Diseño Técnico

YGGDRASOUND

YGGSO

Atteneri de Jesús Torres González

Nombre del fichero:	DAW_PRW_YGGSO_UT01.4. Diseno Tecnico.odt
Fecha de esta versión:	16/01/2026

Historial de revisiones

Fecha	Descripción	Autor
12/01/2026	Creación del documento “Diseño Técnico”	Atteneri de Jesús Torres González
16/01/2026	Cambio del nombre del documento	Atteneri de Jesús Torres González

ÍNDICE

1 INTRODUCCIÓN

Este documento técnico describe la arquitectura y las tecnologías empleadas en el desarrollo del proyecto ***Yggdrasound***, una plataforma de comercio electrónico especializada en la venta de vinilos. Su objetivo es ofrecer una visión clara y estructurada de las decisiones técnicas adoptadas durante el desarrollo, sirviendo como documentación de referencia para el mantenimiento, la evolución y la escalabilidad futura del proyecto.

El proyecto consiste en el desarrollo de una aplicación web **Full-Stack** como proyecto final del ciclo de **Desarrollo de Aplicaciones Web (DAW)**. La aplicación integra funcionalidades clave propias de un comercio electrónico, como un catálogo de productos, carrito de compra, sistema de gestión de usuarios y características avanzadas como listas de deseos, garantizando una experiencia de usuario completa y coherente.

2 REQUISITOS TÉCNICOS

2.1 Plataforma y herramientas de desarrollo software

2.1.1. Requisitos front-end

A continuación se enumeran las diferentes tecnologías de desarrollo front-end, versiones y herramientas de apoyo al desarrollo:

I) Tecnologías / framework

- React | Versión 19.2.3
- Javascript
- Tailwind CSS | Versión 4.1

II) Otras Herramientas de desarrollo, depuración, etc

- Visual Studio Code | Versión más reciente
- Vite (empaquetador y servidor de desarrollo) | Versión 7.2.7
- Zustand (gestión de estados) | Versión 5.0.10

2.1.2. Requisitos back-end

A continuación se enumeran las diferentes tecnologías de desarrollo back-end, versiones y herramientas de apoyo al desarrollo.

I) Plataforma de desarrollo NODE.JS

a) Framework, runtime y versiones

- Runtime: Node.js | Versión 22.18.0
- Framework: Express.js | Versión 5.2.1

b) Esquema de desarrollo

- Arquitectura API REST
- Patrón Modelo-Controlador (MC) con servicios

c) IDE

- Visual Studio Code | Versión más reciente
- Extensiones: ES7+ React/Redux/JS snippets, Tailwind Css Intelissense, Error Lens, Prettier

d) Otras herramientas o gestores de proyecto

- npm | Versión 10.9.3
- Git | Versión 2.51.0
- Docker | Versión 29.1.3
- Docker Compose | 5.0.1

d) Otras dependencias

- Sequelize (ORM) | Versión 6 - stable
- TablePlus (Gestor de BBDD)
- Postman/Insomnia/Nuncius (Cliente HTTP)

2.2 Plataforma de Ejecución

2.2.1. Servidor

A continuación se indican las características técnicas de mínimos para la ejecución de la aplicación descrita en el proyecto:

- **Servidor web y versión:**
  Node.js con framework Express
  - Node.js: versión 22.18.0
  - Express: versión 5.2.1
- **Sistema operativo y versión:**
  - Entorno de desarrollo: Windows 11
- **Runtime/Interprete y versión**:
  - Node.js | Versión 22.18.0
- **Observaciones a la configuración:**
  - En el entorno de desarrollo, la base de datos PostgreSQL se ejecuta dentro de un contenedor Docker independiente.

- Las aplicaciones frontend (React) y backend (Express) se ejecutan directamente en el sistema host utilizando sus respectivos servidores de desarrollo (Vite para el frontend y Node.js para el backend).

- La dockerización completa de la aplicación (frontend, backend y base de datos) se contempla como una fase final del proyecto, con el objetivo de facilitar su despliegue y escalabilidad.

2.2.2. Alojamiento físico/ Entornos de ejecución

En este apartado se describen los distintos entornos de ejecución previstos para la aplicación, así como la ubicación y forma de acceso a cada uno de sus componentes según la fase del proyecto.

- **Entorno de desarrollo:**
  - **Frontend:**
    Ejecutado en entorno local mediante Vite, accesible desde http://localhost:5173.
  - **Backend (API):**
    Servido localmente mediante Node.js y Express, accesible desde http://localhost:3000.
  - **Base de datos:**
    PostgreSQL ejecutándose en un contenedor Docker independiente, accesible desde localhost:5432.
- **Entorno de preproducción / pruebas:**
  - Uso del mismo entorno local descrito para desarrollo.
  - Posible despliegue temporal en servicios gratuitos para pruebas y demostraciones:
    - **Frontend:** Vercel.
    - **Backend y base de datos:** Railway o Render.
- **Entorno de producción:**
  - **Frontend:**
    Despliegue en Vercel, plataforma orientada a aplicaciones frontend basadas en React.
  - **Backend (API):**
    Despliegue en Railway o Render**.**
  - **Base de datos:**
    PostgreSQL.

- **Alternativa de despliegue:**
  Implementación de un entorno completamente dockerizado que incluya frontend, backend y base de datos, una vez finalizada la fase de dockerización del proyecto.

2.3 Puntos de acceso a la aplicación

En este apartado se detallan los puntos de acceso (URLs) a la aplicación según el entorno de ejecución.

- **Entorno de desarrollo:**
  - **Frontend:**
    http://localhost:5173
  - **Backend (API REST):**
    http://localhost:3000/api

3 BASE DE DATOS

La aplicación utiliza una base de datos relacional PostgreSQL, elegida por su robustez, rendimiento y compatibilidad con entornos de desarrollo y despliegue basados en contenedores.

- **Sistema gestor de base de datos:** PostgreSQL
- **Versión:** 15 (imagen Docker postgres:15 utilizada mediante Docker Compose)
- **Nombre de la base de datos:** yggdrasound_db
- **Esquema:** public (esquema por defecto de PostgreSQL)

Modelo de Datos

El modelo de datos es relacional y normalizado, diseñado para garantizar la integridad de la información y facilitar la escalabilidad del sistema. Las principales entidades que componen el modelo son las siguientes:

- **user:** usuarios registrados en la plataforma.
- **credit_card:** tarjetas de crédito asociadas a los usuarios.
- **vinyl:** catálogo de vinilos disponibles para la venta.
- **genre:** géneros musicales.
- **order_history:** historial de pedidos realizados por los usuarios.
- **shopping_cart:** carritos de compra activos.
- **wish_list:** listas de deseos asociadas a cada usuario.
- **comment:** comentarios realizados por los usuarios sobre los productos.

Asimismo, el modelo incluye varias tablas intermedias para representar relaciones de tipo muchos a muchos:

- **vinyl_genre:** relación entre vinilos y géneros musicales.
- **order_vinyl:** relación entre pedidos y vinilos.
- **cart_vinyl:** relación entre carritos de compra y vinilos.
- **wishlist_vinyl:** relación entre listas de deseos y vinilos.

4 INTERFACES EXTERNAS

En este apartado se describen las interfaces externas con las que interactúa la aplicación, tanto a nivel de comunicación interna entre componentes como con servicios externos.

- **API REST interna:**
  La aplicación tendrá una API REST propia, desarrollada con Express.js, accesible bajo el prefijo /api/*. Esta API actúa como intermediaria entre el frontend y la base de datos, proporcionando acceso a los recursos necesarios para el correcto funcionamiento de la plataforma, como el catálogo de productos, la gestión de usuarios, los pedidos y las listas de deseos.
- **Consulta externa mediante Web Scraping (conciertos):**
  El backend incorpora una funcionalidad opcional de web scraping utilizando la librería Cheerio, que permite consultar de forma controlada la página de Bandcamp de un artista concreto cuando el usuario lo solicita. Esta consulta se emplea exclusivamente para verificar la existencia de conciertos programados, mostrando dicha información únicamente en la ficha del artista correspondiente.

5 SEGURIDAD

La aplicación implementa diversas medidas de seguridad orientadas a proteger la información de los usuarios y garantizar un acceso controlado a los recursos del sistema.

- **Autenticación:**
  Sistema de autenticación basado en JSON Web Tokens (JWT), utilizado para validar la identidad de los usuarios en las peticiones a la API.
- **Gestión de contraseñas:**
  Las contraseñas de los usuarios se almacenan de forma segura mediante hashing con bcrypt, evitando en todo momento el almacenamiento de contraseñas en texto plano.
- **Protección de la API:**
  Configuración de CORS (Cross-Origin Resource Sharing) restringida exclusiva-

mente al dominio del frontend, impidiendo el acceso no autorizado desde orígenes externos.

- **Acceso a base de datos:**
  Todas las consultas a la base de datos se realizan a través del ORM Sequelize, lo que permite el uso de consultas parametrizadas y reduce el riesgo de ataques de inyección SQL.

6 CONTROL DE VERSIONES

El desarrollo del proyecto se gestiona mediante un sistema de control de versiones distribuido, lo que permite un seguimiento adecuado de los cambios y facilita el mantenimiento del código.

- **Sistema de control de versiones:**
  Git, con repositorio alojado en GitHub.
- **Estrategia de ramas:**
  Se emplea una estructura de ramas sencilla y clara:
  - **main:** rama estable que contiene versiones funcionales del proyecto.
  - **dev:** rama destinada al desarrollo activo e integración de nuevas funcionalidades.
  - **feature/***: ramas utilizadas para el desarrollo de funcionalidades específicas de forma aislada.
- **Gestión de commits:**
  Los commits se realizan con mensajes claros y descriptivos, facilitando la trazabilidad de los cambios y la comprensión del historial del proyecto.

7 OBSERVACIONES

En este apartado se recogen algunas consideraciones generales sobre la arquitectura, las funcionalidades y el estado actual del proyecto.

- **Arquitectura**:
  La aplicación está estructurada en dos bloques principales: un frontend desarro llado en React y un backend basado en una API REST. Esta separación permite una organización clara del proyecto y facilita su mantenimiento y evolución.

- **Base de datos:**
  El proyecto utiliza una base de datos PostgreSQL, ejecutada en un contenedor Docker durante el desarrollo, lo que permite trabajar en un entorno controlado y reproducible.

- **Funcionalidades clave:**
  - Catálogo de vinilos con sistema de filtrado.
  - Consulta opcional de conciertos mediante web scraping desde Bandcamp.
  - Sistema de compra simulado, sin integración con pasarela de pago real.

- Despliegue:
  - Despliegue del frontend y backend previsto con Vercel y Railway/Render.
  - Posibilidad de realizar una dockerización completa de la aplicación como mejora futura.

8 JUSTIFICACIÓN DE LAS TECNOLOGÍAS EMPLEADAS

Las tecnologías utilizadas en este proyecto se han elegido por encajar bien con los requisitos de la aplicación y por la experiencia previa adquirida en otros cursos y proyectos personales. Esto permitirá desarrollar la aplicación con mayor seguridad y centrarse más en la funcionalidad que en la curva de aprendizaje. En el caso de tecnologías como Docker y PostgreSQL, su uso ha supuesto una oportunidad de aprendizaje adicional, permitiendo adquirir nuevos conocimientos relacionados con la gestión de bases de datos y la contenedorización de aplicaciones.

React se ha utilizado para el frontend porque facilita la creación de interfaces dinámicas y reutilizables. JavaScript se emplea como lenguaje principal por su compatibilidad con los navegadores y su integración directa con React.

Tailwind CSS permite crear estilos de forma rápida y mantener una interfaz consistente y responsive. Vite se ha usado como servidor de desarrollo por su rapidez y simplicidad. Zustand se ha elegido para la gestión del estado global por ser ligero y suficiente para el tamaño del proyecto.

Node.js se utiliza en el backend por su buen rendimiento en aplicaciones basadas en peticiones HTTP. Express.js se ha elegido por su sencillez y flexibilidad a la hora de crear una API REST clara y bien organizada.

La API REST permite separar correctamente el frontend del backend y facilita la comunicación entre ambos mediante peticiones HTTP y datos en formato JSON.

PostgreSQL se ha utilizado como base de datos por su fiabilidad y por adaptarse bien a un modelo relacional. Sequelize simplifica el acceso a la base de datos y ayuda a mantener las consultas seguras.

Para la seguridad, se emplea JWT para la autenticación de usuarios y bcrypt para el almacenamiento seguro de contraseñas. La configuración de CORS limita el acceso a la API únicamente al frontend autorizado.

Docker se utiliza durante el desarrollo para ejecutar la base de datos y mantener un entorno consistente. Git y GitHub se emplean para el control de versiones y la organización del desarrollo del proyecto.

En conjunto, la combinación de estas tecnologías permite una base sólida para el desarrollo de la aplicación, equilibrando el uso de herramientas ya conocidas con la incorporación de nuevas tecnologías aprendidas durante el proyecto. Esto facilita tanto el correcto funcionamiento actual de la aplicación como su posible evolución y mejora en el futuro.