

Desarrollo de Aplicaciones Multiplataforma Online



GENERALITAT
VALENCIANA

CONSELLERIA D'EDUCACIÓ,
INVESTIGACIÓ, CULTURA I ESPORT



Florida

Universitària

Proyecto Final de Ciclo de Desarrollo de Aplicaciones Multiplataforma Online

**Título: Plataforma de gestión
de cursos de capacitación de
empleados**

Ismael Riera Alberó

21/05/2025

Índice

| | |
|--|-----------|
| Índice | 1 |
| Resumen del Proyecto | 2 |
| Justificación y Objetivos del Proyecto | 4 |
| Objetivos para el usuario final | 4 |
| Objetivos Técnicos de Desarrollo | 5 |
| Desarrollo del proyecto | 6 |
| Metodologías utilizadas | 6 |
| Descripción de los componentes de la aplicación: | 10 |
| Base de datos | 10 |
| Arquitectura del Backend | 12 |
| Tecnologías Utilizadas en el Backend | 12 |
| Arquitectura frontend | 16 |
| Tecnologías Utilizadas en el Frontend | 16 |
| Problemas, dificultades, y soluciones adoptadas | 20 |
| Bibliografía | 25 |

Resumen del Proyecto

Este proyecto consiste en el desarrollo integral de una plataforma digital de formación orientada a la gestión de cursos de capacitación para empleados, con un enfoque especial en las necesidades de las pequeñas y medianas empresas (pymes). Su objetivo principal es proporcionar una herramienta eficiente, segura y fácil de usar tanto para administradores como para usuarios finales (formadores y alumnos).

El desarrollo se ha llevado a cabo de forma individual, siguiendo una estructura secuencial tipo *cascada*, comenzando por el diseño de la base de datos, seguido del backend y, finalmente, del frontend. Esto ha permitido mantener una visión clara de los requerimientos y las dependencias entre capas, así como validar funcionalidad por etapas.

El sistema se articula sobre tres pilares fundamentales:

1. Base de Datos

Se ha utilizado **PostgreSQL** como sistema de gestión de base de datos relacional. La estructura se ha normalizado siguiendo buenas prácticas, abarcando entidades como usuarios, cursos, tests, progresos, roles y certificados. Adicionalmente, los PDF de los cursos y certificados se almacenan en la nube usando la api de googleDrive

2. Backend

El backend se ha implementado con **Node.js** y **Express**, y actúa como intermediario entre la base de datos, el almacenamiento externo y el cliente. Entre sus responsabilidades destacan:

- Gestión de operaciones **CRUD** sobre las distintas entidades.
- Sistema de **autenticación** y **autorización** basado en **JWT**, que permite proteger rutas y restringir accesos según el rol del usuario.
- Cifrado de contraseñas con **bcrypt** para asegurar la privacidad de las credenciales.

- Generación dinámica de **certificados PDF** utilizando **PDFKit**, personalizados con los datos del curso y del alumno.
- Comunicación con la **API de Google Drive** para la subida y recuperación de archivos de curso y certificados.

3. Frontend

El frontend se ha desarrollado utilizando **React**, junto con la biblioteca de componentes **Material UI v7**, que permite crear interfaces limpias y accesibles respetando los principios del diseño responsive. Entre las herramientas utilizadas se incluyen:

- **Axios**, para la comunicación HTTP con el backend.
- **Zustand**, para la gestión del estado global de la aplicación, de manera ligera y simple.
- **React Router DOM v7**, para manejar el enrutamiento en una SPA (Single Page Application).
- **Diseño atómico**, como patrón estructural, dividiendo los componentes en átomos, moléculas, organismos, plantillas y páginas. Esto ha facilitado la reutilización y el mantenimiento del código visual.

Justificación y Objetivos del Proyecto

En muchas pymes, la gestión de la formación de los empleados sigue realizándose de forma manual, mediante hojas de cálculo extensas, poco intuitivas y propensas a errores. Incluso en empresas de mayor tamaño, este proceso implica una carga administrativa considerable.

Por parte de los usuarios, también se detectan barreras: muchas soluciones existentes están excesivamente corporativizadas o integran funcionalidades innecesarias, lo que dificulta su uso en determinados perfiles del entorno laboral.

Desde el punto de vista del desarrollo, este proyecto representa una oportunidad para construir una aplicación completa y funcional, de principio a fin, utilizando tecnologías modernas como la API de Google Drive, la generación dinámica de PDFs con PDFKit, el diseño con Material UI, y metodologías actuales como el diseño atómico.

Frente a este contexto, surge esta plataforma automatizada y accesible que centraliza todo el proceso formativo: desde la creación y asignación de cursos, hasta la realización de pruebas y la obtención de certificados digitales.

Objetivos para el usuario final

Para el Usuario

- **Acceso a los cursos:** El usuario podrá visualizar los contenidos formativos asignados a través de una interfaz clara y ordenada. El material estará disponible principalmente en formatos PDF y presentaciones, accesibles desde cualquier dispositivo.
- **Realización de tests:** El backend generará tests vinculados a cada curso, diseñados para comprobar que el usuario ha comprendido correctamente los contenidos. El sistema registra automáticamente los resultados.
- **Descarga de certificados:** Una vez completado un curso y superado el test, el usuario podrá descargar un certificado personalizado en formato PDF, que incluye su

nombre, la fecha de finalización y el nombre del curso.

Para el Formador

- **Creación de cursos:** Los formadores tendrán la capacidad de crear nuevos cursos, incluyendo la carga de materiales didácticos y la definición de preguntas de test para las pruebas de evaluación.
- **Gestión de usuarios:** Se podrá registrar nuevos empleados en el sistema, asignándoles el rol correspondiente (usuario o formador), lo que permite un control jerárquico y funcional del acceso.
- **Asignación de cursos:** El sistema permitirá asignar cursos específicos a usuarios concretos. Esta acción genera automáticamente un seguimiento individualizado para cada combinación de usuario y curso.
- **Seguimiento formativo:** A través de un panel de administración, los formadores podrán consultar el estado de avance de los usuarios, detectar cuándo un curso está por caducar o si ya ha expirado, y actuar en consecuencia para mantener al día la formación del equipo.

Objetivos Técnicos de Desarrollo

- **Gestión de PDFs:** Implementar la generación automática de certificados en formato PDF desde el backend, incluyendo datos dinámicos como el nombre del usuario y la fecha de finalización.
- **Almacenamiento en la nube:** Integrar el sistema con Google Drive para almacenar tanto los archivos de los cursos como los certificados generados, facilitando el acceso y reduciendo la carga local del servidor.

- **Seguimiento temporal:** Diseñar una lógica backend capaz de gestionar fechas de inicio, finalización, y caducidad de cursos, permitiendo alertas o automatismos para facilitar la planificación de formadores y usuarios.
- **Encriptación:** Utilizar técnicas de cifrado como bcrypt para garantizar la seguridad de las contraseñas y proteger la integridad de los datos personales.
- **Diseño de interfaz:** Crear una interfaz moderna, visualmente coherente y accesible, utilizando herramientas como Material UI y técnicas como el diseño atómico para asegurar una experiencia fluida en todos los dispositivos.

Desarrollo del proyecto

Metodologías utilizadas

Enfoque de desarrollo

El proyecto siguió un modelo secuencial (tipología Waterfall) adaptado a mi trabajo individual:

1. **Fase de datos:** Diseño y creación de la base de datos PostgreSQL con su modelo entidad-relación.
2. **Fase de lógica de negocio:** Implementación del backend en Node.js/Express, con APIs REST que operan sobre esa base de datos.
3. **Fase de presentación:** Construcción del frontend en React/MUI, consumiendo los endpoints ya operativos.

Aunque inicialmente organicé el trabajo en tres fases (base de datos → backend → frontend), pronto descubrí que un enfoque tan secuencial tiene sus inconvenientes:

- **Retraso en la retroalimentación:** Cambios en el frontend que requerían ajustes en el modelo de datos o en las rutas del API obligaban a regresar a fases anteriores.
- **Rigidez:** No permitía introducir nuevas funcionalidades de forma incremental sin rehacer partes ya cerradas.
- **Dependencia total entre fases:** Hasta que el backend no estaba completamente listo, el frontend no podía avanzar, y viceversa.

Control de versiones

Para gestionar el código fuente del proyecto se utilizó **Git** como sistema de control de versiones distribuido, junto con un **repositorio remoto alojado en GitHub**. Esta práctica permitió llevar un seguimiento organizado y seguro de los cambios realizados a lo largo del desarrollo, además de facilitar la trazabilidad, la reversión de errores y la colaboración futura si se desea escalar el proyecto.

Diseño de componentes (frontend)

A pesar de que el enfoque de Atomic Design resultó ser uno de los grandes retos del desarrollo debido a muchos factores, demostró ser un enfoque muy beneficioso a la hora de construir el frontend final. Tener preparados los componentes pequeños y ser capaz de reconfigurarlos supuso a la larga un gran ahorro de tiempo

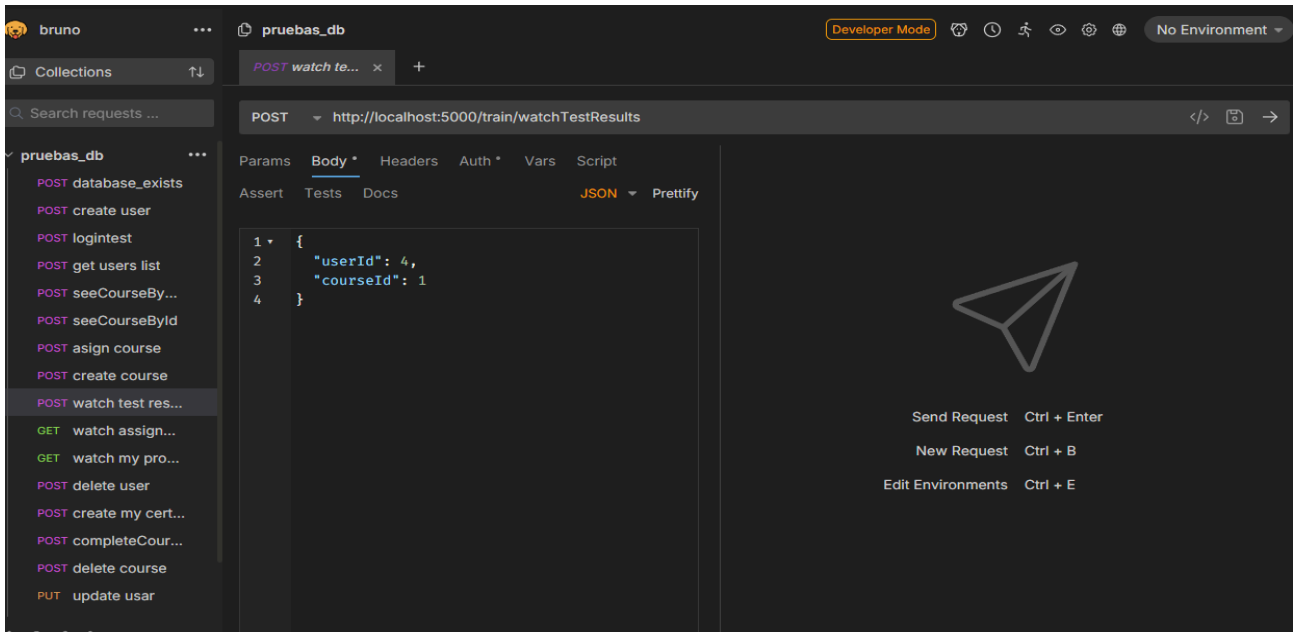
Arquitectura y separación de capas

Tanto para la construcción del backend como la interfaz de frontend han intentado ceñirse a una estructura modular y profesional, separando las responsabilidades en capas jerarquizadas

- Capas de backend: Capa de enrutamiento, capa de control, middleware de autenticación, capa de lógica de negocio
- Capas de frontend: Capas jerarquizadas según el diseño atómico: Átomos, moléculas, organismos, plantillas, páginas.

Gestión de la calidad

Para poder realizar pruebas y comprobar la funcionalidad y calidad de los componentes se han utilizado herramientas como Bruno

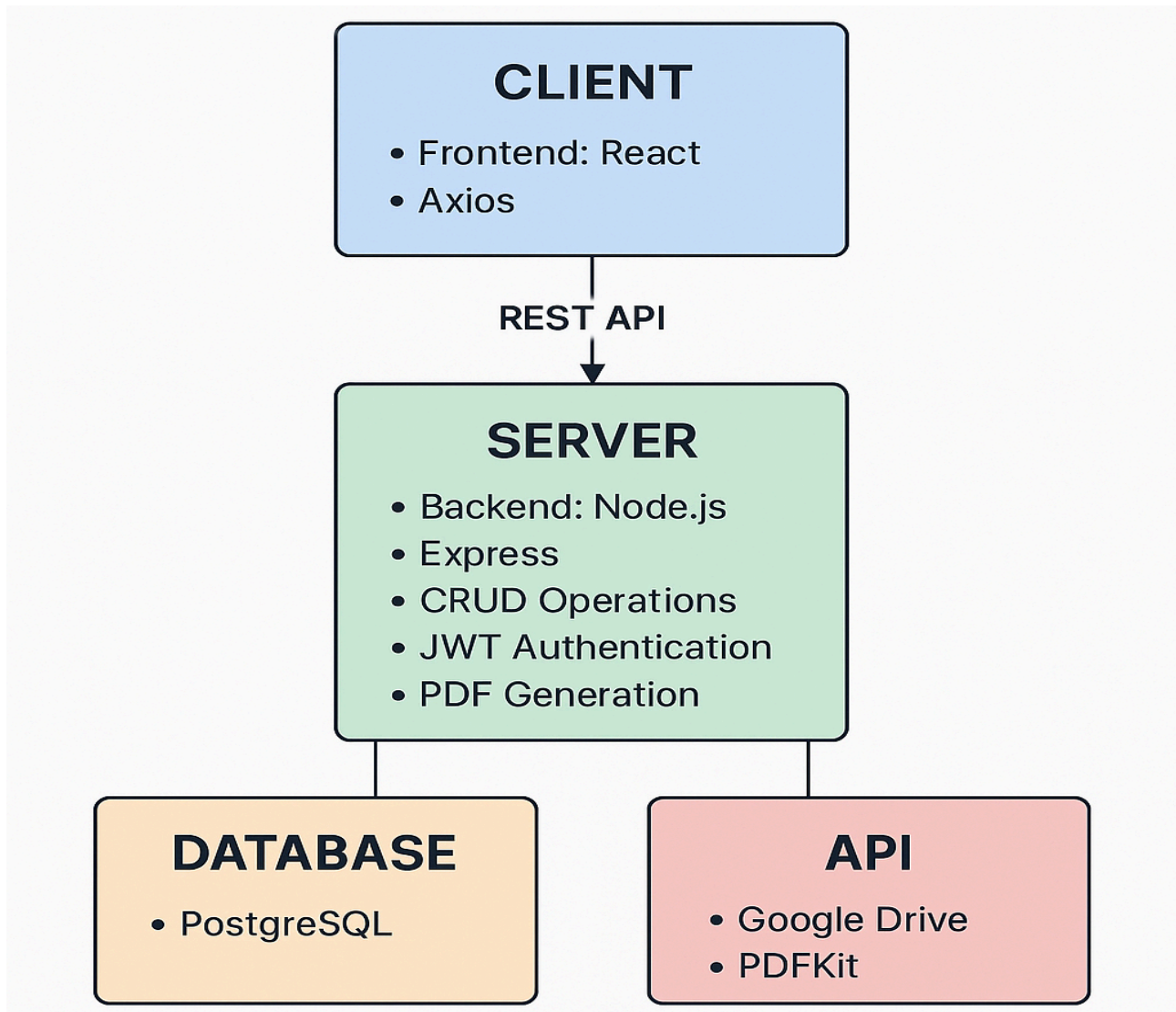


Simultáneamente, el uso de complementos y herramientas de linting y formateo como ESLint y Prettier ayudaron a mantener un código limpio y estructurado.

Documentación y comunicación

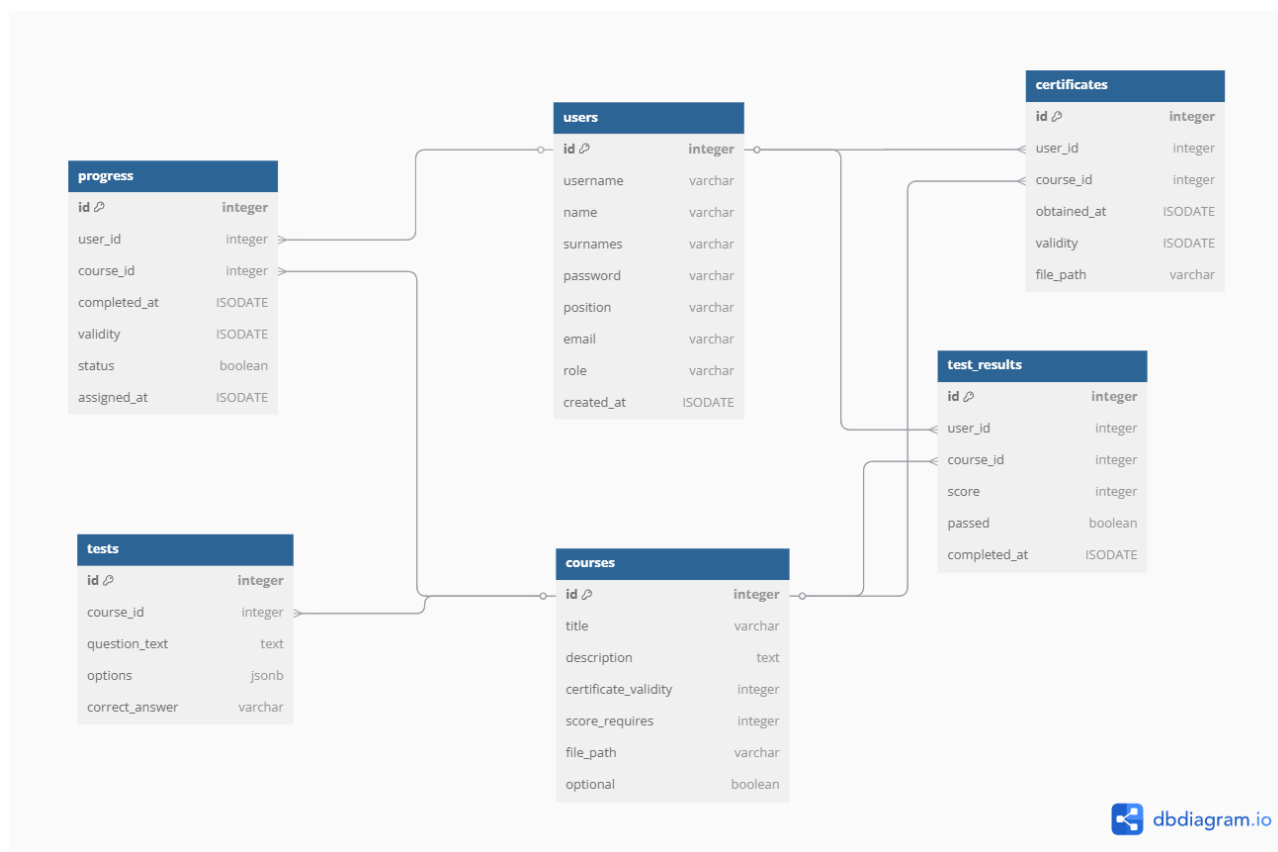
La elaboración de documentación y diagramas, sobre todo en las fases iniciales de modelaje de la base de datos, supuso un cimiento fundamental del proyecto y permitió que el arranque del mismo fuera más fluido y ágil, facilitando también la modificación de componentes realizados en fases anteriores.

Descripción de los componentes de la aplicación:



Base de datos

Aunque la opción de usar **MongoDB** resultaba atractiva por su sencillez y flexibilidad inicial, finalmente opté por **PostgreSQL** debido a la necesidad de contar con una base de datos relacional robusta, segura y más adecuada para un sistema con múltiples relaciones y claves foráneas.



La estructura gira en torno a dos tablas principales: **usuarios** y **cursos**. A partir de ellas se articulan el resto de las funcionalidades básicas mediante tablas secundarias:

- **Usuarios:** Almacena los datos personales y credenciales de acceso, además de la información necesaria para la generación de certificados y la gestión de asignaciones y seguimientos.
- **Cursos:** Contiene la información básica de cada curso, así como la ruta del archivo asociado en Google Drive que contiene el material formativo.
- **Tests:** Guarda las preguntas de evaluación de cada curso, registradas de forma individual mediante JSON para permitir la composición dinámica de pruebas desde el backend.
- **Progreso:** Esta tabla actúa como nexo entre usuarios y cursos. Cada vez que se asigna un curso a un empleado, se crea una entrada única que permite hacer un seguimiento del estado del curso.

- **Certificados:** Registra los certificados generados automáticamente tras completar un curso, incluyendo las fechas de validez y la ruta de acceso al archivo en Google Drive.
- **Resultados:** En esta tabla se guardan los resultados de los test realizados por los usuarios para poder verificar que la obtención del certificado es legítima.

Arquitectura del Backend

Tecnologías Utilizadas en el Backend

- **Sequelize:** ORM (Object-Relational Mapping) para interactuar con la base de datos **PostgreSQL**, permitiendo escribir consultas de forma más abstracta y organizada.
- **pg & pg-hstore:** Drivers necesarios para conectar Sequelize con PostgreSQL y manejar datos del tipo JSON.
- **Jsonwebtoken:** Librería para generar y verificar **tokens JWT** usados en la autenticación de usuarios.
- **bcrypt:** Utilizado para **encriptar contraseñas** antes de almacenarlas en la base de datos, aumentando la seguridad.
- **Cors:** Habilita el uso de **CORS (Cross-Origin Resource Sharing)**, necesario para que el frontend pueda comunicarse con el backend desde dominios distintos.
- **Multer:** Middleware para gestionar la **subida de archivos**, útil en la gestión de PDFs antes de subirlos a Google Drive.
- **Pdfkit:** Librería para **generar certificados en formato PDF** de forma programática desde el backend.
- **googleapis:** Cliente oficial para trabajar con la **API de Google Drive**, utilizado para subir, leer y gestionar los archivos de los cursos y certificados.

El backend está desarrollado con **Node.js** y utiliza el framework **Express.js** para la gestión de rutas y peticiones HTTP. La arquitectura sigue un modelo **modular y desacoplado**, dividiendo el código en diferentes capas y responsabilidades.

Vamos a exponer como ejemplo el endpoint que sirve para la asignación de cursos a usuarios :

1. Capa de Rutas (Routing)

Define los distintos endpoints disponibles, agrupados por funcionalidad (usuarios, cursos, tests, certificados...). Cada ruta recibe la petición y la redirige al controlador correspondiente. Usa middlewares como el de autenticación con **JWT** para proteger rutas privadas por login y por rol de usuario.

```
trainRouter.post("/assignCourse",authenticateToken, requireTrainerRole, newProgress)
```

2. Middlewares

Funciones intermedias para la autenticación de las peticiones y el manejo de errores. También se utiliza middleware para el control de acceso según el rol del usuario.

```
export const requireTrainerRole = (req, res, next) => {
  if (req.user?.role !== "trainer") {
    return res.status(403).json({ message: "Access denied" });
  }
  next();
};

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(" ")[1];

  if (!token) {
    return res.status(401).json({ message: "No token provided" });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(403).json({ message: "Invalid or expired token" });
  }
};

export default authenticateToken;
```

2. Controladores (Controllers)

Encargados del manejo de la petición, de comunicarse con la capa de acceso a datos y de evaluar la respuesta o código de error del servidor: siguiendo el ejemplo, el controlador recibe los datos de la petición, envía el body a la capa lógica, y envía la respuesta.

```
export const newProgress = async (req, res) => {
  try {
    const newAssignment = await assignCourse(req.body);
    sendResponse(res, 200, true, "Assignment created successfully", { newAssignment });
  } catch (error) {
    if (error.message === "Missing field") {
      return res.status(400).json({ message: "Missing field" });
    }
    if (error.message === "Course already assigned") {
      return res.status(409).json({ message: "Course already assigned" });
    }
  }
};
```

3. Modelos o Consultas a Base de Datos (DB layer)

Gestionan la interacción con PostgreSQL a través de la herramienta de sequelize, utilizando conexiones asíncronas para aliviar la carga de trabajo.

```
export const assignCourse = async ({ userId, courseId }) => {
  try {
    if (!userId || !courseId) {
      throw new Error("Missing field");
    }
    const courseIsAssigned = await db.Progress.findOne({
      where: { user_id: userId, course_id: courseId },
    });
    if (courseIsAssigned) {
      throw new Error("Course already assigned");
    }
    const today = new Date();
    return await db.Progress.create({
      user_id: userId,
      course_id: courseId,
      completed_at: null,
      assigned_at: today,
      status: false,
      validity: null,
    });
  } catch (error) {
    throw new Error(error.message || "No se pudo asignar el curso.");
  }
};
```

En este punto del ejemplo accedemos a la base de datos, validamos que el curso no haya sido asignado anteriormente al usuario, y creamos y devolvemos una nueva entrada en el registro.

6. Variables de Entorno

Toda la configuración sensible (claves secretas, IDs de API, rutas...) está almacenada en variables de entorno cargadas desde un archivo `.env`, protegido por `.gitignore`.

```
import dotenv from "dotenv"
dotenv.config();

const sequelize = new sequelize("training_db", "postgres", process.env.DB_PASSWORD,
{
  host: process.env.DB_HOST,
  dialect: "postgres",
  logging: false,
});
export default sequelize;
```

7. Asincronía y Rendimiento

Todas las operaciones intensivas (como llamadas a Google Drive, lectura de archivos o generación de PDFs) están gestionadas de forma **asíncrona** para evitar bloqueos y mejorar el rendimiento.

```
export const generateCertificate = async (user, course) => {
  const doc = new PDFDocument({ size: "A4", layout: "landscape" });

  const certDir = path.resolve("certificados");
  if (!fs.existsSync(certDir)) fs.mkdirSync(certDir);

  const filename = `certificado_${user.username}_${course.title}.pdf`;
  const filePath = path.join(certDir, filename);

  const writeStream = fs.createWriteStream(filePath);
  doc.pipe(writeStream);

  doc.fontSize(26).text("CERTIFICADO DE FINALIZACIÓN", { align: "center" });
  doc.moveDown();
  doc.fontSize(20).text(`Otorgado a: ${user.username}`, { align: "center" });
  doc.moveDown();
  doc.text(`Por completar el curso: ${course.title}`, { align: "center" });
  doc.moveDown();
  doc.text(`Fecha: ${new Date().toLocaleDateString()}`, { align: "center" });

  doc.end();

  return new Promise((resolve, reject) => {
    writeStream.on("finish", () => resolve(filePath));
    writeStream.on("error", reject);
  });
};
```


Arquitectura frontend

Tecnologías Utilizadas en el Frontend

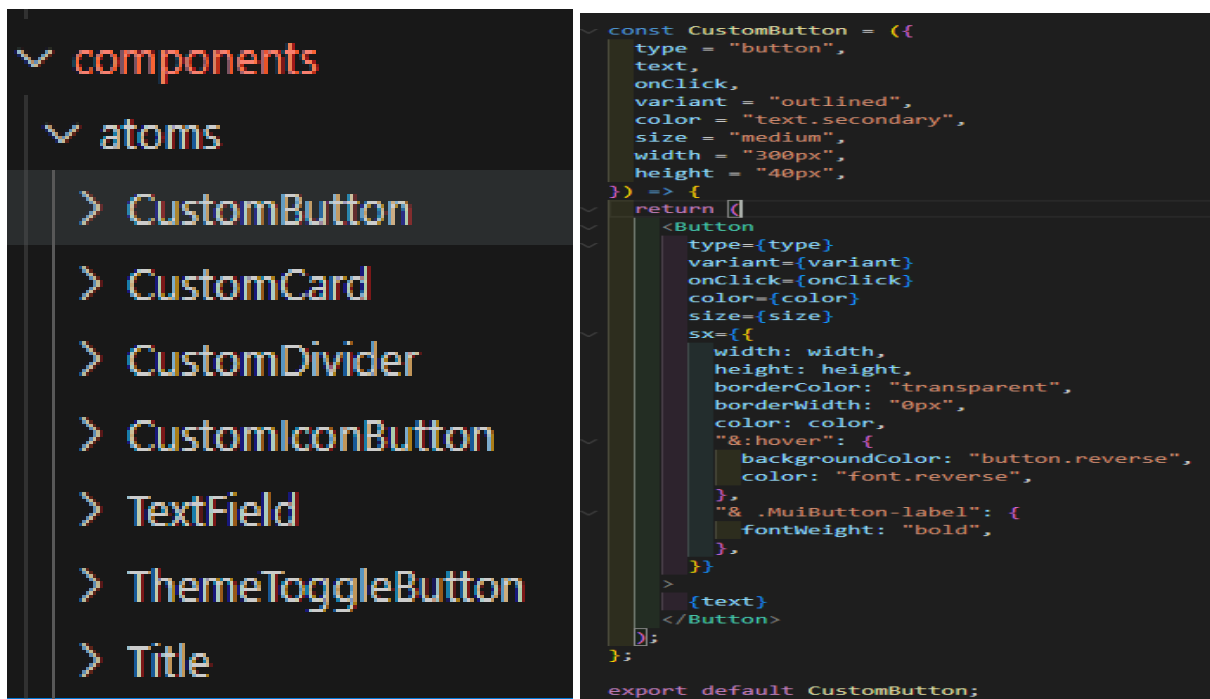
- **React:** Librería principal para construir la interfaz de usuario mediante componentes reutilizables.
- **React Router DOM (v7):** Gestiona el enrutamiento dentro de la aplicación SPA (Single Page Application), permitiendo navegación entre vistas sin recargar la página.
- **Material UI (v7):** Librería de componentes UI basada en el diseño de Google Material. Junto con **@emotion/react** y **@emotion/styled**, se usa para estilizar los componentes con temas personalizados y CSS-in-JS.
- **Axios:** Cliente HTTP para hacer peticiones al backend de forma sencilla y eficiente.
- **Zustand:** Librería ligera de manejo de estado global, usada como alternativa más simple a Redux o Context API.
- **jwt-decode:** Permite decodificar tokens JWT en el frontend para obtener información del usuario (como el rol o el ID) sin necesidad de consultar al backend.
- **Vite:** Herramienta moderna de construcción y desarrollo con recarga rápida y eficiente. Utilizada para compilar, servir y construir el proyecto.

Diseño de componentes con Atomic Design

El frontend se estructuró siguiendo la metodología **Atomic Design**, lo que ha permitido crear una librería de componentes escalable y fácil de mantener:

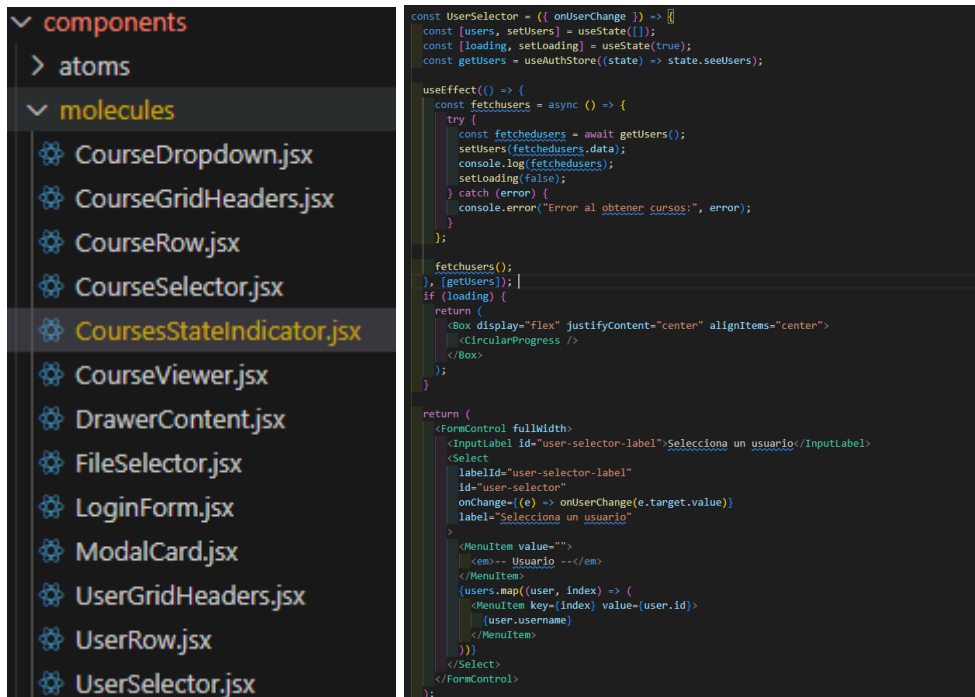
Átomos

Son elementos básicos e indivisibles, como botones (**CustomButton**), divisores o campos de texto. Aquí definimos estilos genéricos y las props necesarias para utilizarlos en distintos contextos.



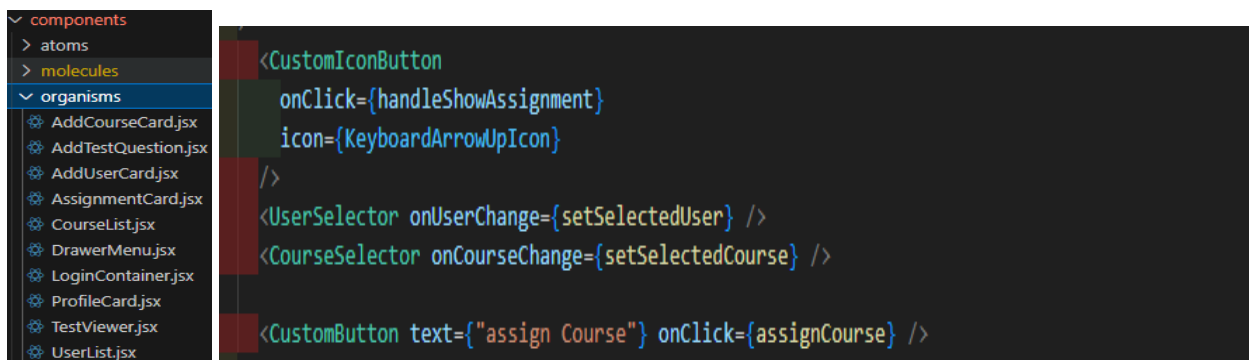
Moléculas

Agrupan átomos para formar componentes con una funcionalidad mínima propia. Un ejemplo es el selector de usuario (**UserSelector**) o de curso (**CourseSelector**), que combina un input y un desplegable para ofrecer una unidad reutilizable en varias vistas.



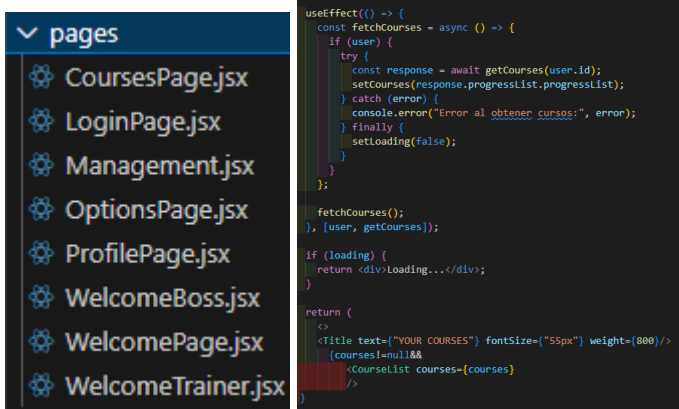
Organismos

Son bloques completos e independientes, contruidos a partir de moléculas y átomos. Por ejemplo, la **AssignmentCard** —que aparece en la página de gestión— reúne **UserSelector**, **CourseSelector**, **CustomButton** y **CustomIconButton** para ofrecer la interfaz de asignación de cursos de un vistazo.



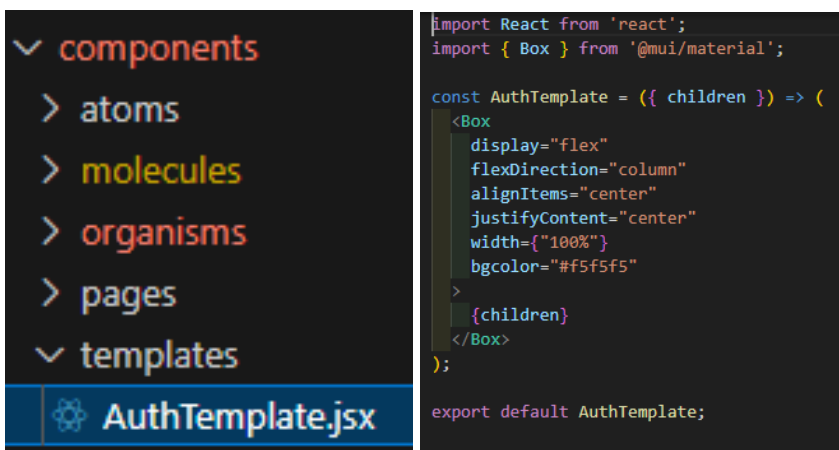
Páginas

Integran la lógica de datos y renderizan múltiples organismos. En cada página (por ejemplo, “Management”) se realizan las **peticiones con Axios** y se extraen datos de la **store de Zustand**, pasando la información resultante como props a los organismos.



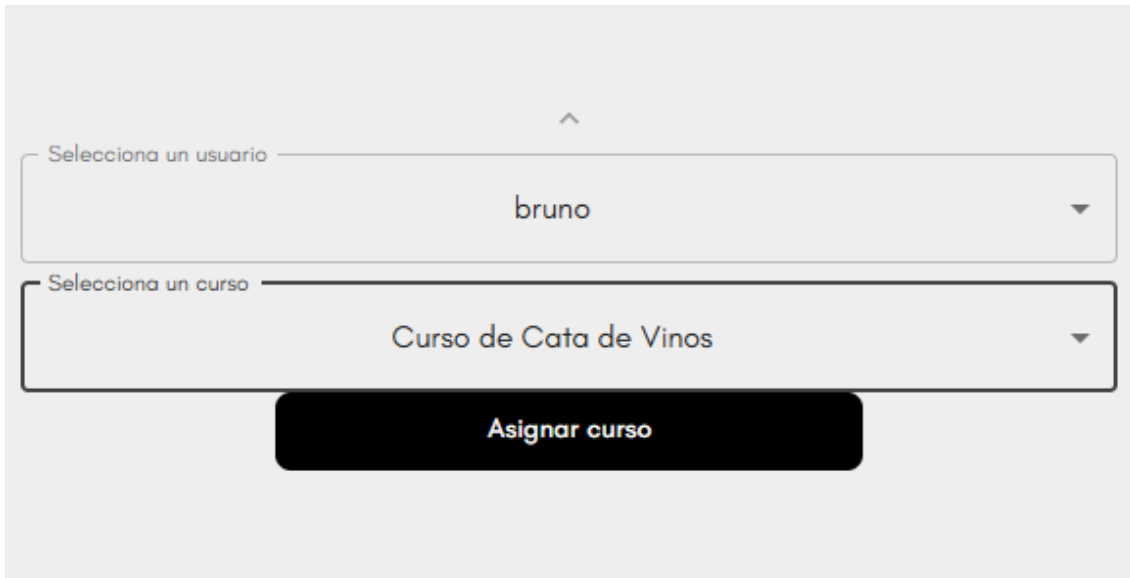
Plantillas

Las plantillas deben encargarse de la disposición de los elementos en una página y, aunque uno de los objetivos del desarrollo era utilizar la metodología de Atomic Design al completo, esta parte de la jerarquía todavía se me escapa a nivel conceptual y solo se pudo integrar una template sencilla para la pantalla de login



Siguiendo el ejemplo del backend, vamos a describir los componentes implicados en el envío de una petición para la asignación de un curso a un usuario

Organismo **AssignmentCard**



The image shows a web form titled "AssignmentCard". It contains two dropdown menus. The first dropdown is labeled "Selecciona un usuario" and has "bruno" selected. The second dropdown is labeled "Selecciona un curso" and has "Curso de Cata de Vinos" selected. Below these two dropdowns is a black button with the text "Asignar curso" in white.

Trae los átomos (**CustomButton**, **CustomIconButton**, **Title**), moléculas (**UserSelector**, **CourseSelector**) y la store de autenticación (**useAuthStore**).

Utiliza dos estados locales con **useState()** para guardar temporalmente **selectedUser** y **selectedCourse**.

Al hacer clic en el botón “assign Course”, el handler llama al método **assignACourse** obtenido de la store, pasándole los IDs seleccionados.

```
const [selectedCourse, setSelectedCourse] = useState();
const [selectedUser, setSelectedUser] = useState();
const assignACourse = useAuthStore((state) => state.assignCourse);

const assignCourse = async () => {
  try {
    await assignACourse(selectedUser, selectedCourse);
  } catch (error) {
    console.error("Error al añadir usuario:", error);
  }
};
```

Store de Zustand (**useAuthStore**)

```
assignCourse: async (selectedUser, selectedCourse) => {  
  const assignedCourse = await AuthService.assignCourse({  
    userId: selectedUser,  
    courseId: selectedCourse,  
  });  
  return assignedCourse;  
},
```

Este método actúa como fachada: recibe los IDs del componente y delega en **AuthService.assignCourse**, devolviendo la respuesta.

Servicio HTTP (**AuthService.assignCourse**)

```
async assignCourse(data) {  
  const serverHost = import.meta.env.VITE_SERVER_HOST;  
  const response = await axios.post(serverHost + "/train/assignCourse", data);  
  return response;  
}
```

El servicio construye la url del backend usando la variable de entorno **VITE_SERVER_HOST**, enviando una petición post al endpoint descrito anteriormente con el payload {userId, courseId} y devuelve su respuesta al método de la store para su posterior evaluación.

Problemas, dificultades, y soluciones adoptadas

Sobre el almacenamiento de archivos

Este fue, sin duda, uno de los mayores retos del proyecto, ya que era el único aspecto que no había trabajado previamente durante los dos últimos años del ciclo. La gestión de archivos PDF —su creación, modificación y almacenamiento— supuso una curva de aprendizaje considerable y me obligó a investigar y comparar distintas alternativas.

Entre las opciones que valoré para el almacenamiento en la nube estuvieron los servicios de Amazon (AWS), OneDrive y Google Drive. También llegué a plantearme soluciones más rudimentarias, como el almacenamiento local o incluso guardar los archivos directamente en la base de datos, lo cual descarté por su ineficiencia y complejidad innecesaria. Finalmente, me decanté por la API de Google Drive por ser relativamente sencilla de configurar, bien documentada y, además, por tratarse de un entorno con el que ya estaba familiarizado.

Sin embargo, en el momento de gestionar imágenes jpg o png, la api de google Drive puso demasiados problemas y, tras horas de intentos, tuve que recurrir a guardar las imágenes en formato de string en base64 en la base de datos (una solución ineficiente pero al menos funcional con imagenes de poco peso).

Sobre la creación de PDFs

Por otro lado, la generación de certificados en PDF abrió un nuevo frente de investigación... y puso de manifiesto mi nula capacidad artística (una dificultad que, probablemente, vuelva a aparecer más adelante). En un principio, consideré usar **Puppeteer**, que permite generar PDFs a partir de páginas HTML, pero la necesidad de plantillas más controladas y simples me llevó a optar por **PDFKit**, una librería que ofrece mayor libertad a la hora de construir documentos desde cero. Esto me permitió diseñar certificados sencillos, sin depender de plantillas visuales complejas.

Sobre el control de fechas y caducidad

Uno de los aspectos clave del proyecto era implementar un sistema que permitiera controlar la validez de los cursos y certificados a lo largo del tiempo. Tras investigar cómo lo gestionan algunas empresas y plataformas similares, decidí implementar un modelo que permitiese establecer tres periodos de validez: 6 meses, 1 año o 2 años.

El control de fechas se gestiona desde el backend. Las fechas se almacenan en la base de datos como cadenas en formato ISODate, y a partir de ahí se calcula la diferencia en días para determinar si un curso o certificado sigue vigente, está próximo a caducar o ya ha expirado.

Más adelante descubrí soluciones más modernas y eficientes, como el uso de librerías especializadas tipo **Day.js**, que facilitan enormemente el manejo de fechas. Sin embargo, por limitaciones de tiempo y dado que la solución nativa cumplía con los requisitos funcionales mínimos, decidí mantener la implementación original basada en el API estándar de JavaScript.

Sobre el entorno de desarrollo

Durante el desarrollo general del proyecto me encontré con varias dificultades derivadas del objetivo autoimpuesto de estructurar y construir la aplicación de la forma más profesional posible. Una de las primeras fue aprender a trabajar con variables de entorno a través de un archivo `.env`, asegurando que no se subiera al repositorio remoto por motivos de seguridad. Esto implicó familiarizarme con configuraciones básicas pero cruciales, como la correcta implementación de `.gitignore`.

Otra fuente frecuente de frustración fue el uso de librerías de terceros. Por ejemplo, más de una vez intenté utilizar componentes o funciones que habían quedado obsoletos o pertenecían a versiones anteriores de la documentación. Estos errores, provocados por desconocimiento o simples despistes, me hicieron perder bastantes horas intentando entender por qué algo no funcionaba como debía. A largo plazo, sin embargo, me ayudaron a ganar experiencia real con la gestión de versiones y la lectura crítica de documentación técnica.

Otro frente complejo fue la organización de la estructura de carpetas y scripts, tanto en el frontend como en el backend. Me esforcé por mantener una arquitectura limpia y

escalable, incluso si eso requería más tiempo en la fase inicial. En este sentido, el frontend presentó un desafío especial que merece una mención aparte.

Sobre el frontend y el uso de Atomic Design

Una de las decisiones más exigentes, pero también más formativas, fue aplicar la metodología de **Atomic Design** en la construcción del frontend. Hasta el inicio de mis prácticas no había escuchado hablar de términos como “átomos” o “moléculas” aplicados al diseño de componentes. A pesar de mi desconocimiento inicial, decidí apostar por esta metodología, sabiendo que iba a duplicar (o más) las horas necesarias para desarrollar una interfaz funcional.

Para equilibrar la carga de trabajo y centrarme en la estructura, opté por un diseño **minimalista y funcional**, dejando de lado elementos visuales complejos o decorativos. El resultado puede parecer algo básico en lo estético, pero está correctamente estructurado. Esto permite que, en el futuro, cualquier diseñador o desarrollador con más talento artístico pueda aplicar un rediseño completo sin tener que rehacer la lógica o la organización interna del proyecto.

Resultados Obtenidos

El producto resultante es una aplicación completamente funcional y con margen de crecimiento. Aunque el diseño del frontend es sencillo, y tanto el backend como la base de datos podrían beneficiarse de optimizaciones (como el uso de librerías más modernas o una reestructuración para aligerar ciertas tablas), la plataforma cumple con todos los objetivos propuestos.

Actualmente, el sistema sería apto para preparar su fase de despliegue en un entorno empresarial real con hasta **300 empleados**, gestionando múltiples cursos, certificados y seguimientos de forma eficiente. Además, gracias a su diseño modular, el uso de **programación asíncrona** y la **externalización del almacenamiento en Google Drive**, la aplicación ofrece una buena base para una futura escalabilidad masiva. Con la integración de herramientas como **balanceadores de carga**, **sistemas de gestión de tareas** o **cachés distribuidos**, sería posible adaptarla a contextos mucho más exigentes sin una reescritura completa del sistema, aunque sería posible optimizarla antes de dar por finalizada la fase de producción. En este sentido, el trabajo futuro a realizar de cara a la fase de despliegue podría incluir :

- Optimización de la Base de datos , indexando búsquedas, delegando el almacenamiento de imágenes de perfil y buscando alternativas más profesionales de almacenamiento en la nube
- Modernización del backend: investigación sobre alternativas de encriptación para mayor seguridad en el login, integración de librerías de gestión del tiempo como dayjs y servicio de avisos al correo electrónico.
- Ampliación de frontend: integración de funcionalidades adicionales: edición de datos de usuario, recuperación de contraseña, gestión de grupos de formación...

Conclusiones

Este proyecto dio comienzo el 10 de marzo de 2025, coincidiendo con mi primer día de prácticas en empresa. Desde entonces, he dedicado una media de dos horas diarias, sin interrupciones, con el objetivo de llevar esta iniciativa a su punto más avanzado posible. El total estimado de tiempo invertido supera ampliamente las 200 horas, en contraste con las 40 horas previstas inicialmente en el módulo de Proyecto. Esta desviación se debe, en gran parte, a la ambición y complejidad inicial de la idea.

La principal lección que extraigo de esta experiencia no está relacionada con una tecnología concreta, sino con la **gestión del tiempo** y la organización del trabajo. En este sentido, podría resumir lo aprendido en tres enseñanzas clave:

1. Preparación

Invertir tiempo en estudiar detenidamente la documentación de una librería puede parecer improductivo al principio, pero he comprobado que este enfoque evita problemas futuros. En más de una ocasión, una buena comprensión previa me ha permitido resolver errores en segundos, cuando, de lo contrario, habrían supuesto horas de bloqueo y frustración.

2. Orden

Trabajar bajo estructuras y metodologías previamente desconocidas ha supuesto un reto considerable. En particular, la jerarquía de diseño atómico en el frontend y la organización por capas en el backend fueron inicialmente fuentes de confusión. No obstante, una vez interiorizados, estos esquemas han facilitado enormemente las tareas de mantenimiento, ampliación y resolución de errores. La experiencia demuestra que una buena estructura inicial repercute directamente en la agilidad del desarrollo posterior.

3. Disciplina

La constancia ha sido la piedra angular de este proyecto. No solo en forma de horas frente a la pantalla, sino también a través de la reflexión fuera del horario de trabajo, las noches de insomnio intentando resolver un problema o las conversaciones diarias con mi compañero de prácticas, que funcionaron como pequeñas reuniones SCRUM improvisadas. Esta dinámica diaria ha sido esencial para mantener la motivación y el enfoque.

En resumen, más allá del resultado funcional alcanzado, este proyecto ha sido una experiencia de aprendizaje integral: técnica, metodológica y personal. Me ha permitido consolidar conocimientos, enfrentarme a nuevos desafíos y, sobre todo, entender mejor cómo abordar un desarrollo completo de principio a fin.

Bibliografía

1. Backend – Node.js y librerías asociadas

The Node.js Foundation. (2024). *Node.js v20.11.1 documentation*.

<https://nodejs.org/dist/latest-v20.x/docs/api/>

Express Team. (2024). *Express.js v4.21 documentation*.

<https://expressjs.com/en/4x/api.html>

PostgreSQL Global Development Group. (2024). *PostgreSQL v16.2 documentation*. <https://www.postgresql.org/docs/16/>

Rowan, D. (2024). *Sequelize v6.37 documentation*.

<https://sequelize.org/docs/v6/>

Devon, M. (2024). *PDFKit v0.17.0 documentation*. <https://pdfkit.org/index.html>

Google LLC. (2024). *Google APIs Node.js Client: googleapis v148.0.0*.

<https://developers.google.com/workspace/docs/api/reference/rest?hl=es-419>

Zabriskie, M. (2024). *jsonwebtoken v9.0.2 documentation*.

<https://jwt.io/#debugger>

Multer Team. (2024). *Multer v1.4.5-lts.2 documentation*.

<https://github.com/expressjs/multer/tree/v1.4.5-lts.2>

CORS Middleware Authors. (2024). *cors v2.8.5 documentation*.

<https://github.com/expressjs/cors#readme>

bcrypt Team. (2024). *bcrypt v5.1.1 documentation*.

<https://www.npmjs.com/package/bcrypt>

2. Frontend – React y librerías relacionadas

Meta. (2024). *React v19 documentation*. <https://react.dev/>

Meta. (2024). *React Router DOM v7.4.1 documentation*.

<https://reactrouter.com/en/main/start/overview>

Material UI Team. (2024). *MUI v7.0.0 documentation*.

<https://mui.com/material-ui/getting-started/>

Poirier, J. (2024). *Zustand v5.0.3 documentation*.

<https://docs.pmnd.rs/zustand/getting-started/introduction>

Axios Maintainers. (2024). *Axios v1.8.4 documentation*.

<https://axios-http.com/docs/intro>

Mdevils. (2024). *jwt-decode v4.0.0 documentation*.

<https://github.com/auth0/jwt-decode>

3. Herramientas de desarrollo

You, E. (2024). *Vite v5.1 documentation*. <https://vitejs.dev/guide/>

Chacon, S., & Straub, B. (2024). *Pro Git* (2nd ed.). Apress.

<https://git-scm.com/book/en/v2>

4. Metodologías y diseño

Frost, B. (2016). *Atomic design*. Brad Frost Web.

<https://bradfrost.com/blog/post/atomic-web-design/>