiene el potencial de causar errores y fallos [8]. Los defectos son las causas de los errores, pero no todo defecto lleva a un error. Un defecto se dice activo cuando produce un error.

Falla o incidente

Una falla (failure) de un sistema se da cuando el mismo no se comporta como está especificado [8].

Un error cometido por una persona se convierte en una falla en un sistema de software.

Si esta falla no es detectada y corregida, se propaga como un defecto en el código ejecutable.

Cuando el código es ejecutado, esta falla puede verse como una anomalía en el sistema (una desviación en la especificación o el comportamiento esperado).

Múltiples errores se pueden originar de un defecto. Si se ve al sistema como un conjunto de componentes, los errores en un componente pueden transformarse en fallas, que originan defectos que se propagan a más componentes a través del sistema (ver Figura 3.2). Las fallas o incidentes de software se denominan, por razones históricas, bugs.

Figura 3.2: Diagrama "Error-Defecto-Falla".

En el anexo “Categorización de incidentes” se presenta la categorización de incidentes proporcionada por el CES, que fue utilizada en este proyecto.

3.3.3. Caso de prueba o test

A continuación se presentan diversas definiciones de casos de prueba [9]:
Según el estándar IEEE 610 (1990), un caso de prueba se define como:

> “(1) Un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados, desarrollados con un objetivo en particular como ejercitar un programa en particular o comprobar un requisito en particular.”
>
> “(2) (IEEE Std 829-1983) Documentación especificando entradas, resultados esperados, y un conjunto de condiciones de ejecución para probar un item.”

Según Ron Patton [10]:

> “Los casos de prueba son entradas específicas que se ingresan y los procedimientos que se siguen cuando se prueba el software.

Para Boris Beizer [11], una prueba se define como:

> “La secuencia de una o más subpruebas que se ejecutan de esa manera ya que la salida o estado final de una subprueba es la entrada o estado inicial de la siguiente.. La palabra “prueba” se utiliza para incluir subpruebas, una prueba en sí y conjuntos de pruebas”

Según Bob Binder [12] un caso de prueba:

> “Especifica el estado previo a la prueba sobre el sistema bajo prueba (IUT, Implementation Under Test) y su ambiente, las entradas o condiciones y los resultados esperados. Los resultados esperados especifican la salida que el IUT debería generar a partir de las entradas ingresadas. Esta especificación incluye mensajes generados por el IUT, excepciones, valores devueltos, estado resultante del IUT y su ambiente. Los casos de prueba también especifican condiciones iniciales y finales del IUT y su ambiente.”

En la práctica, varios elementos son referidos como casos de prueba aunque no están debidamente documentados.

Brian Marick utiliza un término relacionado para describir las pruebas levemente documentadas: idea de prueba. Según él, una idea de prueba es:

> “Un breve resumen de algo de debe ser probado.“

De acuerdo a Cem Kaner, un caso de prueba es:

> “Una pregunta que se le hace al programa con el objetivo de conseguir información.”

Un conjunto de pruebas, test suite o “conjunto de validación”, es una colección de casos de prueba cuya intención es probar un programa de software para mostrar que tiene un determinado comportamiento.

Al momento de seleccionar los casos de prueba que deben formar parte de este conjunto, el objetivo debe ser maximizar el número de los errores encontrados por un número finito de los casos de prueba.

3.3.4. Oráculo

Según la definición del SWEBOK [4], un oráculo es un agente humano o mecánico que decide si un programa se comporta de manera correcta o no en una prueba determinada, y emite un veredicto si aprueba o no: “pass” si pasa, “fail” si falla.

Según Kaner[13], un oráculo, para ser completo, tiene que tener:

- Un generador que provee predicciones y resultados esperados para cada test.
- Un comparador para comparar los resultados esperados contra los obtenidos.
- Un evaluador que determina si los resultados de la comparación están los suficientemente cerca como para pasar la prueba.

Por ejemplo: se pueden generar predicciones basándose en resultados previos de otras pruebas; versiones anteriores del programa o el programa de algún competidor. Éstas se pueden generar manualmente, utilizando una herramienta que alimente la entrada del programa de referencia y capture la salida, o incluso combinando el testing automático y manual.

También se pueden generar predicciones basadas en especificaciones, requisitos regulatorios u otras fuentes de información que implican que una persona evalúe la información y generar así la predicción.
El oráculo más común es el oráculo entrada/salida, que especifica la salida esperada para una entrada específica [14].
Dado que un programa puede fallar en múltiples maneras, el problema de interpretar los resultados es complejo. Es necesario poder decidir cuándo un programa falla, dado un conjunto de datos de entrada y una salida esperada. Determinar la corrección de los resultados se conoce como el “Problema del Oráculo” y surge cuando no puede definirse el procedimiento de decisión.

3.4. Tipos de testing

En la literatura existen diversas clasificaciones según la forma de abordar el testing y éstas varían acorde al autor.
En este proyecto de grado se utilizará la clasificación de tipos de testing dada por Whittaker en [1], donde se propone clasificar las pruebas según:

- Opacidad (visibilidad sobre el código): funcional o estructural. caja blanca ó caja negra.
- Objeto que se prueba: unitaria, de integración ó del sistema.

Cabe destacar que existen otras clasificaciones en la literatura, por ejemplo: según el aspecto que se prueba: pruebas funcionales y no funcionales (performance, carga, estrés); ó según el grado de automatización de las pruebas, entre otros.
En este trabajo se hace énfasis en el testing funcional, debido a que es el tipo de testing que los usuarios del sistema van a poder ejecutar dada la visibilidad sobre el código y los aspectos que se

prueban. Sobre el testing estructural sólo se dará una breve introducción sin profundizar en las técnicas relacionadas.

3.4.1. Por opacidad

3.4.1.1. Testing estructural o de caja blanca

Las técnicas de caja blanca se enfocan en examinar la estructura del código de un programa para saber si éste tiene el funcionamiento esperado.
Este tipo de pruebas toman en cuenta el mecanismo interno de un sistema o un componente [7] y también es conocido como testing estructural o testing de caja de cristal [11], debido a que estos nombres connotan la visibilidad sobre el producto de software, especialmente la lógica y la estructura del código.

3.4.1.2. Testing de caja negra

Las técnicas de caja negra se enfocan en analizar la especificación del software a testear pero no su código; el cual se considera una “gran caja negra” a la cual el tester no debe tener acceso.
El tester solo puede saber los datos de entrada al sistema y la salida generada a partir de ésta, como puede verse en la figura 3.3. Los casos de prueba son elaborados a partir de los requisitos, funcionalidades y casos de uso, por lo cual el tester debe basarse en esa información para verificar si la salida generada por el sistema es la esperada, dada una determinada entrada.

Esta estrategia se basa en seleccionar los casos de prueba analizando la especificación o modelo del programa, en lugar de su implementación [15].

Según Kaner, es un enfoque que consiste en diseñar los casos de prueba y ejecutar el testing sin conocimiento del código (o sin uso del conocimiento del código).
Los testers diseñan las pruebas a partir de su conocimiento sobre las características y necesidades del usuario del sistema, el dominio de aplicación del negocio, el mercado, los riesgos y el ambiente.
El tester de caja negra se transforma en un experto en las relaciones entre el programa y el mundo en el que ese programa se ejecuta.
Y agrega:
¨...Algunos autores restringen este concepto a testear exclusivamente contra las especificaciones. Nosotros no”. (traducción de los autores)

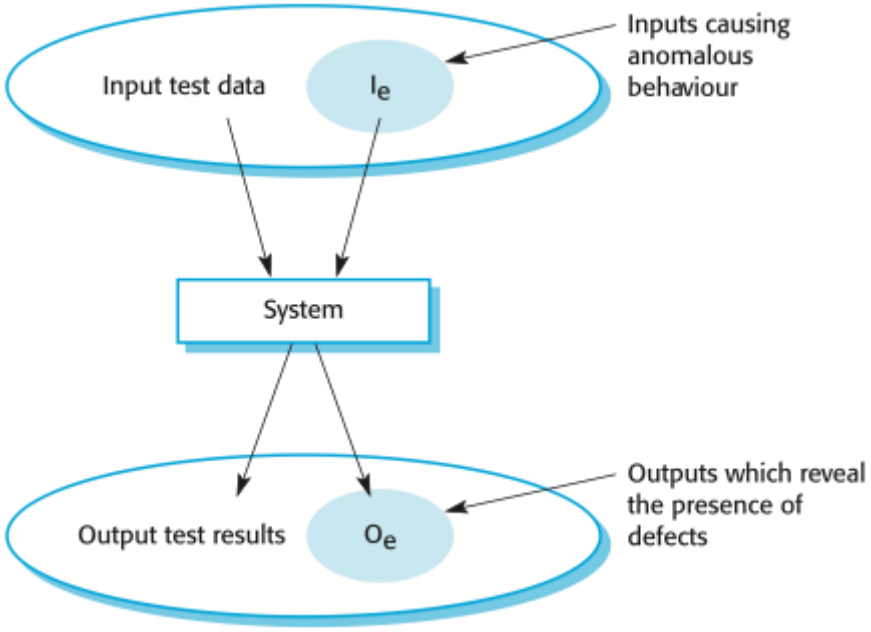


Figura 3.3: Aplicación de técnica de caja negra. Extraído de "Software Engineering", 7ma edición, capítulo 23.

3.4.2. Por objeto bajo prueba

3.4.2.1. Pruebas unitarias

Por su opacidad se las considera de caja blanca y su objetivo es probar unidades de software o hardware, o grupos de unidades relacionadas.

3.4.2.2. Pruebas de integración

Por su opacidad se las considera de caja blanca y negra. Su objetivo es evaluar la interacción entre ciertos componentes de hardware, software o ambos.

3.4.2.3. Pruebas funcionales y del sistema

Por su opacidad se las considera de caja negra. Permiten a los testers examinar el diseño a alto nivel y la especificación de los requisitos para así planear los casos de prueba que verifiquen que el código cumple su función correctamente. Las pruebas funcionales verifican que una determinada funcionalidad cumpla con los requisitos especificados.
Las pruebas de sistema apuntan a evaluar en forma integrada y completa el sistema, verificando que se cumplen los requisitos especificados.

3.4.2.4. Pruebas de aceptación

Por su opacidad se las considera pruebas de caja negra. Después que las pruebas funcionales y de sistema son ejecutadas, el producto es enviado al cliente y éste ejecuta pruebas de aceptación basándose en sus criterios de aceptación y sus expectativas de cada funcionalidad.

3.4.2.5. Pruebas de regresión

Por su opacidad se las considera de caja negra y blanca. Estas pruebas consisten en una selección de pruebas realizadas previamente, que son ejecutadas nuevamente en el sistema o en un componente del mismo, para verificar que ciertas modificaciones no causaron efectos no deseados y que el sistema sigue cumpliendo con los requisitos especificados.

3.5. Testing funcional

3.5.1. Descripción

Myers define el testing funcional como el proceso para intentar encontrar diferencias entre un programa y sus especificaciones [6].
Es visto normalmente como una actividad de caja negra y la estructura interna del programa raramente es tomada en cuenta [16].
Una especificación consiste en una descripción del comportamiento del programa desde el punto de vista del usuario final, y es analizada para derivar los casos de prueba.

Una de las consideraciones al momento de abordar el testing de un programa es que no se puede probar completamente; por lo que es necesario tomar decisiones respecto a cómo se van a diseñar los casos de prueba.

Los principales desafíos que se presentan son:

- Seleccionar casos de test que sean eficaces y que detecten incidentes sin requerir un número excesivo de tests.
- Buscar las combinaciones de entradas y estados del sistema que tengan la más alta probabilidad de encontrar incidentes dentro de un conjunto inmenso que no podemos testear exhaustivamente

Para este tipo de testing se distinguen dos estrategias de pruebas: testing exploratorio y testing planificado.

3.5.2. Testing exploratorio

El término “testing exploratorio” fue introducido por Cem Kaner. Se refiere al aprendizaje del producto, diseño, ejecución de pruebas e interpretación de los resultados como actividades que se complementan y se realizan simultáneamente a lo largo de un proyecto [17].

El lugar de existir un guión, como en el testing planificado, el tester tiene un modelo mental de lo que quiere probar, que se denomina misión, y con ese objetivo, se enfrenta al producto, diseña y ejecuta pruebas simultáneamente (como se ilustra en la figura 3.4), y aprende de los resultados obtenidos para diseñar nuevas y mejores pruebas [18].

Se refiere a un estilo de testing de software que enfatiza la libertad individual y responsabilidad del tester para optimizar constantemente el valor de su trabajo.

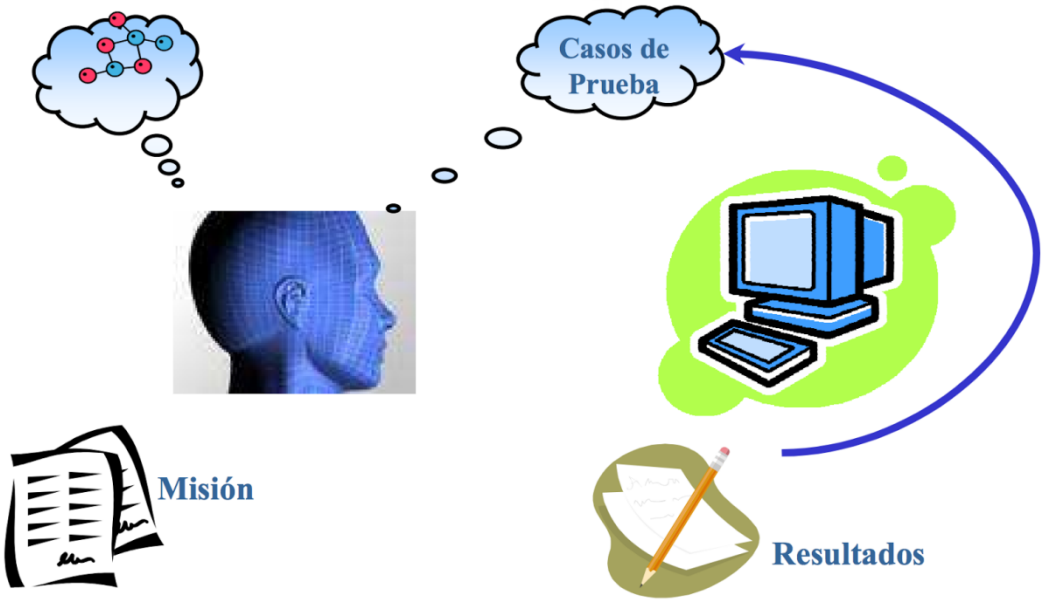


Figura 3.4: Extraído de las notas de TVS [53].

El testing exploratorio puede ser aplicado en cualquier situación donde no sea obvio cuál es la próxima prueba que se debe realizar. También se utiliza cuando se requiere obtener retroalimentación rápida de cierto producto o funcionalidad, se necesita aprender el producto rápidamente, se quiere investigar y aislar un defecto en particular, se quiere investigar el estado de un riesgo particular, o se quiere evaluar la necesidad de diseñar pruebas para esa área.
Una estrategia básica para realizar testing exploratorio es tener un plan de ataque general, pero permitirse desviaciones por periodos cortos de tiempo. Cem Kaner llama a esto el principio del "tour bus", las personas bajan del bus y conocen los alrededores. La clave es no perderse el tour entero [18].
En general, el testing exclusivamente exploratorio requiere testers con mucha experiencia. Como ventaja se encuentra que es barato y rápido; como inconveniente, que según algunos autores, produce escasa documentación y no facilita las medidas de cubrimiento [19].

En contraste con el testing planificado, en el exploratorio [17]:

- Se diseñan las pruebas según la necesidad.
- Se ejecuta la prueba al momento de diseñarla o se reutiliza más tarde.

Sin embargo, no planificar no significa que no haya que prepararse. Frecuentemente se diseñan materiales de soporte y se utilizan en reiteradas ocasiones durante el testing. Éstos incluyen:

- Conjuntos de datos.
- Listas de fallas.
- Tablas de combinaciones.

La principal diferencia entre el testing exploratorio y el planificado es que el tester es responsable de manejar su tiempo. Esto implica que en cualquier momento puede:

- Reutilizar pruebas anteriores.
- Crear y ejecutar nuevas pruebas.
- Crear artefactos de soporte para el testing, como listas de modos de fallas.
- Llevar a cabo investigaciones del contexto que puedan guiar el diseño de nuevas pruebas.

3.5.2.1. Testing basado en sesiones

Esta es una metodología propuesta por James y John Bach [51] para medir y manejar el testing exploratorio que apunta a: descubrimiento rápido de defectos y reportes sobre las pruebas y resultados obtenidos.

Debido a que el testing exploratorio no es guionado, su efectividad depende de varios factores intangibles: la habilidad del tester, su intuición, experiencia y capacidad de seguir sus instintos. Por lo tanto, uno de los principales inconvenientes es obtener métricas y registros de las pruebas realizadas.

El testing exploratorio basado en sesiones surgió como una solución para lidiar con estos aspectos intangibles, estructurar de una manera más ordenada el trabajo y poder gestionarlo.

Es necesario en primer lugar identificar las misiones, que constituyen el propósito de las pruebas. Luego se organizan sesiones, que se consideran la unidad básica de trabajo y se documentan.

Por cada misión pueden llevarse a cabo una o varias sesiones, de las cuales se documenta la siguiente información:

- Especificación de misión.
- Meta o agenda de la sesión.
- Áreas de cobertura.
- Nombre del tester/s.
- Fecha de inicio y duración.
- Notas sobre los pasos seguidos: incluyen el itinerario, que se establece a partir de la misión y eventualmente, algunas de las heurísticas a ser utilizadas.
- Bugs encontrados.
- Incidentes: preguntas abiertas, inquietudes sobre el proyecto o el producto.
- Métricas (detalladas abajo).

Las métricas obtenidas a partir de sesiones son el medio principal para expresar el estado del proceso de testing exploratorio. Éstas involucran:

- Cantidad de sesiones completadas.
- Cantidad de problemas encontrados.
- Áreas de funcionalidades cubiertas.
- Porcentaje de tiempo de sesión utilizado para preparar la sesión.
- Porcentaje de tiempo de sesión utilizado para diseño y ejecución de pruebas.
- Porcentaje de tiempo de sesión utilizado para investigar investigación y reporte de defectos.
- Porcentaje de tiempo invertido en Misión versus Oportunidad

Se espera que los estudiantes apliquen este tipo de testing exploratorio para probar el sistema, documentando las sesiones aplicadas y los resultados obtenidos. A continuación, se presenta un ejemplo de reporte de sesión (ver tabla 3.1).

GUIÓN

Crear un perfil de prueba y una lista de riesgos para DecideBien
#AREAS
DecideBien
OS | Win98
Versión | 1.2
Estrategia | Exploración y Análisis
COMIENZO

4/16/01 11:15pm
TESTER

Jonathan Bach
Tim Parkman
DETALLE DE TAREAS

#DURACIÓN
corta
#DISEÑO DE PRUEBAS Y EJECUCIÓN
100
#INVESTIGACIÓN DE DEFECTOS Y REPORTES
0
#CONFIGURACIÓN DE SESIÓN
0
#GUION VS. OPORTUNIDAD
100/0
ARCHIVOS DE DATOS
---
tco-jsb-010327-A.txt
rl-jsb-010327-A.txt
NOTAS DE PRUEBA

Tim y yo recorrimos la tabla de contenidos y el índice de la guía de usuario para crear el siguiente TCO:
Sistemas Operativos:
Win98
Win2000
Funcionalidades Generales:

Instalación
Manual de Usuario
Ayuda en Línea
UI
Preferencias
Ventana Prominente:
Ventana de la Tabla Principal
Ventana de Criterios de Peso
Ventana de Opciones de Calificaciones
Ventana de Documentos
Ventana de Arranque
Manejadores:
Consejero DecideBien
Editor de Etiqueta Categoría
Editor numérico
Manejador de Escenario

Generador de Reporte
Construcción rápida
Elementos de la Decisión:
Elementos del Lenguaje
Preferencias
Indicadores de Sensibilidad
Ponderación
Opciones de Entrada
Tabla de Decisión
Opciones de Rating
Línea de Base
Interoperabilidad:
OLE
Importar / Exportar
Gráficos
Impresiones
DEFECTOS

#N/A
INCIDENCIAS

#INCIDENCIAS
El manual menciona diferentes plataformas (Win 3.1, WFW, y WinNT 3.51) y no menciona Win2000. Creemos que es importante testear Win 2000 y sobre eso, las Viejas O ya no son significativas.
INCIDENCIAS
Hicimos este análisis en Win98. No tengo datos que sugieran que las funcionalidades puedan ser distintas en otros sistemas operativos, pero no estoy seguro de ello.

Tabla 3.1: Ejemplo de reporte de sesión extraído de "Satisfice Inc.", sitio web de James Bach [20].

3.5.3. Testing planificado

Esta estrategia de testing tiene como premisa contar con un guión para las pruebas que se realizan sobre un sistema, por lo cual deben ser diseñadas con anterioridad [17].
En este contexto el guión no es más que un caso de prueba y por lo tanto su definición coincide con la descrita en la sección 3.3.3.

A diferencia del testing exploratorio, en el planificado el diseño y la ejecución de las pruebas son independientes, por lo tanto, estas actividades pueden ser realizadas por distintas personas (o máquinas) y en momentos distintos, como ilustra la figura 3.5.

Figura 3.5: Testing planificado.

A pesar de requerir una fuerte inversión de tiempo, el testing planificado cuenta con varios beneficios:

- Análisis cuidadoso al diseñar cada prueba.
- Las pruebas pueden ser revisadas por otros autores (stakeholders).
- Reusabilidad.
- Comprensión del conjunto de casos de prueba.
- Si se considera el conjunto de prueba lo suficientemente comprensivo, se pueden calcular métricas como el porcentaje de tests completados.

3.5.4. Técnicas de diseño de casos de prueba

Uno de los principales desafíos al momento de probar un sistema, es que la cantidad de combinaciones y pruebas posibles puede ser demasiado grande y el tiempo reducido. Por este motivo es importante seleccionar casos de prueba que sean eficaces, combinando entradas y estados del sistema que tengan la más alta probabilidad de encontrar incidentes sin requerir un número excesivo de pruebas [21].
Para realizar pruebas funcionales, se utilizan técnicas de testing planificado que especifican una estrategia de selección de entrada de los casos de prueba y de análisis de resultados.
Debido a que cada técnica detecta distintos tipos de errores, es necesario combinarlas para optimizar el tiempo invertido y los incidentes encontrados.
A continuación se presentan las principales técnicas de testing planificado pertinentes a este proyecto de grado aunque si se desea investigarlas en mayor profundidad se pueden ver las referencias.

3.5.4.1. Clases de equivalencia

Esta es una de las técnicas más utilizadas en el testing funcional y consiste en dividir el dominio de entrada en sub-dominios, o subconjuntos, razón por la cual se la conoce como particiones en clases de equivalencia [6]. Se asume que una prueba de un valor representativo de cada clase es equivalente a probar con cualquier otro valor. Eso es, si un caso de prueba en una clase de equivalencia detecta un error, todos los otros casos de prueba en esa clase deberían encontrar el mismo. También se puede dividir el dominio de salida.

Para seleccionar los casos de prueba se debería [21]:

- Identificar las variables y sus posibles valores
- Identificar las clases de equivalencia
- Diseñar un nuevo caso de test que abarque la mayor cantidad de clases válidas que sea posible.
    - Hasta cubrir todas las clases válidas
- Diseñar un caso de test para cada una de las clases inválidas
    - Hasta cubrir todas las clases inválidas

3.5.4.2. Valores límite

De acuerdo a Myers [6], la experiencia muestra que los casos de prueba que exploran condiciones límite tienen una mayor rentabilidad que aquellos que no lo hacen.
Los valores límite son los que están en las fronteras o bordes de las clases de equivalencia y los inmediatamente superiores o inferiores.

El análisis de los valores límites difiere de la partición de equivalencia en dos aspectos:

- En lugar de seleccionar cualquier elemento dentro de una clase como representante, esta técnica requiere que uno o más elementos sean seleccionados de manera que los extremos de la clase de equivalencia sean puestos bajo prueba.
- En lugar de enfocar la atención en los valores de entrada, los casos de prueba también se derivan considerando el espacio de resultado (clases de equivalencia de salida).

3.5.4.3. Combinación por pares

Se utiliza cuando la realidad a probar involucra muchas variables con múltiples valores, y es imposible probar todas las combinaciones de ellos.
Por este motivo es necesario reducir significativamente el número de casos de prueba a ejecutar, eligiendo combinaciones de valores que tengan la mayor probabilidad de detectar defectos
Muchos de los tipos de errores que se generan son:
Singulares: se producen incidentes porque una variable toma un valor específico.
Doble: Se producen incidentes por la combinación de valores específicos de dos variables independientes entre sí.

La técnica “Combinación por pares” permite generar un conjunto de casos de prueba en el cual estén representados, al menos una vez, todos los pares de valores posibles de las variables identificadas [22].
Debido al gran número de combinaciones posibles, existen varias herramientas que generan casos de prueba con esta técnica en forma automática.

Son un buen punto de partida para diseñar otros casos de prueba y además asegura el cubrimiento de todos los pares.
Sin embargo, tienen la limitante de que alguna combinación importante o frecuentemente ejecutada podría no generarse; además algunos valores dependientes pueden generar combinaciones inválidas.

Para utilizar esta técnica es necesario:

- Identificar variables independientes.
- Identificar dominios y valores posibles (Por ejemplo: utilizando clases de equivalencia).
- Generar las combinaciones de a pares.
- Analizar combinaciones inválidas y resolverlas, si la herramienta no permite restringirlos antes.
- Considerar otras combinaciones particulares y agregarlas

3.5.4.4. Máquinas de estado

En términos generales, una máquina de estados es una abstracción que modela un gran número de sistemas (flujos de trabajo, dispositivos de control, entre otros) y describe el ciclo de vida de un sistema o de un objeto del sistema.
En el contexto de testing de software resulta de gran utilidad para derivar casos de prueba y comprobar flujos de ciertas funcionalidades dentro del sistema.
Este modelo consta de cuatro partes básicas:

- Los **estados** en que se puede encontrar el software.
- Las **transiciones** de un estado al siguiente: definidas por las reglas de la máquina.
    - Los **eventos** que ocasionan una transición.
    - Las **acciones** que resultan de una transición.
    - Las **guardas** que pueden establecer condiciones para que un evento ocasione la transición.

De acuerdo a las notas del curso de TVS [23], algunos de los errores que permiten detectar estas pruebas son:

- Omisión de transiciones
- Transiciones incorrectas
- Omisión de acciones
- Acciones incorrectas
- Eventos incorrectos
- Estados corruptos
- Fallas en el manejo de mensajes ilegales

3.5.4.5. Tablas de decisión

Las tablas de decisión representan relaciones lógicas entre las condiciones (entradas) y las acciones (salidas). Los casos de prueba son derivados sistemáticamente considerando cada combinación posible de condiciones y de acciones [4].
Los casos de pruebas que siguen esta técnica resultan útiles cuando hay combinaciones complejas de condiciones, acciones y reglas, ó se requiere un método que efectivamente evite situaciones imposibles, redundancias y contradicciones.
Son una buena manera de capturar los requisitos del sistema que contienen condiciones lógicas y registrar las reglas de negocio complejas.
En primer lugar se analiza la especificación; luego las condiciones y las acciones del sistema se identifican y listan en una matriz, donde cada columna representa una combinación única [52].
Los principales elementos de las tablas son:

- Las condiciones sobre el dominio de entrada.
- Las acciones a ejecutar o salidas resultantes.
- Las entradas válidas para determinada condición.
- Las reglas indican qué acciones se disparan para cada conjunto de entrada.

Las tablas de decisión se clasifican de acuerdo al tipo de entradas en:

- Tablas de decisión con entradas limitadas:
    - si las entradas son binarias (V/F, 0/1, S/N)
- Tablas de decisión con entradas extendidas:
    - si las entradas pueden asumir múltiples valores

La fortaleza de las tablas de decisión es crear combinaciones de condiciones que pueden no ejercitarse de otra manera. Puede ser aplicada cuando la acción del software depende de varias decisiones lógicas [24].

3.5.4.6. Basadas en casos de uso

Un caso de uso es un flujo o secuencia de acciones que un sistema realiza con el fin de lograr un resultado de valor para un actor particular, siendo este un elemento humano o automatizado que interactúa con el sistema [25].
Estos procedimientos cuentan con:

- Precondiciones: éstas deben cumplirse para que el caso de uso finalice con éxito.
- Poscondiciones: serie de resultados observables.
- Estado final del sistema después una vez que se completa el caso de uso.
- Flujo principal y varios flujos alternativos.

Los casos de uso describen los escenarios a través de un sistema basado en su uso probable real, por lo que los casos de prueba derivados a partir de ellos son muy útiles para encontrar defectos en los escenarios durante el uso real del sistema.

Los casos de uso son muy útiles para diseñar pruebas de aceptación con la participación del cliente o del usuario y ayudan a descubrir defectos en la integración causados por la interacción de diversos componentes, que la prueba individual del componente no considera [24].

3.6. Proceso de testing funcional

En esta instancia se hace referencia al proceso de testing funcional y se definen las actividades del testing que luego se van a articular con el proceso de desarrollo.
El proceso presentado es el que utiliza el CES ya que los estudiantes se basarán en él para presentar la documentación requerida.

3.6.1. Etapas

El proceso de testing se articula con el proceso general de software (figura 3.6, el cual consiste de varias etapas y actividades, entre las cuales se destacan:

- Planificación
    - Definición del cronograma
    - Estudio de riesgos
    - Escribir el plan de pruebas e Inventario de Pruebas
    - Seleccionar las herramientas para las pruebas
- Ciclo de Prueba
    - Configuración

        - Obtener el hardware y otros recursos necesarios
        - Instalación del laboratorio de pruebas
        - Instalar las herramientas de prueba y de gestión del testing
- Diseño de las Pruebas
- Ejecución de las Pruebas:
    - Ejecutar las pruebas, registrar su estado y reportar resultados
- Evaluación
    - Se evalúa el proyecto de pruebas y se archivan los elementos.

En particular haremos referencia a algunos de los puntos mencionados arriba.

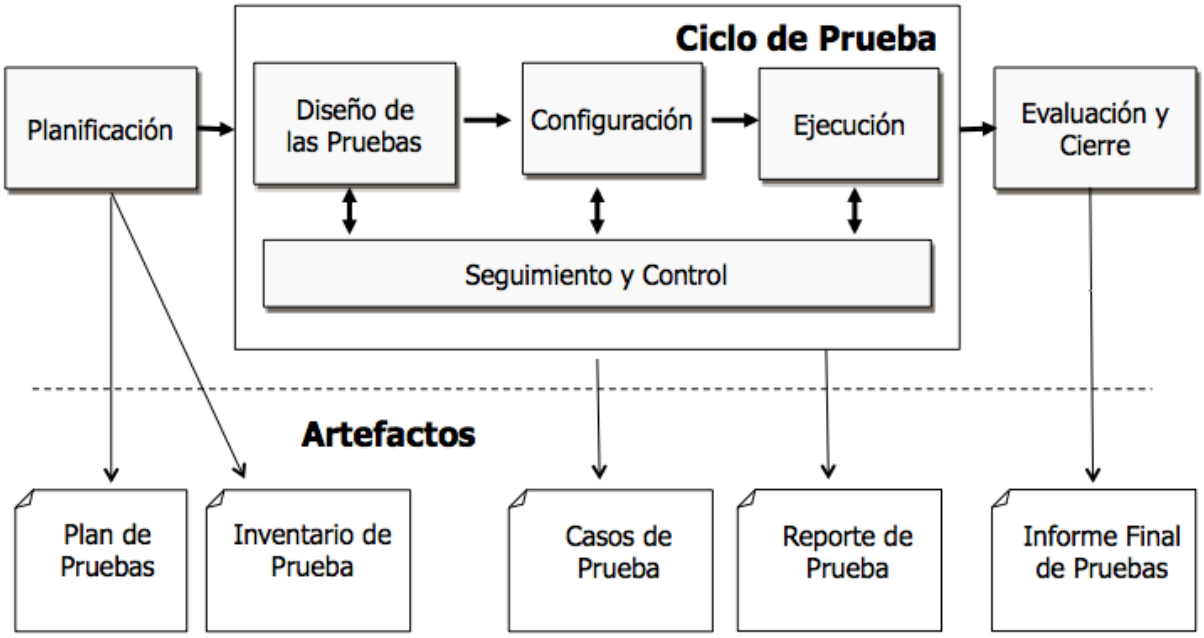


Figura 3.6: Detalle de las etapas del proceso de testing.

3.6.1.1. Inventario de pruebas

Permite asignar prioridades a los elementos a probar (alta, media o baja), establecer una complejidad a cada parte del producto a probar, establecer un orden de trabajo y recortar el alcance de las pruebas, dejando por fuera los elementos de menor criticidad.
Los elementos pueden ser:

- Casos de uso.
- Pantallas.
- Menúes.
- Transacciones.
- Funcionalidades.

3.6.1.2. Plan de pruebas

Algunos de los elementos que incluye son:

- Su objetivo es definir las condiciones en que las pruebas serán ejecutadas.
- Incluye:

    - Alcance del proyecto, especificando qué elementos serán probados y cuáles no.
    - Estrategias y técnicas utilizadas para las pruebas.
    - Entregables que se entregan al final del ciclo de pruebas y al final del proyecto (Documento de casos de prueba, Reportes de incidentes, entre otros).
- Gestión de incidentes: es necesario definir un proceso mediante el cual el tester pueda reportar un incidente y éste sea comunicado debidamente al equipo de desarrollo. Un incidente se puede definir como:
    - Un defecto del producto
    - Una mejora
    - Una observación
- Por más detalles del plan de pruebas ver el anexo “Registro de pruebas”.

3.6.1.3. Diseño de las pruebas

Durante esta etapa se diseñan los casos de prueba a partir de los requisitos, la especificación del producto y todos los elementos que se hayan recopilado, según la estrategia de pruebas definida anteriormente.
Esta actividad implica:

- Revisión de requisitos: es necesario validar los requisitos basándose en la información que hay disponible (o investigar más a fondo si fuera necesario) en documentos de especificación de requisitos, manuales, modelos de casos de uso, etc.
- Exploración del producto: implica un conocimiento del producto explorándolo; realizando un aprendizaje continuo y experimentando con él.
- Diseño de los casos de prueba: ésto implica definir:
    - el conjunto de datos de prueba a utilizar
    - las estrategias y técnicas a utilizar: testing exploratorio o planificado, con todas las posibles técnicas que éstos abarcan.
    - casos de prueba: consiste en un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y post-condiciones de ejecución.
- Validación de los casos de prueba: idealmente los casos de prueba deben ser validados y aprobados por el cliente, para que tengan valor para el negocio.

3.6.1.4. Ejecución de pruebas

El objetivo de esta etapa es contrastar el comportamiento esperado del software con su comportamiento real, analizar las diferencias y reportar los resultados.
Consiste en ejecutar de forma automatizada o manual los casos de prueba seleccionados para el ciclo y observar su resultado.
Al finalizar la etapa se reporta el avance del testing en el ciclo y se evalúan los resultados obtenidos.

3.6.1.5. Reporte de Incidentes

Como parte del proceso de testing, el reporte de incidentes por parte de los testers resulta una actividad de suma importancia ya que es el medio de comunicación principal entre el equipo de desarrollo y el de testing.

Consiste en completar un reporte con información relacionada a la detección de un defecto, la cual debe ser lo suficientemente clara y concisa para que pueda ser interpretada correctamente por el desarrollador y el defecto pueda ser corregido.
El reporte consiste de varias secciones, algunas de las cuales son:

- Título: una breve descripción del defecto detectado
- Descripción: puede implicar una explicación más extensa que el punto anterior si es necesario. Incluye detalles que ayudan al desarrollador a identificar fácilmente el defecto brindando detalles como: pantalla, entrada ingresada, salida obtenida y cualquier otro dato que el tester considere pertinente.
- Reproducibilidad: refiere a la frecuencia con la que el defecto se pudo reproducir. Los valores más comunes son: siempre, ocasionalmente o no se pudo reproducir.
- Severidad: refiere a la severidad del defecto encontrado, la cual se determina por la funcionalidad afectada y las consecuencias de la falla ocasionada.
- Prioridad: este punto está relacionado con el anterior ya que refiere a la prioridad que se le debe dar al defecto reportado, para que éste sea atendido por el equipo de desarrollo y corregido. Si un defecto afecta una funcionalidad importante del sistema o genera consecuencias graves en el mismo, la prioridad será alta.
- Imagen o archivo adjunto: en muchos casos resulta de gran utilidad incluir una imagen o archivo de algún tipo para reflejar la falla encontrada en el sistema. De esta manera es más fácil para un desarrollador comprender el defecto detectado e identificarlo.
- Por más opciones incluidas en un reporte de defecto, ver [poner una referencia].

Normalmente en todo proceso de testing se utiliza una herramienta para reportes de defectos, para poder contar con un sistema organizado de registro de defectos, realizar un seguimiento de los mismos y mantener una comunicación más transparente y fluida entre los equipos de testing y desarrollo.

3.7. Mutaciones

La técnica de mutación de testing consiste en crear un conjunto de versiones con defectos, errores o mutantes de un programa con el objetivo final de diseñar un conjunto de tests que distingue el programa de sus mutantes [26]. Los programas con defectos o errores son mutantes del original y representan fallas potenciales, un error del desarrollador o requieren explícitamente una heurística de testing que debe ser satisfecha [27].
Una de las premisas de la plataforma Test School es que presente versiones mutadas de una aplicación, para que los estudiantes las analicen y reporten los errores encontrados, basándose en los casos de prueba realizados.
En el contexto del proyecto de grado, los mutantes fueron generados manualmente para controlar los defectos inyectados y la dificultad que estos representaban. Cada nivel del sistema tiene asociada una dificultad (implícita) y ciertos conocimientos teóricos de parte del estudiante. Al momento de corregir el trabajo de un estudiante, se puede aparear los defectos encontrados contra los conocidos, evaluando de esta manera la calidad de los casos de prueba.

3.8. Herramientas investigadas

Antes de comenzar el diseño del nuevo sistema, se realizó una búsqueda de herramientas que cuenten con las funcionalidades requeridas o alguna similar para corroborar que no exista otra en el mercado que pueda satisfacer las necesidades del cliente. Esta investigación también sirvió para tener un panorama general en cuanto al avance en la construcción de este tipo de herramientas.
En esta sección también se incluye una breve descripción de las herramientas que sirvieron de inspiración para diseñar el sitio web; y finalmente se listan las aplicaciones analizadas cuando se buscaba la que sería la aplicación inicial del sistema (a ser mutada).

3.8.1. Herramientas para inyección de errores

En el entorno de testing, existe un conjunto de técnicas de inyección de defectos y diversas herramientas para la inyección de errores mediante el uso de técnicas SWIFI (Software Implemented Fault Injection) puras o en conjunción con recursos de depuración hardware presentes en algunos procesadores, siendo la generalización para cualquier tipo de sistema bastante compleja.
En la tabla 3.2 se listan algunas herramientas que se encuentran en el mercado y fueron investigadas para saber si podrían ajustarse a algunos de los requisitos.

Herramienta	Descripción	Referencia
Xception	Provee un conjunto de inyección de defectos, incluyendo defectos espaciales, temporales y defectos relacionados con la manipulación de los datos en la memoria.	[28]
Goofi (Generic Object-Oriented Fault Injection)	Está diseñado para poder adaptarse a varios sistemas bajo prueba y a diferentes técnicas de inyección de defectos.	[29]
Mafalda (Microkernel Assesment by Fault injection AnaLisys and Design Aid)	Está orientada a la validación de microkernels (núcleos del sistema operativo) y permite realizar dos tipos de inyecciones.	[30]
Exhaustif	Está diseñado para analizar dinámicamente sistemas en los que se necesite validar si se satisfacen los requisitos de Fiabilidad	[31]
Bacterio	Es una herramienta de testing implementada en Java y basada en mutaciones. La mutación es aplicada a nivel de bytes.	[27]

Tabla 3.2: Herramientas de inyección de defectos.

Las herramientas mencionadas resultan bastante más complejas de lo que se busca para el nuevo sistema. De hecho algunas de ellas se utilizan en contextos e industrias en que el testing del software y hardware tiene un papel muy importante y delicado, ya que si fallan, las consecuencias pueden ser graves. Por este mismo motivo, todas son pagas y en algunos casos sus costos son elevados.

Luego de analizar las herramientas existentes en el mercado, se concluyó que éstas no se adaptan al sistema que se necesita ya sea por ser demasiado complejas, o porque su costo no vale la pena la inversión.

3.8.2. Aplicaciones de inspiración e-schooling

Se analizaron algunos casos de aplicaciones similares como ejemplo e inspiración para detectar elementos claves del diseño y usabilidad.

- **CodeSchool**: Es una aplicación web que permite a los usuarios realizar cursos sobre tecnologías web [32].
- **Moodle:** Es una aplicación web basada en principios pedagógicos que apunta a compartir información sobre cursos, educación a distancia y otras funcionalidades relacionadas al e-learning [33]. Además, la Plataforma de Capacitación del CES está construida sobre esta aplicación.

En la sección 4.3.2 se explican en detalle los conceptos de estas aplicaciones que fueron adaptados al sistema Test School.

3.8.3. Elección de aplicación a mutar

Como parte del proceso de investigación se consideraron dos enfoques para elegir una aplicación a mutar:

- Tomar una aplicación existente y modificarla o tomar una parte reducida de ella
- Construir una nueva

Construir una nueva aplicación a medida podía resultar beneficioso ya que podríamos incluir las funcionalidades que se consideren más apropiadas, pero sería necesaria una gran inversión de tiempo en la construcción del nuevo sistema y no era el objetivo principal del proyecto.

Optamos entonces por buscar aplicaciones libres para poder integrarlas al sistema y modificarlas.
Inicialmente investigamos aplicaciones basadas en acertijos o juegos, por resultar más atractivas y motivadoras para los usuarios como:

- Puzzles [34]
- 2048 [35]

Sin embargo, las aplicaciones encontradas tenían limitaciones en cuanto a funcionalidades y comprensión de la aplicación, por lo tanto se consideró que no eran apropiadas para el sistema.

Finalmente optamos por elegir una aplicación dentro de cierta categoría y que maneje funcionalidades apropiadas para el público objetivo del sistema.
Debido a la gran cantidad de opciones disponibles, decidimos ordenar las aplicaciones en base a un ranking, tomando ciertos parámetros como referencia y ponderando cada programa en base a ellos.

Parámetros tomados en cuenta para priorizar las aplicaciones y ordenarlas:

1. **Categoría**: debido a los perfiles de los usuarios (testers), la adaptabilidad al sistema y el tamaño de las aplicaciones, se priorizan positivamente algunas categorías y se descartan otras para realizar el ranking:
    - Herramientas de desarrollo
    - E-mail
    - Data Warehouse (DW)
    - Prevención de datos
    - Bases de datos
    - Infraestructura de la nube
    - Respaldo de datos
    - Firewall
    - Anti-Virus/Anti-Malware
    - Manejadores de archivos
    - Aplicaciones para compartir archivos
    - Herramientas multimedia
    - Aplicaciones específicas de asignaturas:
        - Matemática
        - Biología
        - Química
        - Astronomía
        - Física
    - Modificaciones de Kernel

   Una vez priorizadas las categorías, se evaluará el resto de los parámetros sólo para aquellas aplicaciones que estén en categorías altas (con valores del 1 al 5, siendo el 5 el valor más alto y la mejor categoría).
2. **Tamaño**: no solamente se evalúa el tamaño total de la aplicación, ya que se podría utilizar una versión reducida de la aplicación; sino que también la complejidad para comprenderla, tarea vital para modificarla en su totalidad o elegir una fracción de ella.
3. **Variedad de funcionalidades:** contar con funcionalidades varias permite aumentar la capacidad de inyectar diversos defectos.
4. **Adaptabilidad a los usuarios** (testers)
    - Complejidad media del funcionamiento
    - Entretenimiento: refiere a qué tan atractiva puede resultar la aplicación para los usuarios, así como también la facilidad de comprensión. Esto quiere decir que aquellas que sean de uso cotidiano o muy frecuente en la actualidad recibirán mejor puntuación.
5. **Facilidad de acceso al código fuente:** es fundamental poder acceder a la totalidad del código fuente de la aplicación para poder modificarlo e inyectar defectos en el programa.
6. **Facilidad de instalación:** la aplicación debe poder ser instalada en forma directa y sin complicaciones inesperadas, tanto en las máquinas de los desarrolladores como en el servidor final. Este aspecto está directamente relacionado con la documentación disponible sobre la aplicación.
7. **Estabilidad y última actualización:** tomando en cuenta que la aplicación será analizada para detectar defectos y éstos deben ser conocidos, si ésta no es estable puede presentar algún comportamiento inesperado. Ésto afectará los defectos presentes en cada nivel y consecuentemente la evaluación de los estudiantes.
   Las actualizaciones tienen una gran incidencia en la estabilidad de una aplicación ya que es

un indicador del mantenimiento de la misma. Si existen varios defectos reportados para la aplicación que no han sido corregidos en un período considerable de tiempo, ésta será puntuada negativamente.

8. **Sistema operativo donde opera:** es fundamental que la aplicación elegida se pueda correr en un servidor local en el CES, por lo tanto ésta debe ser compatible con Linux.
9. **Lenguaje:** el manejo de los desarrolladores de algunos determinados lenguajes resulta fundamental a la hora de elegir una aplicación, ya que el rendimiento al momento de implementar es mayor cuando se trabaja con un lenguaje conocido, frente a uno nuevo. Por este motivo se ponderó con un valor mayor a los lenguajes conocidos por los desarrolladores frente a los nuevos, o a los que se consideran más complejos.
10. **Documentación disponible y comprensión del código fuente:** es fundamental comprender el funcionamiento de la aplicación para analizar qué defectos se pueden inyectar. En este sentido resulta de gran utilidad contar con documentación apropiada y completa sobre el código fuente y las funcionalidades en sí.
11. **Recomendaciones de usuarios:** si hay recomendaciones de usuarios disponibles son tomadas en cuenta, tanto sobre usabilidad de la aplicación como calidad de su implementación.

Pasos seguidos para refinar búsqueda:

- Se comenzaron a estudiar las candidatas a aplicación asignando valores del 1 al 5 a la categoría a la que pertenecen.
- Se visitaron las páginas web de aplicaciones de las categorías de mayor valor, enfocándonos en tener un primer acercamiento a ellas, revisando documentación y código fuente, descartando aquellas con poca documentación, interfaz o funcionalidades poco convincentes.
- Luego de realizar este primer filtro se procedió a intentar instalar las aplicaciones candidatas (9 en total); descartando en este paso a varias por su dificultad de instalación.
- Entre las últimas candidatas puntuamos los parámetros descritos en la sección anterior y las ordenamos según la sumatoria de dichos valores.

Finalmente se encontró la aplicación Spree, que cumple con todos los requisitos que buscamos y creemos que se ajusta a los usuarios del sistema.

Aplicaciones candidatas luego de filtrado

A continuación se presentan las aplicaciones que pasaron por los filtros descritos en la sección anterior, ordenadas por su puntuación. Para ver el listado completo de aplicaciones investigadas junto con los parámetros evaluados, ver “Anexo: Ranking de aplicaciones evaluadas”.

Aplicación	Categoría	Puntuación
Spree	Comercio Electrónico	50
Moodle	Educación en Línea	47
Broadleaf Commerce	Comercio Electrónico	44
Redmine	Rastreo de defectos, Manejo de Proyectos	43
BORG Calendar	Listas de Tareas /Organizadores/Calendarios	43
OpenCart	Comercio Electrónico	42
FlySpray	Rastreo de defectos	41
ATutor	Educación en Línea	41
Collabtive	Colaboración	40

Tabla 3.3: Aplicaciones mejor valoradas.

Como puede verse, la aplicación Spree[36] fue la que obtuvo mayor puntuación global y por esta razón fue elegida para ser integrada al sistema. Ésta es la aplicación sobre la cual se inyectan defectos conocidos que luego deben ser detectados por los estudiantes (usuarios).

4. Test School

4.1. Descripción

Es un sistema de aprendizaje y evaluación electrónica, donde tanto estudiantes como profesores pueden ágilmente reportar bugs y evaluar los trabajos entregados, respectivamente.
Sus principales objetivos son:

- Ofrecer a los estudiantes una plataforma amigable que intenta simular la experiencia de testear un software de la forma más realista posible. Al mismo tiempo desarrollan sus habilidades como testers, poniendo en práctica los conocimientos teóricos adquiridos.
- Agilizar el proceso de evaluación por parte de los profesores.

Para lograr los objetivos se construyeron dos secciones:

- **Un sitio web** al cual ingresan solo los estudiantes de los cursos del CES, que deben ser habilitados previamente.
  Desde el panel principal, los usuarios podrán acceder a los distintos niveles que componen cada aplicación. Esta página constituye el marco para presentar embebida, la aplicación y sus mutaciones, simulando un escenario típico para un tester. Desde aquí reportarán los bugs encontrados para luego finalizar su sesión de trabajo enviando un registro de pruebas.
- **Un panel de administración** desde donde los profesores analizan, revisan y corrigen las sesiones de testing finalizadas por los estudiantes. Desde aquí también se pueden manejar las entidades más importantes del sistema que son: usuarios, aplicaciones, niveles, bugs conocidos y sesiones.

4.2. Estructura por niveles

El sistema se estructuró en niveles de dificultad, donde cada nivel presenta una versión mutada de una aplicación y evaluar un conjunto de características de los testers, en base a ciertos defectos inyectados y las técnicas que deben aplicar para detectarlos.
Solo el primer nivel aparece habilitado en un principio, y cada nuevo nivel se desbloquea a medida que se aprueba el anterior (esta condición se maneja desde el panel de administración). Este sistema de bloqueo se implementó para que los estudiantes solo accedan a aquellos niveles de mayor dificultad a medida que se avanza en el curso. Además es una forma de motivar a los estudiantes, ya que no podrán avanzar si no aprueban el nivel actual.
En todos los niveles se les proveerá a los estudiantes una o más plantillas como guía del registro de pruebas y se les pedirá que suban la evidencia de su trabajo antes de finalizar el mismo.
La versión inicial del sistema es explicada en detalle en la sección 4.4.

4.3. Diseño de interfaz

En esta sección se explica el concepto que se tuvo en cuenta para diseñar la interfaz de la aplicación, qué aplicaciones existentes en el mercado se evaluaron como ejemplos y qué beneficios presenta el diseño final.

4.3.1. Concepto base

Al tratarse de una aplicación de tipo e-schooling, se tomaron en cuenta ciertos parámetros que son recomendados para estos casos [37]:

1. El principio multimedia: las personas aprenden mejor de una combinación de imágenes y texto que de texto solamente.
2. Los gráficos deberían ser relevantes, no solamente decorativos: aunque pueda resultar tentador cargar varias imágenes para generar un atractivo visual, si se abusa de este recurso el resultado puede ser que el estudiante se enfoque más en procesar ese contenido en lugar de aprender los conceptos presentados. Esto no quiere decir que la interfaz no pueda ser llamativa, pero se recomienda buscar maneras de guiar el enfoque visual hacia lo que verdaderamente interesa, e inclinarse al minimalismo si es necesario.
   Las imágenes y los gráficos deben ser usados como herramientas para facilitar la comunicación con el usuario y la usabilidad de la aplicación en general.

4.3.2. Conceptos tomados de otras aplicaciones

4.3.2.1. CodeSchool

Los principales elementos que nos pareció interesante adaptar a nuestro sistema fueron:

- La forma en que se presentan los cursos a los usuarios o estudiantes
- El panel del usuario, donde éste puede ver sus progresos, qué cursos ha completado y otra información relevante.
- La interfaz resulta muy atractiva en cuanto a combinación de gráficos, texto y colores. También maneja flujos sencillos y rápidos para las principales funcionalidades, como el acceso a los cursos o al panel del usuario.

4.3.2.2. Moodle

Los principales conceptos que se tomaron de esta aplicación son:

- Manejo de sesiones: aunque la aplicación CodeSchool también maneja este concepto, en el caso de Moodle es más explícito ya que los estudiantes pueden comenzar una sesión (de revisión del sitio web en nuestro caso), abandonarla y retomarla luego, para finalmente enviar una versión final de su trabajo para considerarlo completo. Hasta que el estudiante no envía su evaluación, ésta no será visible para los profesores y por lo tanto tampoco será corregida.
- Manejo de documentos: Test School permite a los estudiantes descargar ciertos archivos y subir otros. Esto les permite a los administradores cargar los documentos que consideren necesarios para el curso y que luego pueden ser descargados por los estudiantes. También permite a estos últimos cargar archivos o documentos como evidencia de su trabajo, para su evaluación por parte del docente.
- Panel de administración: desde aquí los profesores tienen acceso a las evaluaciones de los estudiantes y pueden realizar las correcciones correspondientes, así como también escribir una devolución aclaratoria, que será recibida por el estudiante.
  También permite manejar el contenido de la aplicación, que abarca textos, imágenes y documentos sobre cursos.

4.3.3. Descripción de la interfaz

4.3.3.1. Sitio web Test School

Este es el sitio al que ingresan los estudiantes y mediante el cual acceden a la versión mutada de la aplicación que corresponda para un curso.
Se diseñó una interfaz minimalista resaltando en cada sección la funcionalidad principal mediante imágenes o gráficos, y utilizando la cantidad mínima de texto para comprender y dar contexto.

4.3.3.1.1. Principal

El objetivo de esta página es presentar al usuario las aplicaciones disponibles en el sistema y los niveles de cada una de ellas. Además resultó de utilidad mostrar cuáles de estos ya ha aprobado para darle una idea de su progreso y que pueda identificar fácilmente cuáles tiene pendientes.
Aquí se presentan los cursos divididos por aplicación, recordando que por cada aplicación existen varios niveles de dificultad, y cada nivel corresponde a una mutación de ésta.
Sobre cada nivel se muestra el título, una imagen descriptiva (de tamaño pequeño) y una señal de color para indicar al estudiante si ya aprobó o no ese nivel.
En la figura 4.1 se muestra esta sección, para tres niveles disponibles y habilitados.

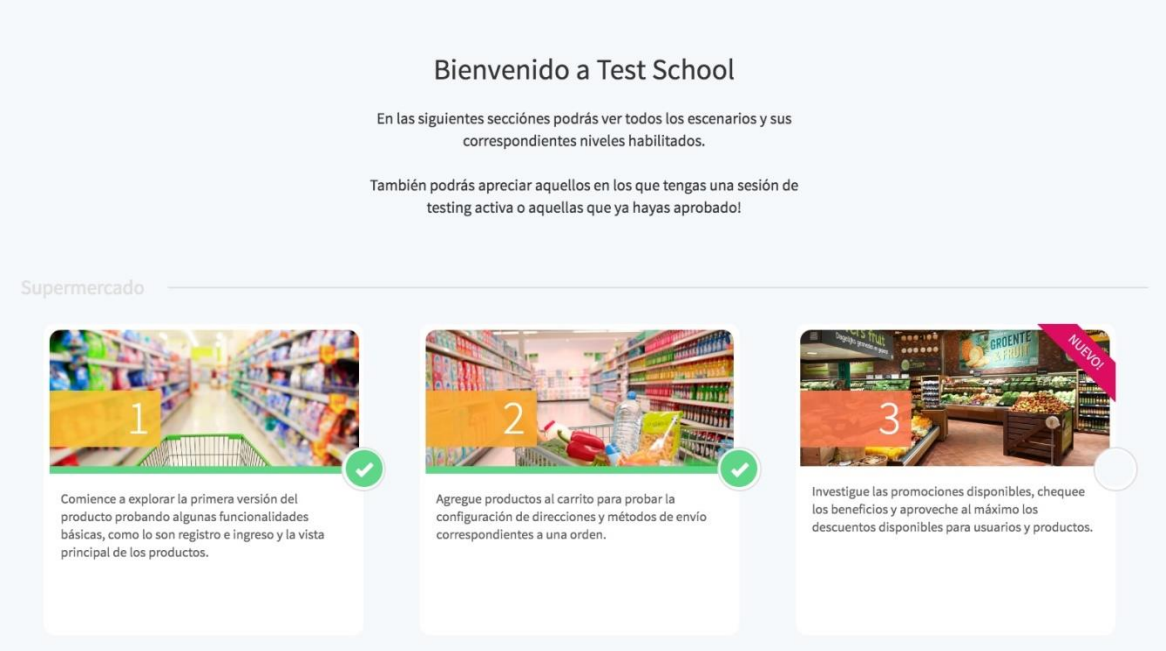


Figura 4.1: Página principal de Test School

4.3.3.1.2. Panel del usuario

Un aspecto importante de usabilidad es darle al usuario acceso a un panel donde pueda ver y editar su perfil, así como también obtener información rápidamente sobre su progreso en el sistema, o un registro de actividades.
Desde el panel del usuario, éste puede editar su nombre real, nombre de usuario y foto de perfil, y acceder a un breve historial de evaluaciones propias, donde se muestra solo la información considerada relevante.
La información disponible consiste en:

- Título de los niveles completados, agrupados por aplicación.
- Por cada nivel se detalla:
    - Devolución por parte de los profesores.
    - Nota de la evaluación.
    - Cantidad de defectos reportados.

4.3.3.1.3. Descripción del nivel

En esta instancia se presenta al usuario un nivel particular donde se describe el escenario que el estudiante debe evaluar.
Esta página se divide en las siguientes secciones:

- Texto introductorio, donde se resume brevemente la consigna del ejercicio
- Descripción del escenario, donde se especifica la realidad y las funcionalidades disponibles en la aplicación.
- Objetivos del ejercicio, donde se deja en claro lo que se espera del estudiante en esta instancia y las técnicas que se recomienda utilizar para detectar los defectos presentes en la versión mutada correspondiente.
- Botón para comenzar la sesión del usuario en ese nivel, o reanudar una sesión ya iniciada.

4.3.3.1.4. Página de la aplicación mutada

El principal objetivo de esta página es simular de forma realista la utilización de una aplicación web, y a su vez tener rápido acceso a un panel de notas y reporte de defectos.
Por estos motivos se optó por cargar la aplicación mutada en un marco embebido dentro de la vista nativa del sistema, que ocupa la ventana totalmente (esto puede verse en la figura 4.2).
Dado que el principal objetivo en esta instancia es reportar defectos, hay un icono siempre visible en la parte inferior de la pantalla que redirige al usuario a la página de Mantis.
Junto a esta opción se encuentran las de subir o descargar el archivo del registro de pruebas y enviar la sesión para su corrección una vez que el estudiante considera que ha terminado.

Para manejar las notas del usuario se optó por crear una barra lateral, que permanece oculta para no obstaculizar la vista de la aplicación. Ya que el objetivo de éstas es que sean una herramienta auxiliar para los usuarios, se eligió un estilo con notas rápidas, o “post-its”, que resultan llamativas y fácilmente visibles.

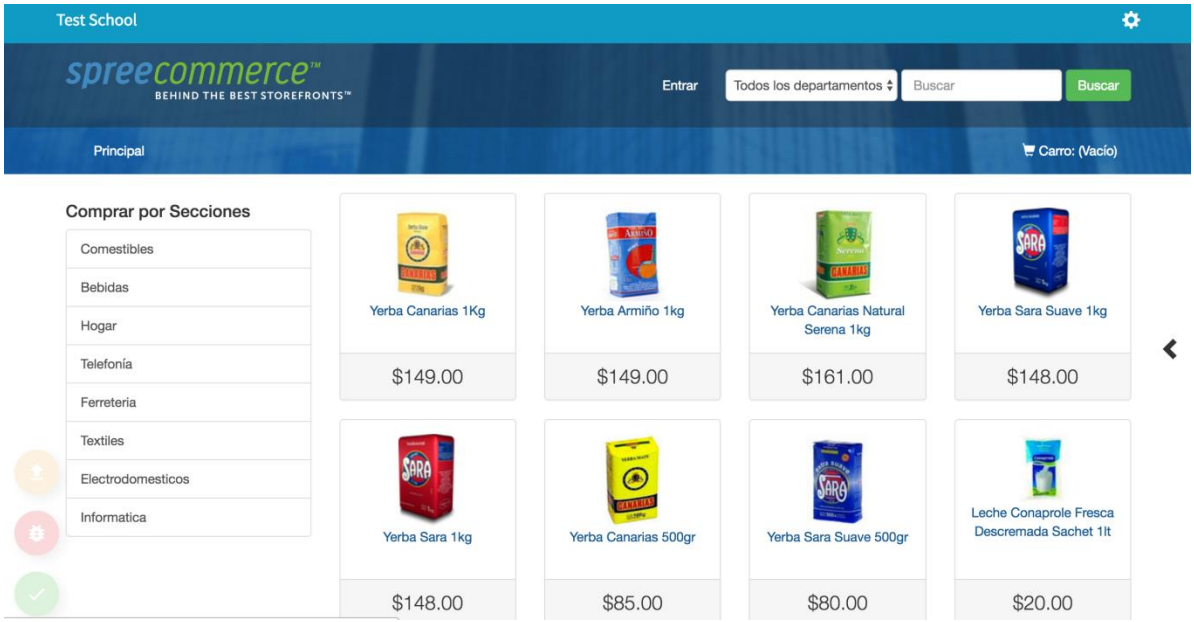


Figura 4.2: Página de la aplicación mutada

4.3.3.2. Panel de administración

Para construir el panel de administración se priorizó practicidad y efectividad sobre el estilo; por este motivo se incluyeron aquellas funcionalidades que resultaran necesarias o útiles para facilitar las tareas de los profesores a la hora de la corrección y manejo en general del sistema.

4.3.3.2.1. Principal

En esta página se incluyen varias secciones para el manejo de los usuarios y sus sesiones, así como también manejo de las aplicaciones que se presentan en el sistema y los distintos niveles para cada una de ellas.

Las secciones disponibles son:

- Administradores: aquí se manejan los administradores del sistema.
- Aplicaciones: aquí se pueden ver las distintas aplicaciones disponibles en el sistema.
- Bugs conocidos: aquí se especifican los defectos inyectados en cada versión mutada de las aplicaciones.
- Niveles: desde esta sección se pueden subir las versiones mutadas de una aplicación.
- Usuarios: desde aquí se pueden manejar los usuarios del sistema.
- Usuarios habilitados: aquí se puede agregar en bloque la lista de usuarios o estudiantes habilitados para ingresar en el sistema.
- Correcciones: por tratarse de una de las funcionalidades más complejas del panel de administración se construyó en un panel independiente, al cual se accede desde el panel de administración. Este panel es explicado en mayor detalle en la siguiente sección.

En la figura 4.3 se muestra el panel de administración y la sección “Administradores”. En el encabezado figuran las secciones descritas.

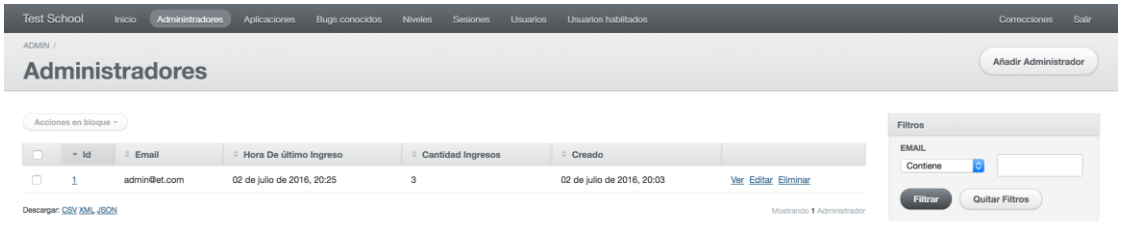


Figura 4.3: Panel de administración de Test School

4.3.3.2.2. Panel de correcciones

La sección de correcciones es una de las más importantes y complejas del sistema, ya que refiere a uno de los principales objetivos del sistema: crear una herramienta que facilite la evaluación de los estudiantes por parte de los profesores del curso.

Por este motivo, se diseñó un flujo conciso que permita realizar una ágil evaluación de los estudiantes, dando rápido acceso a la información necesaria y herramientas auxiliares para manejar dicha información.

Los principios tomados en cuenta para esta sección a nivel de interfaz son:

- Diseño claro; que no presente grandes dificultades de aprendizaje para la persona que evalúa.
- Funcionalidades fácilmente accesibles y herramientas auxiliares, como filtros y orden de listas, siempre visibles.
- Agilidad para evaluar. Esto implica llegar a una instancia final, donde se cuenta con toda la información necesaria para evaluar a un estudiante, en la menor cantidad de pasos posibles.
- Como complemento del punto anterior, para que la evaluación sea rápida la información sobre la sesión del usuario debe ser recolectada de manera eficiente.
- No incluir información redundante. Solo se brinda la necesaria, pero también es posible obtener información más completa si así se desea.

Este panel se divide básicamente en dos partes:

1. Vista principal (figura 4.4), que incluye:
    a. Lista de sesiones de usuarios.
    b. Menú lateral con opciones varias de filtrado para agrupar resultados en forma eficiente y rápida.
2. Vista en detalle de los reportes de un usuario (figura 4.5).
    a. Se accede seleccionando uno de los usuarios de la página anterior.
    b. Se pueden ver los defectos conocidos para el nivel evaluado, y cotejarlos contra los que reportó el usuario.
    c. Para comparar las dos listas de defectos y ver claramente cuáles de ellos fueron reportados y cuáles no, se integraron algunas herramientas para facilitar la correspondencia. Por esta razón, la asignación de un defecto reportado a uno conocido resulta una funcionalidad muy intuitiva.
    d. Una vez completada la primera etapa de la evaluación, se prosigue a una última vista donde se resumen los datos disponibles para que los profesores tengan toda la información necesaria y puedan finalizar la corrección. Esto incluye descargar el registro de pruebas del estudiante y ver la cantidad de defectos reportados, de los cuales se muestra cuántos no fueron detectados.

También se muestra una nota sugerida que se calcula a partir de un algoritmo basado en ciertos criterios de corrección.

e. Finalmente y como parte del resumen de sesión del usuario, los profesores podrán escribir una devolución a los estudiantes y asignar una nota final.

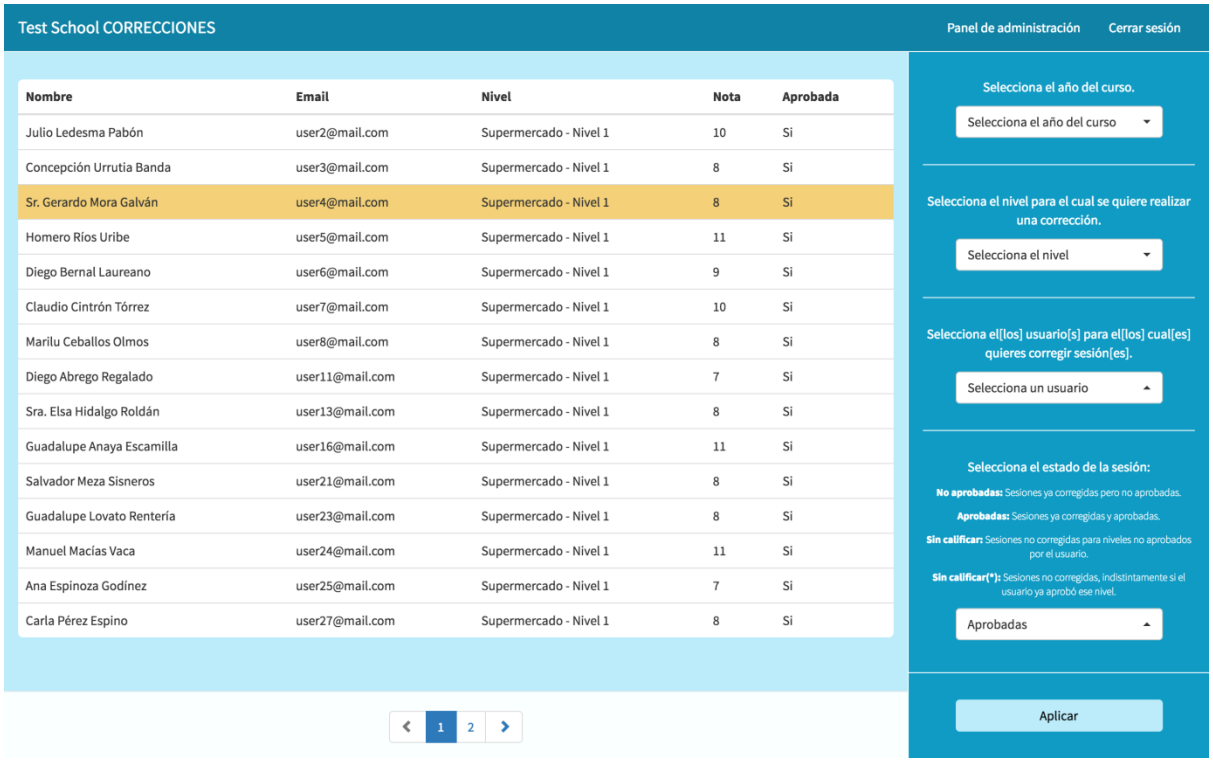


Figura 4.4: Panel principal de correcciones.

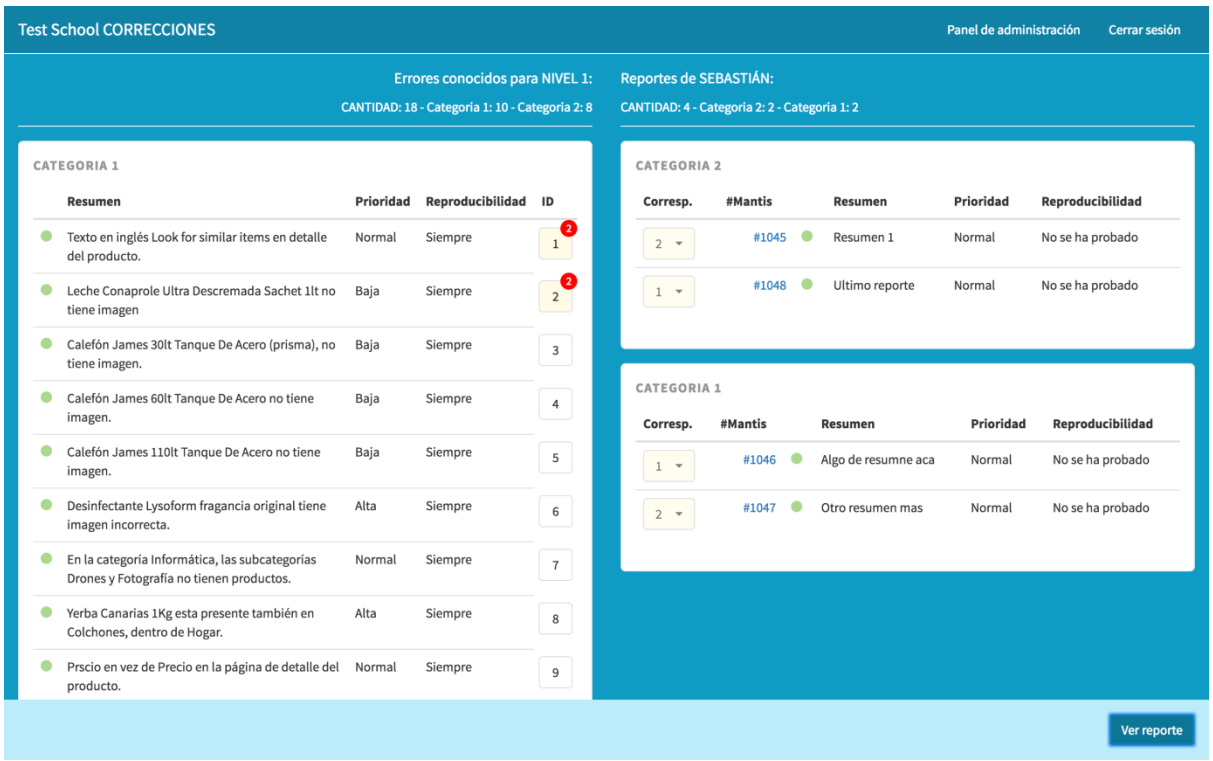


Figura 4.5: Apareo de bugs conocidos y bugs reportados por un usuario.

4.4. Versión inicial

4.4.1. Escenarios iniciales

Se armó un conjunto de escenarios iniciales de la aplicación para que el sistema pudiera ser utilizado de inmediato, luego de ser implantado en los servidores del CES. Esto implica que se debía contar con más de una mutación (ya sea distintas aplicaciones o mutaciones de una misma aplicación) para que al menos una de ellas se ejecute en cada nivel.
Para organizar la versión inicial se crearon 5 niveles con una dificultad asociada, que fueron generados en base a las capacidades de los testers descritas en la sección 1.1.2.
En los siguientes anexos se detallan diversos aspectos de esta versión:

A. Detalle de defectos inyectados para cada mutación, justo con las técnicas que se sugieren para detectarlos: **Defectos inyectados y técnicas sugeridas.**
B. Detalle de los escenarios planteados a los estudiantes, que abarca descripción, objetivo y especificaciones: **Escenarios presentados a estudiantes.**
C. Especificaciones de la realidad para cada nivel, justificando la presencia de los defectos inyectados, apuntando a los conocimientos que los estudiantes deberían tener en cada uno de ellos, incluyendo aquellos parámetros que son manejados desde el panel de administración de Spree: **Especificaciones de escenarios para aplicación Supermarket.**

4.4.2. Aplicación utilizada

Como resultado de la investigación explicada en el estado del arte, en la sección 3.8, se optó por utilizar la aplicación Spree[36], que fue la que obtuvo mayor puntuación global y por esta razón fue elegida para ser integrada al sistema.
Ésta es la aplicación sobre la cual se inyectan defectos conocidos que luego deben ser detectados por los estudiantes (usuarios).

4.5. Lenguajes utilizados

4.5.1. Capa de acceso a datos del sistema

Para soportar la lógica de la aplicación se eligió el framework de aplicaciones web Ruby on Rails, escrito en el lenguaje Ruby, que brinda las facilidades necesarias para soportar el sistema a construir.
Los principales factores tomados en cuenta para elegir esta tecnología fueron:

- entorno de desarrollo conocido: todos los integrantes del grupo tienen experiencia con el lenguaje y el framework, lo cual hace más rápida la implementación.
- lenguaje de código abierto: el principal beneficio es la libre redistribución de los productos implementados en esta plataforma.
- cuenta con una comunidad activa y extensa: esto resulta sumamente beneficioso para resolver problemas que puedan presentarse, ya que existe registro de miles de ellos que han sido publicados y resueltos. Además se puede llegar a obtener una rápida y útil respuesta si surge una duda o problema desconocido.
- versatilidad para construir aplicaciones web con amplia variedad de bibliotecas disponibles

4.5.2. Capa de presentación

Al momento de elegir una tecnología para la capa de presentación, la decisión no resultó tan directa como el caso anterior, ya que existen varias opciones que cumplen con los requisitos y son compatibles con el framework elegido.
Se optó por AngularJS para construir la capa de presentación, tomando en cuenta los siguientes factores:

- Incluye prácticamente todos los aspectos necesarios para crear una aplicación cliente. Esto incluye la generación de vistas, el uso de databinding, las rutas, la organización de componentes en módulos y la comunicación con el servidor.
- El sistema de databinding es muy completo y potente, además de ser fácilmente extensible mediante directivas y filtros, lo que permite crear pequeños componentes que facilitan la organización de la capa de presentación.
- Escalabilidad.
- Posibilidad de utilizar una de las tecnologías emergentes en los últimos años, lo cual resulta un aspecto sumamente motivante para los integrantes del proyecto.
- Cuenta con una comunidad activa.
- Lenguaje de código abierto.

4.6. Herramientas utilizadas

4.6.1. Repositorios

4.6.1.1. Código

Para el manejo colaborativo de repositorios durante el desarrollo se utilizó BitBucket la cual permite:

- contar con un mecanismo de pull request para mantenibilidad y calidad de código.
- privacidad del código.

4.6.1.2. Documentos

Para manejar los documentos en los que se trabajó a lo largo del proyecto se creó un repositorio en Google Drive que brinda una plataforma sencilla e intuitiva para colaborar en la edición de documentos.

4.6.2. Manejo de tareas

Para la asignación de tareas a lo largo del proyecto se utilizó la herramienta Trello, que permite crear un tablero con tareas organizadas según su estado.
También se utilizaron planillas para indicar el progreso de las tareas y persona asignada a ellas.

4.6.3. Calidad de código

Se utilizaron herramientas para evaluar el código implementado y garantizar que éste sigue las buenas prácticas y los estándares recomendados.

- RuboCop: es un analizador estático de código Ruby, basado en la guía de estilo que utiliza como referencia la comunidad. [38]
- Reek: es una herramienta que examina las clases de Ruby, módulos y métodos para luego reportar deficiencias o posibles mejoras en el código.
- Rails best practices: es una herramienta que brinda métricas para chequear el uso adecuado del framework Ruby on Rails. [39]

4.6.4. Test unitario

Para implementar y ejecutar los tests unitarios se integró la gema Rspec.
Ésta provee un framework para el desarrollo guiado por comportamiento, lo cual permite escribir escenarios de una aplicación y probarlos [40]. Se pueden ver ejemplos de estos tests en el anexo “Registro de pruebas”.

4.6.5. Test de integración

Para simular la interacción entre un usuario y el sistema en una aplicación web, se utilizó Capybara. Ésta es una gema de Rails que se integra fácilmente en el proyecto y provee Selenium como soporte para simular un determinado flujo [41]. Se pueden ver ejemplos de estos tests en el anexo “Registro de pruebas”.

4.6.6. Test de rendimiento

Para realizar pruebas de desempeño y carga se utilizó Apache Bench [42], nos permitió obtener algunas métricas básicas pero necesarias en corto tiempo. Se pueden ver ejemplos de estos tests en el anexo “Registro de pruebas”.

4.7. Requisitos que cambiaron

4.7.1. Registro de la sesión

En un principio se había solicitado que las sesiones de los usuarios fueran grabadas en video (de pantalla) para registrar todos sus movimientos y acciones.
Estas grabaciones estarían disponibles en el panel de administración y su durarían desde el momento que el usuario inicia la sesión hasta que la envía para su corrección.
Al momento de analizar esta funcionalidad e intentar construir un prototipo se observaron los siguientes problemas:

- **Tiempo total de la sesión**: si el usuario deja su sesión abierta por un largo tiempo la grabación continuaría, generando un video muy extenso. Esto implicaría una revisión poco viable ya que sería poco productivo mirar videos tan largos
- **Tamaño de los archivos de video**: almacenar archivos de gran tamaño sería un problema si no se cuenta con un repositorio adecuado. Además el manejo y subida de dichos archivos concurrentemente (para varios usuarios al mismo tiempo), puede generar un cuello de botella.
- Al momento de construir el prototipo no existían herramientas que permitan grabar pantallas en servidores remotos: la principal razón por la cual no existen es debido a temas de seguridad. No se permite acceder de forma remota a otros servidores y grabar la pantalla de otra máquina si ésta no da permiso previamente.

Debido a los inconvenientes para construir esta funcionalidad, se discutió la posibilidad de permitir a los usuarios tomar capturas de pantalla y comentarlas (agregarles un texto) para que ésto sirviera como registro de sesión pero finalmente se optó por dejar que los usuarios armen un documento con el flujo que siguieron y lo suban como parte del registro de las pruebas. En caso de que creen distintos documentos como evidencia de las pruebas, se les permite subir un archivo empaquetado con todos los que consideren necesarios.

4.7.2. Inserción de mutaciones

Como se explica en la sección 7.3, se analizó en primer lugar la posibilidad de insertar defectos de forma automatizada en una aplicación y generar la mutación correspondiente.
Finalmente se optó por generarlas manualmente para controlar las modificaciones realizadas y evitar inconsistencias.
Esto nos permitió adaptar las mutaciones a las necesidades del curso ya que, nos focalizamos en que los mutantes tengan defectos similares a los que se generan en el proceso de desarrollo de software.

La inserción automática de defectos automáticamente se considera una posibilidad a futuro ya que brindaría una mayor versatilidad al momento de generar variantes defectuosas de una aplicación.

5. Arquitectura

Este capítulo describe la arquitectura elegida de la aplicación. Se presentan las tecnologías utilizadas, cómo se comunican entre sí, servicios expuestos y modelado de la realidad propuesto.

5.1. Organización

Se comienza dando un panorama general de la aplicación, se presentan los distintos componentes, cómo se comunican y qué tecnología se escogió para cada uno de ellos.

Para un mejor entendimiento y organización, se presentan los distintos agentes que interactúan con el sistema y qué componentes involucra cada uno de ellos.

Luego se introduce un vistazo general en los conceptos básicos y necesarios de las tecnologías utilizadas con el fin de facilitar el entendimiento al lector.

Finalmente, se tiene una presentación y breve descripción de los servicios expuestos y el modelo de dominio, lo que brindará una visión más profunda del sistema.

En el anexo “Gestión de seguridad” se brinda información referente a la seguridad de la sesión de los usuarios, mientras que en el anexo “Servicios de API” se presentan los servicios expuestos por la API.

5.2. Presentación

Al presentar el estilo arquitectónico del sistema se debe mencionar que existen cuatro tipos de usuarios diferentes. Cada uno de ellos se relaciona con distintas partes de la arquitectura pero todos forman parte de una arquitectura cliente-servidor.

A continuación (en la figura 5.1) se exhibe un diagrama de la arquitectura del sistema y cada uno de sus componentes:

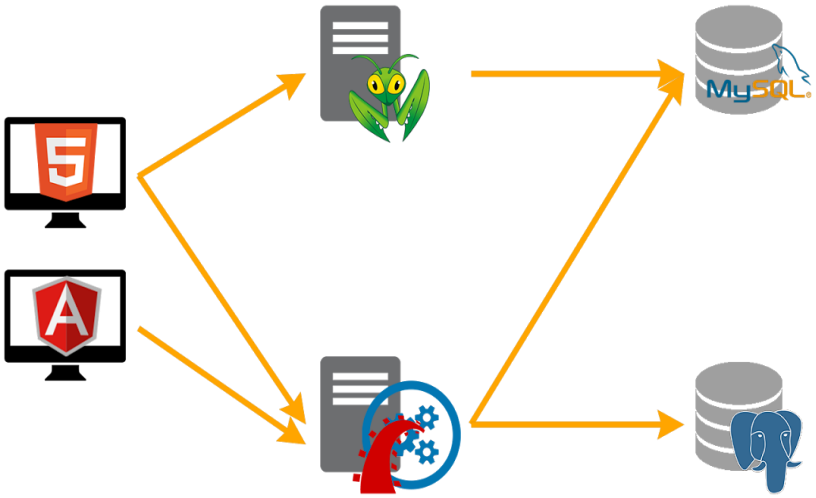


Figura 5.1: Diagrama que representa la arquitectura del sistema y como se comunican los componentes entre sí.

En las secciones venideras se introducen los usuarios, las responsabilidades de cada uno de ellos y una breve descripción de los elementos de la arquitectura a los que involucra.

5.3. Usuario administrador Test School

5.3.1. Descripción

Es el responsable de gestionar cada uno de los componentes del sistema Test School: aplicaciones, niveles, mutaciones, usuarios y lista de invitados. Además, se encarga de corregir las sesiones de los usuarios tester Test School. Comúnmente es un profesor del CES.

5.3.2. Estilo arquitectónico

En la figura 5.2 se destacan los componentes con los que interactúa el usuario administrador.

Figura 5.2: Los componentes resaltados son aquellos alcanzados por un administrador.

Para el panel de administración no se utiliza ningún framework del lado del cliente, simplemente se genera código HTML, CSS y JavaScript en el servidor y el mismo es ejecutado en el navegador del cliente.

El mismo fue construido totalmente con el framework Ruby on Rails, tanto la gestión de elementos del sistema como el panel de correcciones.
Este proyecto exhibe las rutas disponibles, maneja la lógica de negocios, se comunica con la base de datos y finalmente retorna la vista HTML o JSON a cliente.

Para gestionar los datos del panel de administración se utiliza una base de datos PostgreSQL. Además existe una comunicación con la base de datos MySQL para obtener los datos de Mantis.

5.4. Usuario tester Test School

5.4.1. Descripción

Es comúnmente un usuario que está realizando uno de los cursos del CES y accede al sitio Test School donde puede testear las mutaciones disponibles de cada aplicación. A su vez, cuenta con un perfil en donde puede visualizar fácilmente su avance y evaluaciones corregidas de las sesiones en las que ha trabajado hasta el momento.

5.4.2. Estilo arquitectónico

En la figura 5.3 se presenta un diagrama de componentes involucrados para el usuario Test School.

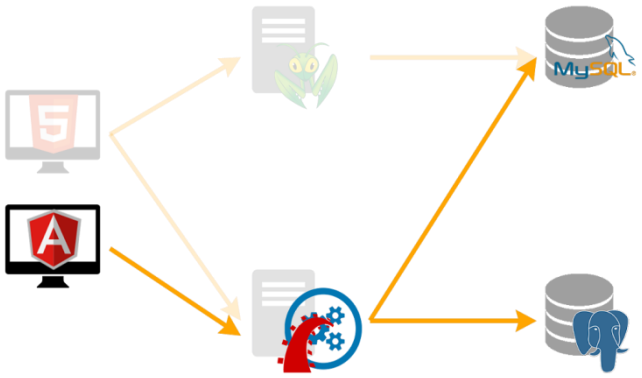


Figura 5.3: Los componentes resaltados son aquellos alcanzados por un usuario administrador.

El usuario hace uso del framework AngularJS el cual se ejecuta directamente en el lado del cliente en el navegador. Este se encarga de la comunicación con el servidor en nombre del usuario. Sigue el patrón de arquitectura MVC y es una SPA lo cual permite al usuario mejorar su experiencia en el sitio de manera considerable.

El servidor es una API REST Ruby on Rails, que al igual que para el usuario administrador, sigue el patrón de arquitectura MVC. La única diferencia es que la vista es una respuesta en formato JSON y no HTML.
Es importante aclarar que existe un solo proyecto Ruby on Rails que contiene ambos componentes, la API y el panel de administración. Las ventajas de este enfoque se explican en 4.5.1.

De la misma manera que con el panel de administración, la API se comunica mayoritariamente con la base de datos PostgreSQL.
Hay también comunicación con la base de datos MySQL por las mismas razones expuestas anteriormente para el panel de administración.

5.5. Usuarios Mantis

5.5.1. Administrador

Al igual que los usuarios administradores de Test School, es un docente de algún curso del CES. Su principal tarea es gestionar los componentes relacionados al proyecto Mantis.

5.5.2. Tester

Una vez que el estudiante se registra como usuario Test School, se crea automáticamente un usuario Mantis con el rol “desarrollador”. Con este usuario se pueden reportar y modificar incidencias de cada uno de los niveles de las aplicaciones del sistema, siempre y cuando el usuario tenga acceso al mismo.

5.5.3. Estilo arquitectónico

Los dos usuarios ya descritos fueron un requisito planteado al comienzo del proyecto.

Para esto, se instaló el proyecto MantisBT, el cual es un proyecto abierto y fácilmente configurable en cualquier entorno. El mismo, en conjunto con una base de datos MySQL conforman la arquitectura necesaria para dar soporte a cualquiera de las acciones que realizan los dos usuarios Mantis. En este caso no existe un framework en el lado del cliente, siendo el servidor el que retorna vistas HTML junto con estilos CSS y código JavaScript.
Su arquitectura no se explica en detalle por ser un proyecto que sólo fue instalado y configurado para funcionar en conjunto con nuestro sistema.

Figura 5.4: Los componentes resaltados son aquellos alcanzados por los usuarios Mantis, administradores y desarrolladores.

5.6. Tecnologías utilizadas

En esta sección se describe cada una de las tecnologías utilizadas por los componentes de la arquitectura que fueron presentados anteriormente. Se explicará el funcionamiento básico de cada una y de qué manera son utilizadas por los componentes.

5.6.1. Ruby on Rails

Ruby on Rails es un framework de código abierto para construir aplicaciones web. Fue creado con la premisa de que sea fácil de programar y altamente productivo.
Algunos de los principios que sigue son:

- Convención sobre configuración: El framework asume un gran número de situaciones haciendo más rápido el desarrollo, pero nunca perdiendo flexibilidad si es necesario un cambio.
- No reinventar la rueda: no construir nuevamente algo que ya fue solucionado. En este contexto existen un gran número de gemas disponibles ya creadas por la comunidad.
- No te repitas: es un principio de software que se basa en reducir la repetición de código.

El framework impone utilizar el patrón de arquitectura MVC, ya sea para construir una API o una aplicación web, el último es el caso del panel de administración.

A continuación se describe cada uno de los componentes principales de un proyecto Ruby on Rails:

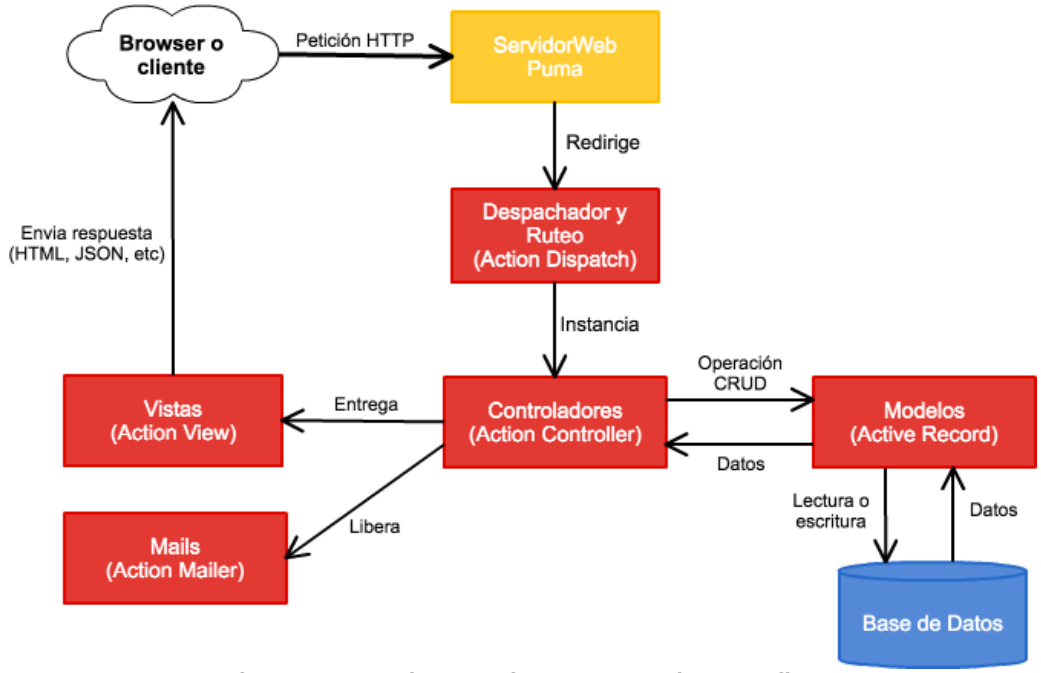


Figura 5.5: Arquitectura de proyecto Ruby on Rails

Servidor Web

El servidor web se encarga de recibir los pedidos HTTP y redirigirlos al stack del proyecto Ruby on Rails. En nuestro caso, se buscó un servidor web rápido y con un alto manejo de concurrencia. Es por esto que elegimos Puma, el cual cumple con estas características. El mismo es considerado el servidor web multihilo más rápido y con mejor uso de memoria dentro de los servidores web disponibles [43].
Puma soporta cualquier aplicación Rack, siendo Ruby on Rails una de ellas. Siguiendo este conjunto de especificaciones el servidor web traduce el pedido HTTP a un lenguaje mucho más entendible para el proyecto Ruby on Rails.

Despachador y Ruteo

El servidor web muchas veces no es considerado parte del stack de un proyecto Ruby on Rails. Por lo tanto el despachador es el primer componente involucrado cuando arriba un pedido HTTP.

El despachador ActionDispatch se encarga del ruteo, parseo de parámetros, manejo de cookies y manejo de sesión, entre otras tareas.
El ruteo es una tarea muy importante, significa mapear una URL y un método HTTP a un controlador Rails y un método del mismo. Esta tarea es considerada invisible para el programador, que solamente define las rutas en el archivo de configuración *routes.rb*.
Vale destacar que cada una de las rutas de la API están bajo el espacio de direcciones */api/v1,* mientras que las rutas expuestas para el usuario administrador por */admin*, en el caso de la pantalla principal del panel de administración y por */corrections,* para el caso del panel de correcciones.

Por otra parte, un usuario está habilitado a acceder y manipular los datos en el panel de administración mediante el manejo de cookies y session. Sin embargo, estos conceptos no tienen sentido en la API, el usuario debe enviar un token para identificarse en cada petición.

Controladores

El módulo ActionDispatch determina qué controlador y qué método será utilizado para la petición, creando una instancia de dicho controlador e indicando que ejecute dicho método.

Las tareas que realiza el controlador son: comunicarse con los modelos en primer lugar y luego retornar la vista correspondiente junto con el estado de la respuesta.
Por esto se dice que se debe seguir el enfoque de modelos gordos y controladores delgados, dado que son simplemente un intermediario entre los modelos y las vistas.

En los controladores también se realizan tareas como el procesamiento de los parámetros que llegan en una petición, manejo de cookies y sesión.

Como se mencionó anteriormente, la API y el panel de administración se dividen en rutas separadas semánticamente. En el caso de los controladores sucede lo mismo. Aquellos relacionados a la API se encuentran bajo la carpeta *api/v1* y los relacionadas al panel de administración bajo la carpeta */admin* y */corrections*.

Modelos

Los modelos son los encargados de la comunicación con la base de datos así como también del manejo de gran parte de la lógica de negocios.

Para lograr lo primero, es menester introducir a uno de los principales módulos de Ruby on Rails, ActiveRecord, el cual es un ORM.
Un ORM se encarga de conectar objetos de la capa de aplicación con tablas de una base de datos relacional. De esta manera las propiedades y relaciones entre objetos de la capa de aplicación pueden ser consultadas y almacenadas fácilmente en la base de datos sin necesidad de escribir SQL.
Es importante destacar que el ORM también permite implementar consultas SQL si es necesario, algo común en consultas de alta complejidad. Sin embargo, dado que es un ORM muy completo y con alta madurez permite realizar la gran mayoría de las operaciones SQL.

El manejo de la lógica de negocios está intrínsecamente relacionado con la comunicación con la base de datos, ya sea para almacenar o consultar datos. El tratamiento u operaciones con los datos es realizado en los modelos.
Cabe destacar que el framework Ruby on Rails ha evolucionado y ante la necesidad de liberar de responsabilidades a modelos extremadamente grandes, nuevos componentes han surgido. Por ejemplo, se implementaron servicios para comunicación con APIs externas, o presentadores para exponer datos que el modelo ya expone, pero con distinta organización.

A diferencia de las rutas y los controladores, los modelos son compartidos por la API y el panel de administración. Esto es posible ya que ambos manejan los mismos datos y aplican a la misma lógica de negocio. Esta es una de las grandes ventajas de tener ambos componentes, la API y el panel de administración en el mismo proyecto Ruby on Rails.

Vistas

Una vista es la forma en la cual la aplicación expone los resultados de la petición HTTP al cliente. Como se mencionó, cada método de un controlador responde a determinada ruta expuesta por el sistema, por lo cual también tiene su vista correspondiente.
La misma puede tener diferentes formatos, en el caso de nuestro proyecto Ruby on Rails estas son HTML con código ruby embebido en el caso del panel administrador o JSON en el caso de la API.

Las vistas HTML con código ruby embebido son instancias de la clase ActionView::Base. El módulo ActionView provee un número muy grande de helpers para el manejo de diferentes escenarios clásicos en las vistas. Por ejemplo, formato adecuado de una fecha.

Las vistas con formato JSON son generados por la gema Jbuilder, con la cual los integrantes han tenido muy buenas experiencias.

Emails

Los mails funcionan de manera muy similar a los controladores, cada mailer hereda de la clase ActionMailer::Base y tiene una vista asociada para cada uno de sus métodos.
Básicamente, la vista se usa para reflejar el cuerpo del mail que recibirá el remitente y comúnmente tiene formato HTML con código Ruby embebido. Las vistas pueden ser texto plano, dando soporte a clientes que no soporten el despliegue de correos con código HTML.

Dado que el envío de un mail puede tomar suficiente tiempo como para ser percibido por el usuario y retrasar la respuesta, es importante destacar que se envían en background. Para esto se utiliza la gema Delayed Job, la cual serializa el código a ejecutar, lo guarda en la base de datos y luego lo ejecuta en proceso aparte.

5.6.2. AngularJS

AngularJS es un framework para construir sitios web dinámicos. Utiliza HTML como lenguaje base para mostrar su contenido y a su vez permite extender su sintaxis para hacer más fácil y accesible la codificación de los componentes del sitio. Además todo el procesamiento sucede del lado del cliente, haciendo que AngularJS sea una muy buena opción a la hora de elegir un lenguaje para acompañar cualquier tipo de servidor.

Se alega que AngularJS es “lo que HTML debería haber sido en un principio”. Esto se debe a su alto grado de utilidad para construir aplicaciones. Además HTML es prácticamente un estándar a la hora de elegir un lenguaje para representar visualmente una aplicación, pero el mismo es estático, por lo que el programador tiene que “engañar al navegador para conseguir el comportamiento que desea”.

La distancia que existe entre la necesidad de aplicación con contenido dinámico y documentos estáticos se suele solucionar mediante el uso de alguna herramienta externa como ser JavaScript, jQuery u otros frameworks basados en este último.

Angular pretende acortar esta brecha mediante componentes llamados “directivas”, como por ejemplo:

- Data binding, mediante la directiva “{{}}”.
- Estructuras de iteración en el DOM.
- Estructuras para mostrar u ocultar elementos del DOM de acuerdo a condiciones o estados en los datos.
- Soporte para formularios y control sobre sus elementos. Por ejemplo que la entrada de email tenga efectivamente una dirección de correo, o que la entrada en un campo de edad sea un valor numérico.
- Agregar manejo de acciones a los elementos del DOM según eventos.
- Uso de vistas parciales o fragmentos, los cuales pueden ser reutilizados en distintas partes de la aplicación.

Mencionaremos los principales componentes de AngularJS que harán más comprensible el framework.

Scopes

Scope es lo que se asocia al modelo en la visión del patrón MVP en AngularJS. Los scopes se ordenan de manera jerárquica y son los encargados de manipular el DOM, observar expresiones (data binding) y propagar eventos en el sistema.
Se puede acceder a él mediante la variable $scope en las vistas y todos descienden del mismo padre que es $rootScope.

Controladores

Un controlador es utilizado para extender el alcance del $scope de Angular hasta la vista, la cual inicializa una nueva instancia del mismo.

El controlador se asocia a la vista mediante la directiva "ngController". A partir de aquí se crea una nueva instancia del controlador invocando la función de creación del mismo.

Los controladores son utilizados para:

- Inicializar el estado del $scope. Por ejemplo, se puede obtener los datos del usuario desde la API para luego guardarlos en el $scope y finalmente ser mostrados cuando se termina de cargar la página de perfil.
- Agregar comportamientos al $scope mediante funciones. Un botón que redirige a otra página del sitio, puede tener asociado al evento click dicho comportamiento mediante una función del scope definido en el controlador.

Los controladores no deben ser utilizados para manipular elementos del DOM como se haría normalmente con JavaScript o jQuery. Para esto existe el concepto de Data Binding y su directiva "{{ }}", así como otras directivas, ya previstas por Angular o definidas por el programador para su aplicación en concreto.

En resumen los controladores no deben hacer mucho, solamente contener la lógica del negocio de una vista en particular.

Vistas

Se utiliza el lenguaje HTML para la construcción de las vistas con la salvedad que se pueden utilizar etiquetas (directivas) brindadas por AngularJS. Éstas, combinadas con la información en los modelos y las acciones definidas en los controladores permiten construir sitios dinámicos.

Directivas

Las directivas son etiquetas o marcadores que le otorgan comportamientos a los elementos del DOM que las contienen, que en algunos casos pueden afectar a sus hijos.
Algunos ejemplos son ngClick, ngClass, ngRepeat, etc, que permiten hacer uso de la información del modelo o el controlador para ejecutar un comportamiento en particular.
Se cuenta con la flexibilidad para crear nuevas directivas para cubrir las necesidades puntuales de un proyecto.

Servicios

Los servicios son objetos que contienen código a distribuir en cualquier otro componente de la aplicación. Sirven como herramienta para organizar de mejor manera el proyecto.
Tienen como particularidad, que los servicios se instancian en el mismo momento que se invocan. De esta manera se pueden crear un gran número de servicios sin afectar el rendimiento del sistema. Además tienen una instancia única (patrón singleton).

5.6.3. PostgreSQL y MySQL

Son gestores de bases de datos relacionales de uso gratuito. PostgreSQL es un proyecto de código abierto desarrollado por una comunidad de programadores, mientras que MySQL es patrocinada por Oracle Corporation haciendo que gran parte de su código sea privado.

No se ahondará en el tema debido a su amplia aceptación en el mercado actual, y todo lo que se pueda especificar resulta poco y redundante.

5.7. Justificación

Una vez presentada la arquitectura y las tecnologías utilizadas, se procederá a justificar la utilización de cada una de ellas para cada componente.

5.7.1. Usuario tester Test School

Para los usuarios testers, el enfoque que se tomó fue utilizar el framework AngularJS, ya que es una tecnología emergente que ha tenido muy buenas críticas según lo expresado en [44]. Además brinda herramientas que facilitan el desarrollo de sitios dinámicos, lo cual mejora la experiencia de usuario. Sigue un patrón MVC al igual que Ruby on Rails y es una SPA.

El servidor Ruby on Rails delega la generación de las vistas a AngularJS, lo cual permite un mayor rendimiento. Esto se debe a que los tamaños de las respuestas son mucho menores si se compara una respuesta JSON contra una respuesta HTML.

Otra ventaja de usar AngularJS es que obliga al servidor a exponer servicios REST, los cuales pueden ser utilizados en un futuro por otra aplicación, ya sea web como en este caso, o móvil.

La ventaja más grande es la posibilidad de distribuir todos los componentes físicamente. Esto permite escalar de manera horizontal de manera sencilla, permitiendo un número alto de usuarios concurrentes. El frontend se puede separar del backend; las distintas aplicaciones de los niveles se podrían ubicar en servidores distintos; las bases de datos se podrían separar del backend en última instancia si fuera necesario.

La desventaja del enfoque presentado es que el equipo de desarrollo no contaba con demasiada experiencia en AngularJS al momento de comenzar el proyecto, lo cual implicó que la velocidad inicial de implementación fuera algo más lenta respecto a la del backend, sin embargo, debido a las ventajas ya mencionadas, se decidió utilizar este framework.

Por otra parte, se eligió utilizar PostgreSQL como base de datos por dos razones; la primera es porque el equipo de desarrollo tiene experiencia trabajando con este manejador de base de datos así como con herramientas relacionadas para la visualización de datos. A su vez, existen razones altruistas dado que este manejador de base de datos es soportado por una comunidad y es de código abierto. Otra opción era MySQL la cual también posee código abierto pero sin embargo es patrocinada por una empresa privada.

5.7.2. Usuario administrador Test School

Inicialmente se consideró para el panel de administración seguir el mismo enfoque que para el usuario tester Test School, lo cual hubiera implicado construir dicho panel utilizando un cliente AngularJS y una API Ruby on Rails.

Los principales beneficios de este enfoque serían: una mejor experiencia de usuario, que se puede lograr utilizando un framework del lado del cliente como AngularJS; el cliente AngularJS consumirá

una API Ruby on Rails que puede ser reutilizada por otros clientes, lo cual resultaría provechoso si se decidiera extender el sistema y construir una aplicación móvil en un futuro.

Sin embargo, estas ventajas no son lo suficientemente relevantes para un panel de administración, donde se busca priorizar la efectividad de las funcionalidades sobre la experiencia del usuario.

Por lo tanto, el usuario administrador se construyó íntegramente con el framework Ruby on Rails. Además, la razón que tuvo más peso a la hora de tomar esta decisión, es la experiencia de los integrantes del grupo utilizando este framework y más específicamente implementando paneles de administración.

Se utilizó la gema ActiveAdmin, la cual es un DSL que facilita la creación de este tipo de paneles y junto con la gema Devise, que se encarga de manejar la autenticación del usuario administrador. Estas librerías permiten agregar un gran número de funcionalidades para este usuario en un tiempo razonablemente rápido.

5.7.3. Usuarios Mantis

En un principio se pensó implementar un sistema propio para el registro de incidentes, el cual nos permitiría la total manipulación y organización de los datos.
Sin embargo, el uso de Mantis fue un requisito obligatorio dado que es de alto interés para el equipo docente del CES que los estudiantes aprendan a utilizar esta herramienta para reportar incidentes.

Como es lógico, el proyecto Mantis tiene su estructura de datos definida la cual necesita de una base de datos propia.
Tanto el usuario administrador como el usuario Test School necesitan utilizar los datos generados por Mantis, ya sea para correcciones en el caso del primero, como para visualizar datos estadísticos en el caso del segundo.
El desafío de la arquitectura como un todo fue cómo gestionar los datos generados por Mantis desde el proyecto Ruby on Rails.

Una primera opción era tener una misma base de datos para ambos proyectos, lo cual fue descartado rápidamente dado que una de las premisas del proyecto era poder escalar tanto horizontal como verticalmente. Además esto brinda la flexibilidad de acoplar un proyecto MantisBT ya existente en un futuro con el mínimo esfuerzo de configuración.
Para compartir los datos entre ambos proyectos se plantearon dos opciones: exponerlos mediante una API SOAP incluida en el proyecto Mantis, o acceder a la base de datos de Mantis desde el proyecto Ruby on Rails.

La primera opción fue descartada rápidamente debido a que los servicios que exponía dicha API eran SOAP, lo cual no cumplía con nuestras premisas expresadas en la sección 2. Además, la API SOAP no proveía todos los servicios necesarios para asociar los datos.

Por lo tanto, se optó por la otra alternativa: acceder a la base de datos utilizada por Mantis desde el proyecto Ruby on Rails. El acceso a una base externa de datos desde Ruby on Rails se logra instalando el controlador de base de datos correspondiente (mediante la gema ¨mysql2¨) e indicando la base de datos a la que apunta el modelo (si no se indica ninguna base de datos, se utiliza la base de datos por defecto, PostgreSQL).

Finalmente, la razón de mayor peso a la hora de utilizar MySQL es que el propio proyecto MantisBT advierte de no utilizar PostgreSQL al elegir la base de datos, indicando la primera por defecto.

5.8. Diagrama de clases

Como última sección dentro del capítulo de arquitectura, se presenta el diagrama de clases para la API Ruby on Rails para una mejor comprensión de la estructura de clases, sus atributos, operaciones y relaciones entre los objetos.

Cada artefacto del modelo UML presentado en la figura 5.6 hace referencia a modelos Ruby on Rails. Como se explicó en 5.6.1, los modelos son los encargados de la lógica de negocio y la comunicación con la base de datos.

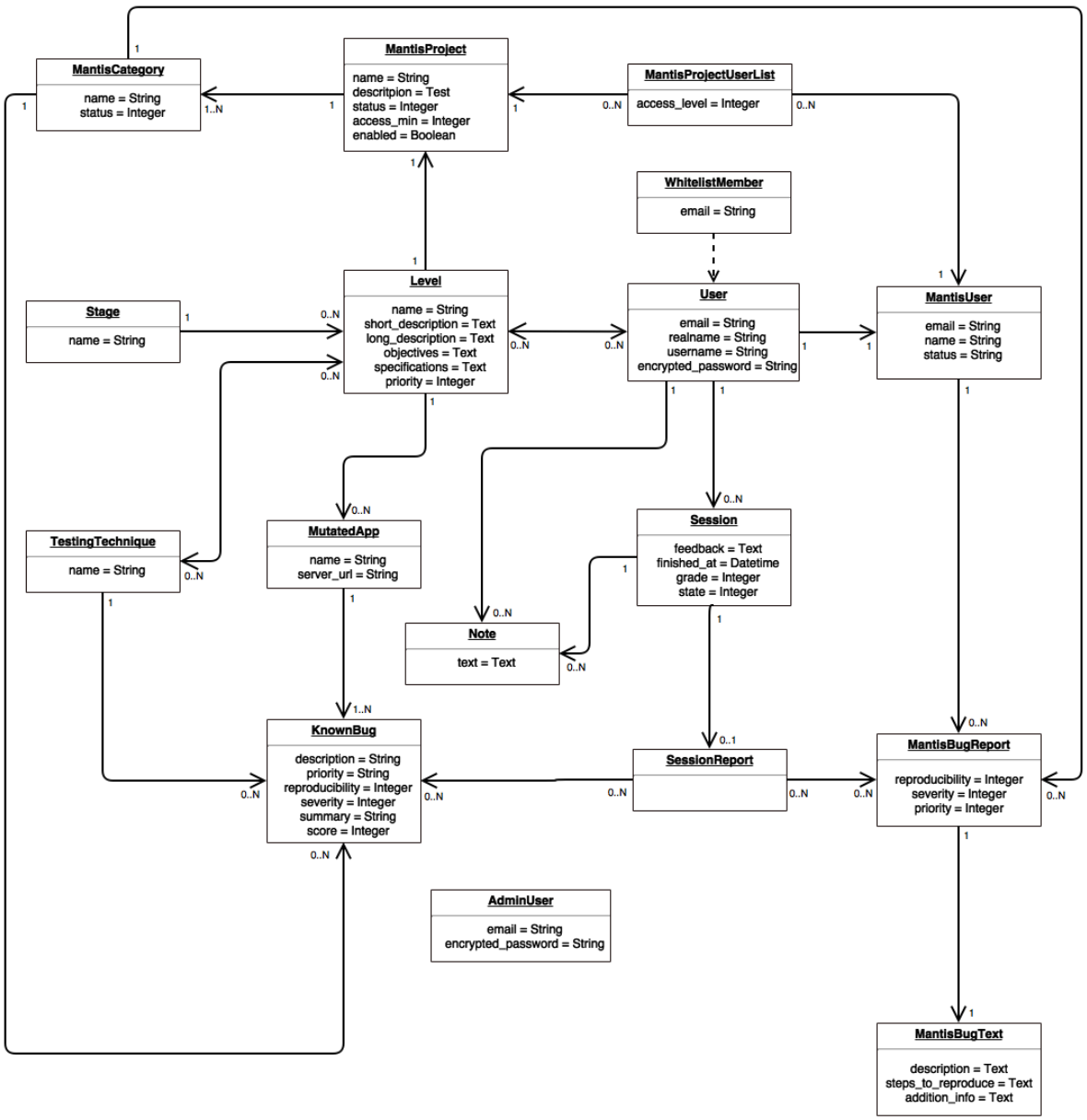


Figura 5.6: Diagrama de clases de Test School.

Se presentan los atributos de cada uno de los modelos así como también una breve descripción de cada uno de sus métodos públicos más importantes.
Se describen brevemente las validaciones de negocio que cada objeto debe cumplir para poder ser persistido en la base de datos correspondiente.

Para describir los métodos se utiliza la nomenclatura *self.nombre_metodo* para representar un método de clase o *nombre_metodo* si es de instancia.

5.8.1. Stage

Representa una aplicación del sistema. Se relaciona con la tabla *stages* en la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
name	String	Nombre.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
levels	Vacía	Colección de Levels	Devuelve todos los niveles de la aplicación.

Validaciones

Nombre/Atributo	***Description***
Presencia de nombre	El nombre no puede ser vacío.
Unicidad de nombre	El nombre no puede estar repetido.

5.8.2. Level

Representa un nivel de una aplicación del sistema. Se relaciona con la tabla *levels* en la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
name	String	Nombre.
priority	Integer	Orden en la colección de niveles respecto a la aplicación a la que pertenece.
short_description	Text	Resumen.
long_description	Text	Descripción.

objectives	Text	Objetivos.
specifications	Text	Especificaciones.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
stage	Vacía	Stage	Retorna la aplicación a la que pertenece.
mantis_project	Vacía	MantisProject	Retorna el proyecto mantis con el cual se relaciona.
mutated_apps	Vacía	Colección de MutatedApps	Retorna todas las aplicaciones mutadas.
parent_levels	Vacía	Colección de Levels	Retorna los niveles padres.
children_levels	Vacía	Colección de Levels	Retorna los niveles hijos.
new?	User	Boolean	Retorna si es un nivel nuevo para el usuario; si no tiene sesiones.
allowed?	User	Boolean	Retorna si el usuario tiene acceso para cursar.
approved?	User	Boolean	Retorna si el usuario ha aprobado.
self.initial_levels	Vacía	Colección de Levels	Retorna los niveles que no tienen padre para cada una de las aplicaciones. Son aquellos que el usuario puede cursar sin haber aprobado un nivel previo.

Validaciones

Nombre/Atributo	***Descripción***
Presencia de nombre	El nombre no puede ser vacío.
Unicidad de nombre en un ámbito determinado	El nombre no puede estar repetido dentro de la aplicación a la que pertenece.
Presencia de prioridad	La prioridad no puede ser vacía.
Presencia de aplicación	La aplicación no puede ser vacía. El nivel debe pertenecer a una aplicación.
Presencia de resumen	El resumen no puede ser vacío.

5.8.3. MutatedApp

Representa una aplicación mutada del sistema relacionada a un nivel. Se relaciona con la tabla *mutated_apps* en la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
name	String	Nombre.
server_url	String	URL en donde se encuentra alojada.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
level	Vacía	Level	Retorna el nivel al que pertenece.
known_bugs	Vacía	Colección de KnownBugs	Retorna los bugs conocidos.
full_name	Vacía	String	Retorna la concatenación del nombre del nivel al que pertenece y el suyo propio.

Validaciones

Nombre/Atributo	***Description***
Presencia de nombre	El nombre no puede ser vacío.
Unicidad de nombre en un ámbito determinado	El nombre no puede estar repetido dentro del nivel al que pertenece.
Presencia de url	La URL no puede ser vacía.
Unicidad de servidor url en un ámbito determinado	La URL no puede estar repetida dentro del nivel al que pertenece.

5.8.4. KnownBug

Representa un defecto conocido en el sistema. Es asociado a una aplicación mutada. Se relaciona con la tabla *known_bugs* de PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***

description	Text	Descripción.
summary	String	Resumen.
score	Integer	Valor numérico del 1 al 10 que representa la importancia del defecto.
severity	Integer	Valor numérico que representa la severidad. Se corresponde con el enumerado que lleva el mismo nombre definido en MantisBT. *feature: 10, trivial: 20, text: 30, tweak: 40, minor: 50, major: 60, crash: 70, block: 80*
reproducibility	Integer	Valor numérico que representa la reproducibilidad. Se corresponde con el enumerado que lleva el mismo nombre definido en MantisBT. *always: 10, sometimes: 30, random: 50, have_not_tried: 70, unable_to_duplicate: 90, 'N/A': 100*
priority	Integer	Valor numérico que representa la prioridad. Se corresponde con el enumerado que lleva el mismo nombre definido en MantisBT. *zero: 10, low: 20, normal: 30, high: 40, urgent: 50, immediate: 60*

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
mutated_app	Vacía	MutatedApp	Retorna la aplicación mutada a la que pertenece.
mantis_category	Vacía	MantisCategory	Retorna la categoría de MantisBT con la cual se relaciona.

Validaciones

Nombre/Atributo	***Description***
Presencia de descripción	La descripción no puede ser vacía.
Presencia de resumen	El resumen no puede ser vacío.
Presencia de mutación	Debe estar asociada a una mutación.
Presencia de prioridad	La prioridad no puede ser vacía.
Presencia de severidad	La severidad no puede ser vacía.
Presencia de reproducibilidad	La reproducibilidad no puede ser vacía.
Presencia de categoría de MantisBT	La categoría de MantisBT no puede ser vacía.
Presencia de puntaje	El puntaje no puede ser vacío.
Mínimo valor de puntaje	El puntaje no puede ser menor a 1.

Máximo valor de puntaje	El puntaje no puede ser mayor a 10.

5.8.5. WhitelistMember

Representa el email un estudiante que puede registrarse; es la lista de invitados del sistema. Se relaciona con la tabla *whitelist_members* de PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
email	String	Dirección de correo electrónico.

Métodos

No tiene métodos de relevancia.

Validaciones

Nombre/Atributo	***Description***
Presencia de email	El nombre no puede ser vacío.
Unicidad de email	El email no puede estar repetido.

5.8.6. Session

Representa la sesion cursada por un estudiante en un nivel y mutación determinada. Se relaciona con la tabla *sessions* de PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
grade	Integer	Calificación.
feedback	Text	Devolución del profesor.
state	Integer	Estado. Los valores posibles son: activa, finalizada y corregida.
finished_at	Datetime	Fecha que el estudiante hizo entrega la sesión para su corrección.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***

mutated_app	Vacía	MutatedApp	Retorna la mutación a la cual pertenece.
level	Vacía	Level	Retorna el nivel a la cual pertenece.
user	Vacía	User	Retorna el usuario a la cual pertenece.
notes	Vacía	Colección de Notes	Retorna las notas realizadas durante la sesión.
self.filter	Hash	Colección de Sessions	Retorna un conjunto de sesiones dentro de los parámetros recibidos.
approved?	Vacía	Integer	Retorna si se encuentra aprobada.
finish!	Vacía	Session	Retorna si se encuentra finalizada.
revert!	Vacía	Boolean	Se retorna al estado activa. Se envía un email al estudiante correspondiente notificandolo del cambio. Se debe encontrar en el estado finalizado.
bug_reports	Vacía	Colección de MantisBugReports	Retorna una colección de los bugs reportados.
suggested_grade	Vacía	Integer	Retorna la nota sugerida. Se debe encontrar en el estado finalizado.
editable?	Vacía	Boolean	Retorna si es editable.

Validaciones

Nombre/Atributo	***Description***
Presencia de aplicacion mutada	La aplicación mutada a la que pertenece no puede ser vacía.
Presencia de usuario.	El usuario al que pertenece no puede ser vacío.
Presencia de nivel.	El nivel al que pertenece no puede ser vacío.

5.8.7. Note

Representa una nota realizada por el estudiante en una sesión determinada. Se relaciona con la tabla *notes* de la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
text	Text	Texto o contenido.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
session	Vacía	Session	Retorna la sesión a la que pertenece la nota.
user	Vacía	User	Retorna el usuario al que pertenece la nota.

Validaciones

Nombre/Atributo	***Description***
Presencia de texto	El texto no puede ser vacío.

5.8.8. TestingTechnique

Representa una técnica de testing del sistema. Se relaciona con la tabla testing_techniques de la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
name	String	Nombre.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
levels	Vacía	Colección de Levels.	Retorna los niveles asociados.
known_bugs	Vacía	Colección de KnownBugs	Retorna los defectos conocidos asociados.

Validaciones

Nombre/Atributo	***Description***
Presencia de nombre.	El nombre no puede ser vacío.

5.8.9. User

Representa un usuario Test School del sistema. Se relaciona con la tabla *users* de la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
email	String	Correo.
realname	Integer	Nombre.
username	Text	Usuario.
encrypted_password	String	Contraseña encriptada.
avatar	String	URL de la imagen.

Métodos

Nombre	***Entrada***	***Salida***	***Descripción***
mantis_user	Vacía	MantisUser	Retorna el usuario mantis con el cual se relaciona.
notes	Vacía	Colección de Notes	Retorna todas las notas realizadas de todas las sesiones.
levels	Vacía	Colección de Levels	Retorna los niveles a los que se tiene acceso.
current_session!	Level.id	Session	Retorna la sesión en progreso para el nivel dado. En caso de no existir, se crea una nueva si se cuenta con acceso al nivel.
permitted_level?	Level.id	Boolean	Retorna si se cuenta con acceso al nivel.
valid_password?	String	Boolean	Retorna si la contraseña es correcta. Este es un método heredado de la gema Devise que debió ser modificado para utilizar MD5 al igual que MantisBT.
update_level_grade!	Level, Integer	Vacía	Actualiza la calificación para el nivel. Si es la primera vez que se califica se envía un mail de notificación al usuario.
grant_level_access	Level	Vacía	Otorga acceso al usuario a todos los niveles hijos del nivel.
ungrant_level_access	Level	Vacía	Remueve el acceso a todos los niveles hijos del nivel.
stage_average_grade	Stage	Float	Retorna la calificación promedio para la aplicación.

Validaciones

Nombre/Atributo	***Description***
Presencia de correo	El email no puede ser vacío.
Unicidad de correo	El email no puede estar repetido.

Presencia de nombre	El nombre no puede ser vacío.
Presencia de usuario	El usuario no puede ser vacío.
Unicidad de usuario	El usuario no puede estar repetido.
Presencia en lista de invitados	El correo debe pertenecer a la lista de invitados.

5.8.10. AdminUser

Representa un usuario administrador del sistema. Se relaciona con la tabla *admin_users* de la base de datos PostgreSQL.

Atributos

Nombre	***Tipo***	***Descripción***
email	String	Correo.
encrypted_password	String	Contraseña encriptada.

Métodos

No tiene métodos de relevancia.

Validaciones

Nombre/Atributo	***Description***
Presencia de email	El email no puede ser vacío.
Unicidad de email	El email no puede estar repetido.

5.8.11. Modelos Mantis

Aquellos modelos que están antepuestos por la palabra “Mantis” se comunican con tablas de la base de datos MySQL. Son utilizados únicamente para lectura y escritura de dicha base y así acceder a los datos relacionados al proyecto MantisBT. Sin embargo no manejan lógica de negocios, por lo cual no serán descriptos. Los mismos fueron ilustrados en el modelo de dominio para mostrar cómo se relacionan con los datos del proyecto Ruby on Rails, y solo muestran atributos relevantes [50].

6. Caso de estudio

6.1. Objetivo

Como parte del proceso se organizó un caso de estudio para que un grupo de estudiantes y profesores pudieran estudiar y evaluar las funcionalidades de la plataforma.

Como efecto adicional se detectaron errores, los cuales permitieron mejorar la calidad de Test School.

6.2. Planificación

En conjunto con las tutoras del proyecto, se conformó un grupo de estudiantes y profesores dispuestos a formar parte de este proceso. Se les suministró material para facilitar su tarea.

Se redactó una encuesta de satisfacción para obtener la opinión de estudiantes y profesores sobre cada uno de los aspectos que nos interesaba evaluar.
Ésta se creó utilizando la herramienta para crear formularios de Google.

6.2.1. Material

Se preparó y proporcionó al grupo participante un conjunto de herramientas para que pudieran utilizar el sistema sin inconvenientes.

6.2.1.1. Manual para usuarios administradores

Se construyó un manual para usuarios administradores donde se detallan los pasos para completar las principales funcionalidades disponibles en el panel de administración.
Este manual se encuentra en un documento llamado “Manual para usuarios administradores”

6.2.1.2. Videos para administradores y usuarios

Se grabaron dos instancias de demostraciones en vivo del funcionamiento del sistema; una que apunta a usuarios administradores y otra a usuarios testers, que solo tendrán acceso al portal web “Test Shool”.
Estas demostraciones se hicieron a modo de tutorial, explicando en cada paso como continuar o llevar a cabo las distintas acciones.
Estos videos se encuentran en posesión del CES.

Además, se les entregó una encuesta para obtener su opinión luego de haber examinado el sistema.

6.2.1.3. Encuesta

Se redactaron dos encuestas: una para profesores y otra para estudiantes.

En la primera se consulta por el flujo de la corrección de sesiones de los usuarios, ya que agilizar la evaluación de los estudiantes es un aspecto fundamental del sistema y uno de los principales objetivos del proyecto de grado.

En la segunda se consulta a los estudiantes acerca del funcionamiento del sitio web, conformidad con la plataforma y si la consideraron de utilidad para aprender.
También se los invita a sugerir mejoras para la misma.

Usuarios administradores

A los profesores que completaron el caso de estudio se les pidió contestar el siguiente cuestionario:

1. Marcar una de las siguientes respuestas para cada uno de los aspectos del panel de administración: Muy bueno, Bueno, Regular, Malo, Muy malo. Aspectos a evaluar:
    a. Claridad de la información
    b. Facilidad de uso
    c. ¿Qué le pareció el flujo para agregar una nueva aplicación al sistema?
    d. ¿Qué le pareció el flujo para agregar niveles y mutaciones a una aplicación?
2. Puntuando entre 1 y 5, siendo 1: “Muy mala” y 5: “Muy buena,
   ¿Cómo califica la cantidad y calidad de la información disponible de los usuarios en el panel de correcciones?
3. ¿Cómo califica el flujo de corrección?
    a. Complejo
    b. Confuso pero aceptable
    c. Claro
4. Puntuando entre 1 y 5, siendo 1: “Lento” y 5: “Muy ágil,
   ¿Cómo calificaría la agilidad de corrección?
5. ¿Qué fue lo que más le gustó del panel de administración y el de correcciones?
6. ¿Tiene usted algún comentario o sugerencia de mejora para la herramienta?
7. ¿Considera que la herramienta es útil en el desarrollo del curso?
8. ¿Cree que facilita ciertas tareas? ¿Cuáles?

Usuario Tester

Este cuestionario busca recabar información respecto a las sensaciones de los usuarios testers al utilizar la aplicación.

1. Marcar una de las siguientes respuestas para cada uno de los aspectos del panel de administración: Muy bueno, Bueno, Regular, Malo, Muy malo. Aspectos a evaluar:
    a. Claridad de la información
    b. Cantidad de la información
    c. Facilidad de uso
    d. Estética
    e. Rapidez de descarga de las páginas
2. Utilizando las mismas respuestas del punto 1, valore los siguientes aspectos relativos a su utilización de la herramienta:
    a. ¿Qué le parecieron las funcionalidades disponibles?
    b. ¿Le resultó de utilidad para aprender?
    c. ¿Qué le parecieron los niveles de dificultad?

3. Marcar una de las siguientes respuestas sobre ciertos aspectos del testing: Muy difícil, Difícil, Normal, Fácil, Muy fácil:
    a. ¿Cómo le resultó la detección de incidentes?
    b. ¿Cómo le resultó el reporte de incidentes?
    c. ¿Cómo le resultó la carga de archivos?
4. Indique qué estrategias y/o técnicas de diseño de casos de prueba aplicó para detectar los incidentes:
    a. Testing exploratorio (TE)
    b. TE basado en sesiones
    c. Clases de equivalencia
    d. Valores límite
    e. Combinación por pares
    f. Tablas de decisión
    g. Árboles de decisión
    h. Máquinas de estados
    i. A partir de casos de uso
    j. Otra
5. ¿Qué fue lo que más le gustó del sitio?
6. ¿Tiene usted algún comentario o sugerencia de mejora para la herramienta?

6.3. Resultados de la encuesta

Para exponer los resultados obtenidos se utilizan los diagramas y gráficos que genera el formulario de Google automáticamente. Estos reflejan en forma clara las respuestas de los usuarios y permiten obtener estadísticas fácilmente.

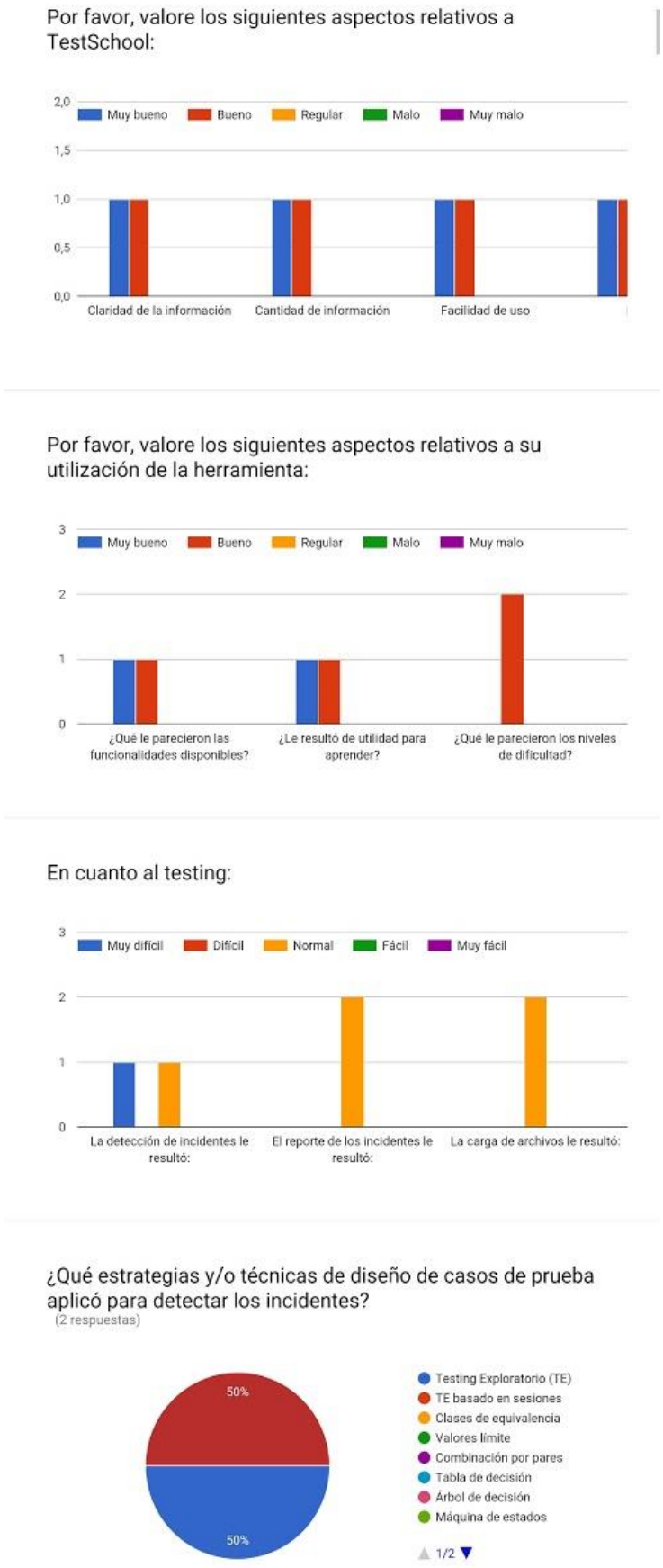


Figura 6.1: Respuestas de la encuesta a los estudiantes.

Figura 6.2: Respuestas de la encuesta a administradores.

En el caso de los estudiantes se obtuvieron resultados positivos. Las dos premisas fundamentales, que el sitio web sea amigable y que los estudiantes consideran que la herramienta es útil para aprendizaje, obtuvieron resultados satisfactorios. El aspecto negativo a tener en cuenta es que un estudiante consideró que los bugs inyectados el nivel inicial no eran fáciles de encontrar. Sin embargo esto no es una crítica sobre Test School, sino más bien, sobre la aplicación mutada.

La respuesta del profesor que completó la encuesta fue por demás satisfactoria. El proceso de corrección obtuvo las mejores calificaciones, lo cual fue muy alentador, dado que, como ya se mencionó, era uno de los objetivos principales de la plataforma.
También se obtuvo una buena calificación en cuanto al proceso de creación de nuevas aplicaciones y niveles para los usuarios.

Nos hubiera gustado contar con más encuestas completadas, pero debido a los tiempos del proyecto no fue posible.

7. Gestión de riesgos

7.1. Objetivo

En esta sección se presenta el plan de gestión de riesgos que consiste en 4 tareas básicas [45]:

- Identificación: se identifican riesgos del proyecto, del producto y del negocio.
- Análisis: para los riesgos identificados se asigna una probabilidad de que éstos ocurran y su impacto en el proyecto.
- Planificación: se desarrollan planes de mitigación para evitar o minimizar los efectos de dichos riesgos.
- Monitoreo: se monitorean los riesgos durante el transcurso del proyecto y se evalúa el estado de éstos para actuar en consecuencia.

7.2. Identificación

Los riesgos identificados se dividen en las siguientes categorías, según el aspecto que afecta:

- riesgos del proyecto: afectan fechas de entrega o recursos:
    - disponibilidad de tiempo de los desarrolladores
    - respecto a las tecnologías
- riesgos del producto: afectan la calidad o rendimiento del software que se está desarrollando:
    - cambios en los requisitos
    - error en estimaciones

7.3. Análisis

En esta etapa se asigna una prioridad a cada riesgo identificado y seriedad de las consecuencias que conlleva si el riesgo se hace efectivo.
La probabilidad puede ser baja, media, alta o muy alta; y las consecuencias pueden ser catastróficas, serias, tolerables o insignificantes.

7.3.1. Riesgos identificados

I. Integración de tecnologías y herramientas

Puede resultar complicada la integración de algunas herramientas con las tecnologías utilizadas.
Probabilidad: Alta.
Impacto: Serio.
Plan de mitigación:

Uno de los principales inconvenientes que se pueden anticipar es la integración de Mantis con la aplicación, por lo tanto se dio prioridad esta tarea ya que es un requisito que difícilmente se pueda cambiar.
Respecto a otras herramientas, excepto el caso de las grabaciones de sesiones (que fueron consideradas un riesgo aparte), se intentará elegir las más adecuadas en base a conocimientos

propios e investigación. Si se pierde demasiado tiempo integrando una herramienta, se considerará buscar una alternativa.

II. No encontrar aplicaciones a mutar que se ajusten a las necesidades

En la versión inicial del sistema se debe incluir un conjunto de escenarios para que pueda ser utilizado de inmediato, y lo cual implica contar con al menos una aplicación sobre la cual se generen mutaciones, que serán evaluadas por los usuarios testers.
La tarea de encontrar una aplicación que cumpla todos los requisitos necesarios no resulta trivial ya que debe ser relativamente sencilla de integrar al sistema, ajustarse a los usuarios finales (estudiantes o testers) y brindar suficientes funcionalidades para evaluarlos.
Probabilidad: Media.
Impacto: Catastrófico.
Plan de mitigación:

Dada la gravedad de las consecuencias si este riesgo se hace efectivo, se consideraron las siguientes medidas preventivas:

- Realizar un listado de posibles aplicaciones priorizado de acuerdo a determinados parámetros.
- Si aún luego de realizar la búsqueda no se encontraran aplicaciones adecuadas se construiría una, aunque ésto sería más costoso.
- Ver sección “Aplicación utilizada” por más detalles sobre el plan de acción de este riesgo.

III. Inserción de mutaciones

Refiere a la posibilidad de inyectar defectos en forma automatizada desde un panel de administración y generar la aplicación mutada correspondiente que contenga dichos errores.
Probabilidad: Alta.
Impacto: Serio.
Plan de mitigación:

Debido a que se trata de una funcionalidad importante de la aplicación, es primordial saber si es viable construirla desde un principio. Se hará un relevamiento de los requisitos para comprender no solo lo que los clientes quieren sino lo que necesitan, y luego plantear una alternativa en caso de que no se logre exactamente lo que quieran y sea necesario cambiar los requisitos.
Si después de investigar se concluye que es posible construir esta funcionalidad en un tiempo razonable, se construirá un prototipo para descartar otras complicaciones que puedan surgir.

IV. Poca experiencia con la tecnología AngularJS

Uno de los motivos para utilizar la tecnología AngularJS para construir la interfaz de la aplicación es aprender dicho lenguaje, por lo cual resulta natural que haya una curva de aprendizaje inicial.
Probabilidad: Muy alta.
Impacto: Tolerable.
Plan de mitigación:

Debido a que este riesgo es imposible de evitar, se intenta minimizar sus efectos sobre los tiempos de implementación y la calidad del código siguiendo las siguientes recomendaciones:

- Seguir documentación disponible.
- Consultar sobre dudas y problemas a personas con experiencia en AngularJS, físicamente o a través de foros.
- Construir un prototipo que consista en un pequeño proyecto con algunas vistas construidas en AngularJS

V. Grabación de sesiones

El requisito inicial implicaba grabar en un video la sesión de un usuario que fuera accesible para los administradores (profesores).
Probabilidad: Media
Impacto: Tolerable.
Plan de mitigación:

Aunque hay desconocimiento respecto a la viabilidad de esta funcionalidad, se le dio prioridad a la mitigación de otros riesgos por tratarse de una menos importante.
Una vez que los otros riesgos de mayor impacto sean mitigados, se investigará éste y se construirá un prototipo si es posible.

VI. Falta de disponibilidad de alguna herramienta o servicio en particular

Puede ocurrir que algunas herramientas no hayan sido actualizadas recientemente o que tengan algún defecto conocido y no sean consistentes.
Probabilidad: Media
Impacto: Tolerable
Plan de mitigación:

Si se trata de alguna herramienta fundamental para la aplicación se va a investigar con mayor prioridad ya que las consecuencias en este caso podrían ser catastróficas. Sin embargo, para el resto de las herramientas o tecnologías de menor importancia, si se detectan problemas se procederá a consultarlo en foros o con otros desarrolladores. Si no se llega a una solución se buscarán herramientas alternativas de las cuales se tenga más información o se hayan usado previamente.

VII. Disponibilidad de los desarrolladores

Alguno o varios integrantes del grupo no cuentan con la disponibilidad horaria adecuada para el proyecto debido a actividades circunstanciales (entregas de facultad, exámenes, etc).
Probabilidad: Alta
Impacto: Tolerable
Plan de mitigación:

Se fijarán días de reunión con anticipación para que el equipo aproveche esa instancia para actualizar el estado de las tareas realizadas y planifique nuevas.
También se hará un fuerte hincapié en mantener una comunicación fluida entre los integrantes del equipo y una clara distribución de tareas para que el tiempo de trabajo sea lo más efectivo posible.

VIII. Requisitos

Cambios en algunos requisitos o aparición de nuevos que requieren rediseño o implican cambios estructurales grandes pueden significar retrasos en el desarrollo, cuya gravedad depende del impacto de dichos requisitos en la aplicación.

Probabilidad: Alta

Impacto: Serio

Plan de mitigación:

Para mitigar la aparición de cambios frecuentes es importante relevar los requisitos iniciales de forma profunda para que surjan varias preguntas y consecuentemente casos de uso que pudieron haber sido pasados por alto.

Si surgen inquietudes para agregar alguna funcionalidad, se evalúa en primer lugar la razón detrás del cambio, analizando las necesidades del cliente para solicitarlo y discutiendo, si es necesario, la urgencia de dicho requisito.

Si se concluye que el nuevo requisito o la modificación pedida pueden ser sustituidos por otra solución alternativa, se implementará esta última.

Si se acuerda agregar el nuevo requisito a la lista de tareas pendientes del proyecto, ésta debe ser estimada para conocer el impacto que ésto tendrá en el alcance.

IX. Estimaciones

El tiempo de implementación de algunas funcionalidades o la integración de algunas herramientas puede ser subestimado/a.

Probabilidad: Media

Impacto: Serio

Plan de mitigación:

Para poder re-estimar en forma más acertada se investigarán principalmente las tareas cuya estimación sea más insegura, que implican un mayor riesgo y que son de mayor importancia para la aplicación.

7.3.2. Monitoreo

Examinar cada riesgo identificado regularmente para decidir si se ha vuelto más o menos riesgoso y también si sus efectos han cambiado.

En primer lugar se atacaron los riesgos de más alto impacto, entre los cuales se encuentran:

I. Integración de tecnologías y herramientas

La principal herramienta que generaba preocupación era MantisBT ya que era necesario evaluar cómo iba a funcionar la comunicación entre esta herramienta y la aplicación, y no existían muchos ejemplos de integración con MantisBT de referencia.

Los principales problemas observados fueron:

- A pesar de contar con una API, ésta no es actualizada frecuentemente y no se ajusta adecuadamente al framework utilizado en nuestra aplicación (Ruby on Rails).

- El lenguaje de la aplicación es PHP el cual no es dominado por los integrantes del equipo. Esto dificultó evaluar la API de MantisBT y analizar qué tan complejo sería realizar modificaciones en ésta, o en algún archivo de Mantis, para adaptarlos a nuestro sistema.
- El código de MantisBT maneja conceptos/modelos/estructuras obsoletos que no son comúnmente usados en la actualidad y se consideran poco usables a nivel de desarrollo.
- Era necesario evaluar si era viable utilizar la API de MantisBT o se manejarían bases de datos independientes que compartan el método de encriptación entre los usuarios del sistema y MantisBT.

El plan de acción implicó comenzar de inmediato con el estudio de esta herramienta, lo cual es explicado en mayor detalle en el documento de arquitectura.
En primer lugar se estudió la documentación de la API y se intentó integrar algunos de los servicios que brinda en nuestro sistema, pero se concluyó que era muy trabajoso adaptar esos servicios a las necesidades de nuestra aplicación.
Se decidió trabajar con una base de datos independiente para MantisBT y compartir la información de registro de los usuarios con la base de nuestra aplicación. También fue necesario modificar y configurar algunos archivos de MantisBT para que ambas aplicaciones se pudieran comunicar correctamente.
Esta tarea requirió más tiempo del esperado pero debía ser analizada y mitigada lo antes posible por la importancia de este requisito en el proyecto.
Una vez finalizada la comunicación, se consideró que el riesgo estaba mitigado y se había logrado integrar la aplicación con MantisBT.

II. No encontrar aplicaciones a mutar que se ajusten a las necesidades

Debido a la gravedad de las consecuencias de este riesgo se investigó en paralelo con la integración de MantisBT, ya que era previsible que consumiera mucho tiempo de estudio, Además, era necesario saber cuánto antes si sería necesario implementar una aplicación nueva en caso de no encontrar ninguna que se adecuara.
En primer lugar se buscaron listas de aplicaciones libres y abiertas; luego se categorizaron y ponderaron en base a ciertos criterios hasta llegar a una aplicación final que se adecuaba a las necesidades del sistema.
Una vez encontrada la aplicación se consideró mitigado este riesgo ya que se evaluó nuevamente y no se identificaron problemas potenciales, tomando en cuenta que:

- Como parte del proceso de elección se había descargado e instalado la aplicación, por lo tanto esto no debería representar una complicación
- La aplicación es levantada en un iframe, entonces no implica una integración con el sistema; solo se debe correr en un servidor independiente al momento de ejecutar Test School.

Por más detalles sobre el plan de acción, ver sección “Aplicación elegida”.

III. Inserción de mutaciones

Este también era un riesgo de alto impacto y por ese motivo se comenzó a estudiar en etapas tempranas del proyecto, ya que de no ser viable sería necesario modificar el requisito y analizar otra manera de integrar versiones mutadas de una aplicación en el sistema.
Originalmente se pretendía que las versiones mutadas de una aplicación se generaran manualmente, indicando desde el panel de administración determinados defectos.

Luego, estas mutaciones se crearían automáticamente con los defectos inyectados y se asignarían a un determinado nivel.
Sin embargo, al analizar esta funcionalidad surgieron las siguientes preocupaciones:

- Permitir a los administradores seleccionar un tipo de defecto implica automatizar la modificación de algunas secciones del código fuente.
- Para dejar la aplicación en un estado consistente, todas las combinaciones de modificaciones deben funcionar correctamente y eso implica mucho tiempo de implementación y testing. De lo contrario, la aplicación podría tener un comportamiento inesperado o colapsar.
- Considerando que puede haber varias versiones para el mismo nivel (en base a la dificultad asociada a los defectos), es importante que los tipos de defectos inyectados sean consistentes para mutaciones del mismo nivel.

Finalmente se optó por generar previamente las versiones mutadas de una aplicación, donde los defectos inyectados fueron analizados según las técnicas que se deberían aplicar para detectarlos y una dificultad asociada al nivel al que pertenece la mutación.
De esta forma las modificaciones son controladas y se evitan varios problemas de inconsistencias.

IV. Poca experiencia con la tecnología AngularJS

Luego de construir un prototipo en AngularJS se ganó confianza en dicho lenguaje y a medida que se avanzó en la construcción del sistema la probabilidad de este riesgo bajó gradualmente. Sin embargo, el riesgo permaneció latente a lo largo de todo el proyecto ya que frecuentemente surgieron inconvenientes en la implementación de la interfaz y en la integración entre AngularJS y Rails.
En caso de no poder solucionar los problemas discutiendo entre los integrantes del equipo, se acudió a foros de ayuda o se consultó a otros desarrolladores con experiencia en el lenguaje.

V. Grabación de sesiones

La evaluación de la viabilidad de esta funcionalidad se explica en mayor detalle en la sección “Requisitos que cambiaron”.

VI. Falta de disponibilidad de alguna herramienta o servicio en particular

Las herramientas principales necesarias para desarrollar el sistema estaban disponibles por lo que no hubo necesidad de buscar alternativas.
Para evitar posibles inconvenientes de disponibilidad se eligieron herramientas estables y confiables, por lo cual no hubieron problemas de este tipo durante el desarrollo.

VII. Disponibilidad de los desarrolladores

Durante varios meses se mantuvo la constancia de tener reuniones de actualización entre los integrantes del equipo al menos una vez a la semana, lo cual permitió a todos ellos mantenerse al tanto del estado del proyecto y los avances de cada uno. En caso de no ser posible una reunión semanal, la comunicación se mantuvo a través de otros medios (correo electrónico o chat).

VIII. Requisitos

Algunos requisitos fueron modificados luego de ser analizados y evaluar la complejidad de su implementación. En esos casos se discutieron alternativas con el cliente y se decidió cambiar la especificación de las correspondientes funcionalidades.

Los principales requisitos que cambiaron son explicados en mayor detalle en la sección “Requisitos que cambiaron”.

IX. Estimaciones

Se estimaron inicialmente las funcionalidades que había que construir y se realizó un cronograma tentativo inicial en base a ellas. Sin embargo, algunas funcionalidades tenían un componente de riesgo importante, por lo cual su estimación era muy insegura ya que primero era necesario investigar su viabilidad y complejidad.

Las principales desviaciones que se identificaron y afectaron las estimaciones fueron:

- Complicaciones para integrar con API de Mantis: este aspecto fue explicado en esta misma sección, en el punto 1.
- Cambios en requisitos: esto fue explicado en el punto anterior.

Elegir una aplicación para mutar en el sistema: no resultó sencillo encontrar una aplicación que se ajustara a las necesidades, por lo cual hubo que estudiar y evaluar varias posibilidades, como fue explicado en el punto 2 de esta sección, lo cual implicó un tiempo considerable.

8. Conclusiones

Para una mejor comprensión se dividirán las conclusiones bajo distintos tópicos.

Necesidad del producto y motivación de los integrantes

En los últimos años, el área de e-learning o e-schooling ha tenido un crecimiento importante siendo una de las herramientas fundamentales en diferentes áreas del aprendizaje. Esto se asocia a la facilidad que le brinda a un estudiante poder trabajar desde su lugar de confort. A esto se le suma el aspecto motivacional que brindan este tipo de herramientas, como lo es un sistema de jerarquía de niveles.

El CES imparte la Carrera de Testing en línea, así como varios cursos o paquetes independientes, pero no cuenta con ninguna herramienta parecida, llevando a cabo las diferentes tareas y evaluaciones de manera manual. Por esto, agilizar la evaluación era la mayor urgencia para los organizadores y docentes del CES.

Por lo tanto, desarrollar una herramienta de e-learning que cumpliera con esta necesidad, se consideró altamente necesario y oportuno, además de haber sido una tarea motivante para los integrantes del proyecto por su utilidad a futuro.

Utilidad y calidad del producto

Para lograr construir un producto con estas cualidades fue necesario comprender las estrategias y técnicas dictadas en los cursos del CES, las realidades a las que se aplican y los tipos de defectos que detectan.

Fue de gran ayuda haber cursado la materia Taller de Verificación de Software para entender rápidamente el área de aplicación de Test School.

Tecnologías utilizadas

Los integrantes del proyecto tenían total libertad para elegir los lenguajes de programación con los cuales se desarrollaría el producto. Se hizo un balance entre cuales eran los lenguajes adecuados y el conocimiento de los integrantes de cada uno de ellos.

La elección de Ruby on Rails para desarrollar el lado del servidor es considerada altamente positiva. Dos de los integrantes tenían más de dos años de experiencia, logrando desarrollar una API Ruby on Rails de buena calidad en un tiempo razonable. Lo mismo sucedió con el panel de administración.

La elección de AngularJS también es considerada positiva, principalmente teniendo en cuenta la calidad del producto generado. El punto negativo de esta elección fue que el tiempo de desarrollo fue bastante más lento en comparación con el servidor. Haciendo un balance, se considera que fue una decisión acertada.

Agilización del mecanismo de evaluación

El principal propósito del proyecto era agilizar el mecanismo de evaluación por parte de los profesores del CES.

Los integrantes del equipo buscaron alternativas en conjunto con los profesores durante el transcurso de todo el proyecto para cumplir con dicho objetivo.

Se lograron mecanismos ágiles de corrección, siendo una tarea sencilla relacionar defectos reportados por estudiantes con defectos conocidos por los profesores. Luego de esto, a partir de una

nota sugerida por el sistema, el profesor puede calificar y entregar una devolución del trabajo al estudiante.
Los integrantes quedamos conformes con la solución implementada, a pesar de que se entiende que existen mejoras que se podrían desarrollar a futuro.
Aún más importante, los profesores del CES encontraron la sección de evaluaciones muy intuitiva y fácil de utilizar, destacando la facilidad de su trabajo. Esto se concluye a partir de los cuestionarios completados por ellos.

Integración con MantisBT

La herramienta MantisBT es muy utilizada en los cursos del CES, por lo cual, integrarla con nuestro proyecto era otro de los desafíos importantes.
El mismo es un proyecto de código abierto desarrollado en el lenguaje PHP, no teniendo ninguno de los integrantes del equipo conocimiento de este lenguaje. Por esa razón, preferimos evitar modificar el código de MantisBT y vincular ambos a través de la base de datos.

Este enfoque nos permitió vincular de manera sencilla un usuario Test School con un usuario en MantisBT e indirectamente obligar al mismo a tener que familiarizarse con las funcionalidades de esta herramienta. A su vez, resultó sencillo presentar los defectos reportados por los testers a los docentes en el panel de administración.
Como aspecto negativo, este requisito no nos permitió gestionar el almacenamiento de defectos a gusto, teniendo que adaptarnos a como lo hacía MantisBT. Considerar que MantisBT es un proyecto con varios años en el mercado con una interfaz de usuario poco amigable y una estructura de base de datos ya poco utilizada.

Aceptación de los estudiantes

La motivación y facilidad de uso de la herramienta era uno de los principales objetivos del proyecto.
Se desarrolló una interfaz amigable y adaptable a cualquier dispositivo. Para esto se contó con la ayuda de una diseñadora gráfica con experiencia en usabilidad de sitios web.
Tantos los integrantes como aquellos estudiantes que probaron la aplicación quedaron conformes. Destacando estos últimos, su interfaz intuitiva y amigable, además de la aceptación que tendría en los cursos del CES.

Generación automática de mutantes

Se generaron mutaciones manualmente, a pesar de que en un comienzo se manejaron opciones de generación automática de mutantes mediante otra aplicación como Bacterio.
Esta fue una tarea engorrosa que nos permitió generar mutaciones adaptadas a las necesidades del curso ya que, nos focalizamos en que los mutantes tengan defectos similares a los que se generan en el proceso de desarrollo de software.
Como conclusión, creemos que contar con una herramienta que genere mutaciones automáticas con defectos de gran calidad resulta muy difícil de construir, pero que, sería un complemento que aumentaría el valor de esta aplicación. Esto podría ser considerado para un futuro proyecto de grado.

Comercialización futura

Una conclusión interesante es que el producto puede ser utilizado en un futuro por institutos con cursos similares a los dictados por el CES. Resultaría interesante realizar un estudio de mercado para encontrar potenciales clientes y de esta manera extender el alcance y funcionalidades de la aplicación. Como queda expuesto en el anexo de pruebas realizadas, la aplicación tiene la capacidad de soportar muchos más usuarios de los que la utilizarán en una primera instancia. Además de la ya mencionada posibilidad de escalar vertical y horizontalmente por su arquitectura.

9. Referencias bibliográficas

[1] Whittaker, J. (2002), How to break Software, a practical Guide, Estados Unidos: Addison Wesley.

[2] ISO/IEC 25000, System and Software Quality Requirements and Evaluation (SQuaRE). 2015. Accedido por última vez Marzo 2016

[3] Kaner C., Bach J., Pretichord B., (2011), Lessons Learned in Software Testing, Nueva York, Estados Unidos: Wiley.

[4] (2004), *Guide to the Software Engineering Body of Knowledge SWEBOK, 2004 version. IEEE Computer Society*

[5] Mark Utting, Bruno Legeard, (2006), Practical Model-Based Testing: A Tools Approach, San Francisco, Estados Unidos: Morgan Kaufmann Publishers Inc.

[6] Glenford J. Myers, (1979), *The Art of Software Testing,* New Jersey, Estados Unidos: John Wiley & Sons.

[7] Estándar IEEE 610.12-1990, (1990), IEEE Standard Glossary of Software Engineering Terminology Institute of Electrical and Electronics Engineers. Accedido por última vez en Agosto 2016.

[8] James Becker, Glenn Flick, (1997), *A practical approach to failure mode,effects and criticality,* Washington, Estados Unidos

[9] Kaner C., (2003), *What Is a Good Test Case?*, Florida, Estados Unidos: STAR East

[10] Ron Patton, (2000*), Software Testing*, Sams Pub.

[11] Boris Beizer. (1995), *Black-box testing: techniques for functional testing of software and systems,* John Wiley & Sons

[12] Robert V. Binder., (1999), *Testing Object-oriented Systems Models, Patterns, and Tools*. Estados Unidos, Addison-Wesley.

[13] Cem Kaner, (2004), *A Course in Black Box Software Testing - Examples of Test Oracles, Center for Software Testing Education & Research*. Publicado en 2004. Accedido por última vez en Agosto 2016.

[14] Beizer, (1990), *Software Testing Techniques, 2nd Edition*

[15] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, (1991), *Fundamentals of software engineering*. New Jersey, Estados Unidos

[16] Kaner C., Falk J., Nguyen H. (1999), *Testing Computer Software, 2nd Edition*, Estados Unidos: Wiley.

[17] Kaner, (2008), *A Tutorial in Exploratory Testing*, Florida, Estados Unidos: QUEST 2008

[18] James Bach, (2002), *The Test Practicioner*

[19] Black R. , (2002), *Managing the Testing Process, 2nd Edition* Estados Unidos: Wiley.

[20] James Bach, *Sample Session Report,* Satisfice Inc. Accedida por última vez Agosto de 2016

[21] Notas del curso “Taller de Verificación de Software”, (2014), *Testing Funcional*, Montevideo, Uruguay: Facultad de Ingeniería, UDELAR.

[22] Notas del curso “Taller de Verificación de Software”, (2014), *Combinación por Pares*, Montevideo, Uruguay: Facultad de Ingeniería, UDELAR.

[23] Notas del curso “Taller de Verificación de Software”, (2014), *Máquinas de Estado*, Montevideo, Uruguay: Facultad de Ingeniería, UDELAR.

[24] *International Software Testing Qualifications Board, Certified Tester Foundation Level Syllabus*. Publicado en 2005. Accedido por última vez en Agosto 2016.

[25] Jacobson I., Booch G., Rumbaugh J., (1999), The Unified Software Development Process, Estados Unidos: Addison Wesley.

[26] Jean-Marie Mottu, Benoit Baudry, Yves Le Traon, (2006), *Mutation Analysis Testing for Model Transformations*, Rennes, Francia: Springer-Verlag Berlin Heidelberg

[27] Universidad de Castilla-La Mancha, (2009), *BACTERIO MUTATION TEST SYSTEM*, Ciudad Real, España

[28] J. Carreira, H. Madeira, J.G. Silva, (1998), Xception: A technique for Experimental Evaluation of Dependability in Modern Computers, Coimbra, Portugal: IEEE Transactions On Software Engineering, Vol 24, pp 125-135, Feb 1998.

[29] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson, (2001), GOOFI: Generic Object-Oriented Fault Injection Tool, Göteborg, Sweden: Procs International Conference on Dependable Systems and Networks.

[30] J. Arlat, Y. Crouzet and JC. Laprie, (1989), Fault Injection for dependability validation of fault-tolerant computing systems, Toulouse, Francia: Laboratoire d’Automatique et d’Analyse des Systèmes du C.N.R.S.

[31] Antonio da Silva, José F. Martínez, Lourdes López, Luis Redondo, (2007), EXHAUSTIF®: UNA HERRAMIENTA DE INYECCIÓN DE FALLOS POR SOFTWARE PARA SISTEMAS EMPOTRADOS DISTRIBUIDOS HETEROGÉNEOS, España: Revista de procesos y métricas de las tecnologías de la información, ISSN 1886-4554, Vol. 4, Nº. 2 (ABR), 2007, págs. 51-64.

[32] Sitio oficial de CodeSchool, CodeSchool. Accedido por última vez en Agosto de 2016.

[33] Sitio oficial de Moodle, Moodle. Accedido por última vez en Agosto de 2016.

[34] Black Box Puzzles, WorkRoom Productions. Publicado en 2014. Accedido por última vez en Octubre, 2015.

[35] 2048, 2048 Game. Accedido por última vez en Octubre, 2015.

[36] Sitio oficial de Spree, Spree. Accedido por última vez en Agosto de 2016.

[37] Dorian Peters, UX for Learning: Design Guidelines for the Learner Experience, UX Matters. Publicado en Julio 2012. Accedido por última vez en Agosto de 2016.

[38] Documentación de gema Rubocop, RubyDoc. Publicado en 2012. Accedido por última vez en Agosto de 2016.

[39] Buenas Prácticas para Ruby on Rails, RubyDoc. Publicado en 2011. Accedido por última vez en Agosto de 2016.

[40] Sitio oficial de gema RSpec, RSpec. Accedido por última vez en Agosto de 2016.

[41] Documentación de gema Capybara. Accedido por última vez en Agosto de 2016.

[42] Javier Cuevas, *Testeando el rendimiento de tu aplicación con Apache Bench*, Diacode. Accedido por última vez en Agosto de 2016.

[43] Sitio oficial de la gema Puma, Puma. Accedido por última vez en Agosto de 2016.

[44] Dmitri Lau, *10 Reasons Why You Should Use AngularJS*, Sitepoint. Publicado en Setiembre 2013. Accedido por última vez en Agosto 2016.

[45] Ian Sommerville, (1995), *Software engineering (5th ed.),* Redwood City, Estados Unidos: Addison Wesley Longman

[46] Cynthia Harvey, 92 Open Source Apps That Replace Everyday Software, Datamation. Publicado en Febrero 2012. Accedido por última vez en Abril 2016.

[47] *The Big List of Open Source Web Applications*, SoftwareSoapbox

[48] Globe Testing, Reporte de incidencias con Mantis, Globe Testing. Accedido por última vez en Julio 2016.

[49] Sitio oficial de MantisBT, MantisBT. Accedido por última vez en Enero 2016,

[50] Modelo de dominio proyecto MantisBT, MantisBT. Accedido por última vez en Agosto de 2016.

[51] James Bach, (2000), *Session Based Test Management*

[52] Notas del curso “Taller de Verificación de Software”, (2014), *Tablas y Árboles de decisión*, Montevideo, Uruguay: Facultad de Ingeniería, UDELAR.

[53] Notas del curso “Taller de Verificación de Software”, (2014), *Testing Exploratorio*, Montevideo, Uruguay: Facultad de Ingeniería, UDELAR.

Anexos

Anexo I: Categorización de incidentes

Existen varias formas de clasificar los defectos, algunas de estas independientemente del tipo de sistema que se esté evaluando. Cada una de estas categorizaciones toma en cuenta aspectos diferentes de los defectos.
En el caso particular del sistema Test School se replicaron las categorías que el CES maneja para el resto de los cursos y las cuales ingresan en Mantis.

Categorías disponibles en el sistema:

- Errores de funcionalidad: éstos ocurren cuando las funcionalidades no se comportan como deberían, de acuerdo a la especificación de los requisitos. Las categorías agrupadas aquí son:
    - Funcionalidad no disponible
    - Implementación incorrecta
    - Salida anormal
- Manejo de errores: los errores que ocurren cuando el usuario interactúa con el sistema deben ser manejados de una manera clara y significativa, de lo contrario se los considera un defecto. Las categorías agrupadas aquí son:
    - Mensajes de error: el mensaje mostrado al usuario no resulta un claro indicador de cuál fue el problema, o directamente no se mostró un mensaje.
    - Manejo de errores: un error debería ser capturado pero no lo es. Puede ocurrir que el sistema siga funcionando normalmente, pero no se notifica al usuario que la acción no se concretó porque hubo algún inconveniente.
    - Validación faltante / incorrecta: ocurre cuando se intenta enviar cierta información que debe ser validada, pero esta acción no se realiza o se hace incorrectamente.
- Errores de control de flujo: refieren al comportamiento inapropiado o inesperado de ciertos pasos en el flujo de un caso de uso. La categoría implicada aquí es “Navegabilidad”.
- Errores de interfaz de usuario: refieren al aspecto de la interfaz y la correcta implementación del diseño de la aplicación y consecuentemente la interacción entre el usuario y el sistema. Las categorías agrupadas aquí son:
    - Interfaz de usuario: abarca pasaje de parámetros incorrecto, alineaciones o ubicaciones de objetos o campos incorrectas.
    - Usabilidad: interfaz poco clara, ambigüedad en elementos del diseño, mala elección de los textos utilizados y posicionamiento poco amigable dentro de las ventanas o páginas.
- Errores de hardware: relacionado a la no disponibilidad de recursos como memoria, que pueden generar problemas o comportamiento inesperado del sistema, como páginas caídas o que no se cargan correctamente. También involucra problemas de performance que el usuario detecte. La categoría asociada a este tipo de errores es “Instalación o ambiente”
- Errores de seguridad: refieren a excepciones o infiltraciones en el sistema que no deberían ser permitidas ya que atentan contra el correcto funcionamiento de la aplicación
- Otros: en esta categoría entran aquellos defectos que no generan fallas necesariamente en el sistema, sino que son detalles subjetivos que el usuario observa y cree que se podrían mejorar. Las categorías agrupadas aquí son:

- Mejora
- No clasificado
- Observación.

Anexo II: Registro de pruebas

Introducción

En este documento se presentan las estrategias y objetivos de las pruebas, como fueron implementadas y un breve resumen de los resultados obtenidos.

Estrategia

Se implementaron pruebas funcionales para probar los requisitos y pruebas no funcionales para probar el rendimiento y carga soportada por el sistema.
Luego de completada esta etapa, profesores y estudiantes del CES llevaron a cabo pruebas de aceptación.

Las pruebas funcionales son del tipo pruebas unitarias y pruebas de integración.
Las primeras fueron implementadas y ejecutadas sobre la API Ruby on Rails, al igual que las segundas, con la salvedad que estas últimas actúan además sobre el cliente AngularJS.

No se realiza ningún tipo de pruebas sobre el componente MantisBT por ser un proyecto implementado, configurado y no desarrollado por los integrantes del grupo. MantisBT es una herramienta de confianza y trayectoria para el equipo del CES.

Las pruebas no funcionales son pruebas de performance y carga. No existía ningún premisa en cuanto a mínimo criterio de aceptación, pero se entendió necesario tener un entendimiento de cuál es la performance y carga soportada por el sistema.

Pruebas unitarias

Las pruebas unitarias se implementaron para la API Ruby on Rails.
Para el cliente AngularJS no se implementaron pruebas unitarias, ya que se entiende que con las pruebas de integración se logra un cubrimiento aceptable para este componente.

API Ruby on Rails

Los componentes sobre los cuales se implementaron las pruebas son los modelos y controladores.

Los modelos se encargan de la lógica de negocios de un proyecto Ruby on Rails (por más información ver capítulo 4.6.1-Modelos en el informe). Por esta razón es el componente en el que se implementan la mayoría de las pruebas unitarias.

Se implementaron pruebas en los controladores que se encargan de los servicios principales que expone la API. Los métodos de los modelos son simulados ya que se asume que funcionan como se espera porque se cuenta con pruebas unitarias previas para los modelos.

La herramienta utilizada es la gema RSpec que es fácilmente configurable en un proyecto Ruby on Rails y permite implementar pruebas unitarias claras y legibles para cada uno de los módulos de código, en este caso, modelos y controladores.

A continuación se presenta, a modo de ejemplo, parte de la salida generada al ejecutar los test unitarios sobre la API Ruby on Rails
Se presenta el resultado de las pruebas para algunos de los métodos del modelo Level bajo diferentes escenarios.

```
Level
  .initial_levels
    there are two initial levels
      there are three children levels
        should contain exactly two initial levels
  #new?
    there is one parent level with two children, the second and third level
      should be falsey
      parent level has been already approved
        the second level has not been started yet
          should be truthy
        the third level has not been started yet
          should be truthy
        the second level has been started
          should be falsey
  #allowed?
    there is only one level
      should be truthy
    there is a second level, it is a child level
      user has not approved the first parent level yet
        should be falsey
      user has apprvoed the first parent level
        should be truthy
  #finished_session?
    there is only one level
      there is an active session for this level
        should be falsey
      there is a finished session for this level
        should be truthy
      there is a finished session but it was already corrected
        should be falsey
    there are two levels
      when user already pass the first level
        when user has not started the second level
          should be falsey
  #approved?
    there is not an approved session for the current level
      should be falsey
    there is an approved session for the current level
      should be truthy
```

Figura A2.1: Test unitarios de modelo.

Otras herramientas utilizadas fueron:

- Gema Faker, utilizada para generar datos aleatorios de datos como direcciones, teléfonos, nombres, etc
- Gema FactoryGirl que permite crear prototipos para cada uno de los modelos y pedir instancias con propiedades importantes para la prueba en cuestión.

La siguiente imagen presenta la definición de una fábrica del modelo MutatedApp. También se observa el uso de Faker para generar una URL aleatoria.

```ruby
FactoryGirl.define do
  factory :mutated_app do
    level
    sequence(:name) { |n| "mutacion_#{n}" }
    server_url      { Faker::Internet.url }
  end
end
```

Figura A2.2: Definición de una fábrica.

Se realizaron cerca de 100 test unitarios de los métodos más importantes de los modelos y controladores.

```
Finished in 32.65 seconds (files took 5.08 seconds to load)
91 examples, 0 failures
```

Figura A2.3: Número de tests y tiempo de ejecución.

La imagen siguiente muestra los test realizados para los métodos de instancia approved? y approved_session y parte de las pruebas del método de clase initial_levels. Como se observa, cada uno de los métodos es probado bajo diferentes escenarios.

```ruby
describe Level do
  let(:stage) { FactoryGirl.create(:stage) }
  let(:user)  { FactoryGirl.create(:user) }
  let(:level) { FactoryGirl.create(:level, stage: stage) }

  describe '#approved?' do
    context 'there is not an approved session for the current level' do
      it { expect(level.approved?(user)).to be_falsey }
    end

    context 'there is an approved session for the current level' do
      let!(:approved_session) { FactoryGirl.create(:approved_session, level: level, user:  user) }
      it { expect(level.approved?(user)).to be_truthy }
    end
  end

  describe '#approved_session' do
    context 'there is not an approved session for the current level' do
      it { expect(level.approved_session(user)).to be_nil }
    end

    context 'there is an approved session for the current level' do
      let!(:approved_session) { FactoryGirl.create(:approved_session, level: level, user:  user) }
      it { expect(level.approved_session(user)).to eq(approved_session) }
    end
  end

  describe '.initial_levels' do
    context 'there are two initial levels' do
      let(:initial_levels) { FactoryGirl.create_list(:level, 2, stage: stage) }

      context 'there are three children levels' do
        let!(:children_levels) do
          children_levels = FactoryGirl.build_list(:level, 3, stage: stage)
          children_levels.each do |child_level|
            child_level.parent_levels << initial_levels
            child_level.save
          end
        end
```

Figura A2.4: Tests de controladores.

Para lograr un criterio unificado y estándares de aceptación se utilizaron como referencia.

Pruebas de integración

Las pruebas de integración se implementaron para testear el correcto funcionamiento en conjunto de los componentes API Ruby on Rails y AngularJS.

Se implementaron pruebas para las funcionalidades principales del estudiante: registro, inicio de sesión, visualización del dashboard, visualización de un nivel y comienzo de una sesión. De esta manera nos aseguramos que las funcionalidades principales funcionan correctamente.

Las herramientas utilizadas fueron RSpec, Capybara y Selenium webdriver que en conjunto nos permiten levantar un navegador y automatizar las acciones del estudiante. Por ejemplo, llenar un formulario o clickear en un botón o link.

Capybara es una gema Ruby on Rails, por lo cual para simular acciones sobre el cliente AngularJS fue necesario acoplar dicho proyecto dentro del proyecto Ruby on Rails, y utilizar el servidor Ruby on Rails para levantar el de AngularJS. Para evitar confusiones, esto solo se realiza bajo el ambiente de test, en producción los proyectos se encuentran separados.

No fue necesario implementar pruebas de integración para el panel de administración porque fue desarrollado utilizando la gema ActiveAdmin que ya cuenta con un alto grado de confiabilidad por parte de la comunidad Ruby on Rails.

```ruby
describe 'POST #sign_in' do
  let(:user) { FactoryGirl.create(:user, password: 'password') }

  context 'when wrong password is provided' do
    it do
      visit '#/sign_in/'

      fill_in 'email', with: user.email
      fill_in 'password', with: 'wrong_password'
      click_button 'Ingresar'

      expect(page).to have_content 'Identidad o contraseña no válida.'
    end
  end

  context 'when a valid password is provided' do
    it do
      visit '#/sign_in/'

      fill_in 'email', with: user.email
      fill_in 'password', with: 'password'
      click_button 'Ingresar'

      expect(response).to redirect_to '#/stages/'
    end
  end
```

Figura A2.5: Tests de integración sobre la pantalla de inicio de sesión.

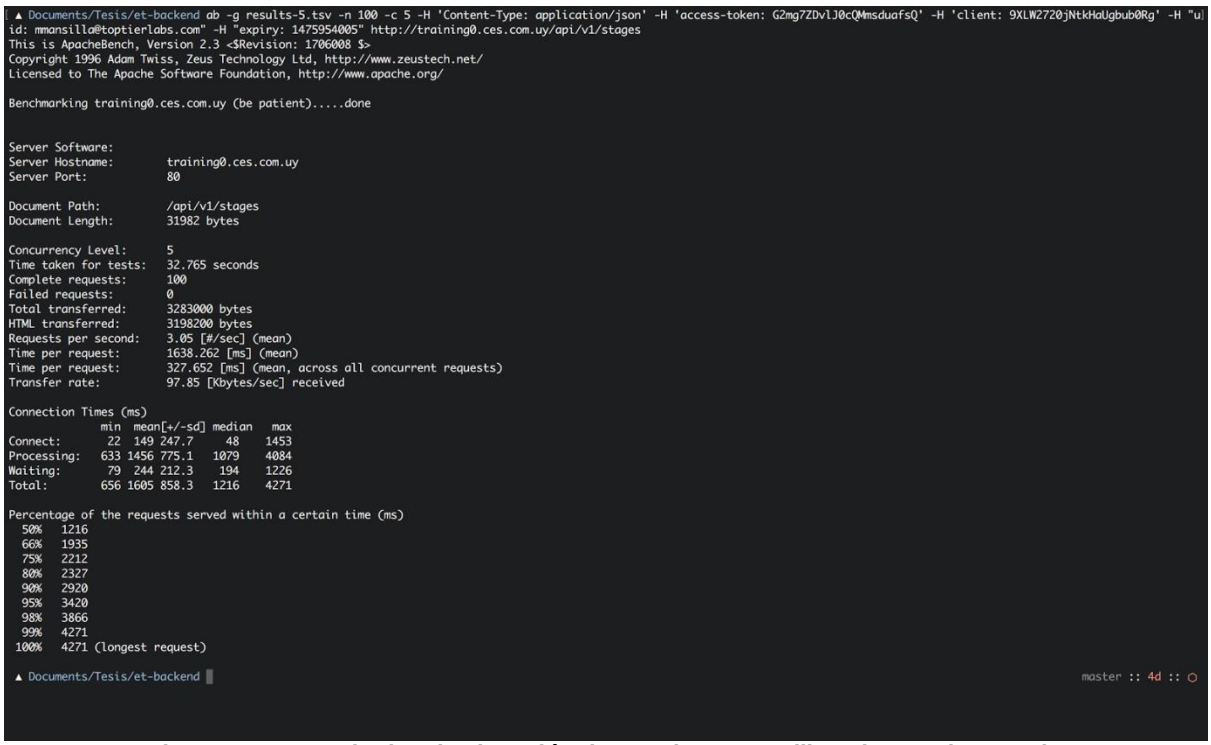
Pruebas de carga y performance

Se implementaron pruebas de carga y performance para evaluar la capacidad o robustez del sistema. Para obtener un escenario más realista, las pruebas se ejecutaron sobre el ambiente de producción.

Las métricas se discutieron con los profesores sobre el final del proyecto dado que no eran un requisito planteado al comienzo del mismo.
Por performance se entiende que debe cumplir con un tiempo de respuesta razonablemente aceptable para cada petición del estudiante o profesor, no existiendo ningún número específico (alrededor de medio segundo).
En cuanto a carga, se tuvo en cuenta que el sitio será utilizado para cursos del CES de 20 a 30 estudiantes, no existiendo más de 2 a 3 cursos en paralelo. Por tanto, la carga no suponía un desafío en el proyecto.
No se realizaron métricas de carga para el panel de administración ya que el número de administradores del sistema es bajo (alrededor de 5 personas).

Se utilizó Apache Bench, herramienta que permite indicar el número de peticiones que se quieren ejecutar, cuántas de ellas se harán en paralelo y cuál es la URL que recibirá el pedido.

A continuación se presenta la salida que resulta de ejecutar 100 peticiones, ejecutando de a 5 al mismo tiempo contra el servicio de la API que devuelve todos los datos necesarios para cargar el dashboard de un usuario “http://training0.ces.com.uy/api/v1/stages”.
Se utilizó este servicio como ejemplo porque entendemos que será uno de los servicios más requeridos.



```
▲ Documents/Tesis/et-backend ab -g results-5.tsv -n 100 -c 5 -H 'Content-Type: application/json' -H 'access-token: G2mg7ZDvlJ0cQMmsduafsQ' -H 'client: 9XLW2720jNtkHaUgbub0Rg' -H "u
id: mmansilla@toptierlabs.com" -H "expiry: 1475954005" http://training0.ces.com.uy/api/v1/stages
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking training0.ces.com.uy (be patient).....done


Server Software:
Server Hostname:        training0.ces.com.uy
Server Port:            80

Document Path:          /api/v1/stages
Document Length:        31982 bytes

Concurrency Level:      5
Time taken for tests:   32.765 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      3283000 bytes
HTML transferred:       3198200 bytes
Requests per second:    3.05 [#/sec] (mean)
Time per request:       1638.262 [ms] (mean)
Time per request:       327.652 [ms] (mean, across all concurrent requests)
Transfer rate:          97.85 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       22  149 247.7     48    1453
Processing:   633 1456 775.1   1079    4084
Waiting:       79  244 212.3    194    1226
Total:        656 1605 858.3   1216    4271

Percentage of the requests served within a certain time (ms)
  50%   1216
  66%   1935
  75%   2212
  80%   2327
  90%   2920
  95%   3420
  98%   3866
  99%   4271
 100%   4271 (longest request)
▲ Documents/Tesis/et-backend                                                    master :: 4d ::
```

Figura A2.6: Resultados de ejecución de test de carga utilizando Apache Bench.

Como se observa es necesario autenticación para realizar este pedido, por lo cual se envían datos de autenticación en los cabezales de la petición.

Vale destacar que aunque se ejecuten 5 peticiones concurrentes, el tiempo promedio de cada petición es de 327ms, tiempo por demás aceptable considerando la cantidad de estudiantes que utilizaran el producto.

Pruebas de aceptación

Una vez completadas las pruebas unitarias y de integración, profesores y estudiantes del CES llevaron a cabo las pruebas de aceptación como clientes.
Se validaron los requisitos y se probaron nuevamente las funcionalidades.
Esta instancia fue de gran utilidad para encontrar errores y diferencias en cuanto a cómo debía funcionar el sistema.

Anexo III: Servicios de API

1. Introducción

En esta sección se presentarán y explicarán los servicios expuestos por la API para una mejor comprensión y posible reutilización de los mismos a futuro. Se organizan por controlador.

Todo controlador se encuentra protegido a los accesos no autorizados. Previo a la ejecución de cada servicio, se evalúa si el usuario existe y se encuentra autenticado. De no ser así se retornará una respuesta con estado 401 y un mensaje indicando que no se encuentra autorizado para realizar dicha acción.

Se especificará en cada servicio cualquier control adicional que exista.

2. Aplicaciones

2.1. Índice

GET /api/v1/stages(.:format)

Este servicio retorna un JSON que contiene un arreglo con cada una de las aplicaciones que existen en el sistema. Cada aplicación contiene su nombre y un arreglo con todos los niveles que dispone. Los detalles del contenido de cada nivel se especifican en la sección 5.8.2.

Si no existen aplicaciones en el sistema se retorna un JSON vacío.

3. Niveles

3.1. Mostrar

GET /api/v1/levels/:id(.:format)

Este servicio retorna el nivel cuyo id corresponda con el parámetro id. Se retorna un JSON con los siguientes valores:

Nombre	***Tipo***	***Descripción***
id	Integer	El identificador en la base de datos.
has_session (opcional)	Boolean	Indicador de si el usuario actual tiene alguna sesión.
session_id	Integer	El identificador de la sesión activa para el usuario actual.
name	String	Nombre.
stage_name	String	Nombre de la aplicación a la que pertenece.
short_description	String	Resumen de la realidad planteada.
long_description	String	Completa descripción de la realidad planteada.
specifications (opcional)	String	Especificaciones o ayudas para el usuario.

objectives	String	Objetivos planteados.
picture	String	URL de la imagen.
test_plan_template	String	URL del registro de pruebas de ayuda.
enabled	Boolean	Indicador de si el usuario actual cumple con las condiciones para cursar el nivel.
approved	Boolean	Indicador de si el usuario actual ha aprobado el nivel.
new	Boolean	Indicador de si el usuario actual puede cursar el nivel pero no tiene una sesión para el mismo (sin importar el estado del mismo).
finished_session	Boolean	Indicador que determina si el usuario actual tiene una sesión finalizada pero pendiente de corrección.

Si el id ingresado no corresponde con el ningún identificador de nivel se retorna un JSON con un mensaje de error.

4. Sesiones

4.1. En vivo

GET /api/v1/levels/:level_id/sessions/live(.:format)

Este servicio brinda la información referente a una sesión activa.

Se retorna un JSON con todos los atributos referentes al nivel (ver sección 3.1 de este anexo) y además se le agregan datos referente a la sesión activa que son retornados en un JSON bajo la clave `session`. Los campos son los siguientes:

Nombre	***Tipo***	***Descripción***
id	Integer	Identificador.
server_url	String	URL de la mutación del nivel.
notes	Array	Contiene las notas realizadas por el usuario para esta sesión. Cada elemento de arreglo contiene los mismos campos que se detallan en la sección 6.1.

Antes de acceder a este servicio se verifica que el usuario tenga acceso a cursar el nivel y no tenga una sesión pendiente de corrección para el mismo.
De no existir una sesión para el nivel especificado, la sesión es creada. Este escenario ocurre cuando un usuario comienza un nivel por primera vez.

De ocurrir un error cuando se crea una sesión se retorna un JSON con un mensaje indicando el error.

4.2. Finalizar

PUT /api/v1/levels/:level_id/sessions/:id/finish(.:format)

El servicio es invocado cuando se finaliza de probar un nivel. El estado de la sesión cambia de activa a finalizada, quedando visible al administrador para su corrección.

El servidor retorna una respuesta vacía (204).

Se debe enviar como parámetro en la ruta el identificador del nivel y la sesión actual, verificando que la sesión corresponda a dicho nivel y que la sesión se encuentre activa, además se debe tener acceso a cursar el nivel. Si alguna de estas condiciones falla, se retorna un JSON con el mensaje de error correspondiente

4.3. Cargar registro de pruebas

PUT /api/v1/levels/:level_id/sessions/:id/test_plan(.:format)

Este servicio se encarga simplemente de asociar el registro de pruebas a la sesión. De ya existir un registro de pruebas cargado previamente por el usuario, éste es sustituido.

Se aceptan los siguientes parámetros anidados bajo la clave `session`:

Nombre	***Tipo***	***Descripción***
test_plan	File	Archivo de registro de pruebas.

Como respuesta se envía un JSON muy parecido a la respuesta de la sección 5.1, donde se envían todos los datos del nivel especificados en la sección 3.1 y además un JSON bajo el atributo `session` de la siguiente forma:

Nombre	***Tipo***	***Descripción***
id	integer	Identificador.
server_url	String	URL de la mutación del nivel.
test_plan_url	String	URL del registro de pruebas cargado por el usuario. De no existir se retorna el valor nulo.
test_plan_identifier	String	Nombre del archivo de registro de pruebas cargado por el usuario. De no existir se retorna el valor nulo.

Antes de ejecutar el servicio se procede a verificar que el identificador del nivel y la sesión se corresponden, que el usuario tenga acceso a cursar el nivel y que la sesión se encuentre activa. Un JSON con el mensaje correspondiente se retorna en caso de fallar alguna de las condiciones anteriores.

5. Notas

5.1. Crear

POST /api/v1/sessions/:session_id/notes(.:format)

Se aceptan los siguientes parámetros anidados bajo la clave `note`”

Nombre	***Tipo***	***Descripción***
text	Text	Texto de la nota.

Se crea una nota para la sesión cuyo identificador coincide con el parámetro id y se retorna un JSON con los siguientes datos:

Nombre	***Tipo***	***Descripción***
id	integer	Identificador.
test	String	Contenido.
session_id	integer	Identificador de la sesión a la que pertenece.
user_id	integer	Identificador del usuario a la que pertenece.

Se retorna un JSON con el error correspondiente en caso de error en la creación.

5.2. Destruir

DELETE /api/v1/sessions/:session_id/notes/:id(.:format)

Se elimina la nota cuyo identificador coincida con el parámetro id. La respuesta es vacía.

De no ser eliminada la nota debido a un error, se retorna un JSON con el mensaje correspondiente.

6. Usuarios

En cada uno de los servicios relacionados con un usuario particular, el identificador es enviado para mantener coherencia en las rutas REST, sin embargo estos datos son obtenidos del usuario autenticado. Si el id enviado no coincide con el id del usuario autenticado se retorna un JSON con el error.

6.1. Mostrar

GET /api/v1/users/:id(.:format)

Este servicio retorna la información correspondiente al usuario. Solo se retornan aquellos datos vinculados directamente con el modelo, no se retornan datos relacionados con los niveles o sesiones correspondidos.

Nombre	***Tipo***	***Descripción***
id	integer	Identificador.
username	String	Alias.
realname	String	Nombre real.
email	String	Correo.
avatar	String	URL de la imagen.
created_at	Datetime	Fecha de creación.
updated_at	Datetime	Fecha de última modificación.

6.2. Actualizar

PUT, PATCH /api/v1/users/:id(.:format)

Se actualiza el usuario autenticado. Se retorna el usuario completo como respuesta como se especifica en la sección 7.1.

6.3. Calificaciones

GET /api/v1/users/:id/grades(.:format)

Este servicio retorna todo lo relacionado con las distintas sesiones del usuario, para todos los niveles de todas las aplicaciones que existen en el sistema. Es por esta razón el servicio con la respuesta más compleja y por ende, con mayor procesamiento.

La respuesta es un JSON que tiene varios niveles de anidación y que tiene la siguiente forma:

Nombre	***Tipo***	***Descripción***
stages	Array	Arreglo de largo igual al número de aplicaciones en el sistema.

Cada entrada del arreglo contiene la siguiente información de la aplicación:

Nombre	***Tipo***	***Descripción***
name	String	Nombre.
average_grade	Integer	Nota promedio del usuario.
completed_levels	Integer	Número de niveles aprobados.
total_levels	Integer	Número total de niveles.
levels	Array	Arreglo de largo igual al número de niveles de cada aplicación.

El arreglo de los niveles muestra la siguiente información del nivel para cada entrada:

Nombre	***Tipo***	***Descripción***
name	String	Nombre.
Si el nivel fue corregido para el usuario		
corrected	Boolean	true
grade	Integer	Nota máxima obtenida.
sessions	Array	Arreglo de largo igual al número de sesiones corregidas del usuario.
Si el nivel NO fue corregido para el usuario		
corrected	boolean	false

Finalmente, el arreglo de sesiones contiene:

Nombre	***Tipo***	***Descripción***
bugs_count	integer	Número de bugs reportados.
grade	integer	Nota.
feedback	String	Devolución.

7. Registros

7.1. Crear

POST /api/v1/auth(.:format)

Servicio que dispone el sistema para que los usuarios se registren. Se aceptan los siguientes parámetros anidados bajo la clave `usuario`:

Nombre	***Tipo***	***Descripción***
username	String	Nombre de usuario.
email	String	Correo.
realname	String	Nombre.
password	String	Contraseña.
password_confirmation	String	Confirmación de contraseña.

Se retorna la misma respuesta especificada en 6.1.

Se valida lo siguiente:

- El correo no puede ser vacío y debe ser único.
- El nombre de usuario no puede ser vacío y debe ser único.
- El nombre no puede ser vacío.
- La contraseña y su confirmación deben coincidir.

En caso de que alguna de las validaciones falle, se retorna un JSON con los errores correspondientes.

8. Sesiones

8.1. Crear

POST /api/v1/auth/sign_in(.:format)

Es el servicio mediante el cual los usuarios se autentican con el sistema. Los parámetros que se aceptan y son necesarios son los siguientes:

Nombre	***Tipo***	***Descripción***
email	String	Correo.
password	String	Contraseña.

Se retorna la misma respuesta especificada en 6.1.
En caso de no coincidir el correo con la contraseña, se retorna un JSON indicando que ambos valores no coinciden.

8.2. Destruir

DELETE /api/v1/auth/sign_out(.:format)

Es el servicio mediante el cual los usuarios terminan su autenticación con el servidor.
El mismo no recibe parámetros ya que el usuario es identificado mediante el token enviado en los cabezales de la petición.

9. Contraseñas

9.1. Crear

POST /api/v1/auth/password(.:format)

Cuando se invoca este servicio se envía un correo al usuario para el cual se haya hecho la petición con las instrucciones a seguir para poder cambiar sus credenciales. Se genera un token específico el cual se encuentra en el correo, para que cuando el usuario quiera cambiar su contraseña, dicha petición se pueda validar, invalidando de esta manera acciones malintencionadas por parte de otro usuario.

Nombre	***Tipo***	***Descripción***
email	String	Correo.
redirect_url	String	URL donde se encuentra el formulario que finalmente actualiza la contraseña de usuario.

9.2. Editar

GET /api/v1/auth/password/edit(.:format)

Este servicio es invocado cuando el usuario sigue las instrucciones de su correo mencionadas en la sección anterior, invoca el servicio mediante un enlace expuesto en el correo.
Este enlace contiene como parámetro el token mediante el cual se busca al usuario (y de esta manera se valida la acción para ese usuario), se genera uno nuevo (por lo que si el usuario no completa la acción de actualizar deberá comenzar nuevamente el proceso de cambio de contraseña) y se redirige al usuario a la ruta enviada como parámetro.

Nombre	***Tipo***	***Descripción***
token	String	Token mediante el cual se identifica al usuario.
redirect_url	String	URL en el frontend donde se encuentra el formulario que finalmente actualiza la contraseña de usuario.

9.3. Actualizar

PUT, PATCH /api/v1/auth/password(.:format)

Es el último servicio que se invoca en el proceso de actualizar la contraseña de un usuario. Se valida nuevamente que el token pertenezca a un usuario y se actualiza su contraseña.

Nombre	***Tipo***	***Descripción***
token	String	Token mediante el cual se identifica al usuario.
password	String	Contraseña.
password_confirmation	String	Confirmación de contraseña.

Se retorna un JSON con mensaje de error en caso de no coincidir la contraseña y su confirmación.

Anexo IV: Gestión de seguridad

Manejo de sesión del usuario tester Test School

El sistema de autenticación manejado para el usuario tester Test School es autenticación sin estado por medio de un token.
El funcionamiento de este sistema es el siguiente:

- El usuario se autentica usando su email y contraseña
- El sistema retorna un token para que el usuario se autentique.
- En cada petición que necesite que el usuario esté autenticado se envía el token en la cabecera del pedido HTTP.

El token es una firma cifrada que permite a la API identificar al usuario que ejecutó el pedido.
Es responsabilidad del cliente o el browser manejar el token de manera correcta y agregarlo en la cabecera de cada pedido HTTP.
Como los tokens son almacenados en el lado del cliente, no hay información de estado y la aplicación se vuelve totalmente escalable.

Se utilizó la gema devise_token_auth en el servidor, y en conjunto con el módulo ng-token-auth utilizado en el cliente angular, permite desarrollar fácilmente este sistema de autenticación.
Estas librerías agregan otros métodos para brindar mayor seguridad a este mecanismo:

- El token es renovado en cada petición. Luego que el usuario efectúa una petición, es procesada por el servidor y se devuelve un nuevo token y se invalida el token anterior.
- Encriptación utilizando MD5. El método de encriptación es editable. Se utilizó MD5 porque necesitamos adaptarnos al mecanismo utilizado por MantisBT debido a que se debe crear un usuario en ambos sistemas con la misma contraseña.
- El token es invalidado cada cierto período de tiempo configurable. En este caso, 2 semanas.
- Soporte de multisesión. Capacidad de manejar varios token simultáneamente para casos donde un usuario realice pedidos desde diferentes dispositivos. Ejemplo desde su dispositivo móvil y su computadora.

Manejo de sesión para el usuario administrador

En la autenticación basada en sesión, el servidor hace todo el trabajo pesado. Una vez que el administrador se autentica enviando su email y contraseña, el servidor responde al cliente un identificador de sesión que es almacenado en las cookies. De esa manera, el navegador adjunta dicho identificador en cada pedido HTTP para que el servidor pueda identificar al administrador.
Para identificarlo, el identificador es guardado en la base de datos, más precisamente en la tabla *admin_users* en donde son almacenados los usuarios administradores.

CORS (Control de acceso HTTP)
Cuando una página web “A” muestra una imagen que se encuentra en una página web “B”, se ejecuta una petición HTTP a un dominio que no es el mismo en que se encuentra la aplicación. Este tipo de solicitudes son denominadas “solicitud HTTP de origen cruzado” o CORS por su sigla en inglés (Cross-Origin Resource Sharing).

En nuestro caso sucede cuando el cliente web AngularJS ejecuta una petición HTTP a la API Ruby on Rails.

Por razones de seguridad los navegadores restringen por defecto este tipo de solicitudes.
CORS da controles de acceso a dominios cruzados para servidores web, lo que habilita la transferencia segura de datos en dominios cruzados.

Este control evita ataques de “denegación de servicios” cuyo objetivo es saturar los servicios computacionales del servidor, no permitiendo el acceso al sitio.

En el caso de la API Ruby on Rails de Test School se configuró el CORS para que acepte pedidos HTTP de tipo GET, POST, OPTIONS, PUT y DELETE solamente del cliente AngularJS.

Anexo V: Especificaciones de escenarios para aplicación Supermarket

Objetivo

En este anexo se describen los escenarios que fueron implantados inicialmente en la aplicación Supermarket junto con los defectos allí inyectados(a grandes rasgos) y las técnicas que se buscan aplicar para detectarlos.

Introducción

Para cada nivel se especifican las funcionalidades disponibles, o las nuevas habilitadas, junto con los elementos que fueron creados desde el panel de administración de Spree para soportar cada realidad.

Cabe destacar que los pasos para ingresar al panel de administración de Spree son:

1. Acceder a la dirección: url_app/admin, siendo "url_app" la dirección de la mutación correspondiente..
2. Ingresar las credenciales, que por defecto son:
    a. Email: spree@example.com
    b. Contraseña: spree123

Escenario 1

Corresponde a la aplicación mutada del nivel 1.

Debido a que es el primer contacto de los estudiantes con el sitio, se espera que los éstos apliquen la técnica de testing exploratorio para familiarizarse con el sistema y detectar defectos en su mayoría leves.

Se presenta una realidad limitada, con varias funcionalidades del sistema inhabilitadas para que los estudiantes puedan enfocarse en un grupo reducido de técnicas durante su primer contacto con la aplicación.

Los productos disponibles en el sistema fueron extraídos de sitio web de la empresa Tienda Inglesa[1].

Las funcionalidades habilitadas para este nivel son:

- Registro e ingreso de usuarios.
- Página principal donde se muestran productos.
- Barra lateral donde se muestran las secciones disponibles y mediante las cuales se pueden filtrar los productos en el listado.
- Acceso a la vista en detalle de cada producto.

La aplicación mutada presenta los siguientes defectos en este nivel:

- Datos erróneos:
    - Errores ortográficos.
    - Errores de traducción.

[1] Sitio de Tienda Inglesa: http://www.tinglesa.com.uy/, accedido en Octubre, 2015.

    - Imágenes erróneas para algunos productos.
    - Productos sin imágenes.
    - Categoría con nombre incorrecto.
    - Categorías sin productos.
    - Algunos productos se muestran en la categoría incorrecta.
- Implementación incorrecta:
    - Durante el inicio de sesión:
        - La contraseña no se muestra encriptada.
        - Validación incorrecta de los correos electrónicos: se permiten mails que no finalizan en ".xyz".
    - Página principal y/o detalle del producto:
        - Precios inconsistentes entre la vista general (índice) y vista en detalle del producto.
        - Precios negativos para algunos productos.

Se busca que los estudiantes apliquen las técnicas de testing exploratorio y clases de equivalencia para detectar los defectos mencionados.

Escenario 2

Corresponde a la aplicación mutada del nivel 2.
Para este nivel se espera que los estudiantes tengan la capacidad de aplicar otras técnicas de testing planificado, para lo cual se habilitaron nuevas funcionalidades del sitio.

Para soportar este caso de estudio se preparó la siguiente realidad:

- Se crearon métodos de envío variando el alcance (nacional o internacional), el cálculo del recargo y el monto de dicho recargo.
- La cadena de supermercados cuenta con tres sucursales, las cuales tienen suficiente stock de todos los productos.
- El sistema cuenta con tres países disponibles para el cobro y/o envío de los paquetes:
    - Uruguay.
    - Argentina.
    - Brasil)y a su vez cada país se divide en zonas de envío.
    - Uruguay:
- Para cada país ingresado en el sistema se agregaron sus correspondientes divisiones políticas, como estados.
  Para ver las divisiones políticas de cada país ver:
  https://es.wikipedia.org/wiki/Organizaci%C3%B3n_territorial_de_Uruguay
  https://es.wikipedia.org/wiki/Provincias_de_Argentina
  https://es.wikipedia.org/wiki/Anexo:Estados_de_Brasil
- Se crearon zonas dentro de cada país, que agrupan algunas de las divisiones mencionadas en el punto anterior. Estas zonas son tomadas en cuenta para determinar el método de envío disponible, en base a la dirección de envío indicada por el usuario:
    - Uruguay:
        - Montevideo: Montevideo.
        - Interior sur: Canelones, Maldonado, Rocha, Treinta y Tres, Lavalleja, Durazno, Río Negro, Soriano, Colonia, Flores, Florida, San José.
        - Interior norte: Artigas, Salto, Paysandú, Tacuarembó, Rivera, Cerro Largo.
    - Brasil: para ver la distribución de las regiones ver: https://es.wikipedia.org/wiki/Regiones_de_Brasil
    - Argentina: se tomó como una zona en su conjunto (zona basada en país y no en estados).
- El sistema cuenta con diversos métodos de envío de los paquetes, según la ubicación de entrega que seleccione el usuario.
  Cada método maneja distintos recargos según la zona de envío:
    - Tiempost:
        - Uruguay:
            - Montevideo, Interior sur
                - Porcentaje plano del 5% sobre el total de la orden.
            - Interior norte.
                - Porcentaje plano del 10% sobre el total de la orden.
    - El Correo:

        - Uruguay:
            - Interior sur.
                - Porcentaje plano del 5% sobre el total de la orden.
            - Montevideo.
                - Porcentaje plano del 3% sobre el total de la orden.
    - Correios brasileiros:
        - Brasil:
            - Norte:
                - Porcentaje plano del 10% sobre el total de la orden.
            - Nordeste, Centro - Oeste, Sudeste, Sur:
                - Porcentaje plano del 7% sobre el total de la orden.
    - FedEx:
        - Brasil: en todas las zonas el recargo es una tarifa plana de USD 20.
        - Argentina: en todas las zonas el recargo es una tarifa plana de USD 15.
        - Uruguay: en todas las zonas el recargo es una tarifa plana de USD 12.
    - DHL:
        - Para todos los países y todas sus zonas maneja una tarifa plana de USD 5 por artículo de la orden.

Además de las funcionalidades del caso de estudio anterior, esta versión cuenta con:

- Manejo del carrito del usuario: implica agregar (especificando la cantidad) y/o quitar productos del carrito.
- Acceso al detalle de la orden de compra, donde se puede configurar la dirección de cobro y/o envío así como también seleccionar el método de envío a partir de la lista de métodos disponibles para el destino seleccionado.

En particular se pretende que los estudiantes profundicen la aplicación de las técnicas clases de equivalencia y combinación por pares, centralizando los defectos en los formularios disponibles en las primeras dos secciones del flujo de compra de una orden.
Principalmente se espera que en la primera sección (“Dirección)” chequeen las validaciones de los campos del formulario, así como también la correcta actualización de la lista de estados disponibles una vez que se selecciona un país.
En la segunda sección (“Envío”) se hará más hincapié en la técnica de combinación por pares al mostrarse los métodos de envío disponibles según la zona seleccionada en la sección anterior. Aquí también será necesario corroborar los cálculos realizados por el sistema mediante testing exploratorio.
Se cuenta con suficiente stock de todos los productos.

Escenario 3

Corresponde a la aplicación mutada del nivel 3.
Para este nivel se espera que los estudiantes se enfoquen fuertemente en la técnica de tablas de decisión, para lo cual se agrega más complejidad al sistema.
A través del panel de administración se agregaron varias promociones sobre las cuales varía el objeto (productos o usuarios), monto y límite de uso del beneficio.

Las nuevas funcionalidades habilitadas para este nivel son:

- Mi cuenta: sección donde el usuario puede ver su correo electrónico y las órdenes que ha generado(compras realizadas). También podrá editar su correo electrónico y contraseña desde este panel.
- Secciones “Pago” y “Completada” dentro del flujo de una compra. En la primera se selecciona el método de pago, que puede ser “Efectivo” o “Tarjeta de crédito”. Para el último caso, los estudiantes deberán corroborar las validaciones de los campos requeridos.
    - Se especificaron a los estudiantes un conjunto de datos válidos de tarjetas de créditos, que también se encuentran en el anexo “Especificación de realidades por nivel”. En realidad el sistema no verifica los campos “Fecha de expiración” y “Código de seguridad” por lo tanto, indiferentemente de qué valores se ingresen allí, el único que el sistema valida es el número de la tarjeta. Sin embargo, se asume que los estudiantes detectarán esto como un defecto y lo reportarán.
    - Una vez realizada la compra, le debe llegar un correo electrónico al usuario con el detalle de la misma. Inicialmente la configuración de los correos es:
        - Asunto: “Supermarket Confirmación de orden #<id_orden>”, donde “id_orden” es el identificador de la orden en el sistema.
        - Correo de origen: ces@supermarket.com.
    -

Para soportar este caso de estudio se crearon las siguientes promociones:

- **Usuario Premium**: algunos usuarios tendrán un beneficio del 5% en todas las compras que realicen (aplicado sobre el subtotal de la orden). Para efectuar el descuento deberán ingresar un código al momento de realizar la compra.
  En la realidad esta promoción es válida solamente para el primer uso del código de cada usuario, pero se alteró la especificación dada a los estudiantes para que lo detecten como un defecto.
  Especificaciones:
    - Código: 1212.
    - Reglas:
        - Un solo uso por usuario.
    - Acciones:
        - Se realiza un ajuste sobre toda la orden realizando un 5% de descuento sobre el subtotal.

- **Feria del Hogar**: todos los productos de la categoría Hogar tienen un 10% de descuento. Para esta promoción se dejaron por fuera las sub-categorías “Libros” y “Muebles”, para que sea detectado como un defecto.
  Especificaciones:
  - Reglas:
    - La orden incluye los siguientes taxons (dentro de la sección “Hogar”):
      - “Colchones”.
      - “Jardín”.
      - “Limpieza ropa”.
      - “Limpieza hogar”.
      - “Mascotas”.
    - La orden debe contener al menos un producto de alguno de los taxons especificados.
  - Acciones:
    - Se crea un ajuste por cada artículo y se aplica un 10% de descuento para cada uno de ellos.
- **Feria de Electrodomésticos:** todos los productos de la categoría “Electrodomésticos” tienen un 10% de descuento.
  Sin embargo, en la especificación dada a los estudiantes se asume que esta promoción ya expiró, por lo que no debería seguir siendo válida.
  Especificaciones:
  - Reglas:
    - La orden incluye los siguientes taxons (dentro de la sección “Electrodomésticos”):
      - “Pequeños Electrodomésticos ”.
      - “Grandes Electrodomésticos”.
      - “Audio - TV - Video”.
    - La orden debe contener al menos un producto de alguno de los taxons especificados.
  - Acciones:
    - Se crea un ajuste por cada artículo y se aplica un 10% de descuento para cada uno de ellos.
- **Promo Lácteos**: Conaprole busca promover el consumo de lácteos, y por eso ofrece un descuento de $5 en cada producto de la línea de leches.
  En realidad esta promoción tiene un descuento de $3 por producto pero se cambió la especificación dada a los estudiantes para que lo detecten como defecto.
  Especificaciones:
  - Reglas:
    - La orden incluye los siguientes productos (dentro de la sección “Comestibles - Lácteos”):
      - Leche Conaprole Fresca Entera Sachet 1lt.
      - Leche Conaprole Fresca Descremada Sachet 1lt.
      - Leche Conaprole Entera Ultra Sachet 1lt.
      - Leche Conaprole Ultra Descremada Sachet 1lt.

        - La orden debe contener al menos un producto de los especificados anteriormente.
    - Acciones:
        - Se crea un ajuste para cada producto de la orden, con tasa plana (descuento) de $3 para cada uno de ellos.
- **Promo Mascotas**: para todos los productos de la categoría mascotas hay un descuento de $10. Esta promoción aplica para un máximo de 4 productos por compra.
  En realidad esta promoción tiene un descuento de $10 para el primer producto y $5 para los dos siguientes. Sin embargo, se alteró la especificación dada a los estudiantes para que lo detecten como un defecto.
  Especificaciones:
    - Reglas:
        - La orden incluye el taxon “Mascotas” (dentro de la sección “Hogar”).
        - La orden debe contener al menos un producto del taxon especificado.
    - Acciones:
        - Se crea un ajuste en toda la orden con tasa flexible donde:
            - El primer artículo tiene un descuento de $10.
            - Los artículos adicionales tienen un descuento de $5.
            - Los descuentos aplican para un máximo de 3 productos.

La especificación de datos para la tarjeta de crédito es la siguiente:

- VISA:
    - Números de tarjeta válidos:
        - 411111111111111
        - 4012888888881881
        - 4222222222222
    - Código de la tarjeta (para todos los números especificados antes): 745.
    - Fecha de expiración: 11/20
- MasterCard:
    - Números de tarjeta válidos:
        - 5500000000000004
        - 5555555555554444
        - 5105105105105100
    - Código de la tarjeta (para todos los números especificados antes): 765.
    - Fecha de expiración: 07/19
- American Express:
    - Números de tarjeta válidos:
        - 378282246310005
        - 371449635398431
        - 378734493671000
        - 340000000000009
    - Código de la tarjeta (para todos los números especificados antes): 785.
    - Fecha de expiración: 08/18
- Discover:
    - Números de tarjeta válidos:

        - 6011000000000004
        - 6011111111111117
        - 6011000990139424
    - Código de la tarjeta (para todos los números especificados antes): 795.
    - Fecha de expiración: 10/21

Los usuarios premium deben ingresar su código al momento de especificar el modo de pago, mientras que los descuentos del resto de las promociones se calculan automáticamente en el detalle de la orden y los ajustes se pueden ver en el detalle del carrito y en las secciones del flujo de la compra.
Las promociones son acumulables para un mismo producto y también pueden aplicarse varias sobre una misma orden.
Se cuenta con suficiente stock de todos los productos.

Escenario 4

Corresponde a la aplicación mutada del nivel 4.
Se introdujo el problema de un ataque al sitio para justificar la presencia de algunos defectos a nivel del flujo de compra así como aspectos de seguridad.

Para este nivel se espera que los estudiantes se enfoquen fuertemente en la técnica de máquinas de estado para lo cual se alteró el flujo normal de compra, aunque también se insertaron algunos defectos a nivel de seguridad y manejo de carro para que los estudiantes apliquen testing exploratorio.
En la figura A1.1 se muestra un diagrama del flujo de una compra, pasando por los posibles estados y especificando qué acción conduce a cada uno de ellos.

Se destacó la importancia de probar el flujo de compras dentro del territorio uruguayo, y en particular dentro de Montevideo para que el estudiante le de prioridad a estas zonas y pueda detectar más fácilmente algunos de los defectos inyectados.

Para soportar la realidad de este caso de uso se preparó la siguiente realidad:

- Existen dos métodos de envío “Tiempost” disponibles para Montevideo. Para esto se modificó el método de envío “Tiempost” que incluía solo la zona “Uruguay - Interior norte”; ahora también incluye “Uruguay - Montevideo”.
- Se eliminaron algunos departamentos de la zona “Uruguay - Interior norte”:
    - Artigas
    - Salto
    - Paysandú
    - Río Negro
- El método de envío Tiempost ya no cubre todos los departamentos debido a la modificación de la zona “Uruguay - Interior norte”.
- Se inyectaron defectos de seguridad a nivel de manejo de sesión del usuario.
- Se inyectaron defectos sobre algunas funcionalidades del carro que deben ser detectadas mediante testing exploratorio.

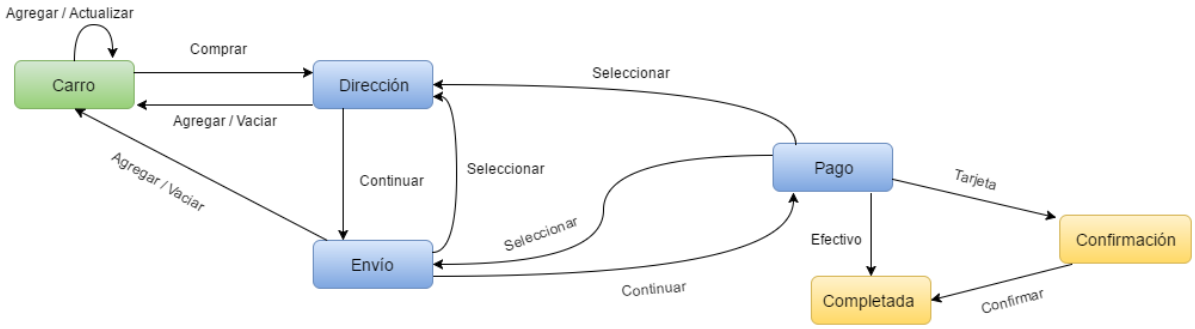


Figura A5.1: Flujo de compra.

Escenario 5

Corresponde a la aplicación mutada del nivel 5.

Para este nivel se asume que los estudiantes tienen el conocimiento para detectar defectos más complejos de usabilidad. Por esta razón se introdujeron algunos defectos que van contra las reglas de usabilidad, de acuerdo a [2]:

- Inconsistencias en el ajuste de pantalla para algunas resoluciones: la cualidad "responsive" no se cumple para algunas vistas.
- Retardos al cargar algunas pantallas: la vista del detalle de un producto tiene un retardo de 8 segundos, lo cual es considerado lento según los estándares para este tipo de páginas.

También se habilitaron nuevas funcionalidades de búsqueda y filtrado de productos, las cuales se deben probar utilizando las técnicas de clases de equivalencia y valores límites.

Nuevas funcionalidades:

- Barra de búsqueda que consiste en un selector de categorías y un campo para ingresar texto:
    - Las categorías disponibles en el selector son las del mismo nivel de la que se encuentre activa en la página principal.
    - Se devuelven las mejores coincidencias dado el texto ingresado.
- Filtros por rango de precios:
    - Esta sección solo aparece disponible en subcategorías o páginas de resultados de búsquedas.

Especificaciones:

- El sitio cuenta con una barra de búsqueda:
    - Junto a la barra de búsqueda, hay un selector donde se puede indicar la sección donde se desea realizar la búsqueda:
        - Desde la página principal, la búsqueda se realiza a nivel de todas las secciones.
        - Desde una categoría, las opciones disponibles son todas las categorías del primer nivel.
        - Desde una subcategoría, las opciones disponibles son aquellas anidadas en su categoría padre.
    - La búsqueda se realiza sobre los nombres de los productos.
    - Los resultados son aquellos que tengan mayor coincidencia con el texto ingresado.
    - La búsqueda se realiza sobre la categoría especificada en el selector.
- Hay filtros de precios disponibles para los productos:
    - Los rangos utilizados son:
        - Por debajo de $10
        - $10 - $50

[2] https://es.wikipedia.org/wiki/Usabilidad#Reglas_de_usabilidad_web

        - $50 - $200
        - $300 - $1,000
        - $1,000 o más
    - Los filtros están disponibles exclusivamente para las subcategorías de productos y los resultados de búsquedas.
    - Al aplicar el filtro, deberían mostrarse solo los productos dentro del rango especificado.
    - En caso de aplicar el filtro sobre los resultados de una búsqueda, se deberían mostrar los productos dentro del rango especificado que a su vez coinciden con los resultados devueltos previamente.

Anexo VI: Defectos inyectados y técnicas sugeridas

1. Referencias

1.1. Indicadores

Para los indicadores de reproducibilidad, severidad y prioridad se manejan las siguientes referencias:

- Prioridad: cero: 10, baja: 20, normal: 30, alta: 40, urgente: 50, inmediata: 60.
- Reproducibilidad: siempre: 10, a veces: 30, random: 50, no se ha probado: 70, no se ha podido replicar: 90, “N/A”: 100.
- Severidad: funcionalidad: 10, trivial: 20, texto: 30, modificación: 40, menor: 50, importante: 60, crash: 70, bloqueo: 80.

1.2. Técnicas recomendadas

Para las técnicas de testing se manejan las siguientes referencias:

- TE: Testing exploratorio.
- CE: Clase de equivalencia.
- TD: Tablas de decisión.
- CP: Combinación por pares.
- ME: Máquinas de estado.
- VL: Valores límite.

2. Niveles

2.1. Nivel 1

Técnicas sugeridas

En la tabla que se muestra a continuación se mencionan las técnicas sugeridas para los distintos tipos de defecto presentes en este nivel:

Tipo de defecto	Técnica
Errores ortográficos.	TE
Errores de traducción.	
Imágenes erróneas para algunos productos.	
Productos sin imágenes.	
Categoría con nombre incorrecto.	
Categorías sin productos.	
Algunos productos se muestran en la categoría incorrecta.	
La contraseña no se muestra encriptada	
Precios inconsistentes entre la vista general (índice) y vista en detalle del producto.	
Precios negativos para algunos productos.	
Validación incorrecta de los emails: se permiten mails que no finalizan en ".xyz".	CE

Tabla A6.1: Tipos de defectos.

Defectos inyectados

En la tabla que se muestra a continuación se detallan los defectos presentes en la versión mutada del nivel 1, incluyendo la reproducibilidad, severidad y prioridad.

Defecto	Localización en la aplicación		Reprod.	Severidad	Prioridad
Errores ortográficos	Vista en detalle del producto	"Prscio" en vez de "Precio"	Siempre	Texto	Normal
Error de traducción	Vista en detalle del producto	Buscar items similares	Siempre	Menor	Normal
Producto sin imagen	Vista en detalle del producto e índice	Leche Conaprole Ultra Descremada	Siempre	Trivial	Baja
		Calefón James 30lt Tanque De Acero	Siempre	Trivial	Baja
		Calefón James 60lt Tanque De Acero	Siempre	Trivial	Baja
		Calefón James 110lt Tanque De Acero	Siempre	Trivial	Baja
Producto con imagen errónea	Vista del índice	Desinfectante Lysoform fragancia original	Siempre	Menor	Alta
Categoría incorrecta	Vista del índice	Perfumería se cambio por Telefonía	Siempre	Importante	Alta
Categoria sin productos	Vista del índice de subcategorías, dentro de Informática	Drones y Fotografía no tienen productos	Siempre	Modificación	Normal
Productos en categoría incorrecta	Vista del índice de la categoría Hogar-Colchones	Yerba Canarias 1Kg: se agregó también en Colchones, dentro de Hogar	Siempre	Menor	Alta
		Yerba Sara Suave 1kg: se agregó también a Colchones, dentro de Hogar.	Siempre	Menor	Alta
		Yerba Canarias Natural Serena 1kg: se agregó también a Colchones, dentro de Hogar.	Siempre	Menor	Alta
	Vista del indice de la categoría Textiles-Zapateria	Carne Picada Super Especial: se agregó también a categoría Zapatería, en Textiles	Siempre	Menor	Alta
		Azúcar Bella Union 1kg: se agregó también a categoría Zapatería, en Textiles	Siempre	Menor	Alta
La contraseña se muestra como texto	Registro de usuario		Siempre	Menor	Normal
Se permiten mails sin el formato ".xyz"	Registro de usuario		Siempre	Importante	Normal

Precios inconsistentes	Índice y detalle del producto	Los productos de la categoría Bebidas tienen precios inconsistentes entre las vistas de índice y detalle.	Random	Importante	Urgente
Precios negativos	Detalle del producto	Los productos de las categorías “Indumentaria” y “Zapatería”, dentro de “Textiles” figuran con precios negativos.	A veces	Importante	Urgente

2.2. Nivel 2

Técnicas sugeridas

En la tabla que se muestra a continuación se mencionan las técnicas sugeridas para los distintos tipos de defecto presentes en este nivel:

Tipo de defecto	Técnica
Validaciones incorrectas en campos de la dirección.	CE
Un campo está marcado como obligatorio y no lo es.	
Estados no corresponden al país seleccionado,	
Montos de envío calculados de forma incorrecta.	TE
En el resumen de la orden, la moneda varía.	
Cuando se varía el método de envío, no se actualiza el monto total de la orden	
No se valida que la ciudad ingresada pertenezca al país correcto.	
Método de envío no corresponde a la zona indicada.	CP

Defectos inyectados

En la tabla que se muestra a continuación se detallan los defectos presentes en la versión mutada del nivel 2, incluyendo la reproducibilidad, severidad y prioridad.
En la segunda columna se indica en cuál de las 2 secciones del flujo de compra de una orden se encuentra el campo mencionado, y consecuentemente el defecto (“Dirección” o “Envío”).

Defecto	Nombre de sección	Reproducibilidad	Severidad	Prioridad
No hay validación del código ZIP	Dirección	Siempre	Menor	Normal
No hay validación del teléfono	Dirección	Siempre	Menor	Normal
Nombre vacío	Dirección	Siempre	Importante	Alta
Apellido vacío	Dirección	Siempre	Importante	Alta
Estado Formosa no pertenece a Uruguay	Dirección	Siempre	Menor	Normal
Estado San Luis no pertenece a Uruguay	Dirección	Siempre	Menor	Normal
El campo "Dirección alternativa" está marcado como obligatorio y no lo es	Dirección	Siempre	Trivial	Baja
En la tab Dirección, no se valida que una ciudad ingresada pertenezca al país indicado debajo.	Dirección	Siempre	Retoque	Normal
Los montos de los métodos de envío están mal calculados (DHL)	Envío	A veces	Importante	Urgente
El método de envío “Tiempost” para Montevideo y la zona “Interior sur” no está calculado correctamente (debería ser 5% y es 7%).	Envío	Siempre	Importante	Inmediata
El métodos de pago "Correios brasileiros" figura disponible para "Uruguay - Montevideo" y "Uruguay - Interior Sur"	Envío	Siempre	Importante	Alta
En el resumen de la orden, la moneda varía.	Envío	A veces	Importante	Inmediata
En la sección de envío, al cambiar el método de envío no se actualiza el monto total de la orden (de acuerdo al nuevo costo de envío)	Envío	Siempre	Menor	Normal

2.3. Nivel 3

Técnicas sugeridas

En la tabla que se muestra a continuación se mencionan las técnicas sugeridas para los distintos tipos de defecto presentes en este nivel:

Tipo de defecto	Técnica
Descuento solo aplica a primeros 3 productos en promo Mascotas	VL
El descuento para la promo Mascotas aplica para el primer producto y los otros 2 tienen otro descuento	
En el detalle de la orden, aparecen un máximo de 10 productos.	TE
Una promoción que ya expiró sigue siendo válida	
En el mail de confirmación de compra, no aparece especificado el método de envío	
El código solo es válido para el primer uso en la promoción usuario premium	TD
Los libros y los muebles no tienen descuento de la promo Hogar	
El descuento no es correcto en la promoción Lácteos	
No se validan algunos campos de la tarjeta de crédito	

Defectos inyectados

En la tabla que se muestra a continuación se detallan los defectos presentes en la versión mutada del nivel 3, incluyendo la reproducibilidad, severidad y prioridad.
En la segunda columna se indica a cuál de las promociones refiere el defecto, en caso que aplique.

Defecto	Nombre de promoción	Reprod.	Severidad	Prioridad
El código solo es válido para el primer uso	Usuario Premium	Siempre	Importante	Alta
Promoción no debería ser válida (ya expiró)	Feria de Electrodomésticos	Siempre	Importante	Alta
Los libros no entran en el descuento	Feria del Hogar	Siempre	Modificación	Normal
Los muebles no entran en el descuento	Feria del Hogar	Siempre	Modificación	Normal
El descuento aplica para el primer producto y los otros 2 tienen otro descuento	Promo Mascotas	Siempre	Importante	Normal
El descuento solo aplica para los primeros 3 productos(y es diferenciado)	Promo Mascotas	Siempre	Importante	Normal
El descuento no es correcto (es de $3 por producto y debería ser $5)	Promo Lácteos	Siempre	Menor	Normal
En el detalle de la orden, aparecen un máximo de 10 productos.		Siempre	Importante	Alta
En la sección Pago, no se valida la fecha de expiración de la tarjeta de crédito		Siempre	Importante	Alta
En la sección Pago, no se valida el código de seguridad de la tarjeta de crédito.		Siempre	Importante	Alta
En el mail de confirmación de compra, no aparece especificado el método de envío		Siempre	Menor	Normal

2.4. Nivel 4

Técnicas sugeridas

En la tabla que se muestra a continuación se mencionan las técnicas sugeridas para los distintos tipos de defecto presentes en este nivel:

Tipo de defecto	Técnica
Desde la sección Pago no se pueden hacer compras en efectivo.	ME
Desde la sección Pago, al intentar guardar e ir a Confirmar redirige a la sección Dirección.	
Se pierden los productos del carrito al ingresar.	
Desde la sección Envío, se genera un crash cuando se intenta ir a Dirección.	
Se cierra la sesión después de realizar una compra.	
La acción "Vaciar" desde la vista del detalle del carro genera un crash.	TE
Para Montevideo aparecen 2 opciones de método de envío Tiempost.	
Para algunos departamentos no hay método de envío disponible	
El botón Actualización (en el detalle del carro) no funciona.	

Defectos inyectados

En la tabla que se muestra a continuación se detallan los defectos presentes en la versión mutada del nivel 4, incluyendo la reproducibilidad, severidad y prioridad.

Defecto	Detalle	Reprod.	Severidad	Prioridad
Desde la sección Pago no se pueden hacer compras en efectivo.	Solo se puede comprar con tarjeta. Si se indica "efectivo" como método de pago, colapsa la aplicación al darle "Continuar".	Siempre	Bloqueo	Inmediata
El botón Actualización (en el detalle del carro) no funciona.	El botón Actualización (en el detalle del carro) no funciona (no hace ninguna acción).	Siempre	Importante	Alta
Desde la sección Pago, al intentar guardar e ir a Confirmar redirige a la sección Dirección.	Desde la sección Pago, al intentar guardar e ir a Confirmar redirige a la sección Dirección aleatoriamente.	A veces	Importante	Urgente
La acción "Vaciar" desde la vista del detalle del carro genera un crash.	La acción "Vaciar" para vaciar el carro desde la vista del detalle del mismo genera un crash.	Siempre	Crash	Urgente
Se pierden los productos del carrito al ingresar.		Siempre	Menor	Normal
Para Montevideo aparecen 2 opciones de método de envío Tiempost.		Siempre	Modificación	Normal
Se cierra la sesión después de realizar una compra.		Siempre	Trivial	Baja
Para algunos departamentos no hay método de envío disponible.		Siempre	Importante	Alta
Desde la sección Envío, se genera un crash cuando se intenta ir a Dirección.		Siempre	Crash	Urgente

2.5. Nivel 5

Técnicas sugeridas

En la tabla que se muestra a continuación se mencionan las técnicas sugeridas para los distintos tipos de defecto presentes en este nivel:

Tipo de defecto	Técnica
La búsqueda no devuelve los productos filtrados por la categoría seleccionada.	CE
En el filtro de precios no se devuelven productos en cierto rango.	VL
Al filtrar productos por un rango de precios, se devuelven productos fuera de dicho rango.	
Algunos mensajes de error son incorrectos.	TE
Algunas páginas no se ajustan a la resolución.	
Hay un retardo importante al cargar algunas de las pantallas.	
Algunos caracteres aparecen mal codificados.	
El correo de confirmación de compra no se envía o se envía duplicado.	

Defectos inyectados

En la tabla que se muestra a continuación se detallan los defectos presentes en la versión mutada del nivel 5, incluyendo la reproducibilidad, severidad y prioridad.

Defecto	Detalle	Reprod.	Severidad	Prioridad
La búsqueda no devuelve los productos filtrados por la categoría seleccionada.	Si se selecciona una categoría y se ingresa texto en la barra de búsqueda, los productos resultantes no están filtrados por dicha categoría.	Siempre	Importante	Alta
Los productos en determinado rango de precios no son detectados nunca en el filtro de precios.	Productos entre 200 y 300 pesos no entran en ningún filtro	Siempre	Importante	Urgente
Mensajes de error incorrectos	En la sección Pago, al intentar avanzar sin completar los campos de la tarjeta de crédito, en lugar de decir "X errores impidieron que se guardara este registro", dice "Bienvenido al sitio"	Siempre	Trivial	Baja
	En la vista de resultados de la búsqueda, si no hay resultados, el mensaje en vez de decir "No se encontraron productos" dice "Se encontraron 50 productos"	Siempre	Trivial	Baja
En el filtro de precios se devuelven productos por fuera del rango.	El filtro de precios mayores a $1000 muestra productos de precios mayores o iguales a $800.	Siempre	Menor	Normal
En la sección "Mi cuenta" se puede scrollear horizontalmente al disminuir los 600px de ancho.	La página Mi cuenta es scrolleable horizontalmente al disminuir los 600px de ancho.	A veces	Retoque	Baja
En las secciones "Mi Cuenta" y "Envío", el encabezado de artículo se sobrepone a los detalles de la orden para resoluciones más pequeñas.	Si el ancho es menor a 1200px el encabezado de articulo se sobrepone a los detalles de la orden.	Siempre	Retoque	Baja
Al seleccionar la sección "Bebidas" tarda mucho en cargar la página.	Al seleccionar la sección "Bebidas", la página tarda entre 6 y 10 seg en cargar.	Siempre	Importante	Normal
Demora mucho en cargar la nueva página al terminar un compra.	Al finalizar una compra, la pantalla tarda entre 12 y 15	Siempre	Importante	Normal

	segundos en cargar.			
Algunas letras parecen mal codificadas.	Para las letras con tilde y "ñ" se utilizó una codificación Western (Windows-1252).	Siempre	Texto	Normal
El correo de confirmación de compra a veces no llega.	El correo de confirmación de compra a veces no llega o llega dos veces.	Random	Importante	Urgente

Anexo VII: Escenarios presentados a estudiantes

Objetivo

En este anexo se presentan los ejercicios que deberán resolver los estudiantes, los cuales son agregados a través del panel de administración para ser luego mostrados en la aplicación web con el mismo formato en que fueron agregados.

Escenario 1

Comience a explorar la primera versión del producto probando algunas funcionalidades básicas, como lo son registro e ingreso y la vista principal de los productos.

La cadena de supermercados Supermarket acaba abrir su tercer sucursal en el país y busca expandir su mercado lanzando un sitio web e-commerce donde los usuarios podrán ver los productos que hay a la venta y realizar compras a través del mismo.

En esta instancia se lanzará una primera versión del sitio para así medir la recepción del público a esta nueva herramienta.
Esta primera versión consta de las siguientes funcionalidades:

- Registro e ingreso de usuarios.
- Página principal donde se muestran productos.
- Barra lateral donde se muestran las secciones disponibles y mediante las cuales se pueden filtrar los productos en el listado.
- Acceso a la vista en detalle de cada producto.

Objetivo
El estudiante debe familiarizarse con el sistema, en una versión limitada, y aplicar las técnicas de testing exploratorio y clases de equivalencia.

Escenario 2

Agregue productos al carrito para probar la configuración de direcciones y métodos de envío correspondientes a una orden.

Luego de obtener los primeros comentarios de los usuarios sobre el sitio web, la empresa decide lanzar una segunda versión sobre la cual éstos podrán tener su propio carrito de compras y configurar sus datos de envío.

Además de las funcionalidades del caso de estudio anterior, esta versión cuenta con:

- Manejo del carrito del usuario: implica agregar (especificando la cantidad) y/o quitar productos del carrito.
- Acceso al detalle de la orden de compra, donde se puede configurar la dirección de cobro y/o envío así como también seleccionar el método de envío a partir de la lista de métodos disponibles para el destino seleccionado.

Objetivo
En este nivel se busca probar las funcionalidades de las primeras dos etapas del flujo de una compra, aplicando técnicas de clases de equivalencia y combinación por pares.

Especificaciones:

- Las tres sucursales de la cadena de supermercados tienen suficiente stock de todos los productos.
- El sistema cuenta con tres países disponibles para el cobro y/o envío de los paquetes (Uruguay, Argentina y Brasil) y a su vez cada país se divide en zonas de envío.
    - Uruguay:
        - Montevideo: Montevideo.
        - Interior sur: Canelones, Maldonado, Rocha, Treinta y Tres, Lavalleja, Durazno, Río Negro, Soriano, Colonia, Flores, Florida, San José.
        - Interior norte: Artigas, Salto, Paysandú, Tacuarembó, Rivera, Cerro Largo.
    - Brasil: para ver la distribución de las regiones ver: https://es.wikipedia.org/wiki/Regiones_de_Brasil
- El sistema cuenta con diversos métodos de envío de los paquetes, según la ubicación de entrega que seleccione el usuario.
  Cada método maneja distintos recargos según la zona de envío:
    - Tiempost:
        - Uruguay:
            - Montevideo, Interior sur
                - Porcentaje plano del 5% sobre el total de la orden.
            - Interior norte.
                - Porcentaje plano del 10% sobre el total de la orden.
    - El Correo:
        - Uruguay:
            - Interior sur.
                - Porcentaje plano del 5% sobre el total de la orden.

- Montevideo.
  - Porcentaje plano del 3% sobre el total de la orden.
- Correios brasileiros:
  - Brasil:
    - Norte:
      - Porcentaje plano del 10% sobre el total de la orden.
    - Nordeste, Centro - Oeste, Sudeste, Sur:
      - Porcentaje plano del 7% sobre el total de la orden.
- FedEx:
  - Brasil: en todas las zonas el recargo es una tarifa plana de USD 20.
  - Argentina: en todas las zonas el recargo es una tarifa plana de USD 15.
  - Uruguay: en todas las zonas el recargo es una tarifa plana de USD 12.
- DHL:
  - Para todos los países y todas sus zonas maneja una tarifa plana de USD 5 por artículo de la orden.

Escenario 3

Investigue las promociones disponibles, chequee los beneficios y aproveche al máximo los descuentos disponibles para usuarios y productos.

Con motivo del primer aniversario de la inauguración de la sucursal Prado, “Supermarket” decidió ofrecer varias promociones (válidas para toda la cadena) sobre sus productos y beneficios especiales para sus mejores clientes.

Para este nivel se encuentran habilitadas las siguientes secciones:

- Mi cuenta: allí se puede editar el mail y/o contraseña del usuario y se puede ver el detalle de las órdenes (compras realizadas).
- Las últimas dos etapas del flujo de la compra: “Pago” y “Completada”. De esta manera los usuarios podrán completar una compra y ver la orden generada en la sección “Mi cuenta”.
    - En la sección “Pago” el usuario debe seleccionar un método de pago a partir de las dos opciones disponibles: Efectivo o tarjeta de crédito. Para éste último es necesario ingresar determinados datos de la tarjeta (especificados abajo).
    - Una vez realizada la compra, le debe llegar un correo electrónico al usuario con el detalle de la compra.

Objetivo: en este nivel se busca que el estudiante aplique las técnicas de tablas de decisión, valores límites y clases de equivalencia para verificar el correcto funcionamiento de las promociones creadas y del resto de las funcionalidades habilitadas.

Especificaciones:
Se cuenta con suficiente stock de todos los productos.
Las promociones disponibles son:

- **Usuario Premium**: algunos usuarios tendrán un beneficio del 5% en todas las compras que realicen (aplicado sobre el subtotal). Para efectuar el descuento deberán ingresar un código al momento de realizar la compra. Código: 1212
- **Feria del Hogar**: durante el mes pasado, el supermercado lanzó la “Feria de electrodomésticos” que ofrecía un descuento del 10% sobre todos los electrodomésticos. Este mes la promoción aplica a todos los productos de la categoría Hogar, que tienen un 10% de descuento.
- **Promo Lácteos**: Conaprole busca promover el consumo de lácteos, y por eso ofrece un descuento de $5 en cada producto de la línea de leches.
- **Promo Mascotas**: para todos los productos de la categoría mascotas hay un descuento de $10. Esta promoción aplica para un máximo de 4 productos por compra.

Las siguientes tarjetas de crédito válidas pueden ser utilizadas para probar las compras:

- VISA:

    - Números de tarjeta válidos:
        - 411111111111111
        - 4012888888881881
        - 4222222222222
    - Código de la tarjeta (para todos los números especificados antes): 745.
    - Fecha de expiración: 11/20
- MasterCard:
    - Números de tarjeta válidos:
        - 5500000000000004
        - 5555555555554444
        - 5105105105105100
    - Código de la tarjeta (para todos los números especificados antes): 765.
    - Fecha de expiración: 07/19
- American Express:
    - Números de tarjeta válidos:
        - 378282246310005
        - 371449635398431
        - 378734493671000
        - 340000000000009
    - Código de la tarjeta (para todos los números especificados antes): 785.
    - Fecha de expiración: 08/18
- Discover:
    - Números de tarjeta válidos:
        - 6011000000000004
        - 6011111111111117
        - 6011000990139424
    - Código de la tarjeta (para todos los números especificados antes): 795.
    - Fecha de expiración: 10/21

Los usuarios premium deben ingresar su código al momento de especificar el modo de pago, mientras que los descuentos del resto de las promociones se calculan automáticamente en el detalle de la orden y los ajustes se pueden ver en el detalle del carrito y en las secciones del flujo de la compra.
Las promociones son acumulables para un mismo producto y también pueden aplicarse varias sobre una misma orden.

Escenario 4

Como consecuencia de un ataque al sistema es necesario probar el flujo completo de una compra y corroborar que la seguridad no ha sido comprometida.

Se sospecha que el sistema de Supermarket fue infiltrado por un hacker y alguna de las funcionalidades fueron afectadas.
Dado que las compras por internet significan un porcentaje importante de las ventas totales, es fundamental que el flujo de una compra funcione correctamente y por esta razón pidieron verificar dicho flujo para saber si es necesario actualizar el sistema cuanto antes.
La mayoría de las compras son realizadas dentro de Uruguay y en particular Montevideo es el departamento con mayor demanda, por lo tanto la prioridad es corroborar que no haya inconsistencias en estas zonas.

Objetivo: en este nivel se busca que el estudiante aplique la técnica máquinas de estado sobre el flujo de una compra y también se enfoque en bugs de seguridad que se puedan haber inyectado en el sistema luego que el mismo fue infiltrado.

Escenario 5

Pruebe las nuevas funcionalidades de la página web; realice búsquedas de productos y filtre según rangos de precios.

Debido a los buenos comentarios de los usuarios del sitio web, éste ha crecido mucho y los dueños de la cadena Supermarket están preocupados por la performance del sitio y la compatibilidad con diversos dispositivos de variadas resoluciones. Por este motivo decidieron mejorarlo, agregando nuevas funcionalidades que mejoren aún más la experiencia de usuario.

Especificaciones:

- El sitio cuenta con una barra de búsqueda:
    - Junto a la barra de búsqueda, hay un selector donde se puede indicar la sección donde se desea realizar la búsqueda:
        - Desde la página principal, la búsqueda se realiza a nivel de todas las secciones.
        - Desde una categoría, las opciones disponibles son todas las categorías del primer nivel.
        - Desde una subcategoría, las opciones disponibles son aquellas anidadas en su categoría padre.
    - La búsqueda se realiza sobre los nombres de los productos.
    - Los resultados son aquellos que tengan mayor coincidencia con el texto ingresado.
    - La búsqueda se realiza sobre la categoría especificada en el selector.
- Hay filtros de precios disponibles para los productos:
    - Los rangos utilizados son:
        - Por debajo de $10
        - $10 - $50
        - $50 - $200
        - $300 - $1,000
        - $1,000 o más
    - Los filtros están disponibles exclusivamente para las subcategorías de productos y los resultados de búsquedas.
    - Al aplicar el filtro, deberían mostrarse solo los productos dentro del rango especificado.
    - En caso de aplicar el filtro sobre los resultados de una búsqueda, se deberían mostrar los productos dentro del rango especificado que a su vez coinciden con los resultados devueltos previamente.

Objetivo: en este nivel se busca que el estudiante aplique las técnicas de testing exploratorio, clases de equivalencia y valores límites para comprobar el funcionamiento de las nuevas funcionalidades. También se busca detectar problemas de usabilidad.

Anexo VIII: Ranking de aplicaciones evaluadas

Objetivo

Aquí se presenta una lista completa de aplicaciones de código abierto que se tomó como punto de partida (esta lista se obtuvo de [46] y [47]) ordenada por los valores de cada categoría, lo cual se tomó como primer filtro.
Si se desea ver la lista completa de aplicaciones, ver [46] ó [47].

Aplicaciones

En esta sección se presentan en primer lugar las aplicaciones que fueron analizadas para ser integradas al sistema, agrupadas en orden descendente por el valor asignado a su categoría. Luego se presentan aquellas que no fueron analizadas en profundidad ya que fueron descartadas por pertenecer a categorías que no se ajustan a las necesidades del sistema.

Debido a la gran cantidad de aplicaciones disponibles se realizaron algunos filtros antes de evaluarlas individualmente, como fue explicado en el capítulo 3, sección “Aplicación utilizada”.
Todos los parámetros allí mencionados fueron evaluados, ponderados y agrupados en conceptos más generales, cuya valoración se calcula como la suma de los parámetros que agrupa.
Estos conceptos generales engloban la adaptabilidad a los usuarios finales (estudiantes de testing) así como también facilidad para integrar cada aplicación al sistema y la estabilidad de la misma.
Si alguno de estos factores resultó notoriamente deficiente, se optó por no continuar con la evaluación de dicha aplicación, y proseguir con la siguiente para optimizar el tiempo de búsqueda.

Los conceptos generales mencionados se distribuyen de la siguiente manera:

1. Adaptabilidad a los usuarios (testers)
    - Complejidad media del funcionamiento
    - Entretenimiento
    - Variedad de funcionalidades
    - Tamaño: la forma de valorar el tamaño es de acuerdo a qué tan adecuado sea el mismo. Una puntuación alta quiere decir que la aplicación cuenta con suficientes funcionalidades y aún así no presenta complejidades respecto a su tamaño a los usuarios ni a los desarrolladores.
2. Facilidad de integración:
    - Facilidad de acceso al código fuente
    - Facilidad de instalación
    - Documentación disponible y comprensión del código fuente
3. Estabilidad:
    - Última actualización (a diciembre de 2015): a este parámetro se le asigna valor 5 si la aplicación fue actualizada en el último año. De lo contrario se decrementa en 1 la ponderación por cada año hasta diciembre de 2011; todas las actualizaciones anteriores a esa fecha serán valoradas en 1.
    - Recomendaciones de usuarios

4. Otros:
    - Valor de la categoría
    - Lenguaje: la forma de evaluar este factor fue muy subjetiva ya que se ponderó con un valor alto a los lenguajes más conocidos por los desarrolladores, y aquellos con los que se sienten más cómodos. La asignación de puntos fue la siguiente:
        - Ruby: 5.
        - Java: 3.
        - PHP: 3.
        - Python: 2.
        - C#: 2.
        - Node.js: 1.
        - Perl: 1.
        - .NET: 1.

Los conceptos generales fueron ponderados en base al siguiente cálculo:

- A cada uno de los parámetros anidados (subcategorías) se les asigna un valor del 1 al 5.
- Dichos parámetros se suman para determinar el valor del parámetro padre (concepto general).
- Una ponderación de 1 significa que ese aspecto de la aplicación es negativo, mientras que el valor 5 significa que tuvo una evaluación positiva.

A continuación se presentan las tablas de valores obtenidos para las categorías mejor ponderadas, junto con los valores totales de los conceptos generales.

Se descartaron aplicaciones que no se pueden ejecutar en un sistema operativo Linux, por lo que se presentan sólo aquellas que son compatibles con este sistema.

Valor de categoría 4

En esta primera sección se pueden ver las aplicaciones con más alto valor de categoría (las consideradas más apropiadas) junto con la especificación de dichas categorías y el lenguaje de la aplicación.

Aplicación	Categoría	Adaptabilidad a los usuarios	Facilidad de integración	Estabilidad	Otros	Lenguaje	Suma
Spree	Comercio Electrónico	15	17	9	9	RUBY	50
Broadleaf Commerce	Comercio Electrónico	13	16	8	7	JAVA	44
Magento	Comercio Electrónico	11	10	6	7	PHP	34
Eclime	Comercio Electrónico	15	10	3	7	PHP	35
OpenCart	Comercio Electrónico	12	15	8	7	PHP	42
osCommerce	Comercio	10	11	6	7	PHP	34

	Electrónico						
X-Cart	Comercio Electrónico	16	6	8	7	PHP	37

Valor de categoría 3

A continuación se presentan las aplicaciones de nivel de categoría 3, las cuales siguen siendo apropiadas para los usuarios aunque con algo más de dificultad que las de valor 4. Al igual que en el cuadro anterior, se muestra la especificación de las categorías y el lenguaje de cada aplicación.

Aplicación	Categoría	Adaptabilidad a los usuarios	Facilidad de integración	Estabilidad	Otros	Lenguaje	Suma
B2evolution	Blogs	12	8	9	6	PHP	35
LibrePlan	Manejo de Proyectos	12	9	9	6	JAVA	36
web2Project	Manejo de Proyectos	11	10	7	6	PHP	34
GanttProject	Manejo de Proyectos	10	12	9	6	JAVA	37
Ofuz	Manejo de Proyectos	10	9	2	6	PHP	27
Digaboard	Manejo de Proyectos	10	8	4	6	PHP	28
Achievo	Manejo de Proyectos	10	10	4	6	PHP	30
Dotproject	Manejo de Proyectos	8	6	4	6	PHP	24
LifeType	Blogging	9	7	5	6	PHP	27
Mantis	Rastreo de defectos	11	11	10	6	PHP	38
Bugzilla	Rastreo de defectos	11	7	8	4	Perl	30
FlySpray	Rastreo de defectos	14	12	9	6	PHP	41
Redmine	Rastreo de defectos, Manejo de Proyectos	12	14	9	8	RUBY	43
Trac	Rastreo de defectos, Wiki	12	13	7	5	Python	37
Collabtive	Colaboración	13	12	9	6	PHP	40
cyn.in	Colaboración	9	5	3	5	Python	22
Feng Office	Colaboración	14	8	7	6	PHP	35
IGSuite	Colaboración	9	7	4	4	Perl	24
Open Atrium	Colaboración	10	8	8	6	PHP	32
phpGroupWare	Colaboración	9	8	1	6	PHP	24
Simple Groupware	Colaboración	7	6	3	6	PHP	22
OnlyOffice	Colaboración	8	5	2	4	Node.js	19
GlobalSight	Traducciones	12	7	8	5	C#	32
ATutor	Educación en línea	13	14	8	6	PHP	41
Moodle	Educación en línea	15	16	10	6	PHP	47
Canvas LMS	Educación en línea	12	11	7	8	RUBY	38
Chamilo	Educación en línea	10	10	8	6	PHP	34
ILIAS	Educación en línea	9	14	8	6	PHP	37
eHour	Rastreo de Tiempo	14	10	7	6	JAVA	37
BORG Calendar	Listas de Tareas/Organizadores/Calendarios	13	15	9	6	JAVA	43
Makagiga	Listas Tareas/Organizadores/Calendarios	14	10	7	6	JAVA	37

RedNotebook	Listas de Tareas/Organizadores/Calendarios	12	11	9	6	PYTHON	38
Task Coach	Listas de Tareas/Organizadores/Calendarios	15	10	8	6	PYTHON	39
ThinkingRock	Listas de Tareas/Organizadores/Calendarios	11	7	6	6	JAVA	30

Aplicaciones no analizadas

Por último se presentan las aplicaciones que pertenecen a categorías que no se consideran apropiadas para nuestro sistema y por lo tanto no fueron investigadas en profundidad.
Estas categorías resultan poco amigables para los usuarios finales del sistema y las aplicaciones son complejas en la mayoría de los casos.

Aplicación	Categoría
Edoceo Imperium	Contabilidad
FrontAccounting	Contabilidad
GnuCash	Contabilidad
Lazy8 Ledger	Contabilidad
LedgerSMB	Contabilidad
TurboCASH	Contabilidad
Argentum	Pagos
jBilling	Pagos
Simple Invoices	Pagos
Siwapp	Pagos
Drupal	Blog/CMS
Joomla	Blog/CMS
WordPress	Blog/CMS
Chromium	Exploradores
Dooble	Exploradores
Firefox	Exploradores
Qt Web Browser	Exploradores
Tor	Exploradores
GraphViz	Gráficos
HighCharts	Gráficos
Scheme Maker	Dibujos
CitrusDB	Manejo de Relaciones con Clientes (CRM)
CiviCRM	Manejo de Relaciones con Clientes (CRM)
ConcourseSuite	Manejo de Relaciones con Clientes (CRM)
Covide	Manejo de Relaciones con Clientes (CRM)
Daffodil CRM	Manejo de Relaciones con Clientes

	(CRM)
SugarCRM	Manejo de Relaciones con Clientes (CRM)
vtiger CRM	Manejo de Relaciones con Clientes (CRM)
Compiz	Mejoras de Escritorio
DropIt	Mejoras de Escritorio
Electric Sheep	Mejoras de Escritorio
Emerge	Mejoras de Escritorio
Enlightenment	Mejoras de Escritorio
Launchy	Mejoras de Escritorio
Safe Exam Browser	Testing Educativo
iTest	Testing Educativo
TCExam	Testing Educativo
ChildsPlay	Educación Elemental
GCompris	Educación Elemental
GPaint	Educación Elemental
KLettres	Educación Elemental
Little Wizard	Educación Elemental
TuxMath	Educación Elemental
Tux Paint	Educación Elemental
Buddi	Manejo de Finanzas Personales
Chartsy	Manejo de Finanzas Personales
Grisbi	Manejo de Finanzas Personales
HomeBank	Manejo de Finanzas Personales
LightWallet	Manejo de Finanzas Personales
LimeSurvey	Manejo de Encuestas
AkelPad	Editor de Texto
Emacs	Editor de Texto
jEdit	Editor de Texto
TEA	Editor de Texto
DokuWiki	Wiki
MediaWiki	Wiki
TWiki	Wiki

Anexo IX: Herramientas que soportan inyección de defectos

A continuación se listan algunas herramientas que soportan inyección de defectos y se encuentran en el mercado:

Xception[28]: Cuando funciona como herramienta SWIFI pura intenta usar las características de depuración y monitorización, presentes en la mayoría de los procesadores para inyectar defectos más realistas. Además, monitoriza la activación de los defectos y su impacto en el sistema, mostrando su comportamiento al detalle. Xception provee un conjunto de inyección de defectos, incluyendo defectos espaciales, temporales y defectos relacionados con la manipulación de los datos en la memoria. Se basa en la utilización de mecanismos de depuración implementados en el procesador a través de las excepciones para realizar la inyección. Primero deja ejecutar a la aplicación hasta el momento del disparo, en el que se entra en modo traza y se ejecuta paso a paso hasta la instrucción a inyectar. Una vez en la instrucción a inyectar se decodifica y se decide qué parámetros de la misma modificar en función del tipo de fallo que se pretenda inyectar.

Goofi [29] (Generic Object-Oriented Fault Injection): está diseñado para poder adaptarse a varios sistemas bajo prueba y a diferentes técnicas de inyección de defectos. Esta herramienta es altamente portable entre diversas plataformas ya que está construida en Java y su base de datos en SQL. Permite realizar inyección de defectos mediante software (SWIFI) y Scan Chain (SCIFI).

Mafalda [30] (Microkernel Assesment by Fault injection AnaLisys and Design Aid) está orientada a la validación de microkernels (núcleos del sistema operativo) y permite realizar dos tipos de inyecciones. La corrupción de los parámetros de entrada al llamar a una primitiva del microkernel y la inyección de fallos en segmentos de memoria, de código y datos.

Exhaustif [31]: está diseñado para analizar dinámicamente sistemas en los que se necesite validar si se satisfacen los requisitos de Fiabilidad. Gracias a la validación flexible del Sistema Bajo Prueba se pueden probar los mecanismos de tolerancia a defectos y se puede verificar la correcta gestión de los defectos de acuerdo a su especificación. La arquitectura hardware/software de la herramienta Exhaustif ha sido diseñada para reducir al máximo el impacto de cualquier cambio (inclusión, modificación o borrado de cualquier módulo).

Bacterio [27]: Es una herramienta de escritorio construida en Java para testing de mutación. Implementa la mayoría de las técnicas de mutación disponibles en la literatura. Resulta bastante adecuada para realizar mutaciones a nivel de sistema y de integración.

Anexo X: Descripción de MantisBT

Objetivo

En este anexo se describen las principales funcionalidades que brinda la herramienta MantisBT, utilizada para reportar incidentes en el sistema Test School.

Descripción

MantisBT es una aplicación de software libre multiplataforma que permite gestionar las incidencias en una empresa, sistema o proyecto. Presenta como principales beneficios la facilidad de uso, adaptabilidad a distintos escenarios y plugins que permiten aumentar la capacidad de trabajo de la herramienta. [49]

Sus principales funcionalidades son:

- Reporte de incidentes:
    - permite a los usuarios crear historias de cualquier tipo (bugs, mejoras, peticiones de soporte) añadiendo título, descripción y detalles técnicos. Dentro de los detalles técnicos se destacan:
        - Categoría
        - Prioridad
        - Criticidad
        - Estado
        - Imágenes
    - permite asignar un responsable al reporte para que la incidencia sea atendida.
    - permite mantener un hilo de conversación entre el tester que reportó la incidencia y el/los desarrolladores responsables asignados a la corrección.
- Sistema de permisos de usuario: permite identificar distintos tipos de usuarios que acceden al sistema. Existen roles específicos para cada uno de ellos según las acciones que pueden realizar y solo el administrador puede configurar estos permisos:
    - espectador
    - informador
    - actualizador
    - desarrollador
    - manager
    - administrador
- Notificaciones de usuario: permite a los usuarios involucrados en un reporte recibir notificaciones mediante correo electrónico para mantenerse al tanto de las novedades.
- Flexibilidad para personalizar el sistema: permite definir nuevos estados para los defectos detectados, generar nuevos campos o personalizar los permisos de los distintos roles de

usuario. También permite gestionar las etiquetas y organizar la información por proyectos, públicos o privados, generando a su vez subproyectos o categorías dentro de ellos.
- Generar informes con las estadísticas del trabajo.

Características [48]:

- Aplicación multiplataforma basada en PHP y con soporte para bases de datos MySQL, PostgreSQL y MS SQL.
- Puede instalarse en cualquier servidor web con PHP y con alguna de las bases de datos descritas.
- A nivel del cliente puede accederse desde cualquier plataforma o sistema operativo con conexión de red y un navegador web.
- Se define como software libre, con libertad de uso, modificación y distribución.
- Puede descargarse de forma gratuita y están disponibles además las versiones inestables de desarrollo.
- Se distribuye en código fuente con un manual de instalación y está disponible en los repositorios de algunas distribuciones Debian, Ubuntu, Gentoo, Fedora, FreeBSD, Sun Solaris y Frugalware.
- El equipo de Mantis ofrece servicios de soporte y consultoría.