UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Documento de Arquitectura

Anexo a informe de Proyecto de Grado presentado por

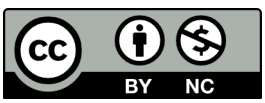
Facundo Barboza y Elizabeth Bennett

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

Sylvia da Rosa Zipitría
Federico Gómez Frois

Montevideo, 15 de abril de 2026



Índice general

Capítulo 1

Introducción

1.1. Propósito

El presente documento tiene la finalidad de describir la arquitectura de MateFun considerando los desarrollos llevados a cabo en proyectos de grado anteriores junto con las extensiones incorporadas en el presente proyecto.

1.2. Alcance

Este documento presentará la arquitectura del sistema, así como un detalle de los módulos que la componen.

Capítulo 2

Arquitectura de MateFun

La arquitectura base de la aplicación MateFun sigue siendo la misma que tenía previo al inicio de este proyecto, ya que el objetivo del mismo era incorporar nuevas funcionalidades construidas sobre la arquitectura ya existente. En las siguientes secciones se detallarán las distintas vistas de la arquitectura de MateFun.

2.1. Vista lógica

En la Figura 2.1 se aprecia el diagrama de capas de la aplicación MateFun junto con las tecnologías utilizadas para la implementación de cada una.

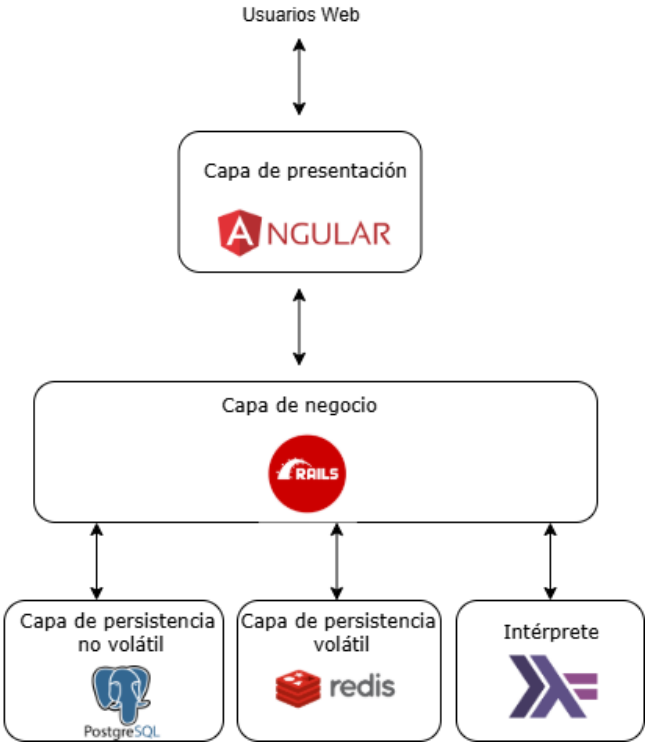


Figura 2.1: Diagrama de capas

1. **Presentación:** La capa de presentación está compuesta por la aplicación web desarrollada en Angular, que se ejecuta en el navegador de los usuarios. Su principal función es proporcionar la interfaz gráfica de usuario (UI), permitiendo la interacción con el sistema de manera dinámica y fluida. En esta capa se implementa el editor colaborativo mediante la integración de CodeMirror y Y.js, lo que posibilita la edición simultánea de código entre múltiples usuarios en tiempo real. Además, esta capa se comunica con la capa de negocio a través de servicios REST (HTTP) y canales WebSocket (Action Cable) para enviar y recibir datos sin necesidad de recargar la página.

2. **Negocio:** La capa de negocio está implementada en Ruby on Rails, y constituye el núcleo lógico del sistema. En ella se gestionan las reglas de negocio, la autenticación y control de sesiones, la coordinación de la colaboración en tiempo real y la orquestación de las operaciones de persistencia y ejecución del intérprete. Rails centraliza la comunicación entre las demás capas mediante dos mecanismos principales:

    a. REST Controllers, que gestionan las peticiones HTTP provenientes del cliente.

b. Action Cable, que maneja la comunicación en tiempo real a través de WebSockets. Esta capa también implementa la lógica de dominio vinculada a la gestión de usuarios, grupos y proyectos colaborativos.

3. **Persistencia volátil:** La capa de persistencia volátil está basada en Redis, una base de datos en memoria que ofrece bajo tiempo de respuesta y almacenamiento temporal. Su función principal es actuar como canal de comunicación y cacheo de información entre los distintos procesos del servidor. En esta capa se mantiene el estado de las sesiones activas y los datos transitorios de colaboración en tiempo real. Redis también es utilizado por Action Cable para la gestión de mensajes entre los distintos clientes conectados, asegurando la sincronización de las ediciones colaborativas.

4. **Persistencia no volátil:** La capa de persistencia no volátil utiliza una base de datos PostgreSQL, responsable del almacenamiento permanente de la información. Aquí se conservan los datos estructurados del sistema, como usuarios, grupos, funciones y registros de actividad. El acceso a esta base se realiza mediante el ORM (Object-Relational Mapping) ActiveRecord de Rails, que abstrae las operaciones de lectura y escritura, asegurando la integridad de los datos y facilitando la portabilidad del sistema.

5. **Intérprete:** El intérprete constituye el componente encargado de ejecutar el código Matefun ingresado por los usuarios. Se trata de un programa binario independiente, desarrollado en Haskell, que se ejecuta como un proceso externo al servidor Rails mediante la librería Open4. La capa de negocio envía al intérprete el código a evaluar y recibe los resultados de ejecución, que luego son devueltos al usuario. Esta separación entre el servidor y el intérprete garantiza aislamiento de procesos, brindando mayor seguridad.

2.2. Vista de distribución

En la Figura 2.2 se puede apreciar el diagrama de distribución de la aplicación MateFun. En el mismo, desde el punto de vista físico se identifican tres nodos: Navegador (browser), Servidor de aplicaciones (Application Server) y Servidor de base de datos (DataBase System).

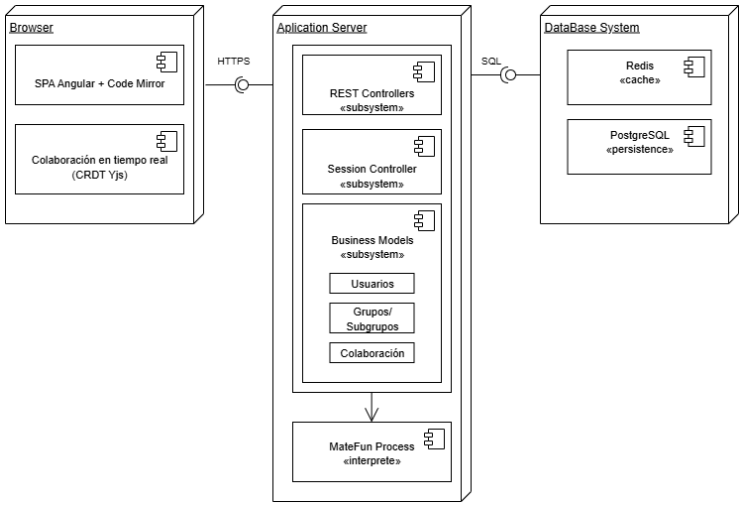


Figura 2.2: Diagrama de distribución

1. **Browser:** El navegador representa el cliente del sistema, donde los usuarios interactúan directamente con la aplicación. En dicho cliente se ejecutan los componentes de la capa de presentación descrita en la sección 2.1.

2. **Application Server:** El servidor de aplicación constituye el núcleo lógico del sistema en donde se ejecutan tanto el intérprete como los componentes de la capa de negocio, desarrollada en Ruby on Rails. Centraliza las peticiones del cliente, gestiona las reglas de negocio y coordina la comunicación con las bases de datos y el intérprete. Este nodo se comunica con el Browser mediante HTTPS y con el Database System mediante consultas SQL.

    - **REST Controllers** Los controladores REST implementan la interfaz HTTP del sistema. Reciben las solicitudes del cliente (Angular), las procesan y devuelven las respuestas en formato JSON.
    - **Session Controller** Este subsistema gestiona el inicio, mantenimiento y cierre de sesiones de usuario. Se encarga de la autenticación, la validación de credenciales y la administración de tokens o cookies de sesión. Puede interactuar con Redis para almacenar sesiones activas o invalidarlas en tiempo real.
    - **Business Models:** Contiene la lógica de negocio y las entidades principales del dominio de Matefun:
        - Usuarios: gestión de cuentas, roles y autenticaciones.
        - Grupos/Subgrupos: administración de jerarquías y relaciones entre usuarios.

- Colaboración: coordinación de ediciones simultáneas, control de concurrencia y permisos.

- **MateFun Process:** representa el intérprete binario del lenguaje Matefun, desarrollado en Haskell. Rails lo invoca como un proceso externo mediante la librería Open4, enviándole el código a ejecutar y recibiendo los resultados.

3. **Database System:** El nodo de base de datos está formado por dos sistemas complementarios que gestionan los dos tipos de persistencia descritos en la vista lógica: la no volátil (permanente) y la volátil (temporal).

   - **PostgreSQL:** Base de datos relacional utilizada para el almacenamiento persistente de la información. Guarda entidades como usuarios, grupos, proyectos y registros de actividad.
   - **Redis:** Base de datos en memoria de tipo clave-valor utilizada para persistencia volátil. Se usa como cache para datos temporales, almacenamiento de sesiones activas y soporte a la colaboración en tiempo real.

2.3. Subsistemas

En la Figura 2.3 se puede apreciar el diagrama de subsistemas y sus dependencias.

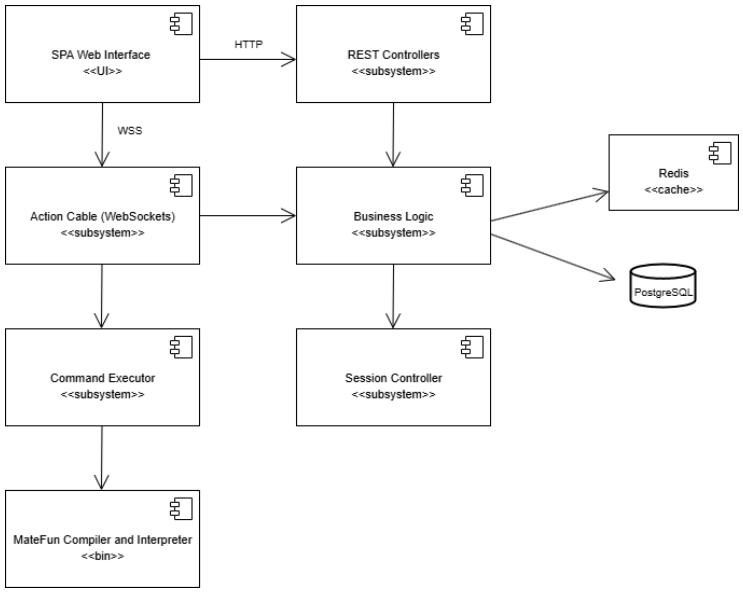


Figura 2.3: Diagrama de subsistemas y dependencias

1. **Interface Web:** Es una aplicación Web basada en el framework de Angular, en conjunto al framework de CSS Bootstrap. También utiliza bibliotecas como JQConsole[4] y CodeMirror para la implementación del intérprete y el editor de texto respectivamente. Su función es permitir la interacción del usuario con el sistema, proporcionando herramientas de edición, visualización y colaboración en tiempo real. Se comunica con el servidor mediante peticiones HTTP (REST) y WebSockets (WSS) para la sincronización colaborativa.

2. **REST Controllers:** Este subsistema implementa la API REST del sistema. Se encarga de recibir las solicitudes HTTP provenientes del cliente Angular, procesarlas y reenviarlas a la lógica de negocio, devolviendo las respuestas en formato JSON.

3. **WebSocket Service:** Este subsistema es el encargado de exponer un endpoint de WebSocket a través del cual la capa de presentación invoca los comandos escritos por el usuario en el intérprete interactivo. Como resultado de la llamada a los distintos servicios REST, se devuelven mensajes en formato JSON con los resultados de la invocación de los comandos. Estos resultados luego son dibujados por el módulo del graficador correspondiente.

4. **Business Logic:** Concentra las reglas de negocio y el comportamiento funcional del sistema. Aquí se encuentra la lógica de negocio encargada de realizar el procesamiento necesario antes de persistir datos o de enviar datos de respuesta ante la invocación de un servicio web.

5. **Session Controller:** Administra las sesiones y autenticaciones de usuarios. Verifica las credenciales, mantiene el estado de sesión y gestiona la expiración de tokens o cookies de acceso.

6. **Command Executor:** Es el módulo encargado de invocar y controlar la ejecución del intérprete Matefun. Recibe solicitudes desde la capa de negocio, ejecuta el proceso externo mediante Open4, y devuelve los resultados al sistema.

7. **Matefun Compiler and Interpreter:** Representa el intérprete binario del lenguaje Matefun, implementado en Haskell. Recibe como entrada un programa MateFun (fichero con extensión .mf) y lo compila y ejecuta en modo interactivo.

Referencias