UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Matefun Colaborativo

Informe de Proyecto de Grado presentado por

Diego Javier Rodríguez Uranga

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

Supervisores

Marcos Viera
Gonzalo Tejera

Montevideo, 4 de septiembre de 2025

Agradecimientos

En primer lugar quiero agradecer a mi Madre cuyas enseñanzas hasta el día de hoy me impulsaron para terminar una carrera universitaria. A mi Padre, quien me ha apoyado todos estos años cada vez que elegía cambiar de carrera y comenzar una nueva. Y a mi Hermano, quien me introdujo al mundo de las computadoras cuando era chico. Agradezco de todo corazón a mis amigas Agus y Manu, nuestra amistad y juntadas semanales fueron un apoyo invaluable todos estos años sin el cual no habría podido perseverar. Agradezco también a Vane quien me prestó la computadora para poder ejecutar el binario de Matefun sin la cual no habría podido hacer este trabajo. Y último pero no menos importante a mis tutores Marcos y Gonzalo, por toda su paciencia y guía estos años.

Resumen

La enseñanza de matemáticas y programación en la educación media cumple un rol fundamental en la formación de habilidades lógico-matemáticas, pensamiento abstracto y resolución de problemas. Estas competencias son esenciales no solo para carreras científicas o tecnológicas, sino también para el desarrollo de un pensamiento crítico en la ciudadanía en general. En particular, el uso de lenguajes funcionales en contextos educativos ha demostrado ser una herramienta poderosa para introducir conceptos de programación de manera estructurada y alineada con contenidos matemáticos.

En este proyecto de grado se trabajó en el desarrollo y extensión de funcionalidades para la plataforma educativa Matefun, una herramienta web orientada al aprendizaje del lenguaje funcional homónimo utilizado en educación media en Uruguay. El objetivo principal fue dotar al sistema de capacidades de edición colaborativa en tiempo real y mejorar la accesibilidad general del sitio mediante la implementación de un sistema de usuarios con autogestión.

Para lograrlo, se analizaron distintos enfoques y tecnologías, incluyendo plataformas Learning Management System, editores colaborativos y técnicas como Operational Transformation y Conflict-free Replicated Data Types (CRDT), optando finalmente por una implementación basada en CRDT. También se rediseñó la arquitectura del sitio incorporando nuevos componentes en la capa de negocios, y una capa de persistencia temporal.

El sistema ahora permite que múltiples usuarios editen simultáneamente archivos compartidos, visualizando en tiempo real los cambios y posiciones del cursor de otros usuarios. Asimismo, se incorporó la creación de cuentas de usuario, grupos y subgrupos, con capacidades de administración y asignación de tareas. Las funcionalidades fueron diseñadas para fomentar el trabajo colaborativo y mejorar la experiencia del usuario, tanto en entornos educativos como en posibles usos más amplios.

Se concluye que la incorporación de estas mejoras hace que Matefun sea una herramienta más moderna, accesible, y con una experiencia de usuario más robusta, facilitando su adopción en aulas y más allá. Finalmente, se presentan propuestas de trabajo futuro para continuar con su evolución.

Palabras clave: Educación, Matemáticas, Programación, Edición colaborativa

Índice general

Capítulo 1

Introducción

La enseñanza de matemáticas y programación en la educación media constituye un pilar esencial en la formación de estudiantes, dado que promueve habilidades de razonamiento lógico, resolución de problemas y pensamiento abstracto. Estas competencias resultan fundamentales en un mundo crecientemente orientado hacia lo digital, y su desarrollo temprano tiene un impacto directo tanto en la inserción académica como en la preparación para entornos laborales complejos. En particular, los lenguajes de programación funcionales ofrecen una forma estructurada y matemática de pensar los problemas, por lo que resultan especialmente adecuados como herramienta educativa en este nivel.

En este contexto, Matefun surge como una plataforma web destinada al aprendizaje del lenguaje funcional homónimo, diseñada específicamente para su uso en la educación media en Uruguay. No obstante, las versiones previas de Matefun presentaban limitaciones importantes en términos de colaboración, accesibilidad y administración de usuarios. Estas restricciones reducían su utilidad en escenarios reales de aula y limitaban su potencial como herramienta flexible y moderna para el trabajo conjunto.

1.1. Motivación

El creciente uso de herramientas digitales en la educación evidencia una necesidad urgente de plataformas que no solo permitan la práctica individual, sino que fomenten el trabajo colaborativo y la interacción entre estudiantes y docentes. Las dinámicas educativas actuales exigen espacios virtuales que permitan compartir recursos, editar documentos en conjunto, gestionar tareas y organizar grupos de trabajo.

A pesar del valor pedagógico que Matefun aporta, su estructura original carecía de funcionalidades esenciales como la edición colaborativa en tiempo real, la creación de cuentas de usuario, o la administración de grupos y permisos. Esta carencia motivó la realización del presente proyecto de grado, que busca dotar a la plataforma de nuevas capacidades orientadas a mejorar la experiencia

del usuario, facilitar el trabajo en grupo y ampliar su aplicabilidad tanto en el aula como en contextos más generales.

1.2. Objetivos

El objetivo general de este proyecto es **extender las funcionalidades de Matefun** para convertirla en una plataforma colaborativa, moderna y más accesible, manteniendo su enfoque educativo.

Para ello se plantean los siguientes **objetivos específicos**:

- Implementar un sistema de edición colaborativa en tiempo real utilizando tecnologías adecuadas para sincronización entre múltiples usuarios.
- Incorporar un sistema de cuentas de usuario con autogestión, que permita iniciar sesión, registrarse y trabajar de manera personalizada.
- Diseñar e implementar funcionalidades de creación y gestión de grupos y subgrupos, que permitan asignar archivos y coordinar tareas colaborativas.

1.3. Estructura del documento

A continuación se describe la estructura de este documento:

1. Introducción: Aquí se introduce la motivación y objetivos del proyecto actual.
2. Requerimientos: En esta sección se presentan los requerimientos funcionales y no funcionales del proyecto.
3. Revisión de antecedentes: En esta sección se presentan plataformas estudiadas y tecnologías que resultarán relevantes para el resto del documento.
4. Parte Central: En esta sección se describen los aportes principales del proyecto. La arquitectura que se creó, las tecnologías que se decidieron utilizar, la implementación que se llevó a cabo, y consideraciones que se tomaron concernientes a la seguridad del sitio.
5. Conclusiones y Trabajo Futuro: Aquí se presenta las conclusioens del proyecto, y se listan una serie de ideas de posibles trabajos para futuras ediciones del sitio.

Capítulo 2

Requerimientos

2.1. Requerimientos funcionales

En esta sección listamos los requerimientos funcionales que debe contar las funcionalidades del sistema. Los mismos parten de los requerimientos ya existentes en ediciones anteriores, y se fueron construyendo en reuniones con los tutores del proyecto en base a las necesidades del sitio por experiencias que han habido en talleres donde se ha usado el sitio Matefun.

2.1.1. Requerimientos funcionales de ediciones previas

Las versiones anteriores de Matefun contaban con una serie de funcionalidades orientadas a facilitar la edición, ejecución e interacción con programas escritos en el lenguaje Matefun. Entre los elementos centrales se encontraba un editor de código, la posibilidad de compilar y ejecutar programas, mostrar errores y advertencias del compilador, y utilizar un intérprete interactivo configurable.

En el aspecto visual, el sistema permitía graficar funciones matemáticas, visualizar y animar secuencias de figuras. Además, los gráficos podían exportarse como imágenes.

Otras funcionalidades destacadas incluían la gestión de archivos mediante un repositorio con soporte para creación, organización y eliminación de archivos y directorios.

En el Apéndice C se incluye una lista más completa y detallada de estos requerimientos.

2.1.2. Nuevos requerimientos funcionales

En esta sección se presentan los nuevos requerimientos funcionales correspondientes a la nueva versión de la plataforma. Los mismos están centrados en facilitar la colaboración entre usuarios del sitio, y la administración de usuarios y grupos.

- **Creación de cuenta:** Debe ser posible crear un usuario fácilmente utilizando tan solo un email y una contraseña.
- **Cuenta de invitado:** Debe ser posible para un usuario probar el sitio con una cuenta de invitado, sin necesidad de ingresar email ni contraseña. Las cuentas de invitado deben borrarse cuando el usuario hace sign out, y también debe haber una forma fácil para hacer que se borren automáticamente cada cierto período de tiempo.
- **Compartir archivos:** Los usuarios deben poder compartir archivos con otros usuarios para editarlos en simultáneo. Deben poder hacerlo tanto desde el editor como desde la vista de archivos.
- **Administrar usuarios de un archivo:** Un usuario con los permisos requeridos sobre un archivo, debe poder administrar los permisos de otros usuarios sobre el archivo, así como agregar o quitar usuarios al mismo.
- **Edición Colaborativa:** Estando en el editor con un archivo que hayan previamente compartido, deben poder ver en vivo las ediciones de los otros usuarios.
- **Posición del cursor de otros usuarios:** Cuando dos o más usuarios estén editando el mismo archivo a la vez, cada usuario debe poder ver dónde está el cursor de los demás usuarios, con un indicador de a qué usuario pertenece.
- **Guardado automático:** Los documentos que ya han sido guardados al menos una vez, se deben guardar de manera automática cada una cierta cantidad de tiempo.
- **Creación de nuevo programa:** Mientras un usuario está editando un programa, debe poder acceder desde el editor de texto a un botón para crear un documento en blanco, el cual tomará el lugar del documento que está editando actualmente en el editor.
- **Grupos:** Un usuario debe ser capaz de crear un grupo, e invitar a otros usuarios al mismo.
- **Subgrupos:** Un usuario con permiso dentro de un grupo debe ser capaz de crear subgrupos dentro del mismo, formados por usuarios del grupo.
- **Creación de archivos dentro del grupo:** Un usuario dentro de un grupo debe ser capaz de crear archivos dentro del grupo.
- **Visualización de archivos de grupo:** Un usuario de un grupo que posea archivos en el mismo, debe ser capaz de visualizar los mismos en una vista tipo directorio.
- **Visualización de usuarios de un grupo:** Un usuario que pertenece a un grupo debe ser capaz de ver una lista de los demás usuarios que pertenecen al mismo grupo.

- **Asignar archivos:** Un usuario con permisos debe ser capaz de asignar archivos que posea dentro del grupo a otros usuarios o a subgrupos del mismo grupo.
- **Pedir retroalimentación:** Un usuario de un grupo puede pedir retroalimentación de cualquier archivo que posea dentro de ese grupo.
- **Devolver retroalimentación:** Un usuario con permisos de un grupo debe poder ver y devolver la retroalimentación que han pedido otros usuarios del grupo.
- **Administrar usuarios de grupo:** Un usuario con permisos de un grupo debe poder administrar los permisos de otros usuarios del grupo, así como agregar o quitar usuarios al mismo.
- **Admin:** Se pueden crear usuarios Administradores, que podrán ver los datos de los demás usuarios del sitio.

2.2. Requerimientos no funcionales

Estos no han cambiado de ediciones anteriores del sitio, se copian aquí las que se encuentran en el Documento de proyecto de grado original hecho por Gonzalo Cameto (Accessed: 2025-06-10a)

- **Tiempo de reacción del intérprete:** El intérprete interactivo debe ser rápido en la ejecución de los comandos. Se establece una cota de un segundo para que el servidor reciba y ejecute el comando ingresado por el usuario, medido luego de que se presionó enter. Si bien el tiempo de respuesta depende de la complejidad del cálculo de lo que se ejecuta y esto no puede ser acotado, sí podemos acotar el tiempo de espera en el cliente para recibir la confirmación del servidor de que éste recibió el comando y lo está procesando.
- **Modularización:** El sistema debe ser construido en módulos de forma tal que las tareas de extensión y mantenimiento puedan ser llevadas a cabo de manera accesible. En particular este proyecto podrá ser continuado por otros proyectos de grado, mejorando determinados aspectos del sistema. Por este motivo es un requisito mantener bajo el nivel de acoplamiento entre los componentes y contar con interfaces bien definidas.
- **Usabilidad:** El sistema debe permitir un fácil aprendizaje de las funcionalidades de la interfaz de usuario, de forma tal que esto no interfiera con el estudio por parte del estudiante del lenguaje Matefun. Esto se realizará manteniendo una interfaz limpia, que no se encuentre sobrecargada con textos, pero que muestre más información de las distintas funcionalidades al posicionar el cursor sobre los botones que las disparan.

- **Disponibilidad:** El sistema debe permitir su utilización por múltiples usuarios en simultáneo. Debe estar asegurado un desenvolvimiento adecuado para una carga de 30 usuarios en simultáneo.

Capítulo 3

Revisión de antecedentes

En este capítulo describiremos las nuevas tecnologías utilizadas, así como los Sistemas de Administración de Aprendizaje (LMS por sus siglas en Inglés) estudiados para las nuevas funcionalidades de la app y dos tecnologías distintas para lograr edición colaborativa.

3.1. Plataformas estudiadas

Para este proyecto se estudiaron varias plataformas distintas. Se buscaron plataformas que hubieran solucionado al menos una de los problemas que se buscan solucionar en el proyecto actual. Entre ellas se estudiaron algunos LMS como ser Blackboard Learn, Canvas LMS, y Moodle, los sitios de edición de código online para programadores Codeshare y CodeSandbox, y dos sitios que tienen sistemas para compartir archivos Overleaf y Google Drive.

3.1.1. LMS

Un LMS es una aplicación de software hecha para la administración, documentación, seguimiento, reporte, automatización, y entrega de cursos educativos, programas de entrenamiento, materiales, o programas de aprendizaje y desarrollo (Ellis, 2009).

Los LMS se estudiaron para ver cómo funciona el manejo de grupos y tareas en los mismos. Se hizo este estudio ya que si bien en esta nueva versión de Matefun buscamos generalizar el sitio para que pueda ser utilizado por cualquier persona sin necesidad de pertenecer a una institución, y que se puedan generar grupos también con otros propósitos si así lo desean, se sigue queriendo mantener el uso en el aula como uno de los principales focos de la aplicación.

Se eligieron estos LMS basado en que son los más usados y premiados LMS del mercado contemporáneo (*EdTech Breaktrhough Award Winners*, s.f.)(*Market Progression and Leaders*, s.f.)(Watkins, s.f.)(Hill, s.f.).

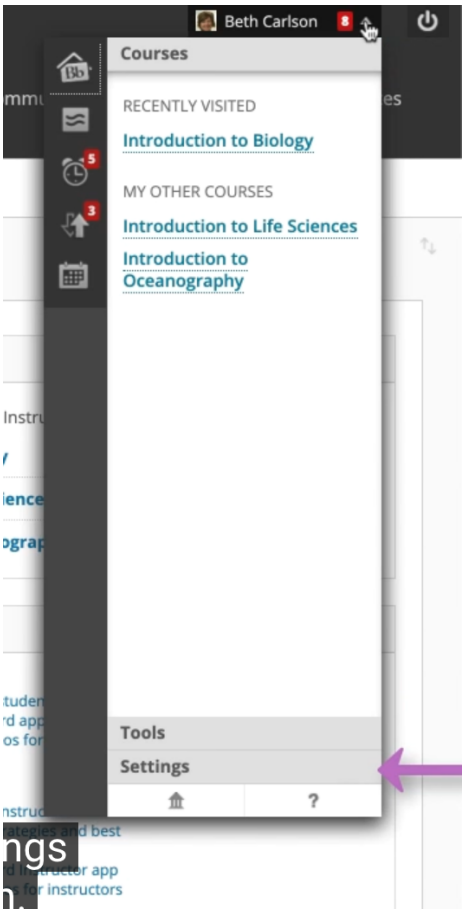


Figura 3.1: Sección de recientemente visitados en la barra superior de Blackboard Learn.

3.1.1.1. Blackboard Learn

Para estudiar Blackboard Learn[1] se estudió la sección tutorial que tiene en su web[2]. Todas las imágenes usadas aquí fueron tomadas de allí.

En todos los casos los LMS tratan con Cursos y dentro de los cursos Grupos.

Se presentan a continuación ideas de la plataforma que resultaron llamativas para Matefun, si bien muchas de ellas no corresponden al proyecto actual, sirven como referencia para futuras ediciones.

- Posee la posibilidad de tener tests con preguntas que se corrigen solas, y tener feedback automático para las mismas si se falla o no. Esto podría ser-

[1] https://www.anthology.com/products/teaching-and-learning/learning-effectiveness/blackboard

[2] https://help.blackboard.com/Learn/Instructor

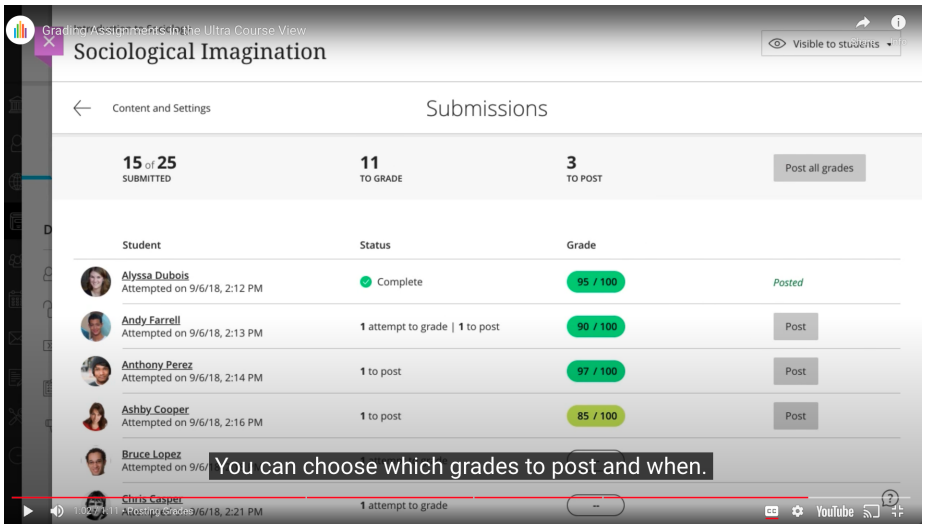


Figura 3.2: Página de notas como la ve un profesor en Blackboard Learn. En Blackboard Learn se puede hacer públicas las notas de una tarea ya sea para todo un grupo a la vez o manualmente uno a uno

vir si se desea hacer un archivo que termine haciendo una salida esperada en Matefun, ya que podría hacer más ágil la correción y posiblemente ayudar con mensajes a alumnos que estén encontrando dificultad en resolver el ejercicio.

- Tiene una sección donde se muestran todas las tareas que un profesor tiene para corregir, lo cual ayudaría a hacer más fácil sobre todo el manejo de varios subgrupos a la vez.

- Tiene notificaciones por email cuando a un alumno le asignan algo, cuando está por llegar una fecha de entrega, si pasó la fecha de entrega, si recibió corrección, etc.

- Como se puede apreciar en la Figura 3.1, tiene un dropdown en la parte superior de la pantalla con una sección de “Recientemente visitados” para hacer más fácil el acceso a los subgrupos y grupos con los que el usuario ha interactuado más recientemente.

- Como se puede ver en la Figura 3.2, es posible trabajar con un sistema de notas/puntajes para las tareas. Si se trabaja con notas, se puede ver una lista de las mismas, y se puede trabajar en ellas en forma de borrador y luego publicarlas, ya sea de a una o todas juntas

- Los usuarios pueden marcar grupos como favoritos para que aparezcan arriba del todo en la lista de grupos.

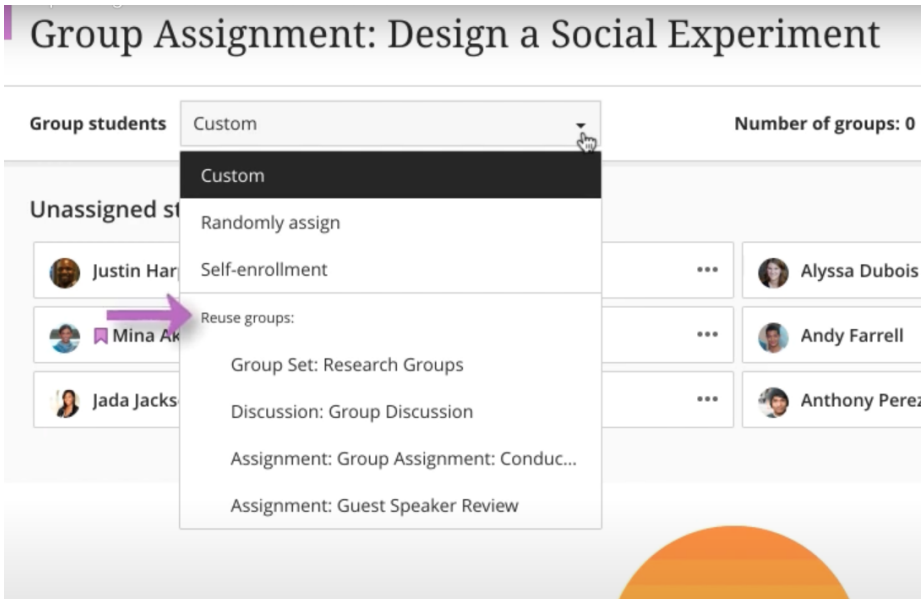


Figura 3.3: Menú para creación de grupos en Blackboard Learn. Al crear una nueva tarea en Blackboard Learn, se pueden crear nuevos subgrupos o reusar subgrupos anteriores.

- Tienen la opción de cambiar los roles de los miembros de un grupo, así como la opción de denegarles el acceso y removerlos del mismo.
- Posee carpetas personales para los usuarios, que no están asociadas a ningún grupo, y carpetas específicas para cada grupo.
- Las carpetas de los grupos sólo pueden ser vistas por profesores por defecto, pero los mismos pueden crear subcarpetas y darle permiso a los alumnos para verlas.
- Tiene versionado para los archivos, donde se puede ver qué usuario hizo los cambios al archivo, qué cambios hizo, y en qué fecha.
- Tiene marcadores para guardar carpetas y archivos específicos para que sean de fácil acceso.
- Hay forma que varios usuarios evalúen la misma tarea sin ver las evaluaciones que otros usuarios le pusieron. Hay un rol específico de Instructor que luego debe ver estas distintas evaluaciones y las reconcilie en una única evaluación final.
- Puede chequear si alguna tarea tiene plagio.
- Como se puede ver en la Figura 3.3, para cada tarea tiene la opción de crear nuevos subgrupos o de asignar subgrupos ya usados nuevamente.

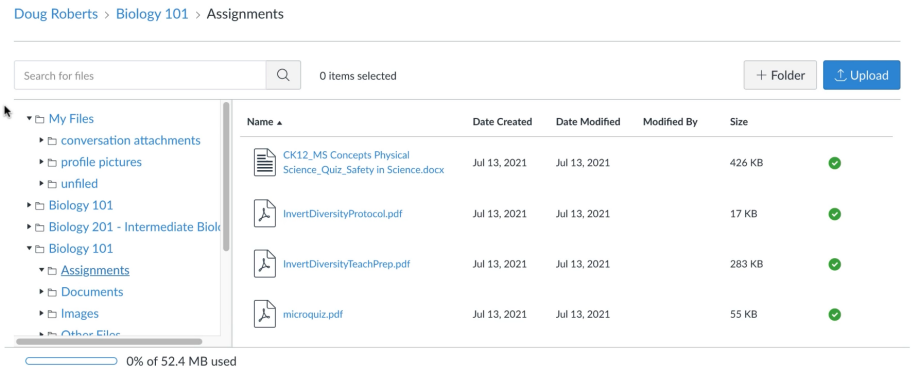


Figura 3.4: Una vista en CanvasLMS de su estructura de carpetas. Como se ve en la raíz de la estructura hay una carpeta para los archivos personales del usuario y luego hay una carpeta extra por cada grupo al que pertenece.

- En vez de sólo tener roles fijos, tiene una serie de permisos y se pueden crear roles custom eligiendo qué permisos se le quiere dar al rol.

De estas características se destacan para la edición actual de Matefun el tener carpetas personales para usuarios y carpetas separadas para los grupos, la capacidad de tener roles en grupos que puedan controlar los roles de otros usuarios y eliminarlos del grupo de ser necesario, y la capacidad de reutilizar subgrupos para distintas tareas. Este proyecto no se centra en la corrección de asignaciones con nota así que las características se dejan aquí sólo para referencia a futuro como posibles mejoras a agregar.

3.1.1.2. Canvas LMS

Se hizo un estudio similar al anterior para Canvas LMS[3], estudiando las guías que se encuentran en su web[4]. Todas las imágenes que aprecen en este documento fueron sacadas de dichas guías.

Las características pertinentes a nuestro caso son:

- Tiene forma de acceder a entregas filtrando por usuario para ver todas las entregas de un usuario, así como filtrando por tarea para ver todos los usuarios que entregaron una tarea dada.

- Las asignaciones se pueden hacer en el momento o programarlas para que sean publicadas en un momento futuro.

[3] https://www.instructure.com/canvas
[4] https://community.canvaslms.com/t5/Canvas-LMS/ct-p/canvaslms

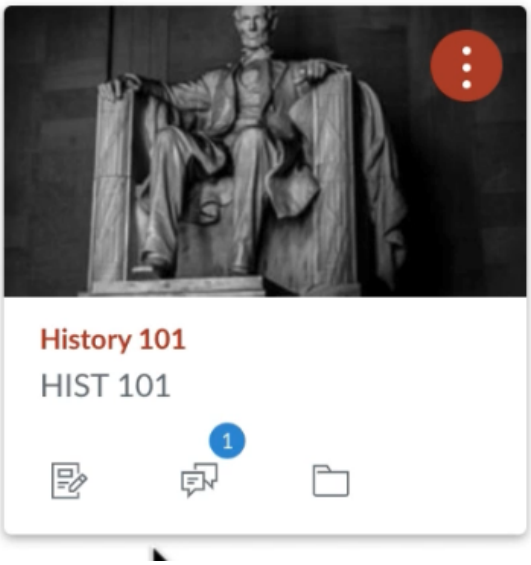


Figura 3.5: Un grupo de CanvasLMS con un indicador de notificación mostrando que hay un mensaje por leer en el grupo.

- Tienen una página centrada de archivos, donde tienen una carpeta con sus archivos personales y aparte una carpeta por grupo al que pertenecen. La misma se puede ver en la Figura 3.4.
- En la sección de grupos posee un indicador de notificaciones si hay algo nuevo en el grupo como se puede ver en la Figura 3.5.
- Posee una sección de actividad reciente como se ve en la Figura 3.6.
- Se pueden agregar usuarios por email, o por distintas ids ingresándolas en un campo de texto, y eligiendo el rol y el grupo al cual serán agregados en un menú, como se puede ver en la Figura 3.7.
- Se pueden crear conjuntos de subgrupos automáticamente como se puede ver en la Figura 3.8. Esto sirve por ejemplo para subdividir a un grupo en subgrupos disjuntos para una tarea.
- Posee distintos tipos de roles para los usuarios, como un rol de observador para los padres para que puedan ver lo que hacen los hijos pero no editar nada.

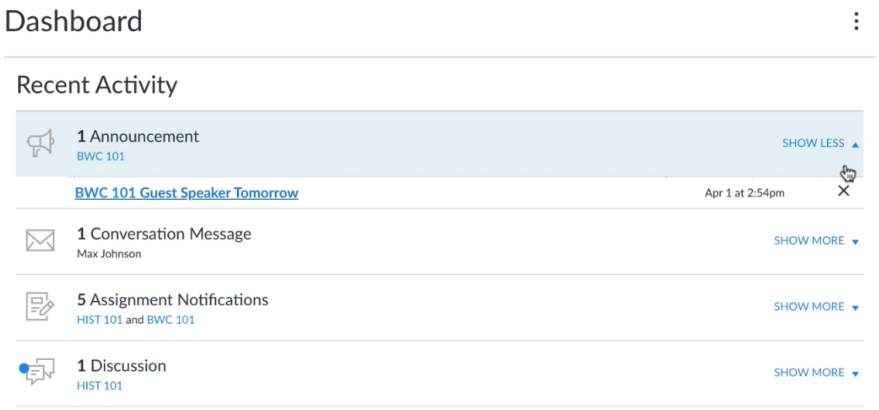


Figura 3.6: Vista de actividad reciente en CanvasLMS.

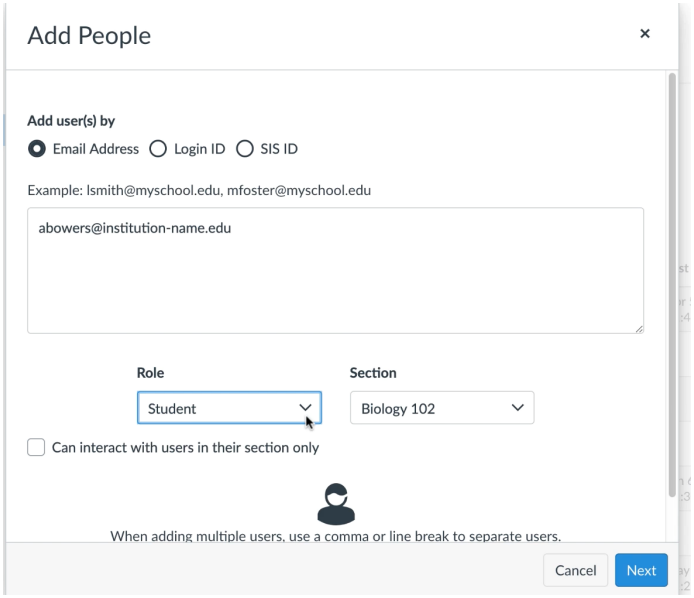


Figura 3.7: Menú para agregar usuarios a un grupo en CanvasLMS.

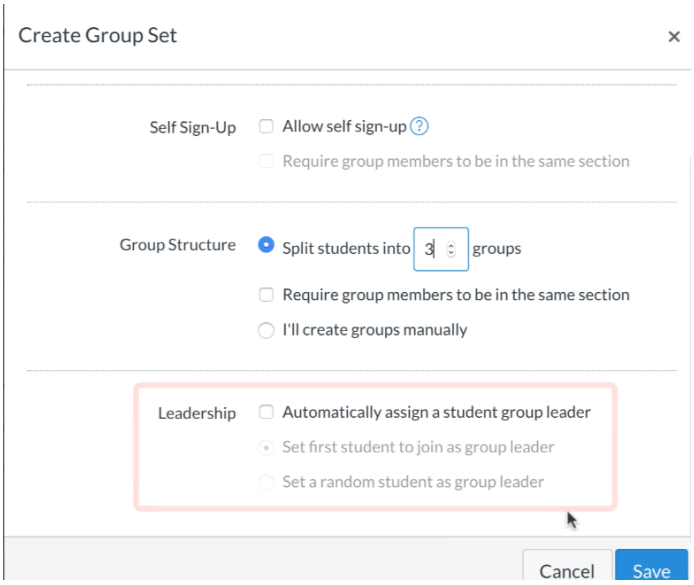


Figura 3.8: Creación de un conjunto de subgrupos en CanvasLMS

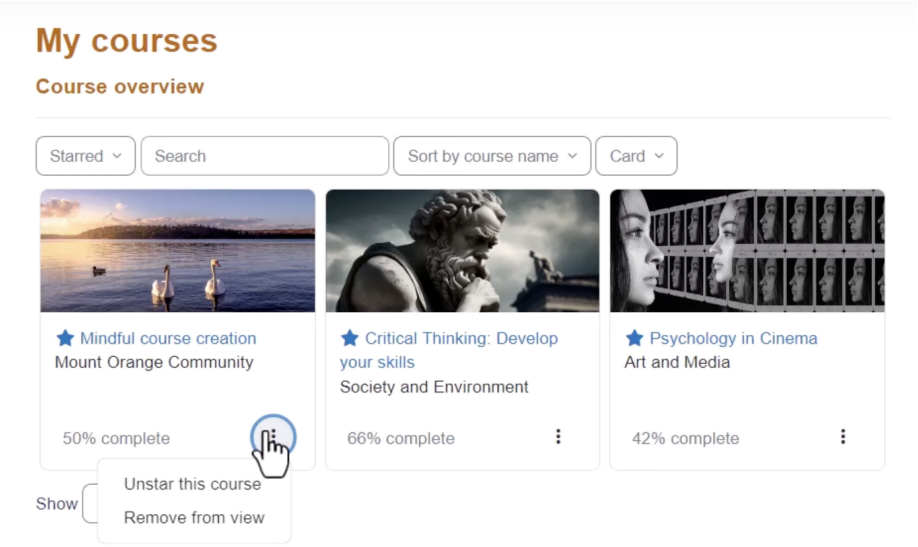


Figura 3.9: Vista de grupos marcados como favoritos en Moodle

3.1.1.3. Moodle

Para Moodle[5] se estudiaron los cursos en vídeo que ellos mismos proporcionan en su web[6].

Las características que son importantes para este proyecto son:

- Los usuarios no tienen roles por defecto al crear una cuenta en el sitio, sino que se les da un rol específico en cada grupo al que se unen. De esta forma un usuario puede ser docente en un grupo y alumno en otro grupo.
- Al agregar usuarios a un grupo se puede elegir con qué rol se les agrega. Sólo un Administrador puede agregar otros usuarios como Docentes por defecto, si bien esto puede ser configurado.
- Se pueden agregar usuarios a un grupo de forma masiva con un archivo CSV.
- Se pueden crear varios subgrupos en un grupo, ya sea con un archivo CSV o manualmente.
- Se puede agregar un grupo como favorito y filtrar los grupos por favoritos como se ve en la Figura 3.9
- Los grupos se pueden ordenar alfabéticamente o por el accedido más recientemente.

[5] https://moodle.org/
[6] https://moodle.academy/

- Todos los usuarios tienen una página con archivos privados, los cuales pueden luego ser compartidos en distintos grupos.
- Se pueden elegir no ver la identidad del alumno al corregir tareas.
- Las tareas tienen opcionalmente una descripción e instrucciones.
- Se le puede poner un tiempo límite a las tareas. Los alumnos pueden igualmente entregar la tarea luego de que se acaba el tiempo, pero esto aparecerá como una nota en la entrega.
- Al corregir tareas se pueden agregar comentarios directamente en el texto de la tarea.
- Los profesores pueden activar notificaciones para que sean avisados cuando un alumno entrega una tarea, y pueden activar notificaciones para que los alumnos sean notificados cuando sus tareas han sido corregidas.
- Moodle cumple con las pautas de accesibilidad AA puestas por la Web Accessibility Initiative[7][8]

Para este proyecto es de interés cómo administran los roles, que cada usuario no tenga roles específicos en todo el sitio sino que puedan ser distintos grupo a grupo, y que puedan haber varios roles distintos dentro de un mismo grupo.

3.1.1.4. Conclusión LMS

Los LMS son sistemas que abarcan mucho más de lo que Matefun está diseñado para abarcar. Aún más, un LMS sería un buen compañero para usar con Matefun si alguien desea hacer cursos para poder organizar mejor la información con funcionalidades que Matefun no está hecho para soportar.

De mayor interés en estos son cómo manejan los roles en el sitio y en los grupos, y las maneras en que agregan usuarios a los grupos y subgrupos, y cómo manejan los archivos. Tener usuarios sin rol específico en el sitio, y que cuando se crea un grupo se puedan agregar usuarios eligiendo sus roles, y que hayan distintos niveles de roles son ideas que se tomaron de los LMS. También otras ideas que quedaron por fuera del alcance de este proyecto, pero quedan anotadas en el Capítulo 5 para su posterior consideración como mejoras al sitio.

3.1.2. Sitios web para programadores

Se estudiaron algunos sitios web con editor de texto para código y que tenían capacidad para compartir los archivos con el fin de ver cómo manejaban este aspecto. Si bien en estas páginas a diferencia de los LMS no existe el concepto

[7] https://moodle.com/functionality-with-moodle/moodle-accessibility/
[8] https://www.w3.org/WAI/WCAG2AA-Conformance

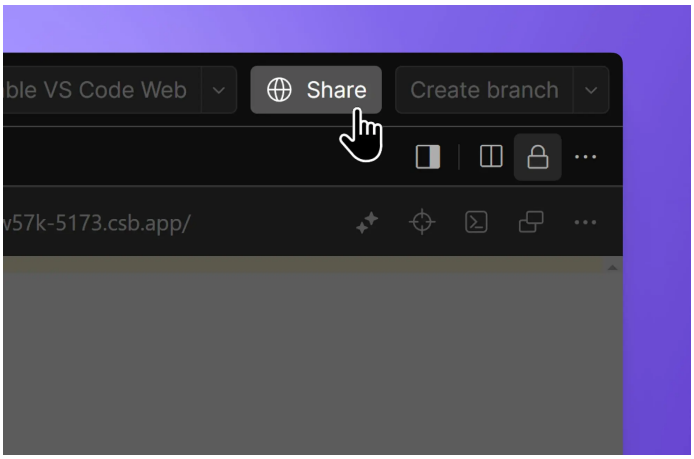


Figura 3.10: Opción para compartir código en CodeSandbox. En CodeSandbox es posible compartir código consiguiendo un enlace una vez se apreta el botón Share.

de grupos, se quería ver el enfoque en un sitio hecho únicamente para compartir código. Los sitios estudiados fueron codeshare[9], codePen[10], y codesandbox[11].

3.1.2.1. CodeSandbox

Como se ve en la Figura 3.10, en CodeSandbox es posible compartir un archivo obteniendo un enlace cuando se hace click en el botón de compartir. Una vez se comparte el enlace y alguien entra al mismo es posible ver su cursor mientras se edita en vivo y compartir también la terminal del sitio web, viendo los comandos que escribe y sus respuestas. Todas las imágenes fueron sacadas de su documentación web[12]

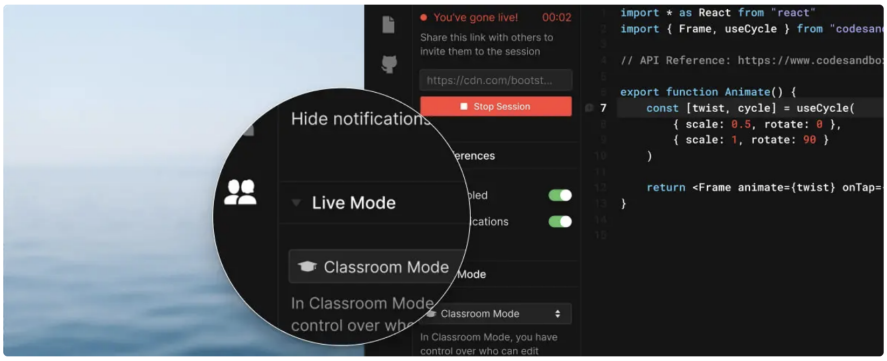
También existe un modo llamado *Classroom Mode* (Figura 3.11) donde es posible compartir la vista del editor y la terminal con otros usuarios que pueden ver todo lo que uno hace pero no interactuar con nada, a menos que se les de permiso para ello.

3.1.2.2. Codeshare

Codeshare es el más sencillo de todos los sitios de programación que se estudiaron, consta de tan solo una ventana de editor, y para compartir la misma basta con copiar la url y compartirla con otra gente. Cuando se accede a la misma en otro explorador se accede al mismo editor, donde ambos usuarios pueden editar el texto a la vez y ver en vivo los cambios que hace el otro.

[9] https://codeshare.io
[10] https://codepen.io
[11] https://codesandbox.io
[12] https://codesandbox.io/docs/learn

You can enable Classroom Mode from the Live Mode drop-down menu.



You can give someone editor rights by pressing the '+' icon next to their name, or make them a viewer by clicking the '-' icon next to their name.

Figura 3.11: Opción para habilitar el modo aula en CodeSandbox.

Es posible desde el menú también obtener una url con modo sólo lectura si es deseado.

No tienen documentación sobre qué tecnologías usan para edición colaborativa, pero en comunicación directa con los desarrolaldores del sitio, nos compartieron que utilizan un algoritmo de la familia de Operational Transformation para dicha tarea, pero que hoy en día si pudieran empezar de cero elegirían hacerlo con un algoritmo de la familia de Conflict-free Replicated Data Type. Estos tipos de algoritmos se describen más adelante en la Sección 3.5.

3.1.2.3. CodePen

En este sitio para poder compartir el código se debe tener una cuenta, y haber guardado el documento al menos una vez manualmente. Luego simplemente se copia la url, cualquiera con la url puede ver lo último que ha sido guardado, pero no existe edición en vivo ni colaborativa. Si se hacen cambios los demás usuarios tienen que recargar la página para verlos, y si los usuarios con quienes un documento ha sido compartido quieren hacer cambios y guardarlos se genera una copia propia para ellos, nunca cambian el archivo original.

3.1.2.4. Conclusión sitios web para programadores

Los sitios estudiados tienen bastante menos posibildades en cuanto a roles que los LMS, pero todos tienen la facilidad de compartir por enlace, algunos incluso si tener que crear una cuenta previamente. El modo aula de CodeSandbox es de particular interés para ciertas situaciones, si bien es posible emularlo con un proyector si es una clase presencial, o compartiendo pantalla si es una clase

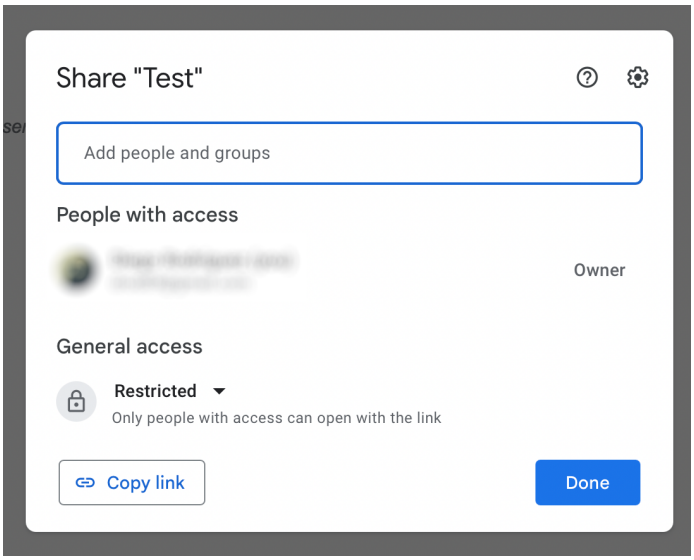


Figura 3.12: Menú para compartir un documento en Google Docs. Se puede compartir enlace para gente que ya tenga permiso.

online.

3.1.3. Sitios web con edición colaborativa de documentos

Para esta sección se estudió cómo funcionan los mecanismos para compartir documentos de Google Docs[13] y Overleaf[14].

3.1.3.1. Google Drive

Se pueden compartir un enlace para acceder al documento. Por defecto el enlace sólo funciona para gente que ya tenga permiso para ver el documento (Figura 3.12), pero se puede habilitar para que cualquier persona con el enlace pueda obtener acceso al documento, e incluso elegir qué permisos tendrán dichas personas (Figura 3.13). Una característica de este enlace sin embargo es que sólo existe un enlace, por lo cual si se comparte un enlace con permisos de solo lectura y luego se le cambian los permisos al enlace las personas que tienen el enlace anterior obtendrán los nuevos permisos.

Por otro lado, si en vez de esto se escribe un email en la parte superior del menú de compartir que se ve en la Figura 3.12, automáticamente el menú cambiará para incluir una opción para notificar a las personas agregadas y mandarles un mensaje en la notificación. También se puede en este caso elegir el rol

[13] https://docs.google.com
[14] https://www.overleaf.com/

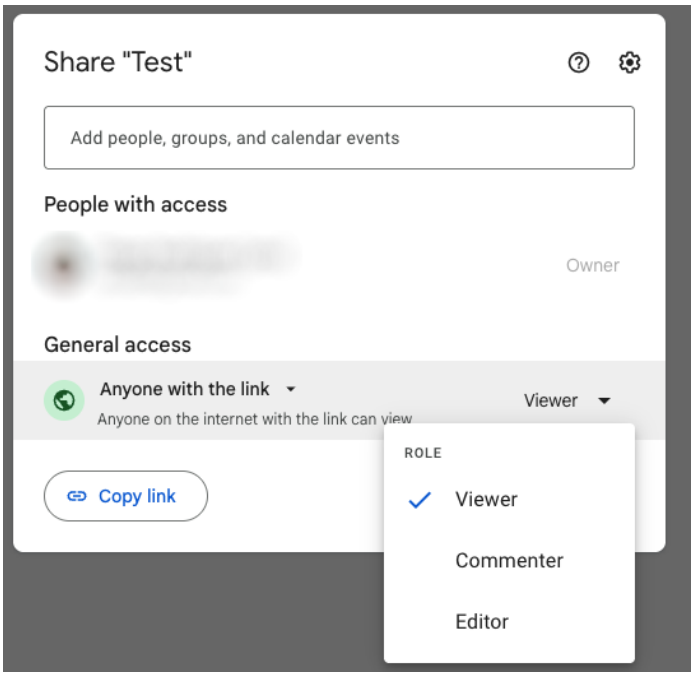


Figura 3.13: Menú para compartir un documento en Google Docs. También se puede compartir un enlace para que cualquier persona con el enlace pueda acceder al sitio.

de todos los agregados de esta forma. Este menú se puede ver en la Figura 3.14. Los roles posibles y sus permisos básicos son:

- **Viewer:** Este rol sólo puede ver el documento, no puede hacer ni recomendar cambios sobre el mismo.
- **Commenter:** Este rol puede acceder a los documentos, dejar comentarios sobre el mismo, y sugerir cambios.
- **Editor:** Este rol puede editar todo aspecto del documento.

También es posible desde la vista de archivos elegir varios archivos a la vez presionando shift y compartir todos a la vez.

3.1.3.1.1 Edición colaborativa

Google Doc utiliza una versión de Operational Transformation para su edición colaborativa. Un ejemplo del algoritmo se encuentra en la Sección 3.5.1

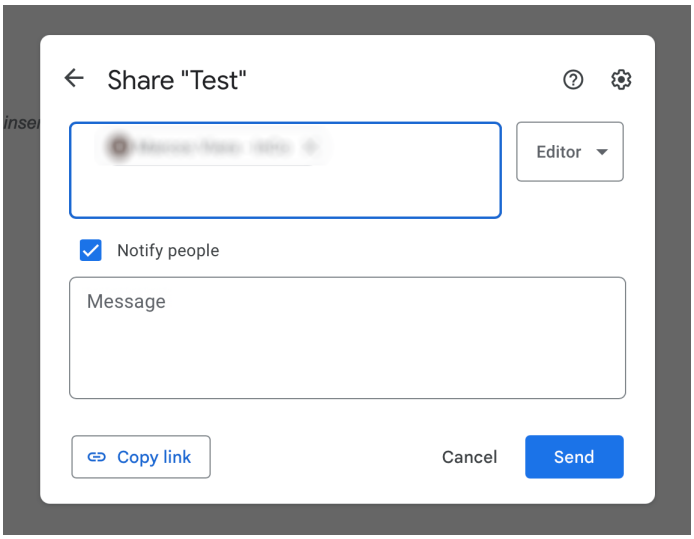


Figura 3.14: Menú para compartir un documento en Google Docs. Si se escriben mails en el campo correspondiente aparece un área para escribir un mensaje.

3.1.3.2. Overleaf

Overleaf tiene formas muy similares de compartir que Google Docs. Se puede activar el compartir por enlace, en cuyo caso automáticamente aparecen dos enlaces, uno para sólo lectura y otro para dar permisos de edición como se puede ver en la Figura 3.15.

También al igual que Google Docs ofrece la posibilidad de compartir ingresando una lista de email y eligiendo qué rol van a tener estos email invitados de esta forma, esto se puede apreciar en la Figura 3.16.

3.1.3.3. Conclusión

Las características más comunes son ofrecer un enlace para compartir archivos, con la posibilidad de cambiar los permisos que el rol concede, o compartir directamente por email rellenando un campo de texto con distintos emails y eligiendo el permiso que se les desea dar.

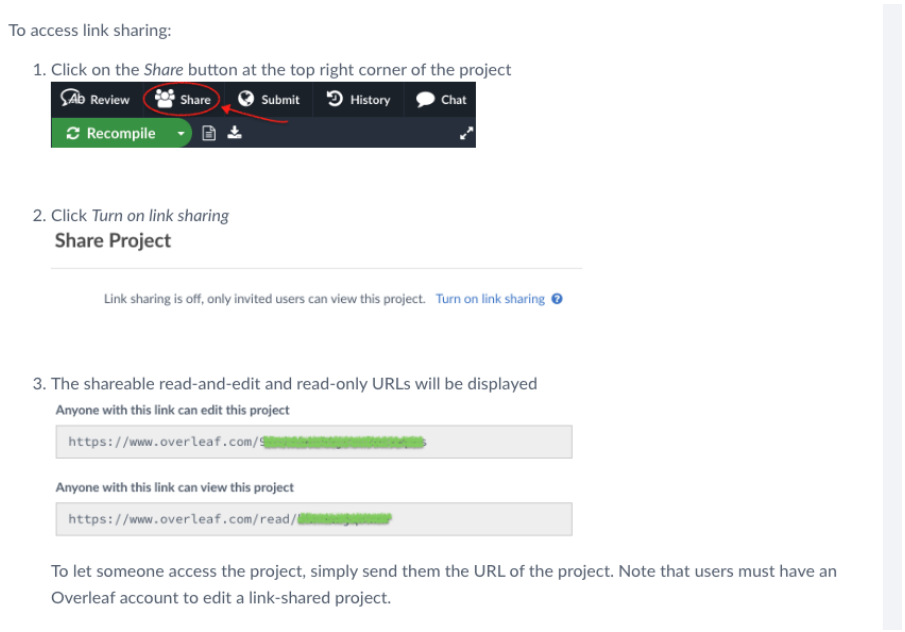


Figura 3.15: Documentación de Overleaf mostrando la función de compartir. Overleaf ofrece la posibilidad de compartir enlaces con permisos distintos.

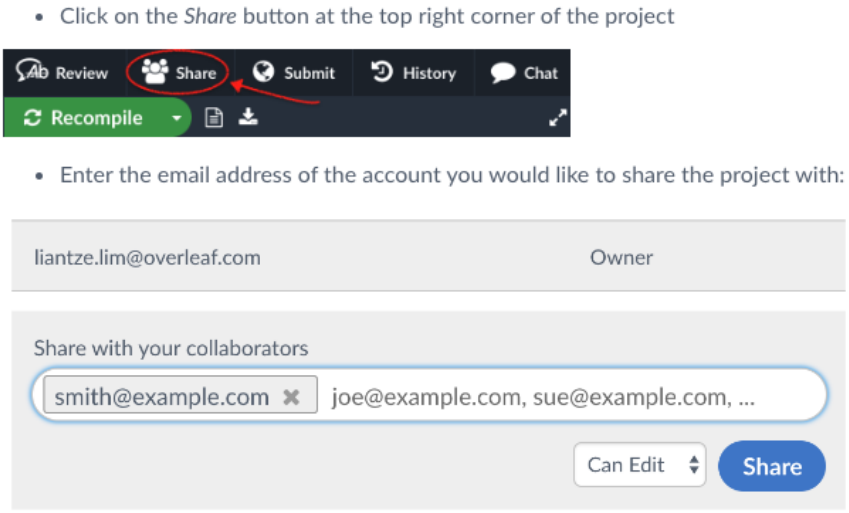


Figura 3.16: Documentación de Overleaf mostrando la función de compartir. Overleaf también ofrece la capacidad de invitar a usuarios por mail, eligiendo qué permisos tendrán.

3.2. Tecnología de capa de presentación

En esta sección describiremos las herramientas utilizadas en la capa de presentación de la aplicación. Estas herramientas son las mismas que se utilizaron en ediciones pasadas sin ningún agregado.

3.2.1. JavaScript

JavaScript[15] es un lenguaje de programación interpretado, dinámico y orientado a objetos, ampliamente utilizado en el desarrollo web. Originalmente concebido para ejecutarse en navegadores y brindar interactividad a las páginas HTML, ha evolucionado para convertirse en una tecnología central en el desarrollo de aplicaciones web modernas, tanto del lado del cliente como del servidor (a través de entornos como Node.js).

Es un lenguaje multiparadigma que permite programar en estilos imperativos, funcionales y orientados a objetos, lo cual le da gran flexibilidad en distintos escenarios. JavaScript es actualmente el lenguaje más utilizado en el desarrollo *frontend*, debido a su integración nativa con los navegadores web y su ecosistema de bibliotecas y frameworks (como Angular, React o Vue).

3.2.2. Angular

Angular[16] es un framework de desarrollo web desarrollado por Google, orientado a la creación de aplicaciones de una sola página (Single Page Applications, SPA). Promueve una arquitectura basada en componentes reutilizables, módulos claramente definidos y servicios inyectables, lo que favorece la mantenibilidad y extensibilidad del código a largo plazo.

Uno de los pilares fundamentales de Angular es su estrecha integración con TypeScript[17], un superset de JavaScript desarrollado por Microsoft. TypeScript extiende JavaScript al incorporar tipado estático, interfaces, clases y anotaciones explícitas, entre otras características propias de lenguajes fuertemente tipados. Esto permite a los editores de código detectar errores en tiempo de desarrollo, mejorar la función de autocompletar de los editores, y facilitar la lectura y documentación del sistema, especialmente en proyectos grandes o mantenidos por múltiples desarrolladores.

3.2.3. StencilJS

StencilJS[18] es una librería para construir librerías de componentes reutilizables agnósticos al framework o librería que se utilice para manejar el resto de la capa de negocio. Varios de los modales del sitio Matefun están escritos con esta herramienta.

[15] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[16] https://angular.dev/
[17] https://www.typescriptlang.org/
[18] https://stenciljs.com/

3.3. Tecnología de capa de persistencia

En esta sección describiremos tecnologías utilizadas para persistir datos.

3.3.1. PostgreSQL

PostgreSQL[19] es una base de datos relacional open source con más de 35 años de desarrollo activo, con una fuerte reputación por confiabilidad, robustez de sus funcionalidades, y performance. Está basada en la base de datos POSTGRES[20] desarrollada en la Universidad de California en el Departamento de Ciencia Computacional de Berkeley.

3.3.2. Redis

Redis[21] es una base de datos volátil NoSQL clave-valor que se utiliza en la memoria RAM. Es utilizada para cache, como store para sesiones distribuídas, store para data NoSQL, motor de Búsqueda y queries, y más aplicaciones.

3.4. Tecnología de capa de negocios

En esta sección presentaremos las tecnologías de la capa de negocio que son de interés para este proyecto.

3.4.1. Ruby on Rails

Ruby[22] es un lenguaje de programación dinámico orientado a objetos, tiene una sintaxis muy expresiva y una gran comunidad detrás. Ruby on Rails[23] es un framework para desarrollo web escrito en Ruby. Está diseñado para hacer que la programación de aplicaciones web sea más fácil tomando suposiciones de lo que todo desarrollador necesita para comenzar. Permite escribir menos código consiguiendo más resultados que muchos otros lenguajes y frameworks (*Getting Started with Rails*, s.f.).

El mismo utiliza la arquitectura *Model View Controller*, la cual consta de tener un componente *router*, que al recibir una *request* la dirige a uno de varios controladores que posee la aplicación, cada controlador se centra en las *requests* de un tipo de recurso, por ejemplo un sólo controlador sabe cómo crear, actualizar, destruir, y mostrar documentos. Los controladores a su vez se comunican con los modelos, cada modelo tiene la lógica misma de un recurso, por ejemplo el controlador de documentos puede comunicarse con los modelos de usuario y documento para conseguir los datos que necesita para contestar una *request*.

[19] https://www.postgresql.org/
[20] https://dsf.berkeley.edu/postgres.html
[21] https://github.com/redis/redis
[22] https://www.ruby-lang.org/
[23] https://rubyonrails.org/

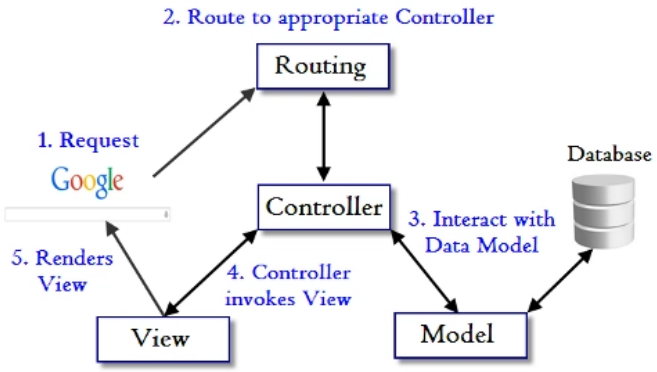


Figura 3.17: Diagrama de la arquitectura *Model View Controller*. Imagen creada por Okolo (2024)

Luego el controlador manda los datos necesarios a una *view*, la cual tiene las instrucciones necesarias para crear la respuesta final que recibirá la computadora que inició la *request*. Un diagrama de estos pasos se puede ver en la Figura 3.17.

Uno de los fundamentos de Rails es çonvención antes que configuración"(*convention over configuration*). Con esto Rails intenta disminuir la cantidad de decisiones sobre configuración que el desarrollador tiene que tomar con el objetivo de un más rápido desarrollo de aplicaciones y hacer que la tarea del desarrollador sea más sencilla.

Un ejemplo de cómo Rails logra este propósito es el ORM (Object-Relational Mapper) propio que tiene para interactuar con bases de datos, llamado Active Record. Si se quieren tener un modelo User y un modelo MatefunFile y establecer una relación one-to-many entre los modelos se puede hacer con los siguientes pasos:

1. Crear migraciones que creen las tablas users y matefun_files, y que esta última tenga una columna llamada user_id. Los nombres son convención de Rails con lo cual ActiveRecord ya sabrá que estos corresponden a modelos User y MatefunFile que se encuentran en la carpeta models, si existen. Las migraciones se pueden ver en las Figuras 3.18 y 3.19.

2. Crear los archivos de los modelos y que hereden de ActiveRecord::Base.

3. En el modelo MatefunFile escribir belongs_to :user. Esto le dice a ActiveRecord que la columna user_id tendrá una referencia a la tabla users.

4. En el modelo User escribir has_many :matefun_files. Esto le dice a ActiveRecord que en la tabla matefun_files existe una columna user_id que señala a los registros que pertenecen al User. En la Figura 3.20 se pueden ver los modelos al final de este paso.

```ruby
class CreateUsers < ActiveRecord::Migration[7.1]
  def change
    create_table :users do |t|

      t.timestamps
    end
  end
end

```

Figura 3.18: Migración para crear la tabla de users. La línea t.timestamps agrega las columnas created_at y updated_at al modelo.

```ruby
class CreateMatefunFiles < ActiveRecord::Migration[7.1]
  def change
    create_table :matefun_files do |t|
      t.references :user

      t.timestamps
    end
  end
end

```

Figura 3.19: Migración para crear la tabla de matefun_files. La línea t.references :user agrega la columna user_id con un entero a la tabla.

5. Esto genera métodos para interactuar con la base de datos. Con esto si se tiene un registro cargado en una variable user, se puede hacer por ejemplo user.matefun_files, y ActiveRecord traerá los registros de la tabla matefun_files que le pertenezcan a ese registro de la tabla users. Esto se puede ver en la Figura 3.21.

Esto es sólo un ejemplo, las columnas y nombres de tablas se pueden configurar de otra forma si el desarrollador lo desea, pero si se sigue la convención de Rails hay que anotar menos cosas. Si bien parece mágico no es tal, belongs_to y has_many no son más que métodos de Rails que generan los métodos auxiliares como el mostrado para referirse a los matefun_files de un usuario.

```ruby
class User < ActiveRecord::Base
  has_many :matefun_files
end

```

```ruby
class MatefunFile < ActiveRecord::Base
  belongs_to :user
end

```

Figura 3.20: Los modelos User y MatefunFile. Las líneas has_many :matefun_files y belongs_to :user hace que ActiveRecord sepa que son asociaciones.

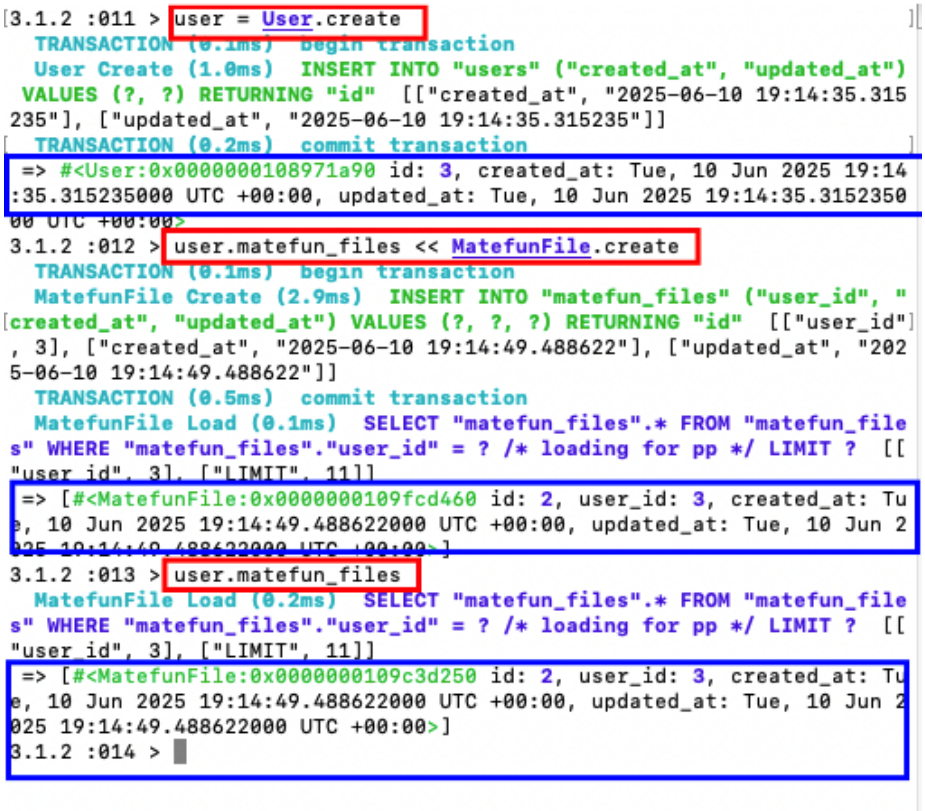


Figura 3.21: Ejemplo de manejo de relaciones entre modelos en Rails. En rojo están encuadrados los comandos y en azul las respuestas. También se pueden apreciar las consultas SQL entre los comandos y las respuestas.

3.4.1.1. Action Cable

Action Cable es el módulo de Rails que se encarga de las conexiones por websocket. Tiene todo lo necesario dentro de Rails para poder fácilmente utilizar conexiones por websockets, así como también una librería JavaScript que se encarga de las conexiones desde la capa de presentación.

Action Cable utiliza el patrón de mensajería Pub/Sub[24], la forma que está implementado es con *channels* y *rooms*. Existen varios *channels*, cada uno encargado de un tipo de comunicación, dentro de cada *channel* pueden haber uno o más *rooms* que son instancias específicas de dichos *channels*. Los *subscribers* se suscriben a un *room* de un *channel* de comunicación, y los *publishers* pueden enviar información a todos los *subscribers* de un *room* particular. Para manejar estas suscripciones ActionCable utiliza una base de datos.

Un ejemplo de un *channel* de ActionCable sería un *channel* documentos, y un ejemplo de un *room* en ActionCable sería un *room* para un documento particular donde se maneja las actualizaciones del mismo. Otro ejemplo de *channel* sería uno para las interacciones con la línea de comandos, mientras que ejemplos de *rooms* es una instancia específica de un usuario en un tab interactuando con su línea de comandos.

Los pasos que se siguen para utilizar los *rooms* son los siguientes:

1. La capa de presentación crea una conexión websocket con la capa de negocios, esto es llamado una Connection. En este paso se hace la autenticación del usuario si es necesaria. Toda la comunicación desde esta instancia de la web hacia la capa de negocios pasará por esta conexión de websocket.

2. La capa de presentación manda un mensaje para subscribirse a un *channel* específico de los que existen en la capa de negocios, como podría ser un *channel* de documentos llamado DocumentChannel. Al mandar la subscripción también se está subscribiendo a un *room* específico dentro de ese *channel*. Podría haber un solo *room* para todo el *channel*, o podrían hacerse varios *rooms* dependiendo de propiedades del usuario o del parámetros que envíe. Por ejemplo al subscribirse al DocumentChannel podría enviar la id de un documento y así subscribirse específicamente al *room* de ese documento.

3. Varios usuarios se pueden suscribir al mismo *room* si la lógica de negocios así lo permite, o sino puede también haber un *room* separado por usuario.

4. A partir de ahora la capa de presentación puede mandar mensajes al *room* al que está suscripto, la cual será manejada por el *channel* correspondiente. Asimismo desde cualquier parte de Rails se puede mandar mensajes a todos los usuarios conectados a un mismo *room* por medio de Broadcast.

[24] https://guides.rubyonrails.org/action_cable_overview.html#pub-sub

3.5. Edición colaborativa

Una de los principales focos de este proyecto fue agregar la funcionalidad de edición colaborativa a Matefun. Para esto se analizaron los dos modos más populares de llevarlo a cabo, Operational Transformation (OT) (C.A. Ellis, 1989) y Conflict-free Replicated Data Type (CRDT) (Shapiro y cols., 2011).

3.5.1. Operational Transformation (OT)

Este modo se centra en transformar las operaciones cuando ocurren varios cambios en el mismo documento de manera simultánea, para llegar a una consistencia final donde todos los usuarios y el servidor tengan una misma copia del documento. Estas transformaciones pueden ser hechas por el servidor central y/o por los usuarios dependiendo el orden en que se reciben.

Pongamos un ejemplo del algoritmo que usa Google para mostrar cómo funciona (Anton Zagorskii, 2018).

Tenemos dos usuarios (Alice y Bob) trabajando en un documento que empezará en blanco, y un servidor central que se encargará de transformar las operaciones de los usuarios en caso de necesidad:

1. Alice escribe “Hello”. Un diagrama del paso se puede ver en la Figura 3.22.

   Cuando Alice escribe “Hello”, se crea una instrucción, en este caso “Insert ’Hello’, @0” que señala que Alice insertó el string “Hello” en el índice 0 del documento. Mientras Alice no ha mandado esta instrucción al servidor central, será guardada como “Pending changes” (Cambios pendientes) en su documento, pero el documento que verá Alice ya mostrará el cambio ya que es importante que para el usuario el proceso de sincronización sea transparente. Una vez se mande la instrucción al servidor central, la misma cambiará de lugar de “Pending changes” a “Sent changes” (Cambios enviados).

   Una vez el servidor recibe la instrucción, su documento local pasará a tener un nuevo “Pending Changes” que tendrá como información la instrucción de Alice “Insert ’Hello’, @0”, así como también el usuario que hizo el cambio “Alice”, y el número de revisión del documento que esa instrucción modificó. Esto último es importante para asegurarnos de que los cambios sean aplicados en el mismo orden por todos los usuarios.

2. Alice escribe “World”, y Bob que aún tiene un documento en blanco escribe “!”. Un diagrama de este paso se puede ver en la Figura 3.23.

   La nueva instrucción de Alice se agregará a sus “Pending Changes”, pero no será enviada al servidor hasta que el mismo no mande una confirmación de que aceptó el último cambio, ya que sólo puede haber un cambio enviado a la vez por usuario.

   También como se ve en la figura, el servidor aceptó la instrucción de Alice, y el documento del mismo tiene el string que Alice había ingresado.

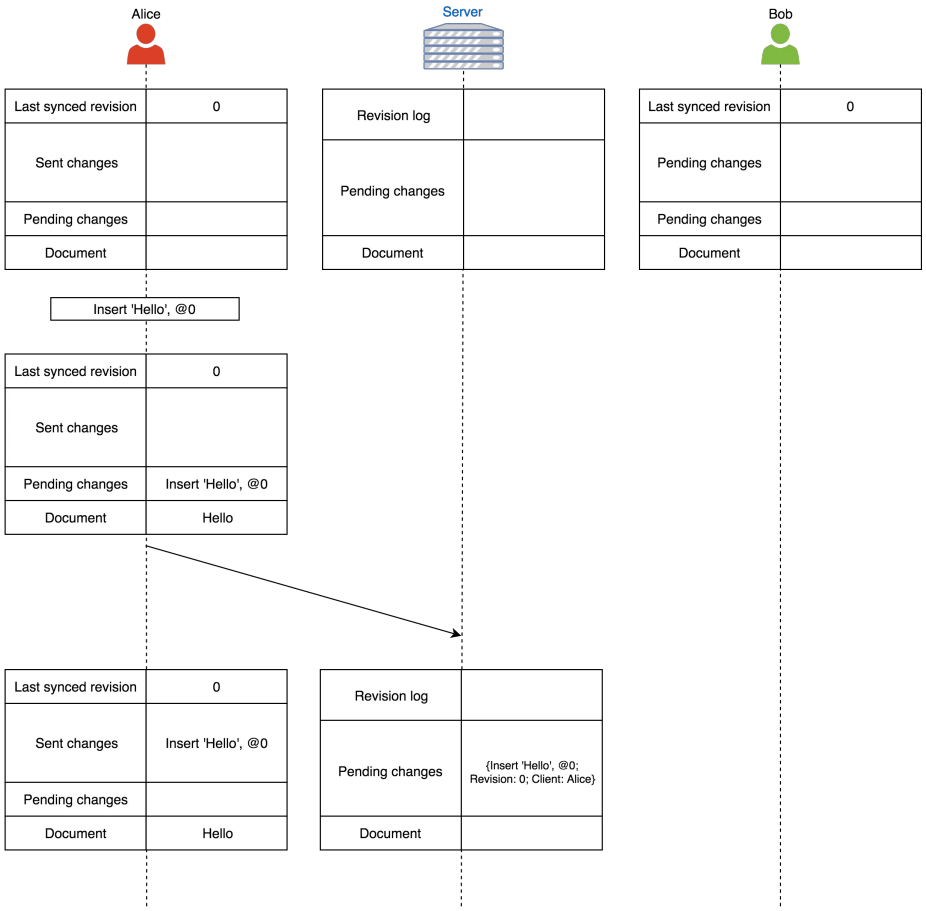


Figura 3.22: Ejemplo de Operational Transformation. Alice escribe “Hello”, el cambio se propaga al servidor

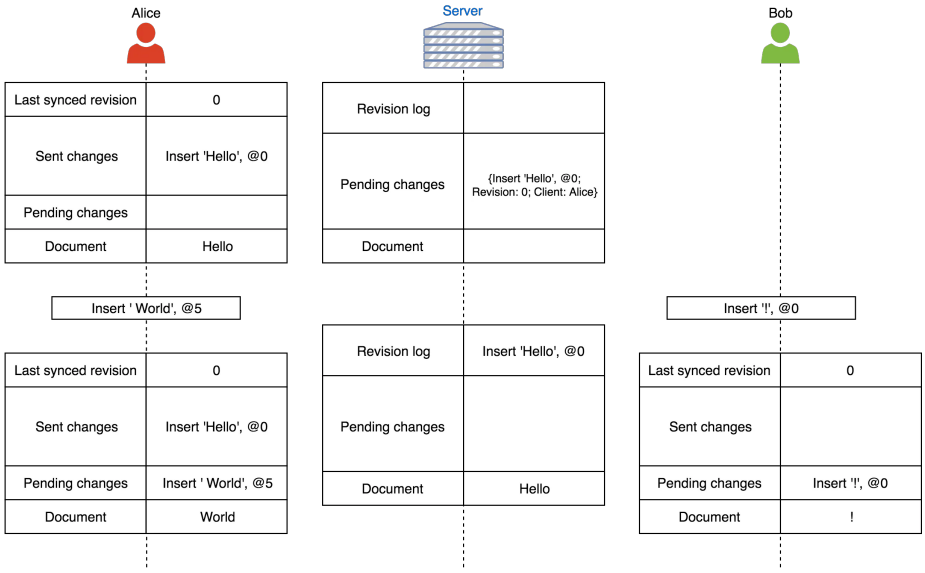


Figura 3.23: Ejemplo de Operational Transformation. Alice escribe “World”, y Bob escribe “!”. El Cambio de Alice no se enviará hasta no recibir confirmación de su último cambio

3. El servidor confirma el cambio a Alice, y lo propaga a Bob. Un diagrama de este paso se puede ver en la Figura 3.24.

   Para Alice el único trabajo que tiene que hacer es actualizar su número de revisión al que le manda el servidor que es 1, y ya está lista para mandar su siguiente instrucción.

   Bob recibe la instrucción y el número de revisión, y tiene que primero transformar su “Pending changes” para acomodarlos a los cambios producidos por la instrucción recibida, en este caso el índice en el que Bob había introducido “!” antes de tener los cambios de Alice se mueve al índice 5, y la instrucción ahora queda “Insert ’!’, @5”. El documento de Bob ahora refleja tanto el “Hello” que había escrito Alice como su propio “!”.

4. Alice y Bob ambos mandan sus “Pending changes” al servidor. Un diagrama de este paso se puede ver en la Figura 3.25.

   Ambos cambios se agregan primero a la lista de “Pending changes” del servidor, quien luego aplica el primero de ellos, y manda confirmación del mismo a Alice y propaga la instrucción a Bob.

   Debido a esto Bob debe cambiar su instrucción nuevamente para que el índice sea el correcto como en el paso 3, sin embargo como ya envió esta operación, no es necesario que la mande devuelta.

5. El servidor transforma la operación de Bob, la confirma y propaga a Alice.

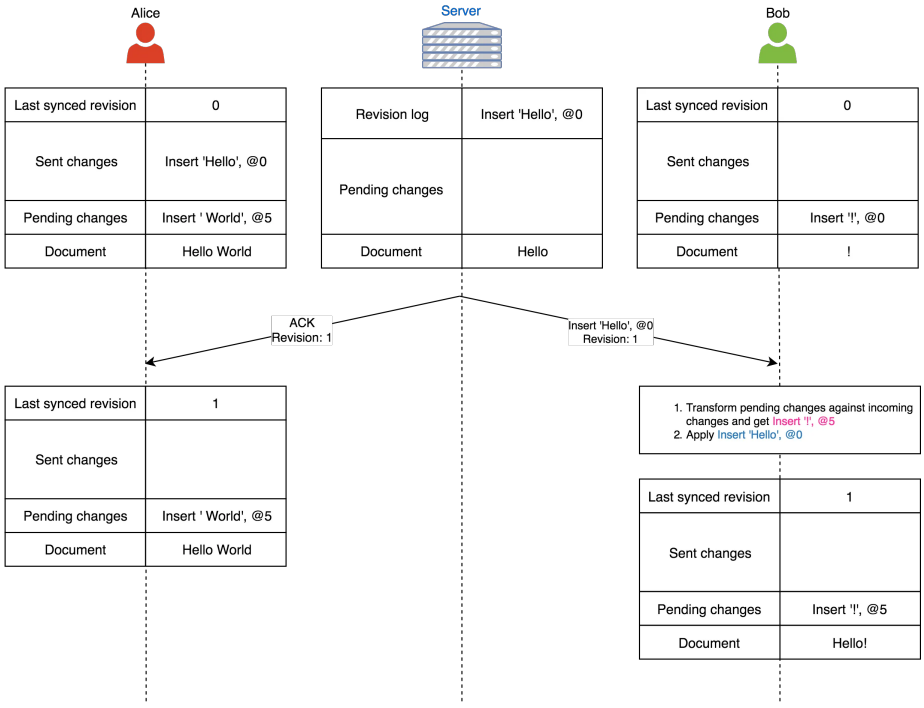


Figura 3.24: Ejemplo de Operational Transformation. El servidor propaga el cambio de Alice

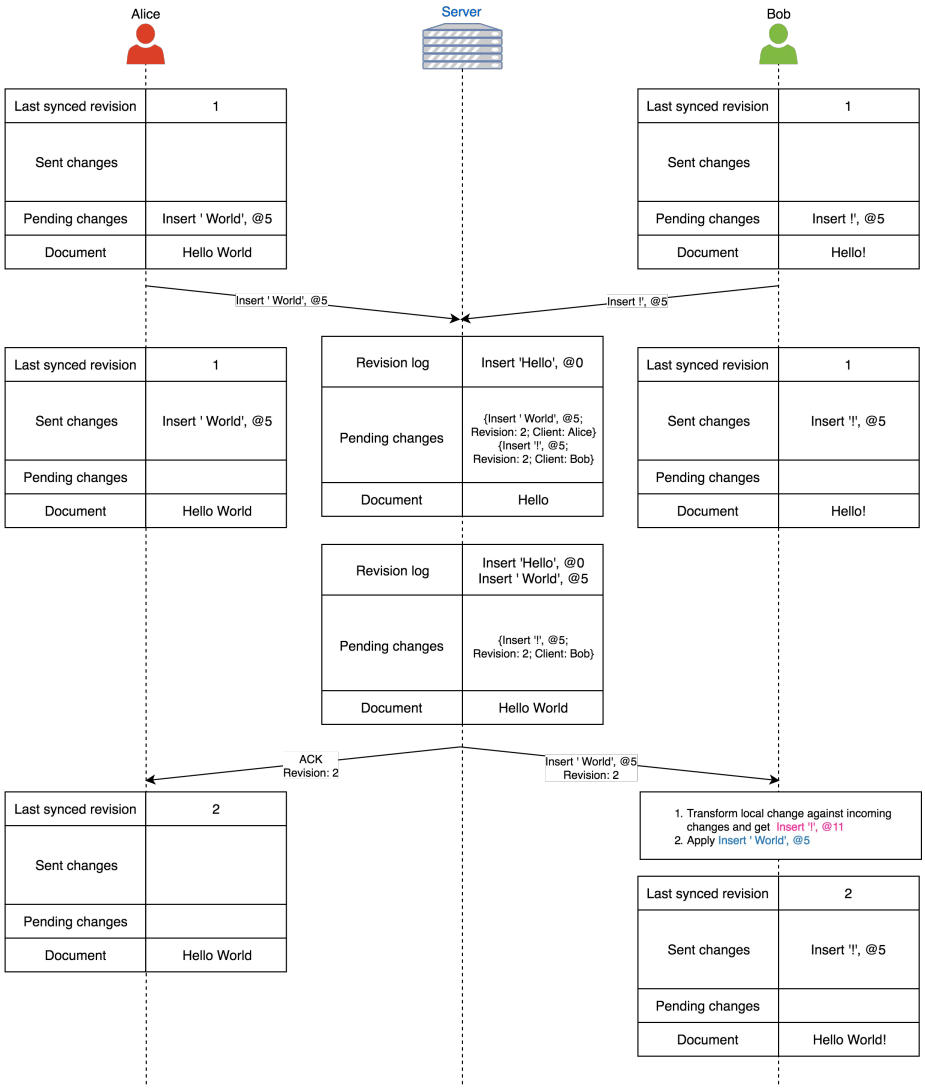


Figura 3.25: Ejemplo de Operational Transformation. El servidor recibe dos cambios en simultáneo, y aplica uno de ellos

Un diagrama de este paso se puede ver en la Figura 3.26.

En este ejemplo vimos como OT trabaja sobre las operaciones para que todos los cambios sean aplicados de manera consistente por todos los usuarios y el servidor.

3.5.2. Conflict-free Replicated Data Type (CRDT)

Los CRDT son algoritmos que ofrecen control de concurrencia optimístico, en particular el subconjunto llamado CmRDT (Commutative Replicated Data Type) lo hace al definir operaciones conmutativas que no generan conflictos entre ellas, por lo cual mientras todas las copias reciban las actualizaciones de las demás copias eventualemnte todas convergerán (Shapiro y cols., 2011)

Los CRDT tienen muchos usos, pero en particular se pueden usar para mantener el estado de un documento de texto. Pongamos como ejemplo la librería Y.js (*yjs*, s.f.). La misma utiliza un algoritmo que es una variación más optimizada del presentado en el artículo Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types (Nicolaescu y cols., 2016)

Este artículo presenta una forma de edición colaborativa llamada YATA (Yet Another Transformation Approach) que asegura la convergencia, preserva la intensión de los usuarios, permite edición offline, y se puede utilizar para varios tipos de datos, si bien nos centraremos en texto.

Para esto usa una lista doblemente enlazada, cuando se agrega un caracter al texto, se hace una inserción en la lista de un elemento, cuando se quiere eliminar un caracter, simplemente se marca como borrado sin eliminarlo de la lista, y más adelante un *garbage collector* se encarga de borrarlo definitivamente.

El elemento ingresado en la lista tiene aparte del caracter a insertar, una id generada por un id único de usuario y un contador de operaciones que incrementa en uno en cada nueva operación que hace el usuario, una referencia al caracter que lo sucede que pueden cambiar una vez se ingresan más caracteres en el texto, y una referencia al caracter que lo precese que nunca cambiará incluso si se insertan nuevos caracteres entre este caracter y ese.

Por ejemplo digamos que tenemos dos usuarios, Alice y Bob, Alice escribe primero “AD”, Bob lo recibe y luego escribe “BC” entre la A y la D de Alice, a su vez Alice escribe una “E” entre la A y la D también, y le propaga esta nueva inserción a Bob. Como se ve en la Figura 3.27 Bob tendrá como datos entre qué dos elementos fue insertado el caracter E, y con un algoritmo que asegura el orden total para cualquier elemento es posible garantizar que cuando Alice reciba el “B” (Figura 3.28) y luego el “C” de Bob el string final será el mismo.

3.6. Plataforma de envío de emails

En esta sección se describe una plataforma muy utilizada para envíos de emails.

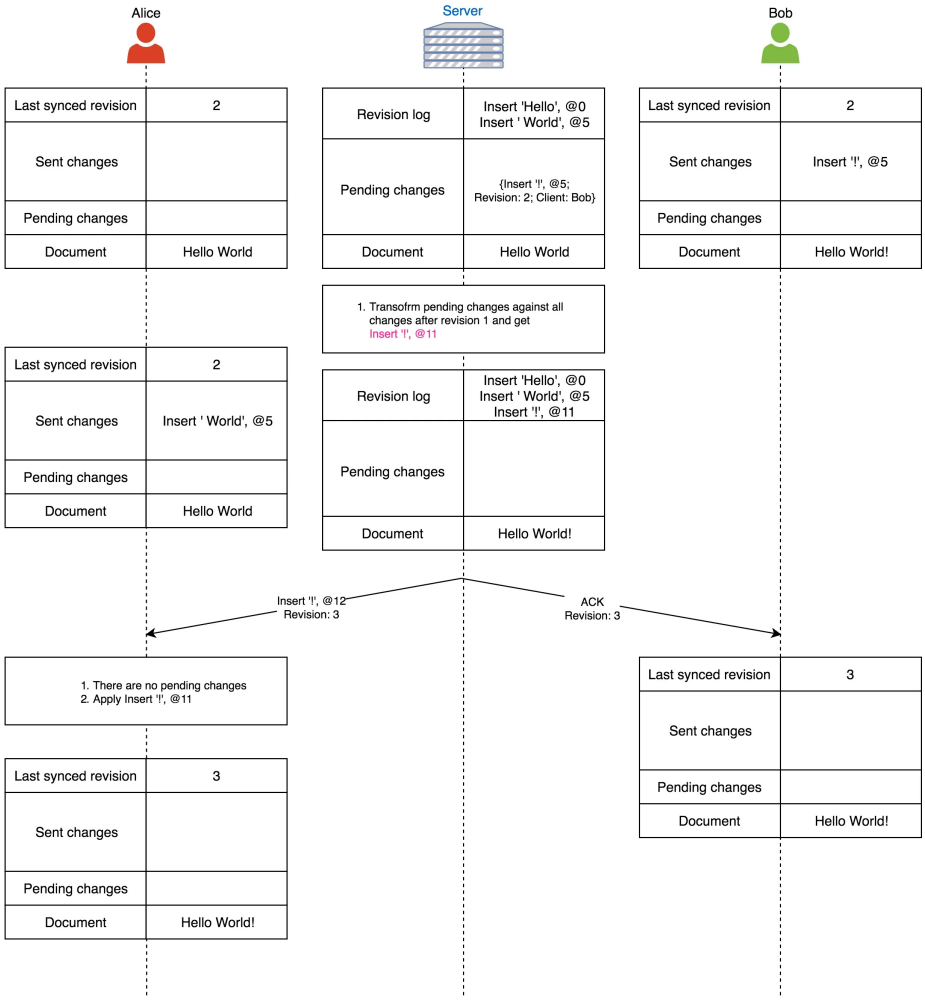


Figura 3.26: Ejemplo de Operational Transformation. El servidor transforma la operación de Bob, la confirma y propaga a Alice

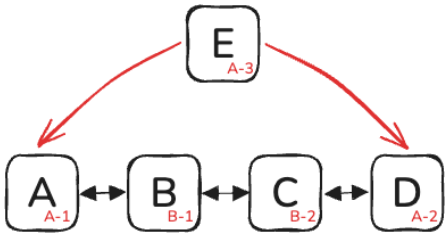


Figura 3.27: Ejemplo de CRDT. Inserción de un caracter enviado por Alice a Bob, abajo a la derecha de cada elemento el id de ese elemento compuesto por el id del usuario y el número de operación de ese usuario

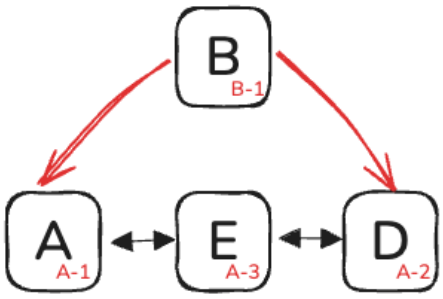


Figura 3.28: Ejemplo de CRDT. Inserción de un caracter enviado por Bob a Alice, abajo a la derecha de cada elemento el id de ese elemento compuesto por el id del usuario y el número de operación de ese usuario

3.6.1. SendGrid

SendGrid[25] es una plataforma de entrega de correos electrónicos en la nube que permite enviar grandes volúmenes de emails de manera segura y confiable. Su API es ampliamente utilizada por aplicaciones web y móviles para gestionar el envío de mensajes transaccionales y de marketing. SendGrid posee las siguientes características:

- **Facilidad de integración:** SendGrid ofrece una API REST simple que permite integrarse rápidamente con aplicaciones construidas en diversas tecnologías.
- **Escalabilidad:** Permite enviar miles de correos sin afectar el rendimiento del servidor.
- **Seguimiento y estadísticas:** Proporciona métricas detalladas sobre los correos enviados, incluyendo tasas de apertura, clics, rebotes y entregas fallidas, lo cual permite monitorear y mejorar la comunicación con los usuarios.
- **Soporte para plantillas:** Es posible definir plantillas HTML para los correos, lo que facilita mantener una apariencia profesional y consistente en todos los mensajes del sistema.

[25] https://sendgrid.com/

Capítulo 4

Parte Central

En esta sección desarrollaremos el trabajo realizado para llegar a los objetivos del proyecto, las decisiones que se tomaron y las razones por las que fueron tomadas.

4.1. Arquitectura

En esta sección se describe la arquitectura general del sistema desarrollado, con énfasis en las decisiones de diseño tomadas durante el proyecto. La arquitectura propuesta busca satisfacer los requerimientos funcionales y no funcionales establecidos previamente, incorporando nuevas funcionalidades como la edición colaborativa, la gestión de usuarios y grupos, y una experiencia de usuario más robusta y flexible.

El diseño del sistema se estructura en varias capas que interactúan entre sí: presentación, negocio y persistencia (tanto volátil como no volátil). Se adoptó una arquitectura modular que facilita el mantenimiento, la extensibilidad y la colaboración entre diferentes desarrolladores a lo largo del tiempo, siguiendo principios como la separación de responsabilidades y el bajo acoplamiento.

A su vez, se describen los **actores** que interactúan con el sistema, sus **casos de uso principales**, y los componentes tecnológicos involucrados, incluyendo frameworks y herramientas empleadas para implementar funcionalidades clave como la comunicación en tiempo real y la sincronización de ediciones.

4.1.1. Actores

El sistema Matefun está construido para soportar dos tipos de actores distintos, los usuarios y los usuarios administradores del sitio.

En versiones pasadas de la aplicación el sistema estaba diseñado para soportar dos tipos de actores, alumnos y docentes. Ambos actores pertenecían a un grupo específico compuesto de uno o más alumnos y uno o más docentes. Ambos actores podían crear archivos propios, editarlos y ejecutarlos. Los docentes

aparte podían asignar archivos a los alumnos, los cuales podían luego editarlos y "entregarlos", lo cual hacía que los docentes pudieran verlos y editarlos para dar retroalimentación y calificarlos.

A diferencia de las versiones pasadas (Gonzalo Cameto, Accessed: 2025-06-10b) donde habían alumnos y docentes como actores separados, en esta versión han sido unidos en el actor usuario. El mismo puede usarse tanto como para hacer tareas y pedir retroalimentación como hacía el actor tipo alumno de antes, así como también para crear dichas tareas y dar la retroalimentación como hacía el actor tipo docente.

El otro tipo de actor que existe nuevo en esta versión es el usuario administrador del sitio. El mismo puede acceder a una web donde podrá ver datos generales del sitio, como los distintos usuarios que lo componen.

4.1.1.1. Roles

Para llevar a cabo los casos de uso cada usuario que tiene un archivo tiene un rol asociado a ese archivo, y cada usuario que pertenece a un grupo tiene un rol asociado a ese grupo. Los roles limitan los permisos que los distintos usuarios tienen sobre el recurso correspondiente.

Los distintos roles asociados a un archivo son:

Lector: Este usuario puede solamente leer el archivo.

Editor: Este usuario puede también editar el archivo.

Administrador: Este usuario puede también compartir el archivo con otros usuarios, así como editar los roles de otros usuarios en el archivo, excepto del usuario con rol de Dueño

Dueño: Este usuario puede editar los roles de todos los usuarios, e incluso hacer que otro usuario pase a ser Dueño del archivo. Si el usuario con el rol de Dueño borra el archivo, el mismo será borrado completamente del sistema para todos.

De manera muy similar, los roles asociados a un grupo son:

Miembro: Este usuario puede crear archivos en el grupo, puede participar de subgrupos, y puede pedir retroalimentación de cualquier archivo al que tenga acceso dentro del grupo.

Moderador: Este usuario puede también ver los archivos con pedido de retroalimentación de otros usuarios, editar dichos archivos, y devolver la retroalimentación pedida. Asimismo puede también crear subgrupos dentro del grupo y agregar y sacar usuarios de los subgrupos ya existentes.

Administrador: Este usuario puede también agregar usuarios al grupo, así como editar los roles de otros usuarios en el grupo, excepto del usuario con rol de Dueño.

Dueño: Este usuario puede editar los roles de todos los usuarios, e incluso hacer que otro usuario pase a ser Dueño del grupo. Si el usuario con el rol de Dueño borra el grupo, el mismo será borrado completamente del sistema para todos.

De esta forma podemos por una parte imitar los roles de Docente y Alumno que se tenía en las versiones anteriores del proyecto, y por otra parte abrir posibilidades a nuevas formas de interactuar con grupos y archivos que antes no eran posibles.

4.1.2. Casos de uso

Aquí se presentan los casos de uso principales que difieren a las versiones anteriores de la aplicación. En el Apéndice A se pueden ver una lista con más casos de uso.

Registrarse: Este caso de uso comienza cuando un actor usuario desea crear un usuario en el sitio web. Para esto debe:

1. Navegar a la página de registrarse.
2. Introducir email, nombre de usuario deseado, y contraseña.
3. El usuario recibirá un email para confirmar la cuenta.
4. Hacer click en el enlace del email para confirmar la cuenta.

Iniciar sesión: Este caso comienza cuando el actor usuario desea entrar al sitio web, habiéndose previamente registrado. Para esto debe:

1. Navegar a la página de login.
2. Introducir su email y contraseña.
3. Hacer click en el botón de login.
4. El usuario será redirigido a la vista principal de la web.

Entrar como invitado: Este caso comienza cuando el actor usuario desea probar utilizar el sitio web sin crear cuenta. Para esto debe:

1. Navegar a la página de login.
2. Presionar el botón de Invitado.
3. El usuario será redirigido a la vista principal de la web.

Los usuarios invitados están hechos para ser temporales, y serán borrados periódicamente, por lo cual su uso es puramente para probar el sitio web antes de crearse una cuenta definitiva.

Compartir archivo: Este caso comienza cuando un usuario con rol de Dueño o Administrador de un archivo desea compartir dicho archivo con otros usuarios del sitio, o cambiar los permisos de otros usuarios con los que ya ha previamente compartido el archivo. Para esto:

1. Navegar hasta la vista de archivos.
2. Navegar en las carpetas hasta encontrar el archivo que desea compartir.
3. Opcionalmente el usuario puede cargar el archivo en la vista principal haciendo click en el botón 
4. El usuario debe hacer click en el botón  .
5. Se despliega el menú de compartir archivo.
6. El usuario puede opcionalmente ingresar los nombres de usuario que desea agregar, con sus respectivos roles.
7. El usuario puede opcionalmente cambiar roles de usuarios con los que ya tenga compartido el documento eligiendo el nuevo rol de un dropdown.
8. El usuario puede opcionalmente sacarle los permisos sobre el archivo a otros usuarios seleccionando el botón de eliminar en la fila donde está listado el usuario.
9. El usuario confirma los cambios haciendo click en el botón Guardar.
10. Alternativamente el usuario cancela los cambios haciendo click en la cruz en la parte superior del menú de crear usuarios.

Crear un grupo: Este caso comienza cuando un usuario desea crear un grupo. Para esto:

1. El usuario navega hasta la sección de grupos por el menú lateral.
2. El usuario hace click en el botón  ubicado en el cabezal de la vista de grupos.
3. El sistema despliega el menú para crear grupos.
4. El usuario ingresa un nombre para el grupo.
5. El usuario confirma la creación del grupo haciendo click en el botón Crear Grupo.
6. Alternativamente el usuario cancela los cambios haciendo click en la cruz en la parte superior del menú de crear grupos.

Administrar usuarios de un grupo: Este caso ocurre cuando un usuario con rol de Dueño o Administrador de un grupo desea administrar a los usuarios de un grupo. Para esto:

1. El usuario navega hasta la sección de grupos desde el menú lateral del sitio.
2. El usuario hace click en el grupo del cual desea administrar los usuarios.

3. El usuario hace click en el botón de ubicado en el cabezal de la vista del grupo.

4. El sistema despliega el menú para administrar usuarios.

5. El usuario puede opcionalmente ingresar los nombres de usuario que desea agregar, con sus respectivos roles.

6. El usuario puede opcionalmente cambiar roles de usuarios que ya pertenezcan al grupo eligiendo el nuevo rol de un dropdown.

7. El usuario puede opcionalmente eliminar usuarios del grupo, seleccionando el botón de eliminar en la fila donde está listado el usuario.

8. El usuario confirma los cambios haciendo click en el botón Guardar.

9. Alternativamente el usuario cancela los cambios haciendo click en la cruz en la parte superior del menú de administrar usuarios.

Pedir retroalimentación: El caso comienza cuando un usuario dentro de un grupo quiere pedir retroalimentación de un archivo que posea dentro de ese grupo. Para esto debe navegar hasta sus archivos en el grupo si el archivo no pertenece a un subgrupo, o a los archivos del subgrupo al cual pertenece el archivo, o bien tenerlo abierto en el editor en la página principal. Luego hace click en el botón lo cual le mostrará un mensaje de confirmación de que desea pedir retroalimentación para el archivo.

Ver archivos con pedidos de retroalimentación: El caso comienza cuando un usuario con rol de Dueño, Administrador, o Moderador dentro de un grupo quiere ver los pedidos de retroalimentación de un grupo. Para esto navega hasta el grupo, luego hasta la vista de usuarios de ese grupo, y selecciona el usuario que tiene archivos con pedido de retroalimentación. Alternativamente si el archivo pertenece a un subgrupo debe navegar en su lugar a la lista de subgrupos del grupo y elegir el subgrupo que tiene el pedido de retroalimentación.

Agregar retroalimentación a un archivo: El caso comienza cuando un usuario con rol de Dueño, Administrador, o Moderador dentro de un grupo quiere agregar retroalimentación a un archivo. Para esto primero navega hasta el archivo con pedido de retroalimentación al que desea agregarle retroalimentación, luego lo abre para edición apretando el botón , y en el editor escribe la retroalimentación que desea devolver como comentarios dentro del código.

Devolver retroalimentación: El caso comienza cuando un usuario con rol de Dueño, Administrador, o Moderador dentro de un grupo quiere devolver retroalimentación de un archivo dentro de ese grupo. Para esto debe navegar hasta el usuario dentro de la lista de usuarios del grupo si el archivo no pertenece a un subgrupo, o a los archivos del subgrupo al cual pertenece el archivo, o bien tenerlo abierto en el editor en la página principal. Luego hace click en el botón lo cual le mostrará un mensaje de confirmación de que desea dar retroalimentación para el archivo.

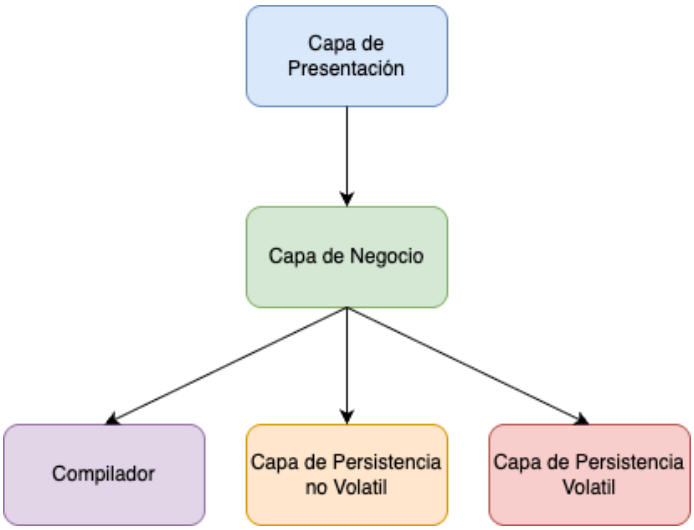


Figura 4.1: Las distintas capaz de la aplicación

Crear subgrupo: El caso comienza cuando un usuario con rol de Dueño, Administrador, o Moderador dentro de un grupo desea crear un subgrupo dentro del mismo. Para esto:

1. El usuario navega hasta la sección de grupos desde el menú lateral del sitio.
2. El usuario hace click en el grupo donde quiere crear un nuevo subgrupo.
3. El usuario hace click en el tab de Subgrupos.
4. El usuario hace click en el botón ✚ ubicado en el cabezal de la vista de subgrupos.
5. El usuario escribe un nombre para el subgrupo que desea crear en el campo de texto señalizado para esto.
6. El usuario elige los usuarios del grupo que desea pertenezcan al subgrupo por crear de la lista de usuarios que se le presenta.
7. El usuario hace click en el botón Crear Subgrupo.

4.1.3. Diagrama arquitectónico

El sistema consiste de las mismas capas que en versiones anteriores, más una nueva capa de persistencia volátil que se utiliza para tener acceso más veloz a los documentos y datos de los usuarios cuando se conectan por sockets. Se puede ver un diagrama de las capas en la Figura 4.1.

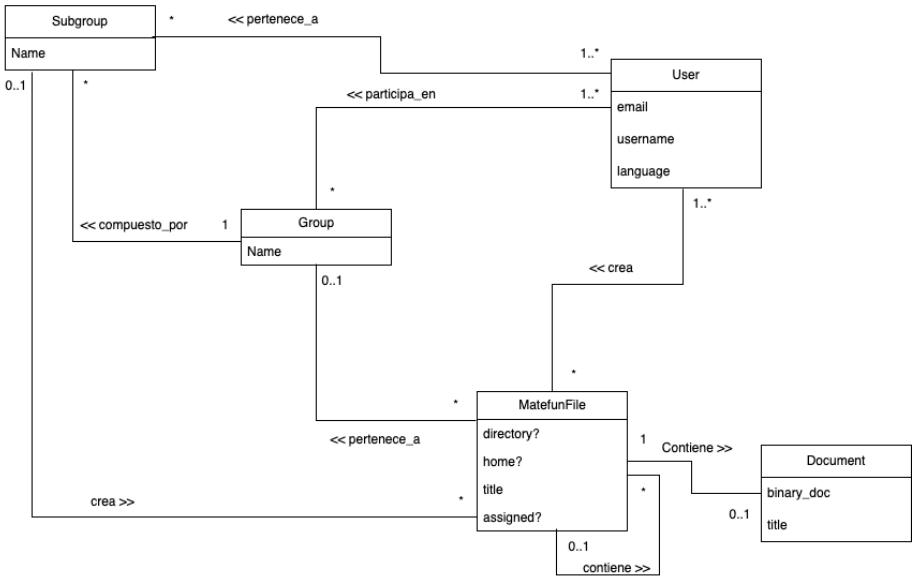


Figura 4.2: Modelo de dominio

4.1.4. Modelo de dominio

Se hicieron varios cambios al modelo de dominio que existía previamente para reflejar la ampliación del enfoque de la web MateFun. El nuevo modelo conceptual se puede encontrar en la Figura 4.2.

Mientras antes el foco estaba puesto en las tareas en aula y sus alumnos y docentes únicamente, ahora el foco es que cualquier persona (incluso docentes y alumnos trabajando en un aula) puedan utilizar el sitio fácilmente. Por esto ya no existe la diferenciación entre usuarios de tipo Alumno y tipo Docente como existía antes como clases separadas, sino que hay sólo un tipo de usuario. Por otro lado existen Grupos, y dentro de cada Grupo un Usuario puede tener distintos roles que pueden reflejar los roles que previamente tenían los tipos de Alumno y Docente, pudiendo el mismo usuario tener distintos roles en distintos grupos a los cuales pertenezca.

Asimismo con la nueva funcionalidad de edición colaborativa, surgió un requerimiento para poder tener Subgrupos dentro de los grupos, a los cuales se les pueda asignar directamente archivos para que los usuarios de dicho subgrupo puedan trabajar en ellos.

4.1.5. Diagrama de Clases

En esta sección mostraremos el resultado del diagrama de clases hecho a partir del modelo de dominio de la sección anterior, el mismo se puede ver en la Figura 4.3. Se presentarán las clases centrales aquí, por un diagrama de clases

más completo referirse al Apéndice B

Aparte de las clases mostradas, el sistema posee clases de tipo Servicio para realizar acciones sobre las clases del diagrama. Por ejemplo, existe el servicio `Documents::CreationService`, el mismo tiene el conocimiento necesario y se encarga de la creación de Documentos, de esta forma para crear un documento en vez de hacer algo como `Document.create(...)`, se hace algo como `Documents::CreationService.new(...).call` y el mismo se encarga de hacer los pasos necesarios para crear un documento.

Se utiliza esta estrategia de tener varios servicios separados con conocimiento específico de cómo hacer estas acciones en vez de la estrategia usual de tener estas acciones dentro de las clases de los modelos ya que presenta algunas ventajas:

- Mantiene el código de los modelos con un tamaño mantenible, evitando tener modelos de cientos de líneas.
- Mantiene el ciclo de cada acción dentro de una clase separada, lo cual hace que sea más fácil de entenderlo.
- Un problema que a veces ocurre al tener modelos muy grandes es que luego de varias refactorizaciones pueden quedar métodos que ya no se usan. Al tener las acciones separadas en archivos más chicos es más fácil ver a la hora de refactorizar si hay métodos que ya no se utilizan.
- Permite una mayor libertad en el caso que en el futuro haya que hacer casos especiales de estas acciones, ya que la lógica no está atada al modelo en sí. Por ejemplo en este momento existe el `Files::SharerService` para compartir archivos, en el futuro se puede crear un `Files::Administrator::SharerService` como un servicio para que un administrador de un grupo comparta archivos de manera especial dentro del mismo, y el mismo puede tener su propia lógica, en vez de tener que estar emparchando la clase original para que funcione con el nuevo caso.

Una lista de los servicios disponibles se encuentra en el Apéndice D

4.2. Nuevas tecnologías utilizadas

Se mantienen de los proyectos anteriores la tecnología Angular utilizada en el componente web, con la única modificación de que ahora también se usan WebSockets para la edición del archivo y no sólo para la consola interactiva. También se mantiene la tecnología utilizada para el intérprete de archivos MateFun. Para el Servidor sin embargo se utiliza una nueva tecnología que se describe a continuación.

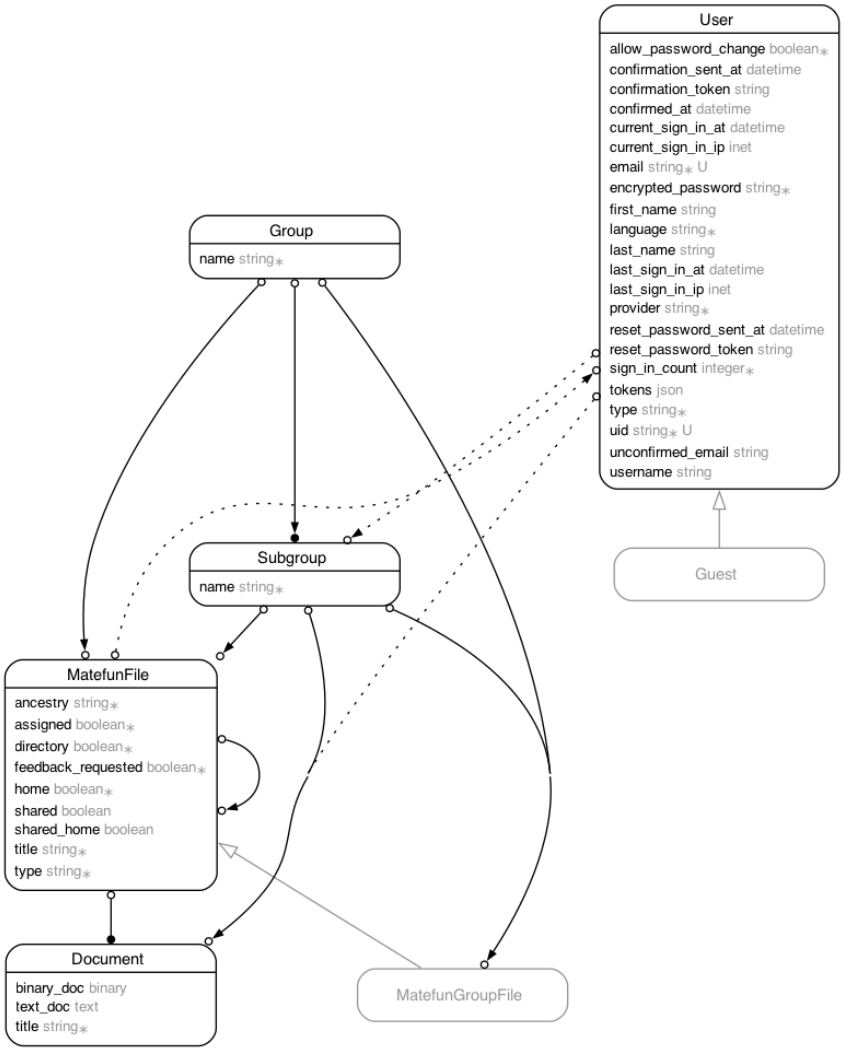


Figura 4.3: Diagrama de clases con las clases más importantes del proyecto. Para un diagrama más completo referirse al Apéndice B

4.2.1. Servidor

En versiones previas del proyecto existían una clase para el actor Docente, y otra clase para el actor Alumno, esto tenía sentido debido a la gran diferencia entre las funcionalidades que estos tipos de usuario tenían. Esto también se reflejaba en que si una persona quería ser un alumno en un grupo y un docente en otros grupos necesitaba tener un usuario completamente distinto para cada caso para lograrlo.

Con la reestructuración de nuestra arquitectura para lograr más flexibilidad y que cada persona pueda cumplir distintos roles en distintos grupos utilizando un único usuario, existen dos posibilidades de desarrollo. Emparchar el código que ya existía y hacerlo funcionar con estas nuevas funcionalidades, o crear un nuevo *backend* que esté diseñado con esta flexibilidad desde el principio. A esto se le suma que la interacción con los documentos y la forma de guardarlos también necesita un cambio para esta nueva versión, lo cual también requeriría hacer cambios grandes a código ya existente.

Debido a los grandes cambios que se necesitan y para que el proyecto tenga una buena base para proseguir, se decidió reescribir la capa de negocios desde cero.

Para esta re-escritura se decidió utilizar Ruby on Rails. Esta elección se debe a que dicha herramienta se caracteriza por agilizar el desarrollo de la web, y por su filosofía de convención antes que configuración.

Esta filosofía de convención antes que configuración significa que si una nueva persona ingresa a un proyecto ya existente, puede saber dónde se encuentra cada cosa y cómo es su funcionamiento, sin más explicación que leyendo las guías que otorga Ruby on Rails, lo cual lo hace un buen sistema para un proyecto como MateFun en el cual trabajarán varias personas distintas a lo largo del tiempo. También su escalabilidad está probada por su uso masivo en plataformas grandes como son github.com, twitter en su momento, masterclass.com, y shopify.com por dar algunos ejemplos.

4.2.1.1. Comunicación con el binario Matefun

Para comunicarse con el binario de Matefun se utilza la librería open4[1] para abrir un hilo y comunicarse con una terminal, posee entrada y salida estandar así como canal de errores con lo cual se puede interactuar de la misma manera que se hacía con la tecnología anterior lo cual hace que se acople bien al uso que buscamos en este proyecto.

4.2.2. Capa de persistencia volátil

Para la comunicación por WebSockets, Rails utiliza una base de datos, como se indica en la sección 3.4.1.1, ofreciendo algunos adaptadores ya incluidos, async, inline, Redis y PostgreSQL. De estos async no está recomendado para un ambiente de producción (*Action Cable Overview - Async Adapter*, Accessed:

[1] https://github.com/ahoward/open4

2025-06-20), y de inline no hay más que una mención en la documentación, pero viendo el código fuente (*ActionCable::SubscriptionAdapter::Async*, Accessed: 2025-06-20) se observa que es simplemente una versión no asíncrona de async, por lo cual si async no está recomendado para producción inline tampoco serviría.

Como se mencionó en la Sección 3.3.2, Redis es una base de datos que guarda su información en la memoria RAM, esto hace que sea muy rápida para el guardado y obtención de datos, pudiendo llegar a menos de un milisegundo en muchos casos (*Why choose Redis?*, s.f.). Como se puede ver en el artículo de Lio (s.f.) Redis puede atender muchos más pedidos por segundo que PostgreSQL (900.000 Redis vs 15.000 Postgres), y con velocidades en el orden de 6 veces más rápidas (0.095 ms Redis vs 0.679 ms Postgres).

Ya que el tiempo de respuesta en la edición colaborativa es muy importante, se eligió utilizar Redis por su clara rapidez comparado con PostgreSQL.

4.3. Diseño e implementación

En esta sección se hablará de las decisiones de diseño e implementación que se tomaron en el transcurso de este proyecto.

4.3.1. Compartir archivos

4.3.1.1. Elección de algoritmo para edición colaborativa

La elección del algoritmo a utilizar para edición colaborativa abarca tanto la capa de negocios como la capa de presentación ya que ambos tienen que utilizar el mismo. Para esta decisión tuvimos en consideración los algoritmos OT y CRDT presentados en la Sección 3.5.

Ambos tipos de algoritmos OT y CRDT tienen aspectos positivos y negativos:

Como característica positiva para ambos es que los dos pueden ser escalables si están bien implementados, que Google use OT es muestra de que OT puede escalar, y la implementación Y.js de CRDT tiene un set de benchmarks que muestra cómo puede funcionar (*crdt-benchmarks*, s.f.), aparte de que también es utilizado en sitios en producción según su propio README (*yjs*, s.f.) incluyendo entre ellos AWS SageMaker.

Por otra parte en OT el documento final siempre es simplemente un documento de texto, mientras que en CRDT es su propio tipo que tiene que ser parseado cada vez que se quiere leer, y tiene datos extra que hacen que ocupen algo más de espacio, en los benchmarks mencionados previamente se puede ver cuáles son los tiempos de parseo y el espacio tomado para un documento de 6000 caracteres, que es de considerable tamaño para los tipos de documentos que se hacen en la web MateFun.

Y por último la facilidad de implementar los algoritmos y las utilidades que traen cada uno. Si bien ambos tienen plugins para el editor web que utilizamos

(Codemirror), sólo CRDT tiene librería que también funciona en *backend* y hace que todo sea automático, mientras tanto implementar OT no es tan sencillo, incluso teniendo sólo las operaciones de inserción y borrado probar que el algoritmo cumpla las propiedades de convergencia y que mantenga intención del usuario no es una tarea sencilla (Li y Li, 2010).

También podemos agregar que la librería para CRDT trae funcionalidades extra, como ver el cursor de la otra persona y deshacer ya incluidos, lo cual es una implementación que habría que haber hecho por separado en caso de usar OT.

Para corroborar que la elección funcionaba antes de hacer la implementación definitiva se hizo un prototipo con Angular, Rails, y la librería Y.js. Se probó que múltiples usuarios pudieran escribir en el mismo editor desde distintas computadoras. Se utilizó el mismo editor web que se utiliza en el proyecto de Matefun para que el ambiente fuera lo más cercano al original y se necesitara la mínima cantidad de cambios. En dichas pruebas también pudimos ver que dado una falla en la red o si cae el servidor, los usuarios pueden seguir editando y una vez se levanta el servidor todos los documentos se actualizan automáticamente sin inconvenientes.

Por todo esto la decisión final fue utilizar la implementación Y.js de CRDT.

4.3.2. Capa de negocio

La capa de negocios es la encargada de enforzar la lógica de negocio, así como de ser intermediaria entre la capa de presentación y la de persistencia, y la capa de presentación y el intérprete MateFun.

4.3.2.1. Gestión de documentos

En esta nueva versión además de utilizar la conexión por websockets para manejar la interacción con el binario de MateFun, la plataforma también la utiliza para manejar los documentos. De esta forma cuando un usuario edita un documento automáticamente se mandan las actualizaciones por websocket a la capa de negocios y la misma los persiste temporalmente en la capa de persistencia volátil. Con este proceso se puede dar una respuesta mucho más rápida al usuario al momento de editar documentos, y a su vez se pueden repartir las actualizaciones del usuario a los demás usuarios que estén conectados simultáneamente.

Existen dos *channels* de Action Cable para manejar los documentos, el DocumentsChannel y el AuxDocumentsChannel. El DocumentsChannel se usa para la actualización del cuerpo de los documentos, mientras que el AuxDocumentsChannel se utiliza para el proceso de guardado de un documento borrador, y para la actualización del nombre de un documento.

El DocumentsChannel es el *channel* más usado, si un usuario abre un documento ya existente para editar se conectará a un *room* en este *channel* que es identificado por la id del documento. El *channel* se encargará de mantener

actualizado el documento en las bases de datos volátil y no volátil y de esparcir las actualizaciones a todos los usuarios conectados al mismo *room*.

El otro caso que se usa el DocumentsChannel es cuando un usuario abre la página principal de Matefun sin ningún documento seleccionado. En este caso el usuario se conecta a un *room* identificado por su id de usuario y el string 'draft', y se crea un documento borrador el cual el usuario podrá editar. Este documento se guardará en la capa de persistencia volátil, pero por defecto no se guardará en la capa no volátil.

Para guardarlo en la capa de persistencia no volátil, el usuario deberá hacer click el botón que se encuentra en la barra del editor. Esto iniciará una *request* para crear el nuevo Documento, y en esa *request* se usará el AuxDocumentsChannel para avisar al cliente que se guardó el documento correctamente y que debe cambiar de *room* al *room* correspondiente. De esta forma si el usuario estaba logueado en más de un tab o computadora, todas serán actualizadas al mismo *room* correctamente. Este proceso se puede observar en los diagramas de secuencia de las Figuras 4.4, 4.5, 4.6 y 4.7, donde también se puede ver la interacción cuando un usuario tiene más de una ventana abierta con el documento borrador, y lo que pasa cuando hace la *request* de guardar el documento desde una de las ventanas.

El otro caso en el que se usa el AuxDocumentsChannel es cuando un usuario actualiza el nombre de un documento, para esto debe mandar una *request* al *endpoint*de actualizar documentos, y el mismo mandará un update a todos los usuarios conectados al AuxDocumentsChannel para avisarles del cambio de nombre del documento.

4.3.2.2. Interacción con el binario MateFun

Para la interacción con el binario de MateFun se mantuvo el uso de websockets, actualizándolo para que usara Action Cable. Cada usuario se conecta en el *channel* GhciChannel a un *room* propio identificado por la id del usuario, con lo cual incluso personas que están editando un mismo documento compartido a la vez pueden tener su propia instancia de terminal ejecutando cosas distintas.

La lógica para interpretar y procesar los mensajes entre la capa de negocio y la capa de presentación se mantuvo idéntica a la utilizada en ediciones anteriores con lo cual se mantuvieron todas las funcionalidades ya existentes.

En las Figuras 4.8, 4.9 y 4.10 se puede ver un diagrama de secuencia de la interacción desde que un cliente se conecta al Websocket hasta que termina la conexión. La misma fue separada en tres partes por razones de espacio.

En la Figura 4.8 se puede apreciar la conexión inicial, la creación del proceso hijo que ejecuta el binario Matefun, y el primer mensaje que devuelve el Binario. Fueron obviados en este caso los mensajes iniciales que escriben Matefun de manera artística por razones de espacio en el diagrama.

En la Figura 4.9 se puede ver cómo el cliente manda un comando con una operación, y el ciclo que pasa hasta quedar disponible para un nuevo comando.

En la Figura 4.10 se puede ver un comando para dibujar una figura, y luego del mismo al *frontend* desubscribiéndose del channel al cerrar el websocket.

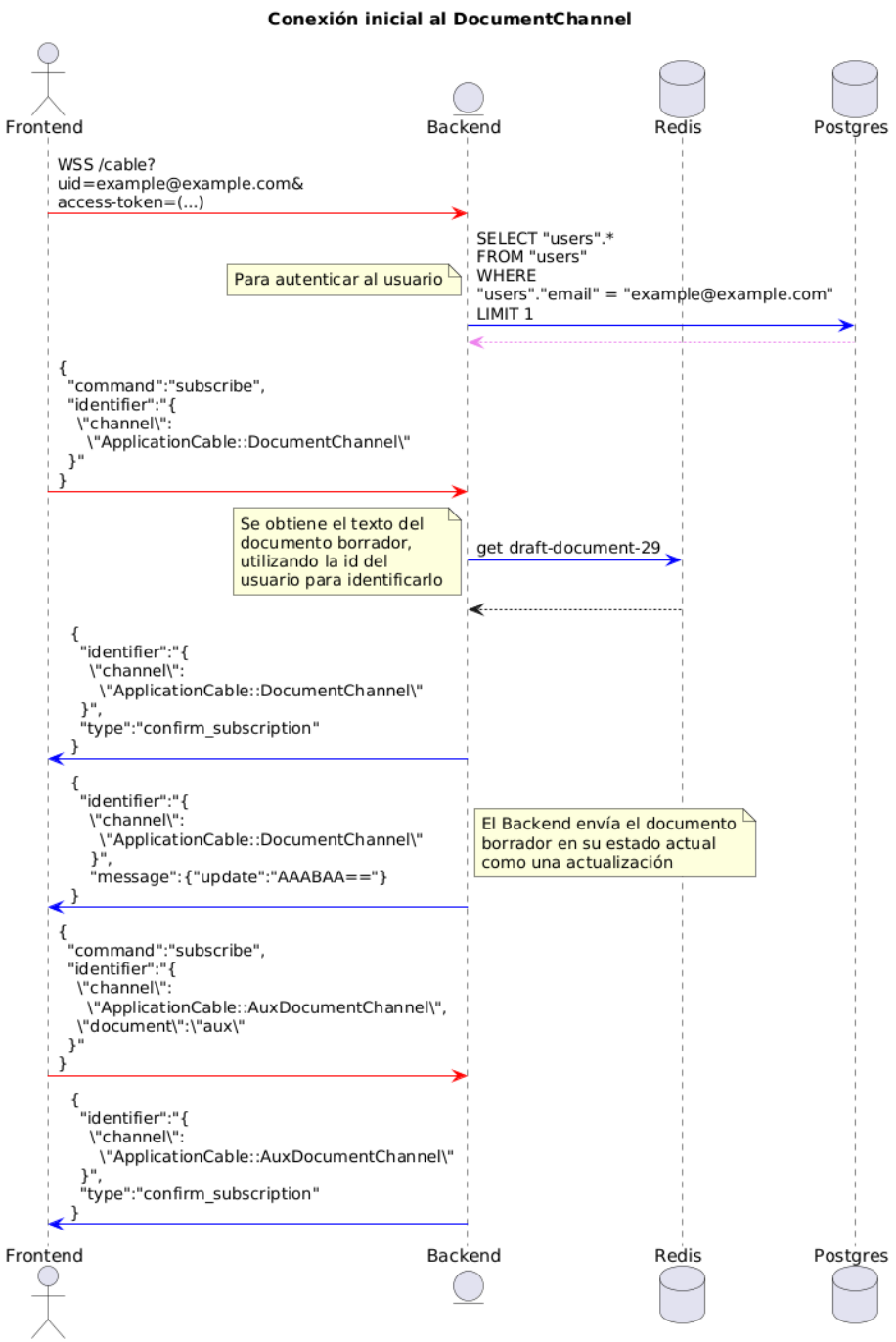


Figura 4.4: Diagrama de secuencia de un usuario conectándose al *DocumentChannel* para editar el documento borrador.

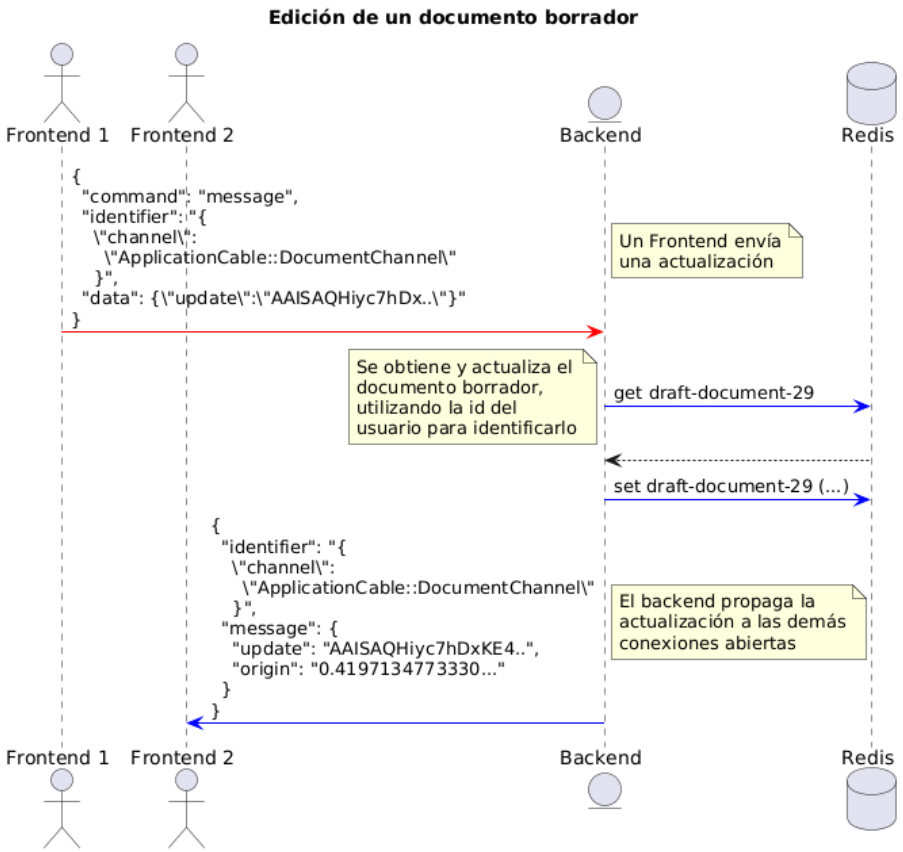


Figura 4.5: Diagrama de secuencia de un usuario con dos ventanas abiertas actualiza el cuerpo del documento borrador en una de ellas.

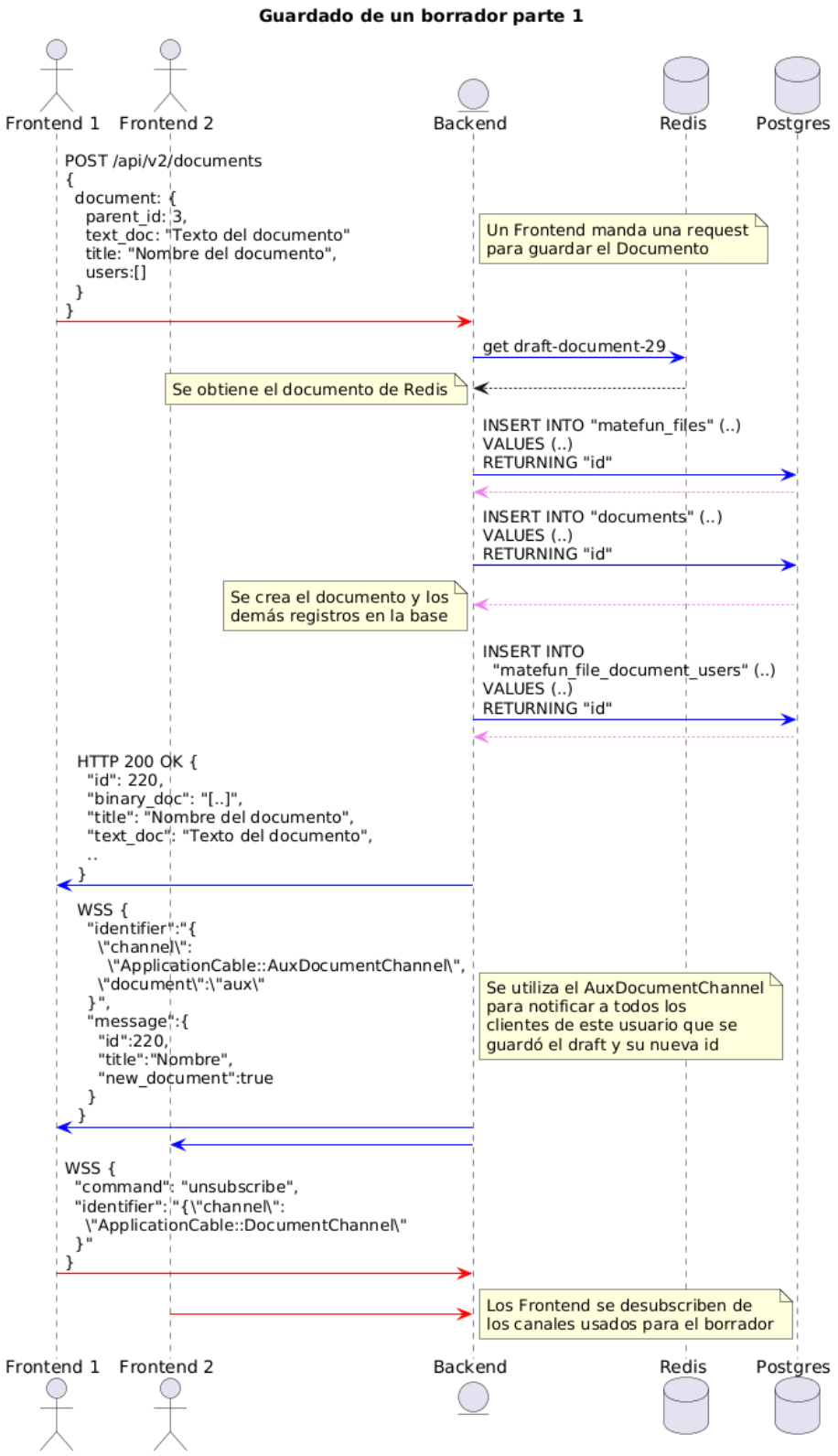


Figura 4.6: Diagrama de secuencia de un usuario con dos ventanas abiertas hace una *request* para guardar el documento en la capa de datos no volátil. Continúa en la Figura 4.7

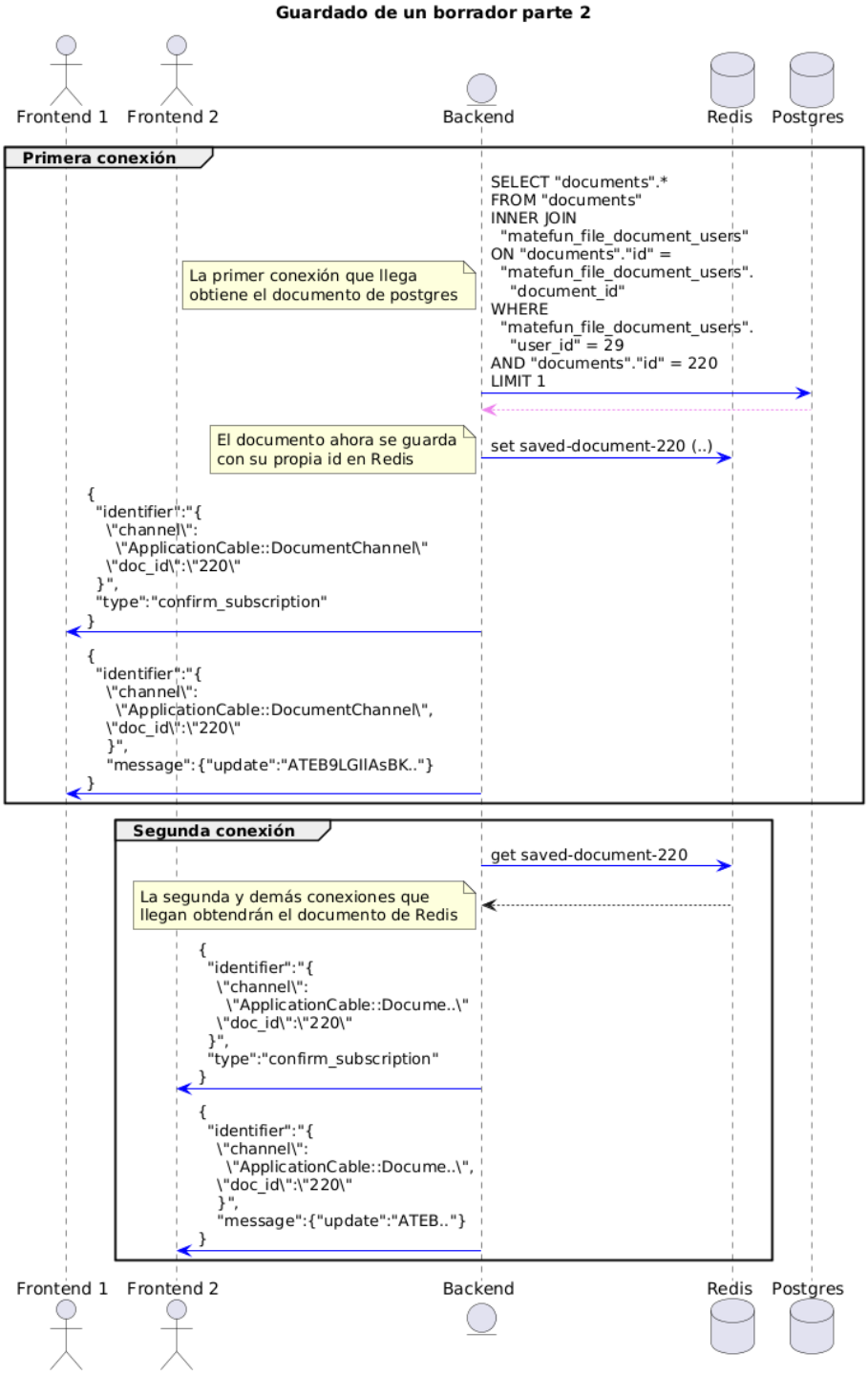


Figura 4.7: Diagrama de secuencia de una conexión al DocumentsChannel. Las distintas ventanas del cliente automáticamente se conectan a los *channels* para el nuevo documento ahora que está guardado en la capa de persistencia no volátil

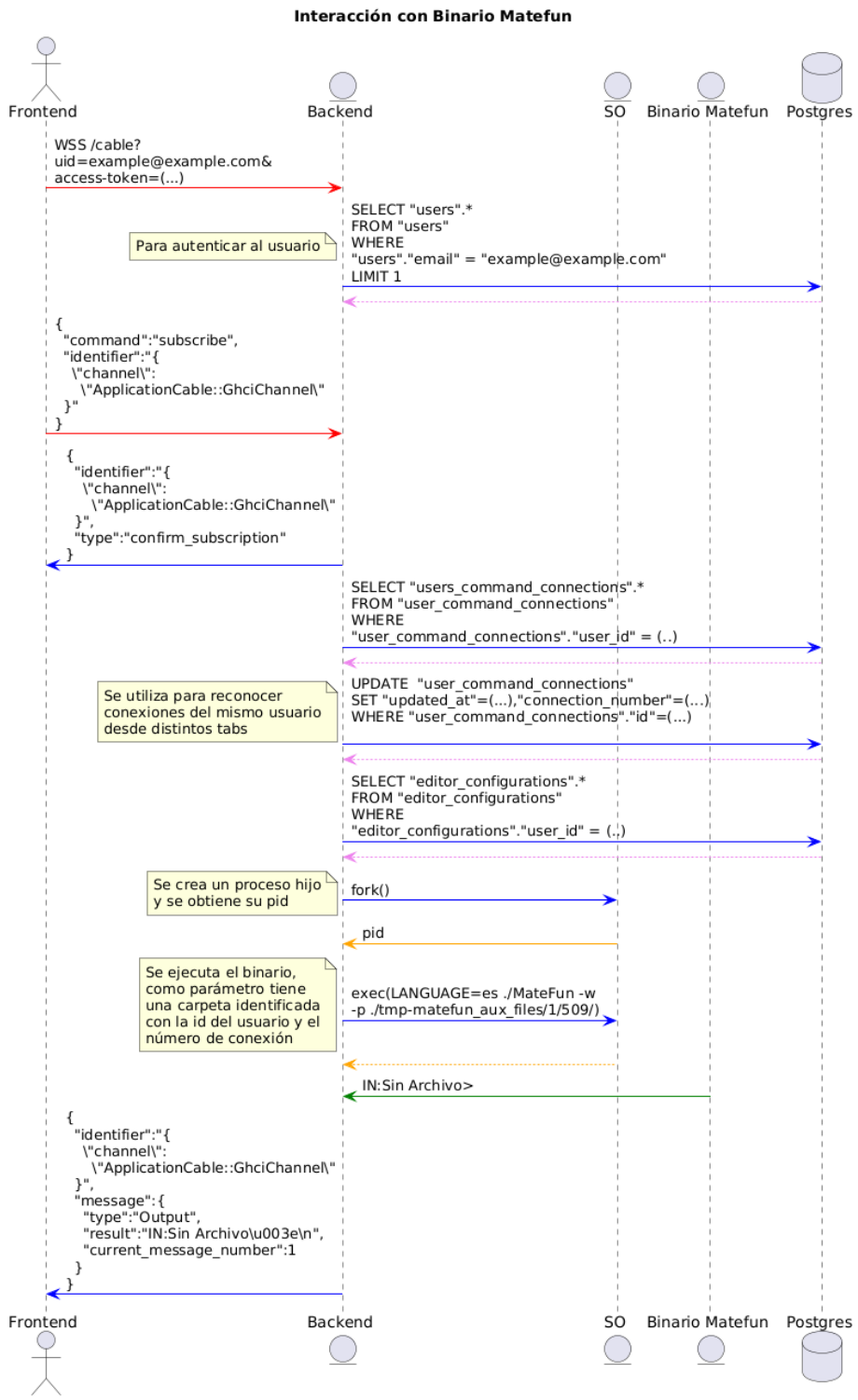


Figura 4.8: Diagrama de secuencia de la conexión inicial del *frontend* con ActionCable e iniciación del Binario Matefun

Figura 4.9: Diagrama de secuencia de la ejecución de un comando para obtener la tangente de 3 en el binario de Matefun

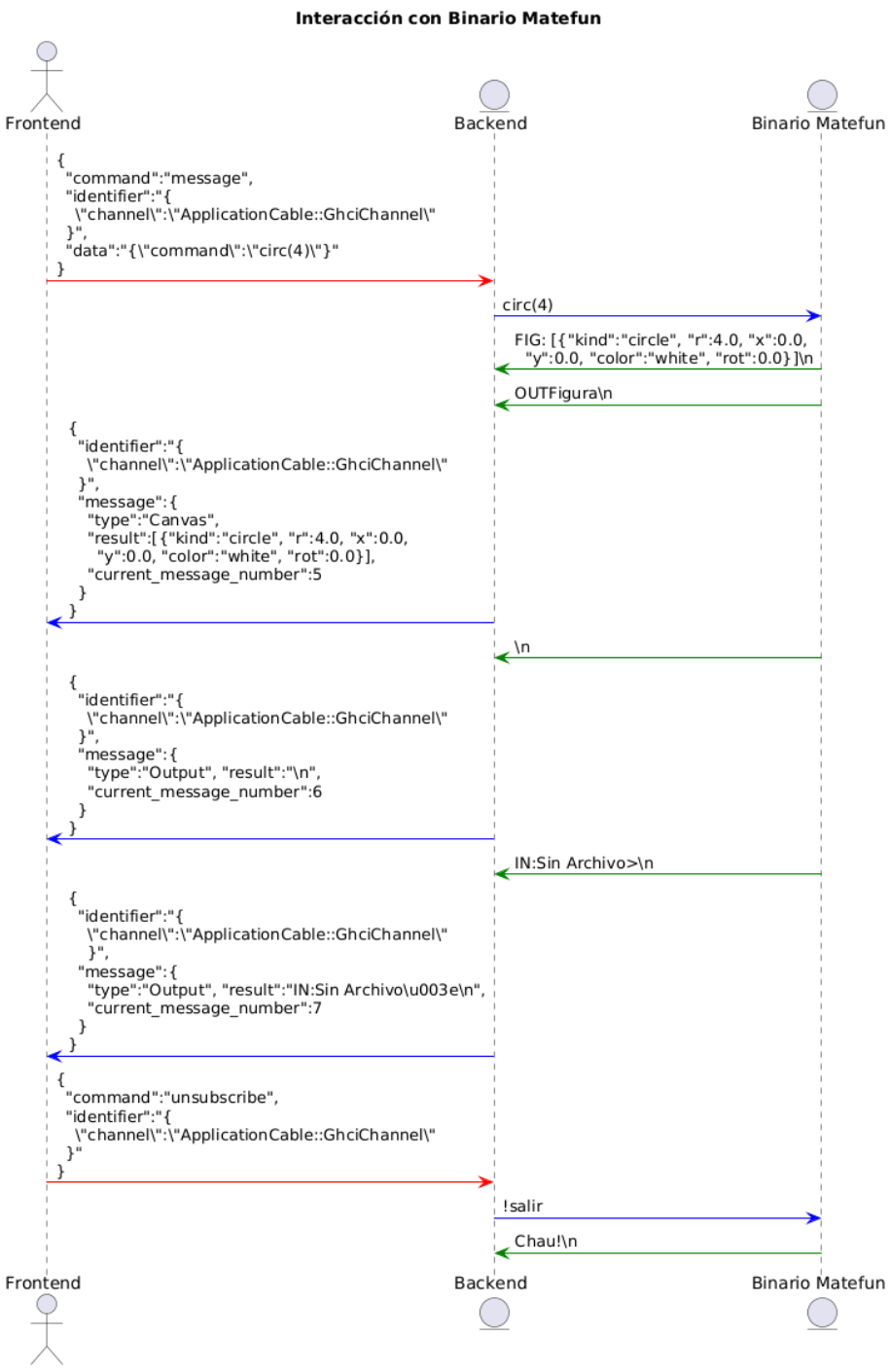


Figura 4.10: Diagrama de secuencia de la ejecución de un comando para obtener un círculo en el binario Matefun, y luego desconectarse del mismo

4.3.2.3. Creación de cuentas

En ediciones anteriores de MateFun los usuarios debían ser creados a mano en la base de datos o importados desde moodle. Para esta edición se cambió este sistema para que fuera más fácil y la web estuviera abierta a ser usada por más personas.

En esta nueva edición basta con enviar una *request* al RegistrationsController con email, nombre de usuario, y contraseña para crear una cuenta. Esto hará que el sistema mande un mail de confirmación de cuenta. En el mismo se encontrará un enlace para confirmar la cuenta, después de lo cual se podrá ingresar con su email y contraseña. Se puede ver en la Figura 4.11 un diagrama de secuencia que muestra esta interacción en más detalle.

4.3.2.3.1 Invitados

Una de las funcionalidades del sitio Matefun es la de poder probar el sitio con un usuario invitado. El usuario debe tener las mismas funcionalidades que un usuario normal, pero al hacer logout deben borrarse sus datos.

En versiones anteriores del proyecto esto se lograba teniendo un único usuario invitado en la base de datos, archivos de prueba para el mismo, y un Singleton que manejaba las sesiones en memoria. Una vez se mandaba un pedido de login con la cédula "invitado" y la contraseña "invitado" el sistema cargaba al usuario invitado en memoria, lo guardaba en un hash utilizando como clave un token generado que le era devuelto a la capa de presentación, la misma mandaba ese token en todas las *requests* y con el mismo se encontraba en el hash del Singleton los datos del usuario invitado de la sesión actual. De esta forma los datos de la base de datos nunca eran modificados sino que simplemente se cargaban en memoria al hacer login con los datos de invitado, y de ahí en más sólo se editaban los datos que estaban en memoria. Al hacer logout se borraba de memoria los datos relacionados al token correspondiente.

En esta nueva versión las sesiones de invitado funcionan de manera distinta. Hay un *endpoint* específico para crear una sesión de invitado, el cual crea un usuario en la base de datos el cual se marca como invitado. De ahí en más el *frontend* interactúa con el *backend* como si fuera un usuario normal, excepto que cuando hace logout lo hace a un *endpoint* específico para invitados, el cual destruye los datos de la base de datos que fueron creados por ese usuario invitado. También se creó una tarea diseñada para que sea ejecutada cada cierto período de tiempo, para borrar a los usuarios invitados que no han editado documentos en un cierto período de tiempo, para limpiar recursos del sistema si por alguna razón no hacen logout.

Como todo usuario necesita un username y este debe ser único, se tiene en la base de datos una tabla llamada guest_counters que contiene un único registro, en el mismo se guarda un número natural llamado counter. Cuando se crea un usuario invitado se utiliza para su username el string `"guest#{counter}"`, donde `counter` es sustituido por el valor actual del counter de la base de datos, el cual se incrementa luego en uno. Al llegar a un valor de 1 millón el mismo se vuelve

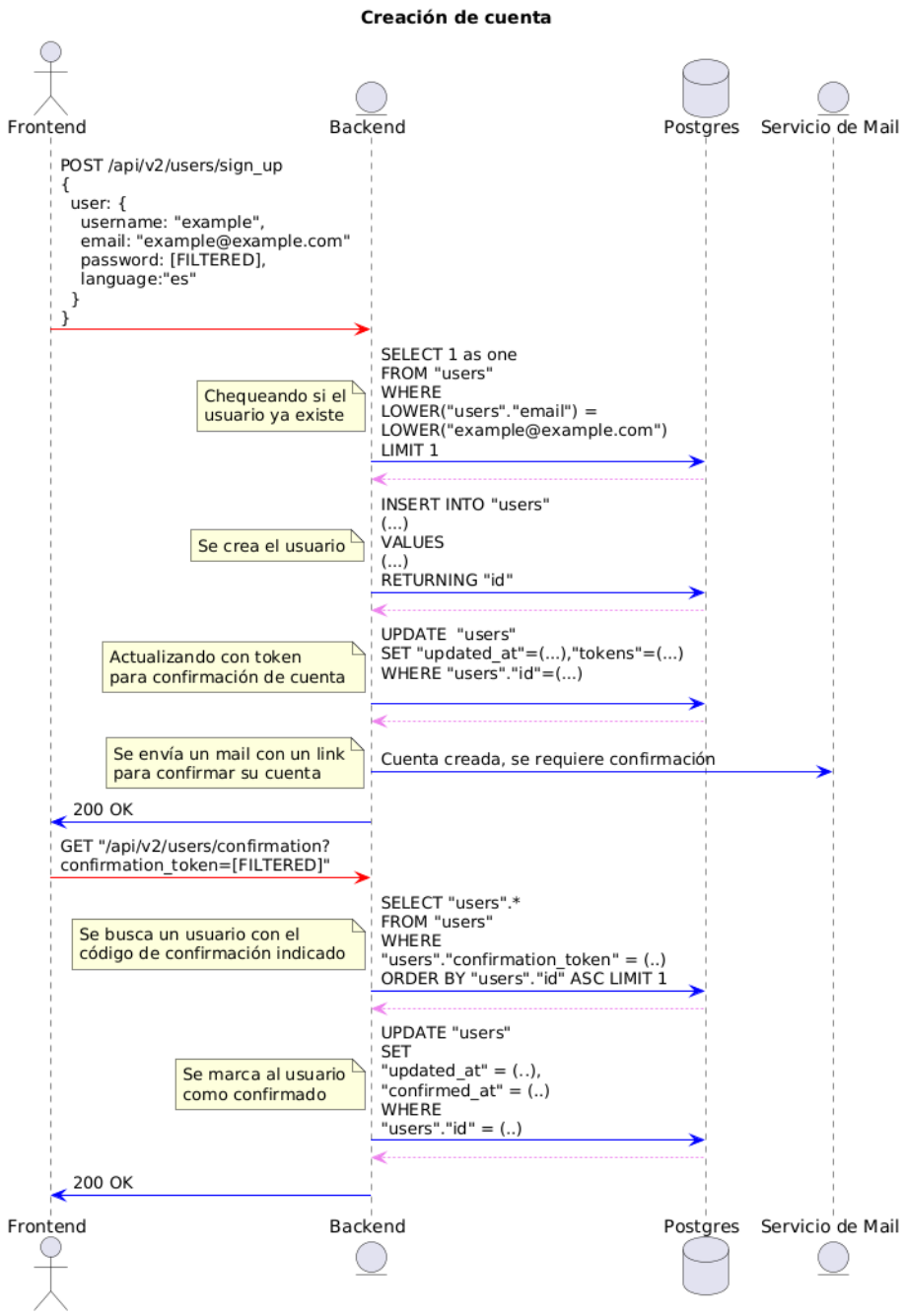


Figura 4.11: Diagrama de secuencia de la creación de una cuenta. En algunas consultas SQL se usa la elipsis para resumir las consultas y que se puedan leer bien.

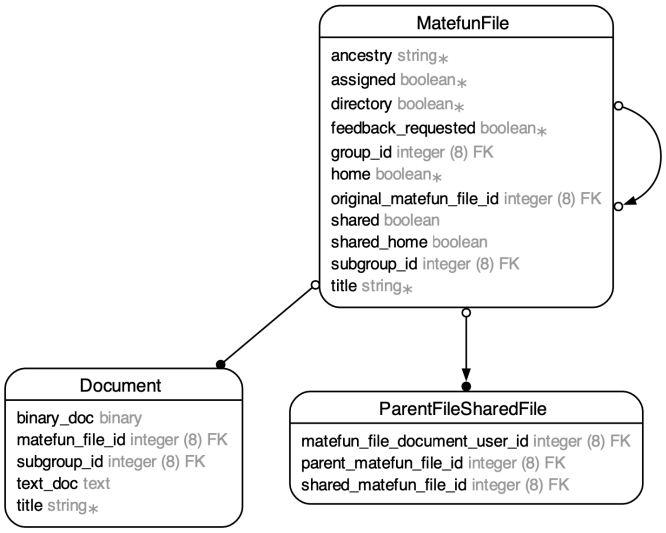


Figura 4.12: Relación entre MatefunFile y Document

a poner en 0 para evitar tanto que puedan intentar de crearse dos usuarios invitados con el mismo username, como que el counter llegue al valor máximo posible para un entero en la base de datos.

4.3.2.3.2 Comunicación por correos

Para el deploy final de MateFun se planea utilizar el servicio de emails de la Facultad. Para el deploy actual, para probar las nuevas funcionalidades, se utiliza SendGrid ya que tiene una fácil integración con Ruby on Rails y posee 100 emails gratuitos por mes que es más de lo necesario para este tipo de pruebas.

4.3.2.4. Representación interna de Archivos

En esta edición se separaron los documentos en dos clases distintas, MatefunFile que es la clase utilizada para la estructura de archivos y directorios, y Document que es el que efectivamente tiene el texto. En la Figura 4.12 se puede ver una representación de su relación y sus atributos.

Una de las razones de separación es que conceptualmente el archivo y el documento en sí son dos cosas distintas, la primera es un punto en una estructura de datos que puede tener un documento asociado o ser un directorio, mientras que la segunda son los datos mismos de un documento. Otra de las razones es que de esta forma se hace más sencilla una obtención más veloz de los archivos al recorrerlos en la vista de archivos.

Mientras que antes existía un único *endpoint* para traer los documentos, los cuales cargaban con ellos el texto de todos los documentos, ahora hay un *endpoint* que devuelve la estructura de archivos sin los documentos, y los mismos

sólo se cargan cuando efectivamente se elije uno. Esto hace que distminuya la cantidad de datos cargados a memoria de la base de datos, y también la cantidad de datos enviados desde la capa de negocios a la de presentación mejorando la eficiencia de todo el sistema.

4.3.2.4.1 Documentos

Como se puede ver en la Figura 4.12, los Documentos se componen simplemente por un binary_doc que contiene la representación interna en Y.js del documento, un text_doc que contiene la representación en string del documento, y el título del documento.

4.3.2.4.2 Archivos

Los archivos tienen varios booleanos para marcar distintas cosas:

- **assigned:** indica si el archivo es un archivo de grupo asignado como tarea por un miembro con permisos para esto.
- **directory:** indica si el archivo es un directorio (una carpeta). Si está en falso debe estar asociado a un documento.
- **feedback_requested:** para un archivo que pertenece a un grupo indica si el dueño del archivo pidió retroalimentación.
- **home:** indica si el directorio es la base de la jerarquía de archivos. Cada usuario tiene un home por defecto para sus archivos personales, y cuando se une a un grupo se crea un nuevo home para los archivos de grupo.
- **shared:** indica si el archivo es compartido con otros usuarios.
- **shared_home:** indica si este archivo es el directorio donde aparecerán por defecto todos los archivos que le sean compartidos al usuario.

Además de estos, el otro atributo de mayor importancia es el llamado ancestry. En este modelo estamos utilizando el patrón de diseño *materialized path* (Hansen, 2021) para la organización de las carpetas en la base de datos. El mismo consiste en guardar en un atributo (ancestry en este caso) de tipo string el id de todos los antecesores del archivo en la jerarquía de carpetas. En la Figura 4.13 se puede ver un ejemplo de una estructura de archivos con sus atributos de id y ancestry.

Utilizando este patrón e indexando la columna ancestry se pueden obtener las relaciones usualmente requeridas cuando se manejan árboles de manera eficiente sin dejar de usar tablas en SQL. En particular para nuestro proyecto utilizamos la librería ancestry[2] que también permite que automáticamente se ordenen en forma de hash de Ruby (estructura similar a un diccionario), y se puedan devolver al *frontend* ya ordenados como un árbol, lo cual facilita su posterior procesamiento.

[2] https://github.com/stefankroes/ancestry

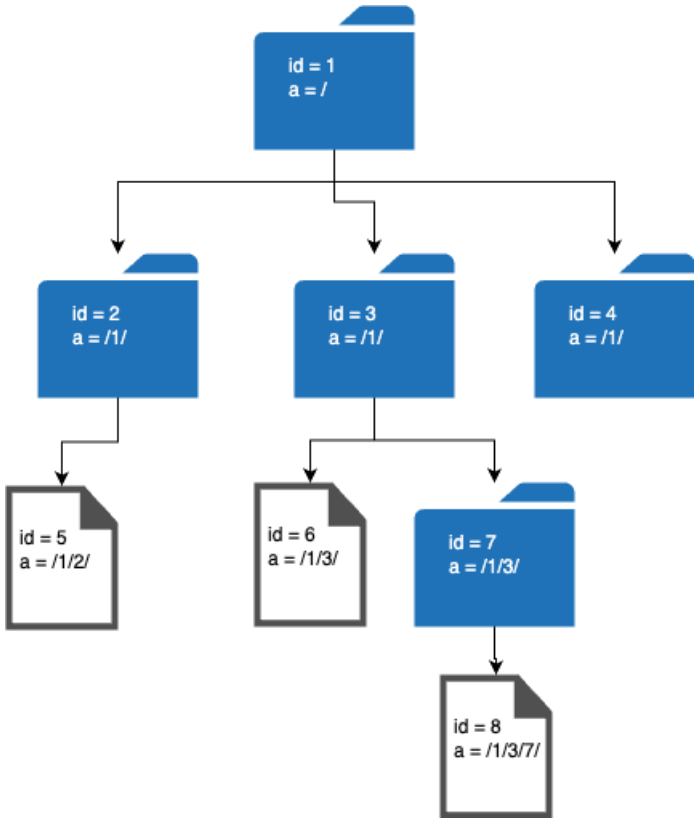


Figura 4.13: Estructura de archivos con los atributos de id y ancestry (etiquetado como a por temas de espacio)

4.3.2.4.3 Archivos compartidos

Si un archivo es compartido tendrá conflictos en su columna ancestry, por lo cual aparte de esto tenemos una relación many to many utilizando la tabla intermedia ParentfileSharedfile para los archivos compartidos. Un directorio puede tener muchos archivos compartidos, y un archivo compartido puede pertenecer a muchos directorios. Como sólo se comparten archivos y no directorios, una vez se tienen todos los directorios del usuario, se pueden obtener todos sus archivos compartidos por medio de dos consultas más, una para obtener los ParentFileSharedFile y otra para obtener los MatefunFile compartidos.

4.3.3. Capa de presentación

Para esta edición de MateFun se mantuvo el uso de Angular en la capa de presentación para el uso de la mayoría de la plataforma, y se utiliza Rails específicamente para el nuevo Admin. Los cambios y extensiones que se realizaron

en esta capa fueron hacer la sección de grupos utilizable y expandirla, actualizar la versión de Angular y el editor de texto para que pueda funcionar con edición colaborativa, y crear el flujo para que un usuario pueda registrarse.

4.3.3.1. Creación y manejo de cuenta

Para esta funcionalidad se crearon las páginas de registrarse, confirmar cuenta, y recuperar contraseña. Para saber los pasos específicos referirse a la Sección 4.3.2.3, la Sección 4.4.2 y al Manual de Usuario creado por Rodríguez (2025).

4.3.3.2. Actualización de la librería del editor de texto

Si bien el editor de texto que se usa en las versiones anteriores funciona bien con nuestra tecnología de edición colaborativa, la librería que se utiliza que es un wrapper sobre el editor de texto está desactualizada y no soporta la funcionalidad necesaria del editor. A su vez también limita la versión de Angular que se puede tener en el proyecto ya que sólo funciona con Angular 13 o inferior. Por estas razones tuvo que ser cambiada por otro wrapper del mismo editor que fuera más nuevo.

La nueva librería elegida requiere una versión más nueva de Angular, por lo cual se actualizó a Angular 15. También al ser un wrapper similar pero distinto al que usábamos previamente se tuvo que ajustar la lógica utilizada en el cursor del usuario activo, el coloreo de sintaxis, el indicador de errores y advertencias, y el manejo de temas en el editor.

4.3.3.3. Manejo de archivos

En las ediciones anteriores para mostrar los archivos, se traían todos los archivos que incluía el texto de los documentos ya que eran un solo tipo de registro. Los mismos venían en un array y para recorrer la estructura de directorios había que buscar los archivos hijos del directorio seleccionado cada vez se quería acceder a un directorio nuevo.

En esta nueva versión el *endpoint*retorna la estructura de archivos en un json donde cada hijo está directamente anidado en el padre en el atributo *children*. Esto hace que la búsqueda de archivos sea instantana al sólo tener que recorrer este json para encontrar los archivos buscados. A su vez los documentos con su texto tienen su propio *endpoint*y ya no vienen en la *requests* de archivos, por lo cual cuando se selecciona un archivo con un documento para leer se tiene que hacer una *request* para obtener su texto, esto sirve para evitar cargar el texto de todos los documentos a la vez al recorrer los archivos.

4.3.3.4. Compartir archivos

Se crearon nuevos menús para compartir archivos, en los mismos se puede agregar usuarios por su nombre de usuario y darles los roles deseados. Para saber más de cómo funciona referirse al Manual de Usuario creado por Rodríguez (2025).

Al editar un documento a la vez que otra persona es posible ver el cursor y los cambios que hace la otra persona en vivo gracias a la librería Y.js (*yjs*, s.f.).

4.3.3.5. Grupos y subgrupos

Se actualizó y dejó funcional la vista de Grupos, ampliándose para también tener subgrupos. En las mismas se pueden crear grupos, administrar sus usuarios y sus roles, crear subgrupos, asignar archivos, y pedir y dar retroalimentación. Para saber más cómo funciona referirse al Manual de Usuario creado por Rodríguez (2025)

4.3.3.6. Modales

El sitio consta de varios modales para actividades como crear archivos y carpetas, o moverlos dentro de la estructura de archivos. Estos fueron hechos con StencilJS y actualmente residen en un repositorio externo al repositorio de Matefun[3] y es publicado en npm[4] (node package manager) para su posible utilización en el stitio por un usuario al cual no se tiene acceso. Por esto y para evitar futuros problemas en los casos que se tuvo que cambiar modales del sitio se recrearon directamente dentro de la aplicación escritos en Angular, y lo mismo se hizo para los nuevos modales que se tuvieron que crear. De esta forma en futuras ediciones se podrán hacer cambios de manera más fácil en el mismo repositorio que se encuentra el resto del código de la aplicación. Se siguen utilizando los modales en StencilJS que no necesitaron cambios.

4.3.3.7. Admin

Una de las nuevas funcionalidades es una vista de administrador. Consiste en una página web donde se pueden crear usuarios "Admin" con acceso total a ver todos los demás usuarios, documentos, grupos y subgrupos de MateFun. La misma está hecha completamente en Rails por lo cual su capa de presentación también es Rails y no Angular, para implementarla se utilizó la librería ArcticAdmin[5]

4.3.4. Capa de persistencia no volátil

Por los cambios del sistema se vio oportuna la creación de una nueva base de datos, ya que la anterior no se acoplaba tan fácilmente al nuevo sistema por tener un modelado tan distinto debido a las nuevas funcionalidades, como se explica en la Sección 4.1.4. Para esto siguiendo la filosofía de Ruby on Rails de Convención antes de Configuración, se utiliza la base PostgreSQL que es la que viene por defecto con el framework.

Para interactuar con la base de datos se utiliza el ORM ActiveRecord de Rails, lo cual permite un fácil y rápido acceso a la misma, y también permite

[3] https://github.com/ncamera/matefun-components
[4] https://www.npmjs.com/package/matefun-components?activeTab=code
[5] https://github.com/cprodhomme/arctic_admin

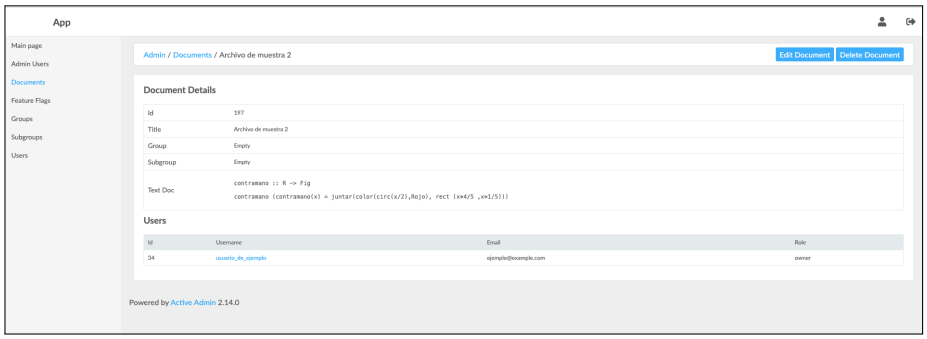


Figura 4.14: Vista de un documento en el Admin. Se pueden ver todos los datos del documento, incluso su contenido, y también los usuarios que están asociados al mismo con sus roles

que si en el futuro se quiere hacer un cambio de base de datos la interacción esté abstraida y el cambio sea más sencillo.

4.3.5. Capa de persistencia volátil

En esta edición del sitio se agregó una capa de persistencia volátil para el manejo de las conexiones por websocket y para permitir la rápida actualización de los documentos.

Para poder compartir los mismos datos entre distintas conexiones websockets al mismo *room*, Action Cable hace uso de Redis para su guardado y obtención, de esta forma si hay dos hilos distintos que estén respondiendo a conexiones que se unieron al mismo *room*, ambas pueden rápidamente acceder a los mismos datos, y el sistema entero puede mantener el estado de quién está conectado a qué *rooms* mientras las conexiones estén activas.

4.4. Seguridad

En esta sección profundizaremos en algunos puntos de seguridad tomados en cuenta al hacer la aplicación.

4.4.1. Passwords

Para la creación y mantenimiento de cuentas se utiliza la librería DeviseTokenAuth[6]. Esta librería es ampliamente utilizada y reconocida en la industria por su seguridad, es muy sencilla de usar una vez que ya está integrada, ya que hace todo automático sin más configuración, y posee una comunidad activa que la mantiene constantemente actualizada. La misma guarda las contraseñas

[6] https://github.com/lynndylanhurley/devise_token_auth

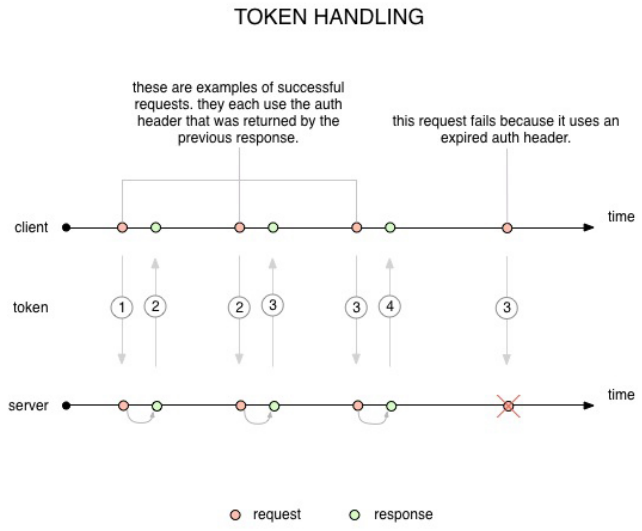


Figura 4.15: Ejemplo de manejo de tokens para autenticación con DeviseTokenAuth. En cada *request* se crea un nuevo token. En el último ejemplo el cliente manda un token expirado por lo cual su *request* no es aceptada. Imagen tomada de la documentación *devise-token-auth Conceptual Diagrams* (s.f.)

encriptadas utilizando el algoritmo Bcrypt Niels Provos (1999) para esto, por lo cual las mismas están seguras en caso de un robo de datos.

4.4.2. Token de autenticación

La librería DeviseTokenAuth funciona generando un token que el usuario debe mandar con su *request* para que sea validada. En cada *request* se crea un token nuevo, y el usuario debe mandar este nuevo token en su siguiente *request* como se ve en la Figura 4.15

En algunos casos se tienen que hacer varias *requests* en consecutivo, para este tipo de *request* en lote devise-token-auth utiliza un buffer de 5 segundos. Si recibe varias *requests* en un tiempo menor a 5 segundos aceptará el mismo token para todas como se ve en la Figura 4.16

4.4.3. Autorización

Para la autorización utilizamos la librería Pundit[7]. Se utiliza esta librería por su simplicidad en su uso, su seguridad con los métodos que ofrece para asegurarse que toda llamada pase por un chequeo de autorización, lo cual facilita que

[7] https://github.com/varvet/pundit

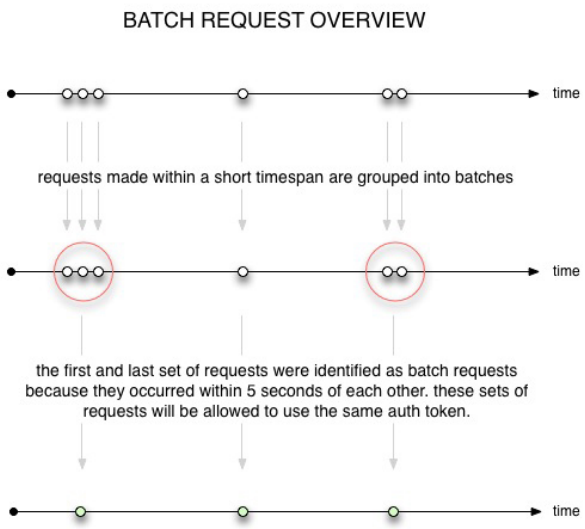


Figura 4.16: Ejemplo de manejo de tokens para autenticación con DeviseTokenAuth. Si se envían múltiples *requests* en menos de 5 segundos, se considera son parte de un único lote de *requests* y se acepta el mismo token para todas. Imagen tomada de la documentación *devise-token-auth Conceptual Diagrams* (s.f.)

los desarrolladores no se olviden de hacerlos, por tener una comunidad activa que la mantiene siempre actualizada, y por su probado uso en la industria. La misma funciona con *policies*, archivos donde escribimos la lógica de la autorización, y también nos brinda dos métodos que ejecutamos en cada *request* luego de la llamada a los controladores para asegurarnos que hayamos chequeado la autorización en algún momento dentro de los mismos.

Se crea un archivo policy para cada modelo, por ejemplo para documentos se puede crear la `DocumentPolicy`. La misma tendrá métodos con nombres que corresponden a los métodos de un controlador, de esta forma, tendrá métodos llamados `show?`, `create?`, y `update` y `destroy?`. Si se llama a la función `authorize` en un controlador, Pundit buscará la clase que corresponda al modelo que se está buscando autorizar, y la acción que corresponda a la acción del controlador desde donde se llama `authorize`.

Este proceso sirve para el caso que se quiera autorizar el acceso a un registro único de la base para los métodos `show`, `create`, `update` y `destroy`. En el caso del método `index` donde lo que se retorna es una colleción de registros en vez de uno solo, se utiliza un método `policy_scope` que acepta un conjunto de registros y acota los registros devueltos a sólo los que el usuario tiene autorización para ver.

Hay dos métodos que configuramos para que se ejecuten luego de las acciones de los controladores, dependiendo de la acción que se está llevando a cabo. `verify_authorized` que configuramos para que se ejecute en toda llamada a controlador excepto en los métodos index, verifica que hayamos llamado al método `verify` en algún momento dentro de la acción del controlador.

Como ejemplo, digamos que estamos resolviendo una llamada al método `show` del `DocumentsController`, y dentro de la acción llamamos al método `verify`. El mismo espera un objeto ActiveRecord que representa a un registro de nuestra base de datos. Luego se fija qué clase tiene y llama a la policy correspondiente. En este ejemplo lo llamamos con un documento particular, y llamará al método `show?` de la clase `DocumentPolicy` para chequear si el usuario autenticado tiene autorización sobre dicho documento.

Dentro de los métodos de una policy tenemos disponible el objeto del usuario que está autenticado como `user` y el objeto que pasamos como variable en la llamada al método como `record`. En este caso podemos chequear `record.users.include?(user)` que chequea que el usuario autenticado esté dentro de los usuarios asociados al documento, si esta función retorna `true` la llamada procederá sin problemas, si retorna `false` se retornará un 403 Forbidden a la llamada.

Algo muy similar ocurre con el método `verify_policy_scoped` que fue configurado para que se ejecute luego de la llamada al método index de cada controlador, el cual verifica que se haya llamado a `policy_scope` en algún momento dentro de la acción del controlador. `policy_scope` espera un `ActiveRecord::Relation` y la restringe a sólo los registros que el usuario autenticado tenga autorización para ver. Un `ActiveRecord::Relation` es una clase de `ActiveRecord` que contiene una llamada a la base de datos que tiene a más de un recurso. Por ejemplo, ante la llamada `Document.where(user_id: 3)` ActiveRecord retornará un `ActiveRecord::Relation` con todos los documentos que tienen `user_id` de 3, la cualidad que tiene

esta clase es que se le pueden concatenar más condiciones y seguirá retornando un `ActiveRecord::Relation`.

Como ejemplo para `policy_scope`, supongamos que en el `index` del `DocumentsController`, la primer línea de la función dice `documents = policy_scope(Document.where(id: params[:document_ids]))` donde `params[:document_ids]` es un arreglo de enteros que representan las ids de varios documentos. La expresión que está dentro del `policy_scope` haría una llamada a la base del estilo:

```
SELECT "documents".* FROM "documents" WHERE "documents"."id" IN (...)
```

Con las ids del parámetro. Pero al llamar a `policy_scope` se llama al método `resolve` de la `DocumentPolicy::Scope` que es así:

```
def resolve
    user.documents.and(scope)
end
```

Donde `scope` es el parámetro que recibe la función. Esto transforma la llamada a:

```
SELECT "documents".*
FROM "documents"
INNER JOIN "matefun_file_document_users" ON "documents"."id" = "
    matefun_file_document_users"."document_id"
WHERE "matefun_file_document_users"."user_id" = .. AND "documents".
    "id" IN (...)
```

restringiendo nuestra query a que sólo traiga documentos del usuario correspondiente.

Si no se llama a los métodos esperados `authorize` o `policy_scope`, se lanzarán las excepciones `Pundit::PolicyScopingNotPerformedError` y `Pundit::AuthorizationNotPerformedError` respectivamente, lo cual hará que la capa de negocios retorne un error 500 a la capa de presentación.

4.4.4. Inyección de SQL

Inyección de SQL es cuando se mandan datos en una *request* con la intención de que sean ejecutados por la capa de persistencia no volátil y así obtener o cambiar datos a los cuales no se debería tener acceso.

Como un ejemplo, si tuviéramos la siguiente consulta: `User.find_by('username = '#{params[:unsername]}')` el atacante podría mandar en el parámetro username `'' OR '1'='1'`, lo cual haría que la consulta quede: `SELECT * FROM users WHERE username = 'a' OR '1'='1' LIMIT 1`, que retornaría el primer registro de la tabla automáticamente.

Para solucionar este tipo de ataques tomamos dos medidas. Por una parte, tenemos una lista de parámetros permitidos en todas las *requests* de creación y actualización de registros. Esto quiere decir que si un ataque intenta mandar campos extra estos serán ignorados.

Por otra parte Rails tiene formas de evitar la inyección para evitar que se haga el tipo de consulta mostrada en el ejemplo anterior. Todos los métodos `User.find(id)` automáticamente escapan los caracteres `'`, `"`, y `NULL`. Si tenemos una consulta como la del ejemplo, también son automáticamente escapados si

la consulta se hace de la siguiente manera: `User.find_by('username = ?', params[:username])` en donde `?` será remplazado por el valor de `params[:username]` pero con los caracteres mencionados más arriba escapados.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusión

El presente proyecto de grado tuvo como objetivo principal modernizar y extender las funcionalidades de la plataforma educativa Matefun, con el fin de convertirla en una herramienta más accesible y colaborativa. La motivación central radicó en la necesidad de adaptar la plataforma a las dinámicas actuales de enseñanza de programación y matemáticas en la educación media, donde la colaboración, la autogestión y la interacción en tiempo real se han vuelto aspectos fundamentales.

Para alcanzar estos objetivos, se realizó un análisis de plataformas LMS, tecnologías de edición colaborativa y herramientas de desarrollo modernas, seleccionando aquellas que mejor se ajustaban al contexto de Matefun. Se incorporaron tecnologías como Y.js para sincronización en tiempo real mediante CRDT, Redis para persistencia en memoria y Ruby on Rails como framework principal para el *backend*, junto con Angular y StencilJS en la interfaz.

Entre los principales logros del proyecto se destacan la implementación de un sistema de edición colaborativa en tiempo real, la gestión de cuentas de usuario, la creación y administración de grupos y subgrupos, y el diseño de una arquitectura modular que permite mayor escalabilidad y mantenibilidad del sistema. Estas funcionalidades no solo amplían el alcance potencial de Matefun en el aula, sino que también habilitan su uso en contextos más generales, como tutorías virtuales o proyectos de colaboración académica.

El desarrollo del sistema planteó desafíos técnicos importantes, particularmente en la integración de tecnologías diversas y en la sincronización entre múltiples usuarios en entornos concurrentes. Sin embargo, dichos desafíos fueron superados con éxito, permitiendo validar la viabilidad técnica del enfoque propuesto.

En conclusión, este proyecto ofrece una herramienta renovada, más cercana

al estándar de una plataforma web moderna. La creación de cuentas es más sencilla, y es posible trabajar en equipo y con docentes de manera más flexible. Esto puede representar un aporte al ecosistema educativo digital uruguayo, y tiene el potencial de mejorar más con futuros desarrollos y extensiones, que se detallan en la siguiente sección.

5.2. Trabajo futuro

En esta sección plantearemos posibles trabajos a futuro para la plataforma. Estos son tanto trabajos que no se hicieron por no entrar en el alcance del proyecto y se piensa que podrían ser útiles considerar a futuro dependiendo el camino que se le quiera dar al sitio.

5.2.1. Mejora a la interfaz de usuario del sitio

Hay algunas faltas notorias en la interfaz de usuario del sitio, aquí se presentan algunas generales:

- Por una parte se podrían mejorar algunos mensajes de error donde la web sólo señala que hubo un error pero no cuál, si bien el *backend* suele mandar errores específicos.
- Sería muy bueno que cuando una persona desea registrarse y escribe un nombre de usuario el sistema señale inmediatamente si está disponible o no en lugar de tener que intentar de crear la cuenta primero para luego recibir el mensaje de error.
- Si bien es posible que un usuario Admin cambie el email o el nombre de usuario de un usuario, sería también bueno agregar estas capacidades en la configuración del usuario para que puedan hacerlo ellos mismos sin tener que contactar a un administrador.
- Los emails de confirmación y de reset del password son texto plano muy básico, una mejorar posible sería crear un diseño más agradable para los mismos.

A continuación se presentan otras más centradas en la sección de grupos del sitio:

- Un rediseño entero de la misma para que sea más amigable sería ideal. La forma que está diseñada ahora con tabs es muy limitante si se quieren agregar más funcionalidades.
- La url no cambia al navegar por esta sección, por lo cual si uno está dentro de un grupo y recarga la página vuelve a la página de selección de grupos, una mejora que agregaría usabilidad sería que la url fuera cambiando al navegar por la sección de grupos.

- Poder crear grupos a partir de un CSV y no tener que escribir usuario por usuario.
- La retroalimentación de archivos es de lo que más trabajo necesita aún, una gran mejora sería crear una página para ver todos los pedidos de retroalimentación disponibles y no tener que ir usuario por usuario adivinando cuál puede tener un pedido y cuál no.
- Relacionado al punto anterior, otra posible mejora sería agregar notas de texto en las retroalimentaciones, separadas de las que se escriben en el código mismo.
- Al compartir archivos dentro de grupos, agregar la lista de miembros del grupo para elegir en vez de la forma actual que es igual a la usual y requiere saber sus nombres de usuario e ingresarlos de a uno.
- La funcionalidad de asignar tareas también es bastante limitada, en este momento lo único que hace es crear una copia del archivo en los usuarios/-subgrupos elegidos. Esto funciona bien de base pero al ser un archivo sin diferencia de ninguno otro, no hay forma de tener un grupo lógico de asignaciones para delimitar qué es una entrega de una misma tarea de varios usuarios y qué no. Una posible mejora sería hacer una subclase de MatefunFile que sea un Assignment específicamente y agregar funcionalidades a la misma, de esta forma sería más fácil para un docente ver todas las entregas de una asignación, ver quién falta entregar, mandar un archivo con instrucciones que vayan separadas del mismo, y posiblemente dejar notas de manera más sencilla y menos intrusiva al alumno al corregir.
- En las asignaciones, se puede agregar la posibilidad de ver el archivo como estaba originalmente al ser asignado, por si un alumno lo modifica de una manera que haga que le sea imposible continuar con la tarea.
- Una característica que era mejor en versiones anteriores es que al asignar un archivo no se creaban las copias directamente como se hace ahora, sino que se creaba una asignación, y la copia del texto en sí se creaba en el momento que el usuario abría el archivo por primera vez. Si bien esto puede hacer que una asignación demore algo más en abrirse la primera vez, también implica un ahorro de recursos ya que se crearán menos copias de archivos grandes innecesariamente.
- Podría haber forma de autogenerar subgrupos, o generarlos a partir de un CSV.
- Poder ver las entregas de un subgrupo cuando se está viendo las entregas de un usuario particular que pertenece al mismo.
- Poder acceder a los subgrupos a los que pertenece un usuario cuando se están viendo los datos del usuario dentro del subgrupo.

- El archivo actual de la sección de grupos es un archivo enorme que tiene grupos, usuarios, archivos y subgrupos todos en el mismo lugar. Es demasiado grande y debería ser reescrito de una mejor manera separándolo más en más componentes más chicos, mantenibles, y flexibles.

Sería positivo que alguien que sepa efectivamente de diseño web, alguien como un estudiante de la Licenciatura en Diseño de Comunicación Visual de la Universidad de la República de Uruguay, haga un diseño con un uso fluido y atractivo para el público objetivo del sitio. Si bien todos quienes aportamos a este sitio seguro hicimos lo mejor que pudimos, hay una razón por la cual Diseño es una carrera separada a Ingeniería, y seguro podría salir un diseño mucho mejor, más usable y amigable de esta forma.

5.2.2. Notificaciones

Una funcionalidad que se planificó hacer pero finalmente no entró en el alcance es la de notificaciones. En este momento una de las características que no tiene el sitio y que sería muy deseable es alguna forma de saber si te comparten un archivo, si te agregan a un grupo, si te asignan una tarea, si alguien pide retroalimentación, etc. Para que un usuario sea notificado de estas acciones, es necesario al día de hoy utilizar alguna herramienta externa al sitio para notificar a la persona, o que el usuario mismo haga el chequeo a mano de cada cosa.

Un sistema de notificaciones no es particularmente difícil de hacer desde el punto de vista lógico. Sólo hace falta un modelo de notificaciones donde se guarden las mismas, y un servicio de notificaciones al que se pueda llamar cada vez que se quiera notificar algo con los mensajes, modelos asociados, y usuarios a los que se desea notificar.

Para que los usuarios reciban las notificaciones el sistema más sencillo que se puede hacer es que reciban un email con el mensaje y posiblemente un enlace para acceder a la parte del sitio referente a la notificacion. También sería muy bueno tener una sección de notificaciones en el sitio que tenga un indicador cuando hay nuevas notificaciones y se pueda ver un historial de las mismas, y que al hacer click en una se pueda ir a la sección del sitio correspondiente. Esto haría de por sí al sitio muchísimo más utilizable incluso con lo tosca que es la sección de grupos desde el punto de vista de uso, y consideramos sería más fácil de desarrollar que hacer un rediseño de esa sección para una primer instancia de mejora.

5.2.3. Docker

Otras de las cosas que no entraron en el alcance del proyecto es Dockerizar la aplicación. Docker[1] es una aplicación que permite correr fácilmente contenedores en distintas máquinas sin tener que estar configurando cada una por separado. También permite que si el servidor se cae por alguna falla se vuelva a

[1] https://www.docker.com/

levantar de manera automática. Esto sería muy útil para tener deploys más fáciles, que no haya necesidad de levantar los servidores manualmente si se caen por alguna razón, y que nuevos desarrolladores puedan ejecutar aún más fácilmente la aplicación.

5.2.4. Accesibilidad

Como se vio en la sección de antecedentes Moodle es un sitio que cumple las guías de accesibilidad AA de la Web Accessibility Initiative. Esto es particularmente importante para que el acceso al sitio esté disponible para todo el que quiera usarlo. Es posible no sea una tarea fácil en particular el editor colaborativo, pero analizar plataformas como google docs y otras que puedan mostrar gráficas puede servir, y lograr que al menos el flujo de importar archivos hechos en otros programas y ejecutarlos es accesible ya sería un gran paso para que el sitio fuera más accecible para todos.

Accessibility Insights[2] es una buena herramienta que puede servir para analizar la accecibilidad del sitio, y JAWS[3] como un lector de pantalla muy popular.

5.2.5. Persistencia de archivos en caso de que sean borrados

En la versión actual de la aplicación si un dueño borra un archivo el mismo desaparecerá de la cuenta de todos los usuarios con quien estaba compartido. Lo mismo ocurre a mayor escala si el dueño de un grupo borra el mismo, en cuyo caso todos los archivos que pertenecían al grupo serán borrados, incluso los que no le pertenecían al dueño del grupo en sí.

Sería muy bueno agregar una funcionalidad para que se guarde una copia de los mismos en los archivos personales del usuario si esto ocurre para que no pierda la información, y que ante un error del dueño de un grupo o de archivo no haya una pérdida total de datos.

Podría investigarse también tener una papelera similar a la de Google Docs donde los archivos eliminados vivan por un determinado tiempo antes de ser efectivamente eliminados, para mayor resiliencia ante este tipo de errores.

5.2.6. Reescritura de la capa de presentación

En esta sección hay dos cosas que serían positivas para el sitio.

Rehacer la capa de presentación de una manera mejor estructurada, ya se mencionaron algunos problemas en la Sección 5.2.1. Otro sitio que ha crecido demasiado y debería ser reestructurado sería la pantalla principal de Matefun, que consiste de un sólo archivo de más de 1500 líneas, donde es incluso posible que haya funciones que ya no se utilizan, y es muy difícil encontrar un flujo específico por tener tantas cosas agrupadas en el mismo.

[2] https://accessibilityinsights.io/docs/web/overview/
[3] https://www.freedomscientific.com/products/software/jaws/

5.2.7. Edición de documento sin websockets

En esta edición siempre que un usuario edita documentos se utiliza automáticamente la misma conexión por websockets ya sea si está editando con otros usuarios o si está editando documentos a los que tiene acceso únicamente él. Sin embargo para el caso que el usuario está editando un documento que no ha compartido con otros usuarios, podría usarse el anterior modelo de edición donde se usan llamadas https comunes para guardar el documento. Esto podría liberar recursos de *backend* ya que no hay una comunicación constante que resulta innecesaria para estos casos, y podría ser completamente transparente para el usuario a su vez.

5.2.8. Posibildad de compartir la terminal

La posibilidad de compartir la terminal sería muy buena cuando un usuario quiera mostrarle a otros el uso de la misma, o simplemente al trabajar en conjunto para que cada usuario no tenga que probar cosas por su cuenta, lo cual también consume más recursos del sitio que si varios usuarios trabajando a la vez tuvieran una sola instancia del binario Matefun ejecutándose.

Al ser la interacción con la terminal a base de comandos, los cuales son atómicos y llegan en un orden específico, es factible que implementar esta funcionalidad no requiera de y.js y se pueda implementar directamente con el *backend* asignando el orden de instrucciones a medida que llegan al mismo.

También tener la posibilidad de que la terminal sea compartida como sólo lectura para el caso que un docente quiera enseñarles a los alumnos podría resultar útil.

5.2.9. Chat

Una mejora que los tutores marcaron como tentativa es tener un chat cuando se están editando documentos para poder comunicarse con los demás miembros del grupo, e incluso posiblemente con el docente.

Si bien hoy en día hay muchas formas de comunicarse con otras personas, y para el caso de los alumnos de secundaria que se conocen personalmente seguro no hay problema en conseguir otro *channel* para hablar, tenerlo todo centrado en el sitio sería de mucha ayuda, y también ayudaría a otros casos de uso donde personas que no se conocen personalmente terminen editando el mismo documento.

5.2.10. Compartir directorios

En este momento no es posible compartir directorios. Si bien se buscó implementar, esto se hizo con el proyecto un poco más avanzado y se encontró incompatibilidad con algunas decisiones que ya se habían tomado.

El problema yace cuando dos usuarios se comparten entre ellos dos directorios, y cada uno mueve el directorio que le fue compartido adentro del directorio que compartió, generando un loop de directorios. Esto conlleva a dos problemas.

Por una parte como está diseñada la interacción entre las capas de negocio y presentación donde se trae toda la estructura de directorios en forma de árbol, generaría un árbol infinito. Una solución posible es en vez de traer en un *end-point*de una todo el árbol de directorios, traer sólo los archivos y directorios que pertenezcan a un directorio particular, y quizá los hijos de los mismos para una respuesta más veloz cuando un usuario entra en un directorio. De esta manera si se genera un loop no ocurre nada porque siempre habrá sólo tres niveles cargados a la vez de la estructura (nivel padre, nivel actual, y nivel hijos). Otra posibilidad sería traer sí el árbol, y que estos loops se resuelvan buscando el directorio hijo en todo el árbol si no se encuentra anidado en el mismo, es decir, habría que detectar los loops y al armar la estructura de árbol evitarlos y traer una referencia al id por ejemplo del directorio que causa loop.

El otro problema posible es en cómo está siendo guardado en este momento en el *backend*. En este momento se aprovecha de la librería Ancestry que utiliza el patrón materialized path para guardar la estructura de árbol, y si bien hay archivos con más de un padre, estos son fácilmente administrados porque cada nodo del árbol puede tener sólo un nivel de hijos que sea compartido, por lo cual se pueden traer todos de la base con tan solo dos consultas, sin embargo si se desea que directorios puedan ser compartidos, genera el problema de que pueda ser necesario varias consultas por cada directorio que sea compartido, los cuales pueden a su vez tener más directorios compartidos adentro.

5.2.11. Tareas con tiempo límite para pruebas

Como se vio cuando se hablaba de Moodle, el mismo tiene la posibilidad de tener tareas con tiempo límite. Esto podría ser una funcionalidad útil para docentes que quieran usar Matefun para poner pruebas a alumnos.

5.2.12. Multi-tenancy

Multi-tenancy es una técnica de arquitectura de software que consiste en tener una sola instancia de software ejecutándose para servir a diferentes clientes u organizaciones Krebs y cols. (2012). Si bien es una sola instancia, a efectos prácticos a cada institución le parecerá que tiene su propia instancia con datos separados y aislados de los demás.

En algún momento podría ser útil para tener algo más similar a un LMS, donde cada institución pueda tener sus propios usuarios con nombres específicos para esa institución, y sus propios grupos. Esto podría abrir la página a tener una lista de grupos por ejemplo y que sea más fácil unirse, y otras funcionalidades que sería impráctico o imposible de implementar en un sitio único. También podrían haber Admins especiales con acceso para ver únicamente los usuarios, documentos y grupos de su propia organización.

Una forma posible de lograr esto sería con la librería apartment[4], aunque incluso con el uso de la librería es seguro un trabajo no menor de implementación.

[4] https://github.com/rails-on-services/apartment

Referencias

Action cable overview - async adapter. (Accessed: 2025-06-20). https://guides.rubyonrails.org/action_cable_overview.html#async-adapter.

Actioncable::subscriptionadapter::async. (Accessed: 2025-06-20). https://github.com/rails/rails/blob/aa66aece44f191e6330c04ce4942c3edf31d85d7/actioncable/lib/action_cable/subscription_adapter/async.rb#L7.

Anton Zagorskii. (2018). *Operational transformations as an algorithm for automatic conflict resolution.* https://medium.com/coinmonks/operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447. (Accessed: 2025-05-01)

C.A. Ellis, S. G. (1989). Concurrency control in groupware systems. *ACM SIGMOD Record*, *18*, 399-407.

crdt-benchmarks. (s.f.). https://github.com/dmonad/crdt-benchmarks. (Accessed: 2025-06-01)

devise-token-auth conceptual diagrams. (s.f.). https://devise-token-auth.gitbook.io/devise-token-auth/conceptual. (Accessed: 2025-06-05)

Edtech breaktrhough award winners. (s.f.). https://edtechbreakthrough.com/edtech-2023-winners/. (Accessed: 2025-06-18)

Ellis, R. K. (2009). *A field guide to learning management systems.* Alexandria, NV, USA: American Society for Training Development (ASTD).

Getting started with rails. (s.f.). https://guides.rubyonrails.org/getting_started.html#rails-philosophy. (Accessed: 2025-06-10)

Gonzalo Cameto, M. M. (Accessed: 2025-06-10a). *Matefun (ide web + lenguajge funcional específico) - documento de arquitectura y diseñno.* https://gitlab.fing.edu.uy/-/project/1651/uploads/41c0624b45ad9d14d855c08262cc31cd/arquitectura-y-diseno__1_.pdf.

Gonzalo Cameto, M. M. (Accessed: 2025-06-10b). *Matefun (ide web + lenguajge funcional específico) - informe final.* https://gitlab.fing.edu.uy/

-/project/1651/uploads/f39c9814edf617c0ecc228b35d88afbf/informe-final.pdf.

Hansen, T. (2021). The materialized path technique: Tree structures for relational database systems. *Data Persistence*. (Accessed: 2025-07-02)

Hill, P. (s.f.). *Academic lms market share: A view across four global regions.* https://eliterate.us/academic-lms-market-share-view-across-four-global-regions/. (Accessed: 2025-06-18)

Krebs, R., Momm, C., y Kounev, S. (2012). Architectural concerns in multi-tenant saas applications. En *Proceedings of the 2nd international conference on cloud computing and services science - closer* (p. 426-431). SciTePress. doi: 10.5220/0003957604260431

Li, D., y Li, R. (2010, 01 de Feb). An admissibility-based operational transformation framework for collaborative editing systems. *Computer Supported Cooperative Work (CSCW)*, *19*(1), 1-43. Descargado de https://doi.org/10.1007/s10606-009-9103-1 doi: 10.1007/s10606-009-9103-1

Lio, R. D. (s.f.). *Can postgres replace redis as a cache?* https://medium.com/redis-with-raphael-de-lio/can-postgres-replace-redis-as-a-cache-f6cba13386dc. (Accessed: 2025-06-09)

Market progression and leaders. (s.f.). https://listedtech.com/wp-content/uploads/2023/06/lms-hed-northamerica-portal.png. (Accessed: 2025-06-18)

Nicolaescu, P., Jahns, K., Derntl, M., y Klamma, R. (2016, 11). Near real-time peer-to-peer shared editing on extensible data types. En (p. 39-49). doi: 10.1145/2957276.2957310

Niels Provos, D. M. (1999). A future-adaptable password scheme. En *Proceedings of the 1999 usenix annual technical conference* (p. 81-91). Santa Clara, CA, USA: USENIX.

Okolo, D. (2024). *Ruby on rails: Mvc and you!* https://dev.to/dumebii/model-view-controller-in-rails-a-deep-dive-into-the-mvc-architecture-4oi1. (Accessed: 2025-07-16)

Rodríguez, D. (2025). *Matefun colaborativo.* https://gitlab.fing.edu.uy/matefun/Frontend/-/wikis/uploads/0ce2e45eb1b4498c1bfb54f9b70159a9/Manual_de_usuario_-_Matefun_Colaborativo.pdf. (Accessed: 2025-07-02)

Shapiro, M., Preguiça, N., Baquero, C., y Zawirski, M. (2011). Conflict-free replicated data types. En X. Défago, F. Petit, y V. Villain (Eds.), *Stabilization, safety, and security of distributed systems* (pp. 386–400). Berlin, Heidelberg: Springer Berlin Heidelberg.

Watkins, B. (s.f.). *Duke university selects canvas lms, additional instructure learning platform solutions to enable accessible, high-quality learning.* https://www.instructure.com/press-release/duke-university-selects-canvas-lms-additional-instructure-learning-platform-solutions. (Accessed: 2025-06-18)

Why choose redis? (s.f.). https://github.com/redis/redis/blob/b7c6755b1b44da36866c0376da0a000873177270/README.md#why-choose-redis. (Accessed: 2025-06-04)

yjs. (s.f.). https://github.com/yjs/yjs. (Accessed: 2025-06-01)

Anexo A

Casos de uso

En este anexo se presenta una lista mayor de casos de uso que la que está en la Sección 4.1.2.

A.1. Nuevos casos de uso

Aquí mostramos los casos de uso que difieren de las ediciones pasadas de Matefun.

Limpiar el editor con un archivo en blanco: Si el usuario está editando un archivo ya existente, y quiere comenzar a trabajar uno nuevo en su lugar, debe apretar en el botón ✚, lo cual lo llevará a la vista principal con el editor en blanco esperando la creación de un nuevo archiv.

Crear un Archivo nuevo: El caso comienza cuando un usuario quiere crear un nuevo archivo. Tiene dos posibles caminos:

1. El usuario tiene un archivo aún no guardado en el editor en la página principal.
2. El usuario escribe un nombre en el cabezal del editor.
3. El usuario hace click en el botón 🖫 en el cabezal del editor.
4. El Sistema despliega el menú de guardar archivo.
5. El usuario puede opcionalmente cambiar el nombre que eligió para el archivo.
6. El usuario elige en qué directorio guardar el archivo.
7. El usuario hace click en el botón de Crear Archivo.
8. Alternativamente el usuario cancela la creación de archivo haciendo click en la cruz en la parte superior del menú.

La otra forma consiste en:

1. El usuario navega hasta la página de archivos por el menú lateral.
2. El usuario navega hasta el directorio donde quiere crear el archivo.
3. El usuario hace click en el botón .
4. El usuario selecciona la opción Archivo.
5. El sistema despliega el menú de crear archivo.
6. El usuario escribe el nombre para el archivo que desea crear.
7. El usuario hace click en el botón Crear.
8. Alternativamente el usuario cancela la creación haciendo click en la cruz en la parte superior del menú de crear archivo o el menú de crear directorio.

Guardar programa: Este caso ocurre cuando un usuario tiene un programa cargado en el editor en la web. Cada 10 segundos el programa se guarda automáticamente, o alternativamente el usuario puede guardarlo de manera manual haciendo click en el botón de la barra del editor de código.

Eliminar directorio o archivo: Este caso de uso permite al usuario eliminar de su lista de archivos a un archivo o directorio particular. Para esto, el usuario debe:

1. Navegar hasta la vista de archivos.
2. Navegar en las carpetas hasta encontrar el archivo que desea eliminar.
3. Seleccionar el archivo a eliminar.
4. Hacer click en el botón en la barra ubicada sobre la vista previa.

En caso de que el usuario tenga el rol de Dueño sobre el archivo elegido, o se intente borrar un directorio, el archivo o directorio será eliminado del sistema y ya nadie tendrá acceso al mismo. En caso de que el usuario tenga un rol distinto, el archivo será eliminado sólo de su árbol de archivos, pero aún estará disponible para las demás personas que tenían acceso al mismo.

Crear archivo o directorio en un grupo: Este caso comienza cuando un usuario de un grupo quiere crear un archivo en el mismo. Para esto:

1. El usuario navega hasta la vista de grupos desde el menú lateral del sitio.
2. El usuario selecciona el grupo en el cual desea crear un archivo o directorio.
3. El usuario selecciona la pestaña de Archivos.
4. El usuario navega hasta el directorio donde desea crear el archivo o directorio.

5. El usuario hace click en el botón ✚.
6. El usuario selecciona si desea crear un archivo o un directorio.
7. El sistema despliega el menú de crear archivo o directorio.
8. El usuario escribe el nombre para el archivo o directorio que desea crear.
9. El usuario hace click en el botón Crear.
10. Alternativamente el usuario cancela la creación haciendo click en la cruz en la parte superior del menú de crear archivo o el menú de crear directorio.

A.2. Lista de casos de uso presentados en ediciones anteriores que no han cambiado

Aquí listamos los casos de uso presentes en ediciones anteriores que no han sufrido cambios.

- **Cerrar sesión**
- **Ejecutar programa**
- **Exportar programa**
- **Cambiar configuración del editor**
- **Habilitar modo avanzado**
- **Desactivar modo avanzado**
- **Pasar a vista gráfica**
- **Reiniciar intérprete**
- **Graficar función**
- **Graficar figura**
- **Graficar secuencia**
- **Crear directorio**
- **Mover archivo o directorio**

Anexo B

Diagrama de Clases

Aquí se presenta un diagrama de clases más completo al ya presentado en la Sección 4.1.5. Por temas de espacio en la página se presentará fragmentado en cuatro partes y no se mostrarán las claves primarias, las claves foráneas, ni los campos "created_at" y "updated_at" que todo registro de Rails tiene y señalan la fecha y hora de su creación y de su última actualización respectivamente.

Las Cuatro figuras son la Figura B.1 que muestra la mayor cantidad de clases posibles, faltan aquí clases que están sólo relacionadas al User que se muestran en la Figura B.2, la clase MatefunGroupFile y sus relaciones que se muestra en la Figura B.3, y la clase AdminUser no tiene conexión directa con ninguna otra clase, y se puede ver en la Figura B.4

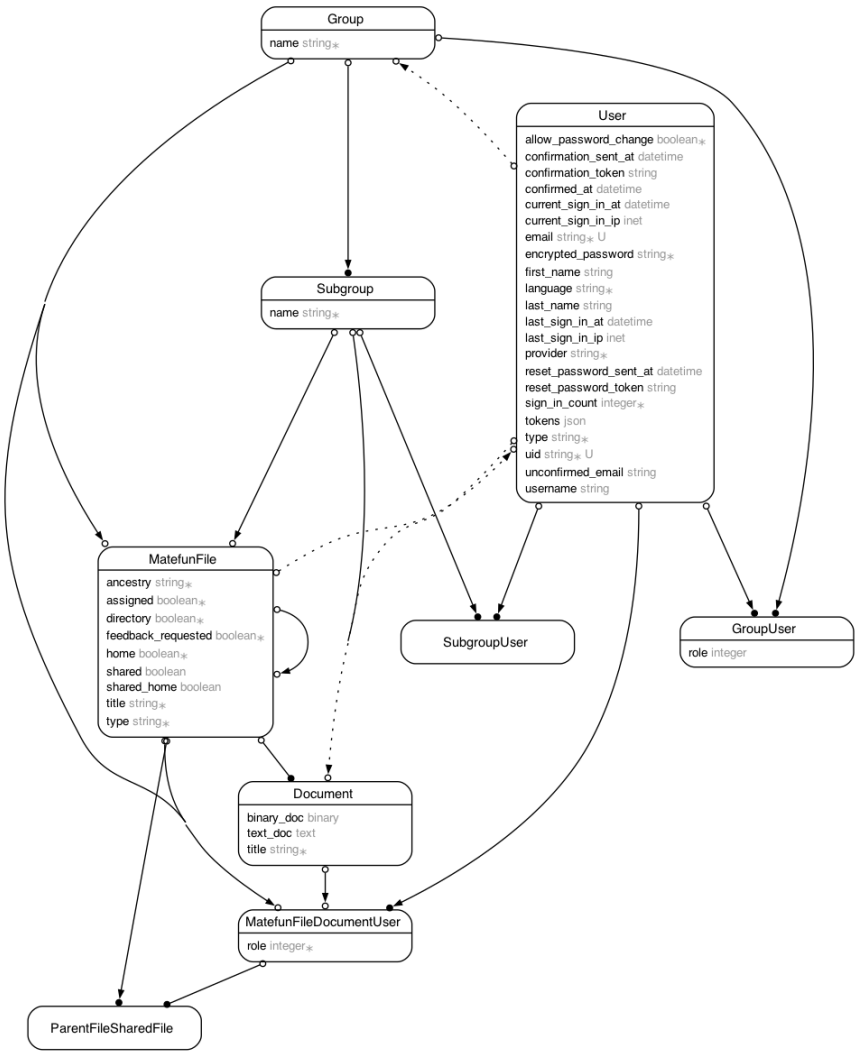


Figura B.1: Diagrama de Clases. Conexión general entre las grandes partes de la aplicación.

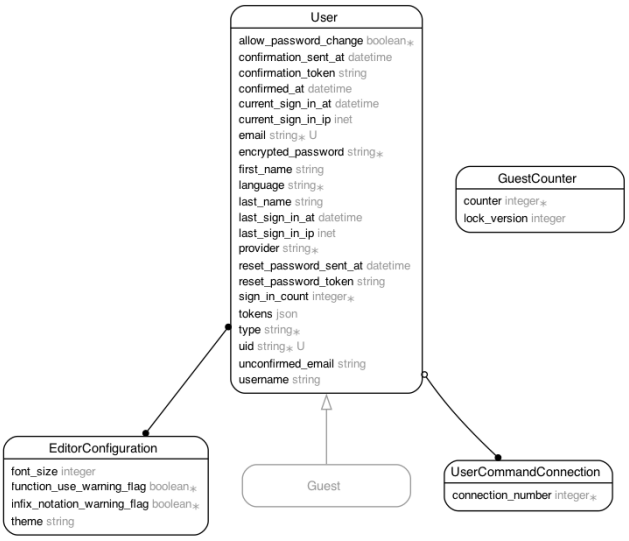


Figura B.2: Diagrama de Clases. User y sus clases asociadas.

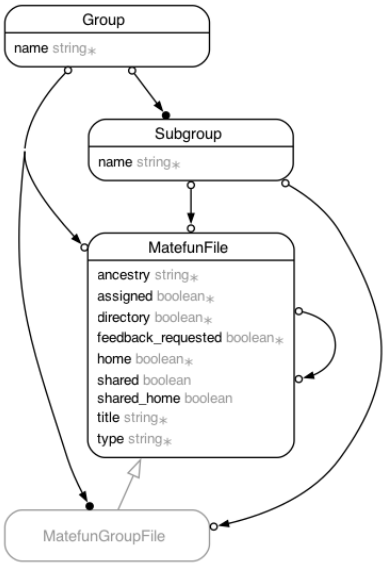


Figura B.3: Diagrama de Clases. MatefunGroupFile y sus clases asociadas

Figura B.4: Diagrama de Clases. AdminUser

Anexo C

Requerimientos funcionales de ediciones pasadas

A continuación se listan los requerimientos que ya existían en ediciones previas y no han cambiado, creadas por Gonzalo Cameto (Accessed: 2025-06-10a)

- **Editor de código:** El sistema debe contar con un editor de texto que permita escribir programas en lenguaje Matefun. Se debe contar con las funcionalidades clásicas de un editor como buscar y reemplazar palabras, coloreo de sintaxis, auto completar y posicionar en determinada línea.

- **Tema del editor:** Debe darse distintas opciones de estilos para que el usuario pueda configurar a su gusto la visualización del editor. Modificando los colores de fondo y de la sintaxis.

- **Posición del cursor:** El sistema debe mostrar en todo momento la posición actual del cursor dentro del editor (línea y columna).

- **Autocompletar:** El editor de programas debe contar con función de autocompletar. Al presionar ctrl+space se debe desplegar la lista de palabras que ya fueron ingresadas y que coinciden con el prefijo escrito hasta el momento.

- **Exportar programa:** La interfaz debe permitir descargar un programa. En este caso se descarga un archivo con extensión mf que contiene el código del programa. Este archivo podría ejecutarse directamente con el binario del compilador.

- **Intérprete interactivo:** Se debe contar con un intérprete interactivo en donde el usuario pueda ejecutar las funciones nativas del lenguaje y también las funciones que se hacen disponibles luego de la compilación de un programa.

- **Errores de compilación:** Luego de que el usuario de la orden para compilar un programa, el sistema debe mostrar (si existen) los errores de compilación en el intérprete interactivo. Adicionalmente es deseable poder señalar en el editor de texto la línea en donde se detectó el error (con un elemento gráfico) y mostrar el mensaje de error específico al posicionar el cursor arriba.

- **Advertencias del compilador:** El compilador de Matefun puede generar advertencias que ayuden al usuario a entender y a usar de mejor forma los programas. Estas advertencias deben ser mostradas en el intérprete interactivo. Además es deseable que se muestren en el editor de texto, posicionando un elemento gráfico en la línea que generó la advertencia y mostrando el mensaje de advertencia al posicionar el cursor arriba.

- **Gráficos:** El sistema debe poder generar gráficos para funciones matemáticas definidas de $R \to R$.

- **Figuras:** El sistema debe poder dibujar figuras. En el compilador Matefun existe el tipo primitivo "Fig", que representa una figura (circulo, rectángulo, polígono, entre otros). Estos elementos deben poder ser mostrados en el sistema.

- **Secuencias:** El sistema debe poder dibujar una secuencia de figuras. Aquí el intérprete de Matefun puede retornar una secuencia (lista) de figuras. Se deben poder dibujar de forma secuencial cada determinado tiempo, generando una animación.

- **Cambiar rango de visualización:** Se debe poder ampliar y reducir el área de visualización en el visor de gráficos, tanto en el eje x, como en el eje y, o en ambos de forma simultánea. También se debe contar con un botón para centrar el dibujo en su posición original.

- **Mover área de visualización:** Se debe poder mover el dibujo para explorar secciones que no son visibles para la visualización actual.

- **Exportación de gráficos:** El sistema debe permitir descargar el gráfico dibujado en algún formato de imagen.

- **Elementos de guía en gráfico:** Se debe contar con la posibilidad de visualizar los ejes cartesianos, una grilla de fondo y para el caso de funciones reales, indicar el punto (x,y) al posicionar el cursor sobre el gráfico. Estas características deben poder ser habilitadas y deshabilitadas por el usuario.

- **Alternar vistas:** Se debe poder alternar entre la vista del editor de programas y la vista de visualización de gráficos. El intérprete interactivo siempre debe ser visible.

- **Personalización del intérprete:** El intérprete interactivo debe poder ser configurado en cuanto a los colores del texto. El usuario debe poder elegir el color para la entrada de texto, la salida y para los errores.
- **Modo Avanzado:** El intérprete interactivo debe contar con un modo avanzado. Este modo permite registrar todas las interacciones que ocurren con el servidor relacionadas al intérprete. Se debe activar escribiendo "modo avanzado" y desactivar escribiendo "modo normal".
- **Reiniciar intérprete:** Debe existir la posibilidad de reiniciar el intérprete. Esto supone reejecutar Matefun sin cargar ningún programa, quedando en el mismo estado que cuando recién se accede al sistema (a excepción de que el editor conserva el código escrito).
- **Guardar programa:** Desde la vista del editor se debe poder guardar el contenido del programa actual (si es un programa editable).
- **Compilar y cargar programa:** Desde la vista del editor se debe permitir compilar y ejecutar el programa actual. Luego de una compilación exitosa, las funciones definidas por el usuario se hacen disponibles en el intérprete (en el cual se pone el foco).
- **Repositorio de Archivos:** Los usuarios tendrán disponible un repositorio en donde podrán crear archivos (programas) y directorios. Se debe contar con la funcionalidad de mover archivos y directorios, así como la posibilidad de eliminarlos.
- **Previsualizar programas:** Se debe permitir previsualizar un programa. Esto supone que desde la vista de exploración del repositorio del usuario, al hacer click sobre un programa, se despliega una previsualización del mismo, sin la necesidad de cargarlo en el editor de código.
- **Atajos de teclado:** El sistema debe contar con atajos de teclado para poder realizar las siguientes acciones: poner el foco en el intérprete, poner el foco en el editor, poner el foco en el visor de gráficos, compilar y ejecutar, guardar programa actualmente en edición, crear nuevo programa. Esta función se espera esté disponible sólo para la utilización desde equipos de escritorio.

Anexo D

Servicios

En este apéndice se presenta una lista de los servicios existentes en el sistema y su funcionalidad.

- **Files:** los servicios en este namespace se encargan de varias de las acciones posibles con archivos.
  - **Files::CopyService:** Este servicio se encarga de copiar archivos. No se utiliza de momento para una funcionalidad directa del sitio, sino que se utiliza al momento de asignar archivos a otros usuarios en grupos. Podría utilizarse en un futuro si se quiere tener un botón que haga copia de archivos también.
  - **Files::CreationService:** El mismo se encarga de la creación de archivos que sean directorios. Si se desea crear archivos que tengan documentos utilizar el Documents::CreationService en su lugar.
  - **Files::DestroyerService:** Este servicio se encarga de la acción de borrar un archivo. Si el usuario no es el dueño del archivo el mismo sólo será borrado de su lista de archivos, pero no afectará su disponibilidad para las demás personas que tengan acceso al mismo. Si el usuario es el dueño el archivo se borrará completamente de la base de datos.
  - **Files::SharerService:** Este servicio se encarga de compartir archivos entre varios usuarios, ya sea agregar, sacar usuarios, o cambiar los roles que tengan los mismos sobre el archivo en cuestión.
  - **Files::UpdaterService:** Este Servicio se utiliza cuando se mueve un archivo, cuando se hace un pedido de retroalimentación, una devolución de retroalimentación, o también cuando se quieren cambios en los usuarios de un archivo. Para este último caso este servicio llama al `Files::SharerService` para hacer la actualización de usuarios y sus roles.
- **Group:** Estos servicios se encargan de la administración de los grupos.

- **Groups::CreationService:** Este servicio es el encargado de crear grupos.
- **Groups::DestroyerService:** Este es el servicio llamado cuando un usuario desea borrar un grupo. Si el usuario es dueño del grupo, el mismo se borrará del sistema por completo. Si por otro lado el usuario no es dueño del grupo, el grupo será borrado de su lista de grupos, pero el grupo no cambiará para los demás usuarios del mismo.
- **Groups::UpdaterService:** El mismo se utiliza para actualizar el nombre y los permisos de los usuarios del grupo. Para esto último utiliza el `Groups::UsersService`.
- **Groups::UsersService:** Este servicio se utiliza para administrar a los usuarios del grupo y sus roles.
- **Groups::Files::Assignments::CreationService:** Este servicio se utiliza para crear asignaciones en el grupo. Funciona haciendo una copia de los archivos elegidos por cada usuario o subgrupo elegido, utilizando el `Files::CopyService`.
- **Groups::Subgroups:** Estos servicios se encargan de las acciones relacionadas a los subgrupos de un grupo.
  - **Groups::Subgroups::CreationService:** Este servicio se utiliza para crear subgrupos.
  - **Groups::Subgroups::Service:** Este servicio se utiliza al momento de destruir subgrupos.
  - **Groups::Subgroups::UpdaterService:** Este servicio se utiliza para actualizar el nombre del subgrupo, así como la lista de usuarios que pertenecen al mismo.

- **Guests::CounterService:** Este es un servicio auxiliar, que se utiliza para tener nombres de usuario únicos al momento de crear invitados en el sitio. El mismo funciona con un contador global que se utiliza en el nombre de usuario y sube con cada Invitado que se crea.

- **FrontendUrlService:** Este servicio se utiliza como helper para obtener las urls a las que hay que redireccionar al usuario cuando confirman su cuenta, o piden hacer reset de su password. Para esto utilizan variables de entorno donde se guarda la url del *frontend*.

- **MatefunService:** El mismo es utilizado por el GhciChannel para interactuar con el binario de Matefun cuando un usuario manda un comando.
  - **Matefun::FileService:** El mismo se utiliza al momento de ejecutar archivos de los usuarios ya que es necesario crear archivos en el sistema de archivos del sistema operativo donde está corriendo el servidor mismo con el contenido de los documentos de los usuarios para que el binario Matefun los reconozca.

- **Users::DestroyerService:** Este servicio se utiliza si se quiere destruir usuarios. Borrará también todos los grupos y archivos de los que el usuario era dueño.