Universidad Europea

SYNAPS

TRABAJO FINAL DE CICLO

INTEGRANTES

Héctor Augusto Chango Tapia
Daniel Correa Villa
Ian Isaac Santillan Cueva

TUTORES

Sara Villanueva Rosa
Irene del Rincón Bello

CURSO

2º Desarrollo de Aplicaciones Web

Introducción

En un entorno donde la información crece de forma exponencial, gestionar el conocimiento de manera eficiente se ha convertido en una necesidad tanto para individuos como para equipos. En respuesta a esta necesidad, han surgido herramientas como **Notion** y **Obsidian**, cada una con enfoques distintos: **Notion** destaca por su capacidad de colaboración y uso de bases de datos; **Obsidian**, por su estructura descentralizada. Sin embargo, estas plataformas no integran todas las funciones de forma unificada.

Synaps nace como una solución que combina lo mejor de ambas, incorporando además funcionalidades propias. Se trata de una plataforma modular de notas y gestión de conocimiento que permite trabajar con contenido estructurado, visualizar relaciones entre elementos, colaborar en tiempo real y gestionar datos con bases de datos sincronizadas.

Características principales

- Vista de galaxia interactiva para explorar relaciones entre notas.
- Edición colaborativa de notas.
- Sistema de notificaciones integrado.
- Bases de datos reutilizables entre distintos documentos.

Comparativa de funcionalidades

Herramienta	Características destacadas	Synaps incluye
Notion	Bases de datos sincronizadas	Tablas compartidas entre secciones
Obsidian	Vista gráfica de notas (Note Graph)	Vista de galaxia interactiva
Navegadores	Notas simples sin integración avanzada	Notas avanzadas, tareas y sistema de avisos

Palabras clave

Backend

PHP, Laravel, Node, Redis, WebSocket, Python, Flask, Keycloak, Gestión de rutas dinámicas, Modelo Vista Controlador, Gestión de controladores según rutas, Monolog, PHPUnit, UI Components

Frontend

React, HTML, CSS, JavaScript, Lazy Loading, UI/UX, Figma, accesibilidad, minificación, diseño responsive, validaciones en tiempo real

Bases de datos

MySQL, Redis, Modelos de Datos, Escapado de consultas, Modelos.

Despliegue

Linux, Debian Bookworm, Docker, Docker-Compose, Apache, Git, GitHub, SSH Agent

Índice

Módulos formativos incluidos

Durante el desarrollo de la aplicación ha sido necesario aplicar numerosas aptitudes aprendidas durante el Grado Superior de Desarrollo de Aplicaciones Web:

Diseño de Interfaces: Aplicación de técnicas de diseño centradas en el usuario (UX), basadas en las leyes de usabilidad aprendidas. Creación de prototipos y maquetas de las interfaces utilizando Figma funcionales, planificando la estructuración de la página web y el panel desde el inicio del proyecto.

Desarrollo en Entorno Cliente: Uso de JavaScript y sus métodos para realizar validaciones en tiempo real. Estas validaciones incluyen comprobaciones inmediatas al interactuar con los campos y mensajes de error dinámicos que se presentan junto a los elementos afectados.

Desarrollo en Entorno Servidor: Implementación de la lógica de la aplicación utilizando Laravel, Redis y conexión a bases de datos MariaDB.

Despliegue de Aplicaciones Web: Configuración y gestión del servidor Apache utilizando las configuraciones aprendidas en clase, incluyendo el uso de archivos **.htaccess** para la gestión de rutas y control de accesos. Además, se emplea Docker para crear un entorno de desarrollo, optimizando la compatibilidad y el despliegue del proyecto al servidor de producción.

Entornos de desarrollo: Uso de tecnologías como Git y GitHub para mantener un historial de cambios coherente y un repositorio con backups con el código de la aplicación en el caso de que ocurra algún tipo de error que borre el código o los recursos utilizados dentro del sistema.

Lenguaje de marcas: Uso de HTML y CSS para el desarrollo de la estructura y estética de la aplicación. Implementación de JavaScript para programar el comportamiento de la página.

Bases de datos: Desarrollo de bases de datos relacionales (MySQL) para guardar los datos de la aplicación.

Programación: Implementación de la lógica, estructura de ficheros y programación orientada a objetos inspirada en Java. Conexión con MySQL y prepared statements.

Sistemas informáticos: Uso de Linux como sistema operativo principal del proyecto. Instalación de MySQL y Apache para su uso en la aplicación. Navegación de directorios y modificación de ficheros mediante **Bash**.

Herramientas utilizadas

Durante el desarrollo de la aplicación Synaps se han empleado distintas herramientas que han permitido estructurar, desarrollar y desplegar el proyecto de forma eficiente. A continuación, se describen las más relevantes.

Docker

Se ha utilizado como entorno de desarrollo Docker para que todos los integrantes del grupo utilicen las mismas dependencias y versiones. Además, esto nos permite simular diferentes entornos para mantener la independencia de cada entorno (Node, Laravel).

Docker-Compose

Ha servido para organizar todos los servicios del sistema: Laravel, Redis, MariaDB, phpMyAdmin y el puente WebSocket. Permite levantar todo el entorno con un solo comando.

Git / GitHub

Git ha sido el sistema de control de versiones y GitHub el repositorio compartido del proyecto. Se ha utilizado para registrar cambios, gestionar ramas y colaborar en equipo.

Apache

Utilizado como servidor HTTP dentro del contenedor Laravel, ha permitido servir la aplicación de backend sin necesidad de usar comandos como php artisan serve.

phpMyAdmin

Herramienta visual para la gestión de la base de datos MariaDB, ha facilitado la inspección de datos, estructura de tablas y ejecución de consultas.

Visual Studio Code

Editor de código principal del proyecto. Se ha utilizado junto a extensiones como Docker, ESLint, Prettier y soporte para PHP, TypeScript y Redis.

Lenguajes utilizados

PHP

Lenguaje principal del backend utilizado Laravel. Su integración con Redis, MariaDB y librerías como Monolog y PHPUnit lo han hecho ideal para la parte de servidor.

JavaScript

Lenguaje principal del frontend, utilizado para dotar de interactividad a la aplicación y facilitar la comunicación con el backend.

HTML y CSS

Usados para la estructura y los estilos básicos de la interfaz

React

Biblioteca utilizada en el frontend para construir la interfaz de usuario de forma modular.

Node.js

Usado en el servicio redis-ws-bridge, que conecta Redis con los clientes WebSocket.

Redis

Sistema clave en la comunicación en tiempo real. Se ha utilizado para la cola de eventos, cacheo, sesiones de usuario y mensajes pub/sub.

MariaDB

Base de datos relacional para el almacenamiento persistente de toda la información generada por los usuarios y el sistema.

Flask

Servidor web utilizado en Flask para manejar las peticiones GET y POST de Keycloak. Usado principalmente para generar los access_token de la aplicación.

Keycloak

Servicio utilizado para la generación de tokens JWT.

Objetivos

Desarrollar una aplicación que sirva como espacio centralizado para capturar, organizar y relacionar información, pensada para el trabajo en equipo y el aprendizaje continuo.

El desarrollo se ha planteado con un enfoque modular y escalable, utilizando microservicios, contenedores y herramientas modernas de desarrollo web. **Synaps** no solo busca ser funcional, sino también ofrecer una experiencia de usuario cómoda y productiva.

Objetivos específicos del proyecto

Crear una plataforma moderna y escalable para la gestión del conocimiento con notas, bases de datos y edición colaborativa.

N.º	Objetivo
1	Crear una infraestructura Dockerizada para facilitar despliegue.
2	Desarrollar una infraestructura con Laravel.
3	Diseñar el frontend en React.
4	Integrar WebSocket para comunicación en tiempo real.
5	Usar Redis y MariaDB para datos y eventos.
6	Implementar roles de usuario y edición compartida con control de cambios.

Estas funcionalidades permitirán que **Synaps** se convierta en una herramienta completa para equipos y usuarios individuales que buscan una gestión más conectada e inteligente de su información.

Desarrollo del proyecto

Estudio del mercado

Notion

Aspectos positivos

- **Diseño y navegación**

  Interfaz limpia y minimalista, con panel lateral que facilita la organización jerárquica de páginas y bases de datos.
- **Bases de datos sincronizadas**

  Tablas, calendarios y tableros enlazables entre páginas; permiten filtros, vistas y relaciones.
- **Colaboración en tiempo real**

  Edición concurrente con comentarios, menciones y control de versiones.
- **Plantillas y API pública**

  Gran ecosistema de plantillas y conexión con servicios externos mediante API REST.

Aspectos por mejorar

- **Vista gráfica de relaciones**

  No dispone de un grafo visual entre notas; la navegación entre enlaces es lineal.
- **Rendimiento en espacios grandes**

  Las bases de datos masivas pueden ralentizar la carga en navegadores.
- **Privacidad y trabajo sin conexión**

  Requiere conexión estable; el cifrado y el modo offline son limitados.

Obsidian

Aspectos positivos

- **Vista de grafo**

  Representación visual de vínculos entre notas tipo “galaxia”, ideal para descubrir conexiones.
- **Almacenamiento local**

  Archivos Markdown guardados en el dispositivo; control total de los datos.
- **Sistema de plugins**

  Mercado abierto con extensiones para diagramas, tareas, IA, etc.
- **Modo sin conexión**

  Uso completo offline, sin depender de servidores externos.

Aspectos por mejorar

- **Colaboración nativa**

  Carece de edición multiusuario integrada; depende de plugins o servicios de terceros.
- **Bases de datos estructuradas**

  No ofrece tablas relacionales ni vistas tipo kanban sin plugins adicionales.
- **Curva de aprendizaje**

  Configuración inicial y gestión de enlaces pueden resultar complejas para usuarios noveles.

Comparativa de funcionalidades

Característica	Notion	Obsidian	Propuesta Synaps
Bases de datos enlazadas	✔	✘	✔
Vista gráfica de notas	✘	✔	✔
Trabajo sin conexión	✘/✔ (parcial)	✔	✔

Conclusión de mercado

Notion ofrece una experiencia colaborativa robusta, pero carece de un Gráfico de Galaxia y puede ralentizarse con bases extensas. Obsidian destaca en visualización y control local de datos, aunque no incluye colaboración nativa ni bases estructuradas. Synaps aprovechará las fortalezas de ambos: bases de datos sincronizadas y colaboración (estilo Notion) más vista de galaxia y control offline (estilo Obsidian), añadiendo un sistema de notificaciones, modo offline completo y arquitectura escalable basada en microservicios. De esta forma se cubren los vacíos detectados y se ofrece una solución integral de gestión del conocimiento.

Desarrollo e implementación

Para facilitar el desarrollo de la aplicación, se han creado varios diagramas:

Diagrama de casos de uso

El diagrama de casos de uso ha sido desarrollado para visualizar las diferentes acciones que puede realizar cada actor según su rol dentro de la aplicación, así como la forma en que cada rol interactúa con el sistema.

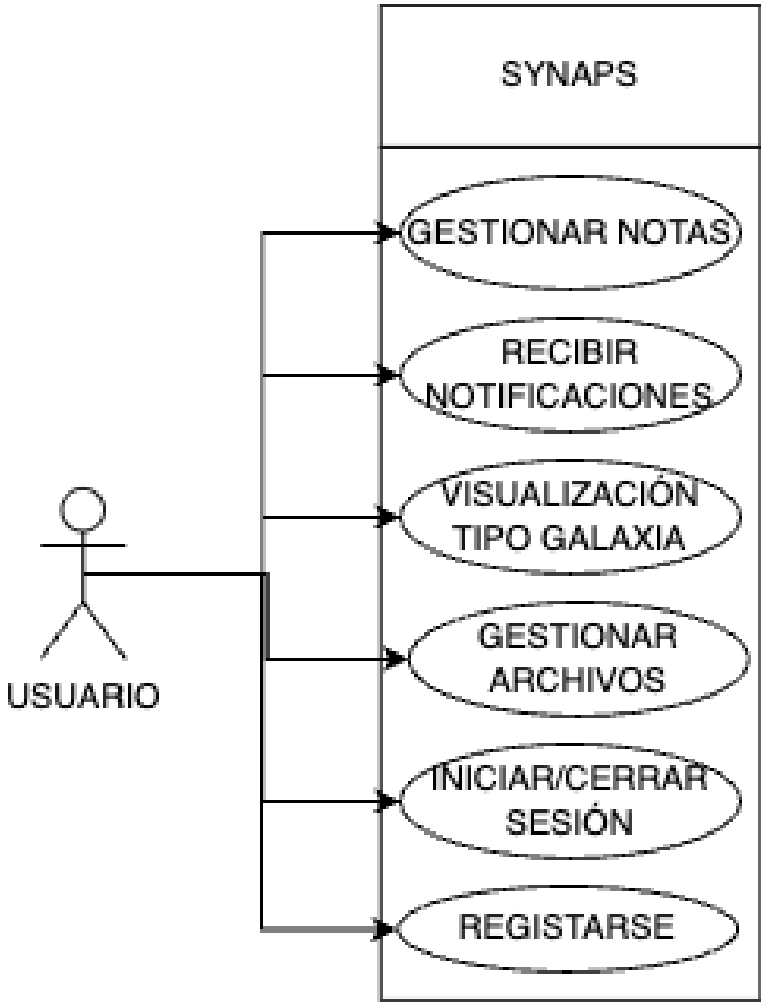


Diagrama de modelo relacional

El **modelo relacional** ha sido desarrollado para definir las tablas de la base de datos, sus columnas, tipos de datos y relaciones mediante claves primarias y foráneas.

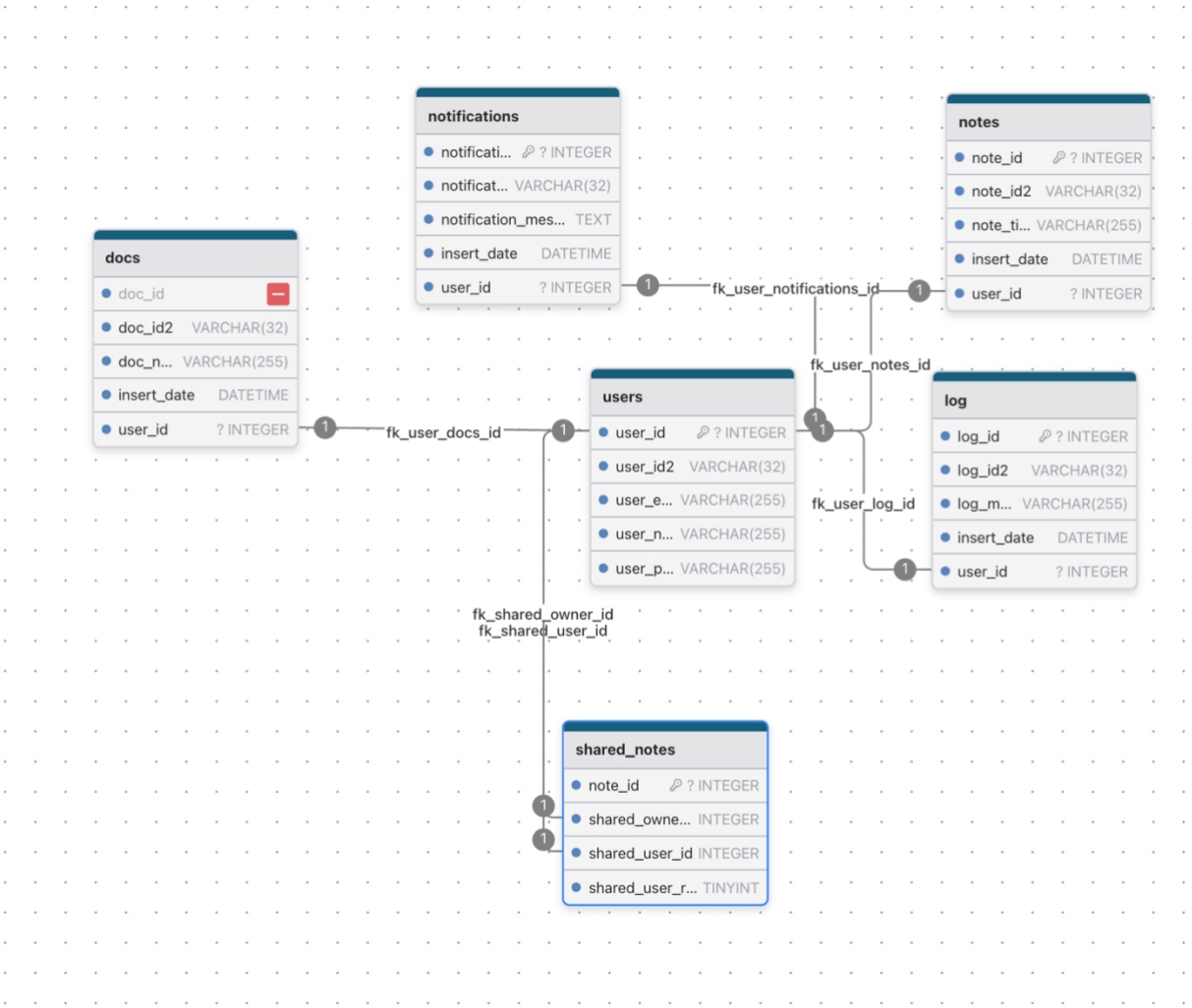


Diagrama de entidad-relación

El **diagrama entidad-relación** ha sido desarrollado para representar las entidades principales del sistema, sus atributos y las relaciones entre ellas. Este modelo permite visualizar cómo los datos están estructurados y cómo las entidades interactúan entre sí, facilitando la organización y gestión de la información en la base de datos.

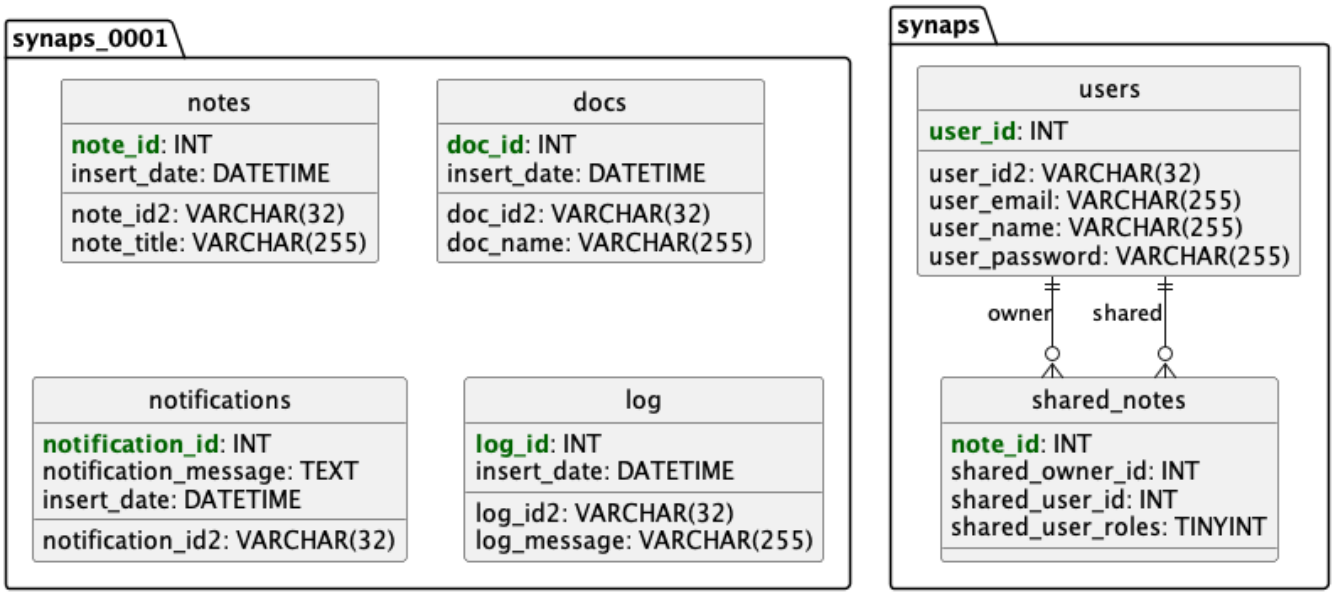
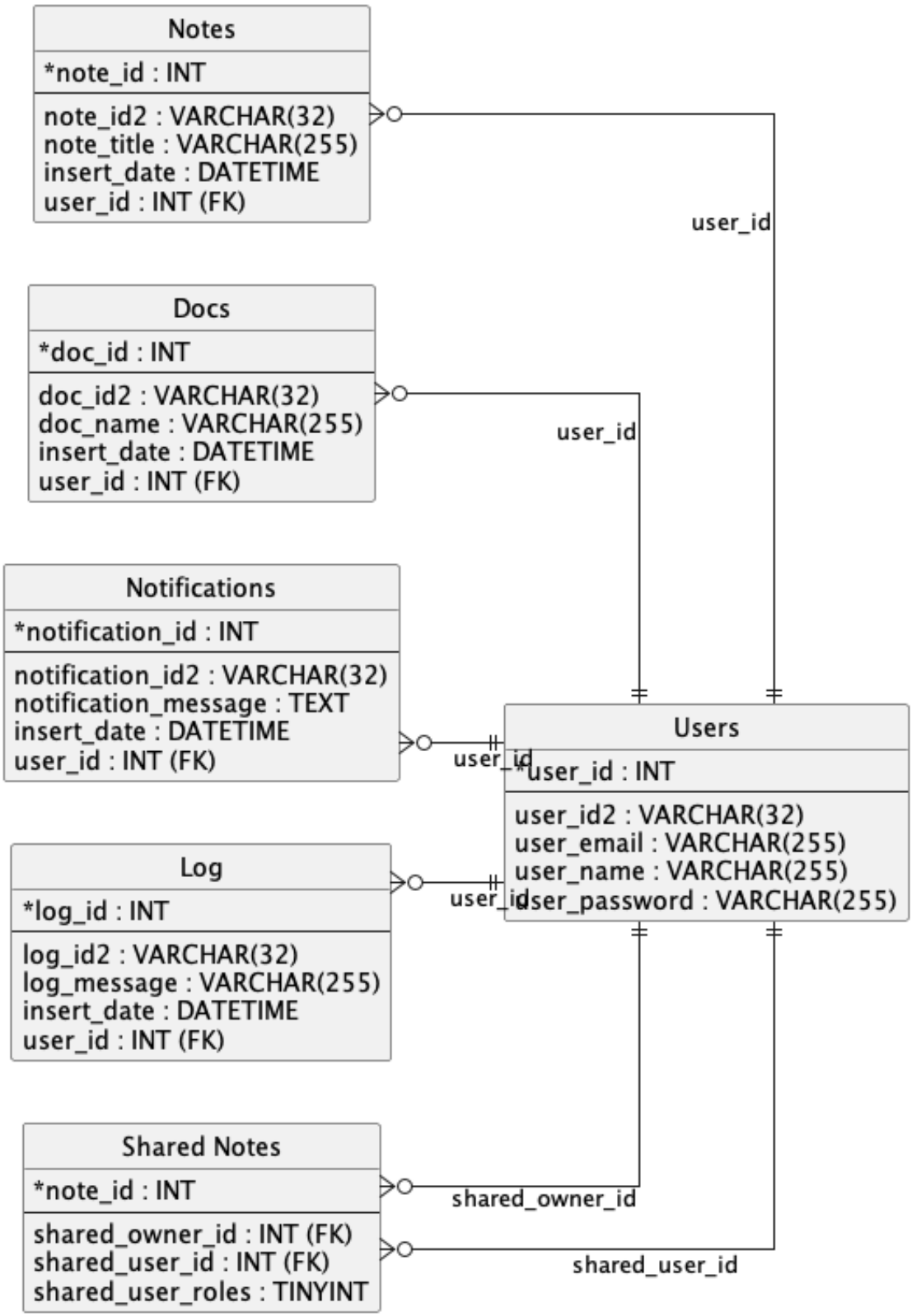


Diagrama UML



Flujo de trabajo de Synaps

1. Estructura general

Synaps se apoya en tres capas diferenciadas:

- **Interfaz React**: Renderiza la página web estática que el navegador muestra al cliente.
- **Servidor Laravel**: Recibe las peticiones, aplica la lógica de negocio y accede a la base de datos.
- **Servicios de mensajería**: Redis y un puente WebSocket que difunden los cambios a todos los clientes conectados.

Cada capa se ejecuta en su propio contenedor. La comunicación se realiza mediante la red interna definida en Docker Compose.

2. Renderizado en el navegador

Al cargar la URL, el navegador descarga los recursos estáticos y React monta el componente. Cuando se realiza una acción (por ejemplo, “Guardar”), el componente:

1. Compone el cuerpo de la petición en JSON.
2. Añade el access_token en la cabecera *Authorization*.
3. Lanza la llamada fetch al endpoint de Laravel.
4. Espera la respuesta del Backend.

3. Lógica de la aplicación

Laravel centraliza toda la lógica de la aplicación:

1. El router de Laravel identifica la ruta de la petición, reconociendo la función a ejecutar en el controlador.
2. Un middleware verifica la firma y la caducidad del token con la clave pública del proveedor de identidad.
3. El controlador valida los datos de entrada, ejecuta la operación solicitada y escribe en MariaDB.
4. Con la transacción confirmada, genera la respuesta HTTP y la devuelve al navegador.

5. Publica un mensaje en Redis con los detalles de la actualización para que otros clientes sincronicen su vista.

4. Mensajería en tiempo real

Un servicio Node mantiene un WebSocket abierto con cada navegador autenticado.

- Al recibir un evento, determina a qué sesión pertenece y envía el mensaje al socket asociado.
- También acepta mensajes del cliente (Por ejemplo, ediciones colaborativas) y los publica para su tratamiento posterior.

El resultado es una experiencia en la que los cambios aparecen simultáneamente en todas las pestañas abiertas, sin necesidad de recargar la página.

5. Autenticación y sesión

El inicio de sesión sigue el flujo *login-check*:

1. El usuario introduce sus credenciales.
2. Keycloak las valida contra la base de datos.
3. Se emite un access_token firmado.
4. El navegador almacena el token y lo reenvía en cada petición HTTP y en la apertura del WebSocket.
5. Si el token expira, la aplicación fuerza un nuevo inicio de sesión.

Con un único token se protegen tanto las llamadas REST como el canal de tiempo real.

Caso de uso del flujo

React: Envía POST /api/notes con la nota y el token

Laravel: Valida el token, guarda la nota, responde 200 OK. Se publican los cambios en Redis mediante un update.

Puente WS: Reenvía el evento a las pestañas del mismo usuario

React: Actualiza la interfaz sin recarga y confirma la operación al usuario

7 · Beneficios del flujo

- **Separación de responsabilidades**: La interfaz se concentra en la experiencia de usuario; el servidor, en la lógica y la persistencia; Redis, en la distribución de eventos.
- **Escalabilidad**: Cada servicio puede duplicarse en nuevos contenedores sin alterar el resto.
- **Consistencia inmediata**: Redis garantiza que todos los dispositivos reflejen el mismo estado en tiempo real.
- **Seguridad**: Utilizando un **access_token** en todas las llamadas garantiza una barrera más de seguridad en el sistema.

Inicio de sesión con Keycloak

1. Componentes implicados

El proceso de autenticación se compone de tres servicios:

- **Keycloak:** Actúa como servidor de identidad y emisor de tokens.
- **Servicio Flask:** Valida el usuario y la contraseña frente a la tabla users en MariaDB/MySQL.
- **MariaDB/MySQL**: Almacena las credenciales y la información básica de la cuenta.

2. Flujo de autenticación

Nº1	**Entrada de credenciales**	El usuario introduce el correo y la contraseña en el formulario React.
Nº2	**Petición al servidor de identidad**	La aplicación envía la solicitud **grant_type=password** al endpoint de **Keycloak**.
Nº3	**Delegación al servicio Flask**	Keycloak reenvía la petición al microservicio Flask.
Nº4	**Verificación en la base de datos:**	Flask comprueba las credenciales en MariaDB y, si coinciden, responde afirmativamente a Keycloak.
Nº5	**Emisión del token**	Keycloak firma un access_token (JWT) y lo devuelve al navegador junto con la información de caducidad.
Nº6	**Almacenamiento en el cliente**	React guarda el token en memoria y lo adjunta en la cabecera *Authorization* de cada llamada al backend.

3. Validación y uso del token

Solicitudes HTTP

Un middleware de Laravel extrae el JWT, comprueba la firma con la clave pública de Keycloak y rechaza la petición si el token ha expirado o es inválido.

4 · Renovación y cierre de sesión

- El access_token posee un tiempo de vida limitado; al expirar, la aplicación redirige al formulario de login.
- El cierre voluntario de sesión elimina el token en el cliente y notifica a Keycloak, lo que invalida el JWT existente.

Con este mecanismo se asegura que cada petición al backend y cada conexión en tiempo real se encuentran respaldadas por una credencial firmada y verificada, manteniendo la plataforma protegida sin añadir complejidad a la experiencia del usuario.