



# PROYECTO FINAL DE GRADO

CHECKERED FLAG

Página web de creación y seguimiento de  
ligas de F1

Alen Tokalic  
alen.tc02@gmail.com

## Índice

Motivación .....	4
Resumen .....	4
Introducción.....	4
Objetivo.....	4
Punto de vista Objetivo .....	4
Punto de vista del creador .....	4
Definición del problema.....	4
Estimación de tiempos inicial .....	5
Diseño de la base de datos.....	5
Backend .....	5
Frontend.....	5
Assets .....	5
Tecnologías de uso .....	6
Estándares, Librerías y Programas.....	6
Entity Framework Core .....	6
JWT Tokens.....	6
Angular .....	6
Git .....	6
Fases del Trabajo.....	7
Análisis.....	7
Requisitos Funcionales.....	7
Requisitos No Funcionales .....	7
Diseño de la aplicación .....	9
Introducción .....	9
Arquitectura Inicial .....	9
Modelos .....	9
Endpoints y Controladores.....	9
Migraciones .....	9
Funcionamiento .....	10
Enrutamiento API REST   Endpoints .....	11
Autenticación y Autorización.....	17
Estructura de la base de datos .....	20
Explicación de la base de datos .....	21
Implementación.....	26

Cliente.....	26
Servidor .....	28
Manual de usuario.....	29
Componentes Importantes.....	29
Pruebas Importantes.....	37
Pantalla de admin-races .....	37
Pantalla de admin .....	37
Pantalla Añadir Resultados.....	38
Ampliación y Mejoras .....	39
Modo Mi Equipo.....	39
Gestión de pilotos .....	39
Presupuesto de Equipo.....	39
Patrocinadores.....	40
Mercado de Fichajes.....	40
Gestión de Equipo .....	40
Calculo de tiempos final .....	41
Preparación de la idea y de las herramientas: <b>3H</b> .....	41
Diseño de la estructura del servidor y de la base de datos: <b>15H</b> .....	41
Diseño y creación de la interfaz web: <b>85H</b> .....	41
Pruebas en la aplicación: <b>5H</b> : .....	42
Documentación de la aplicación: <b>15H</b> .....	42
Preparación de la presentación: <b>2:30H</b> .....	42
Tiempo total: .....	42
Bibliografía .....	43
Documentación .NET 6 .....	43
Creación de una web API .....	43
Entorno Servidor, Entidades y Relaciones de Bases de datos y Seguridad JWT .....	43
Para la preparación de ítems despleables .....	43
Para la preparación de gráficos responsivos y animados .....	43
Para cualquier duda o problema que surgía .....	43
Dudas de lenguajes y de diseño para poder implementar soluciones y mejoras, además de .....	43
Para cualquier duda de estructuración de Angular. ....	43
Para el diseño y estructuración que llevara mi base de datos .....	43

Información general de equipos, pilotos y cualquier cosa relacionada .....	43
Conclusión.....	44

## Motivación

Este proyecto nace con la idea e ilusión de crear una manera de guardar los momentos que ocurren cuando los amigos se reúnen y juegan al F1. Cada vez que se celebra una carrera en la vida real, decidimos correr ese circuito y por ello surge la necesidad de llevar ese seguimiento a otro nivel.

## Resumen

Este trabajo tiene como fin crear un servicio de creación de ligas o eventos de F1 22 (el juego) para cualquier tipo de público con el fin de llevar un seguimiento de sus carreras.

El proyecto funcionará de manera similar a un campeonato del mundo de F1.

Los jugadores que participen evolucionarán en estadísticas y habilidades.

El proyecto cuenta con un servidor API REST que sirve de intermediario de la base de datos y el cliente. Con esto nos permite registrar usuarios, modificar estadísticas y cumplir con las peticiones del usuario.

Y el cliente contará con una interfaz de Angular para poder interactuar con más usuarios, y poder realizar un seguimiento de sus puntos y de sus carreras realizadas.

## Introducción

### Objetivo

#### Punto de vista Objetivo

El objetivo de la aplicación como tal es crear registros de carrera, además de pilotos e incluso equipos, para que el usuario en este caso director pueda realizar un seguimiento correcto de su mundial o liga.

#### Punto de vista del creador

El objetivo principal es aprender a crear peticiones personalizadas en función de los servicios que el cliente requiera, además de proveer a la página web de constantes actualizaciones después de añadir los resultados de una carrera disputada, así como estadísticas personalizadas.

### Definición del problema

En la actualidad, existen pocas páginas web en Internet que te proveen de un servicio como es el de creación de ligas personalizadas. Especialmente en mi caso mi campo es mucho más concreto ya que indagamos en el mundo del motorsport, que trabaja con muchos datos que pueden resultar vitales a la hora de visualizar estadísticas.

Esta falta de opciones lleva a los aficionados de la F1 a llevar sus seguimientos en hojas de cálculo como Excel o incluso hojas de papel para apuntar los resultados de los participantes.

Por ello brindo la oportunidad a los aficionados una solución moderna, eficiente e interactiva con un seguimiento más moderado y visualmente más atractivo.

## Estimación de tiempos inicial

Hay que tener en cuenta que esto es una estimación aproximada y pueden variar dependiendo de su futura complejidad y otros factores. Estos datos son solo una guía principal

### Diseño de la base de datos

Este apartado requerirá de entre 4 a 8 horas sin mayores problemas, todo realizado a mano incluido la tarea de añadir los campos para cada tabla. Con esto conseguimos un esquema mental más elaborado y más organizado que más tarde pasaremos al ámbito digital.

### Backend

El lado del servidor siempre necesita un trato diferente que el de restos de elementos, por así decirlo equivaldría a los cimientos de una casa. Además es el encargado de intercambiar información entre la base de datos y el cliente, por ello concentraré esfuerzos en este apartado que una vez que esté bien diseñado y libre de errores solo quedará reunir fuerzas para el otro gran apartado que es el Frontend.

Estimo que hasta un 50% de tiempo tomara realizar esta pesada tarea.

### Frontend

En cuanto a este apartado, utilizaré varias librerías y herramientas para poder aligerar el trabajo que supone la maquetación, para hacer más fácil tanto la creación de formularios como la página en general ya que el CSS puede llegar a ocupar extensas líneas sin fin.

Gracias a librerías como Angular Material o librerías como Bootstrap podría reducir el porcentaje de trabajo en un 40%.

### Assets

Este apartado puede suponer el más liviano de todos, dedicaré hasta un 10% de mi tiempo en preparar las imágenes que puedan elegir los pilotos, entre las fotos de los logotipos de los equipos y de los propios monoplazas. Además de imágenes de pilotos celebrando victorias o campeonatos para más tarde publicar en el apartado de noticias y así ofrecer una mayor inmersión al participante.

También tendrá que seleccionar las imágenes de los circuitos, tanto en su versión de mapa como una foto paisajística.

## Tecnologías de uso

### Estándares, Librerías y Programas

Por parte del backend voy a utilizar Visual Studio, que me ayudará a desarrollar el servidor con el que voy a desplegar la API.

### Entity Framework Core

Entity Framework Core es un paquete NuGet de Visual Studio que utilizamos como asignador relacional de datos, es decir se va a encargar de añadir las entidades (tablas) a la base de datos cada vez que yo cree las clases en la API.

Permite a los desarrolladores de .NET trabajar con una base de datos usando objetos .NET.

Permite prescindir de la mayor parte del código de acceso a datos que normalmente es necesario escribir.

### JWT Tokens

Es un estándar abierto que define un formato compacto y seguro para la transferencia de información entre partes en forma de objetos JSON

Se encarga de desplegar una funcionalidad para determinar la identidad de un usuario, con una serie de claims, también conocidos como privilegios. Su principal función es autorizar y autenticar usuarios en un sistema.

Este se compone de tres partes separadas por puntos (.)

- Encabezado: Contiene información sobre el tipo de token y el algoritmo de cifrado utilizado
- Carga Útil: Contiene la información que se quiere transmitir, en este caso la del usuario (nombre, password, email)
- Firma: Es una firma digital que se utiliza para verificar la integridad del token

### Angular

Utilizaremos este framework de HTML y CSS para realizar de interfaz entre el usuario y la API. Dentro de este utilizaremos TypeScript, similar a JavaScript

En Angular para visualizar la página web utilizaremos componentes que iremos acoplando uno encima de otro hasta conseguir visualizar la página web.

Crearemos los modelos de las clases creadas en la Base de Datos y después con los servicios acceder a ellos mediante las URL obteniendo así los JSON para poder desplegar la información de la BD en la web.

### Git

Durante el desarrollo de mi proyecto he utilizado Git.

Git ha sido fundamental para gestionar y controlar los cambios realizados en el código de mi proyecto.

He creado un repositorio donde he almacenado tanto los directorios de servidor como los de cliente, en los que he podido realizar un commit para registrar los cambios.

## Fases del Trabajo

### Análisis

Para realizar este trabajo he tenido que identificar los objetivos generales que va a tener mi proyecto, es decir su función o su propósito principal. Por ello lo he dividido en dos categorías

#### Requisitos Funcionales

##### *Gestión de objetos*

- Los administradores en este caso los directores de cada liga deben ser capaces de crear su propio ambiente de carreras, como si de un mundial se tratase, en este caso con operaciones CRUD de pilotos, equipos y carreras.

##### *Visualización de datos*

- El director debe de ser capaz de ver información relacionada con los datos de su mundial.
- El entorno técnico realizará operaciones matemáticas que devolverán al usuario en forma de gráficas toda la información que el usuario solicite así como datos de rendimiento del piloto, del equipo y de múltiples campos más.

##### *Actualizaciones Automáticas*

- El director no tiene necesidad de realizar un seguimiento constante de las carreras de los pilotos, ni de los resultados de estos. La única tarea del director en este caso es introducir el orden de parrilla de clasificación y el orden de carrera en la que finalizan y una vez que realiza esto, el programa es capaz de determinar los puntos que merece cada uno tanto puntos de mundial, como de evolución de pilotos y aumento o descenso de precios en el mercado de pilotos.

#### Requisitos No Funcionales

##### *Seguridad*

- Mi aplicación garantiza la seguridad y protección de la privacidad del usuario gracias a las funciones y métodos de cifrado por parte del servidor y el recibimiento de claves de seguridad y su almacenamiento por parte de la interfaz web.

##### *Usabilidad*

- La interfaz web a lo largo del proyecto ha sufrido numerosos cambios en todas y cada una de las partes en la que se incluyen formularios. Se han desarrollado varios cambios de diseño para que no resulte agónico crear registro, como la función de select o drag and drop.



### *Escalabilidad*

- El código de la aplicación, y sobre todo su arquitectura está de tal manera desarrollada que es capaz de evolucionar e incluso recibir futuras actualizaciones que no conlleven una reestructuración de código bastante amplia.

### *Mantenibilidad*

- La interfaz web es capaz de adaptarse a cualquier adversidad o cambio que se pueda producir en cualquier momento del mundial.
- Un ejemplo de ello sería la suma de los puntos de los pilotos, o su tardía incorporación al mundial entre otros temas.

## Diseño de la aplicación

### Introducción

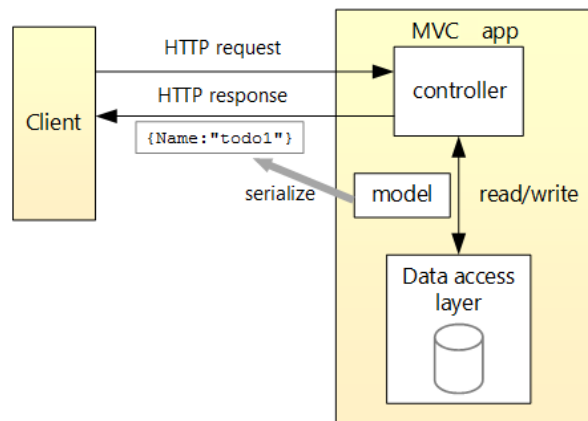
En esta sección se abordarán aspectos como la arquitectura de la aplicación, en este caso la del servidor y la de la base de datos además de una visión clara y detallada de las rutas que seguirá un usuario cuando se registre y cuando inicie sesión

Entender este diseño asegura ya los conocimientos de cómo funciona la seguridad, integridad y funcionalidad de la mayoría del servidor así como de la base de datos.

### Arquitectura Inicial

Vamos a utilizar el modelo vista controlador de una API WEB basada en un controlador que utiliza una base de datos.

Como se puede observar separamos en dos programas el cliente y el servidor, en este apartado hablaremos de la base de datos y del servidor



### Modelos

Las clases también denominados modelos, en el código de la aplicación representan las entidades o tablas que más tarde se reflejarán en nuestros datos. Cada una de estas contiene atributos y relaciones entre una o más tablas

### Endpoints y Controladores

Los controladores de cada modelo trabajan con las solicitudes HTTP del cliente, es decir son los intermediarios entre el cliente y la base de datos, su tarea consiste en acceder a los recursos para devolvérselo de vuelta al cliente con puntos de accesos o URL a las que llamaremos Endpoints

### Migraciones

Las migraciones son una forma de gestionar y mantener la estructura de la base de datos a lo largo del tiempo sin necesidad de acceder directamente al programa que hace funcionar la base de datos, es decir, por medios de comandos agregaremos diferentes estructuras a esta.

Además las migraciones mantienen un historial de los cambios realizados tanto en el servidor como en la base de datos, lo que facilita el seguimiento de cambios y la colaboración de más miembros en el grupo

### Funcionamiento

#### 1. Cliente realiza una petición:

El cliente a través de una solicitud HTTP, envía una petición al servidor para acceder a un recurso o seleccionar algún servicio.

#### 2. El Servidor recibe la petición

Mediante un enrutamiento configurado en el api ("endpoint"), determina que controlador debe manejarlo

#### 3. Acción del Controlador

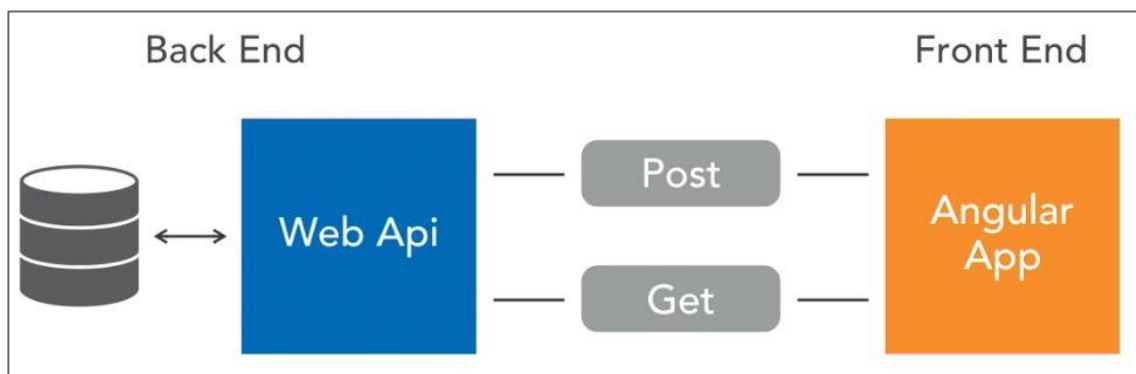
El controlador procesa la petición y dependiendo de la ruta realiza las operaciones necesarias

#### 4. Acceso a la base de datos con Entity Framework

El controlador utiliza Entity Framework para interactuar con la base de datos. Este paquete simplifica el acceso a la base de datos al mapear las entidades de la base de datos

#### 5. Respuesta al cliente

Una vez que el controlador ha procesado la petición y ha obtenido los datos se genera una respuesta, como en mi caso un JSON que después recibe el cliente.



## Controlador Equipo

GET	/api/team	AUTH
DEV*	Devuelve una lista de todos los equipos de toda la base de datos	NO
GET	/api/team/{points}	AUTH
DEV*	Devuelve una lista de todos los equipos de la base de datos ordenados de mayor a menor en función de sus puntos.	NO
GET	/api/team/league/{leagueId}	AUTH
	Este endpoint sirve para obtener una lista de todos los equipos que pertenecen a una liga específica. Se pasa la Id de la liga	NO
GET	/api/team/points/{leagueId}	AUTH
	Este endpoint sirve para obtener una lista de los equipos de una liga ordenados de mayor a menor en base a los puntos que tengan. Se le pasa la id de la liga	NO
GET	/api/team/{id}	AUTH
	Este endpoint sirve para obtener un equipo en específico. Se le pasa la id del mismo	NO
GET	/api/team/teams-available/league/{leagueId}	AUTH
	Este endpoint devuelve una lista con los equipos disponibles (sin ningún piloto) de una liga específica. Se le pasa la id de la liga	NO
GET	/api/team/teams-without-drivers/league/{leagueId}	AUTH
	Este endpoint devuelve una lista con los equipos sin ningún piloto. Se le pasa la id de la liga	NO
GET	/api/team/teams-without-one-driver/league/{leagueId}	AUTH
	Este endpoint devuelve una lista con los equipos pertenecientes a una liga que tienen un solo piloto. Se le pasa la id de la liga	NO
POST	/api/team/post	AUTH
	Este endpoint sirve para añadir un objeto Team (Equipo) a la base de datos. Se le pasan los campos del objeto	NO
PUT	/api/team/{id}	AUTH

	Este endpoint sirve para modificar un objeto Team (Equipo) a la base de datos. Se le pasa la id del equipo que queremos cambiar	NO
--	---	----

#### Controlador Circuitos

GET	/api/circuit/	AUTH
DEV*	Devuelve una lista con todos los circuitos de la base de datos	NO
GET	/api/circuit/{id}	AUTH
	Devuelve el circuit por su id. Se le pasa la id del circuito	NO
POST	/api/circuit/post	AUTH
	Este endpoint sirve para añadir una clase Circuito a la base de datos. Se le pasa el objeto	NO
PUT	/api/circuit/{id}	AUTH
	Este endpoint sirve para modificar la clase Circuito con su ID.	NO
DELETE	/api/circuit/{id}	AUTH
	Este endpoint sirve para eliminar un objeto Circuito de la base de datos	NO

### Controlador Driver

GET	/api/driver	AUTH
DEV	Devuelve todos los pilotos de la base de datos	NO
GET	/api/driver/league/{leagueId}	AUTH
	Devuelve todos los pilotos que pertenezcan a una liga específica. Se le pasa la id de la Liga	NO
GET	/api/driver/price/{leagueId}	AUTH
	Devuelve todos los pilotos pertenecientes a una liga específica ordenados de mayor a menor por su precio de mercado. Se le pasa la Id de la liga	NO
GET	/api/driver/team/{teamId}	AUTH
	Devuelve todos los pilotos pertenecientes a un equipo en específico. Se le pasa la id del equipo	NO
GET	/api/driver/{id}	AUTH
	Devuelve un objeto piloto(Driver).Se le pasa la id de piloto	NO
POST	/api/driver/post	AUTH
	Este endpoint sirve para añadir una clase Driver a la base de datos. Se le pasa el objeto	NO
PUT	/api/driver/{id}	AUTH
	Este endpoint sirve para modificar el objeto Driver por medio de una Id. Se le pasa la Id	NO
DELETE	/api/driver/{id}	AUTH
	Este endpoint sirve para borrar de la base de datos un objeto Driver por medio de un identificador. Se le pasa la ID	NO

### Controlador Race

GET	/api/race	AUTH
	Devuelve una lista con todas las carreras de la base de datos	No
GET	/api/race/{id}	AUTH
	Devuelve un objeto Carrera por su id. Se le pasa la id de Carrera	NO
GET	/api/race/sponsor/{sponsorId}	AUTH
	Devuelve la carrera que contiene un sponsor específico. Se le pasa el sponsorId	NO
PUT	/api/race/{id}	AUTH
	Este endpoint sirve para modificar la clase Carrera	NO
POST	/api/race/	AUTH
	Este endpoint sirve para agregar un objeto Carrera a la base de datos. Se le pasa el objeto	NO
DELETE	/api/race/{id}	AUTH
	Este endpoint sirve para eliminar un objeto Carrera a través de su ID. Se le pasa la id de Carrera	NO

### Controlador Stat

GET	/api/stat/	AUTH
DEV*	Devuelve una lista con todas las estadísticas de la base de datos	NO
GET	/api/stat/{id}	AUTH
	Devuelve un objeto Estadística por su ID. Se le pasa la ID.	NO
GET	/api/stat/driver/{driverId}	AUTH
	Devuelve las estadísticas de un piloto por la id de un piloto. Se le pasa la ID.	NO
POST	/api/stat/	AUTH
	Este endpoint sirve para añadir a la base de datos el objeto Estadística.	NO
PUT	/api/stat/{id}	AUTH
	Este endpoint sirve para modificar objetos Estadística por medio de su ID.	NO
DELETE	/api/stat/{id}	AUTH
	Este endpoint sirve para borrar objetos Estadística a través de su ID.	NO



### Controlador Ability

GET	/api/ability/	AUTH
	Devuelve una lista con todas las habilidades de la base de datos	NO
GET	/api/ability/{id}	AUTH
	Devuelve un objeto Habilidad al pasarle la id. Se le pasa la id de habilidad	NO
GET	/api/ability/driver/{driverId}	AUTH
	Devuelve la Habilidad perteneciente a un Piloto a través de la id de un piloto. Se le pasa la id piloto	NO
POST	api/ability/	AUTH
	Este endpoint sirve para agregar una clase Abilidad	NO

## Autenticación y Autorización

### Introducción Registro

La seguridad en toda aplicación es vital así como salvaguardar la integridad de datos del usuario que se registra o inicia sesión.

Por ello tuvimos que desarrollar un sistema de registro e inicio de sesión de manera que sea seguro pero a la vez eficaz

### Esquema de Registro

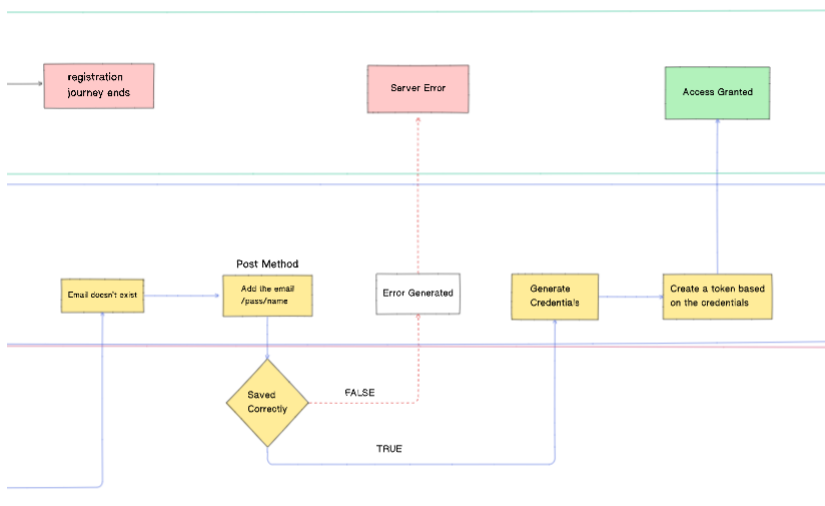
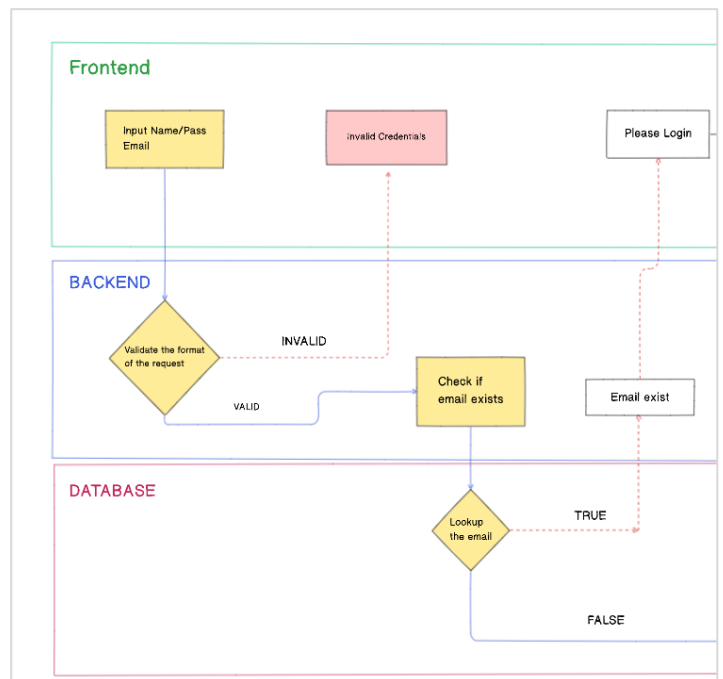
En el esquema se pueden observar 3 bloques, que son los entornos por los que se mueve la aplicación constantemente Frontend, Backend y Database

La dirección es de arriba hacia abajo, empieza con el usuario introduciendo los datos en la web

El usuario ingresa los datos, que pasan al backend para comprobar que son datos válidos para su verificación y proceso

Se realiza una petición a la base de datos para comprobar la existencia del email

Si el email existe pasa a ser de la arquitectura Login por tanto esta estructura ya no tiene autoridad



En caso contrario seguimos con su creación, le pasamos los datos a la base de datos con un método POST.

Generamos las credenciales de creación para almacenar la contraseña con una clave de seguridad y por último lo autenticamos automáticamente al usuario recién registrado

### Introducción Login

Este esquema representa el proceso mediante el cual los usuarios autenticados pueden acceder a su cuenta en la API.

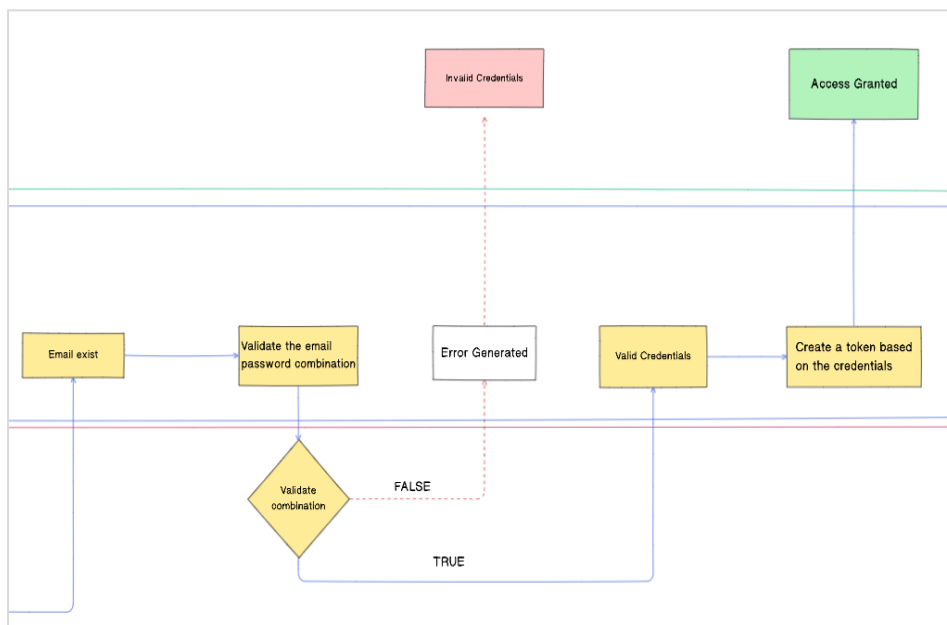
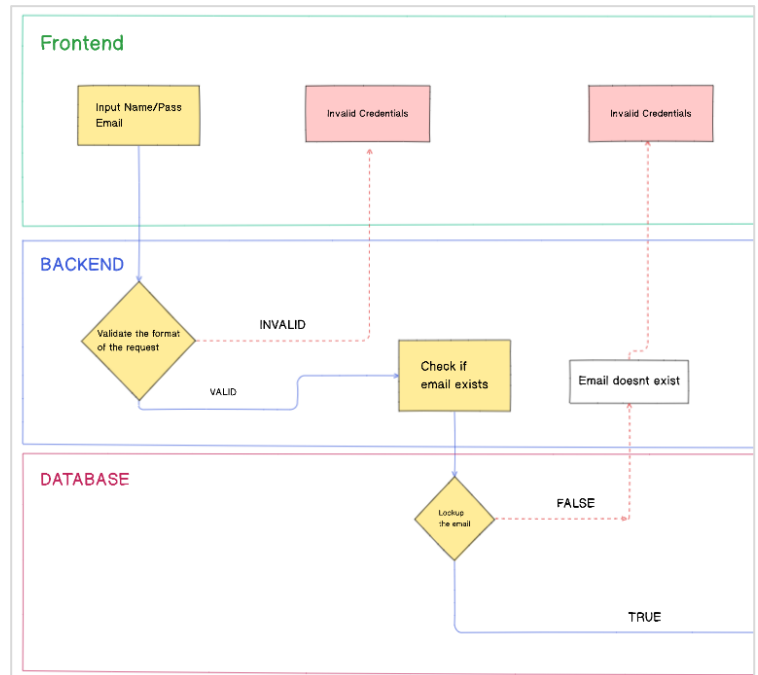
El inicio de sesión implica el intercambio de credenciales del usuario por un token válido.

### Esquema Login

Como se puede observar, al principio se le pasan los valores de los inputs en el Frontend, como si fuera un formulario.

El backend lo recoge y comprueba que son datos válidos.

Comprueba que existe el email del usuario

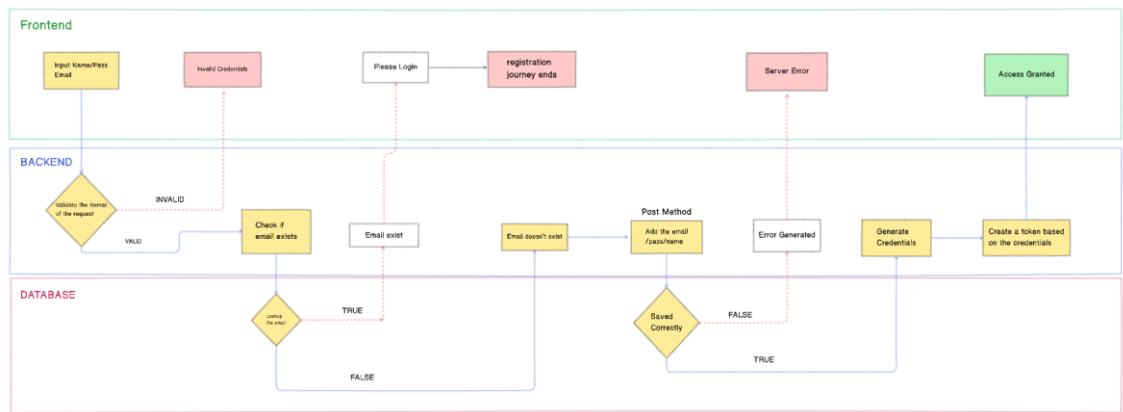


En caso de que exista comprueba que los datos coinciden con los estándares de contraseña y email.

En caso positivo se validan las credenciales y se prepara un token para estas que le permita al usuario acceder a su página web o a cualquier servicio

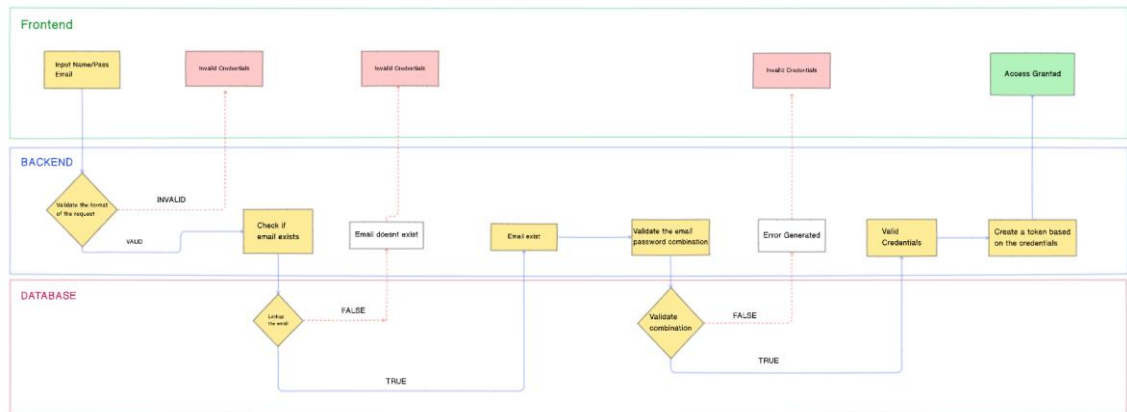
Esquemas completos de Login y Register

Register Journey



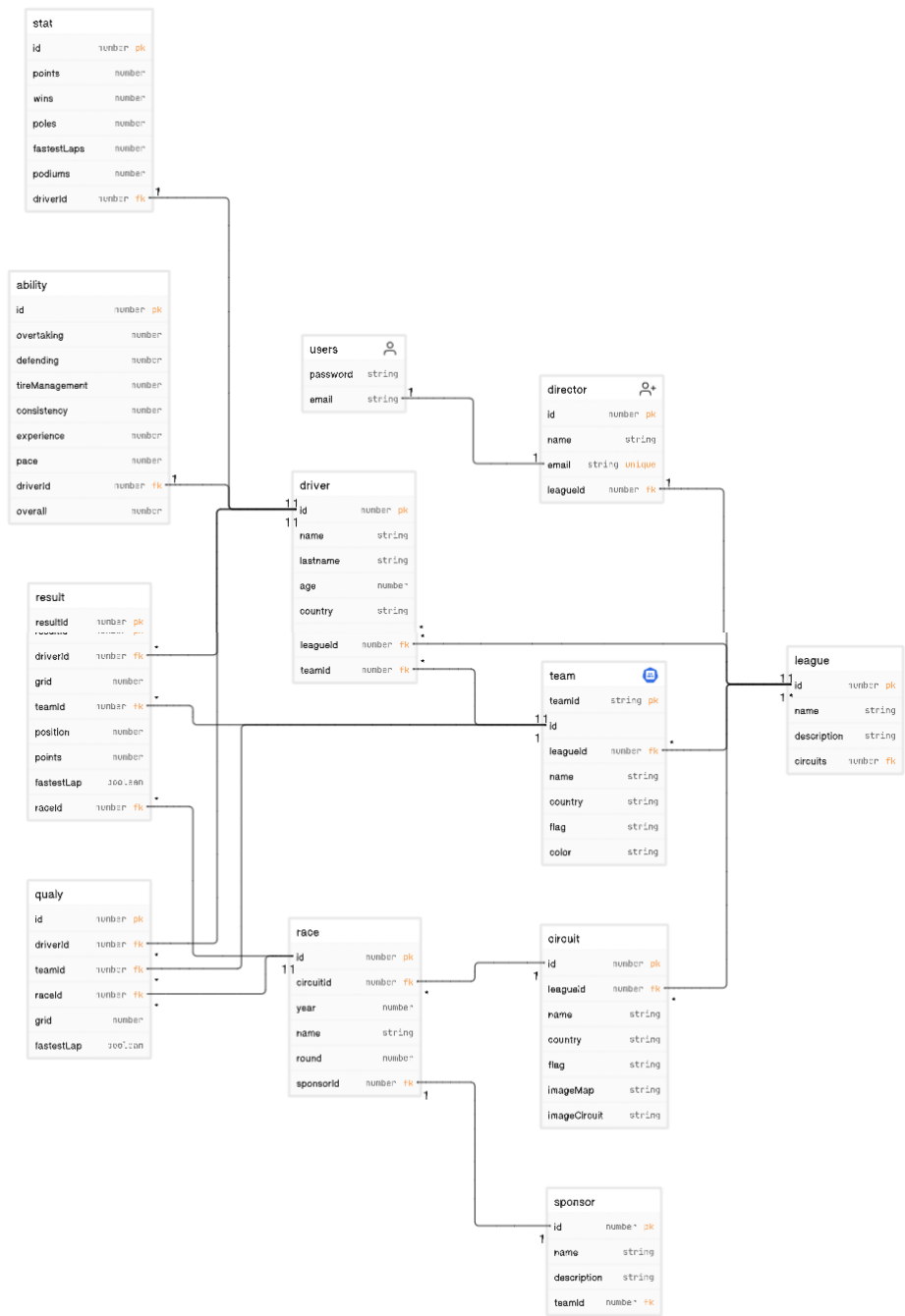
REGISTER JOURNEY

Login Journey



LOGIN JOURNEY

# Estructura de la base de datos



## Explicación de la base de datos

### Objetivo

El objetivo principal de la base de datos es garantizar la privacidad y la seguridad de los datos de los usuarios. Para lograr esto se ha establecido un esquema por el cual se tiene en cuenta la liga que ha creado el usuario, es decir todo está vinculado a la liga creada.

### Diseño

El diseño esta implementado de manera que se intente reducir la redundancia de datos, es decir el administrador no tiene por qué crear el mismo circuito cada vez que quiere celebrar una carrera.

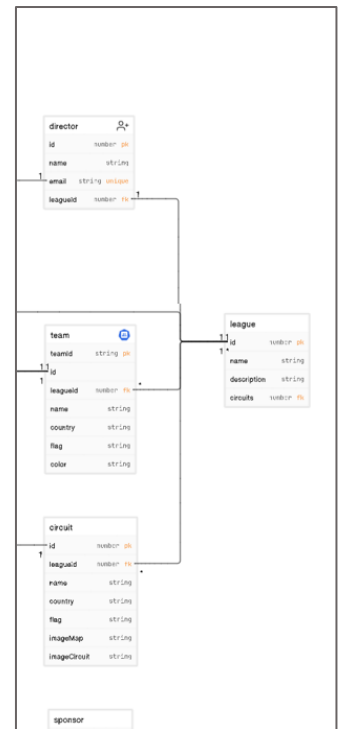
Por lo se hace referencia a ese valor a través de una clave foránea como es el del caso anterior.

### Controladores y Clases JOIN

Para acompañar a la reducción de la duplicación innecesaria de datos, decidimos crear en el backend clases y controladores JOIN que nos ayudaran a liberar carga de trabajo a la base de datos.

Las clases JOIN se utilizan para unir datos entre dos o más entidades relacionadas en una consulta, así facilitamos la combinación de datos de diferentes tablas mediante la especificación de condiciones de unión.

Esto proporciona flexibilidad y potencia al diseño de la aplicación, ya que podemos involucrar más datos de manera eficiente, gracias a esto nos ayuda a mantener una relevancia mayor y más precisa a la hora de proporcionar datos al usuario.



```
public class RaceResult
{
    //Result
    public int ResultId { get; set; }
    public int RaceId { get; set; }

    //Team
    public string TeamColor { get; set; }
    public string TeamName { get; set; }
    public string TeamShieldImage { get; set; }
    public string TeamVehicleImage { get; set; }

    //Race
    public int RaceYear { get; set; }
    public int RaceRound { get; set; }
    public string RaceName { get; set; }

    //Piloto
    public int DriverId { get; set; }
    public string DriverName { get; set; }
    public string DriverLastName { get; set; }
    public string DriverCountry { get; set; }
    public string DriverFlag { get; set; }
    public int DriverNumber { get; set; }
    public string DriverImageDriver { get; set; }

    //Result
    public int ResultGrid { get; set; }
    public int ResultPosition { get; set; }
    public int ResultPoints { get; set; }
    public bool ResultFastestLap { get; set; }
}
```

Clase

```
[HttpGet]
public ActionResult<List<RaceResult>> GetAllRaceResult()
{
    //var _context = new UserRegistrationContext();
    var raceResultList = (from r in _context.Results
        join d in _context.Drivers on r.driverId equals d.driverId
        join t in _context.Teams on r.teamId equals t.teamId
        join ra in _context.Races on r.raceId equals ra.Id
        join re in _context.Results on r.raceId equals re.resultId
        select new RaceResult()
        {
            ResultId = r.resultId,
            RaceId = r.raceId,

            DriverName = d.Name,
            DriverLastName = d.LastName,
            DriverCountry = d.Country,
            DriverFlag = d.Flag,
            DriverNumber = d.Number,
            DriverImageDriver = d.imageDriver,

            RaceYear = ra.year,
            RaceRound = ra.round,
            RaceName = ra.name,

            ResultGrid = r.grid,
            ResultPosition = r.position,
            ResultPoints = r.points,
            ResultFastestLap = re.fastestLap,

            TeamColor = t.color,
            TeamName = t.name,
            TeamShieldImage = t.shieldImage,
            TeamVehicleImage = t.vehicleImage,
        }).ToList();

    return raceResultList;
}
```

Controlador

## Modelos

### Team/Escudería

Esta clase tiene como función proveer a los pilotos de medios para poder participar. Por cada equipo pueden participar dos pilotos, aunque puedan entrar tantas reservas como se deseen.

Así como un piloto tiene estadísticas el equipo las tiene por igual sumando las estadísticas de los pilotos que están a su cargo.

También el coche tiene mucha información que dar por lo tanto he decidido introducir 3 campos nuevos que hacen referencia al coche de la escudería, para más tarde tener la oportunidad de filtrar por quien tiene el motor más competitivo

Una clase Equipo tiene una id Leagueld que es la que utiliza el director para desplegar los equipos en una liga específica.

### Stat

Esta tabla al igual que la de Ability pertenece al piloto.

Cada piloto poseerá una serie de estadísticas que le permitan al director observar mediante gráficos quien es el más consistente en cuanto a clasificaciones o incluso en ritmo de carrera

Todas y cada una de las gráficas tomarán datos y referencias de esta tabla mayoritariamente.

```
public class Stat
{
    public int Id { get; set; }
    public int DriverId { get; set; }
    public int Points { get; set; }
    public int Dnfs { get; set; }
    public int Wins { get; set; }
    public int Poles { get; set; }
    public int FastestLaps { get; set; }
    public int Podiums { get; set; }
    public int HighestGridPos { get; set; }

    //Veces que le ha ganado a su compañero
    public int beatTeamMateRate { get; set; }

    //Circuito Id
    public int HighestScoringTrack { get; set; }
}
```

```
public class Ability
{
    public int driverId { get; set; }
    public int abilityId { get; set; }

    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int overtaking { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int defending { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int tireManagement { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int consistency { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int experience { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    public int pace { get; set; }
    [Range(20, 99, ErrorMessage = "Los puntos deben estar entre 20 y 99")]
    //public int wetClimate { get; set; }
    public int overall { get; set; }
}
```

### Ability

Este modelo nace como soporte a la clase Driver o Piloto.

Esta clase consiste en proveer al piloto de habilidades que más tarde adquirirá al participar en carreras.

Como se puede observar tiene dos claves DriverId y AbilityId

El identificador de AbilityId y la clave externa que pertenece a DriverId

Las etiquetas que aparecen arriba determinan el rango que debe seguir el número entre 20 el más pequeño y 99 el máximo valor.

## Sponsor

Esta es la clase Sponsor, tendrá un papel minoritario en este proyecto ya que apenas va a hacer apariciones.

Su función es proveer a los Equipos y Carreras del propio patrocinador y darse a conocer.

Esta clase está incluida por su gran capacidad de evolución como futura fuente de ingresos, elevando así el potencial de ampliación del proyecto.

```
public class Sponsor
{
    public int sponsorId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }

    public string Link { get; set; }
    public string imgShieldBlack { get; set; }

    public string imgShield { get; set; }
    public int Team { get; set; }
}
```

```
public class Driver
{
    public int driverId { get; set; }

    //Piloto
    public string Name { get; set; }
    public string Lastname { get; set; }
    public int Age { get; set; }
    public string Country { get; set; }
    public string Flag { get; set; }
    public int Number { get; set; }
    public string imageDriver { get; set; }

    //TeamId
    public int team { get; set; }

    //Fantasy
    public int seasonStartPrice { get; set; }
    public int currentPrice { get; set; }
    public int seasonChange { get; set; }

    //League Id
    public int leagueId { get; set; }
}
```

## Driver/Piloto

El modelo de la derecha corresponde al Piloto, en este caso Driver.

Como se puede observar tiene su Id propia que será una clave única para poder identificar por un número que no se va a repetir.

Tiene diversos campos que se podrían corresponder con los de una persona normal que por lo general serán tipo string.

Además contienen id 's de diversas tablas que utilizará el controlador para relacionar al piloto con diversas características únicas, como la id de equipo al que pertenece y también la id de la liga en la que participa.

## Circuit

La clase "Circuit" representa un circuito en el sistema, que incluye información como su nombre, país, imagen del mapa y del circuito, entre otros detalles relevantes.

Además, se encuentra asociada a una lista de ligas en las que participa dicho circuito. Esta clase permite gestionar y acceder a la información específica de cada circuito dentro de la aplicación.

```
public class Circuit
{
    public int circuitId { get; set; }
    public string name { get; set; }
    public string country { get; set; }
    public string flag { get; set; }
    //Imagen del mapa del circuito
    public string imageMap { get; set; }

    //Distancia de carrera
    public int laps { get; set; }
    public int length { get; set; }
    //Informacion historica
    public string driverRecord { get; set; }
    public string lapRecord { get; set; }

    //Imagen del circuito
    public string imageCircuit { get; set; }
    [JsonIgnore]
    public List<Liga> Ligas { get; set; }
}
```



### Result

La clase Result representa los resultados de una carrera en el sistema. Contiene atributos como el identificador del resultado, el identificador de la carrera, el identificador del piloto, el identificador del equipo, la posición en la parrilla de salida, la posición final, los puntos obtenidos y un indicador de si se logró la vuelta más rápida. Esta clase permite almacenar y acceder a los resultados de las carreras en la aplicación.

```
public class Result
{
    public int resultId { get; set; }
    public int raceId { get; set; }
    public int driverId { get; set; }
    public int teamId { get; set; }
    public int grid { get; set; }
    public int position { get; set; }
    public int points { get; set; }
    public bool fastestLap { get; set; }
}
```

```
public class Liga
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Descripcion { get; set; }
    public DateTime FechaInicio { get; set; }
    public DateTime FechaFin { get; set; }
    public string Ubicacion { get; set; }

    public int DirectorId { get; set; }

    public int currentRound { get; set; }

    public List<Circuit> Circuits { get; set; }
}
```

### Liga

La clase Liga representa una liga en el sistema. Contiene atributos como el identificador de la liga, el nombre, la descripción, la fecha de inicio y finalización, la ubicación y el identificador del director de la liga.

También incluye el número de ronda actual y una lista de circuitos asociados a la liga. Esta clase permite almacenar información sobre las ligas, como su nombre, fechas y ubicación, así como establecer la relación con el director y los circuitos correspondientes.

### Director

Permite obtener y manipular la información básica de un director, como su nombre, correo electrónico y la liga a la que está asignado.

```
public class DirectorDTO
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public int LeagueId { get; set; }
}
```

### Race

Esta clase se utiliza para almacenar información sobre las carreras, como su año, nombre, ronda y detalles relacionados.

Permite identificar el circuito en el que se llevará a cabo la carrera, así como el patrocinador principal asociado. Además, la carrera está vinculada a una liga específica mediante su identificador de liga.

```
public class Qualy
{
    public int Id { get; set; }
    public int RaceId { get; set; }
    //Tiempo marcado
    public string Time { get; set; }
    public int DriverId { get; set; }
    public int Grid { get; set; }
    public int TeamId { get; set; }
    //Fastest Lap
    public bool FastestLap { get; set; } = false;
}
```

### Qualy

Esta clase se utiliza para almacenar información sobre las sesiones de clasificación en las carreras.

Guarda el tiempo marcado por el piloto, su posición en la parrilla y si logró la vuelta más rápida. Además, se registra el identificador de la carrera y el identificador del piloto y equipo asociados.

```
public class Race
{
    public int Id { get; set; }
    public int year { get; set; }
    public string name { get; set; }
    public int round { get; set; }

    //Id del circuito en el que se disputa
    public int Circuit { get; set; }

    public string date { get; set; }

    //Id del sponsor principal de la carrera
    public int Sponsor { get; set; }

    public int leagueId { get; set; }
}
```

La clase Qualy permite manejar los datos relacionados con las sesiones de clasificación y realizar operaciones como almacenar, recuperar y actualizar información sobre el tiempo marcado, la posición en la parrilla y otros detalles relevantes de las sesiones de clasificación.

## Implementación

En esta sección, se detalla la implementación de la aplicación, incluyendo las tecnologías, librerías y paquetes utilizados. A continuación, se presentan los componentes clave que han sido utilizados para desarrollar la aplicación.

Estas tecnologías, librerías y paquetes han sido seleccionados cuidadosamente para garantizar un desarrollo eficiente y robusto de la aplicación.

Los dividiremos en dos categorías, en este caso las librerías para el lado del cliente y el lado del servidor.

### Cliente

#### *Jwt-decode*

Esta librería es una herramienta que permite decodificar Tokens de JWT, que es precisamente la tecnología que utilizamos en el lado del servidor para encriptar los códigos de acceso a la base de datos o al frontend. Además de que es compatible con Angular.

Por tanto es indispensable y no podía no mencionarla.

#### *Ngx-cookie-service*

La librería Ngx-cookie-service es otra de tantas herramientas que permite manejar cookies en aplicaciones Angular. Muy parecida a LocalStorage, proporciona una manera fácil y segura de manipular las cookies en nuestra aplicación de forma que podemos almacenar el código encriptado del usuario o incluso datos de inicio de sesión.

#### *Flag-icons*

Dependemos de este paquete por su utilidad a la hora de soportar los atributos Flag que se sitúan en varias clases. Básicamente su función es de suministrarnos todas las banderas de todos los países del planeta.

Así que en el momento que un piloto es de un determinado país, recogiendo el código del país al que pertenece podemos extraer la bandera perteneciente a este, ahorrándonos así una carga excesiva de descargas de banderas y una búsqueda automática de este.

#### *Bootstrap*

Este framework es esencial para los estilos css de todos los archivos, además de que es el rey en su ámbito en cuanto a su fácil instalación, integración y diseño. Sin duda un referente que no podíamos dejar atrás.

#### *Bootstrap Icons*

Como muestra su nombre Bootstrap icons es una sección orientada solo a la descarga y obtención de iconos. Tiene una amplia biblioteca en la que puedes seleccionar cualquier icono que necesites. Además de que tiene un gran surtido de variedades de formas y de colores. Muy útil y sencillo de utilizar.

### *Angular Material*

Angular Material es una biblioteca de componentes UI para Angular que proporciona una amplia variedad de elementos de interfaz de usuario, muy parecido a Bootstrap pero con la diferencia de que están listos para usar en las aplicaciones Web.

Utilizaré esta herramienta ya que se integra en Angular de manera perfecta lo que facilita su uso, además tiene un estilo muy elegante basado en Material Design.

También vamos a utilizar la directiva Drag and Drop de Angular Material que nos permitirá reordenar las listas e incluso arrastrar elementos a ella sin necesidad de formularios.

### *Ng-Select*

Cualquier persona que vea formularios interminables con números que memorizar se va a acabar cansando, es por eso que yo he elegido ng-select.

Esta librería proporciona una interfaz de usuario elegante y fácil de usar para la selección de opciones. Te permite filtrar y mostrar sugerencias mientras escribes, además de que tiene una amplia personalización que lo hacen un componente muy fuerte.

Además utiliza la herramienta Bootstrap, por lo que su personalización y compatibilidad con los demás componentes es un éxito rotundo.

### *Charts.js*

Charts.js es una biblioteca de gráficos interactivos que vamos a utilizar para visualizar todo tipo de datos.

Permite crear gráficos de todo tipo y con una gran cantidad de opciones de personalización. Esto nos permitirá mostrar múltiples datos al Administrador de una manera útil y sencilla pero elegante al mismo tiempo.

También existe la probabilidad de agregar animaciones y múltiples series de datos, por lo que lo vuelve una herramienta muy potente.

Servidor

#### *EntityFrameworkCore*

Framework de mapeo objeto-relacional (ORM) que permite interactuar con bases de datos utilizando objetos y consultas LINQ.

#### *AutoMapper*

Facilita la configuración e inyección de dependencias en ASP.NET Core.

#### *Identity.EntityFrameworkCore*

Proporciona implementaciones de las interfaces de ASP.NET Core Identity basadas en Entity Framework Core. Esto facilita la gestión de la autenticación y autorización de usuarios en tu aplicación.

#### *SQL Server:*

SQL Server permite utilizar el proveedor de base de datos de SQL Server con Entity Framework Core. Permite interactuar con bases de datos SQL Server de manera eficiente.

#### *EntityFrameworkCore.Tools:*

Proporciona herramientas adicionales para Entity Framework Core, como la generación de scripts de bases de datos y la ejecución de migraciones.

#### *Newtonsoft.Json:*

Newtonsoft.Json es una popular librería de serialización y deserialización JSON para .NET. Proporciona métodos para convertir objetos .NET en formato JSON y viceversa.

#### *Swashbuckle:*

Es una librería que genera automáticamente una interfaz de usuario interactiva (UI) para la documentación de la API basada en Swagger. Permite explorar y probar los endpoints de la API de manera sencilla.

#### *Filters:*

Filters es una extensión de Swashbuckle.AspNetCore que proporciona filtros personalizados para documentar y validar los parámetros de los endpoints de la API.

#### *Tokens.Jwt:*

Tokens.Jwt es una librería que proporciona soporte para trabajar con JSON Web Tokens (JWT) en .NET. Permite generar, firmar, validar y manipular Tokens JWT en tu aplicación.

#### *JwtBearer:*

Es una librería que facilita la autenticación basada en Tokens JWT en aplicaciones ASP.NET Core. Proporciona middleware que valida y procesa los Tokens JWT enviados en las solicitudes HTTP.

### BCrypt.Net-Next:

Es una librería que proporciona funciones de hashing y verificación de contraseñas utilizando el algoritmo BCrypt. Es útil para almacenar y comparar contraseñas de forma segura en tu aplicación.

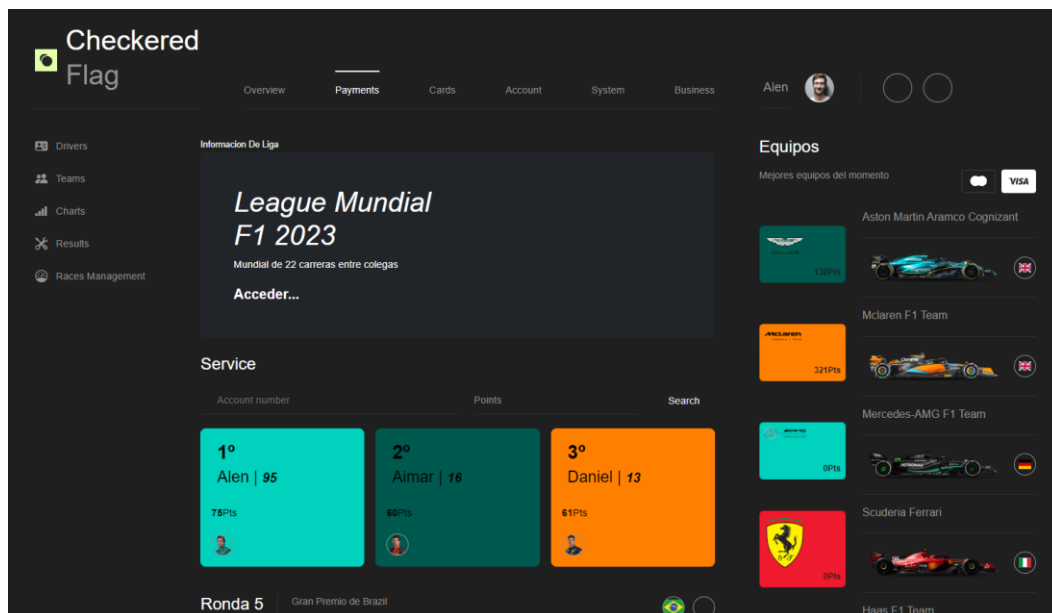
### Manual de usuario

#### Componentes Importantes

#### Dashboard / Página Principal

El cliente está desarrollado de manera que un Usuario se registra y pasa a ser un Administrador, a partir de ahí tiene una página principal también llamada Dashboard que interactúa con el usuario.

En este caso, Angular tiene una cualidad que vamos a aprovechar al máximo y es que podemos dividir los apartados en componentes, por lo que cuando el usuario pida un servicio o que se visualice algún gráfico, no será necesario el refresco constante de la página si no que mostrara solo los datos que les pedimos reduciendo la carga continua que le supone a la página.



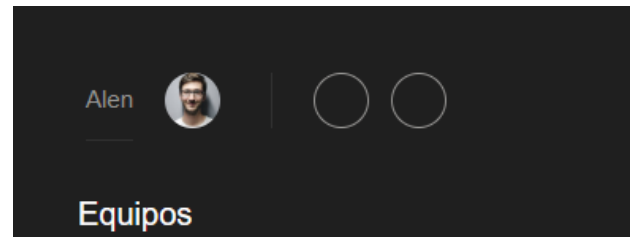
Como se puede observar en la figura tenemos un conjunto de 3 pilotos que en este caso podemos cargarlos con un endpoint que recoja los 3 pilotos que van liderando el mundial.

A la izquierda tenemos un panel de navegación que nos muestra todas las opciones de navegación que nos ofrece la página web para poder acceder.

A nuestra derecha tenemos los equipos que más han aumentado sus puntos

Cada card Piloto tiene su propio estilo y estándar de personalización además de un conjunto de eventos que se ejecutan al hacer clic como por ejemplo la navegación a la página dedicada del piloto seleccionado.

Arriba se muestra el Administrador con su correspondiente foto seleccionada cuando te registras además del nombre de la página y más elementos de navegación



### Add Driver | Añadir Piloto

Este componente está únicamente reservado para la creación de pilotos

**New User**  
Elige Foto

**About**  
Menu de Creacion de pilotos  
Introduce los datos de tu piloto y elige un aspecto para el

**Personal Details**

Name

Lastname

Age

Number

Haz click en el boton amarillo  
Añade tu Foto!

Submit

Como se puede apreciar en la parte izquierda aparece una foto de un piloto de la vida real de F1 que el administrador asignará al piloto que está creando.

Y un poco de información al respecto de cómo funciona esta página debajo del título About

En la parte derecha podemos observar que es el formulario que vamos a utilizar para añadir los datos del futuro participante, así como el nombre, apellido, edad... y entre estas un botón que te abre un modal en el que aparecen todas las imágenes que el administrador puede asignar.

Y por último el Botón Submit que añade al participante piloto.

Elige Foto



Esta imagen pertenece como resultado al botón de elegir foto.

Aquí el administrador tiene una gran variedad de pilotos de F1 que compitieron entre 2021 y 2022 con el mismo tamaño de resolución para que no haya problemas de tamaño.


Cada uno de ellos se carga encima del título New User, mostrando así la imagen que el administrador ha elegido



New User

En la parte de arriba tenemos un header, un componente del que vamos a hablar más tarde

El formulario también incluye a la hora de seleccionar equipos, un desplegable con todas las opciones viables. Hay que tener en cuenta que no se puede seleccionar un equipo que no existe, por eso debes crearlo antes.



New User

Elige Foto

About

Menu de Creacion de pilotos

Introduce los datos de tu piloto y elige un aspecto para el

Personal Details


Name

Lastname


Age

Number


Haz click en el boton amarillo  
Añade tu Foto!




Scuderia Ferrari



Fredric Vasseur



Haas F1 Team



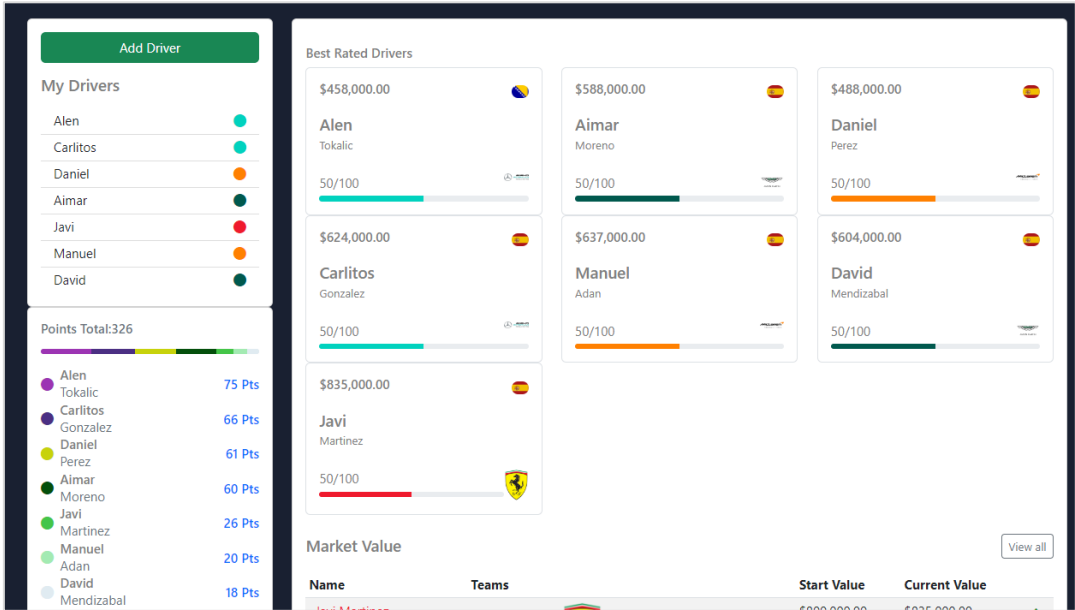
Activar Wind



Admin-drivers | Administrador de Pilotos

Este componente muestra toda la información acerca de los pilotos.

Todo lo relacionado tanto como estadística, valor o habilidad está en esta página.



Tenemos el Apartado principal que es el de la derecha, contiene los pilotos que pertenecen en esta liga, con su nivel de habilidad mostrado con una barra de progreso.

A la izquierda utilizamos una barra de progreso compartida que muestra la diferencia de puntos entre pilotos en la liga y diferenciados cada uno con un color diferente.

Y encima de esta una pequeña lista con los pilotos y los colores de su equipo.

Debajo una tabla que muestra campos como el nombre y apellidos, así como el nombre del equipo al que pertenecen, su valor de mercado al empezar la temporada, su valor actual que dependerá de su rendimiento en las carreras y como último campo, una columna más que realiza el seguimiento de su valor, si ha bajado muestra una flecha roja hacia abajo en caso contrario una flecha verde que apunta hacia arriba y en el caso de que se mantengan igual una flecha gris.

Market Value				View all
Name	Teams	Start Value	Current Value	
Javi Martinez	Scuderia Ferrari	\$800,000.00	\$835,000.00	▲
Manuel Adan	Mclaren F1 Team	\$800,000.00	\$637,000.00	▼
Carlitos Gonzalez	Mercedes-AMG F1 Team	\$400,000.00	\$624,000.00	▲
David Mendizabal	Aston Martin Aramco Cognizant	\$800,000.00	\$604,000.00	▼
Aimar Moreno	Aston Martin Aramco Cognizant	\$400,000.00	\$588,000.00	▲
Daniel Perez	Mclaren F1 Team	\$400,000.00	\$488,000.00	▲
Alen Tokalic	Mercedes-AMG F1 Team	\$400,000.00	\$458,000.00	▲

## Admin-AddTeam | Añadir Equipo

Project Teams [Añadir equipo](#)

Teams	Car	Points
 McLaren F1 Team		321
 Aston Martin Aramco Cognizant		132
 Mercedes-AMG F1 Team		0
 Scuderia Ferrari		0
 Haas F1 Team		0

Componente dedicado a la creación de un equipo o escudería

Le damos al botón añadir Equipo y nos lleva al componente o formulario de creación

**New Team**

[Elige Foto de Vehículo](#)

[Elige Foto de Escudería](#)

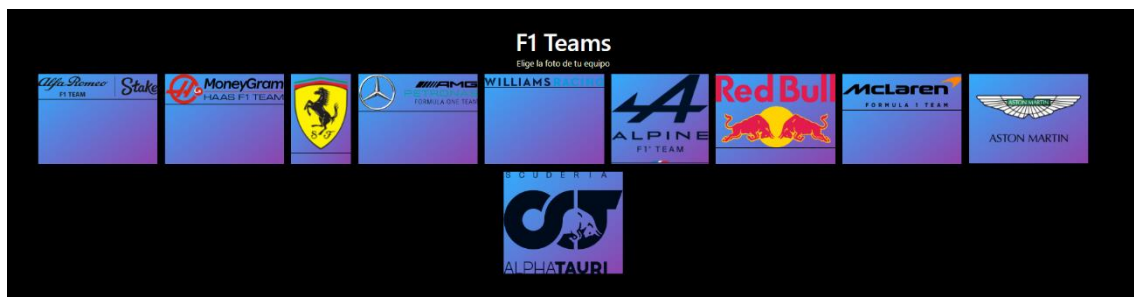
**About**

Menu de Creacion de Equipos

Introduce los datos de tu piloto y elige un aspecto para el

Similar al formulario de adición de pilotos, cuentas con dos botones para añadir una imagen, en este caso la del coche y la de la escudería.

El usuario tiene la opción de crear un equipo real de la parrilla de la F1 o puede personalizarlo a su manera, mezclando datos que a él le parezca



### Admin-AddRaces / Añadir Carreras

En este apartado se visualizan dos partes, tenemos un panel a la derecha y una tabla a la izquierda.

La función de este apartado es añadir circuitos a la liga para después celebrar carreras en él.

Los circuitos están ya introducidos en la base de datos para que cada director pueda elegir como quiere celebrar su mundial con los circuitos de su elección y con un orden personalizado.



En la figura de la izquierda se puede observar que estamos tratando con una tabla en la cual tenemos información del circuito en cuestión, con un botón rojo de eliminar.

Esto es el resultado de hacer click en la sección de la derecha encima de cualquier icono.

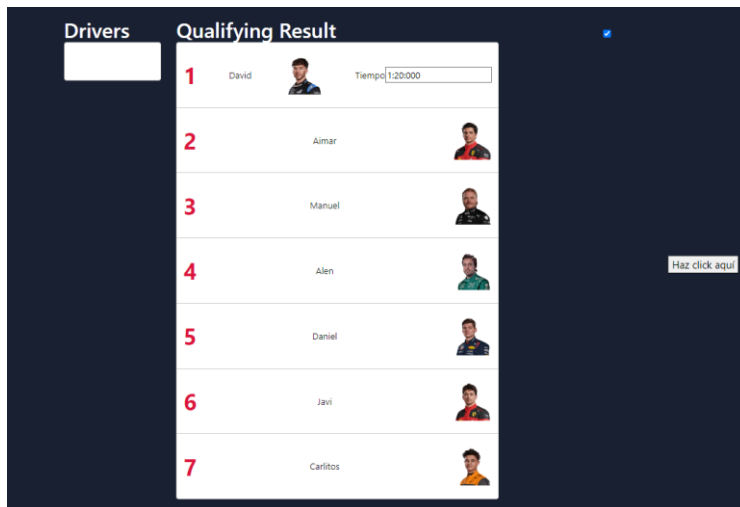
Como se puede observar tenemos un botón eliminar y un icono de ordenar, en caso de que no queramos que aparezca un circuito en la lista,

lo eliminamos

Una vez lo tenemos como queremos le damos al botón de submit se agregan los circuitos a la clase Liga para que el administrador pueda trabajar con ellas.

### Crear Qualy | Crear Clasificación

Este apartado se ocupa de crear la clasificación de carrera



En este componente tenemos dos secciones.

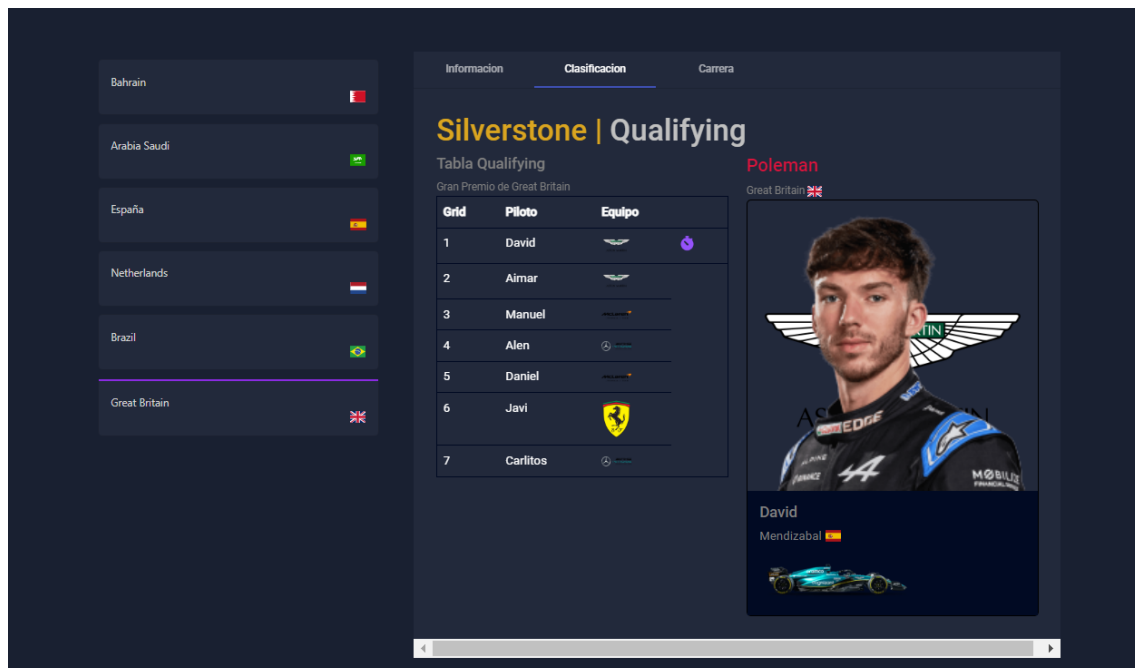
La sección de la izquierda debajo de la etiqueta Drivers despliega todos los pilotos para que el administrador se encargue de pasarlos al contenedor de la derecha, el de la etiqueta Qualifying Result.

Una vez traspasados se pueden ordenar tal como

hayan quedado en el juego.

Cuando se termina este proceso se ilumina el checkbox de arriba y se muestra en la sección derecha un componente card con la información del piloto que haya quedado 1º en la clasificación.

Este es el resultado de completar la pantalla anterior



La barra morada muestra en que ronda nos encontramos actualmente, que es la última del mundial.

Se puede observar a la izquierda que estamos en la ronda número 6 y que es la última carrera, en este caso nos ha tocado competir en Silverstone en Reino Unido y tras haber completado la pantalla anterior, nos muestra el poleman y la tabla en orden.

### Crear-Race / Crear Carrera

Este apartado se ocupa de crear el resultado de la carrera, (clase Result) en el que el director o administrador recoge los datos de la clasificación anterior y se muestran en una tabla.

Los puede manipular desplazando a los pilotos arriba y abajo, a la izquierda hay una sección que te muestra el orden de clasificación y cómo están cambiando.

Parrilla de Clasificación

Tabla Final

Piloto	Equipo	Puntos	Fastest Lap
66 Manuel Adan	Mclaren FORMULA 1 TEAM	25	<input type="checkbox"/>
55 Alen Tokalic	AMG PETRONAS FORMULA ONE TEAM	18	<input type="checkbox"/>
13 Daniel Perez	Mclaren FORMULA 1 TEAM	15	<input type="checkbox"/>
78 David Mendizabal	ASTON MARTIN	12	<input type="checkbox"/>
16 Aimar Moreno	ASTON MARTIN	10	<input type="checkbox"/>
55 Javi Martinez	Ferrari	8	<input checked="" type="checkbox"/>
45 Carltos Gonzalez	AMG PETRONAS FORMULA ONE TEAM	6	<input type="checkbox"/>

Guardar

Las columnas de la tabla muestran información del piloto, del equipo, de los puntos que obtiene al quedar en determinada posición y de la vuelta rápida que suma un punto al poseedor de dicha.

Information Classification Carrera

Silverstone | Carrera

Tabla Carrera

Gran Premio de Great Britain

Grid	Piloto	Equipo
1	Manuel	
2	Alen	
3	Daniel	
4	David	
5	Aimar	
6	Javi	
7	Carltos	

Winner

Great Britain

Manuel Adan

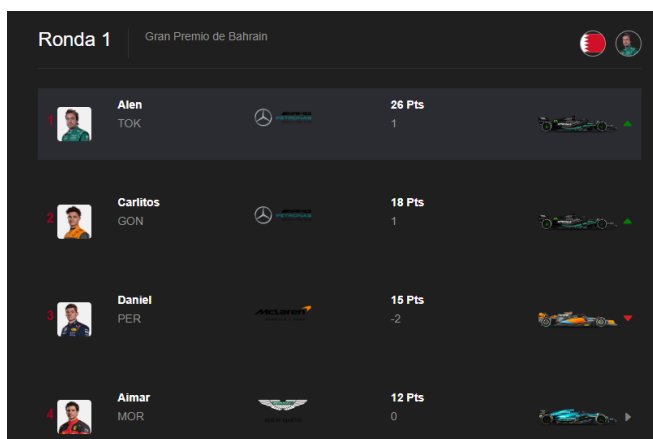
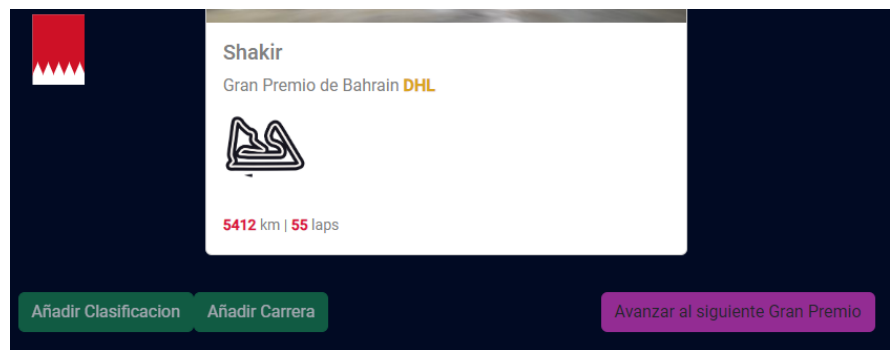
En la última columna muestra si los pilotos han sufrido pérdida de posición respecto a la salida de parrilla representado con una flecha roja hacia abajo, si es lo contrario con una flecha verde hacia arriba y si no flecha gris

Cuenta también con un checkbox que muestra la vuelta rápida.

## Pruebas Importantes

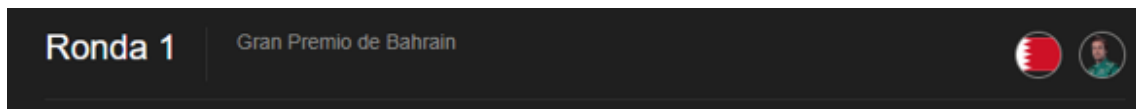
### Pantalla de admin-races

Comprobaciones en los botones Añadir Clasificación y Añadir Carrera, en caso de que exista la carrera ya terminada pasan a ser disabled y aparece un botón nuevo para pasar de ronda.



### Pantalla de admin

La página tiene endpoints que comprueban si se ha celebrado una carrera o solo su clasificación, en caso de que se haya celebrado la carrera se añade la foto del piloto al lado de la bandera del país en el que ha ganado.



### Pantalla Añadir Resultados

La página comprueba automáticamente si ha cambiado el orden desde la parrilla de salida hasta el final de la carrera, añadiendo una flecha que cambia según el cambio de posiciones.

Es decir si el piloto que sale 4º acaba 2º, aparecerá una flecha verde apuntando hacia arriba mostrando que el piloto ha conseguido acabar en una mejor posición y si no lo contrario.

Parrilla de Clasificación













	TOK	2º
	MOR	4º
	PER	1º
	GON	3º

Tabla Final

Piloto		Equipo		Puntos	Fastest Lap		
	95 Alen Tokalic		AMG PETRONAS FORMULA ONE TEAM	25	<input type="checkbox"/>	<input type="checkbox"/>	▲
	16 Aimar Moreno		ASTON MARTIN	18	<input type="checkbox"/>	<input type="checkbox"/>	▲
	13 Daniel Perez		McLaren FORMULA 1 TEAM	15	<input type="checkbox"/>	<input type="checkbox"/>	▼
	45 Carlitos Gonzalez		AMG PETRONAS FORMULA ONE TEAM	12	<input type="checkbox"/>	<input type="checkbox"/>	▼

### Ampliación y Mejoras

Este apartado es un breve recordatorio de que esta aplicación puede llegar a aumentar su funcionalidad y se pueden re implementar diversas ideas en el futuro con el fin de seguir enriqueciendo más aun la experiencia de los usuarios.

### Modo Mi Equipo

En cuanto a este apartado, el proyecto tiene un potencial de escalabilidad amplio, al ser una página web que trata de jugadores que van evolucionando se puede integrar un apartado “Mi Equipo”.

Con escalabilidad me refiero a que el proyecto puede crecer a una aplicación más amplia y compleja con posibilidad de simular el negocio y el mundo de la F1.(Lo explico detalladamente abajo).Esta característica constaría de un nuevo nivel de “realismo” y un paso más allá de ser una aplicación web.

En el modo mi equipo, los participantes contarán con un nuevo jugador (el director/jefe del equipo).

### Gestión de pilotos

Es importante para un director determinar quién es el piloto más hábil del equipo, por ello se comprueba quién de los dos ha rendido mejor en la temporada o durante para hacer posibles cambios de plantilla.

Este jugador cuenta con un panel de administración de la situación actual de su equipo.

Tiene la posibilidad de gestionar a sus pilotos de manera que compitan entre ellos por el puesto en el equipo.

El director tendrá una pestaña de pilotos donde podrá comprobar con datos reales y gráficos, el rendimiento de temporada de los pilotos de una misma escudería al igual que con los de la parrilla entera.

### Presupuesto de Equipo

En la realidad una escudería rinde mejor que otra por una causa, el dinero, pues esto es una buena idea para poder desarrollar el equipo de una manera equilibrada y poder darle a la web un objetivo que ocurre en la vida real.

Además de las carreras y premios que reciben los equipos por su trabajo. En esta web se premiarán a los equipos por su desempeño en las carreras, es decir, una mejor posición en las carreras por parte de los dos pilotos corresponde a más cantidad de capital, sin embargo un mal desempeño puede suponer ganancias insuficientes e incluso pérdidas.

Los patrocinadores también dotarán a los equipos de un capital extra, pero esto lo explico detalladamente en el siguiente apartado.



### Patrocinadores

Al comienzo de la temporada, los jefes de equipo deberán determinar con qué patrocinadores o 'sponsors' tienen que contar.

Habría dos tipos de patrocinadores: Dos principales, que permanecen con el equipo hasta el fin de la temporada y otros 4 temporales con opción de renovaciones. Los patrocinadores anuales son los que más capital ofrecen y los que piden objetivos de largo plazo, en cambio los temporales suelen durar  $\frac{3}{4}$  carreras y proporcionan objetivos a corto plazo con un capital más bajo.

Una vez que firmes con los patrocinadores, estos se sumarán a tu equipo reflejándose en la pestaña de Equipos, donde todo el mundo puede observar que patrocinadores tienes.

### Mercado de Fichajes

Un mercado de fichajes funcional en el que habrá determinados momentos de la temporada donde se podrán renegociar contratos o incluso contratar pilotos para tu equipo (Todo esto con dinero ficticio claro está).

### Gestión de Equipo

Los directores pueden influir en la estrategia de carrera, planificando en su pestaña de estrategia que neumáticos que hay que poner, las ventanas de parada de boxes, la preferencia de los pilotos.

Los directores tendrán un esquema que podrán editar en función de lo que ocurra en la carrera, aplicando una estrategia en función de la situación de carrera.

## Calculo de tiempos final

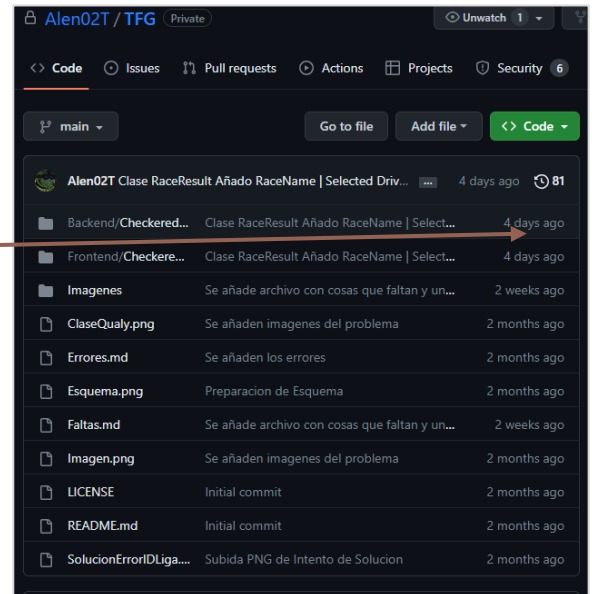
En esta sección, la estimación de cálculos de tiempos final se registrara a partir del repositorio y de la consola de servidor que guarda los minutos que lleva activa, así podremos calcular una aproximación cercana a la real.

Se puede comprobar la veracidad de esto y el seguimiento en github gracias a los commits y a la fecha de estos.

Un total de **81** commits desde abril hasta junio

Preparación de la idea y de las herramientas: **3H**

- Que es lo que quiero conseguir con esta aplicación: **1:30H**
- Búsqueda de posibles librerías que me pueden hacer el trabajo más sencillo: **2H**
- Selección de que base de datos voy a utilizar: **30min**



Diseño de la estructura del servidor y de la base de datos: **15H**

- Generación de todos los modelos y controladores del servidor: **3H**
- Reestructuración por falta de posibles atributos: **3H**
- Reestructuración de los métodos de los controladores por falta de respuestas a peticiones del usuario: **2:30H**
- Reestructuración por respuestas mal formuladas por el propio servidor: **3:30H**
- Fallos de integridad de datos en la base de datos con los datos del usuario(No se cifraba la contraseña en la base de datos): **1H**
- Fallos de mapeo en el Program.cs con DirectorService: **1H**

Diseño y creación de la interfaz web: **85H**

- Preparación del diseño y todas las rutas que va a llevar: **10H**
- Generación del interceptor de rutas y protección de rutas para el administrador: **2H**
- Descarga de todos los Assets y redimensión de las fotos de los pilotos y de los vehículos: **2H**
- Creación de los modelos y servicios que recogen los json de la API: **3H**
- Resolución de errores de los servicios por rutas confusas o rutas incorrectas: **5H**
- Creación de todos los componentes con su código HTML y su CSS correspondiente final: **30H**
- Resolución de errores de diseño de TypeScript(errores en adición de carreras y qualys): **3:30H**
- Resolución de errores con Tabla Circuitos y Tabla Liga / Problema del backend: **4H**

- Comprensión e implementación de todas las librerías como ng-select y chart.js hasta conseguir el resultado esperado: **10H**
- Adición de animaciones en ciertos componentes: **1:30H**
- Cambios de diseño en pantalla Dashboard y charts entre otras más para mostrar una versión mejorada, menos cargada y más interactiva: **10H**
- Problemas con campos undefined y nulos en las clases JOIN / Cambios en el Backend: **3:30H**

Pruebas en la aplicación: **5H:**

- Borrado de registros para comenzar de nuevo: **30min**
- Entrada con diferentes usuarios para observar si puedo acceder a los datos de cualquier otro usuario: **30min**
- Generación de endpoints para cubrir problemas de acceso de datos para los usuarios: **1:30H**
- Generación e instalación del swagger para comprobar todos los endpoints y lo que devuelve cada uno: **2:30H**

Documentación de la aplicación: **15H**

- Representación del apartado Diseño de la aplicación: **4H**
- Estructuración de la base de datos con sus esquemas y modelos: **3H**
- Preparación del esquema de la base de datos en la aplicación web ERASER: **2H**
- Preparación de los esquemas de las sendas LOGIN y REGISTER en ERASER: **4H**
- Explicación de los componentes principales y más importantes con sus capturas: **2H**
- Apartado Ampliación y Mejoras en constante actualización: **45min**
- Actualización constante de capturas para las versiones mejoradas: **30min**
- Estilización del documento: **30min**

Preparación de la presentación: **2:30H**

- Búsqueda de la herramienta que voy a utilizar: **30min**
- Preparar capturas e imágenes para añadir: **1:30min**
- Estilización del documento: **30min**

Tiempo total:

**125 horas y 30 minutos**

## Bibliografía

Documentación .NET 6

- <https://learn.microsoft.com/es-es/dotnet/>

Creación de una web API

- <https://learn.microsoft.com/es-es/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio>

Entorno Servidor, Entidades y Relaciones de Bases de datos y Seguridad JWT

- <https://www.youtube.com/@PatrickGod>

Para la preparación de ítems desplegables

- <https://www.npmjs.com/package/@ng-select/ng-select>

Para la preparación de gráficos responsivos y animados

- <https://www.chartjs.org/docs/latest/>

Para cualquier duda o problema que surgía

- <https://stackoverflow.com/>
- <https://chat.openai.com/auth/login>

Dudas de lenguajes y de diseño para poder implementar soluciones y mejoras, además de ideas.

- <https://chat.openai.com/auth/login>

Para cualquier duda de estructuración de Angular.

- <https://angular.io/docs>

Para el diseño y estructuración que llevara mi base de datos

- <https://www.kaggle.com/datasets/cjgdev/formula-1-race-data-19502017?select=races.csv>

Información general de equipos, pilotos y cualquier cosa relacionada

- <https://www.formula1.com/>

## Conclusión

Este es mi segundo trabajo con estas herramientas y siento que no he podido explotar el potencial total de las herramientas del servidor ASP NET Core y de las APIS que me han demostrado que se pueden lograr cosas que no sabía que podía implementar a mi aplicación.

A raíz de hacer un proyecto desde cero y con un margen de tiempo corto y limitado, he descubierto lo que es de verdad un desafío a contrarreloj, como a pesar de todos los errores, tanto de código como de diseño se puede dificultar la realización de la cosa más mínima que te hace retroceder varios pasos atrás y el peligro que eso conlleva.

Antes de la realización de un servidor de este calibre, tendré más en cuenta el diseño a la hora de la creación de los primeros modelos y de todas las opciones que puedo utilizar, porque pienso que esto puede y debe mejorar.

De lo que estoy contento es de la estructuración de la API, todos los endpoints funcionan correctamente y devuelven lo que se supone que deben devolver, además de poder utilizarlos todas las veces que quiera y devolver respuestas personalizadas al usuario es algo que simplemente me alegra haber aprendido.

En cuanto a la parte de Angular, me hubiera gustado cambiar la estructura pero no tenía el tiempo suficiente para ello y para cuando sabía de estas técnicas de organización ya era demasiado tarde.

He aprendido muchas cosas en la empresa que me hubieran gustado haber implementado en la web, así como directivas, pipes y paginas dinámicas. Si hay una próxima vez procurare ser más delicado con la organización de componente y la utilización más continua de componentes padre e hijo, además de pasar parámetros a componentes sin tener que repetir mismo código una y otra vez en distintos componentes.

Me quedo con lo que he aprendido en este segundo gran proyecto, sin duda lo tendré guardado en el repositorio como recordatorio de lo que una vez logre y de lo que significa el esfuerzo y el sacrificio que le he dedicado a pesar de ciertos problemas que una vez tuve.