Raspi Screen Control

Manual del Programador

Isabel Pineda Lozano

Índice

1. Contextualización y justificación

1.1 Descripción del proyecto

Este proyecto tiene como objetivo desarrollar un sistema de gestión centralizada de las pantallas de FEVAL (Institución Ferial de Extremadura) optimizando el control y la programación de contenido audiovisual.
Actualmente, estas pantallas están gestionadas a través de Screenly, una plataforma que requiere administrar cada pantalla de manera individual mediante su dirección IP, lo que dificulta la operación y mantenimiento

Se desarrollará un software propio, basado en Node.js, Express y MySQL, que permitirá la gestión de todas las pantallas desde una única interfaz web y móvil, ofreciendo los siguientes servicios:

- Control remoto centralizado: Modificación y gestión de contenido sin necesidad de acceder manualmente a cada Raspberry Pi.
- Aplicación Android (Java en Android Studio): Facilita la administración desde smartphones y tablets.
- Mayor eficiencia y estabilidad: Optimización del tiempo y los recursos, con una plataforma adaptable a futuras necesidades.
- Usuario de escritorio (administrador): Gestión rápida y eficiente de todas las pantallas desde una interfaz centralizada.
- Usuario android: Control en movilidad, con acceso instantáneo desde cualquier lugar.

Esta solución no solo mejorará la usabilidad y operatividad del sistema, sino que también reducirá la carga de trabajo, automatizando procesos y simplificando la administración de los contenidos audiovisuales.
Arquitectura del sistema:

1.2 Requisitos del proyecto

<u>REQUISITOS FUNCIONALES</u>

- Gestión centralizada de pantallas:
    - Permitir la administración de todas las pantallas desde una única interfaz
    - Control remoto del contenido en Raspberry Pi con Screenly API.
- Gestión de assets (contenidos):
    - Los usuarios pueden subir imágenes, videos o URL externas como contenido.
    - El sistema debe permitir actualizar los assets subidos.
    - El sistemas debe permitir eliminar los assets subidos
    - Programar fechas de inicio y fin para cada asset

    - Configurar la duración de reproducción y orden de visualización.
- Usuarios y autenticación segura
    - Registro e inicio de sesión con contraseña cifrada.
    - Roles de usuario: Admin, usuario
- API REST en Node.js + Express con web en React:
    - Gestión eficiente de la comunicación entre Android y la base de datos
    - Soporte para peticiones CRUD de assets y pantallas
    - Uso de WebSockets o SSE para actualizaciones en tiempo real
    - Interfaz de gestión accesible desde cualquier navegador.
    - Permitir la administración de pantallas sin necesidad de la app móvil
- Seguridad y cifrado de datos:
    - Uso de HTTPS y cifrado de contraseñas

REQUISITOS NO FUNCIONALES

- Eficiencia y rendimiento
    - Base de datos MySQL optimizada con índices y consultas eficientes
    - Carga rápida de assets y previsualización en tiempo real.
- Escalabilidad:
    - Arquitectura modular para soportar más pantallas y usuarios.
- Usabilidad y accesibilidad
    - Interfaces intuitivas y optimizadas para web y móvil
    - Monitoreo del estado de las Raspberry Pis
- Compatibilidad:
    - API REST accesible desde cualquier cliente compatible
    - Soporte para futuras integraciones con otros servicios.
- Mantenibilidad y documentación:
    - Código limpio y documentado.
    - Diagramas UML de estructura y comportamiento.
    - Manuales detallados para usuarios y programadores

1.3 Estudio inicial y planificación del proyecto

Metodología:

El ciclo de vida más adecuado sería el modelo incremental Scrum con sprints semanales para desarrollar cada módulo del sistema.

Planificación de Sprints:

Tenemos alrededor de 10 semanas, lo que da tiempo para 5 sprints de aproximadamente 2 semanas cada uno.

Sprint 1 (26 marzo-8 abril)

- Objetivo: Definir y diseñar el sistema
  - Diagrama de casos de uso (web + App Android)
  - Modelo entidad-relación
  - Modelo lógico de la base de datos
  - Backlog del producto listo
  - Diseño inicial de la web
  - Diseño inicial de la app Android

1ª Revisión con tutora (2/4/25): Presentación de diagramas y base de datos

Sprint 2 (9 abril-22 abril)

- Objetivo: Implementar la base de datos y la API
  - Implementar la base de datos en MySQL
  - Pruebas en la base de datos
  - Crear la API REST en Node.js con Express
  - Configurar autenticación (JWT)
  - Crear endpoints básicos para Usuarios, pantallas y contenido
  - Pruebas unitarias de endpoints básicos
  - Pruebas de autenticación
  - Estructura inicial de la web con React
  - Inicio de la app Android

2ª Revisión con tutora (9/4/25): Mostrar API y hacer pruebas

Sprint 3 (23 abril-6 mayo)

- Objetivo: Desarrollar la app Android y la web
  - CRUD de usuarios y pantallas para la web
  - Conectar la web con la API
  - Pruebas de integración web-API
  - Pruebas de interfaz web
  - Web con diseño responsivo y mejoras
  - Conectar la app Android con la API
  - Establecer la subida de contenido y gestión básica en la app
  - Pruebas de conexión de la app android con la API

3ª Revisión con tutora (23/4/25): Mostrar web y app conectadas a la API

4ª Revisión con tutora (7/5/25): Revisión general progreso

Sprint 4 (7 mayo-20 mayo)

- Objetivo: Funciones avanzadas y pruebas
    - Programación de contenido y permisos de usuario/admin en la web
    - Pruebas de roles
    - Historial de actividad y control de pantallas en la web
    - Pruebas del historial
    - CRUD de contenido en la app
    - Sincronización entre dispositivos en la app
    - Pruebas encriptación de contraseñas
    - Seguridad: Encriptación de contraseñas y validaciones

5ª Revisión con tutora (14/5/25): Mostrar sistema casi completo y pruebas de seguridad

Sprint 5 (21 mayo-6 junio)

- Objetivo: Optimización, pruebas finales y documentación
    - Pruebas completas de API, web y app
    - Pruebas multiplataforma
    - Pruebas de usabilidad
    - Optimización del código
    - Redacción de manual de usuario y programador
    - Generar instaladores y documentación final

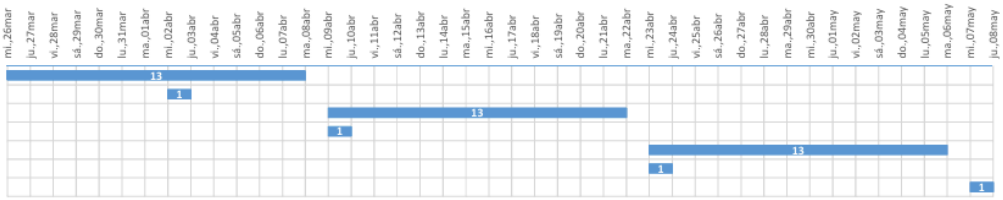
6ª Revisión con tutora (21/5/25): Revisión general progreso

. . .

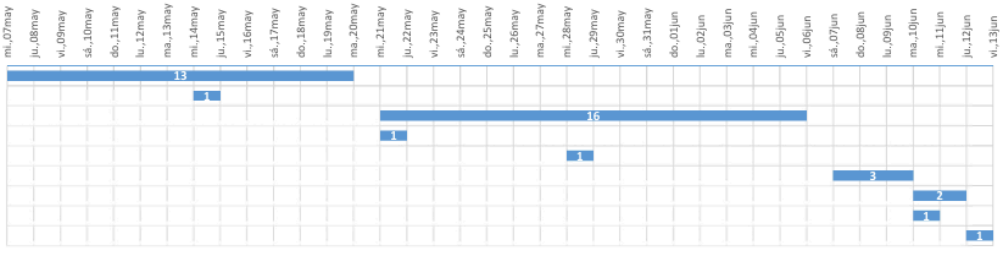
10ª Revisión con tutora (21/5/25):Proyecto listo para la entrega

DIAGRAMA DE GANTT

	Inicio	Duracion	Fin
SPRINT 1:Definir y diseñar el sistema	26/03/2025	13	08/04/2025
1ª TUTORÍA MARISOL	02/04/2025	1	02/04/2025
SPRINT 2:Implementar BD y la API	09/04/2025	13	22/04/2025
2ª TUTORÍA MARISOL	09/04/2025	1	09/04/2025
SPRINT 3: App y web	23/04/2025	13	06/06/2025
3ª TUTORÍA MARISOL	23/04/2025	1	23/04/2025
4ª TUTORÍA MARISOL	07/05/2025	1	07/05/2025

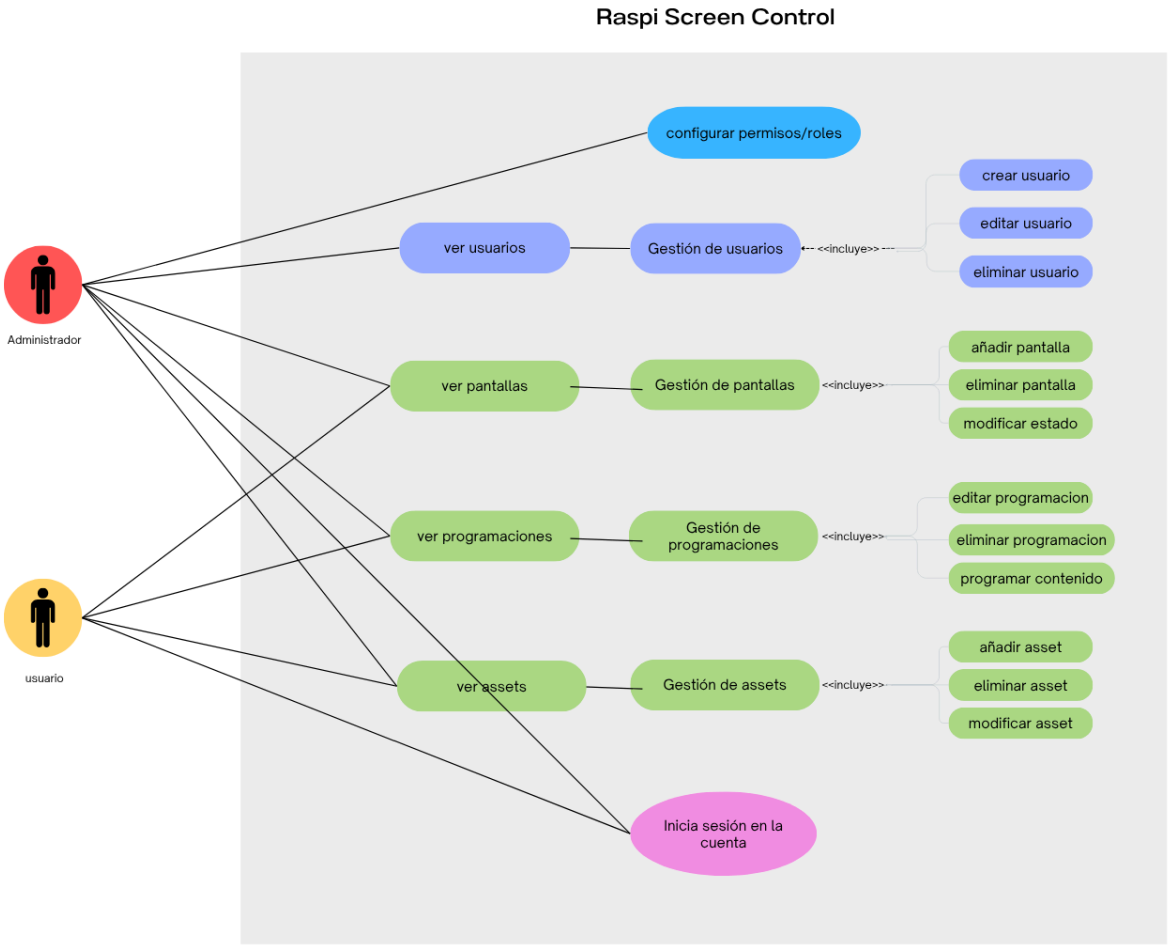


	Inicio	Duracion	Fin
SPRINT 4:Func avanzadas y pruebas	07/05/2025	13	20/05/2025
5ª TUTORÍA MARISOL	14/05/2025	1	14/05/2025
SPRINT 5:Optimización,pruebas,docu	21/05/2025	16	06/06/2025
6ª TUTORÍA MARISOL	21/05/2025	1	21/05/2025
7ª TUTORÍA MARISOL	28/05/2025	1	28/05/2025
MEJORAS Y CORRECIONES	07/06/2025	3	10/06/2025
PRUEBAS FINALES Y ENTREGA	10/06/2025	2	12/06/2025
9ª TUTORÍA MARISOL	10/06/2025	1	10/06/2025
10ª TUTORÍA MARISOL	12/06/2025	1	12/06/2025
ENTREGA EMPRESA/FIN PRÁCTICAS			13/06/2025

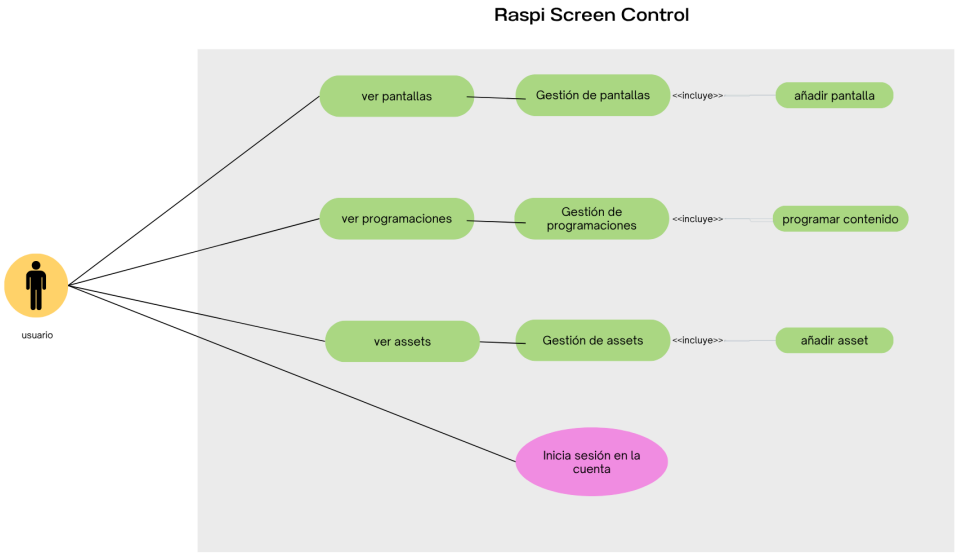


1.4 Diagrama de casos de uso

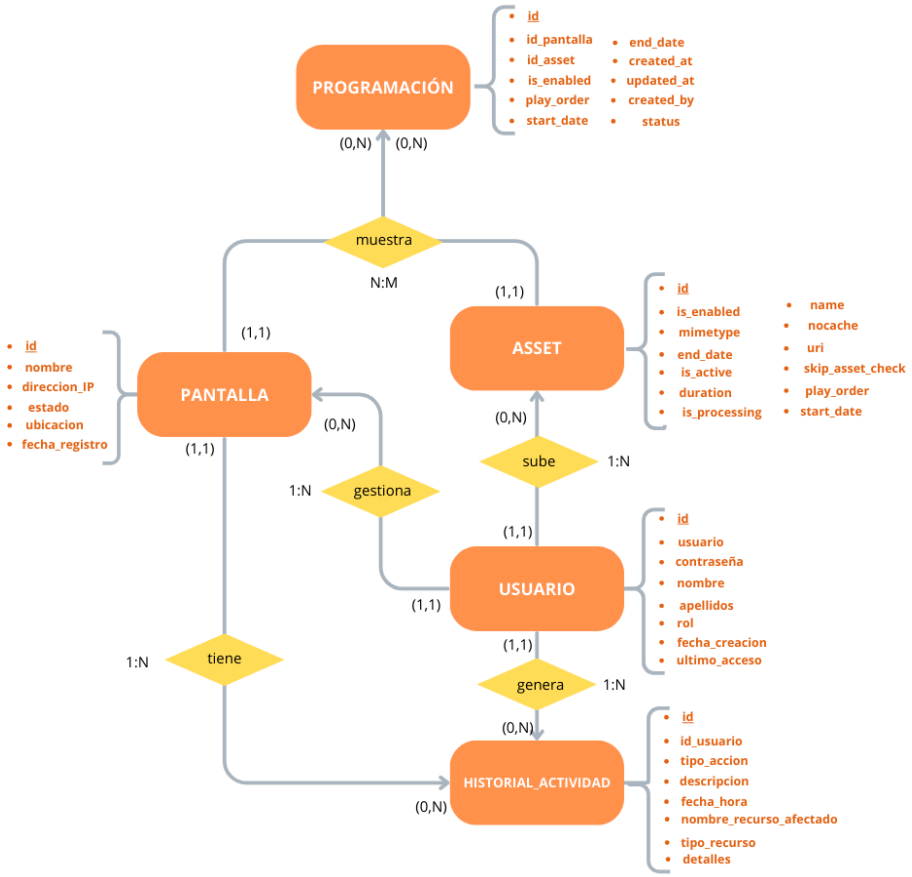
- Diafragma de casos de uso de la web



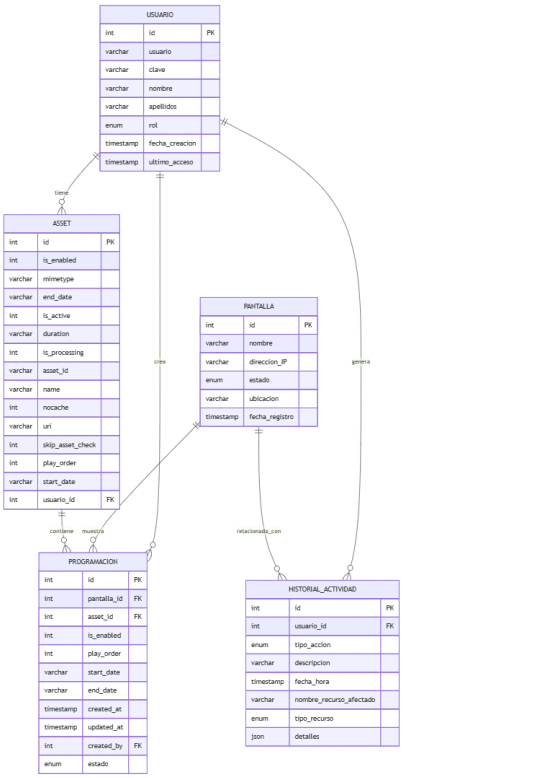
- Diafragma de casos de uso de la aplicación android



1.5 Modelo Entidad-Relación

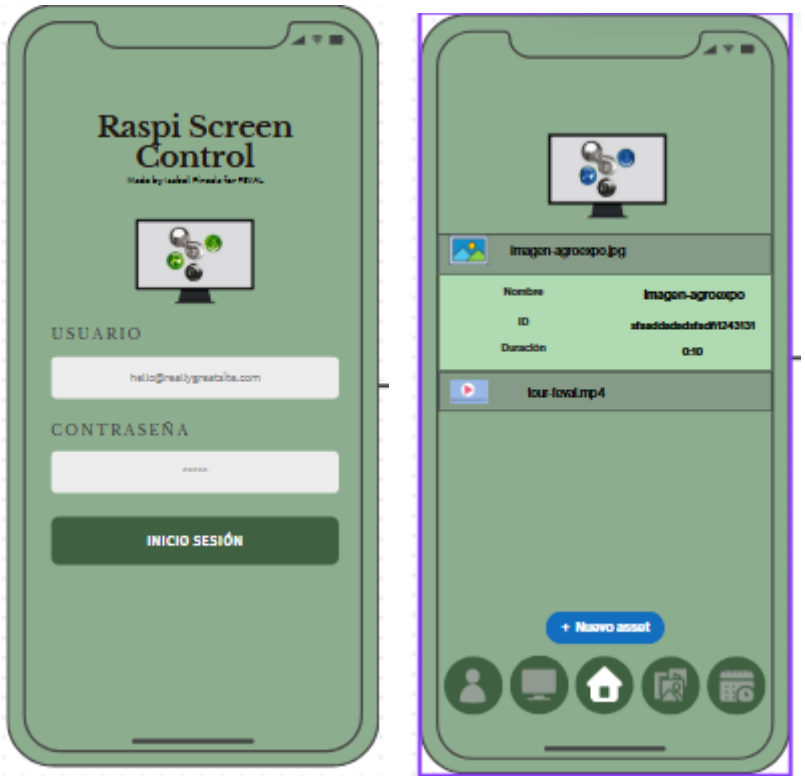


1.6 Modelo lógico



1.7 PROTOTIPO VISTAS WEB

1.8 PROTOTIPO VISTAS ANDROID



2. Arquitectura del sistema

El sistema se divide en tres medios principales, cada uno con un rol específico en la gestión de pantallas, assets y programación de contenido.

- Cliente Web (Dashboard de administración)
  ~ Usuarios: administradores y usuarios estándar.
  ~ Funciones principales:
    - Gestión de pantallas
    - Subida y gestión de assets multimedia
    - Programación de contenido en las pantallas
    - Gestión de usuarios
  ~ Tecnologías: React
- Cliente móvil (App android)
  ~ Usuarios: administradores y usuarios que necesiten gestionar las pantallas desde cualquier lugar
  ~ Funciones principales:
    - Subir pantallas
    - Subir y programar assets
    - Listar pantallas, assets y programaciones
  ~ Tecnologías: Java

- Servidor (API REST y base de datos)
  ~ Funciones principales:
    - Punto de comunicación entre la web, la app y la base de datos
    - Control de autenticación y gestión de permisos
    - Endpoints para consultas y gestión de datos
  ~ Tecnologías: Node.js, Express y MySQL.

2.1 Tecnologías utilizadas

Backend (Servidor)

- Lenguaje: JavaScript (Node.js)
- Framework: Express.js
- Base de datos: MySQL
- ORM/Query Builder: Sequelize
- Autenticación: JWT (JSON Web Tokens)
- Gestión de variables de entorno: dotenv

Aplicación web

- Framework: React.js
- Estado global: Context API

- Estilos: CSS/Tailwind CSS
- Routing: React router
- Consumo de API: Axios

  Aplicación android
- Lenguaje: Java
- IDE: Android Studio
- Consumo de la API: Retrofit2 + OkHttp + Gson
- Arquitectura de UI Android: Fragments + ViewPager2 + TabLayout + NavigationView + DrawerLayout
- Para pickers de fecha y hora: DatePickerDialog & TimePickerDialog

  Otras Tecnologías
- Canva: Diseño vistas
- Postman: Pruebas de API REST

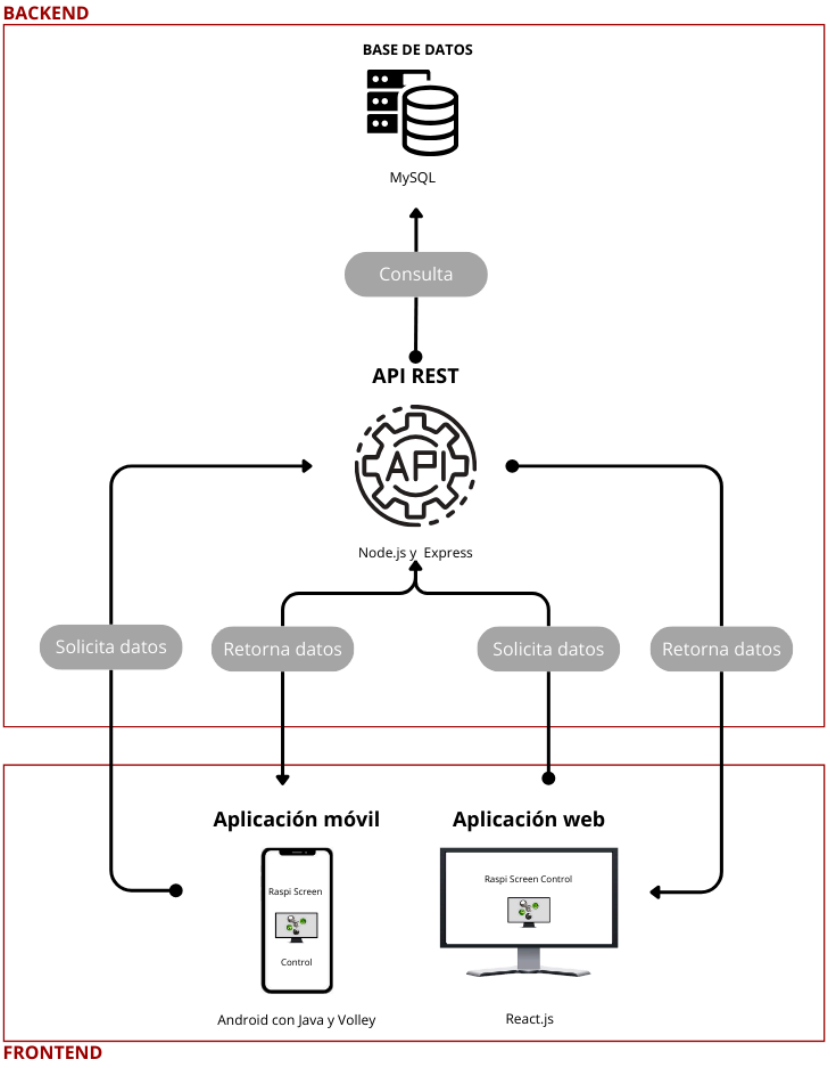
2.2 Relación entre componentes

La API REST actúa como intermediario entre la base de datos y los clientes (web y móvil)
El cliente web y la aplicación android consumen la API REST para gestionar contenido y pantallas.
La base de datos MySQL almacena usuarios, pantallas, assets y programación,
El cliente Android organiza su UI en tres pestañas (Pantallas, Assets, Programaciones) mediante un ViewPagerFragmentAdapter y se actualiza desde el backend usando callbacks de Retrofit.

Flujo de datos:

1. Usuario se autentica (`POST /api/auth/login`) → recibe JWT.
2. Llama a rutas protegidas (`/api/pantallas`, `/api/assets`, `/api/programacion`) con header `Authorization: Bearer <token>`.
3. Backend consulta MySQL y devuelve JSON snake_case.
4. Frontends (web/Android) mapean con GSON o fetch y actualizan UI.

. 2.3. Diagrama de Arquitectura



3. Estructura del código

3.1 Backend (API REST en Node.js y Express)

END POINTS PRINCIPALES

- ASSETS
  - GET /assets = listar todos
  - POST /assets = crear asset, envía JSON
- PROGRAMACIÓN
  - GET /programacion = listar todos, (JSON con snake_case: `pantalla_id`, `asset_id`, `start_date`, `end_date`, `play_order`)
  - POST /assets = crear programación

3.2 Aplicación Web

Estructura:

3.3 Aplicación Android (Java y Android Studio)

Lenguaje: Java
IDE: Android Studio
Consumo de la API: Retrofit2 (no Volley), con interceptor OkHttp para JWT y conversor Gson
Arquitectura UI:

- Fragments gestionados por un FragmentStateAdapter (ViewPagerFragmentAdapter)
- Pestañas: ViewPager2 + TabLayout
- Menú lateral: DrawerLayout + NavigationView
- Diálogos: AlertDialog
- DatePickerDialog + TimePickerDialog para elegir fechas
- Persistencia local: SharedPreferences (JWT)
- Dependencias principales:

  implementation 'com.squareup.retrofit2:retrofit:2.9.0'

  implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

  implementation 'androidx.recyclerview:recyclerview:1.2.1'

  implementation 'com.google.android.material:material:<versión>'

4. Estrategia de Pruebas

4.1 Pruebas Unitarias de la API

Postman: ejecutar cada método (GET, POST, PUT, DELETE) y verificar

- Respuestas HTTP 200/201 en éxito
- Códigos 400/404 ante datos inválidos o recursos no existentes

4.2 Pruebas de Integración (Web + API)

Flujo de usuario en la web:

1. Login → recibe token
2. Crear Pantalla → aparece en la lista
3. Crear Asset → listado actualizado
4. Programar Asset → visible en “Programaciones”

- Pruebas con las raspberries en la empresa

4.3 Pruebas de la App Android

- Pruebas de UI con DatePickerDialog y validación de formato (yyyy-MM-dd HH:mm:ss).
- Verificación en Logcat de los callbacks de Retrofit (Log.d("API",...), Log.e("API_ERROR",...)).