



# Trabajo de Fin de Ciclo

CFGS Desarrollo de Aplicaciones Web (Dual)

Curso 2024 -2025

**Tutor:** Antonio León García

## **Autores**

Alba Chicharro

Lorena Prieto

Romel Romero

5 de junio de 2025

## ÍNDICE

<b>1. Introducción.....</b>	<b>6</b>
<b>1.1. Herramientas, IDEs y tecnologías utilizadas. ....</b>	<b>6</b>
1.1.1. Postman.....	6
1.1.2. Visual Studio Code (VS Code). ....	6
1.1.3. Git.....	6
1.1.4. GitHub.....	7
1.1.5. PostgreSQL. ....	7
1.1.6. Lucidchart. ....	7
1.1.7. PlantUML. ....	7
1.1.8. Google Chrome.....	7
1.1.9. PowerShell y Git bash. ....	7
1.1.10. Render. ....	7
<b>1.2. Lenguajes de programación, frameworks y tecnologías. ....</b>	<b>8</b>
1.2.1. JavaScript.....	8
1.2.2 TypeScript. ....	8
1.2.3 React 18.....	8
1.2.4. Next.js 13+.....	8
1.2.5. Node.js.....	8
1.2.6. SQL. ....	8
1.2.7. HTML ( <i>HyperText Markup Language</i> ). ....	8
1.2.8. CSS ( <i>Cascading Style Sheets</i> ).....	9
1.2.9. JSON.....	9
1.2.10. JWT.....	9
1.2.11 Plantillas EJS (Embedded JavaScript Templates). ....	9
<b>1.3. UI y Estilos.....</b>	<b>9</b>
1.3.1. Tailwind CSS.....	9
1.3.2. Shadcn/ui.....	9
1.2.3. Lucide React. ....	9
<b>1.4. Librerías utilizadas en el proyecto y librerías <i>NPM</i> de <i>Node.js</i>. ....</b>	<b>10</b>
1.4.1 Express.js.....	10
1.4.2 Sequelize. ....	10
1.4.3. CORS. ....	10

1.4.4. Dotenv.....	10
1.4.5. Nodemailer .....	10
1.4.6 bcrypt.....	10
1.5 Bibliotecas importantes utilizadas de react. ....	11
1.5.1. Next-intl. ....	11
1.5.2. Date-fns. ....	11
1.5.3. Zod. ....	11
2. Justificación del proyecto. ....	11
3. Objetivos.....	12
3.1. Objetivos iniciales.....	12
3.2. Objetivos específicos. ....	12
4. Arquitectura de proyecto y estructura y diseño.....	13
4.1. Organización de archivos y directorios parte cliente. ....	13
4.1.1. Arquitectura y estructura parte cliente ( <i>React</i> ).....	13
4.1.2. Sistema y estructura de rutas parte cliente ( <i>React</i> ). ....	14
4.1.3. Middleware. ....	14
4.1.4. Organización de módulos. ....	15
4.1.5. Servicios y hooks personalizados. ....	15
4.1.6. Estado y gestión de datos. ....	15
4.1.7. Jerarquía de componentes. ....	16
4.1.8. Gestión del estado.....	17
4.1.9. Protección de rutas .....	17
4.1.10. Internacionalización. ....	17
4.1.11. Flujo de autenticación.....	18
4.1.12. Integración con API. ....	18
4.2. Organización de archivos y directorios parte servidora. ....	19
4.2.1. Estructura general del proyecto parte servidora.....	19
4.2.2. Otros ficheros/directorios relevantes. ....	20
4.2.3. Configuración con la Base de Datos. Mapeo de tablas relacionales ( <i>Secualize</i> ).....	20
4.2.4. Uso de <i>NodeMailer</i> para el envío de correos electrónicos. ....	22
4.2.5 Esquema del diseño general de la parte MVC.....	24
4.2.6. Token y autenticación con JWT. ....	24
4.2.7.Encriptación de contraseñas en Node.js con Bcrypt. ....	26

4.2.8. Generación de <i>PDF</i> con <i>EJS</i> y <i>html-pdf</i> .	27
5. Desarrollo del proyecto.	28
5.1. Peticiones cliente-servidor.	28
5.2. Funcionalidades principales e implementación de APIs.	28
5.2.1. Funcionalidad que aplica a un usuario con rol <i>User</i> .	28
5.2.2. Funcionalidad que aplica a un usuario con rol <i>Admin</i> .	31
5.3. Funcionalidades principales a nivel cliente.	32
5.3.1. Autenticación.	32
5.3.2. Reserva del vuelo.	32
5.3.3. Gestión de reservas.	34
5.3.5. Panel de administración.	35
6. Despliegue de la aplicación.	35
6.1. Despliegue de la parte servidor (Node y PostgreSQL).	36
6.1.1. Plataforma de despliegue.	36
6.1.2. Proceso de Despliegue.	36
6.1.3. Uso de Variables de Entorno en local y en Render (.env).	37
6.1.4. Logs y monitoreo.	37
6.1.5. Proceso de Build y despliegue en Render.	38
6.2 Despliegue de la parte cliente.	38
6.2.1. Configuración de despliegue.	39
6.2.2. Extensibilidad y mantenimiento y actualización de dependencias.	39
6.2.3. Requisitos previos y configuración del entorno.	40
7. Accesibilidad.	40
8. Responsividad.	41
9. Componentes UI/UX.	41
9.1. Componentes base.	41
9.2. Componentes compuestos.	41
9.3. Feedback al usuario.	41
10. Manejo de errores.	42
11. Optimización de rendimiento.	42
12. Desarrollo de la BBDD.	42
12.1. Diagrama Entidad-Relación .	43
12.1.1. Cardinalidad y relaciones de las entidades.	43

12.1.2. Atributos, tipos de datos y restricciones de las tablas. ....	44
13. Planificación y control de cambios. ....	46
14. Resultados y discusión. ....	47
16. Bibliografía y referencias. ....	49

## **1. Introducción.**

El presente documento es la memoria del Proyecto de Fin de Ciclo Formativo de DAW Dual (Desarrollo de Aplicaciones Web). AirLink es una aplicación web diseñada para la gestión de reservas de vuelos, alojada en un servidor y construida utilizando tecnologías modernas como Next.js 13+, Node y React.

La aplicación permite a los usuarios buscar y reservar vuelos, gestionar sus billetes y cuentas de usuarios, y ofrece a los administradores herramientas para la gestión de vuelos. Podemos decir también que proporciona soporte multilingüe en español e inglés, garantizando una experiencia de usuario accesible.

### **1.1. Herramientas, IDEs y tecnologías utilizadas.**

#### **1.1.1. Postman.**

Postman es una herramienta utilizada para probar y desarrollar *APIs* (*Interfaz de Programación de Aplicaciones*). Permite a los desarrolladores enviar solicitudes HTTP (como *GET*, *POST*, *PUT*, *DELETE*) a servidores web, analizar las respuestas y automatizar pruebas. Es muy popular en el desarrollo de aplicaciones web y móviles, porque facilita: enviar solicitudes *HTTP*, probar y depurar *APIs*, automatizar pruebas, documentación de *APIs* y simulación de respuestas.

#### **1.1.2. Visual Studio Code (VS Code).**

*Visual Studio Code* (*VS Code*) es un editor de código fuente desarrollado por Microsoft, diseñado para ser ligero, rápido y altamente personalizable. Se incluyen la autocompletado inteligente mediante IntelliSense, la depuración integrada, y el control de versiones con Git. Es gratuito y multiplataforma.

#### **1.1.3. Git.**

*Git* es un sistema de control de versiones distribuido y colaborativo. Permite así trabajar de forma individual, mediante ramas trabajos, que luego serán subidas a una rama conjunta en el servidor.

#### **1.1.4. GitHub.**

*GitHub* es una plataforma en la nube que permite a los desarrolladores almacenar, compartir y colaborar en proyectos de software utilizando el sistema de control de versiones Git.

#### **1.1.5. PostgreSQL.**

Como herramienta elegimos *PostgreSQL*, un sistema de gestión de bases de datos relacional orientado a objetos.

#### **1.1.6. Lucidchart.**

Lucidchart es una herramienta de diagramación que permite crear diagramas de flujo, organigramas, esquemas de sitios web, diseños UML, mapas mentales, prototipos de software y muchos otros tipos de diagrama.

#### **1.1.7. PlantUML.**

PlantUML es una herramienta de software para crear diagramas a partir de texto simple. Soporta la creación de diagramas UML y otros formatos como Mapa mental, ArchiMate, Diagrama de bloques, BPMN, Diagrama de Gantt y otros.

#### **1.1.8. Google Chrome.**

Es un navegador web gratuito desarrollado por Google. Lanzado oficialmente en 2008, está disponible para múltiples plataformas, incluyendo Windows, macOS, Linux, Android e iOS.

#### **1.1.9. PowerShell y Git bash.**

Son herramientas de automatización que funcionan un intérprete de comandos con un lenguaje de scripting permitiendo a los administradores de sistemas y usuarios avanzados automatizar tareas y administrar sistemas de manera eficiente.

#### **1.1.10. Render.**

Es una plataforma de computación en la nube que facilita el despliegue, el hosting y la ejecución de aplicaciones, bases de datos, tareas programadas y otros servicios.

## 1.2. Lenguajes de programación, frameworks y tecnologías.

### 1.2.1. JavaScript.

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

### 1.2.2. TypeScript.

Es un lenguaje de programación que extiende JavaScript que se basa en un sistema de tipos estáticos, interfaces y tipos para definir contratos genéricos para código reutilizable y detección de errores en tiempo de compilación.

### 1.2.3. React 18.

Es una biblioteca UI, basada en *concurrent mode* para renderizado más eficiente, *hooks* para gestión de estado y efectos, *suspense* para carga de datos y *strict mode* para detectar problemas.

### 1.2.4. Next.js 13+.

Es un framework *React* con *App Router*. Basado en el renderizado del lado del servidor (SSR), generación estática (SSG), *API Routes* para endpoints de backend *Middleware* para control de rutas y *Server Components* y *Server Actions*.

### 1.2.5. Node.js.

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

### 1.2.6. SQL.

SQL es un *Lenguaje de Consulta Estructurado*, utilizado para gestionar y manipular bases de datos relacionales. Permite realizar operaciones como almacenar, actualizar, eliminar, buscar y recuperar información en bases de datos.

### 1.2.7. HTML (*HyperText Markup Language*).

Es un lenguaje de marcas que define el significado y la estructura del contenido web.



### **1.2.8. CSS (*Cascading Style Sheets*).**

CSS (*Cascading Style Sheets*) u “*Hojas de Estilo en Cascada*”, es un lenguaje de diseño gráfico para la presentación visual de un documento web e interfaces como HTML o XHTML.

### **1.2.9. JSON.**

Es el acrónimo de *JavaScript Object Notation*. Con un formato de texto se ha usado en el proyecto para el intercambio de datos.

### **1.2.10. JWT.**

Es un estándar abierto (RFC 7519) que define un método compacto y autocontenido para transmitir información de forma segura entre dos partes como un objeto JSON. Estos tokens son firmados digitalmente, lo que garantiza la integridad y confiabilidad de los datos que contienen.

### **1.2.11 Plantillas EJS (*Embedded JavaScript Templates*).**

Es un motor de plantillas que permite generar *HTML* dinámico usando JavaScript directamente dentro de las vistas. Lo utilizamos para renderizar los correos y archivos en formato *pdf*.

## **1.3. UI y Estilos.**

### **1.3.1. Tailwind CSS.**

Es un framework de utilidades CSS predefinidas. Basado en un diseño responsive con breakpoints, personalización mediante configuración y optimización automática en producción.

### **1.3.2. Shadcn/ui.**

Es una colección de componentes UI basados en Radix UI. Basado en componentes accesibles e integración con Tailwind CSS. Componentes como Button, Card, Dialog, Form, etc con temas personalizables.

### **1.2.3. Lucide React.**

Biblioteca de iconos SVG. Personalización de tamaño y color y optimización para React utilizando carga bajo demanda.

## **1.4. Librerías utilizadas en el proyecto y librerías *NPM* de *Node.js*.**

### **1.4.1 Express.js.**

Framework web minimalista y flexible que facilita la creación de aplicaciones web y APIs de manera rápida y sencilla.

**Instalación:** *npm install express.*

### **1.4.2 Sequelize.**

ORM (*Object-Relational Mapping*) que soporta múltiples bases de datos SQL, permitiendo trabajar con ellas de manera sencilla.

**Instalación:** *npm install sequelize.*

### **1.4.3. CORS.**

Middleware que permite configurar y habilitar el intercambio de recursos de origen cruzado (*CORS*), facilitando las solicitudes entre diferentes dominios.

**Instalación:** *npm install cors.*

### **1.4.4. Dotenv.**

Carga variables de entorno desde un archivo *.env* en *process.env*, permitiendo gestionar configuraciones sensibles de manera segura.

**Instalación:** *npm install dotenv.*

### **1.4.5. Nodemailer**

Permite enviar correos electrónicos desde aplicaciones Node.js, soportando diversos transportes como SMTP, SES y más.

**Instalación:** *npm install nodemailer.*

### **1.4.6. Bcrypt.**

Permite generar hashes seguros de contraseñas, incorporando un proceso de "salting" que añade aleatoriedad y refuerza la seguridad.

**Instalación:** *npm install bcrypt.*

## **1.5 Bibliotecas importantes utilizadas de react.**

### **1.5.1. Next-intl.**

Biblioteca para soporte multilingüe con traducciones basadas en archivos JSON, que proporciona componentes y hooks para acceder a las traducciones, integración con App Router, y formato de fechas y números adaptado según el *locale*.

### **1.5.2. Date-fns.**

Es una biblioteca para manipulación de fechas que ofrece funciones para formato, *parsing* y modificación, soporte para zonas horarias, comparación de fechas e internacionalización.

### **1.5.3. Zod.**

Biblioteca para validación de esquemas que permite la definición de reglas, inferencia de tipos en TypeScript, personalización de mensajes de error y transformación de datos.

## **2. Justificación del proyecto.**

El desarrollo de AirLink responde a la necesidad de diseñar y construir una solución tecnológica que simule un sistema de gestión de reservas de vuelos.

Este proyecto se justifica por su capacidad para abordar problemas complejos y reales, al tiempo que consolida las competencias adquiridas durante el *Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web (DAW)*.

La gestión de reservas de vuelos es un pilar fundamental en el sector del comercio electrónico, presente en plataformas como *Booking*, *Edreams* o *Skyscanner*. AirLink permite aplicar conocimientos técnicos en un contexto práctico, replicando escenarios reales que requieren integración de frontend, backend, bases de datos, lo que lo convierte en un gran reto.

El proyecto ofrece una oportunidad única para dominar herramientas y frameworks de alta demanda en el mercado laboral, como *Next.js 13+*, *Node.js*, *React*, y *PostgreSQL*.

*AirLink* abarca todas las capas de una aplicación web moderna, desde el diseño de interfaces responsive y accesibles hasta la implementación de una API RESTful segura y una base de datos relacional. Este enfoque integral permite demostrar competencias full-stack, integrando conocimientos de los módulos de DAW como *Desarrollo Web en Entorno Cliente*, *Desarrollo Web en Entorno Servidor* y *Despliegue de Aplicaciones Web*.

En resumen, AirLink es un proyecto nos prepara para los retos del mercado laboral, demostrando la capacidad de diseñar, implementar y desplegar una aplicación web de forma completa.

### **3. Objetivos.**

#### **3.1. Objetivos iniciales.**

Los objetivos iniciales que se buscaban alcanzar son los siguientes:

Profundizar en el conocimiento y dominio de tecnologías *Open Source*, como *Node.js*, *React* y *Next.js*, para el desarrollo de aplicaciones web.

Familiarizarse con herramientas y entornos de desarrollo, como *Visual Studio Code*, *Git*, *GitHub*, *Postman* y *Render*, para optimizar el flujo de trabajo y la gestión de proyectos.

Implementar una arquitectura modular basada en el patrón *MVC (Modelo-Vista-Controlador)* en el servidor y patrones de diseño como módulos y servicios en la parte del cliente, asegurando un código organizado y mantenible.

Crear una aplicación web atractiva, eficiente y responsive, que ofrezca una experiencia de usuario intuitiva, con funcionalidades avanzadas como búsqueda de vuelos, selección de asientos y gestión de reservas, envío de correos electrónicos y creación de archivos en formato pdf, simulando escenarios similares de la vida real.

#### **3.2. Objetivos específicos.**

Profundizar en el uso de *Node.js*, *React*, *Next.js*, y *PostgreSQL* para el desarrollo de aplicaciones web modernas.

Diseñar la aplicación utilizando el patrón MVC en el servidor y patrones modulares en el cliente, asegurando un código organizado y mantenible.

Crear una interfaz atractiva, responsive y accesible, con soporte multilingüe y funcionalidades avanzadas como búsqueda de vuelos, selección de asientos, y gestión de reservas.

Implementar autenticación basada en JWT, encriptación de contraseñas con bcrypt, y comunicaciones seguras mediante HTTPS y SSL/TLS.

Aplicar técnicas como carga diferida, memorización de componentes, y optimización de imágenes para mejorar la velocidad y eficiencia de la aplicación.

Publicar el backend en Render y el frontend en Vercel, configurando entornos de producción.

Implementar el envío de correos electrónicos con *Nodemailer* y la generación de PDFs con *EJS* y *html-pdf*, simulando flujos reales de una aplicación de reservas.

Documentar el proceso detallado que describa la arquitectura, tecnologías, y decisiones de diseño tomadas durante el desarrollo.

#### **4. Arquitectura de proyecto y estructura y diseño.**

El proyecto usa el *patrón MVC en la parte servidora*, una arquitectura de software que divide una aplicación en varios componentes: *Modelo*, *Vista* y *Controlador*. En cuanto a *la parte cliente*, la aplicación sigue una arquitectura modular basada en dominios, donde cada módulo funcional está encapsulado en su propia carpeta con sus componentes, servicios y tipos. Esta forma de agrupar el proyecto facilita el mantenimiento y la escalabilidad de nuestra aplicación.

##### **4.1. Organización de archivos y directorios parte cliente.**

###### **4.1.1 Arquitectura y estructura parte cliente (React).**

La parte de *React* de nuestra aplicación, presenta una serie de patrones de arquitectura y organización de código, entre ellos están:

- **Patrón módulo:** Encapsulamiento de funcionalidades relacionadas.

- **Patrón contenedor/presentación:** Separación de lógica y UI.
- **Patrón hook:** Extracción de lógica reutilizable.
- **Patrón servicio:** Encapsulamiento de lógica de comunicación con API.

#### 4.1.2. Sistema y estructura de rutas parte cliente (React).

Basado en el sistema de carpetas de Next.js App Router. Cada carpeta representa una ruta.

- *page.tsx*: el componente de la página.
- *layout.tsx*: el layout compartido.
- *[locale]*: soporte multilingüe. El middleware detecta y maneja el idioma.

Encontramos el siguiente grupos de rutas principales:

- ***/(auth)***: login, registro, recuperación de contraseña.
- ***/(booking)***: flujo de reserva
  - /[locale]/(booking)/search***: búsqueda y selección de vuelos.
  - /[locale]/(booking)/seats***: selección de asientos.
  - /[locale]/(booking)/passengers***: datos de pasajeros.
  - /[locale]/(booking)/confirmation***: confirmación de reserva.
- ***/(dashboard)***: página principal.
  - /[locale]/(dashboard)/dashboard***: búsqueda de vuelos
- ***/(profile)***: gestión de perfil.
  - /[locale]/(profile)/profile***: datos de usuario y reservas
- ***/admin***: panel de administración.
  - /[locale]/(admin)/admin/dashboard***: es el panel principal.
  - /[locale]/(admin)/admin/vuelos***: gestión de vuelos.

#### 4.1.3. Middleware.

Redirección basada en autenticación.

- Verificación de token JWT.
- Redirección a login para rutas protegidas.
- Verificación de permisos y comprobación de rol de administrador.
- Redirección a *dashboard* para usuarios sin permisos.
- Manejo de idiomas detección de idioma en la URL. Redirección a idioma por defecto si no se especifica.

#### 4.1.4. Organización de módulos.

Podemos hacer mención a que la aplicación en la parte cliente está organizada en los siguientes módulos:

- *components/*: Son los componentes UI específicos del módulo, de presentación, contenedores con lógica y formularios específicos.
- *services/*: contiene las funciones para interactuar con la API, métodos para obtener datos, métodos para enviar y transformar los datos.
- *hooks/*: contiene hooks personalizados para lógica de negocio, el encapsulamiento de lógica compleja, el manejo de estado específico y efectos secundarios.
- *types/*: contiene las definiciones de tipos TypeScript, las interfaces para datos, tipos para props y las enumeraciones.

#### 4.1.5. Servicios y hooks personalizados.

Hacemos mención de los Hooks principales de la aplicación.

- ***useSearchFlights***: gestiona la búsqueda de vuelos y almacenamiento de resultados. Así mismo, maneja la obtención y actualización de datos del perfil.
- ***useProfile***: maneja la obtención y actualización de datos del perfil.
- ***useSessionStatus***: verifica el estado de la sesión y su validez.

Además, tenemos que destacar los servicios importantes.

- ***auth.ts***: configuración de NextAuth.js y manejo de credenciales.
- ***api.ts***: funciones base para interactuar con la API.

Por consiguiente, cada módulo tiene servicios especializados para sus necesidades:

- ***auth/services/login.ts***: autenticación de usuarios.
- ***flights/services/flights.ts***: búsqueda y gestión de vuelos.
- ***seats/services/seats.ts***: obtención y reserva de asientos.
- ***profile/services/profile.ts***: gestión de perfil de usuario.
- ***bookings/services/booking.ts***: creación y gestión de reservas.

#### 4.1.6. Estado y gestión de datos.

- **React Context API.**

Utilizado para manejar estado global. Se definen proveedores de contexto,

por ejemplo, para autenticación, permitiendo el acceso a datos compartidos entre componentes.

- **React Hooks.**

Empleados para manejar estado local y efectos secundarios. Se emplea el *useState* (estado local), *useEffect* (efectos secundarios), *useCallback* y *useMemo* (optimización). Se definen también Hooks personalizados para lógica reutilizable.

- **SWR/React Query.**

Herramientas para *fetching* y caché de datos. Ofrecen caché automática, revalidación automática, manejo de estados de carga y error y deduplicación de peticiones.

- **Autenticación.**

Usamos *NextAuth.js*, un framework de autenticación para Next.js. Soporta múltiples proveedores, gestión de sesiones, protección de rutas y callbacks personalizables. Además, hacemos uso de *JWT (JSON Web Tokens)*, una autenticación basada en backend mediante tokens firmados y permiten la verificación de identidad, almacenamiento seguro e inclusión de información de usuario en el payload.

#### 4.1.7. Jerarquía de componentes.

- **Páginas.**

Son componentes de nivel superior que representan rutas, definidas mediante la estructura de carpetas del App Router. Pueden ser *Server Components* o *Client Components* e importan componentes contenedores.

- **Contenedores.**

Son componentes que manejan la lógica y el estado, gestionan efectos, realizan llamadas a la API y pasan datos a componentes de presentación.

- **Los componentes de presentación.**

Representan la UI pura, reciben *props*, renderizan datos, manejan eventos de usuario y definen estilos y layout.

- **Componentes de UI.**

Incluyen elementos reutilizables como botones, inputs o cards, provienen de *shadcn/ui*, se personalizan con Tailwind CSS y son accesibles y responsivos.



#### 4.1.8. Gestión del estado.

- **Estado local.**

Uso de *useState* y *useReducer*. Maneja el estado de componentes, formularios y validación

- **Estado global.**

Contextos React para estado compartido entre componentes. Sesión de usuario gestionada por NextAuth.js.

- **Estado de formularios.**

Gestión manual de formularios con validación personalizada y validación con esquemas Zod.

- **Estado de navegación.**

Almacenamiento en *sessionStorage* para persistir datos entre páginas y uso de parámetros de URL para estado compartible.

#### 4.1.9. Protección de rutas

El Middleware de Next.js verifica la autenticación, realiza la redirección a login para rutas protegidas y lleva a cabo la verificación de roles para acceso a áreas específicas.

#### 4.1.10. Internacionalización.

Soporte para español (es) e inglés (en). Se ha realizado una definición de locales disponibles y configuración de *locale* por defecto.

Archivos de traducción separados por idioma: estructura JSON con claves anidadas y organización por módulos funcionales.

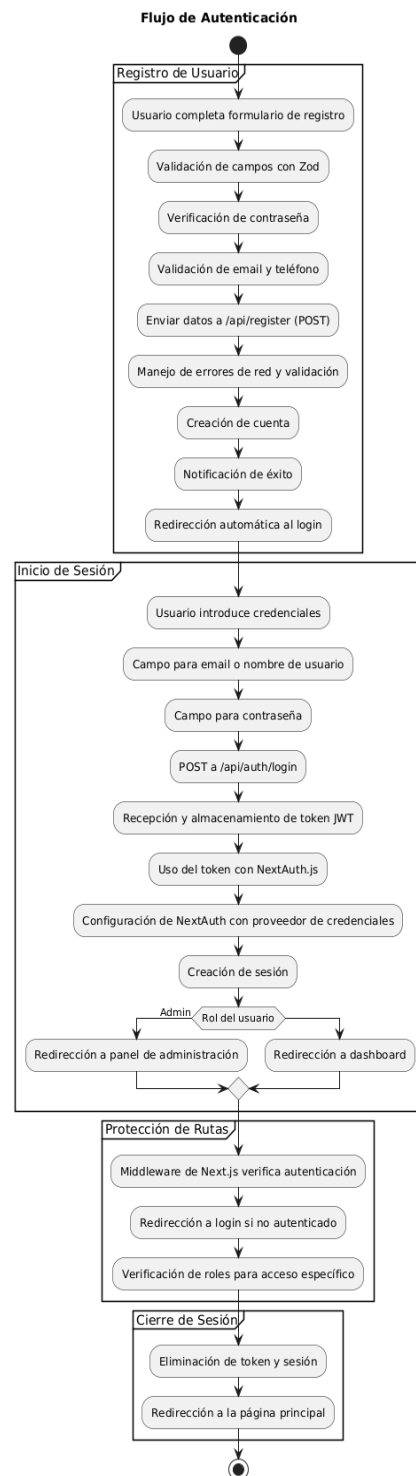
- **Implementación.**

Uso de *useTranslations* de next-intl para textos. Se ha llevado a cabo un cambio de idioma mediante *LanguageSwitcher* y una preservación de idioma en rutas. La generación de enlaces se realiza con *locale* y la aplicación redirecciona manteniendo el idioma.

- **Formato de fechas y números.**

Adaptación según el idioma seleccionado usando *date-fns*.

#### 4.1.11. Flujo de autenticación en la Figura 1.



*Figura 1 Flujo de autenticación cliente*

#### 4.1.12. Integración con API.

Cada módulo tiene sus propios servicios para interactuar con la API.

- **Manejo de autenticación en API.**

Se incluye el token JWT en la cabecera, se renueva automáticamente el token expirado, se verifica la respuesta 401, se solicita un nuevo token, se reintenta la petición original, se maneja el error de autenticación, se redirige al login y se limpia la sesión expirada.

- **Gestión de errores.**

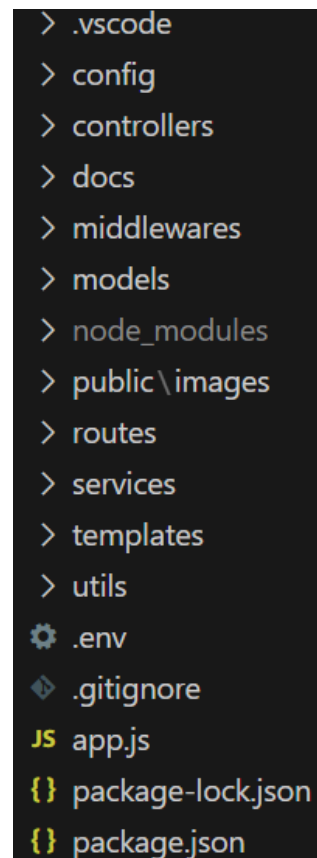
Captura y procesamiento de errores de API y notificaciones al usuario mediante toasts. Se emplean Fallbacks y estados de error en UI.

## 4.2. Organización de archivos y directorios parte servidora.

Estructuramos nuestro proyecto de la siguiente manera:

### 4.2.1. Estructura general del proyecto parte servidora Figura 2.

- **/config:** configuraciones de la base de datos, o herramientas de terceros.
- **/controllers:** gestionan las solicitudes entrantes HTTP, contiene la lógica de la petición en una ruta determinada.
- **/docs:** documentación importante sobre la configuración del proyecto.
- **/middlewares:** sirve para guardar funciones intermedias que se ejecutan antes de que una petición *HTTP* llegue al controlador final, o antes de que se mande la respuesta.
- **/models:** modelos o entidades que actuarán en la base de datos.
- **/public:** Archivos estáticos accesibles públicamente, como imágenes, CSS, JavaScript del lado del cliente.
- **/routes:** Rutas de los controladores (aquí se definen los endpoints<sup>1</sup> de la nuestra aplicación).
- **/services:** contiene la lógica de negocio.



```
> .vscode
> config
> controllers
> docs
> middlewares
> models
> node_modules
> public\images
> routes
> services
> templates
> utils
⚙ .env
💎 .gitignore
JS app.js
{} package-lock.json
{} package.json
```

Figura 2 Esquema MVC Nodejs

<sup>1</sup> URL específica que permite la interacción entre un *cliente* y un *servidor*. Cada endpoint representa una función o recurso particular de la API, y las solicitudes HTTP enviadas a estos endpoints permiten acceder o manipular datos en el servidor.

- **/templates:** Plantillas para renderizar las vistas de los correos y archivos pdfs (motor de plantillas usado: EJS).
- **/utils:** Funciones de ayuda que se usan para reutilizar código.
- **app.js:** Punto de entrada del servidor.

#### 4.2.2. Otros ficheros/directorios relevantes.

- **package-lock.json:** Este archivo es generado por npm (*Node Package Manager*) y contiene información sobre las versiones exactas de las dependencias utilizadas en el proyecto. Se utiliza para garantizar compilaciones consistentes en diferentes entornos. Figura 3

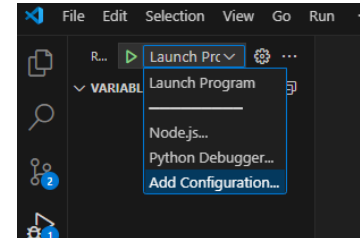


Figura 3 Ventana de depuración de VS

- **package.json:** Este archivo es un archivo importante que contiene metadatos sobre el proyecto, como el nombre, la versión, las dependencias y los scripts.
- **.vscode:** Este directorio contiene configuraciones específicas para el editor Visual Studio Code, como configuraciones de depuración y preferencias del espacio de trabajo. Figura 4

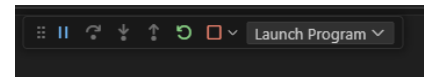


Figura 4 Panel para controlar la depuración

#### 4.2.3. Configuración con la Base de Datos. Mapeo de tablas relacionales (*Secualize*).

En el módulo config concretamente en el fichero postgresSQL.js se configura Sequelize, un *ORM* (*Object Relational Mapping*) para *Node.js*, para conectarse a una base de datos PostgreSQL usando variables de entorno. Usamos Secualize, una librería para mapear Base de Datos. Figura 5

```
const sequelize = new Sequelize({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT || 5432,
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  dialect: 'postgres',
  dialectOptions: {
    ssl: {
      require: true,
      rejectUnauthorized: false,
    },
  },
  define: {
    timestamps: false,
  }
});
```

Figura 5 Configuración de la Base de Datos (*Secualize*)

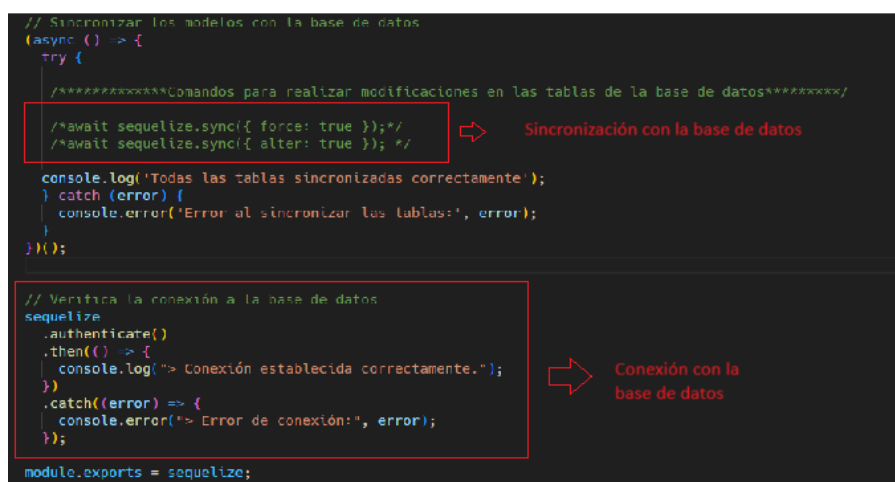
- **Parámetros clave de configuración:**
  - *host, port, username, password, database:* Se toman de variables de entorno (*process.env*) para mayor seguridad y flexibilidad.
  - *dialect: 'postgres':* Indica el uso de PostgreSQL.

- `dialectOptions.ssl.require: true`: Obliga a usar SSL<sup>2</sup>.
- `dialectOptions.ssl.rejectUnauthorized: false`: No verifica el certificado SSL, lo cual es inseguro en producción.
- `define.timestamps: false`: Desactiva campos automáticos `createdAt` y `updatedAt`.
- **Importancia de la Seguridad (SSL):**
  - `require: true`: Protege los datos cifrándolos.
  - `rejectUnauthorized: false`: Acepta cualquier certificado, incluso uno falso o no verificado.

### Sincronización con la Base de Datos Figura 6.

Usamos un mapeo *Objeto-Relacional*, que quiere decir esto, que modificamos las propiedades de la base de datos directamente trabajando con la *entidades/clases* creadas en Node.js. Para ello, cualquier modificación que se realice se sincronizará con la base de datos. Usamos los siguientes comandos:

- `sequelize.sync({ force: true })`: Forzamos y eliminamos todas las tablas que están en la base de datos y volver a crearlas todas acorde al modelo de datos que se haya definido en Node.js.
- `sequelize.sync({ alter: true })`: Cualquier modificación que se haga en el modelo de datos del objeto se actualiza en la Base de Datos, sin eliminar desde cero todas las tablas.



```
// Sincronizar los modelos con la base de datos
(async () => {
  try {
    //*****Comandos para realizar modificaciones en las tablas de la base de datos*****//
    /*await sequelize.sync({ force: true });*/
    /*await sequelize.sync({ alter: true });*/

    console.log('Todas las tablas sincronizadas correctamente!');
  } catch (error) {
    console.error('Error al sincronizar las tablas:', error);
  }
})();

// Verifica la conexión a la base de datos
sequelize
  .authenticate()
  .then(() => {
    console.log('> Conexión establecida correctamente.');
```

The image shows a code editor with two sections of JavaScript code. The first section, titled 'Sincronizar los modelos con la base de datos', contains an asynchronous function that attempts to sync the database. It includes comments for two methods: `sequelize.sync({ force: true })` and `sequelize.sync({ alter: true })`. A red box highlights these two lines, with an arrow pointing to the text 'Sincronización con la base de datos'. The second section, titled 'Verifica la conexión a la base de datos', shows the `sequelize.authenticate()` method. A red box highlights this section, with an arrow pointing to the text 'Conexión con la base de datos'. The code ends with `module.exports = sequelize;`.

Figura 6 Configuración para la sincronización de la base de datos

<sup>2</sup> Los *certificados SSL (Secure Sockets Layer)*, se utilizan para establecer una conexión cifrada entre un navegador o el ordenador de un usuario y un servidor o un sitio web.

Primero actualizamos el mapeo desde Node.js a PostgreSQL y luego realizamos la conexión con la base de datos.

#### 4.2.4. Uso de **NodeMailer** para el envío de correos electrónicos.

*Nodemailer* es un módulo de Node.js que permite enviar correos electrónicos fácilmente.

En cuanto a sus características podemos decir que no posee dependencias externas (más seguro y fácil de auditar), es compatible con emoji y caracteres Unicode.

Se puede realizar el envío de correos en HTML con adjuntos e imágenes embebidas. Soporte para *TLS/STARTTLS*<sup>3</sup> y autenticación OAuth2.

#### Configuración.

1) **Instalar la dependencia de NodeMailer:**

*npm install nodemailer.* Figura 7

2) **Crear un *transporter*** (objeto que configura la conexión con el servidor de correo). Usamos *SMTP*<sup>4</sup>.

3) **Creamos un servicio específico para el envío de correos que contiene la lógica correspondiente al envío de correos.**

4) **Definir el mensaje:** creamos un servicio que obtiene los datos y enviamos los correos a los pasajeros y/o clientes que están implicados en la reserva.

5) **Definimos el html mediante *EJS***, un motor de plantillas que permite generar HTML con JavaScript incrustado. Su propósito principal es **renderizar vistas dinámicas** en el servidor antes de enviarlas al cliente.

6) Una vez renderizada la vista enviamos los correos correspondientes

```
const nodemailer = require('nodemailer');
require('dotenv').config();

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});

module.exports = transporter;
```

Figura 7 Creación del objeto *transporter* para el envío de correos

---

<sup>3</sup> **TLS (Transport Layer Security)** es un protocolo de seguridad que cifra la comunicación entre dos sistemas, como tu ordenador y un servidor de correo, para que nadie pueda leer lo que se transmite.

**STARTTLS** es un comando que actualiza una conexión no segura a una segura utilizando TLS, sin necesidad de cambiar de puerto.

<sup>4</sup> **SMTP** (Simple Mail Transfer Protocol) es el protocolo estándar utilizado para enviar correos electrónicos a través de Internet.

## Interfaz de los envíos de correo.

### Correo con la información de la reserva Figura 8.

**AirLink**

Confirmación de vuelo

**Detalles del vuelo - Romel Romero**

Número de vuelo	BA2025
Aerolínea operadora	British Airways
Fecha de salida	Tue Jun 10 2025 16:00:00 GMT+0200 (hora de verano de Europa central)
Fecha de llegada	Tue Jun 10 2025 20:30:00 GMT+0200 (hora de verano de Europa central)
Aeropuerto de origen	Aeropuerto de Londres-Heathrow
Aeropuerto de llegada	Aeropuerto Internacional de Nueva York-JFK
Asiento	1D (Fila: 1, Columna: D )
Precio	0 EUR
Localizador	OGUFBN

Gracias por volar con nosotros. ¡Buen viaje!

*Figura 8 Interfaz de correo de reserva de billetes*

### Correo de verificación de código para logueo sin la contraseña Figura 9.

Hola **romel1223**,

Hemos recibido una solicitud para verificar tu identidad y poder cambiar tu contraseña. Tu código de verificación es:

**612838**

Este código expirará en 5 minutos. Si no has solicitado esta verificación, puedes ignorar este mensaje.

Atentamente,  
**AirLink**

*Figura 9 Interfaz de correo de envío de código de verificación*

#### 4.2.5. Esquema del diseño general de la parte MVC Figura 10.

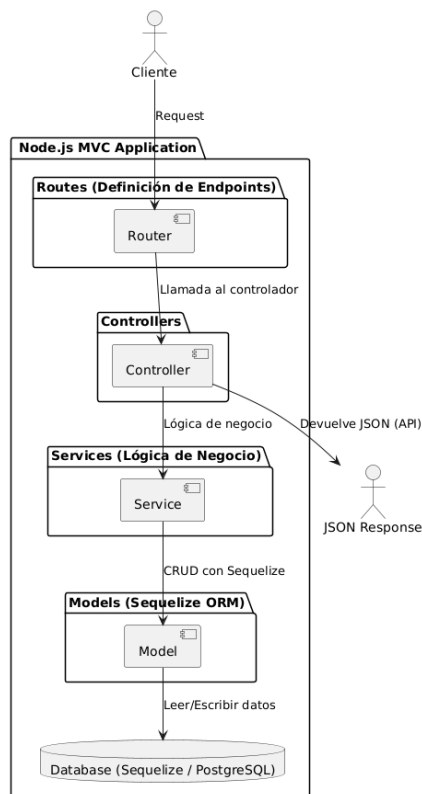


Figura 10 Esquema de la arquitectura de Nodejs

#### 4.2.6. Token y autenticación con JWT.

La autenticación basada en *JSON Web Tokens (JWT)* es un esquema de seguridad que utiliza tokens para validar la identidad de un usuario. Un JWT es un token firmado digitalmente que contiene información sobre el usuario y puede ser usado para autenticar solicitudes posteriores sin necesidad de enviar credenciales repetidas.

El token encriptado contiene esta información (*id* del cliente, el *email*, el *nombre de usuario* y si es *administrador*) almacenada en un objeto (*payload*) que posteriormente será encriptado.

```
const payload = {
  sub: cliente.id_cliente,
  email: cliente.email,
  name: cliente.nombre_usuario,
  is_admin: !!cliente.es_admin,
};
```

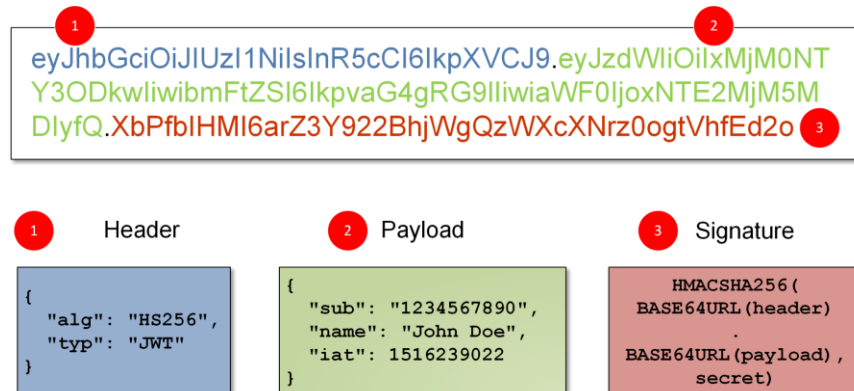
Figura 11 Objeto contenido en el payload del JWT

#### Estructura del token.

- **Header:** indica el algoritmo y el tipo de token.



- **Payload:** contiene los datos. Los que se observan en el objeto de la imagen. Figura 11
- **Signature:** es la firma generada con una clave secreta para validar que el token no fue modificado.



### Funciones importantes (*Middlewares*<sup>5</sup>) para manejar las autenticaciones.

- **verificarToken:** Verifica que el token JWT esté presente y sea válido; si es válido, lo decodifica y lo asocia al objeto `req.user`, permitiendo el acceso a la siguiente función Figura 12.

```

/*****RUTA RESERVA CONTROLLER*****/
router.put('/reservas/realizar-reserva', verificarToken, reservaController.realizarReserva);
router.put('/reservas/realizar-reserva-aleatoria', verificarToken, reservaController.realizarReservaConAsignacionAleatoria);
router.get('/reservas/mis-reservas', verificarToken, reservaController.obtenerReservaCliente);
router.delete('/reservas/:id_reserva', verificarToken, reservaController.eliminarReservaCliente);
router.delete('/reservas/:id_reserva/billetes/:id_billete', verificarToken, reservaController.eliminarBilleteDeReserva);

```

Figura 12 Uso en el módulo de routes.js del método `verificarToken`

- **verificarAdmin:** Verifica si el usuario tiene permisos de administrador, y si es así, permite continuar, de lo contrario, deniega el acceso con un *error 403* Figura 13.

```

router.post('/vuelos/crear', verificarToken, verificarAdmin, vueloController.createVuelo);
router.put('/vuelos/modificar-estado-vuelo', verificarAdmin, vueloController.modificarEstadoVuelo);

```

Figura 13 Uso del método que verifica si el usuario es admin (`verificarAdmin`)

<sup>5</sup> Los *middlewares* actúan como **puentes o filtros** entre el cliente (usuario que hace la solicitud) y el servidor (donde se maneja la lógica de negocio).

## DIAGRAMA CON LA LÓGICA DEL LOGIN Y LA AUTENTICACIÓN Figura 14.

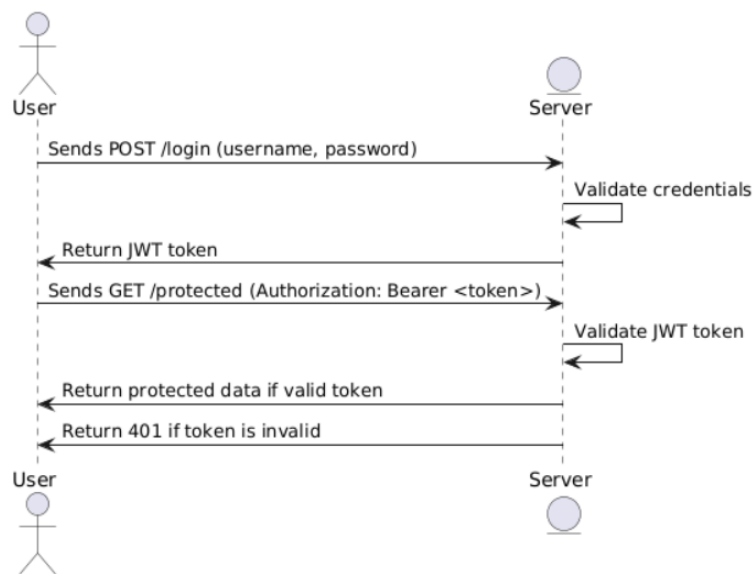


Figura 14 Flujo cliente-servidor del JWT

### 4.2.7. Encriptación de contraseñas en Node.js con Bcrypt.

En el proyecto desarrollado, se ha empleado la librería bcrypt para llevar a cabo esta tarea, que implementa un algoritmo de hash robusto basado en Blowfish<sup>6</sup>.

Durante el proceso de creación de un nuevo cliente Figura 15, la contraseña proporcionada por el usuario es transformada mediante una *función de hash* antes de su persistencia en la base de datos. Esta operación se lleva a cabo utilizando la función bcrypt.hash, la cual acepta como parámetros:

- La contraseña en texto plano.
- El número de rondas de *salting*<sup>7</sup> (en este caso, 10).

```
16
17
18 async create(data) {
19   try {
20     data.contraseña = await bcrypt.hash(data.contraseña, 10);
21     return await Cliente.create(data);
22   } catch (error) {
23     console.error("Error al crear el cliente:", error);
24     throw new Error("Error al crear el cliente.");
25   }
26 }
```

Función hash

Variable que almacena la contraseña introducida por el usuario

Figura 15 Método que registra a un usuario y encripta su contraseña

<sup>6</sup> Es un algoritmo criptográfico de cifrado simétrico por bloques que utiliza una clave variable de hasta 448 bits para cifrar bloques de 64 bits mediante 16 rondas de sustitución y permutación.

<sup>7</sup> El salting consiste en agregar una cadena aleatoria (llamada *salt*) a la contraseña antes de hashearla.

Cabe destacar, que se valida la entrada de datos tanto en cliente como en servidor, usando esquemas Zod para validación estructurada, validación en tiempo real en formularios y limpieza para prevenir XSS mediante escape de caracteres.

Se protege contra CSRF con tokens en formularios, validación en servidor y cookies seguras.

No se almacenan contraseñas en cliente, se transmiten datos por HTTPS y se usan tokens en lugar de credenciales. Todas las comunicaciones emplean HTTPS con certificados SSL/TLS y redirección automática de HTTP a HTTPS. Los tokens se almacenan en cookies HttpOnly con expiración y renovación segura.

#### 4.2.8. Generación de *PDF* con *EJS* y *html-pdf*.

EJS permite crear archivos HTML dinámicos usando los datos enviados desde la lógica de service.

*Html-pdf* es una librería para Node.js que permite convertir un string de HTML directamente en un archivo PDF. Internamente usa *PhantomJS* (un navegador headless) para renderizar el HTML y convertirlo en PDF.

##### Flujo.

- Renderizar HTML con datos usando EJS.
- El string HTML se pasa a *html-pdf* que crea el PDF basado en el contenido del html.
- Generar el PDF directamente desde esa vista renderizada Figura 16.



**AirLink**

Detalles del vuelo - Romel Romero	
Número de vuelo	BA2025
Fecha de salida	10 jun 2025, 14:00
Fecha de llegada	10 jun 2025, 18:30
Aeropuerto de origen	Aeropuerto de Londres-Heathrow
Aeropuerto de llegada	Aeropuerto Internacional de Nueva York-JFK
Asiento	6B (Fila: 6, Columna: B )
Precio	570 EUR
Localizador	5J53ES
Aerolínea operadora	British Airways

Figura 16 Interfaz gráfica del pdf con la información de los billetes de los pasajeros

A continuación, mostraremos un diagrama con las funcionalidades que realiza nuestra aplicación.

Implementamos **APIs**, mediante endpoints que permiten que el cliente realice solicitudes al servidor para obtener la información solicitada. Estas APIs devolverán en formato JSON, la información que se solicite y serán mostradas en la parte cliente.



- **GET:** Recupera datos del servidor.
- **POST:** Envía datos al servidor para crear un nuevo recurso.
- **PUT:** Actualiza un recurso existente en el servidor.
- **DELETE:** Elimina un recurso del servidor.

### 5.2.1 Funcionalidad que aplica a un usuario con rol *User* Figura 18.



A continuación, se describe detalladamente el proceso del usuario, destacando los principales *puntos de integración* con los servicios expuestos por el backend mediante rutas RESTful.

### **Autenticación de usuario.**

- **Registro.**

Un nuevo usuario puede registrarse mediante el formulario de registro, que realiza una petición ***POST /clientes***.

- **Inicio de sesión.**

Un usuario existente puede autenticarse en el sistema mediante el formulario de login. Esta acción genera una solicitud ***POST /clientes/login***. Si las credenciales son válidas, el backend devuelve un token de acceso (JWT), que se utilizará para autenticar las peticiones futuras.

- **Inicio de sesión sin contraseña.**

Un usuario existente puede autenticarse en el sistema mediante un código de verificación que se envía a su correo si no recuerda su contraseña de login. Esta acción genera varias solicitudes en cadena: en primer lugar se llama a ***POST /clientes/enviar-codigo***, endpoint que envía el código de verificación al usuario; cuando ya se tiene el código se procede a logear al usuario correspondiente llamando a ***POST /clientes/verificar-codigo***. Si el código es correcto, el backend devuelve un token de acceso (JWT), que se utilizará para autenticar las peticiones futuras.

### **Búsqueda y selección de vuelos.**

- **Carga de aeropuertos.**

Una vez autenticado, el cliente accede a la página de inicio, donde puede buscar vuelos. Para ello, se obtiene información de los aeropuertos disponibles mediante la ruta ***GET /aeropuertos***.

- **Filtrado de vuelos.**

El cliente introduce los parámetros de búsqueda (origen, destino, fecha, etc.), y la aplicación realiza una solicitud para obtener los vuelos que cumplen con los parámetros solicitados, mediante la ruta ***POST /vuelos/buscador-vuelos***.

## Selección de asiento y reserva.

- **Visualización de asientos.**

Al seleccionar un vuelo, se muestra la disponibilidad de asientos asociados a dicho vuelo, mediante la ruta **GET /asientos/vuelo/:numero\_vuelo**.

- **Reserva de vuelo.**

Si el cliente selecciona un asiento específico, se realiza la reserva a través de **PUT /reservas/realizar-reserva**. Si no se selecciona asiento, el sistema asigna automáticamente uno disponible.

- **Confirmación.**

Tras una reserva exitosa, el cliente es redirigido a una página de confirmación y recibe un correo electrónico con la información del billete. Además, en la página de confirmación encontramos un *botón* con opción a descargar un archivo en formato pdf con la información con los billetes que se han reservado.

## Gestión de cuenta del cliente.

Desde la sección de administración del perfil del cliente, se accede a distintas funcionalidades, todas ellas protegidas por autenticación:

- **Ver perfil del cliente.**

El cliente puede mirar la información de su perfil mediante **GET /clientes/perfil-cliente**.

- **Actualización de datos.**

El cliente puede modificar su información personal mediante **PUT /clientes**.

- **Eliminación de cuenta.**

Si desea eliminar su cuenta, puede hacerlo a través de la ruta **DELETE /clientes**.

- **Consulta de reservas.**

El cliente puede visualizar todas sus reservas activas mediante **GET /reservas/mis-reservas**.

- **Eliminación de billetes específicos.**

También puede eliminar billetes individuales asociados a una reserva mediante **DELETE /reservas/:id\_reserva/billetes/:id\_billete**.

### 5.2.2. Funcionalidad que aplica a un usuario con rol **Admin** Figura 19.

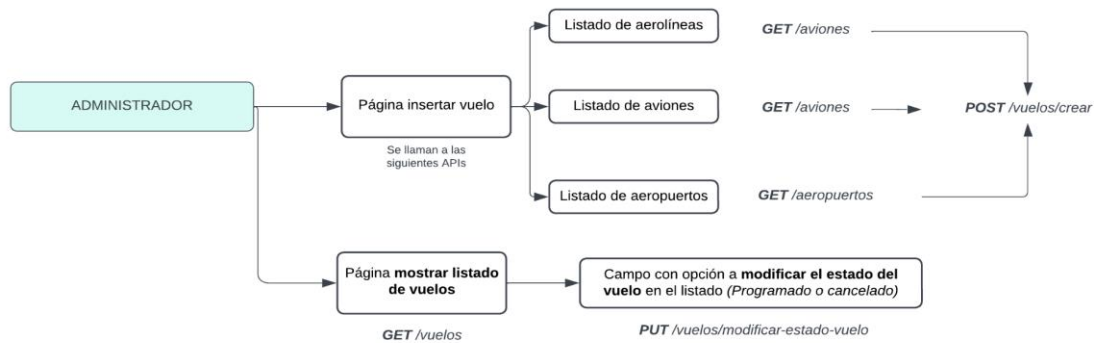


Figura 19 Diagrama de integración rol Administrador

El punto de inicio es el administrador, quien realiza acciones dentro del sistema.

- **Insertar vuelo:** El administrador tiene la opción de acceder a una página para insertar un nuevo vuelo.

Al acceder a esta página, se realizan varias llamadas a diferentes *APIs* para obtener información necesaria para el formulario de inserción del vuelo:

- **Listado de aerolíneas:** Se hace una petición **GET /aerolineas** para obtener la lista de aerolíneas disponibles.
- **Listado de aviones:** Se hace una petición **GET /aviones** para obtener la lista de aviones disponibles.
- **Listado de aeropuertos:** Se hace una petición **GET /aeropuertos** para obtener la lista de aeropuertos de origen y destino.
- *Se añadirán parámetros extras* que serán introducidos manualmente por el usuario Admin.

Una vez que el administrador completa el formulario con la información del nuevo vuelo (aerolínea, avión, origen, destino, fecha, hora,...) y lo envía, se realiza una petición **POST /vuelos**. Esta petición enviará los datos del nuevo vuelo al servidor para que sea creado.

- **Listado de vuelos.**

El administrador también tiene la opción de acceder a una página para ver la lista de vuelos existentes.

Para mostrar esta lista, se realiza una petición **GET /vuelos**. El servidor responderá con la información de todos los vuelos. Además, se pueden filtrar los vuelos según el estado del vuelo, la fecha de ida, vuelta, así como por origen y destino del vuelo.

Dentro del listado de vuelos, existe un elemento (un botón para cancelar) que permite al administrador modificar el estado de un vuelo programado específico. Los estados posibles que se mencionan son "*Programado*" o "*Cancelado*". Cuando el administrador realiza un cambio en el estado de un vuelo ("*Cancelado*"), se realiza una petición **PUT /vuelos/modificar-estado-vuelo**. Esta petición enviará el nuevo estado. El servidor procesa esta petición y actualiza la información del vuelo en el sistema.

### 5.3. Funcionalidades principales a nivel cliente.

#### 5.3.1. Autenticación.

El usuario puede registrarse completando un formulario de registro. Para iniciar sesión, puede autenticarse con su contraseña a través del formulario de login.

Si no recuerda su contraseña, puede optar por el inicio de sesión sin contraseña: primero solicita un código de verificación enviado a su correo y luego lo introduce para completar el acceso. En ambos casos, tras una autenticación exitosa, se le concede un token para futuras peticiones.



#### 5.3.2. Reserva del vuelo.

El proceso inicia con la *búsqueda de vuelos desde el dashboard*, donde el usuario selecciona el origen, destino, fechas y el número de pasajeros. Se lleva a cabo una validación de datos antes de envío: verificación de campos obligatorios, validación de fechas (ida anterior a vuelta) y origen y destino diferentes.

Los datos de búsqueda son validados en la API. Una vez recibidos los resultados, se presentan *los vuelos disponibles* para la selección de ida y, si aplica, de regreso.

El usuario elige el vuelo de ida y, en caso de viaje de ida y vuelta, también selecciona el vuelo de regreso. Se muestra la información de precio, horarios y aerolínea. Los vuelos se filtran de más temprano al más tardío.



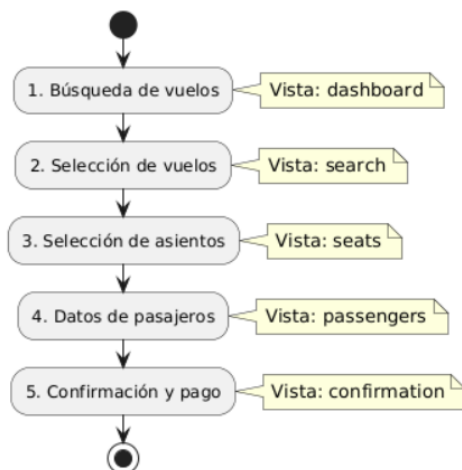
A continuación, se *muestra el mapa de asientos para el vuelo de ida*, permitiendo que el usuario seleccione los asientos según la cantidad de pasajeros. Cabe destacar que se realiza una representación gráfica de la distribución de filas y columnas etiquetadas y una selección interactiva con validación, límite de selección según número de pasajeros, imposibilidad de seleccionar asientos ocupados y visualización de selecciones actuales.

Los asientos se diferencian por clases y estados: asientos de primera clase, business y económica; estados disponible, ocupado, seleccionado y una leyenda explicativa.



Si es un viaje de ida y vuelta, este proceso se repite para el vuelo de regreso. Se da la opción al usuario de que elija asiento o que el sistema elija por él.

Luego, se *generan formularios para ingresar los datos de cada pasajero*, donde el usuario completa la información personal y de contacto, y todos los datos son validados.



Finalmente, se presenta un resumen completo de la reserva para su confirmación y pago, se crea la reserva en el sistema, se muestra la página de éxito y se envía un email de confirmación con la opción de imprimir y descargar un archivo en formato pdf.

### 5.3.3. Gestión de reservas.

Para gestionar las reservas, el usuario accede a la sección "*Mis reservas*" desde su perfil, navegando hasta allí y seleccionando la opción correspondiente.

Se despliega una *lista con las reservas existentes* que incluye información básica. Al seleccionar una reserva específica, se *muestra la información detallada, incluyendo los datos de vuelos y pasajeros*, incluyendo los billetes que hay reservados por cada reserva.



Además, el usuario tiene la *opción de cancelar la reserva y/o billetes*, procedimiento que incluye una confirmación previa para evitar acciones accidentales.

### 5.3.4. Gestión de perfil.

El acceso al perfil se realiza desde el menú de usuario, haciendo clic en el avatar y seleccionando la opción "Mi Perfil".

Dentro del perfil, el usuario puede visualizar su información básica y el historial de actividad registrado.

Para actualizar sus datos, se ofrece un formulario con la información actual, que incluye validación de los cambios y la correspondiente confirmación para aplicar las modificaciones.



También es posible cambiar la contraseña. Por último, el usuario puede eliminar su cuenta, lo cual implica mostrar una advertencia sobre las consecuencias, solicitar confirmación mediante contraseña y ejecutar el proceso de eliminación.

### 5.3.5. Panel de administración

- **Creación de nuevos vuelos.**

A través de un formulario, el administrador puede seleccionar el aeropuerto de origen y destino, el avión y la aerolínea. Además, se introduce el número de vuelo, el precio del billete y se selecciona la fecha y hora de salida y llegada.

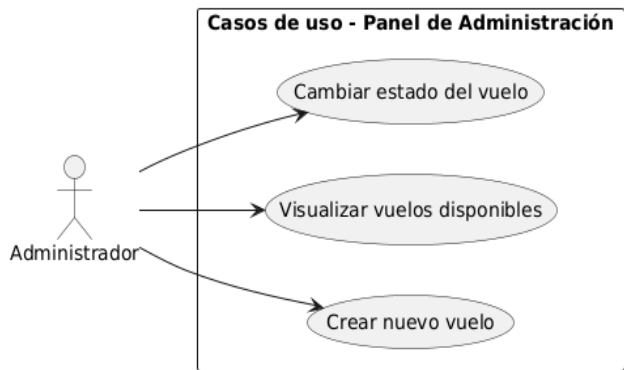
El campo de estado del vuelo se inicia con el valor por defecto “Programado”.

- **Visualización de vuelos disponibles.**

Se muestran los vuelos disponibles junto con los datos correspondientes de cada uno. Además, se pueden filtrar los vuelos según el estado del vuelo, la fecha de ida, vuelta, así como por origen y destino del vuelo.

- **Cambio de estado de vuelos.**

En la vista de vuelos disponibles, se presenta un botón que permite cambiar el estado del vuelo a “Cancelado” al ser presionado.



## 6. Despliegue de la aplicación.

En este apartado se explicará todo lo relacionado con el despliegue de la aplicación. Aquí podemos describir una visión global de cómo es nuestra estructura Figura 20.

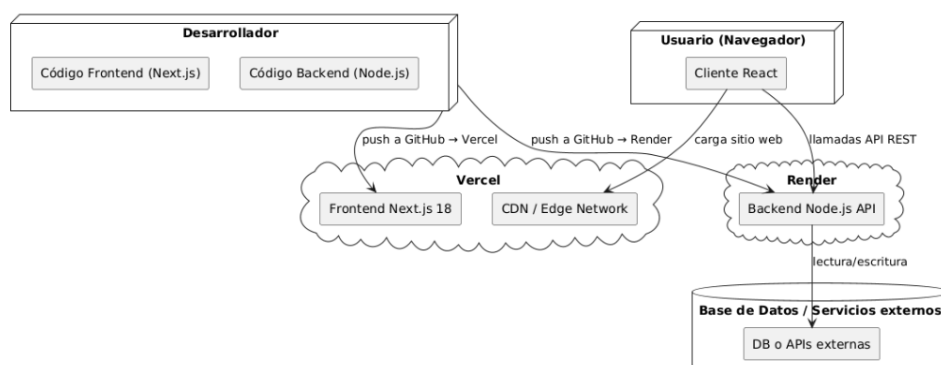


Figura 20 Diagrama de despliegue general cliente-servidor

## 6.1. Despliegue de la parte servidor (Node y PostgreSQL) Figura 21.

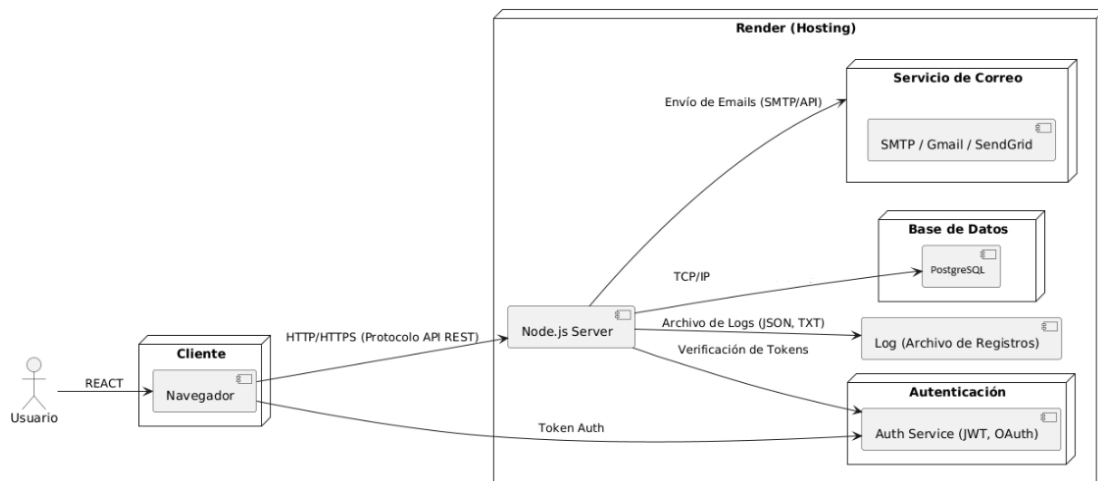


Figura 21 Diagrama de despliegue parte servidor

### 6.1.1. Plataforma de despliegue.

La plataforma para alojar nuestro backend ha sido Render que permite alojar aplicaciones de Nodejs. Se trata de un *Platform-as-a-Service (PaaS)* que gestiona automáticamente aspectos como balanceo de carga, reinicios ante errores, y despliegue continuo desde un repositorio remoto.

El entorno de ejecución proporciona acceso a *variables de entorno*, *logs* en tiempo real y soporte para *bases de datos PostgreSQL*.

### 6.1.2. Proceso de Despliegue.

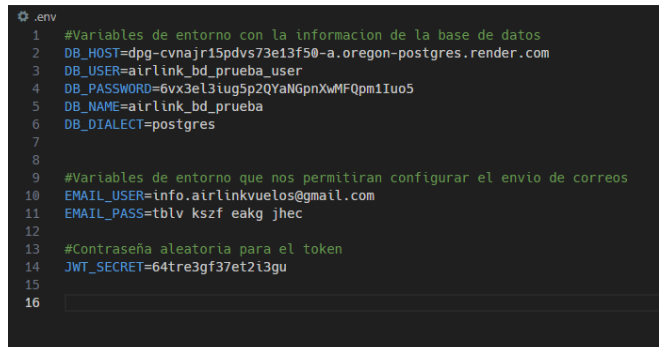
El proyecto está alojado en la *rama master* y automáticamente Render detecta ese cambio y despliega la nueva versión. Llevamos a cabo por ello un *Despliegue Continuo (CD)*. El despliegue es automático después de pasar las pruebas. Destacamos las fases por las que pasa el código antes de ser subido. Con solo hacer un git push Render se encarga de:

- Descargar el código actualizado.
- Ejecutar npm install.
- Iniciar la aplicación con npm start.

### 6.1.3. Uso de Variables de Entorno en local y en Render (.env) Figura 22.

Para evitar exponer datos sensibles (como contraseñas, usuarios de base de datos, claves API, etc.) directamente en el código fuente, utilizamos variables de entorno mediante un archivo `.env`.

Estas variables permiten que la aplicación sea más segura, configurable y adaptable a distintos entornos



```
1 #Variables de entorno con la informacion de la base de datos
2 DB_HOST=dpg-cvnajr15pdvs73e13f50-a.oregon-postgres.render.com
3 DB_USER=airlink_bd_prueba_user
4 DB_PASSWORD=6vx3e13iug5p2QYaNGpnXwMFQpm1Iuo5
5 DB_NAME=airlink_bd_prueba
6 DB_DIALECT=postgres
7
8
9 #Variables de entorno que nos permitan configurar el envio de correos
10 EMAIL_USER=info.airlinkvuelos@gmail.com
11 EMAIL_PASS=tblv kszf eakg jhec
12
13 #Contraseña aleatoria para el token
14 JWT_SECRET=64tre3gf37et2i3gu
15
16
```

Figura 22 Fichero `.env` donde están las variables de entornos

(desarrollo, testing, producción). Variables de entorno utilizadas:

- Configuración de la base de datos: `DB_HOST`, `DB_USER`, `DB_PASSWORD`, `DB_NAME` y `DB_DIALECT`.
- Configuración del envío de correos: `EMAIL_USER` y `EMAIL_PASS`.
- Contraseña secreta para el token: `JWT_TOKEN`.

Render proporciona un panel de configuración para definir estas variables de entorno de forma segura. Se ingresa en la sección *Environment*. Render las inyecta automáticamente en el entorno de ejecución de la aplicación cuando esta se despliega o reinicia.

### 6.1.4. Logs y monitoreo.

Render ofrece un soporte para la visualización y análisis de los logs generados por la aplicación desplegada.

Render captura automáticamente toda la salida estándar y de error que genera la aplicación Node.js. Estos logs se pueden visualizar en tiempo real desde el panel web de Render, facilitando la identificación rápida de errores y facilitando la depuración de código Figura 23.

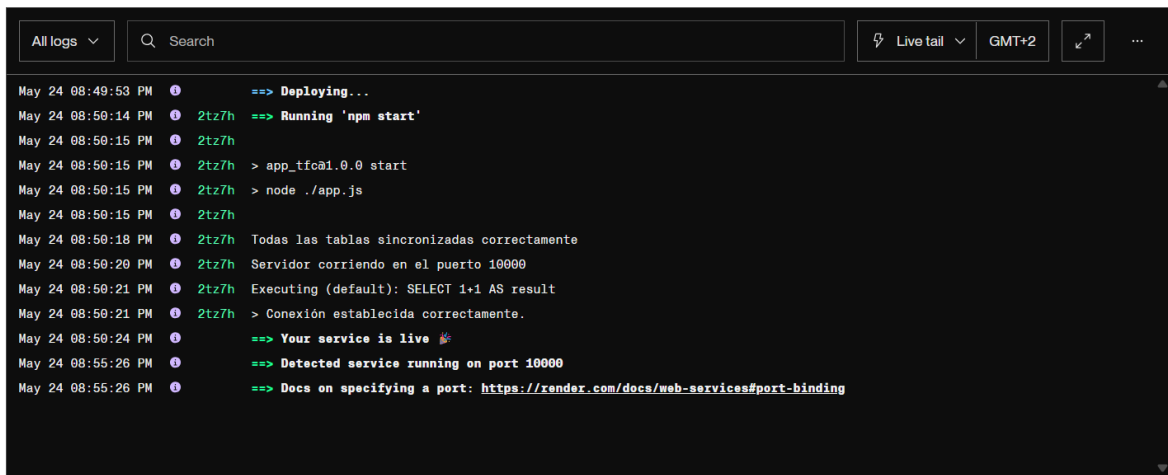


Figura 23 Ventana donde se miran los logs en Render

### 6.1.5. Proceso de Build y despliegue en Render.

Cuando desplegamos una aplicación en Render, se sigue un flujo definido para poner en ejecución el servidor. Incluye la instalación de dependencias, tareas previas, y el arranque de la aplicación.

### 6.2 Despliegue de la parte cliente Figura 24.

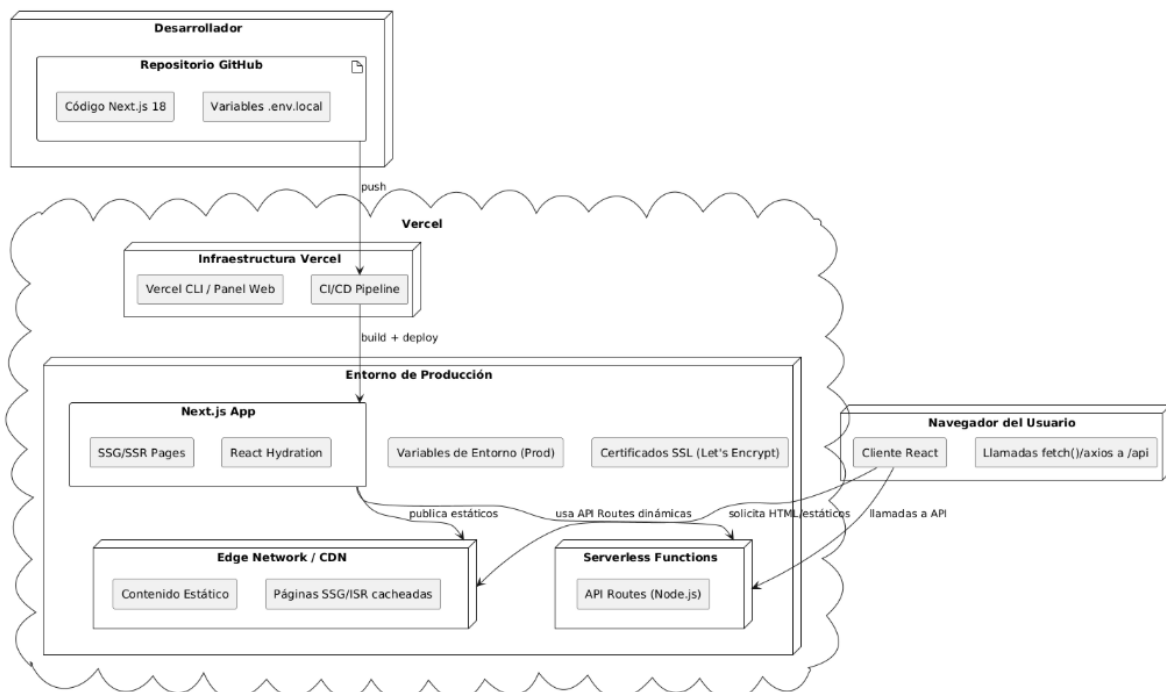


Figura 24 Diagrama de despliegue parte cliente

Para el despliegue de la aplicación en la parte cliente utilizaremos Vercel, una plataforma de despliegue enfocada en aplicaciones frontend, con integración nativa

para proyectos basados en Next.js. Podemos mencionar algunas características clave:

- Despliegue automático desde Git.
- Previsualización por Pull Request.
- Análisis de rendimiento.
- Configuración sencilla y rápida para entornos de producción.

### **6.2.1. Configuración de despliegue.**

Para conectar el repositorio a Vercel tendremos que hacer lo siguiente:

- Crear cuenta en Vercel.
- Importar repositorio de GitHub/GitLab/Bitbucket.
- Seleccionar rama principal.

Además, tendremos que configurar las variables de entorno:

- Añadir las mismas variables que en desarrollo.
- Ajustar URLs para producción.

Por otro lado, llevamos a cabo una configuración de dominios y redirecciones:

- Añadir dominio personalizado.
- Configurar redirecciones si es necesario.
- Configurar certificados SSL.

### **6.2.2. Extensibilidad y mantenimiento y actualización de dependencias.**

Asegurarnos regularmente de vulnerabilidades ejecutando *npm audit* periódicamente y resolver esas vulnerabilidades críticas inmediatamente. Además, llevamos a cabo una actualización de dependencias con *npm update*.

Además, ha habido que llevar a cabo pruebas, entre ellas, verificación de funcionalidades clave, revisión de compatibilidad con navegadores y comprobación de rendimiento.

### 6.2.3. Requisitos previos y configuración del entorno.

- **Requisitos previos.**

Se requiere tener instalado *Node.js en su versión 16* o superior, siendo recomendable utilizar la versión 18 o más reciente. Además, es necesario contar con un gestor de paquetes, *como npm o yarn*, para la instalación y administración de dependencias. Finalmente, se debe disponer de acceso a la *API de AirLink* para poder realizar las integraciones correspondientes.

- **Configuración del entorno.**

Se debe clonar el repositorio, instalar las dependencias y configurar las variables de entorno creando un archivo *env.local* con la información necesaria.

- **Comandos principales.**

- npm run dev: Iniciar servidor de desarrollo.
- npm run build: Construir para producción.
- npm run start: Iniciar versión de producción.
- npm run lint: Ejecutar linter.

## 7. Accesibilidad.

Uso de atributos ARIA, empleo de roles para definir propósito de elementos. Así mismo, se emplean estados para componentes interactivos y etiquetas para elementos sin texto visible.

- **Estructura semántica HTML.**

Uso de elementos semánticos (nav, main, etc.), jerarquía correcta de encabezados y listas para grupos de elementos relacionados.

- **Contraste de colores adecuado.**

Relación de contraste mínima de 4.5:1 para texto normal, relación de contraste mínima de 3:1 para texto grande y modos de alto contraste.

- **Soporte para navegación por teclado.**

Orden de tabulación lógico, indicadores de foco visibles y atajos de teclado para acciones comunes.

- **Textos alternativos para imágenes y atributos alt descriptivos.**



## **8. Responsividad.**

El diseño de nuestra aplicación se basa en un enfoque *mobile-first*, utilizando Tailwind CSS para definir breakpoints que permiten adaptar el diseño a tabletas y escritorios. Se emplean layouts flexibles que responden a distintos tamaños de pantalla, con los siguientes breakpoints configurados son: sm (640px), md (768px), lg (1024px), xl (1280px) y 2xl (1536px).

En dispositivos móviles, se usa un menú hamburguesa mediante un componente Sheet que despliega un menú lateral. Además, el sistema detecta el tamaño de pantalla para ajustar la navegación y dar una experiencia adaptada según el dispositivo.

## **9. Componentes UI/UX.**

### **9.1. Componentes base.**

Incluyen botones, inputs, selects, etc., provenientes de la librería *shadcn/ui*. Estos componentes son accesibles y personalizables, con variantes para diferentes estados y estilos, y cuentan con integración completa con Tailwind CSS. La personalización se realiza mediante clases utilitarias de Tailwind, soportando diseño responsivo con breakpoints y temas personalizados.

### **9.2. Componentes compuestos.**

Se incluyen cards para mostrar información de vuelos, con detalles como origen, destino, horarios, precio y clase, además de acciones como seleccionar o ver detalles. Los formularios cuentan con validación, mostrando campos con etiquetas y mensajes de error, validación en tiempo real, y estados de carga y éxito. Las tablas de datos ofrecen columnas personalizables, ordenación por múltiples campos y filtros avanzados.

### **9.3. Feedback al usuario.**

Se usan toasts para notificaciones de éxito, error, advertencia e información, con duración configurable y posicionamiento flexible.

Los estados de carga incluyen spinners y skeletons, desactivando controles durante la carga y mostrando mensajes informativos. Los mensajes de error y éxito se

presentan como alertas contextuales con mensajes descriptivos y acciones de recuperación.

Las confirmaciones para acciones importantes se gestionan mediante diálogos modales con botones de confirmación y cancelación, además de explicaciones claras de las consecuencias.

## **10. Manejo de errores.**

Se capturan errores mediante bloques try/catch en funciones asíncronas, con manejo específico según el tipo de error y feedback visual al usuario. Esto mismo además se aplica a la parte servidora al trabajar con la lógica de negocio que hace las solicitudes a la base de datos. Lo vemos también aplicado en los controladores que envían las solicitudes de las APIs.

Para errores no capturados, se utiliza un componente *ErrorBoundary* que atrapa errores en el árbol de componentes, muestra una UI de fallback y registra los errores para análisis posterior.

El feedback visual se complementa con alertas y toasts que ofrecen mensajes claros y accionables, diferenciados según gravedad, y opciones de recuperación cuando es posible.

## **11. Optimización de rendimiento.**

Se aplica carga diferida de componentes mediante importación dinámica, reduciendo el bundle inicial y cargando bajo demanda. La optimización de imágenes se realiza con next/image, que permite carga optimizada, redimensionamiento automático, soporte para formatos modernos como WebP y AVIF, carga diferida y placeholders difuminados.

La memorización de componentes con React.memo previene re-renderizados innecesarios, optimizando listas y componentes pesados con comparación personalizada de props.

Además, se usan *useMemo* y *useCallback* para memorizar funciones y valores calculados, asegurando que los cálculos costosos solo se ejecuten cuando cambian las dependencias, estabilizando referencias para optimizar efectos y callbacks.

## **12. Desarrollo de la BBDD.**

En este punto se documentará todo lo referido con la *Base de Datos*.

La base de datos se comprenderá de 9 tablas: *t\_aerolineas*, *t\_aeropuertos*, *t\_asientos*, *t\_aviones*, *t\_billetes*, *t\_clientes*, *t\_pasajeros*, *t\_reservas*, *t\_vuelos*.

## 12.1. Diagrama Entidad-Relación Figura 25 .

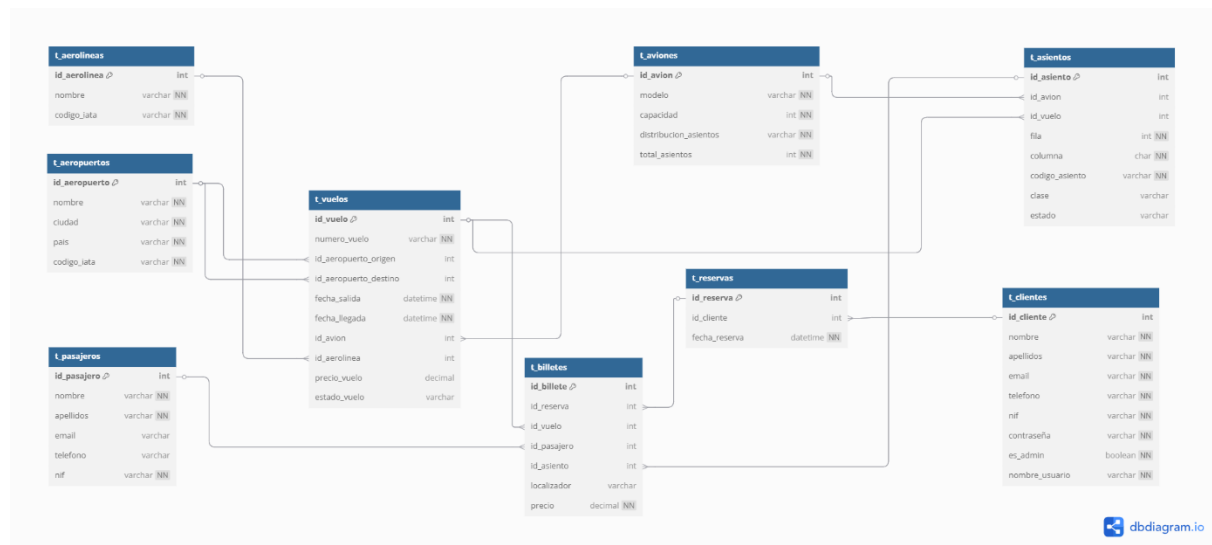


Figura 25 Esquema relaciona de la BBDD

### 12.1.1. Cardinalidad y relaciones de las entidades.

- *t\_aerolineas* y *t\_vuelos*.
  - Una aerolínea opera muchos vuelos (1:N).
  - Cada vuelo lo opera solo una aerolínea.
- *t\_aeropuertos* y *t\_vuelos*.
  - Un aeropuerto puede ser origen o destino de muchos vuelos (1:N en ambos casos).
  - Cada vuelo tiene un solo aeropuerto de origen y uno de destino.
- *t\_aviones* y *t\_vuelos*.
  - Un avión puede hacer muchos vuelos (1:N).
  - Cada vuelo se asigna a un solo avión.
- *t\_aviones* y *t\_asientos*.
  - Un avión tiene muchos asientos (1:N).
  - Cada asiento pertenece a un avión.
- *t\_vuelos* y *t\_asientos*.
  - Un vuelo puede tener muchos asientos asignados (1:N).
  - Un asiento puede estar asignado a un vuelo.

- *t\_clientes* y *t\_reservas*.
  - Un cliente puede hacer muchas reservas (1:N).
  - Cada reserva la hace un único cliente.
- *t\_reservas* y *t\_billetes*.
  - Una reserva puede tener muchos billetes (1:N).
  - Cada billete pertenece a una sola reserva.
- *t\_vuelos* y *t\_billetes*.
  - Un vuelo puede tener muchos billetes vendidos (1:N).
  - Cada billete es para un único vuelo.
- *t\_pasajeros* y *t\_billetes*.
  - Un pasajero puede tener varios billetes (vuelos) (1:N).
  - Cada billete es de un solo pasajero.
- *t\_asientos* y *t\_billetes*.
  - Un asiento puede estar vinculado a muchos billetes a lo largo del tiempo (en diferentes vuelos) (1:N si hay reciclaje de asientos).
  - Cada billete tiene asignado un asiento.

### 12.1.2. Atributos, tipos de datos y restricciones de las tablas.

Descripción detallada de las tablas de la Base de Datos.

- **Tabla *t\_asientos*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_asiento	integer	NO	nextval('t_asientos_id_asiento_seq'::regclass)	PRIMARY KEY	[null]	[null]
id_avion	integer	YES	[null]	FOREIGN KEY	t_aviones	id_avion
id_vuelo	integer	YES	[null]	FOREIGN KEY	t_vuelos	id_vuelo
fila	integer	NO	[null]	NO RESTRICTION	[null]	[null]
columna	character	NO	[null]	NO RESTRICTION	[null]	[null]
codigo_asiento	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
clase	character varying	YES	'economica'::character varying	NO RESTRICTION	[null]	[null]
estado	character varying	YES	'disponible'::character varying	NO RESTRICTION	[null]	[null]

- **Tabla *t\_aeropuertos*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_aeropuerto	integer	NO	nextval('t_aeropuertos_id_aeropuerto_seq'::regclass)	PRIMARY KEY	[null]	[null]
nombre	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
ciudad	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
pais	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
codigo_iata	character varying	NO	[null]	UNIQUE	[null]	[null]

- **Tabla *t\_aerolineas*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_aerolinea	integer	NO	nextval('t_aerolineas_id_aerolinea_seq'::regclass)	PRIMARY KEY	[null]	[null]
nombre	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
codigo_iata	character varying	NO	[null]	UNIQUE	[null]	[null]

- **Tabla *t\_clientes*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type character varying	foreign_table_name name	foreign_column_name name
id_cliente	integer	NO	nextval('t_clientes_id_cliente_seq'::regclass)	PRIMARY KEY	t_clientes	id_cliente
nombre	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
apellidos	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
email	character varying	NO	[null]	UNIQUE	t_clientes	email
telefono	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
nif	character varying	NO	[null]	UNIQUE	t_clientes	nif
contraseña	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
es_admin	boolean	YES	false	NO RESTRICTION	[null]	[null]
nombre_usuario	character varying	NO	[null]	UNIQUE	t_clientes	nombre_usuario

- **Tabla *t\_reservas*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_reserva	integer	NO	nextval('t_reservas_id_reserva_seq'::regclass)	PRIMARY KEY	[null]	[null]
id_cliente	integer	YES	[null]	FOREIGN KEY	t_clientes	id_cliente
fecha_reserva	timestamp with time zone	NO	[null]	NO RESTRICTION	[null]	[null]

- **Tabla *t\_pasajeros*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_pasajero	integer	NO	nextval('t_pasajeros_id_pasajero_seq'::regclass)	PRIMARY KEY	[null]	[null]
nombre	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
apellidos	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
email	character varying	YES	[null]	NO RESTRICTION	[null]	[null]
telefono	character varying	YES	[null]	NO RESTRICTION	[null]	[null]
nif	character varying	NO	[null]	NO RESTRICTION	[null]	[null]

- **Tabla *t\_vuelos*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_vuelo	integer	NO	nextval('t_vuelos_id_vuelo_seq'::regclass)	PRIMARY KEY	[null]	[null]
numero_vuelo	character varying	NO	[null]	UNIQUE	[null]	[null]
id_aeropuerto_origen	integer	YES	[null]	FOREIGN KEY	t_aeropuertos	id_aeropuerto
id_aeropuerto_destino	integer	YES	[null]	FOREIGN KEY	t_aeropuertos	id_aeropuerto
fecha_salida	timestamp with time zone	NO	[null]	NO RESTRICTION	[null]	[null]
fecha_llegada	timestamp with time zone	NO	[null]	NO RESTRICTION	[null]	[null]
id_avion	integer	YES	[null]	FOREIGN KEY	t_aviones	id_avion
id_aerolinea	integer	YES	[null]	FOREIGN KEY	t_aerolineas	id_aerolinea
precio_vuelo	double precision	NO	[null]	NO RESTRICTION	[null]	[null]
estado_vuelo	USER-DEFINED	NO	'Programado'::enum_t_vuelos_estado_v...	NO RESTRICTION	[null]	[null]

- **Tabla *t\_aviones*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_avion	integer	NO	nextval('t_aviones_id_avion_seq'::regclass)	PRIMARY KEY	[null]	[null]
modelo	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
capacidad	integer	NO	[null]	NO RESTRICTION	[null]	[null]
distribucion_asientos	character varying	NO	[null]	NO RESTRICTION	[null]	[null]
total_asientos	integer	NO	[null]	NO RESTRICTION	[null]	[null]

- **Tabla *t\_billetes*.**

column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying	constraint_type text	foreign_table name	foreign_column name
id_billete	integer	NO	nextval('t_billetes_id_billete_seq'::regclass)	PRIMARY KEY	[null]	[null]
id_reserva	integer	YES	[null]	FOREIGN KEY	t_reservas	id_reserva
id_vuelo	integer	YES	[null]	FOREIGN KEY	t_vuelos	id_vuelo
id_pasajero	integer	YES	[null]	FOREIGN KEY	t_pasajeros	id_pasajero
id_asiento	integer	YES	[null]	FOREIGN KEY	t_asientos	id_asiento
localizador	character varying	NO	[null]	UNIQUE	[null]	[null]
precio	numeric	NO	[null]	NO RESTRICTION	[null]	[null]

### 13. Planificación y control de cambios.

Hemos usado Git que nos ha permitido desarrollar de forma paralela mediante ramas. Esto nos permitió un trabajo colaborativo para revisar y validar código antes de subirlo a la rama principal de desarrollo que sería desplegada (Vercel o Render).

Hemos tenido varias ramas:

- **Master (o main):** Aquí solo se hace merge de versiones que ya pasaron pruebas y están listas para ser desplegadas.
- **Develop:** se fusionan aquí todas las ramas de trabajo, feature.
- **Feature/funcionalidad:** son las ramas temporales para cada nueva funcionalidad y se crean a partir de develop. Cuando la funcionalidad está lista y revisada, se hace merge a develop. Antes de subir los cambios ha sido necesario bajarnos los cambios en el servidor mediante pull o rebase, solucionar conflictos si los hubiese y hacer push.

Cabe destacar, que han sido necesario realizar Pull Requests para revisar el código de cada compañero y así comprobar si hubiera algún error que diera error en develop.

## 14. Resultados y discusión.

El resultado final de AirLink es una aplicación web completamente funcional que cumple con los objetivos establecidos. Los principales logros incluyen:

Usuarios pueden registrarse, iniciar sesión (con o sin contraseña usando códigos de verificación), buscar vuelos, seleccionar asientos, realizar reservas, y gestionar sus perfiles. Los administradores pueden gestionar vuelos y otros recursos a través del panel de administración. Envío de correos electrónicos con confirmaciones de reservas y generación de PDFs con detalles de billetes.

La implementación del patrón MVC en el servidor y una arquitectura modular en el cliente asegura un código organizado, escalable, y fácil de mantener.

La interfaz es intuitiva, responsive (mobile-first), y accesible, con soporte multilingüe y un diseño optimizado con Tailwind CSS y Shadcn/ui.

Autenticación segura con JWT, encriptación de contraseñas con bcrypt, y comunicaciones cifradas con HTTPS/SSL.

Optimización mediante carga diferida, memorización (React.memo, useMemo, useCallback), y uso de next/image para imágenes.

La aplicación cumple con los estándares esperados, integrando tecnologías modernas y ofreciendo una experiencia de usuario fluida. La modularidad del código facilita futuras ampliaciones, como agregar nuevas funcionalidades (por ejemplo, pagos en línea, mejor distribución en la distribución de asientos para cualquier tipo de avión, meter un servicio de API externo para información de vuelos en tiempo real, añadir vuelos con escalas, soporte para más idiomas o mejoras en accesibilidad WCAG).

Configurar el soporte multilingüe con next-intl y manejar la sincronización de la base de datos con Sequelize requirieron ajustes para garantizar consistencia. La integración de Nodemailer y html-pdf presentó retos iniciales debido a la configuración de SMTP y la renderización de PDFs en producción.

La aplicación no incluye funcionalidades avanzadas como pagos en línea o integración con APIs de aerolíneas reales, lo que podría ser un paso futuro.

## 15. Conclusiones.

El desarrollo de AirLink ha sido una experiencia que ha permitido aplicar y consolidar los conocimientos adquiridos durante el ciclo formativo de DAW. La aplicación demuestra la capacidad de diseñar, desarrollar, y desplegar una solución web completa, abarcando aspectos clave como arquitectura, seguridad, accesibilidad, y optimización de una aplicación web. Entre los principales aprendizajes que se pueden mencionar:

- La importancia de una arquitectura modular para mantener un código escalable y mantenible usando una arquitectura limpia, usando componentes, módulos, servicios y el patrón MVC.
- La relevancia de la seguridad en aplicaciones web, especialmente en la gestión de autenticación y datos sensibles (uso de JWT, encriptación de contraseñas, así como protección de rutas).
- La necesidad de pruebas exhaustivas para garantizar la calidad y usabilidad.
- La eficacia de herramientas modernas como Next.js, Tailwind CSS, Vercel, Nodejs o Render para agilizar el desarrollo y despliegue.

AirLink sirve como una base sólida para la integración de nuevas funcionalidades o la mejora de la escalabilidad. Este proyecto refleja el potencial de las tecnologías Open Source para crear aplicaciones web robustas y prepara al desarrollador para desafíos reales.



## 16. Bibliografía y referencias.

- <https://medium.com/@diego.coder/debugging-de-aplicaciones-node-js-con-vscode-5f02e5d2e900>
- <https://sequelize.org/>
- [https://github.com/carlos-paezf/NodeJS\\_MVC\\_MySQL\\_MongoDB/tree/main](https://github.com/carlos-paezf/NodeJS_MVC_MySQL_MongoDB/tree/main)
- [https://www.youtube.com/watch?v=Ck9O-QhSnNo&t=501s&ab\\_channel=MonkeyWit](https://www.youtube.com/watch?v=Ck9O-QhSnNo&t=501s&ab_channel=MonkeyWit)
- <https://www.postgresql.org/>
- <https://qalified.com/es/blog/postman-para-api-testing/>
- <https://medium.com/@fitodac/bibliotecas-y-frameworks-de-node-js-que-mejorar%C3%A1n-tu-desarrollo-web-728ab8034ce5>
- <https://medium.com/@diego.coder/autenticaci%C3%B3n-en-node-js-con-json-web-tokens-y-express-ed9d90c5b579>
- [https://significadosweb.com/concepto-de-protocolo-tcp-ip-que-es-definicion/?utm\\_source=chatgpt.com](https://significadosweb.com/concepto-de-protocolo-tcp-ip-que-es-definicion/?utm_source=chatgpt.com)
- [https://elblogdelprogramador.com/posts/enviar-correos-electronicos-desde-tu-servidor-nodejs-con-nodemailer/?utm\\_source=chatgpt.com#gsc.tab=0](https://elblogdelprogramador.com/posts/enviar-correos-electronicos-desde-tu-servidor-nodejs-con-nodemailer/?utm_source=chatgpt.com#gsc.tab=0)
- <https://www.digitalocean.com/community/tutorials/how-to-use-ejs-to-template-your-node-application-es>
- <https://es.react.dev/blog/2022/03/29/react-v18>
- <https://nextjs.org/>