



I.E.S. "ÁGORA"

Dpto.: Informática

Ciclo Formativo de G. Superior: Desarrollo de  
Aplicaciones Multiplataforma



Autor: Luis Arjona Aguilar

Tutora: Alicia Cámara Casares

Cáceres, a 10 de Junio de 2025

## Índice

<b>Introducción .....</b>	<b>3</b>
Descripción .....	3
Justificación .....	3
<b>Propuesta de Proyecto .....</b>	<b>3</b>
<b>Manual Técnico .....</b>	<b>4</b>
Introducción .....	4
Arquitectura del Sistema .....	5
Fases de desarrollo .....	7
Análisis y diseño.....	7
Desarrollo .....	9
Pruebas .....	21
Despliegue .....	22
Futuras mejoras.....	23
<b>Manual del usuario.....</b>	<b>24</b>
Introducción .....	24
Requisitos e instalación .....	24
Tipos de usuario.....	26
Interfaz general y navegación .....	27
Funcionalidades del usuario general .....	28
<b>Alternativas.....</b>	<b>32</b>
<b>Conclusión.....</b>	<b>33</b>
<b>Referencias bibliográficas .....</b>	<b>33</b>

## 1. Introducción

### Descripción

**F1GameInfo** es una **aplicación web** desarrollada como Proyecto de Fin de Ciclo del Grado Superior de Aplicaciones Multiplataforma. Haciendo uso de **diversas tecnologías** como **MySQL** para la base de datos, **Spring Boot** para el Backend y **Angular** para el Frontend.

La aplicación ofrece al usuario una experiencia de uso que incluye poder ver información sobre pilotos y circuitos de la temporada 2025 de la Formula 1, participar en un juego de compra de cartas de pilotos y circuitos, crear su propia carta personalizada, visualizar las cartas del resto de usuarios, y competir en una tabla global con el resto de usuarios por ver quien ha coleccionado más cartas, además de poder ver también las colecciones del resto de usuarios.

### Justificación

La idea del Proyecto Fin de Ciclo **surge de mi interés por la Formula 1**, tenía claro desde el principio que quería hacer algo relacionado con el tema, pero sin entrar en nada más específico. Finalmente, las funcionalidades que he elegido relacionadas con la Formula 1, han sido simplemente porque las he visto mas factibles y realistas, evitando intentar hacer algo irrealista y que no lo consiga a tiempo.

Respecto a las tecnologías usadas, he elegido Spring Boot debido a que Java es el lenguaje de programación con el que mas cómodo me siento y además me gusta, por lo que la elección de un Framework Java para el Backend estaba clara. En concreto es Spring Boot debido a que durante las prácticas es la tecnología que más hemos tocado.

He elegido Angular para el desarrollo web debido a que es el Framework para el Frontend del que más conocimientos tengo, gracias a haberlo usado tanto en clase como en las prácticas. La elección de MySQL como gestor de base de datos viene dada primariamente por ser SQL y además respecto a otros gestores SQL por la facilidad que ofrece para desplegarlo.

## 2. Propuesta de Proyecto

CICLO FORMATIVO: Desarrollo de Aplicaciones Multiplataforma

TÍTULO DEL PROYECTO: F1GameInfo

DESCRIPCIÓN GENERAL:

API REST conectada a una base de datos y consumida por Angular que permite crear tu usuario y ver información acerca de Formula 1 y que principalmente tiene un juego que

permite al usuario comprar cartas de Pilotos y Circuitos de F1.

ALUMNO: Luis Arjona Aguilar

TELÉFONO: 693800044

E-MAIL: larjonaa01@educarex.es

Fecha presentación:

OBJETIVO: Diseño de un Backend compuesto por una base de datos y un servicio API REST el cual consumirá un Frontend.

JUSTIFICACIÓN: Desarrollar una API REST conectada a una BBDD que siendo consumida por Angular permita al usuario visualizar información acerca de la Fórmula 1 y jugar a un juego de colección de cartas de Pilotos y Circuitos.

ASPECTOS PRINCIPALES QUE SE PRETENDEN ABORDAR:

Diseño de una base de datos, acceso a datos, diseño de una API REST, diseño de una interfaz de usuario.

MEDIOS QUE SE UTILIZAN: MySQL, Spring Boot con Java, y Angular.

ÁREAS DE TRABAJO Y OTROS ELEMENTOS ESTABLECIDOS POR EL EQUIPO DOCENTE DEL CICLO FORMATIVO A EFECTOS DE VALORACIÓN DE LA PROPUESTA:

### 3. Manual Técnico

#### 1. Introducción

El objetivo de la aplicación es proporcionar al usuario una experiencia que le permita **ver información actual de los pilotos y circuitos de la temporada 2025** de la Formula 1, **comprar cartas** de sus pilotos y circuitos favoritos para luego poder **competir en una tabla general** por ver que usuario consigue coleccionar más cartas. Además, también pueden **crear su propia carta** y que todos los usuarios **puedan ver las cartas** personalizadas de los demás. Aparte de poder ver las cartas personalizadas del resto de usuarios también pueden **ver las colecciones de los demás**.

El público objetivo de la aplicación son usuarios normales, con interés en el deporte de motor y sobre todo el gran circo de la Formula 1.

Las **tecnologías utilizadas** para este proyecto son:

1. Spring Boot con Java.
2. Angular con Typescript, HTML, y CSS.
3. MySQL.
4. Railway y Vercel.
5. Github.

## 2. Arquitectura del Sistema

### Frontend:

- Angular: es un **Framework web** de código abierto creado por Google. El gran punto fuerte respecto a su competencia como React o Vue es que **es una solución completa**, te ofrece todo lo necesario, por lo que, aunque existen herramientas externas esas funcionalidades ya las cubre Angular. Cabe destacar que no todo son ventajas, si Angular tiene algo malo es su **curva de aprendizaje, mucho mayor** que la de su competencia. En el proyecto si se ha hecho uso de librerías externas como:
  - PrimeNG: es una biblioteca de componentes para la interfaz de usuario. Específicamente en el proyecto se ha implementado para mostrar las validaciones de las contraseñas, filtros para bloquear ciertos caracteres y espacios en algunos inputs, iconos a lo largo de toda la aplicación, o botones entre otros.
  - SweetAlert: es una biblioteca para mostrar ventanas emergentes. Específicamente en el proyecto se ha implementado en el momento de registrar una cuenta, iniciar sesión, y crear tu carta personalizada, tanto para mostrar que ha funcionado correctamente o el error que ha sucedido.
- Typescript: se puede denominar la versión avanzada de JavaScript, desarrollado por Microsoft. Su éxito radica en que **mejora a JavaScript** con múltiples funcionalidades extra, pero destacando sobre todo el **tipado estático**, lo que mejora la seguridad y calidad de código, detectando errores antes de la ejecución y no causándolos durante la misma. Typescript es culpable en gran medida de lo bueno y malo que tiene Angular como su curva de aprendizaje, que es mayor a la de JavaScript.
- HTML: es el **lenguaje de marca estándar** en el desarrollo web, está conformado por etiquetas que describen la estructura de la página web, y destaca por su gran facilidad. Pero por si solo se le puede denominar como “inútil” ya que carece de estilos e interactividad, el corazón de una página web.
- CSS: lenguaje de estilos utilizado para **definir la apariencia del contenido HTML** de una página web, resolviendo esa carencia de estilos. Destaca por su flexibilidad e infinidad de posibilidades para estilizar una página web. Aunque para aplicar estilos básicos parece inofensivo a nivel avanzado su curva de aprendizaje se puede volver muy grande. Cabe destacar que es en gran parte el limitante en la tabla de compatibilidad detallada en el manual del usuario, muchas características solo funcionan en navegadores muy actuales.
- Visual Studio Code: entorno de desarrollo, diseñado por Microsoft. Destaca por **su gran compatibilidad**, no solo a nivel de Sistemas Operativos, también a nivel de lenguajes de programación, **dispone de extensiones para cualquier lenguaje de programación** medianamente conocido.



Backend:

- Spring Boot: es un framework de código abierto para **desarrollo de APIs** y páginas web. Es una extensión de Spring Framework pensada para facilitar la configuración inicial de la aplicación. Es un framework **muy potente**, pero por lo tanto **pesado**. Cuenta con un **gran ecosistema de funcionalidades** como Spring Data, Spring Security o Spring Cloud, que, si bien a la larga son muy útiles, para un desarrollador nuevo al no estar familiarizado con las peculiaridades de Spring supone una **gran curva de aprendizaje**.
- Java: lenguaje de programación **orientado a objetos**, propiedad de Oracle. Destaca por la famosa JVM la cual hace que los programas Java puedan ejecutarse de forma nativa en cualquier plataforma gracias a que ejecuta el bytecode generado por el compilador, el cual es un formato independiente de la plataforma. Si bien la JVM está excepcionalmente optimizada al límite de sus posibilidades sigue haciendo a Java un **lenguaje más lento y pesado** que aquellos compilados directamente a código máquina.
- MySQL: sistema de gestión de **bases de datos relacionales**, también propiedad de Oracle. Utiliza el lenguaje de consulta estructurado (**SQL**) que organiza los datos en tablas compuestas de filas y columnas. Se caracteriza por su facilidad de uso al contar con herramientas gráficas, buen rendimiento y compatibilidad multiplataforma. El esquema fijo en tablas de SQL ofrece ventajas, pero en temas de consultas complejas se queda corto comparado con las bases de datos NoSQL gracias a su flexibilidad. Además, comparado con otros sistemas de gestión relacionales carece de funciones avanzadas, mientras que en MySQL para la generación de claves automáticas hay que conformarse con el AUTO\_INCREMENT en otros sistemas como PostgreSQL existen las secuencias, que aumentan la personalización al definir saltos específicos, valores mínimos y máximos, reinicios automáticos y un mejor rendimiento.
- JWT: es un estándar que define una **forma segura de transmitir información** haciendo uso de tokens, que están firmados digitalmente para garantizar su integridad y autenticidad, y contienen los datos en **forma de JSON**. Los tokens contienen toda la información del usuario como correo, nombre de usuario o sus permisos lo que facilita el flujo de la aplicación, y gracias a que son compactos se pueden transmitir fácilmente. Si cabe destacar algo es que los tokens JWT no están cifrados, tan solo firmados para verificar su integridad, lo que significa que si cualquier persona se hace con un token que no sea suyo **puede ver todo su contenido** como el correo o los permisos del usuario. Además, estos tokens son válidos hasta que llegue su fecha de expiración, para conseguir revocarlos sería necesario almacenarlos en el servidor por ejemplo en una base de datos, lo cual desvirtúa en gran parte la naturaleza y el propósito de JWT.

- Eclipse: es un entorno de desarrollo, principalmente pensado para el desarrollo de aplicaciones en C, C++ y Java. Cuenta con una **gran comunidad**, gran cantidad de extensiones y compatibilidad multiplataforma. El gran obstáculo de Eclipse es su consumo de recursos y su interfaz más tradicional. La elección de Eclipse por ejemplo sobre otros entornos como IntelliJ ha sido por su facilidad para integrarlo con Spring Boot, con una extensión prácticamente “Plug & Play”, mientras que IntelliJ presentaba más problemas.

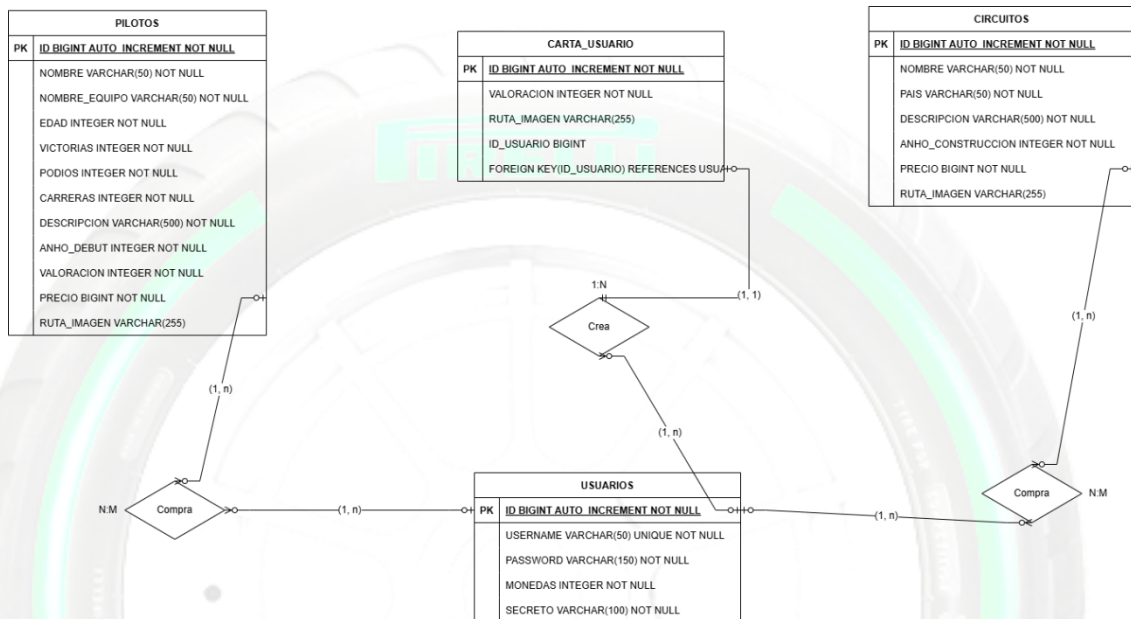
### 3. Fases de desarrollo

#### Análisis y diseño:

- La base de datos está compuesta de 6 tablas
  - o Usuarios: **representa los usuarios** de la aplicación. Su Primary Key es el id, un bigint auto\_increment. Además, contiene el email, número de monedas, secreto (código para obtener los OTP), y la contraseña la cual está cifrada para aumentar la seguridad.
  - o Pilotos: **representan los pilotos**. Su Primary Key es el id, un bigint auto\_increment. Contiene además su nombre, nombre de su equipo, edad, victorias, podios, carreras, una descripción corta, su año de debut, valoración, precio y la ruta donde se almacena su imagen.
  - o Circuitos: **representan los circuitos**. Su Primary Key es el id, un bigint auto\_increment. Contiene además su nombre, país, una descripción corta, su año de construcción, precio y la ruta donde se almacena su imagen.
  - o Carta\_Usuario: **representan las cartas customizadas** que han creado los usuarios. Su Primary Key es el id, un bigint auto\_increment. Contiene además su valoración, ruta donde se almacena su imagen y el id del usuario al que pertenece.
  - o Usuario\_Pilotos: **representan las cartas de los pilotos** que ha comprado el usuario. Su Primary Key son la combinación del id del piloto y el id del usuario, siendo ambos todo el contenido de la tabla.
  - o Usuario\_Circuitos: **representan las cartas de los circuitos** que ha comprado el usuario. Su Primary Key son la combinación del id del circuito y el id del usuario, siendo ambos todo el contenido de la tabla.
- Las relaciones que conforman la base de datos son las siguientes:
  - o Usuarios y Carta\_Usuario: un Usuario puede tener de una a muchas Carta\_Usuario (1, n) mientras que una Carta\_Usuario pertenece solo a un Usuario (1, 1) lo que conforma una relación uno a muchos 1:N.
  - o Usuarios y Pilotos: un usuario puede comprar de una a muchos Pilotos (1, n) y un piloto puede ser propiedad de uno a muchos Usuarios (1, n) lo que conforma una relación muchos a muchos N:M que provoca la

creación de una tercera tabla Usuario\_Pilotos para representar la relación correctamente.

- Usuarios y Circuitos: un usuario puede comprar de uno a muchos Circuitos (1, n) y un circuito puede ser propiedad de uno a muchos Usuarios (1, n) lo que conforma una relación muchos a muchos N:M que provoca la creación de una tercera tabla Usuario\_Circuitos para representar la relación correctamente.



- Ilustración 1. Diagrama E/R de la base de datos.

### Diagrama de uso de la aplicación:



- Ilustración 2. Diagrama de uso de la aplicación.



Desarrollo:

Los **endpoints disponibles** son los siguientes

**AUTENTICACIÓN**

URL	/autenticación/login	
Método	POST	
Descripción	Permite el inicio de sesión de un usuario	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	username	Correo del usuario
	password	Contraseña
	otp	Código OTP
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	401	OTP incorrecto
	403	Credenciales incorrectas
Cuerpo de respuesta	{ "token": "string" }	
Autenticación	Código OTP (Autenticación de 2 Factores)	

Tabla 1. Endpoint de Inicio de Sesión

URL	/autenticación/registro	
Método	POST	
Descripción	Permite la creación de una nueva cuenta	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	{ "username": "string", "password": "string" }	
Códigos de respuesta	200	Correcto
	404	Usuario ya existente
Cuerpo de respuesta	{ "qr": "String", "secreto": "String" }	
Autenticación	<del>No aplica</del>	

Tabla 2. Endpoint de Registro

## CARTAS

URL	/cartas/uploads/{id}	
Método	POST	
Descripción	Permite crear una nueva carta	
Variable de ruta	Id: Identificador del usuario que crea la carta	
Parámetros de consulta	imagen	Archivo con la imagen
	valoracion	Valoración (1-99) de la carta
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	400	El usuario ya tiene una carta
Cuerpo de respuesta	<del>No aplica</del>	
Autenticación	Token JWT	

Tabla 3. Endpoint para crear una carta personalizada

URL	/cartas/{id}	
Método	GET	
Descripción	Obtiene las cartas compradas de un usuario	
Variable de ruta	Id: Identificador del usuario	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	404	El usuario no existe
Cuerpo de respuesta	<pre>{   "pilotos": [     {       "id": int,       "nombre": "string",       "nombreEquipo": "string",       "edad": int,       "victorias": int,       "podios": int,       "carreras": int,       "descripcion": "string",       "anhoDebut": int,       "valoracion": int,       "precio": int,       "rutaImagen": "string"     }   ],   "circuitos": [     {       "id": int,       "nombre": "string",       "pais": "string", </pre>	

	<pre> "descripcion": "string", "anhoConstruccion": int, "precio": int, "rutaImagen": "string" } ] } </pre>
Autenticación	Token JWT

Tabla 4. Endpoint para obtener las cartas compradas de un usuario

URL	/cartas/{id}	
Método	PATCH	
Descripción	Actualiza las cartas y/o monedas de un usuario	
Variable de ruta	Id: Identificador del usuario	
Parámetros de consulta	monedas	Monedas del usuario
	idPiloto	Id del piloto (si aplica)
	idCircuito	Id del circuito (si aplica)
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	404	El usuario no existe
Cuerpo de respuesta	<del>No aplica</del>	
Autenticación	Token JWT	

Tabla 5. Endpoint para actualizar las cartas y/o monedas de un usuario

URL	/cartas/usuarios	
Método	GET	
Descripción	Obtiene las cartas personalizadas de todos	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	500	Error inesperado
Cuerpo de respuesta	<pre> [ { "valoracion": int, "rutaImagen": "string", "username": "string" } ] </pre>	
Autenticación	Token JWT	

Tabla 6. Endpoint para obtener las cartas personalizadas de todos los usuarios

URL	/cartas/usuarios/{id}	
Método	GET	
Descripción	Obtiene la carta personalizada de un solo usuario	
Variable de ruta	Id: Identificador del usuario	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	404	El usuario no existe
Cuerpo de respuesta	<pre>{   "valoracion": int,   "rutaImagen": "string",   "username": "string" }</pre>	
Autenticación	Token JWT	

Tabla 7. Endpoint para obtener la carta personalizada de un usuario

## USUARIOS

URL	/usuarios/{id}/monedas	
Método	GET	
Descripción	Obtiene las monedas de un usuario	
Variable de ruta	Id: Identificador del usuario	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	404	El usuario no existe
Cuerpo de respuesta	<pre>{   "monedas": int }</pre>	
Autenticación	Token JWT	

Tabla 8. Endpoint para obtener las monedas de un usuario

URL	/usuarios/ranking	
Método	GET	
Descripción	Obtiene el ranking de los usuarios	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	500	Error del servidor
Cuerpo de respuesta	<pre>[   {     "username": "string",     "cantidadPilotos": int,     "cantidadCircuitos": int, </pre>	

	<pre> "cantidadTotal": int, "pilotos": [   {     "id": int,     "nombre": "string",     "nombreEquipo": "string",     "edad": int,     "victorias": int,     "podios": int,     "carreras": int,     "descripcion": "string",     "anhoDebut": int,     "valoracion": int,     "precio": int,     "rutalmagen": "string"   } ], "circuitos": [   {     "id": int,     "nombre": "string",     "pais": "string",     "descripcion": "string",     "anhoConstruccion": int,     "precio": int,     "rutalmagen": "string"   } ] ] </pre>
Autenticación	Token JWT

Tabla 9. Endpoint para obtener el ranking de los usuarios

## PILOTOS

URL	/pilotos	
Método	GET	
Descripción	Obtiene todos los pilotos	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	500	Error del servidor
Cuerpo de respuesta	<pre> [   { </pre>	



	<pre> "id": int, "nombre": "string", "nombreEquipo": "string", "edad": int, "victorias": int, "podios": int, "carreras": int, "descripcion": "string", "anhoDebut": int, "valoracion": int, "precio": int, "rutaImagen": "string" } ] </pre>
Autenticación	Token JWT

Tabla 10. Endpoint para obtener todos los pilotos

## CIRCUITOS

URL	/circuitos	
Método	GET	
Descripción	Obtiene todos los circuitos	
Variable de ruta	<del>No aplica</del>	
Parámetros de consulta	<del>No aplica</del>	
Cuerpo de la solicitud	<del>No aplica</del>	
Códigos de respuesta	200	Correcto
	500	Error del servidor
Cuerpo de respuesta	<pre> [ {   "id": int,   "nombre": "string",   "pais": "string",   "descripcion": "string",   "anhoConstruccion": int,   "precio": int,   "rutaImagen": "string" } ] </pre>	
Autenticación	Token JWT	

Tabla 11. Endpoint para obtener todos los circuitos

La aplicación sigue una **arquitectura general de cliente-servidor** y además **desacoplada**, ya que por una parte está el Frontend desarrollado como una **SPA** mediante Angular, y por otro lado tenemos el Backend desarrollado como una **API REST** mediante Spring Boot, las cuales se comunican constantemente para el funcionamiento de la aplicación. Si desglosamos la arquitectura de la aplicación diferenciando entre ambas capas:

- Frontend: Angular sigue el **patrón MVVM**:
  - Model: interfaces y clases que **representan los datos** para interactuar con el Backend, ya sean datos recibidos o mandados.
  - View: formado por todo **lo que el usuario ve**, en este caso son el HTML, CSS, e imágenes.
  - ViewModel: **conecta la vista con los datos** de la aplicación, es el encargado de manejar toda la lógica de la aplicación, en este caso son los archivos Typescript.
- Backend: Spring Boot sigue el **patrón MVC**:
  - Model: en este caso **representa los datos** como por ejemplo DTOs o las entidades recogidas de base de datos mediante un ORM, pero **también toda la lógica de negocio** dentro de los servicios.
  - View: en este caso debido a Angular la vista no se genera desde Spring Boot, pero en caso de que no fuese así el servidor se encargaría de generar el HTML con Thymeleaf o JSP.
  - Controller: es el encargado de atender las peticiones del Frontend, recibe datos del mismo y llama a los servicios para que los procesen correctamente y luego se encarga de devolver la respuesta correspondiente y los datos necesarios al Frontend, ya sea correcto o no. El controlador simplemente **actúa como un intermediario**, solo se encarga de llamar al servicio con la información que ha recibido y de interpretar el resultado del servicio para comunicarlo al Frontend.

La **organización de paquetes** es la siguiente:

```
com.arjona.f1gameinfo/
├── F1GameInfoApplication.java    # Clase principal de arranque de la aplicación
├── business/
│   ├── model/                  # Modelo y DTOs
│   │   ├── CartaCompradaDTO.java
│   │   ├── CartaUsuario.java
│   │   └── CartaUsuarioDTO.java
```

```

| | └─ Circuito.java
| | └─ Piloto.java
| | └─ RankingDTO.java
| | └─ UsuarioDTO.java
| |
| └─ repositories/          # Interfaces de acceso a datos (JPA)
| | └─ CartaUsuarioRepository.java
| | └─ CircuitoRepository.java
| | └─ PilotoRepository.java
| |
| └─ services/              # Interfaces de servicios de negocio
| | └─ CartaServices.java
| | └─ CircuitoServices.java
| | └─ PilotoServices.java
| | └─ UsuarioNoAuthServices.java
| |
| └─ services/impl/         # Implementaciones de los servicios
| | └─ CartaServicesImpl.java
| | └─ CircuitoServicesImpl.java
| | └─ PilotoServicesImpl.java
| | └─ UsuarioNoAuthServicesImpl.java
└─ presentation/
| └─ WebConfig.java          # Configuración general del contexto web
| |
| └─ config/                 # Gestión de errores y excepciones personalizadas
| | └─ GestorExcepciones.java
| | └─ HttpCustomError.java
| | └─ StatusException.java

```

```

| |
| └─ restcontrollers/      # Controladores REST que exponen la API
|   └─ CartaController.java
|   └─ CircuitoController.java
|   └─ PilotoController.java
|   └─ UsuarioNoAuthController.java
| └─ security/
|   └─ FiltroAutenticacionJWT.java # Filtro de autenticación con JWT
|   └─ FiltroAutenticacionOTP.java # Filtro de autenticación con OTP
|   └─ SeguridadConfig.java      # Configuración de seguridad (Spring Security)
|   └─ UsuarioDetails.java       # Implementación de UserDetails
|   └─ UsuarioDetailsService.java # Servicio de carga de usuarios para autenticación
|   └─ UtilsJWT.java             # Utilidades para manejo de JWT
|   └─ UtilsOTP.java             # Utilidades para manejo de OTP
|
| └─ security/integration/
|   └─ model/                    # Entidad de usuario para autenticación
|     └─ Usuario.java
|
|   └─ repositories/            # Repositorio JPA para autenticación
|     └─ UsuarioRepository.java
|
|   └─ services/                # Servicio de autenticación
|     └─ UsuarioServices.java
|
|   └─ services/impl/           # Implementación del servicio de autenticación
|     └─ UsuarioServicesImpl.java
|
| └─ security/payloads/         # Clases para peticiones y respuestas de autenticación

```

```

| └─ JwtResponse.java
| └─ LoginRequest.java
| └─ RegisterRequest.java
| └─ RegisterResponse.java
└─ security/presentation/restcontrollers/
    └─ AutenticacionController.java # Controlador REST para login y registro

```

Un **fragmento de código** interesante de explicar en Angular sería el **registro del usuario**:

- Lo primero es establecer el booleano *cargando* a verdadero, para mostrarle al usuario que se está procesando el registro.
- Lo siguiente es llamar al servicio *this.usuServicio.registrarUsuario* (ver tercera imagen) para que realice el registro, recogiendo el correo y contraseña del usuario, previamente validadas.
- ¿Cómo se validan? (Ver segunda imagen) al correo se le establece una validación de email que comprueba que tenga un formato de correo real. La contraseña pasa por un validador propio que comprueba que contenga al menos una mayúscula, una minúscula, un número y un carácter especial. Si estas validaciones no se cumplen el usuario no puede pinchar en el botón de registrarse.
- Asumiendo que los datos son válidos se realiza la petición y nos suscribimos a ella *.subscribe*, aquí se distinguen dos casos:
  - Si todo va bien entrará en el apartado *next*, donde establecemos el atributo *cargando* a falso para que deje de mostrar que está procesando la petición y los booleanos *exito* y *registrado* a verdadero para indicar que todo ha ido bien. Luego hacemos uso de la librería SweetAlert (explicada dentro del apartado “Arquitectura del Sistema”) para mostrar al usuario que todo ha ido bien. Y como último paso recuperamos el qr y el secreto de la petición.
  - Si el Backend devuelve un error entrará en el apartado *error*, donde igualmente pondremos el atributo *cargando* a falso, pero haremos lo mismo con *exito* y *registrado* para mostrar que ha fallado. Establecemos un mensaje de error por defecto *errorMessage* que solo debería mostrarse si ha ocurrido algo inesperado. Comprobamos el código de error que nos ha devuelto, si es un 400 significa que el correo ya está registrado, si no comprobamos si el código es 0 y si el mensaje hace



referencia a conexión rechazada, que significaría que el servidor no está disponible o que el usuario no tiene conexión.

Finalmente con el mensaje de error elegido hacemos uso de la librería SweetAlert y mostramos una notificación de error al usuario para informar.

```
registrarUsuario() {
  this.cargando = true;
  this.usuServicio.registrarUsuario(
    this.form.get('email').value,
    this.form.get('pass').value
  ).subscribe({
    next: (data) => {
      this.cargando = false;
      this.exito = true;
      this.registrado = true;
      Swal.fire({
        title: "Perfecto",
        text: "Has creado tu cuenta",
        icon: "success",
        timer: 2000,
        timerProgressBar: true,
        showConfirmButton: false
      });
      this.qr = data.qr;
      this.secreto = data.secreto;
    },
    error: (error) => {
      this.cargando = false;
      this.exito = false;
      this.intentado = true;

      let errorMessage = "Ocurrió un error inesperado";

      if (error.status === 400) {
        errorMessage = "El correo ya está registrado";
      }
      else if (error.status === 0 || error.message?.includes('Connection refused')) {
        errorMessage = "No se puede establecer la conexión.";
      }

      Swal.fire({
        title: "Algo ha ido mal",
        text: errorMessage,
        icon: "error",
        confirmButtonColor: "#d71928",
        confirmButtonText: "Entendido"
      });
    }
  });
}
```

Ilustración 3. Código del registro del usuario en Angular.

```
constructor(private fb: FormBuilder, private usuServicio: UsuarioServicioService, private router: Router) {
  this.form = this.fb.group({
    email: ["", [Validators.required, Validators.email]],
    pass: ["", [Validators.required, this.validarPass.bind(this)]],
    confirm_pass: ["", [Validators.required]]
  });
}

validarPass(control: AbstractControl): { [key: string]: boolean } | null {
  const value = control.value;
  if (!value) return null;

  const hasUpperCase = /[A-Z]/.test(value);
  const hasLowerCase = /[a-z]/.test(value);
  const hasNumber = /[0-9]/.test(value);
  const hasSpecialChar = /[!@#$%^&*(),.?":{}|<>]/.test(value);
  const isValid = hasUpperCase && hasLowerCase && hasNumber && hasSpecialChar;

  return !isValid ? { passwordComplexity: true } : null;
}
```

Ilustración 4. Código de las validaciones en el registro del usuario en Angular.

```
registrarUsuario(email:string,pass:string): Observable<any>{
  return this.http.post('http://localhost:8080/autenticacion/registro', {"username": email, "password":pass});
}
```

Ilustración 5. Código de la llamada a la API en el registro del usuario en Angular.

Un **fragmento de código interesante** de explicar en **Spring Boot** sería el filtro de autenticación OTP, **la autenticación de 2 Factores** mediante el qr o secreto mencionados previamente:

- El primer paso (línea 32) es comprobar si la ruta de la petición corresponde con el inicio de sesión, caso contrario el filtro lo ignora y no hace nada.
- Lo siguiente (líneas 33 y 34) es recuperar el código otp y correo de la petición.
- Para verificar el código es necesario pasarlo de texto a entero (línea 37) y luego cargamos los datos del usuario (línea 38 y ver la segunda imagen) haciendo uso de la clase *UsuarioDetails* que pertenece al ecosistema de Spring Security.
- Si el usuario es nulo o el código no es válido (línea 39 y ver ilustración 8) entonces responde con un código 401 (línea 40) indicando que no es válido.
- Los dos pasos anteriores estaban dentro de un try-catch, ya que si al pasar el código de texto a entero salta una excepción eso significa que el formato del código no es válido (caracteres no numéricos), por lo que devuelve un código de error 400 (línea 44) indicando que el código otp no sigue un formato válido.
- Finalmente indicamos a la cadena de filtros (línea 48) que continúe con el resto de filtros de la petición o si no queda nada que termine de procesar la petición.

```

29     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
30         throws ServletException, IOException {
31
32         if (request.getRequestURI().equals("/autenticacion/login")) {
33             String otpCode = request.getParameter("otp");
34             String email = request.getParameter("username");
35             |
36             try {
37                 int otpCodeInt = Integer.parseInt(otpCode);
38                 UsuarioDetails usuario = (UsuarioDetails) usuarioDetailsService.loadUserByUsername(email);
39                 if (usuario == null || !utilsOTP.verifyCode(usuario.getSecreto(), otpCodeInt)) {
40                     response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Código OTP no válido");
41                     return;
42                 }
43             } catch (NumberFormatException e) {
44                 response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Formato de código OTP no válido");
45                 return;
46             }
47         }
48         filterChain.doFilter(request, response);
49     }
50

```

Ilustración 6. Código del filtro de la autenticación de 2 factores en Spring Boot.

```

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Usuario usuario = usuarioRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("Usuario: " + username + " no encontrado."));
    return new UsuarioDetails(usuario);
}

```

Ilustración 7. Código de la búsqueda del usuario en Spring Boot.

```

public boolean verifyCode(String secreto, int code) {
    return gAuth.authorize(secreto, code, new Date().getTime());
}

```

Ilustración 8. Código de la verificación del código OTP en la autenticación de 2 factores en Spring Boot.

Pruebas: para el testing se ha hecho uso de:

- JUnit: es un marco de **pruebas unitarias**. Permite la creación de métodos que **verifican el comportamiento del código** de la aplicación en diferentes escenarios, ya sea cuando todo va bien o cuando ocurren errores. Su mayor ventaja es la simplicidad.
- Mockito: es utilizado **para crear objetos simulados** en el testing. Digamos que quieres probar el comportamiento de los servicios de tu aplicación que dependen a su vez del repositorio, con Mockito lo que haces es simular el comportamiento del repositorio, y así verificas el comportamiento del servicio dependiendo de lo que reciba del repositorio, y, además, reduces las pruebas solo al servicio, teniendo que crear otras para el repositorio lo que **mejora la calidad** de las mismas.
- TestContainers: es una biblioteca de pruebas que permite hacer uso de **contenedores de Docker** a la hora de probar la aplicación. A la hora de hacer pruebas de los controladores Spring Boot intenta arrancar el contexto entero de la aplicación, lo que hace que intenten conectarse a la base de datos de la aplicación la cuál no está en marcha, con esta biblioteca se **crea un entorno aislado** que aloja la base de datos solo durante el tiempo de ejecución de las pruebas, **facilitando la configuración** de las mismas.

La cantidad del código de la aplicación que las pruebas cubren, es decir, la cantidad de **cobertura de la aplicación** se sitúa en un **84,1%**, por encima del recomendado 80%.

Element	Coverage	ed Instructions	ed Instructions	tal Instructions
> f1gameinfo	84,1 %	3.905	739	4.644

Ilustración 9. Porcentaje de cobertura de las pruebas en toda la aplicación Spring Boot.

Con lo mas importante siendo los servicios y controladores de la aplicación que contienen prácticamente toda la lógica situandose en un 85,6% y 100% respectivamente, por encima del resto de la aplicación.

com.arjona.f1gameinfo.pi	100,0 %
> CartaController.java	100,0 %
> CircuitoController.java	100,0 %
> PilotoController.java	100,0 %
> UsuarioNoAuthContro	100,0 %

com.arjona.f1gameinfo.bi	85,6 %
> CartaServicesImpl.java	80,9 %
> CircuitoServicesImplja	100,0 %
> PilotoServicesImpl.java	100,0 %
> UsuarioNoAuthService	100,0 %

Ilustraciones 10 y 11. Porcentaje de cobertura de las pruebas en los controladores y servicios respectivamente.

#### 4. Despliegue

Para el **despliegue** de la aplicación se ha hecho uso de **dos plataformas distintas**, una para la base de datos y el Backend, y otra para el Frontend:

##### 1. Railway

- a. ¿Qué es?: es una **plataforma de despliegue** de aplicaciones que permite desarrollar, desplegar y escalar aplicaciones de manera sencilla, **especializada en bases de datos y Backend**. Aventura a toda su competencia debido a las facilidades que ofrece mediante Nixpacks. Nixpacks es una herramienta propia de Railway que **automatiza el proceso** de creación, empaquetamiento y despliegue de aplicaciones que permite desplegar una aplicación sin ninguna configuración avanzada, lo consigue mediante la creación de una imagen OCI, que integra con Docker y un plan de construcción de manera automática.
- b. Proceso de despliegue: el despliegue de la base de datos ha sido bastante rápido, simplemente consiste en **crear una imagen Docker** de MySQL desde la interfaz de Railway y crear tu esquema dentro.  
El despliegue del Backend tiene más dificultad, pero sigue sin ser demasiado complicado. Lo primero es dentro del mismo proyecto donde está la base de datos **crear un nuevo servicio**, en mi caso desde un repositorio de GitHub, seleccionar la carpeta dentro del mismo donde se encuentra ubicado mi proyecto Spring Boot y luego conectar la rama principal para que cada vez que subo un nuevo cambio a la misma redespliegue la aplicación automáticamente. Gracias a que está dentro del mismo proyecto puedes conectar el Backend a la base de datos **dentro de la red interna** que crea Railway, **reduciendo así costes** de uso mediante las variables de entorno que te da la plataforma como la url, contraseña y usuario de la base de datos.

##### 2. Vercel

- a. ¿Qué es?: es una **plataforma de desarrollo y despliegue** de aplicaciones web, que soporta múltiples Frameworks web incluyendo **Angular**, React o Vue.js.
- b. Proceso de despliegue: lo primero de todo es seleccionar el Framework, en este caso Angular. Al igual que el Backend, es necesario conectar el repositorio de Github, escoger la rama, y la carpeta donde está ubicado el proyecto. El mayor problema fue que al construir la aplicación no se creaba el index.html como punto de entrada para que Vercel pudiese desplegar la aplicación automáticamente, esto es debido a la arquitectura Angular Universal, para conseguir que crease el punto de entrada es necesario **pasar a la arquitectura SPA** (aplicación de una sola página) retocando los archivos de configuración dentro del proyecto de Angular.

## 5. Futuras mejoras

1. SonarQube: es una herramienta que permite **identificar errores y vulnerabilidades** en el código. Lo consigue mediante:
  - a. Cobertura de pruebas: evalúa la **cobertura de las pruebas** unitarias, para asegurar que el código esté correctamente probado.
  - b. Code Smells: son **problemas dentro del código** que pueden derivar en problemas de mantenibilidad y seguridad a lo largo del tiempo.
  - c. Deuda técnica: Es el **coste acumulado de malas prácticas** dentro del código incluyendo los Code Smells, falta de documentación, tecnología obsoleta. Representado en forma de tiempo estimado para resolverlo.
  - d. Valoración final: mediante lo explicado anteriormente SonarQube incluye en el reporte una valoración final que indica si el código **cumple el estándar necesario o no**.
2. Convertir el proyecto en **microservicios**: gracias a dividir el proyecto en múltiples servicios se vería beneficiado en:
  - a. Resistencia: en un monolito sin falla algo cae la aplicación entera, mientras que con microservicios si uno falla el resto **siguen activos**.
  - b. Escalabilidad: cada servicio puede desarrollarse de manera independiente según sus necesidades **sin depender del resto o verse limitado** en sus capacidades.
  - c. Detección de fallos: si suceden errores es más fácil **aislarlo y detectar su procedencia** para resolverlo con mayor rapidez.  
 Estos microservicios gracias a Spring Cloud estarían conectados a un Eureka Server, y mediante una Api Gateway también conectada **simplificaría toda la comunicación** en un solo punto. De esta manera el Frontend en vez de llamar a un microservicio distinto dependiendo de sus necesidades, solo tendría que llamar a la Api Gateway la cual se preocupa de identificar y llamar al microservicio correspondiente.
3. Tests automáticos: mediante herramientas como Selenium o Katalon es posible testear la página web de **manera automática**. Realizas una grabación interactuando en la página web, navegando por los menús, interactuando con botones, introduciendo datos y luego estas herramientas son capaces de reproducir exactamente lo grabado y **validar que el test se completa** sin ningún error.



#### 4. Manual del usuario

##### 1. Introducción

El objetivo de la aplicación es proporcionar al usuario una experiencia que le permita **ver información actual** de los pilotos y circuitos de la temporada 2025 de la Formula 1, **comprar cartas** de sus pilotos y circuitos favoritos para luego poder **competir en una tabla general** por ver que usuario consigue coleccionar más cartas. Además, también pueden **crear su propia carta** y que todos los usuarios **puedan ver las cartas personalizadas** de los demás. Aparte de poder ver las cartas personalizadas del resto de usuarios también pueden **ver las colecciones de los demás**.

El público objetivo de la aplicación son usuarios normales, **con interés en el deporte de motor** y sobre todo el gran circo de la Formula 1.

La aplicación se puede usar siempre que cumpla los requisitos mínimos especificados más adelante en ordenadores de sobremesa, ordenadores portátiles, tabletas y teléfono móvil.

##### 2. Requisitos e instalación

###### Requisitos mínimos y recomendados:

Las versiones compatibles de los navegadores con todas las funcionalidades de la página web son los siguientes:

Navegador	Versiones soportadas	Fechas de salida
Google Chrome	94-138	21/09/2021 – 17/04/2025
Microsoft Edge	94-135	23/09/2021 – 03/04/2025
Apple Safari	18-18.5	18/09/2024 – 14/04/2025
Mozilla Firefox	103-140	26/07/2022 – 15/04/2025
Opera	80-117	14/10/2021 – 13/02/2025

Tabla 12. Navegadores compatibles.

La compatibilidad de los navegadores está basada en la compatibilidad de Angular 19, los distintos estilos y animaciones CSS usadas, y la reproducción de imágenes de formato *avif*. La tabla solo detalla los navegadores más usados globalmente, pero por regla general cualquier otro navegador que cuente con una actualización de 2024 hacia delante debería funcionar correctamente.

Los requerimientos mínimos de Sistema Operativo son los siguientes:

Sistema Operativo	Versión mínima
Windows	Windows 10
macOS	macOS Big Sur 11
Linux	Ubuntu 18.04
Android	Android 8.0 Oreo
iOS	iOS 18.0

Tabla 13. Sistemas Operativos compatibles.

La compatibilidad de los Sistemas Operativos está basada en los navegadores compatibles con la página web.

- Es recomendable usar la página web en un dispositivo de resolución de **1920x1080** para conseguir la mejor experiencia del usuario, aun así, la página web está diseñada para ser responsiva y proporcionar una buena experiencia del usuario con pantallas de menor resolución como las de tabletas o teléfonos móviles.
- El ancho de banda mínimo para usar la página web es de 1 Mbps y una latencia de ~200ms, siendo recomendable contar con al menos **5 Mbps** y una latencia de **~100ms** para poder disfrutar de una experiencia más fluida dentro de la página web.

#### Instalación:

- Desde [aquí](#) puedes acceder directamente a la página web o escribiendo manualmente la dirección `f1gameinfo.vercel.app` en tu navegador.
- Lo primero de todo será instalar una aplicación de autenticación en tu teléfono móvil como *Microsoft Authenticator*, *Google Authenticator* o *Authy*. A través de un ordenador también es posible instalar en Google Chrome la extensión de [Autenticador](#).
- Una vez dentro de la página web lo primero que veremos será la pantalla de Iniciar Sesión, podemos pinchar abajo en “¿Necesitas crear una cuenta?” para ir al registro.
- A la hora de crear una cuenta además de proporcionar un correo electrónico con un formato válido es necesario introducir una contraseña que cumpla los siguientes requisitos con el fin de **reforzar la seguridad**:
  - Mínimo de 8 caracteres
  - Mínimo 1 mayúscula
  - Mínimo 1 minúscula
  - Mínimo 1 número
  - Mínimo 1 carácter especial

- Al pulsar el botón de “Registrarse” aparecerá una pantalla nueva, con un QR y un secreto. Su funcionalidad es proporcionar una capa de seguridad adicional mediante la autenticación de 2 factores (2FA). Con la aplicación o extensión de autenticación instalada puedes escanear el QR con la cámara o en caso de no disponer de cámara puedes introducir el secreto manualmente junto a la dirección de correo electrónico asociada.
- Una vez escaneado el QR la aplicación generará un código nuevo de 6 dígitos cada 30 segundos, al iniciar sesión además de introducir tu dirección de correo electrónico y contraseña será necesario introducir el código generado para verificar tu identidad y poder acceder al resto de la página web.

### 3. Tipos de usuario

La aplicación cuenta con un usuario general que puede realizar las siguientes funciones:

1. **Visualizar información detallada** sobre las estadísticas de los pilotos de la parrilla de 2025 de la Formula 1 y también sobre los circuitos pertenecientes al calendario de 2025 de la Formula 1. Pudiendo filtrar los pilotos según carreras disputadas y victorias, y circuitos según más modernos o antiguos.
2. **Obtener monedas** para después poder **comprar con esas monedas cartas** de esos mismos pilotos y circuitos cuyo precio está basado en la valoración del piloto o calidad del circuito y así coleccionarlas.
3. **Crear una carta propia** y personalizada, con una imagen a su elección y la valoración que desee.
4. **Visualizar las cartas personalizadas** que han creado el resto de usuarios dentro de la aplicación.
5. **Competir en una tabla global** por ver quien es el usuario que más cartas ha conseguido coleccionar, pudiendo filtrar según el número de pilotos, circuitos o el total de cartas coleccionadas.
6. **Visualizar las colecciones** del resto de usuarios con las cartas que han comprado mediante el juego.

#### 4. Interfaz general y navegación

La aplicación está dividida en 7 componentes distintos cuya navegación consiste en:

1. Inicio de sesión: el usuario puede navegar al registro haciendo clic en el hipervínculo donde pone “¿Necesitas crear una cuenta?”

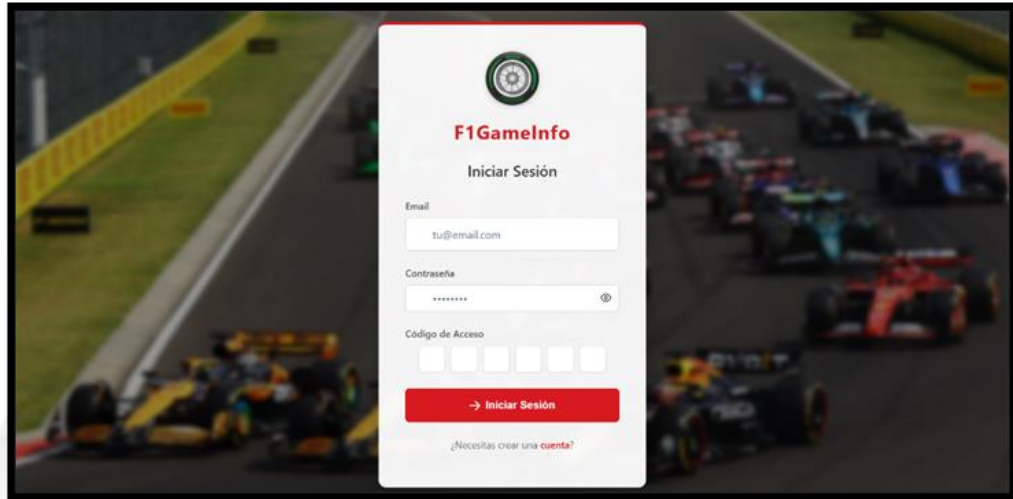


Ilustración 12. Interfaz del inicio de sesión y navegación al registro.

2. Registro: una vez registrada una nueva cuenta, además del QR y secreto mencionados anteriormente aparecerá un botón para navegar de vuelta al inicio de sesión.

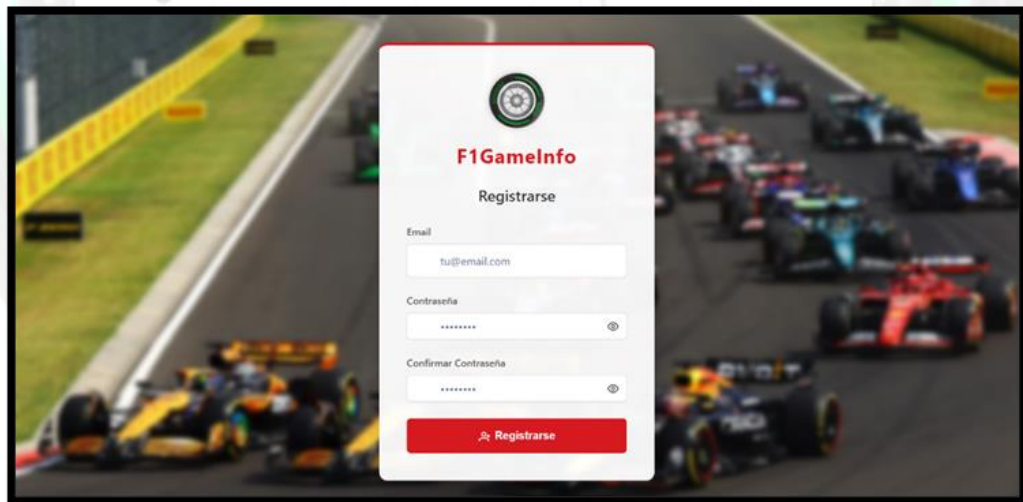


Ilustración 13. Interfaz del registro.

- Al iniciar sesión aparecerá el siguiente menú de navegación, que está presente en toda la aplicación y mediante el cual el usuario puede navegar a todas las distintas funcionalidades de la aplicación, también el menú incluye la opción de cerrar sesión. El lugar de la aplicación donde se encuentra el usuario estará en todo momento resaltado en color amarillo.

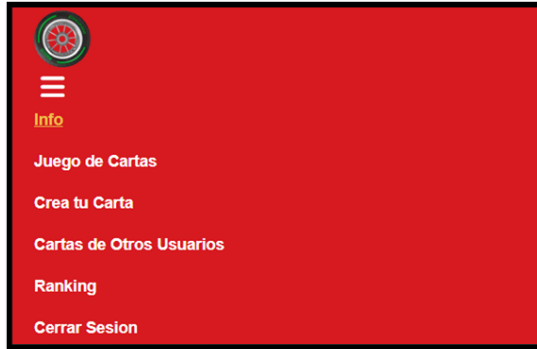


Ilustración 14. Menú de navegación en pantallas pequeñas.



Ilustración 15. Menú de navegación en pantallas grandes.

## 5. Funcionalidades del usuario general

- Visualizar información detallada** sobre las estadísticas de **los pilotos** pertenecientes a la parrilla de 2025 de la Formula 1. Dispone de dos filtros, el primero según la cantidad de carreras disputadas, y el segundo según la cantidad de victorias conseguidas, por defecto no aplica ningún filtro.
- Visualizar información detallada** sobre **los circuitos** pertenecientes al calendario de 2025 de la Formula 1. Dispone de la opción de filtrarlos según más modernos o más antiguos, por defecto no aplica ningún filtro.

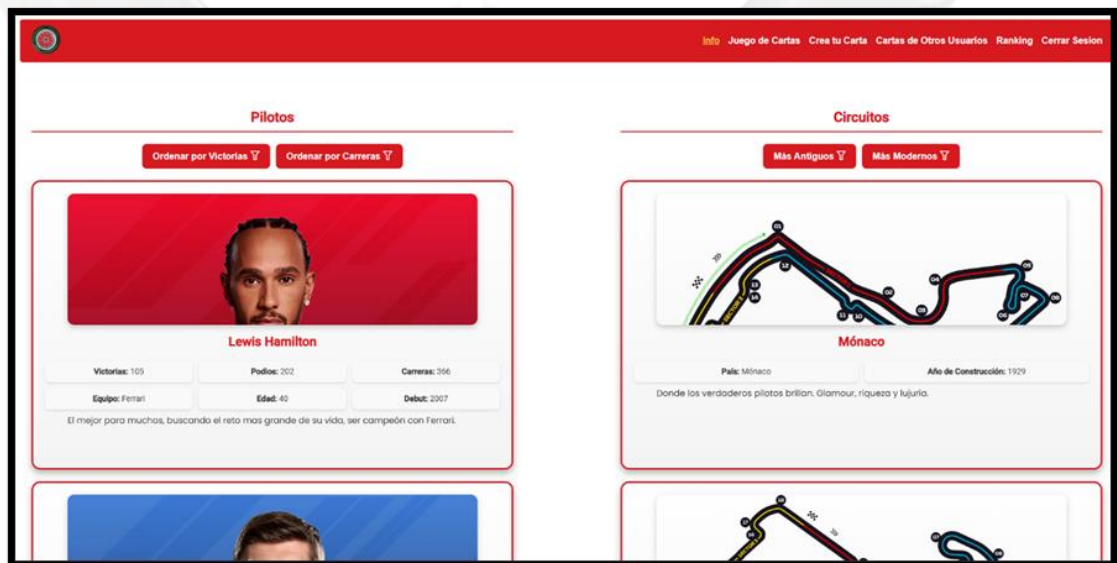


Ilustración 16. Pantalla de información de los pilotos y circuitos.



3. **Coleccionar cartas** de pilotos y circuitos de la Formula 1. El usuario **consigue monedas** pulsando en el botón situado justo debajo de **“Comprar Cartas”**. Si tiene monedas suficientes para **comprar la carta** del piloto o circuito, puede coleccionarla. El usuario dispone de un contador de sus monedas, y las cartas tienen indicadas claramente el número de monedas que cuesta comprarla. Arriba cuenta con dos barras de progreso, una para los pilotos y otra para los circuitos que indican el progreso en del 0% al 100%. Arriba a la derecha de cada piloto está indicada su valoración, el factor principal en el que se basa el precio. Haciendo click en el botón **“Mi colección”** puede ver **su colección**. Si un piloto no está en su colección el botón de comprar aparecerá con un carrito, si es de su colección mostrará un tick verde.

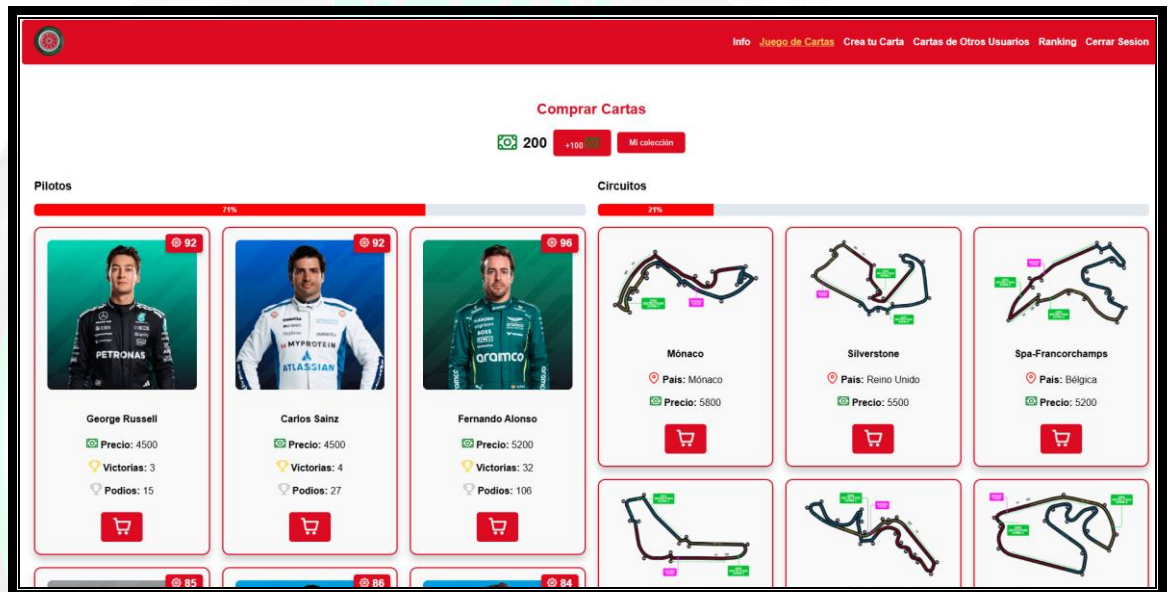


Ilustración 17. Pantalla del juego de colección de cartas.

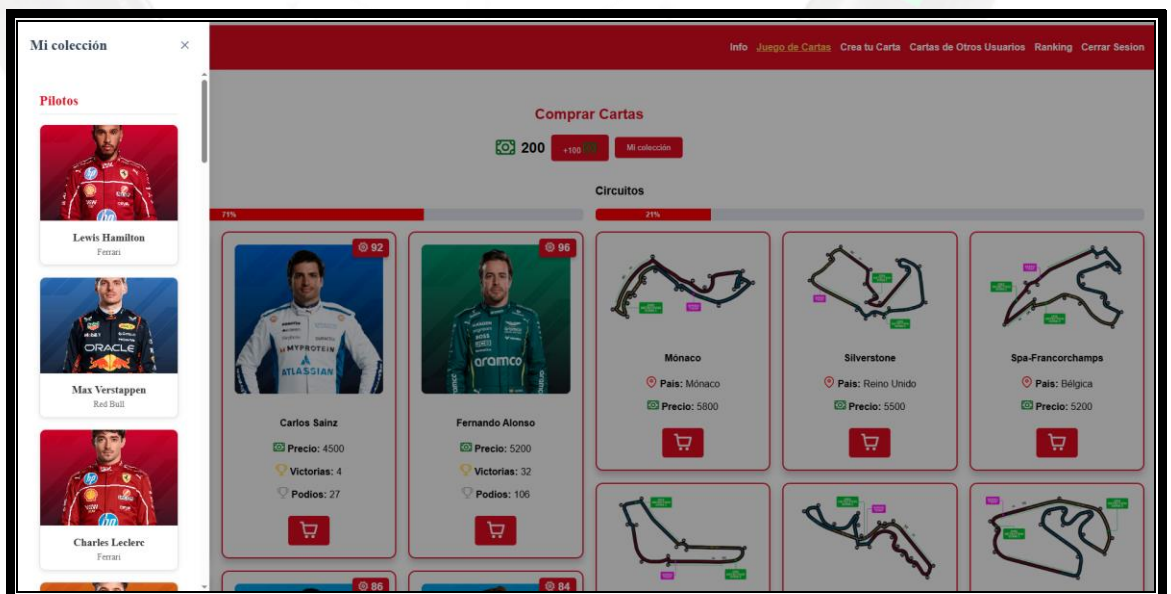


Ilustración 18. Pantalla con tu propia colección de cartas.

4. **Crear una carta personalizada y propia** del usuario, el usuario puede seleccionar la imagen que más le guste, ya sea una suya propia o de algo que le guste, le parezca gracioso, y luego seleccionar la valoración que tendrá su carta que debe estar comprendida de 1 a 99.

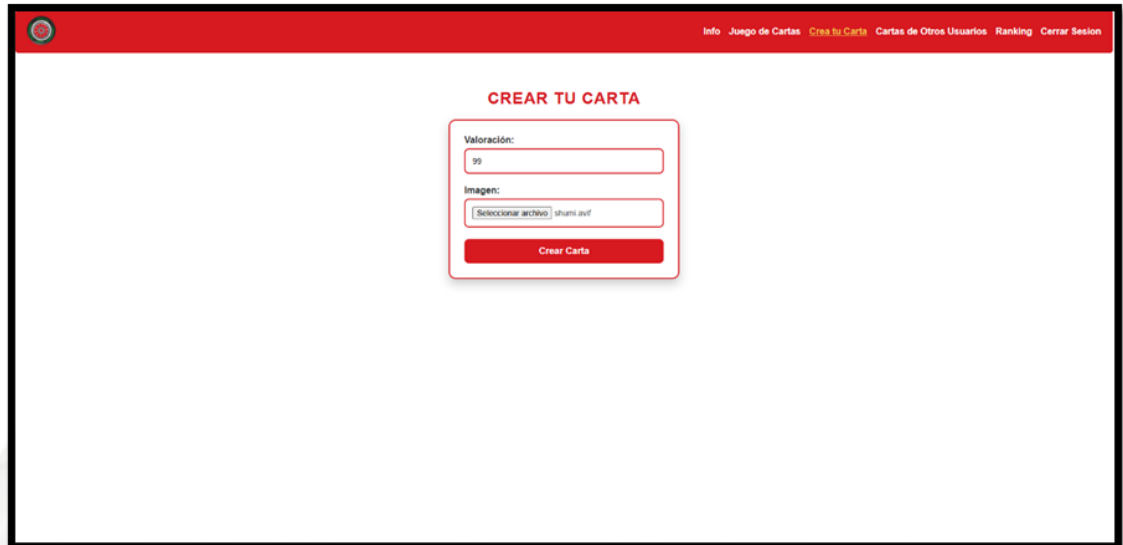


Ilustración 19. Pantalla para crear tu carta personalizada.

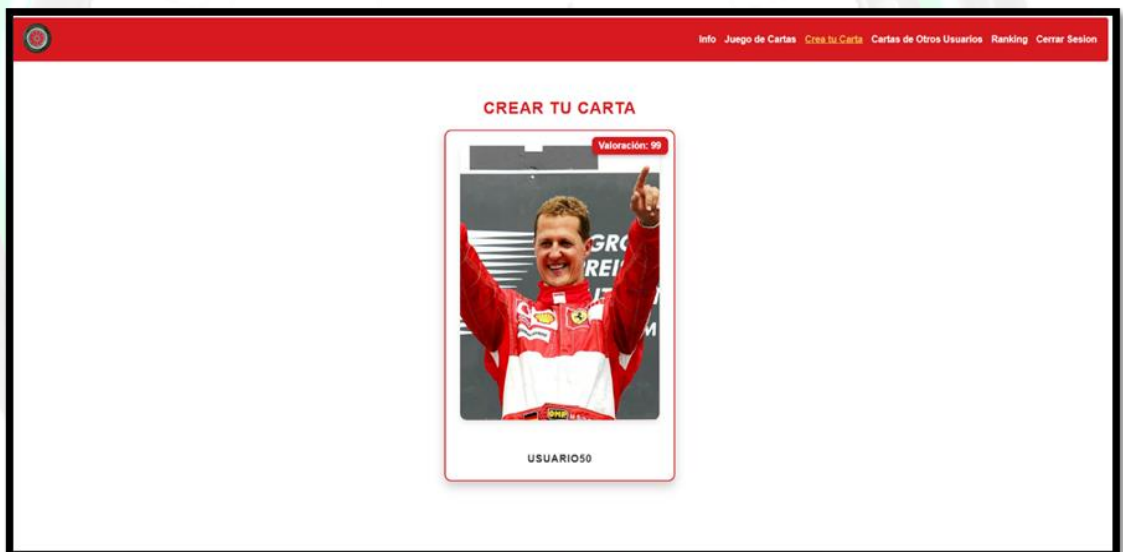


Ilustración 20. Pantalla para crear tu carta personalizada tras haberla creado.

- Visualizar además de su carta propia, **el resto de cartas** que han creado la comunidad de usuarios.

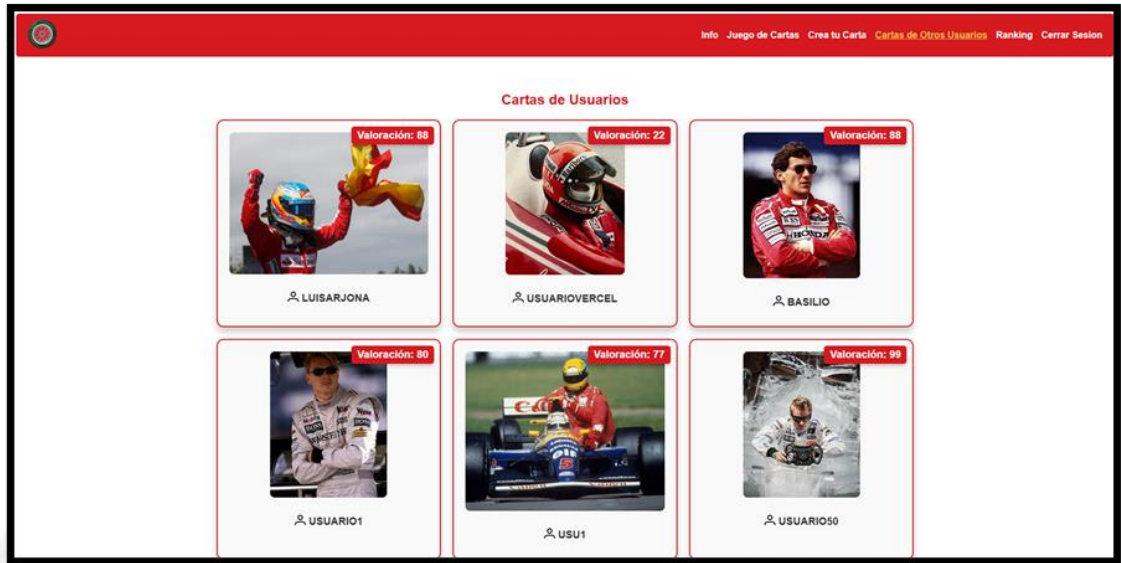


Ilustración 21. Pantalla para visualizar las cartas creadas por otros usuarios.

- Competir en una tabla global** que muestra las cartas que tienen compradas todos los usuarios.

Cuenta con tres filtros, el primero para ordenar por el total de cartas coleccionadas “Ordenar por Total”, el segundo para ordenar por el total de cartas de pilotos coleccionadas “Ordenar por Pilotos”, y el tercero para ordenar por el total de cartas de circuitos coleccionadas “Ordenar por Circuitos”.

Los usuarios así ven incentivado su progreso para intentar ser el que más cartas tenga y mostrárselo al resto.

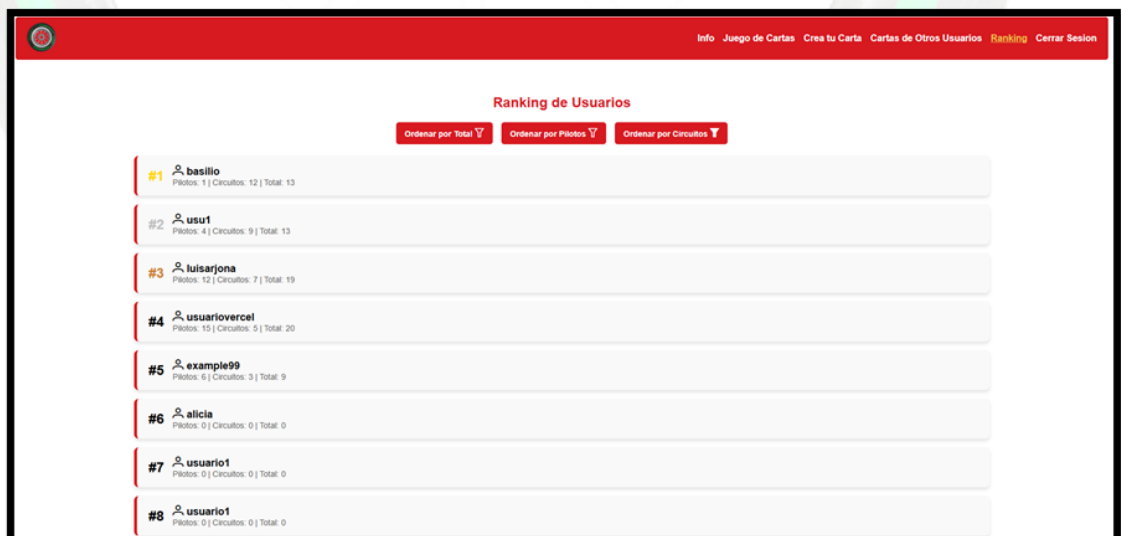


Ilustración 22. Pantalla para visualizar la tabla global de los usuarios.

- En la tabla global al pinchar sobre un usuario, aparecerá a la izquierda de la pantalla la **colección** de ese usuario, con todas las cartas de pilotos y circuitos que tenga coleccionadas.

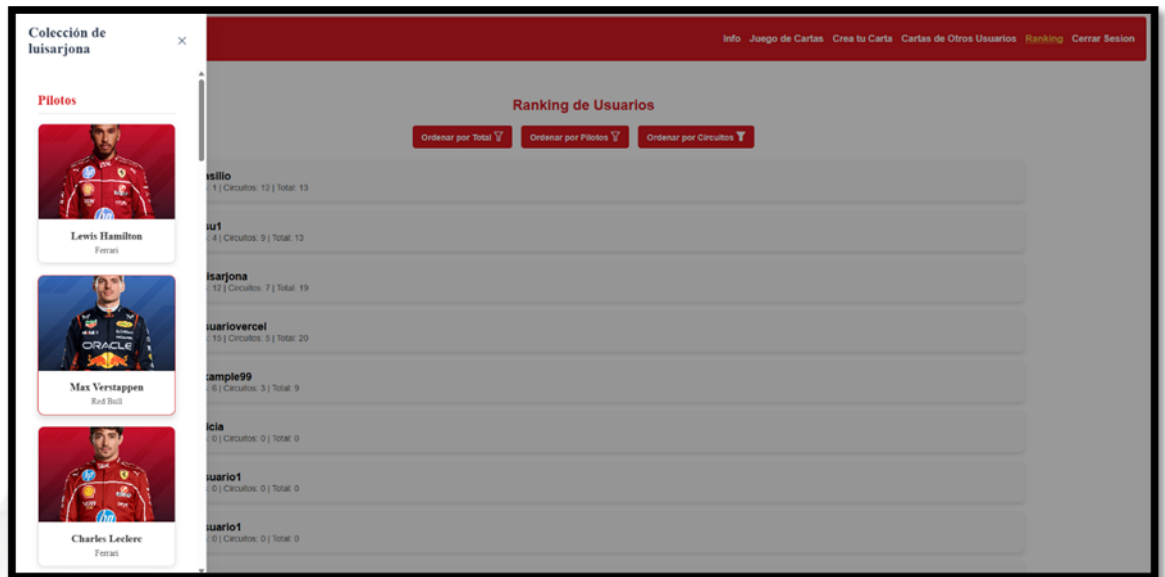


Ilustración 23. Pantalla para ver la colección de un usuario específico.

## 5. Alternativas

Al principio surgió la posibilidad de usar Express.js en vez de Spring Boot para desarrollar el Backend por los siguientes motivos:

- Simplicidad: Express.js está pensado para ser un **Framework simple**, con una curva de aprendizaje baja que te permita desarrollar tu aplicación en menor cantidad de tiempo. Por lo que a pesar de no tener muchos conocimientos en Javascript, o al menos bastante menos que en Java no parecía un gran problema aprender este nuevo Framework.
- Ecosistema: como he mencionado antes, el lenguaje de programación de Express.js es Javascript, lo que **facilita más integrarlo** con Frameworks de Frontend debido a que casi todos usan Javascript o en el caso de Angular la variación Typescript.
- Despliegue: las plataformas de despliegue ponían bastantes dificultades para poder desplegar una aplicación Spring Boot, mientras que con otros Frameworks incluyendo Node Express parecía que con un clic podías **desplegar fácilmente la aplicación**, esta era la mayor razón para elegir Express.js sobre Spring Boot.

Finalmente, ¿Por qué deseché la opción de usar Express.js?:

- Seguridad: Spring Boot es un Framework **más completo**, que tiene un ecosistema de funcionalidades muy amplio, incluyendo principalmente a Spring Security, lo que aseguraba que la autenticación JWT y de 2 Factores iba a ser **más robusta** que en otros Frameworks, si bien conseguirlo es más difícil.
- Conocimiento previo: a pesar de que Express.js tiene una curva de aprendizaje muy baja, el hecho de tener **una base de conocimientos** sobre Spring Boot y Java mucho mayor lo hace ver como una gran ventaja.

- Encontrar una plataforma para desplegarlo: si bien parecía que todas las plataformas ponían bastantes problemas para desplegar una aplicación Spring Boot, gracias a Railway encontré la manera de **desplegar la aplicación fácilmente**, por lo cual la mayor razón para no usar Spring Boot desapareció.

## 6. Conclusión

Gracias al desarrollo de esta aplicación para el Proyecto Fin de Ciclo he conseguido aprender y desarrollar nuevas habilidades como:

- Spring Security: hoy en día la seguridad de las aplicaciones **es clave**, esta es la primera vez que he securizado una aplicación correctamente. Específicamente he conseguido aprender como reforzar la seguridad de mi Backend gracias a usar tokens JWT, realizando un **filtro JWT** que intercepta las peticiones y verifica la identidad del usuario para permitir el acceso. Además a ello le he añadido otra capa de seguridad gracias a la **autenticación de 2 factores**, lo que significa que el usuario necesita proveer un código correcto para poder llegar a obtener ese token JWT que le permite el acceso al resto de la aplicación.
- Authguard: al igual que el Backend, he **securizado el Frontend** haciendo uso también de los tokens JWT, si el token no es válido o está expirado el usuario no puede acceder al resto de la aplicación, tan solo al Inicio de Sesión y el Registro.
- Comentarios: todo el código del Backend está lleno de **Javadocs** para documentar correctamente que funciones realizan todas las clases y los métodos Java, así como los parámetros que reciben y lo que devuelven, facilitando que, si el día de mañana necesito esta aplicación para algo, gracias a esa documentación puedo **entender fácilmente** lo que hace de nuevo.
- Documentación: esta es la primera vez que realizo una documentación tan extensiva para una aplicación. Explicando a fondo todo sobre la arquitectura, tecnologías, endpoints, código, compatibilidad, las distintas pantallas, navegación... Una **habilidad necesaria** muchas veces para un programador que hasta ahora nunca había puesto en práctica.

## 7. Referencias bibliográficas

- Cámara Casares, A. (s.f.). *acamarac02.github.io*. Obtenido de Acceso a Datos: <https://acamarac02.github.io/DevTacora/>
- Espinoza, J. (s.f.). *medium.com*. Obtenido de Protegiendo tu aplicación web con Spring Security y autenticación basada en Tokens JWT: <https://medium.com/@espinozajge/protegiendo-tu-aplicaci%C3%B3n-web-con-spring-security-y-autenticaci%C3%B3n-basada-en-tokens-jwt-1321cbe4c4c3>
- García de Paredes, J. (s.f.). *Apuntes de Desarrollo de Interfaces*. Obtenido de *jgarciadeparedesh02.github.io*: <https://jgarciadeparedesh02.github.io/site/di-apuntes/index.html>
- github.com*. (s.f.). Obtenido de Spring Boot Otp: <https://github.com/hedza06/spring-boot-otp>