TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

Departamento de Informática

RADIO FALLOUT

Manual Técnico

Autor/es:Victor Silva MAtamoros
Curso Académico:2024/2025

Índice

1. Introducción

Fallout Radio es una aplicación web interactiva creada por y para seguidores de la emblemática saga de videojuegos *Fallout*, así como su reciente adaptación televisiva. Este proyecto nace con el propósito de **recrear una experiencia inmersiva** que transporte al usuario al universo post-apocalíptico del "Yermo", manteniendo una fidelidad estética y funcional a los elementos más reconocibles de la franquicia: el estilo **retro-futurista de los años 50**, la iconografía de Vault-Tec, y el mítico **Pip-Boy**.

Inspirada tanto en la ambientación sonora como visual de los juegos, Fallout Radio no solo actúa como reproductor musical, sino como **una experiencia envolvente** que aúna diseño, tecnología, narrativa y nostalgia gamer.

Objetivos del Proyecto

A continuación se detallan los objetivos específicos que guían el desarrollo de esta aplicación:

1.1 Crear una experiencia auténtica e inmersiva

Se ha diseñado una interfaz temática cuidadosamente inspirada en el universo Fallout. Mediante el uso de modelos 3D, tipografías customizadas, paletas de color retro-verdosas y elementos interactivos como el **Pip-Boy**, la radio o el televisor vintage, se genera una atmósfera coherente con la ambientación del videojuego, facilitando la inmersión emocional del usuario.

1.2 Ofrecer funcionalidades interactivas enfocadas al usuario

La aplicación permite:

- Escuchar emisoras temáticas de *Fallout 3*, *Fallout New Vegas* y *Fallout 4*.
- Reproducir canciones aleatorias.
- Marcar canciones como favoritas (previo login).
- Acceder a un reproductor exclusivo de favoritos.
- Navegar entre secciones interactivas como habitaciones y televisores retro.

Estas funciones se presentan mediante una interfaz intuitiva, con animaciones y transiciones que refuerzan la experiencia lúdica y estética del entorno Fallout.

1.3 Garantizar escalabilidad y adaptabilidad a cambios futuros

Gracias al uso de tecnologías modernas como:

- **React** para el frontend (SPA).
- **Supabase** como backend sin servidor (BaaS).
- **Vite** como bundler para optimización.
- **GSAP** y **Three.js** para animaciones y modelos 3D.

...la arquitectura resultante es fácilmente ampliable. Esto permite la integración futura de nuevas playlists, emisoras, funcionalidades como playlists personalizadas o incluso integración con APIs externas de música.

1.4 Proporcionar una experiencia completa, responsive y accesible

Se ha aplicado un enfoque **mobile-first**, asegurando que todos los componentes y secciones de la app funcionen de forma fluida en dispositivos móviles y tabletas. Se implementaron adaptaciones específicas para tamaños ≤ 480px, 360px y 320px, y se ha optimizado el rendimiento visual y la interacción táctil.

Público objetivo

Fallout Radio está dirigido principalmente a:

- Fans de la franquicia *Fallout*.
- Amantes de la música ambiental de videojuegos retro-futuristas.
- Usuarios que buscan una experiencia digital narrativa y estética.
- Estudiantes o desarrolladores interesados en ver una integración práctica de tecnologías web modernas aplicadas a un producto temático.

2. Arquitectura de la aplicación

2.1. Introducción

En cuanto a las herramientas empleadas para el desarrollo del proyecto **Fallout Radio**, se han utilizado principalmente las siguientes tecnologías y entornos de desarrollo:

- **Sistema Operativo**:
  - Linux (Ubuntu 22.04 LTS)
- **Entorno de Desarrollo Integrado (IDE)**:
  - Visual Studio Code (> v1.80.0)
- **Herramientas Frontend**:
  - React (con Hooks, Router DOM)
  - Vite (> v4.0.0)
  - React Three Fiber (para modelos 3D)
  - GSAP (GreenSock Animation Platform) (> v3.12.0)
  - CSS personalizado (Media Queries y diseño responsive)
- **Herramientas Backend**:
  - Supabase (Backend-as-a-Service)
    - Autenticación de usuarios
    - Base de datos PostgreSQL
    - Almacenamiento de archivos multimedia (Storage)
- **Aplicaciones y Herramientas Adicionales**:
  - GitHub (> v3.0.2) para control de versiones
  - Postman (> v10.0.0) para pruebas y gestión de peticiones HTTP
  - Figma para el diseño inicial y prototipado visual
  - lucid.app y app.diagrams.net para la creación de diagramas y documentación técnica
- **Despliegue y Hosting**:
  - Vercel para alojamiento frontend y despliegue automático desde GitHub
- **Aplicaciones en Consola de Comandos**:
  - NodeJS (> v18.0.0)
  - npm (> v9.0.0)
  - Git (> v2.40.0)
  - Vercel CLI (> v28.0.0)

En los siguientes apartados detallaremos cuáles han sido las tecnologías específicas utilizadas, así como los patrones de diseño y arquitectura aplicados durante el desarrollo del proyecto.

2.2. Backend

La implementación del backend de **Fallout Radio** se ha centrado en garantizar una comunicación eficiente y segura entre el cliente (desarrollado en React) y los servicios en la nube proporcionados por Supabase.

Partiendo del principio fundamental de desacoplar claramente las responsabilidades entre cliente y servidor, se han tenido en cuenta los siguientes aspectos arquitectónicos y de diseño:

- **Arquitectura Serverless**:
  El proyecto aprovecha la plataforma Supabase, un Backend-as-a-Service basado en PostgreSQL, que gestiona automáticamente la infraestructura de servidores, lo que simplifica enormemente la escalabilidad, el mantenimiento y las actualizaciones del backend.

- **Modelo Cliente-Servidor basado en REST**:
  Para la comunicación entre la aplicación cliente (React) y los servicios de Supabase se han utilizado endpoints REST (Representational State Transfer), los cuales permiten un manejo claro y estructurado de las peticiones HTTP y facilitan el intercambio de información en formato JSON.

- **Autenticación y autorización mediante JWT (JSON Web Token)**:
  Supabase proporciona autenticación integrada mediante JWT, que permite gestionar sesiones de usuario seguras y sin estados, facilitando así el intercambio seguro y eficiente de datos entre el cliente y el backend.

- **Estructura de datos y persistencia con PostgreSQL**:
  Se emplea PostgreSQL como motor de base de datos, gestionado por Supabase, para persistir información relacionada con usuarios, canciones favoritas y metadatos relacionados.

Las herramientas y características concretas empleadas del servicio Supabase son:

- **Supabase Database**:
  Gestor de bases de datos PostgreSQL integrado para la gestión eficiente y escalable de información como canciones favoritas, usuarios registrados y configuraciones específicas.
- **Supabase Storage**:
  Sistema de almacenamiento en la nube para alojar archivos multimedia (principalmente archivos MP3 de audio) accesibles a través de URLs públicas o privadas según la configuración aplicada.

- **Supabase Auth**:
  Módulo para la gestión segura y sencilla de usuarios mediante registro, inicio de sesión, recuperación de contraseñas y gestión de tokens JWT.

Para el manejo del código JavaScript en la parte frontend que interactúa con el backend, se han desarrollado funciones específicas reutilizables en los módulos:

- `supabaseClient.js`**:** Gestiona la conexión inicial y autenticación con Supabase.
- `favorites.js`**:** Maneja operaciones específicas relacionadas con canciones favoritas (añadir, eliminar y recuperar información almacenada).

Esta elección tecnológica permite una arquitectura simplificada, robusta y fácil de mantener, adecuada para un proyecto web orientado a la experiencia del usuario y el consumo de contenido multimedia como lo es **Fallout Radio**.

2.3. Frontend

Para el desarrollo de la parte cliente de la aplicación **Fallout Radio**, se han empleado las siguientes tecnologías modernas y ampliamente utilizadas en el ámbito del desarrollo web:

Lenguajes de programación y marcado

- **JavaScript** : lenguaje base empleado para toda la lógica de la aplicación.
- **JSX**: utilizado para describir la interfaz gráfica de manera declarativa en React.
- **HTML5**: estructura semántica básica de la aplicación.
- **CSS3**: estilos avanzados y personalizados para la interfaz.

Frameworks y librerías principales

- **React v18**: biblioteca principal utilizada para desarrollar la interfaz de usuario interactiva y eficiente.
- **React Router DOM**: facilita la navegación y el enrutamiento entre diferentes componentes y vistas de la aplicación.
- **Vite**: herramienta rápida y moderna para el empaquetado, la compilación y optimización del proyecto.

Librerías específicas complementarias

- **Supabase JS Client**: Librería cliente para interactuar fácilmente con la plataforma Supabase desde la aplicación React.
- **React Three Fiber**: Librería para representar modelos 3D interactivos dentro de React (empleada para la TV interactiva del login inicial).
- **GSAP (GreenSock Animation Platform)**: Librería avanzada de animaciones para efectos visuales fluidos y transiciones atractivas.
- **Anime.js**: Complemento utilizado puntualmente para animaciones específicas más dinámicas en ciertos componentes.
- **Axios (opcional según configuración)**: para realizar peticiones HTTP claras y eficientes (en caso de usar peticiones adicionales externas).

Diseño y adaptabilidad

- **Responsive Design**: mediante el uso avanzado de CSS Media Queries, la interfaz está completamente adaptada a dispositivos móviles de diferentes tamaños de pantalla (principalmente ≤ 480px, ≤ 360px, ≤ 320px).
- **Estilos personalizados**: Se han desarrollado hojas de estilo CSS específicas, con temática inspirada en la estética retro-futurista de Fallout, garantizando coherencia visual y una experiencia inmersiva y uniforme.

Estructura de carpetas principal del frontend

La aplicación React sigue una estructura clara y modular, orientada a componentes reutilizables. A continuación se presenta la estructura básica de carpetas del proyecto:

```
parallaxtfg
├ public
│ ├ efectosAudio   // Archivos de sonido para efectos y canciones
│ ├ img            // Imágenes y recursos visuales
│ └ models         // Modelos 3D (gltf)
├ src
│ ├ components     // Componentes React reutilizables
│ ├ data           // Datos estáticos o auxiliares usados por componentes
│ ├ Desuso         // Componentes o código no utilizado temporalmente
│ ├ lib            // Lógica específica del cliente y conexión backend
│ │ ├ favorites.js
│ │ └ supabaseClient.js
│ ├ App.jsx        // Componente principal de la aplicación
│ ├ index.css      // Estilos globales
│ └ main.jsx       // Punto de entrada de la aplicación React
├ index.html
├ package.json
├ vite.config.js
```

La elección de estas tecnologías y herramientas ha permitido construir una aplicación interactiva, visualmente atractiva y con un rendimiento óptimo, asegurando así una experiencia satisfactoria e inmersiva para todos los usuarios y seguidores del universo Fallout.

3. Documentación técnica

3.1. Análisis

A continuación, se exponen los requisitos funcionales y no funcionales implementados adecuadamente en la aplicación Fallout Radio. Para facilitar la comprensión y organización del contenido, se tomarán como referencia los distintos roles definidos dentro de la aplicación:

3.1.1. Requisitos funcionales

Usuario sin cuenta (invitado)

- Acceder a la pantalla inicial con el modelo 3D del televisor (TV interactiva).
- Seleccionar versión móvil o escritorio desde la TV.
- Acceder a la radio Pip-Boy, pudiendo:
    - Escuchar música desde emisoras temáticas (Fallout 3, New Vegas, Fallout 4).
    - Cambiar entre emisoras mediante botones físicos del Pip-Boy.
    - Utilizar el botón de reproducción aleatoria ("modo random").
- Navegar por las distintas secciones parallax del yermo.
- Acceder a la habitación final, donde puede interactuar con los objetos:
    - Pip-Boy (sin funciones de favoritos).
    - Radio (emisoras aleatorias).
    - Mr. Mañoso (da instrucciones).

Usuario registrado (autenticado con Supabase)

- Iniciar sesión o registrarse mediante formulario (email y contraseña).
- Acceder a todas las funcionalidades del usuario invitado.
- Gestionar canciones favoritas:
  - Añadir una canción desde el Pip-Boy a favoritos pulsando el botón del corazón.
  - Reproducir canciones favoritas desde el reproductor dedicado en la "habitación final".
  - Eliminar canciones desde el reproductor de favoritos.
- Reproducción de favoritos con controles:
  - Play, pausa, siguiente, anterior, volumen.
  - Reproducción aleatoria de favoritos.
- Ver indicador visual de canciones favoritas (corazón verde relleno).
- Cerrar sesión desde el menú superior.

3.1.2. Requisitos no funcionales

- **Diseño responsive completo**: la interfaz se adapta automáticamente a diferentes resoluciones de pantalla, con estilos específicos aplicados a móviles (≤480px, 360px y 320px) y escritorio, garantizando una experiencia fluida en todos los dispositivos.
- **Experiencia de usuario inmersiva e intuitiva**: la aplicación utiliza elementos visuales inspirados en la saga Fallout (estética retro-futurista, Pip-Boy, Mr. Mañoso, TV 3D interactiva), junto con animaciones suaves y una navegación clara, facilitando la exploración de sus funcionalidades.
- **Autenticación segura mediante Supabase**: se implementa un sistema de login y registro basado en correo electrónico y contraseña gestionado por Supabase Auth, garantizando una verificación fiable y segura de usuarios.
- **Arquitectura modular y mantenible**: la aplicación está dividida en componentes React reutilizables, con una estructura clara en carpetas (`components`, `views`, `lib`, `data`, etc.), lo que permite

escalar y extender nuevas funcionalidades sin comprometer la base del proyecto.

3.2. Desarrollo

En este apartado se presentan los elementos técnicos y estructurales más relevantes desarrollados para la aplicación. Incluye la representación de la base de datos utilizada, la cual ha sido creada y gestionada a través de **Supabase**, una plataforma backend como servicio que proporciona una base de datos **PostgreSQL**, almacenamiento de archivos y autenticación integrada.

3.2.1. Diseño de la Base de Datos

La base de datos de la aplicación ha sido diseñada de forma sencilla pero eficaz, centrada en la relación entre usuarios y canciones favoritas. A continuación se describe su estructura:

Tabla `favorites`

Esta tabla almacena las canciones que un usuario ha marcado como favoritas. Fue creada desde el panel SQL de Supabase, y cuenta con las siguientes columnas:

```sql
create table favorites (
  id uuid default uuid_generate_v4() primary key,
  user_id uuid references auth.users(id),
  song_url text,
  song_title text,
  created_at timestamp default now()
);

create index on favorites(user_id);
```

Relaciones

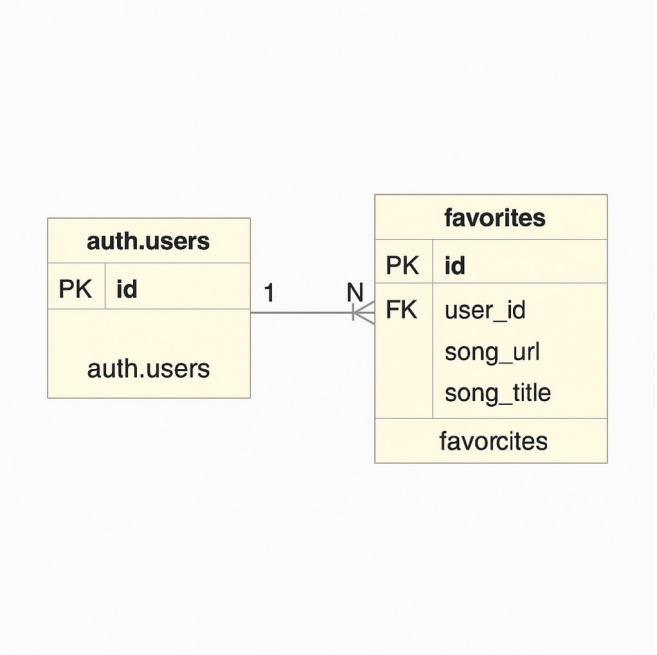
- **Relación 1:N entre `auth.users` y `favorites`**: cada usuario (gestionado automáticamente por Supabase Auth) puede tener múltiples canciones favoritas.
- **Restricciones**: `user_id` es clave foránea que referencia al sistema de autenticación interno de Supabase (`auth.users`).

Índice por `user_id`: mejora el rendimiento de las consultas para recuperar los favoritos de un usuario en concreto.

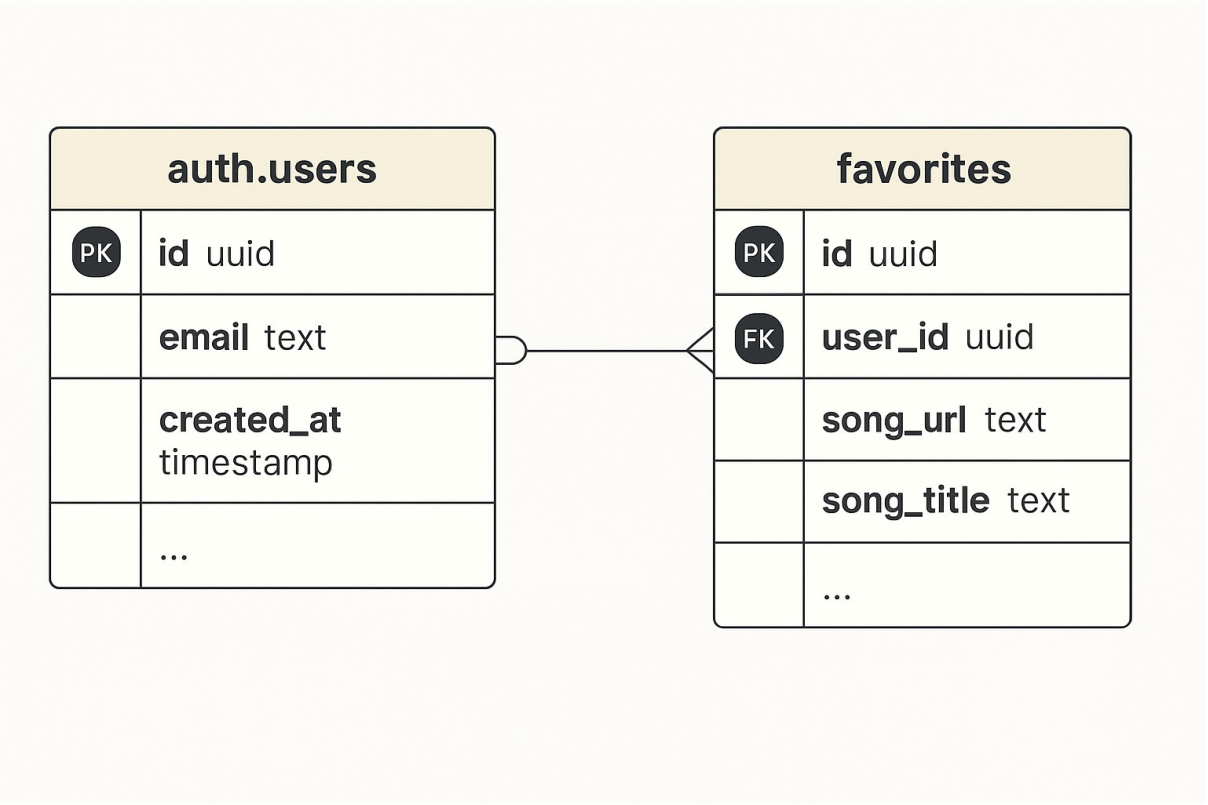
Almacenamiento de Archivos

Los archivos `.mp3` que corresponden a las canciones reproducidas están alojados en **Supabase Storage**, dentro de carpetas específicas como `fallout3`, `newvegas`, `fallout4`, organizadas por listas o juegos. Cada canción favorita contiene la URL correspondiente para su reproducción directa.

Modelo Entidad / Relación de la BBDD



Modelo relacional de la BBDD



3.2.2. Justificación del diseño del modelo relacional

¿Por qué ciertas relaciones presentes en el modelo entidad-relación se resuelven de forma distinta en el modelo relacional?

En el caso de **Fallout Radio**, es interesante observar cómo las distintas relaciones entre las entidades `auth.users`, `favorites`, y los recursos multimedia (canciones en Supabase Storage) han sido modeladas de manera distinta en el esquema relacional.

Esto se debe principalmente a la **cardinalidad y dependencia lógica** entre las entidades implicadas. A continuación se explican las dos relaciones clave y cómo han sido implementadas:

◈ Relación `auth.users` ↔ `favorites`

Esta relación se ha resuelto **mediante una clave foránea directa** (`user_id`) desde `favorites` hacia la tabla de autenticación de Supabase (`auth.users`). Se trata de una relación **1:N** (un usuario puede tener muchas canciones favoritas, pero cada favorito pertenece a un solo usuario).

Esta resolución mediante clave foránea simple es adecuada porque:

- La relación es **obligatoria y dependiente** (un favorito no existe sin un usuario).
- Se mantiene la integridad referencial con una **indexación eficiente** (`create index on favorites(user_id);`).

◈ Relación `favorites` ↔ Canciones (almacenadas en Supabase Storage)

Esta relación no se refleja como una clave foránea en una tabla tradicional, sino que **el campo `song_url` actúa como un enlace lógico al recurso multimedia almacenado en Supabase Storage**.

Esta resolución se ha hecho así por varias razones:

- Los archivos en Storage no están gestionados en una tabla explícita como entidad con ID (no hay una tabla `songs`, por ejemplo).
- La relación es **flexible y no necesariamente estructurada**, permitiendo reproducir canciones desde cualquier URL válida.
- Supabase Storage actúa más como un sistema externo vinculado por referencia que como una tabla relacional tradicional.

Conclusión

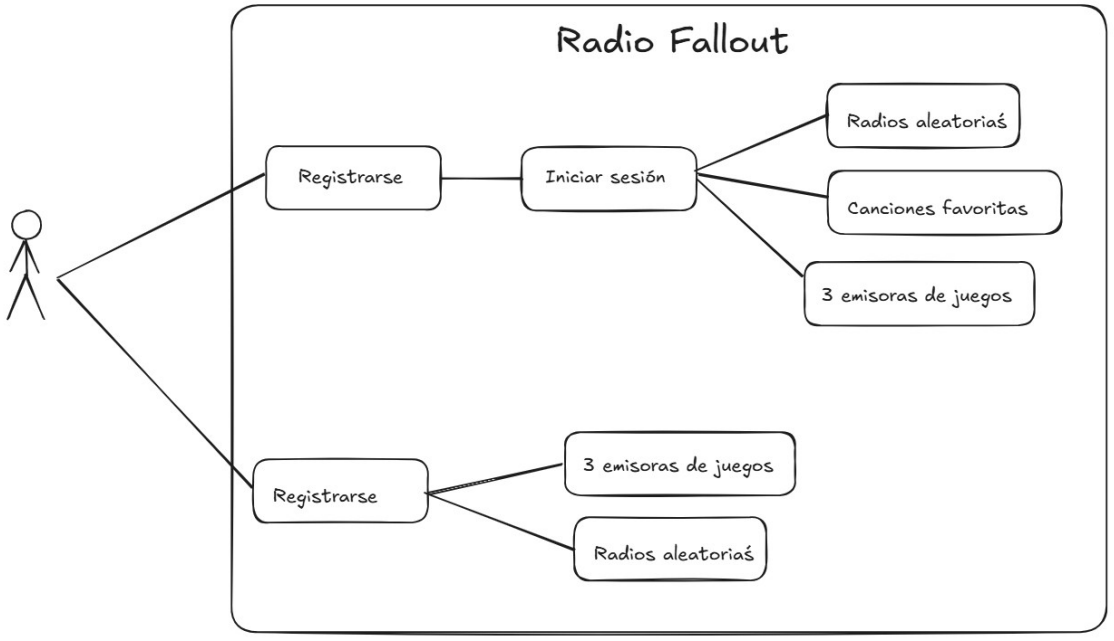
En resumen, las relaciones en la base de datos se resuelven de forma distinta en el modelo relacional en función de:

- La **cardinalidad** de las entidades implicadas (1:1 vs 1:N).
- La **naturaleza del recurso relacionado** (registro estructurado vs enlace a almacenamiento externo).

- La **necesidad de integridad referencial**, donde se usa clave foránea explícita solo cuando es necesario.

Esta decisión permite mantener una base de datos eficiente, escalable y perfectamente integrada con Supabase, facilitando además una lógica de recuperación rápida de datos desde React.

3.2.2. Diagrama de casos de uso



3.3. Pruebas realizadas

Para verificar la correcta implementación de los servicios y funcionalidades de la aplicación *Fallout Radio*, se llevaron a cabo pruebas manuales y exploratorias, principalmente centradas en los endpoints gestionados por **Supabase** y la correcta integración con el frontend desarrollado en **React**.

Aunque la aplicación no cuenta con una API RESTful desarrollada desde cero (como sería el caso de un backend en Java/Spring), se ha utilizado **Supabase** como Backend-as-a-Service (BaaS), el cual proporciona endpoints automáticos para autenticación, almacenamiento y operaciones CRUD sobre la base de datos.

Herramientas de prueba utilizadas

- **Supabase Studio**: Interfaz web para consultar tablas, ver registros insertados, modificar datos manualmente y observar logs en tiempo real.
- **Postman**: Se utilizó para realizar pruebas de autenticación, validación de tokens JWT proporcionados por Supabase y consultas directas sobre la REST API de Supabase (por ejemplo, para probar inserciones o eliminaciones en la tabla `favorites`).

Consola de navegador (DevTools): Útil para verificar errores en tiempo real, validar la respuesta de peticiones fetch y supervisar la carga de archivos mp3 desde Supabase Storage.

Ejemplo de flujo de pruebas

1. **Prueba de inicio de sesión**
   - Se accede a Supabase Auth mediante el formulario de inicio.
   - Se valida la respuesta de Supabase (`user !== null`) y que se almacene correctamente en el estado global de React.
   - Se comprueba que el usuario autenticado puede acceder al Pip-Boy, añadir favoritos y reproducirlos.
2. **Prueba de funcionalidad como invitado**
   - Se entra sin autenticar usando el botón “Entrar como invitado”.
   - Se comprueba que el botón de “Añadir a favoritos” muestra el mensaje de advertencia: “*Inicia sesión para agregar a favoritos”*.
   - El icono de favoritos se desactiva o emite un error visual al intentar acceder a la lista.
3. **Prueba de inserción y eliminación de favoritos (con Postman)**
   - Se simulan inserciones vía `POST` a la tabla `favorites`, utilizando un `Bearer Token` válido.
   - Se prueba que las canciones aparecen correctamente en el componente `ReproductorFavoritos`.
   - Se realiza un `DELETE` sobre un registro y se valida que desaparece del reproductor en React al recargar.

4. **Prueba de carga de archivos de audio**
    - Se comprueba en Supabase Storage que los `.mp3` estén correctamente subidos.
    - Se accede a ellos desde el navegador mediante su URL pública.

Se validan la reproducción secuencial, pausa, reanudación y cambio automático al finalizar.

4. Proceso de Despliegue

4.1. Introducción

Para el despliegue de la aplicación *Fallout Radio*, se ha optado por una arquitectura **frontend centrada** con servicios backend gestionados a través de **Supabase**, una solución Backend-as-a-Service moderna, eficiente y totalmente integrada con bases de datos PostgreSQL y almacenamiento de archivos.

La aplicación frontend, desarrollada con **React** y **Vite**, ha sido desplegada mediante la plataforma **Vercel**, por su facilidad de uso, integración directa con GitHub y óptimo rendimiento en aplicaciones SPA.

Esta elección ha simplificado considerablemente el flujo de despliegue, eliminando la necesidad de montar un servidor propio en Heroku o similares. Toda la lógica de backend (autenticación, almacenamiento, y base de datos) se aloja en Supabase, lo que facilita tanto el desarrollo como el mantenimiento.

Además, contar con un repositorio en **GitHub** no solo ha permitido una mejor gestión del código y versiones, sino que también ha sido necesario para enlazar el proyecto con Vercel de forma automática.

4.2. Despliegue del Frontend en Vercel

El despliegue de *Fallout Radio* se realizó utilizando **Vercel**, una plataforma moderna para alojar proyectos frontend desarrollados con frameworks como **React** y **Vite**. Esta herramienta destaca por su integración directa con **GitHub**, permitiendo desplegar automáticamente el proyecto con cada nueva actualización en el repositorio.

A continuación, se describen los pasos seguidos para realizar el despliegue:

1. Acceso e inicio de sesión

Se accede a la página principal de Vercel y se selecciona la opción “**Continuar con GitHub”**. Esto enlaza la cuenta de GitHub del desarrollador con Vercel, permitiendo importar repositorios directamente.

2. Importación del repositorio

Una vez iniciada la sesión, Vercel muestra todos los repositorios públicos y privados asociados a la cuenta de GitHub. Se selecciona el repositorio del proyecto *Fallout Radio*.

3. Configuración inicial del proyecto

Al importar el repositorio, Vercel detecta automáticamente el framework (en este caso, **Vite**) y configura las opciones de despliegue. No es necesario modificar la configuración por defecto, salvo definir las **variables de entorno** necesarias para conectar con Supabase, como:

```
VITE_SUPABASE_URL=https://<nombre-del-proyecto>.supabase.co
VITE_SUPABASE_ANON_KEY=eyJhbGciOi...
```

Estas variables se añaden en el apartado **Environment Variables** dentro de la configuración del proyecto en Vercel.

4. Despliegue inicial

Se hace clic en “**Deploy”** para iniciar la primera implementación. Vercel construye la aplicación automáticamente y despliega el resultado en una URL pública del tipo:

```
https://fallout-radio.vercel.app
```

5. Actualizaciones automáticas

Gracias a la integración con GitHub, cualquier cambio que se realice en la rama principal del repositorio desencadena automáticamente una nueva compilación y despliegue. Esto garantiza que la versión publicada esté siempre actualizada con la última versión del código.

5. Propuestas de Mejora

El desarrollo de *Fallout Radio* ha supuesto no solo la implementación de un proyecto completo, sino también un proceso formativo continuo, en el que se ha aprendido a trabajar con herramientas modernas como React, Supabase y Vite. A pesar de haber alcanzado los objetivos iniciales, existen diversas áreas susceptibles de mejora o expansión. A continuación se detallan por ámbitos:

5.1 Funcionalidades Pendientes

Editor de perfil de usuario	Actualmente no existe una sección para editar nombre o correo desde la interfaz.
Gestión completa de listas personalizadas	Posibilidad de crear, renombrar o eliminar playlists personalizadas por el usuario.
Sistema de búsqueda avanzada	Filtro por canción, artista o tipo dentro de las emisoras o favoritos.
Historial de reproducción	Mostrar al usuario las últimas canciones escuchadas.
Modo oscuro alternativo	Adaptar la interfaz a diferentes estilos según preferencia del usuario.
Comentarios o votaciones	Permitir valorar o comentar canciones desde la interfaz (como feedback).

5.2 Mejora Técnica del Backend (Supabase y lógica)

Mejora técnica	**Justificación**
Separación lógica de funcionalidades en archivos más modulares	Actualmente algunas funciones como login, favoritos o reproducción comparten espacio y podrían separarse para mayor claridad.
Control de errores más robusto	Capturar fallos en la autenticación, inserción o carga de favoritos para mostrar feedback al usuario.
Validación de datos	Comprobar si los datos enviados al backend (como URLs o títulos de canciones) cumplen ciertos criterios antes de ser guardados.
Optimización de queries	Aprovechar más filtros o paginación en las llamadas a Supabase para escalar mejor si aumenta el

	volumen de datos.
Auditoría o logs internos	Implementar registro de acciones de usuario (opcional) para análisis posterior del uso.

5.3 Mejora Técnica del Frontend (React y estilo visual)

Mejora del código CSS	Unificar estilos comunes en archivos globales y reducir repeticiones.
Mayor modularización de componentes	Algunos componentes pueden dividirse en subcomponentes (por ejemplo, PipBoy, botones físicos, reproductor de favoritos).
Implementación de contexto global o Zustand	Para compartir el estado de sesión o favoritos entre componentes sin prop drilling.
Sistema de notificaciones internas	Feedback al usuario al añadir favoritos, errores de conexión, etc.

6. Webgrafía

6.1. Cursos online y tutoriales utilizados

Plataforma	Curso / Contenido	Autor / Canal
YouTube	**Tutorial completo de React con Vite y Firebase/Supabase**	***Fazt Code***
YouTube	**Introducción a Supabase y alternativas a Firebase**	***Midudev***
YouTube	**Curso de React para principiantes**	***FreeCodeCamp***
YouTube	**Curso completo de React y animaciones prácticas**	***The Net Ninja***
GSAP Docs	**ScrollTrigger y Timeline: Guías prácticas**	***GreenSock***

de animación

6.2. Webs de referencia

Sitio web	Descripción
ReactJS.org	Documentación oficial de React.
Supabase.io	Plataforma backend como servicio (BaaS).
ViteJS.dev	Herramienta de build ultrarrápida para proyectos web.
greensock.com	Animaciones avanzadas con GSAP.
StackOverflow	Comunidad para resolver dudas técnicas.
MDN Web Docs	Referencia completa de tecnologías web.
CSS-Tricks	Consejos y técnicas avanzadas de CSS.

6.3. Artículos y documentación técnica

- Supabase Auth Docs: guía oficial de autenticación.
- JWT.io – Introducción a JSON Web Tokens: explicación detallada de los tokens de autenticación.

6.3.2. Estructura de proyectos con Vite y React

- "How to Structure a Large React App": recomendaciones de la documentación oficial.
- Artículos técnicos y ejemplos prácticos en GitHub sobre Vite + React.

6.3.3. Diseño responsive y accesibilidad

- **A11Y Project: buenas prácticas de accesibilidad web.**
- **Responsive Design Basics – web.dev: fundamentos del diseño adaptable.**

6.3.4. Animaciones con GSAP

- **Documentación oficial de GSAP: referencia completa para animaciones.**
- **Colección de ejemplos en CodePen: inspiración visual y técnica.**