

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the text "[Fecha]".

[Fecha]

Proyecto de final de Ciclo

Gestión de pistas deportivas

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Gil Carlos Hermoso Sánchez
IES FERNANDO III

Especificaciones del Proyecto

Descripción General

Gilca's Sport Center es una plataforma web completa para la gestión de reservas de instalaciones deportivas. El proyecto consta de dos partes principales: el backend y el front.

Backend

El backend de la aplicación es un servidor construido con Node.js y Express.js, que proporciona una API RESTful para interactuar con la base de datos MongoDB. El backend se encarga de las siguientes tareas:

1. **Gestión de Usuarios:** Permite la creación, lectura, actualización y eliminación de usuarios. También se encarga de la autenticación y autorización de usuarios.
2. **Gestión de Deportes:** Permite la creación, lectura, actualización y eliminación de deportes disponibles en el centro deportivo.
3. **Gestión de Pistas:** Permite la creación, lectura, actualización y eliminación de pistas disponibles para reservar en el centro deportivo.
4. **Gestión de Reservas:** Permite la creación, lectura, actualización y eliminación de reservas realizadas por los usuarios.

Frontend

El frontend de la aplicación es una interfaz de usuario construida con Angular, que permite a los usuarios interactuar con el backend de manera fácil e intuitiva. El frontend se encarga de las siguientes tareas:

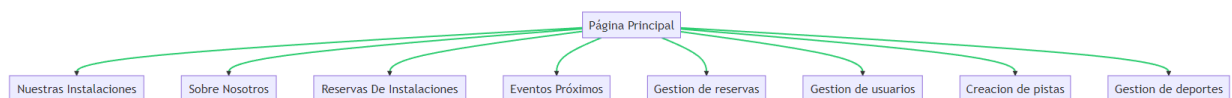
1. **Navegación:** Proporciona una navegación fluida entre las diferentes secciones de la aplicación, incluyendo la página de inicio, las instalaciones, sobre nosotros, reservas de instalaciones, eventos próximos, clases de tenis, nuestro gimnasio y noticias y actualizaciones.
2. **Interacción con el Backend:** Realiza solicitudes al backend para obtener, enviar, actualizar y eliminar datos. Esto incluye la gestión de usuarios, deportes, pistas y reservas.
3. **Autenticación y Autorización:** Gestiona el inicio de sesión y el registro de usuarios, así como la autorización para ciertas acciones basadas en el rol del usuario.
4. **Presentación de Datos:** Presenta los datos obtenidos del backend de una manera fácil de entender y atractiva visualmente.

En resumen, el proyecto Gilca's Sport Center consiste en desarrollar una plataforma web completa para la gestión de reservas de instalaciones deportivas, que incluye un backend para la gestión de datos y un frontend para la interacción del usuario.

Elementos innovadores

En este proyecto he realizado diferentes tareas no vistas anteriormente en clase como el manejo de correos desde el back, despliegue de nuestro back, uso de nuevos pipes para la gestión de peticiones desde el front con el uso de la librería **rxjs** y otras nuevas funciones incluidas en el back.

Prototipo y diagrama de como se estructuraría la web



Documentación del Backend

a. Tecnologías, herramientas y lenguajes usados

El proyecto utiliza **Node.js** como su principal tecnología de backend. Node.js es un entorno de ejecución de JavaScript que permite ejecutar JavaScript en el servidor.

Además, el proyecto utiliza varias bibliotecas y módulos de Node.js, a continuación enumero cada una de ellas junto con una breve explicación de su funcionamiento:

1. **bcryptjs**: Esta es una biblioteca de Node.js que se utiliza para hashear y verificar contraseñas. Es una versión optimizada de bcrypt en JavaScript puro, con cero dependencias.
2. **cors**: Este módulo es un middleware de Node.js que se utiliza para habilitar CORS (Cross-Origin Resource Sharing) en nuestras aplicaciones Express. CORS es una especificación que permite a nuestra aplicación hacer solicitudes AJAX a otro dominio.
3. **dotenv**: Este módulo se utiliza para cargar variables de entorno desde un archivo .env en process.env. Esto es útil para ocultar información sensible como las claves de la API, las contraseñas de la base de datos, etc.
4. **express**: Express es un marco de aplicación web para Node.js que proporciona un conjunto robusto de características para las aplicaciones web y móviles. Es muy utilizado para construir aplicaciones web en Node.js debido a su simplicidad y flexibilidad.

5. **express-validator:** Este es un conjunto de middlewares de validación y saneamiento para Express. Permite validar los datos de las solicitudes entrantes en el lado del servidor.
6. **jsonwebtoken:** Esta biblioteca se utiliza para trabajar con JSON Web Tokens (JWT). Los JWT son una forma abierta de representar reclamaciones que se pueden transferir entre dos partes.
7. **mongoose:** Mongoose es una biblioteca de Node.js que proporciona una solución de modelado de objetos para trabajar con MongoDB. Proporciona una forma sencilla de definir esquemas para modelar los datos de nuestra aplicación.
8. **nodemailer:** Este es un módulo que se utiliza para enviar correos electrónicos desde una aplicación Node.js. Soporta diferentes transportes de correo y tiene muchas características para enviar correos electrónicos.

b. Fundamentos de las tecnologías utilizadas

Node.js utiliza el motor V8 de Google para ejecutar código JavaScript. Esto permite a los desarrolladores escribir código de servidor en JavaScript, un lenguaje que originalmente se desarrolló para el lado del cliente en la web. Node.js es asíncrono y basado en eventos, lo que significa que puede manejar múltiples solicitudes al mismo tiempo sin bloquear el hilo principal.

c. Diseño de la base de datos

Como base de datos he utilizado MongoDB que es una base de datos NoSQL orientada a documentos y como servicio de alojamiento en la nube he utilizado MongoDB Atlas que es un servicio de base de datos en la nube para MongoDB, por lo que no necesitas gestionar la infraestructura de tu base de datos.

A continuación dejo una información algo más detallada de ambos:

MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, lo que significa que almacena datos en un formato similar a JSON (en realidad BSON, una versión binaria de JSON). En lugar de almacenar los datos en tablas como se hace en las bases de datos SQL, MongoDB puede almacenarlos en colecciones flexibles de documentos.

Cada documento puede tener su propio conjunto único de campos de pares clave-valor, la estructura de los cuales puede cambiar con el tiempo. Los campos pueden variar de un

documento a otro y la estructura de datos puede cambiar con el tiempo. Esto proporciona una gran flexibilidad a la hora de manejar los datos.

MongoDB Atlas

MongoDB Atlas es un servicio de base de datos en la nube para MongoDB, por lo que no necesitas gestionar la infraestructura de tu base de datos. Proporciona todas las características de MongoDB, sin la complejidad operativa, lo que permite el escalado automático y el equilibrio de carga.

Con MongoDB Atlas, puedes configurar clusters de MongoDB en la nube, lo que te permite almacenar tus datos de manera segura y acceder a ellos desde cualquier lugar. Atlas tiene una serie de características de seguridad incorporadas, como el cifrado en reposo, la autenticación y el control de acceso. También se encarga de todas las tareas de administración de la base de datos, como el escalado, las copias de seguridad y la recuperación.

En resumen, MongoDB Atlas te permite centrarte en el desarrollo de tu aplicación sin tener que preocuparte por la gestión y administración de tu base de datos.

Estructura

La estructura que tendrán los documentos que se guardaran en mi base de datos serán los siguiente:

1. Pista (Archivo: pista.js)

- **reservaSchema:** Este esquema define una reserva con los siguientes campos:
 - **fecha:** Fecha de la reserva (tipo: Date, requerido: true).
 - **hora:** Hora de la reserva (tipo: String, requerido: true).
 - **usuario:** Usuario que realiza la reserva (tipo: String, requerido: true).
 - **pista:** Referencia a la pista reservada (tipo: ObjectId, ref: 'Pista', requerido: true).
 - **cancelada:** Indica si la reserva ha sido cancelada (tipo: Boolean, default: false).
- **deporteSchema:** Este esquema define un deporte con los siguientes campos:
 - **nombre:** Nombre del deporte (tipo: String, requerido: true).
 - **icono:** Icono del deporte (tipo: String).
- **horarioSchema:** Este esquema define un horario con los siguientes campos:

- **Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo:** Horarios para cada día de la semana (tipo: Array de String).
- **pistaSchema:** Este esquema define una pista con los siguientes campos:
 - **nombre:** Nombre de la pista (tipo: String, requerido: true).
 - **horariosDisponibles:** Horarios disponibles para la pista (tipo: horarioSchema).
 - **reservas:** Reservas realizadas en la pista (tipo: Array de reservaSchema).
 - **deporte:** Deporte para el que se utiliza la pista (tipo: ObjectId, ref: 'Deporte', requerido: true).

2. Usuario (Archivo: usuario.js)

- **UsuarioSchema:** Este esquema define un usuario con los siguientes campos:
 - **nombre:** Nombre del usuario (tipo: String, requerido: true, único: true).
 - **correo:** Correo electrónico del usuario (tipo: String, requerido: true, único: true).
 - **password:** Contraseña del usuario (tipo: String, requerido: true).
 - **img:** Imagen del usuario (tipo: String).
 - **rol:** Rol del usuario (tipo: String, requerido: true, enum: ['ADMIN_ROLE', 'USER_ROLE', 'VENTAS_ROLE']).
 - **estado:** Estado del usuario (tipo: Boolean, default: true).
 - **google:** Indica si el usuario se registró con Google (tipo: Boolean, default: false).

3. Role (Archivo: role.js)

- **RoleSchema:** Este esquema define un rol con el siguiente campo:

- **rol**: Rol del usuario (tipo: String, requerido: true).

d. Documentación de los servicios CRUD

Los servicios Crud mi proyecto están repartidos de forma ordenada por los diferente archivos que este tiene. A continuación tienes una explicación detallada de las rutas y controladores que a los que estas llevan según el tipo de dato

- Rutas y controladores para las pistas.

Rutas (/api/pistas):

1. **GET /**: Esta ruta no realiza ninguna comprobación antes de llamar al controlador. Llama al controlador **getAllPistas**.

2. **GET /:id:** Esta ruta recibe un ID como parámetro en la ruta. Llama al controlador **getPistaById** con el ID proporcionado.
3. **POST /:** Esta ruta recibe en el cuerpo de la petición los datos de la nueva pista. Llama al controlador **pistaPost** con los datos proporcionados.
4. **POST /:id/reserva:** Esta ruta recibe un ID como parámetro en la ruta y en el cuerpo de la petición recibe los datos de la nueva reserva. Llama al controlador **hacerReserva** con el ID y los datos proporcionados.
5. **PUT /:id/horarios:** Esta ruta recibe un ID como parámetro en la ruta y en el cuerpo de la petición recibe los nuevos horarios. Llama al controlador **horariosPut** con el ID y los horarios proporcionados.
6. **GET /:deporte/:fecha:** Esta ruta recibe el ID del deporte y la fecha como parámetros en la ruta. Llama al controlador **getPistasByDeporte** con el ID del deporte y la fecha proporcionados.
7. **GET /deportes/reservas/:idDeporte:** Esta ruta recibe el ID del deporte como parámetro en la ruta. Llama al controlador **getPistasByDeporte** con el ID del deporte proporcionado.
8. **DELETE /deletePista/:id:** Esta ruta recibe un ID como parámetro en la ruta. Llama al controlador **deletePista** con el ID proporcionado.
9. **POST /deleteReservas:** Esta ruta recibe en el cuerpo de la petición un array de IDs de reservas que se quieren eliminar. Llama al controlador **deleteReservas** con los IDs proporcionados.
10. **GET /reservas/verReservas/allReservas:** Esta ruta no realiza ninguna comprobación antes de llamar al controlador. Llama al controlador **obtenerReservas**.
11. **DELETE /reservas/:id:** Esta ruta recibe un ID como parámetro en la ruta. Llama al controlador **eliminarReserva** con el ID proporcionado.

Controladores:

1. **getAllPistas:** Este controlador no recibe ningún parámetro. Se conecta a la base de datos y recupera todas las pistas almacenadas en ella. Devuelve un array con todas las pistas.
2. **getPistaById:** Recibe como parámetro el ID de la pista que se quiere obtener. Este parámetro es necesario. Se conecta a la base de datos y busca la pista que corresponde al ID proporcionado. Devuelve la pista encontrada.
3. **pistaPost:** Recibe en el cuerpo de la petición los siguientes parámetros: **nombre**, **horariosDisponibles**, **deporte**. Todos estos parámetros son necesarios. Crea una nueva pista con los datos proporcionados y la guarda en la base de datos. Devuelve la pista que se ha creado.
4. **hacerReserva:** Recibe el ID de la pista en los parámetros de la ruta y en el cuerpo de la petición recibe **fecha**, **hora**, **usuario**. Todos estos parámetros son necesarios. Crea una nueva reserva con los datos proporcionados y la añade a la pista correspondiente en la

base de datos. Además, envía un correo electrónico de confirmación de la reserva al usuario. Devuelve la reserva que se ha creado.

5. **horariosPut:** Recibe el ID de la pista en los parámetros de la ruta y en el cuerpo de la petición recibe **horarios**, que es un objeto con los horarios para cada día de la semana. Todos estos parámetros son necesarios. Actualiza los horarios de la pista correspondiente en la base de datos con los horarios proporcionados. Devuelve la pista con los horarios actualizados.
6. **getPistasByDeporte:** Recibe el ID del deporte en los parámetros de la ruta. Este parámetro es necesario. Se conecta a la base de datos y busca las pistas que corresponden al deporte proporcionado. Devuelve un array con las pistas encontradas.
7. **deletePista:** Recibe el ID de la pista que se quiere eliminar en los parámetros de la ruta. Este parámetro es necesario. Se conecta a la base de datos y elimina la pista que corresponde al ID proporcionado. No devuelve nada.
8. **deleteReservas:** Recibe en el cuerpo de la petición un array de IDs de reservas que se quieren eliminar. Este parámetro es necesario. Se conecta a la base de datos y elimina las reservas que corresponden a los IDs proporcionados. No devuelve nada.
9. **obtenerReservas:** Este controlador no recibe ningún parámetro. Se conecta a la base de datos y recupera todas las reservas almacenadas en ella. Devuelve un array con todas las reservas.
10. **eliminarReserva:** Recibe el ID de la reserva que se quiere eliminar en los parámetros de la ruta. Este parámetro es necesario. Se conecta a la base de datos y elimina la reserva que corresponde al ID proporcionado. No devuelve nada.

- Rutas y controladores para los usuarios.

Rutas (/api/usuarios):

1. **GET /:** Esta ruta no realiza ninguna comprobación antes de llamar al controlador. Llama al controlador **usuariosGet**.
2. **PUT /:id:** Esta ruta recibe un ID como parámetro en la ruta. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del ID y la existencia del usuario. Llama al controlador **usuariosPut** con el ID proporcionado.
3. **POST /:** Esta ruta recibe en el cuerpo de la petición los datos del nuevo usuario. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del nombre, correo, password y rol del usuario. Llama al controlador **usuariosPost** con los datos proporcionados.
4. **DELETE /:id:** Esta ruta recibe un ID como parámetro en la ruta. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del JWT, la verificación del rol del usuario y la existencia del usuario. Llama al controlador **usuariosDelete** con el ID proporcionado.

Controladores:

1. **usuariosGet:** Este controlador no recibe ningún parámetro. Se conecta a la base de datos y recupera todos los usuarios cuyo estado es true. Devuelve un array con todos los usuarios.
2. **usuariosPost:** Recibe en el cuerpo de la petición los siguientes parámetros: **nombre**, **correo**, **password**, **rol**. Todos estos parámetros son necesarios. Crea un nuevo usuario con los datos proporcionados, encripta la contraseña y lo guarda en la base de datos. Devuelve el usuario que se ha creado.
3. **usuariosPut:** Recibe el ID del usuario en los parámetros de la ruta y en el cuerpo de la petición recibe los datos que se quieren actualizar. Si se proporciona una nueva contraseña, la encripta antes de actualizar el usuario. Devuelve el usuario actualizado.
4. **usuariosDelete:** Recibe el ID del usuario en los parámetros de la ruta. Actualiza el estado del usuario a false en lugar de eliminarlo de la base de datos. Devuelve el usuario cuyo estado se ha actualizado y el usuario que realizó la petición.

- Rutas y controladores para los deportes.

Rutas:

1. **GET /:** Esta ruta no realiza ninguna comprobación antes de llamar al controlador. Llama al controlador **deportesGet**.
2. **POST /:** Esta ruta recibe en el cuerpo de la petición los datos del nuevo deporte. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del nombre del deporte. Llama al controlador **deportePost** con los datos proporcionados.
3. **DELETE /:id:** Esta ruta recibe un ID como parámetro en la ruta. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del ID del deporte. Llama al controlador **eliminarDeporte** con el ID proporcionado.
4. **PATCH /:id:** Esta ruta recibe un ID como parámetro en la ruta. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del ID del deporte. Llama al controlador **actualizarDeporte** con el ID proporcionado.

Controladores:

1. **deportePost:** Recibe en el cuerpo de la petición los siguientes parámetros: **nombre**, **icono**. Si no se proporciona un icono, se asigna un icono por defecto. Crea un nuevo

- deporte con los datos proporcionados y lo guarda en la base de datos. Devuelve el deporte que se ha creado.
2. **deportesGet:** Este controlador no recibe ningún parámetro. Se conecta a la base de datos y recupera todos los deportes. Devuelve un array con todos los deportes.
 3. **eliminarDeporte:** Recibe el ID del deporte en los parámetros de la ruta. Primero elimina todas las pistas asociadas al deporte y luego elimina el deporte de la base de datos. Devuelve el deporte que se ha eliminado.
 4. **actualizarDeporte:** Recibe el ID del deporte en los parámetros de la ruta y en el cuerpo de la petición recibe los datos que se quieren actualizar. Actualiza el deporte con los datos proporcionados. Devuelve el deporte actualizado.

- Rutas y controladores para el registro y login de usuario.

Rutas:

1. **POST /login:** Esta ruta recibe en el cuerpo de la petición los datos de inicio de sesión del usuario. Realiza una serie de comprobaciones antes de llamar al controlador, incluyendo la validación del correo y la contraseña. Llama al controlador **login** con los datos proporcionados.
2. **GET /renew:** Esta ruta no recibe ningún parámetro en la petición. Realiza una comprobación del JWT antes de llamar al controlador. Llama al controlador **revalidarToken**.

Controladores:

1. **login:** Recibe en el cuerpo de la petición los siguientes parámetros: **correo**, **password**. Todos estos parámetros son necesarios. Verifica si el correo existe en la base de datos, si el usuario está activo y si la contraseña es correcta. Si todo es correcto, genera un JWT y lo devuelve junto con los datos del usuario.
2. **revalidarToken:** Este controlador no recibe ningún parámetro. Extrae el **uid**, **name** y **rol** del JWT de la petición. Genera un nuevo JWT y lo devuelve junto con los datos del usuario.

Middlewares:

Los middlwares que he creado para validar parámetros son los siguientes:

1. **validar-campos.js:** Este middleware se encarga de validar los campos de las peticiones. Utiliza la librería **express-validator** para realizar las validaciones. Si encuentra algún error en las validaciones, devuelve un error 400 con los detalles de los errores.
2. **validar-jwt.js:** Este middleware se encarga de validar el JWT (JSON Web Token) de las peticiones. Extrae el token de los headers de la petición y lo verifica utilizando la

librería **jsonwebtoken**. Si el token es válido, extrae el **uid**, **nombre** y **rol** del token y los añade a la petición. Si el token no es válido, devuelve un error 401. Además, verifica si el usuario existe en la base de datos y si su estado es **true**.

3. **validar-roles.js**: Este middleware contiene dos funciones: **esAdminRole** y **tieneRole**. **esAdminRole** verifica si el usuario tiene el rol de administrador. Si no lo tiene, devuelve un error 401. **tieneRole** verifica si el usuario tiene uno de los roles especificados. Si no lo tiene, devuelve un error 401. Ambas funciones extraen el rol del usuario de la petición, que debe haber sido añadido por el middleware **validar-jwt**.
4. **validar-deportes.js**: Este middleware contiene una función llamada **validarDeporte** que se encarga de validar si el deporte proporcionado en el cuerpo de la solicitud es válido. Si el deporte no se proporciona o si el nombre del deporte no se proporciona, se devuelve un error. Este middleware se puede utilizar para asegurarse de que todas las solicitudes que implican deportes contengan un deporte válido.
5. **validar-pistas.js**: Este archivo contiene varios middlewares que se utilizan para validar las pistas:
 1. **validateHorario**: Este middleware se utiliza para validar los horarios proporcionados en el cuerpo de la solicitud. Asegura que los horarios sean un arreglo y que cada horario sea una cadena de caracteres que cumpla con el formato hh:mm-hh:mm.
 2. **checkPistaName**: Este middleware se utiliza para comprobar si el nombre de la pista proporcionado en el cuerpo de la solicitud ya existe en la base de datos. Si el nombre de la pista ya existe, se devuelve un error.
 3. **overlappingHorarios**: Este middleware se utiliza para comprobar si los horarios proporcionados en el cuerpo de la solicitud se solapan con los horarios existentes en la base de datos. Si hay solapamiento, se devuelve un error.
 4. **validarDeporte**: Este middleware se utiliza para validar si el deporte proporcionado en el cuerpo de la solicitud es válido. Si el deporte no se proporciona o si el nombre del deporte no se proporciona, se devuelve un error.
 5. **validarFecha**: Este middleware se utiliza para validar si la fecha proporcionada en los parámetros de la solicitud es válida. Si la fecha no se proporciona o si la fecha es anterior a la fecha actual, se devuelve un error.
6. **validar-reserva.js**: Este middleware contiene una función llamada **validarReserva** que se encarga de validar si la reserva proporcionada en el cuerpo de la solicitud es válida. Comprueba si se han proporcionado todos los datos necesarios para realizar la reserva, si la fecha y hora de la reserva no son anteriores a la actual, si la pista está disponible en el horario deseado y si la pista ya está reservada para la fecha y hora especificadas. Si alguna de estas comprobaciones falla, se devuelve un error.

Helpers:

Los helpers que he creado como ayuda para procesar y trabajar con la data son los siguiente:

1. **db-validators.js**: Este archivo contiene varias funciones de ayuda para validar datos en la base de datos. Las funciones incluyen **esRoleValido** para validar si un rol existe en la base de datos, **emailExiste** para verificar si un correo electrónico ya está registrado, **existeUsuarioPorId** para verificar si un usuario existe en la base de datos por su ID, **existePistaPorId** para verificar si una pista existe en la base de datos por su ID, y **existeDeportePorId** para verificar si un deporte existe en la base de datos por su ID.
2. **generar-jwt.js**: Este archivo contiene una función **generarJWT** que se utiliza para generar un JWT. La función toma tres parámetros: **uid**, **nombre**, y **rol**, y devuelve una nueva promesa. La promesa resuelve con el token si se genera correctamente, y rechaza con un error si algo sale mal.

Documentación del Front

El front es una aplicación web construida con Angular para la gestión de reservas de pistas deportivas. Se conecta con el backend para proporcionar una interfaz de usuario completa para la gestión de reservas.

a. Tecnologías y Dependencias

Angular: Angular es un marco de trabajo para la construcción de aplicaciones web en TypeScript. Es desarrollado y mantenido por Google. Angular es conocido por su poderosa capacidad para construir aplicaciones de una sola página (SPA) y proporciona una estructura clara y coherente para el desarrollo de aplicaciones web front-end.

Las dependencias utilizadas son:

@angular/animations: Este paquete proporciona herramientas para construir animaciones animadas y más complejas en Angular. Las animaciones pueden mejorar la experiencia del usuario al proporcionar retroalimentación visual y hacer que la aplicación parezca más pulida y profesional.

@angular/cdk: El Component Dev Kit (CDK) es un conjunto de herramientas que implementa patrones comunes de interacción, ofreciendo más flexibilidad que los componentes de Angular Material. Proporciona una manera de reutilizar código y capacidades sin estar atado a cualquier material de diseño específico.

@angular/common: Este paquete contiene muchos de los elementos comunes que se utilizan en las aplicaciones de Angular, como pipes y directivas.

@angular/compiler: El compilador de Angular es lo que toma tu código TypeScript y lo convierte en JavaScript que el navegador puede entender.

@angular/core: Este paquete es el núcleo de Angular. Contiene el código que hace que Angular sea Angular, incluyendo el sistema de inyección de dependencias, los decoradores de componentes y directivas, y el cambio de detección.

@angular/forms: Este paquete proporciona herramientas para construir formularios en Angular. Los formularios son una parte esencial de cualquier aplicación web, y este paquete hace que sea fácil construir formularios robustos y fiables.

@angular/material: Angular Material es una biblioteca de componentes de interfaz de usuario para Angular. Proporciona una serie de componentes preconstruidos que siguen las directrices de diseño de Material Design.

@angular/platform-browser: Este paquete proporciona servicios que son esenciales para lanzar y ejecutar una aplicación de navegador.

@angular/platform-browser-dynamic: Este paquete proporciona herramientas para compilar y ejecutar dinámicamente la aplicación en el navegador.

@angular/router: Este paquete proporciona un servicio de enrutamiento para Angular. El enrutamiento es esencial para construir aplicaciones de una sola página, ya que permite navegar entre diferentes componentes y vistas sin recargar la página.

rxjs: RxJS es una biblioteca para la programación reactiva utilizando Observables. Facilita la composición de código asíncrono o basado en callback.

sweetalert2: SweetAlert2 es una biblioteca de JavaScript para crear alertas personalizadas. Es una excelente alternativa a las alertas de JavaScript estándar, ya que proporciona alertas mucho más atractivas y personalizables.

tslib: Esta es una biblioteca de tiempo de ejecución para ayudantes de TypeScript. En lugar de incluir los ayudantes de TypeScript en cada archivo que los necesita, puedes importarlos de tslib para reducir el tamaño del código.

zone.js: Zone.js es una biblioteca que proporciona un contexto de ejecución que persiste a través de las tareas asíncronas. Angular utiliza Zone.js para activar su cambio de detección.

b. Diseño de la Aplicación

Diseño de la Aplicación

La aplicación web de GestionDeportivaFront presenta una interfaz de usuario limpia y fácil de usar con varias secciones:

1. **Página de inicio:** La página de inicio presenta una visión general de Gilca's Sport Center. Proporciona enlaces a varias otras secciones de la aplicación, incluyendo

"Nuestras Instalaciones", "Sobre Nosotros", "Reservas De Instalaciones", y "Eventos Próximos".

2. **Nuestras Instalaciones:** Esta sección proporciona información detallada sobre las instalaciones deportivas disponibles en Gilca's Sport Center.
3. **Sobre Nosotros:** Esta sección proporciona información sobre Gilca's Sport Center, incluyendo su misión, visión, y equipo.
4. **Reservas De Instalaciones:** Esta sección permite a los usuarios hacer reservas para las instalaciones deportivas. Los usuarios pueden seleccionar la instalación que desean reservar, elegir una fecha y hora, y completar su reserva.
5. **Eventos Próximos:** Esta sección muestra los próximos eventos que se celebrarán en Gilca's Sport Center.
6. **Gestión de reservas:** Esta sección proporciona información sobre las reservas y la capacidad de poder gestionarlas fácilmente, solo puede acceder un usuario con permisos en la app.
7. **Creación de pistas:** Esta sección permite crear pistas fácilmente, solo pueden acceder usuarios con permisos a la vista.
8. **Gestión de deportes:** Esta sección permite realizar todas las operaciones Crud sobre los deportes, solo pueden acceder usuarios con permisos en la app.
9. **gestión de usuarios:** Esta sección permite gestionar los usuarios que pueden acceder a las opciones de gestión de la app, solo pueden acceder aquellos usuarios que tengan rol de administrador.

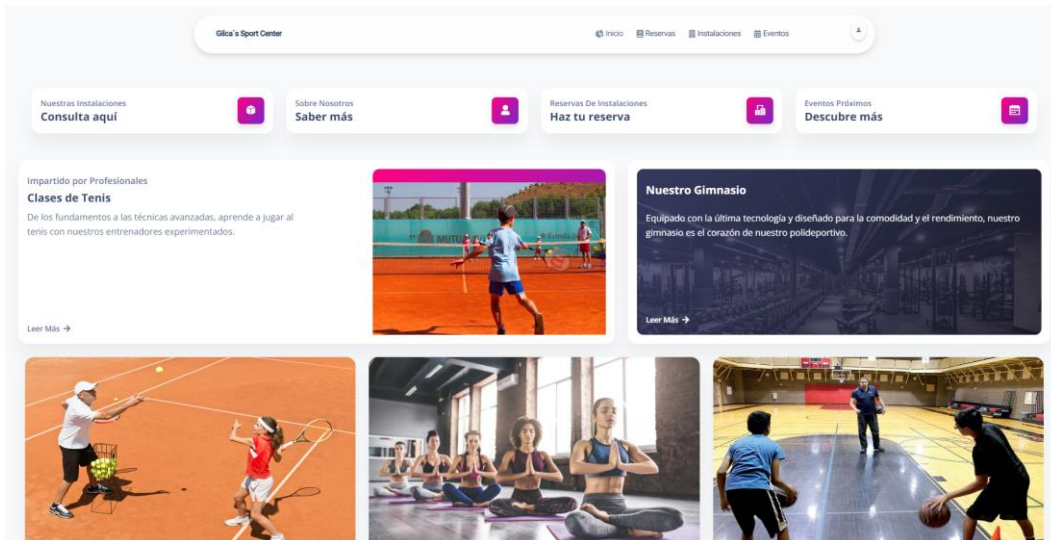
La navegación entre estas secciones es sencilla y directa, con enlaces a cada sección disponibles en la página de inicio. Los usuarios pueden navegar a cualquier sección con un solo clic.

El diseño de la aplicación es moderno y atractivo, con una paleta de colores suave y una tipografía clara. Las imágenes de alta calidad y los iconos intuitivos mejoran la experiencia del usuario y hacen que la navegación por la aplicación sea agradable y fácil.

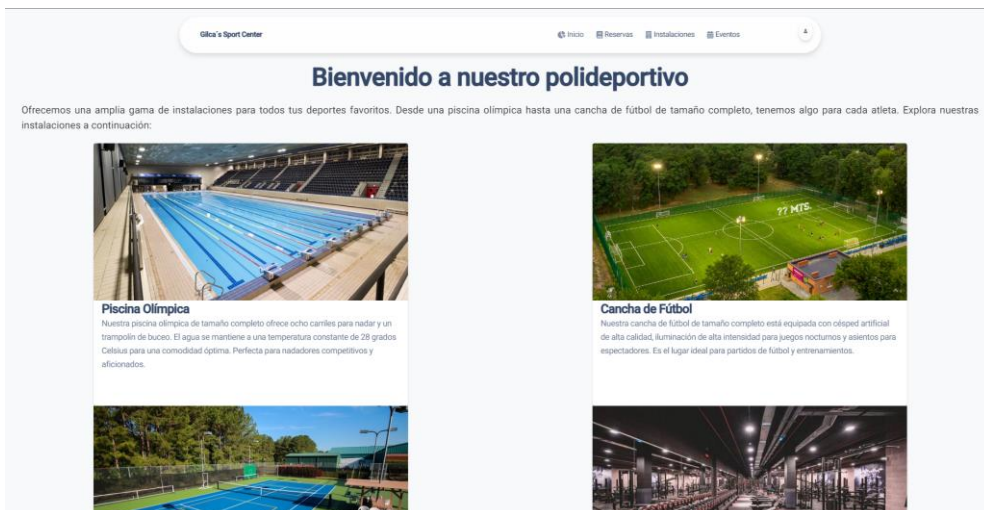
c. Mapa Web

El mapa web de la aplicación es el siguiente:

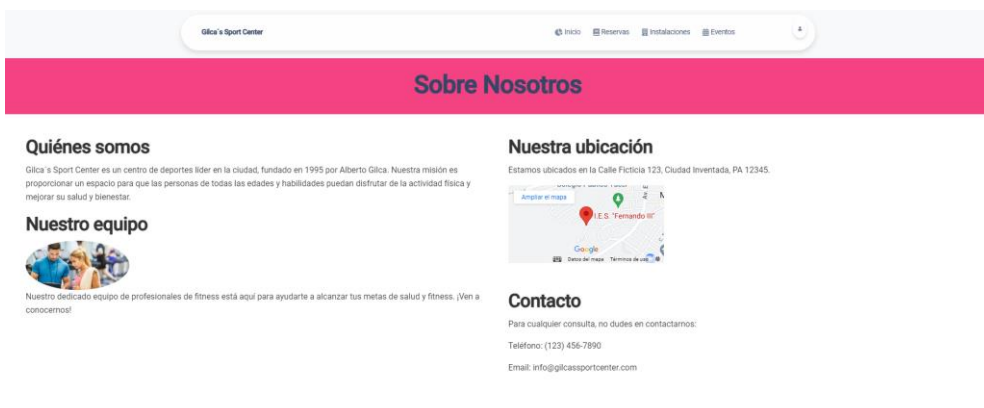
- **Inicio**
 - Descripción general de la aplicación
 - Enlaces a las demás secciones



- **Nuestras Instalaciones**
 - Detalles de las instalaciones deportivas disponibles

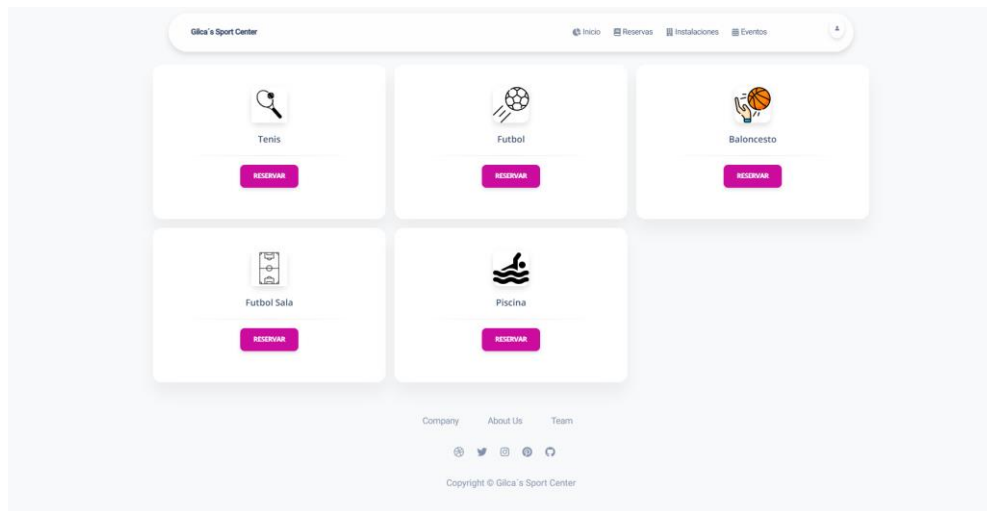


- **Sobre Nosotros**
 - Información sobre la misión, visión y equipo de la aplicación



- **Reservas De Instalaciones**

- Funcionalidad para hacer reservas para las instalaciones deportivas y gestionar pistas siempre que se tenga permisos



- **Selección de pista**
 - Funcionalidad para seleccionar pistas y hacer reservas.




- **Edición de pista**
 - Funcionalidad para editar y eliminar la pista siempre y cuando se tengas permisos. Se accede a través de la vista de selección de pista

Gilca's Sport Center

[Inicio](#)[Reservas](#)[Instalaciones](#)[Eventos](#)[Administración](#)

Editar Pista De Tenis 1



Pista De Tenis 1

ELIMINAR PISTA

Horarios disponibles para Pista de Tenis 1

Lunes

09:00-10:00

ELIMINAR

11:00-12:00

ELIMINAR

Martes

08:00-09:00

ELIMINAR

10:00-11:00

ELIMINAR

- **Eventos Próximos**
 - Información sobre los próximos eventos


Gilca's Sport Center

[Inicio](#)[Reservas](#)[Instalaciones](#)[Eventos](#)[Administración](#)

Próximos eventos


Bienvenido a nuestra página de eventos. Aquí encontrarás todos los próximos eventos que se celebrarán en nuestro polideportivo. Desde torneos deportivos hasta clases de fitness y eventos comunitarios, siempre hay algo emocionante sucediendo aquí.

Eventos destacados




Torneo de Natación

Únete a nosotros para el Torneo de Natación anual. Este emocionante evento contará con nadadores de todo el país compitiendo en una serie de eventos. No te pierdas la oportunidad de ver a algunos de los mejores talentos de la natación en acción.




Competencia de Fútbol

Ven a ver a los equipos locales competir en nuestra Competencia de Fútbol anual. Este torneo promete mucha emoción y talento, con equipos de toda la región compitiendo por el título.



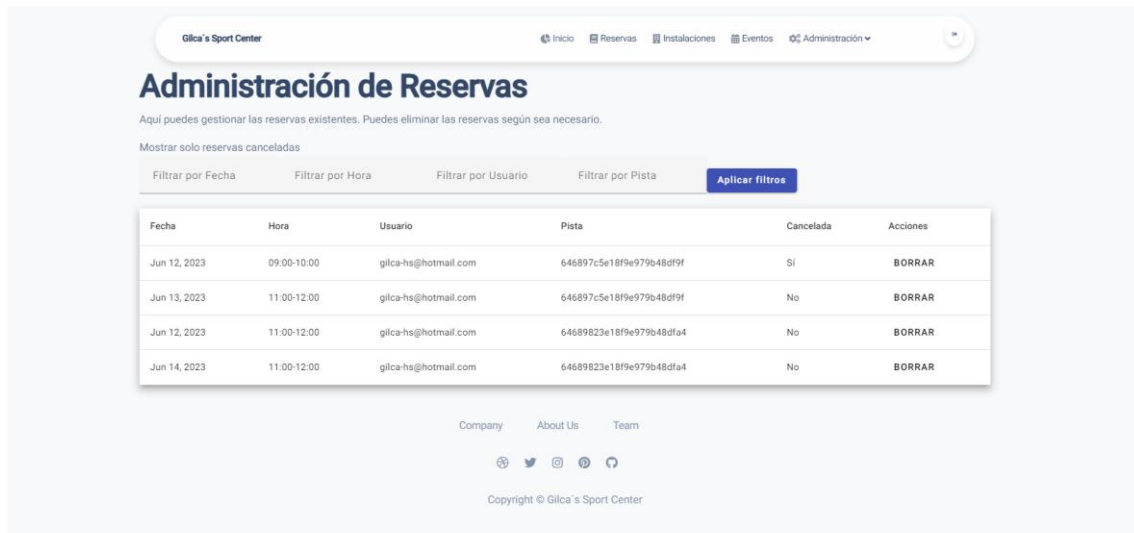
torneo de tenis

PISTA DE TENIS MUNICIPAL

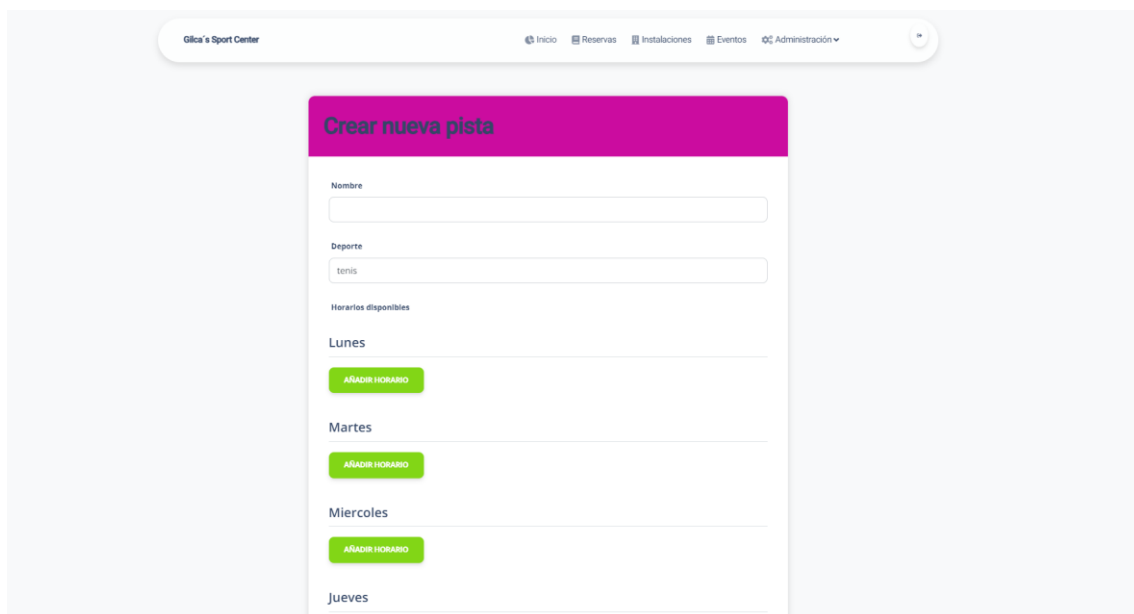


MUNICIPAL DE BASQUETBOL

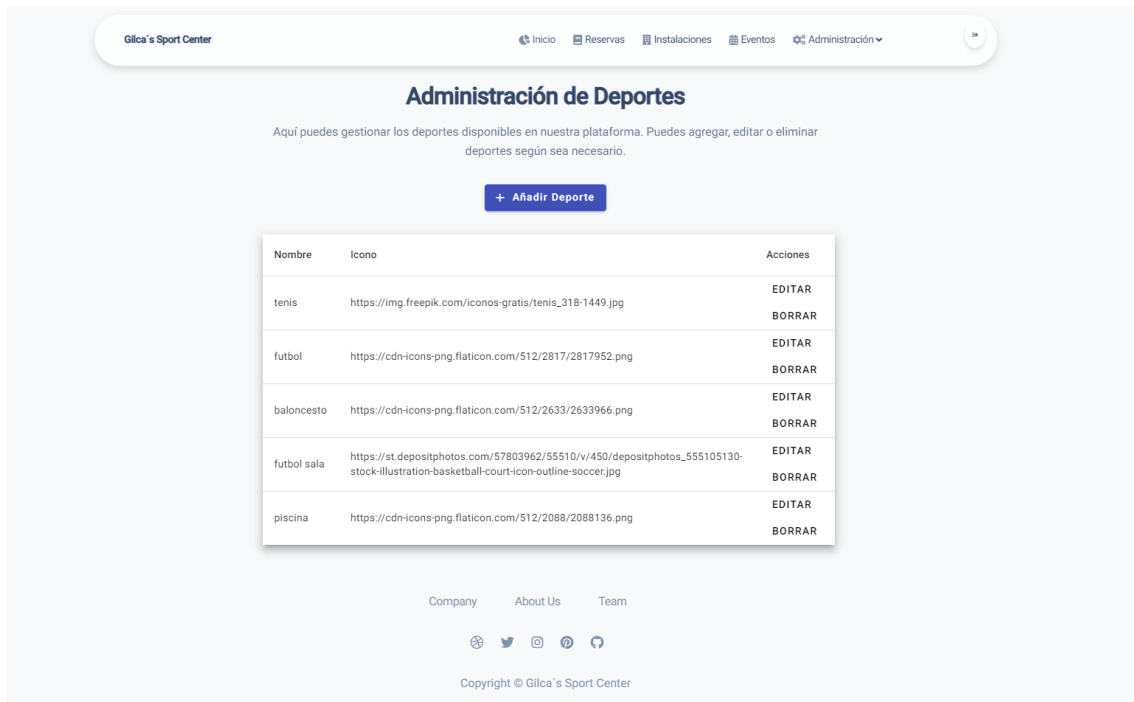
- **Gestión de reservas**
 - Información sobre las reservas y gestión de ellas



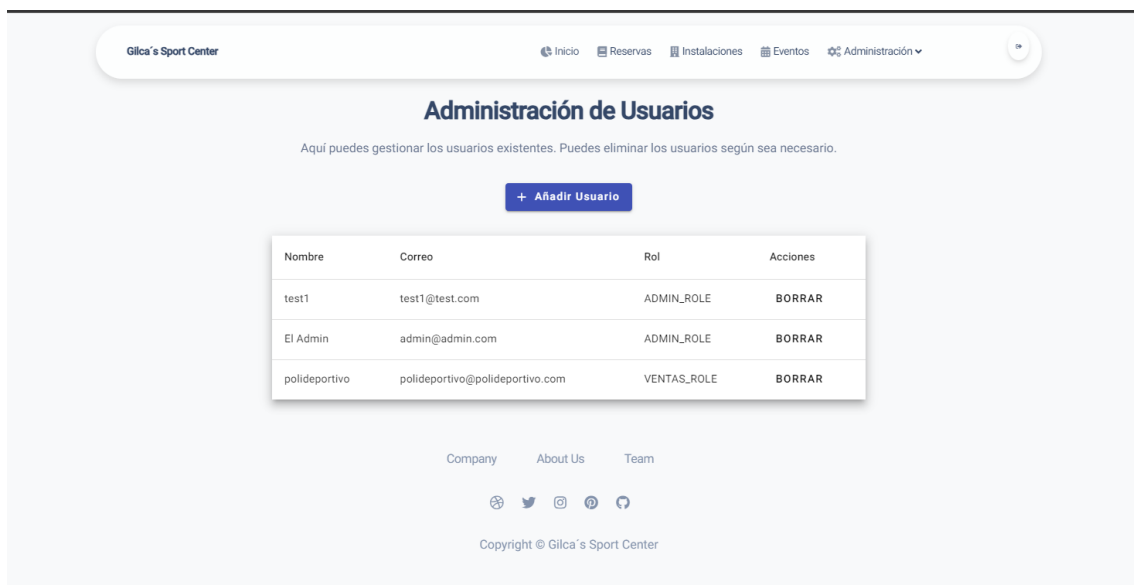
- **Creación de pistas**
 - Creación de nuevas pistas



- **Gestión de deportes**
 - gestión de los deportes. Borrar, editar y crear



- **Gestión de usuarios**
 - gestión de los usuarios. Borrar y crear



Cada uno de estos elementos representa una sección de la aplicación a la que los usuarios pueden navegar. Los usuarios pueden navegar a cualquier sección con un solo clic.

Propuestas de mejora

1. **Implementación de Pruebas Automatizadas:** Se podría considerar la implementación de pruebas unitarias y de integración para tu backend y frontend. Esto ayudará a asegurarte de que todas las partes de tu aplicación funcionen como se espera y te permitirá detectar y solucionar problemas más rápidamente.
2. **Implementación de gestión de eventos desde la web:** Se podría considerar la implementación de añadir una vista para gestionar le apartado de eventos de la web.
3. **Implementación de gestión de instalaciones desde la web:** Se podría considerar la implementación de añadir una vista para gestionar le apartado de eventos de la web.
4. **Implementación de sección y gestión de clases:** Se podría considerar la implementación de añadir una vista para gestionar clases impartidas en el centro y el usuario poder acceder a ellas desde la web.

Bibliografía

Para algunas parte del diseño del front me he fijado en esta plantilla:

[soft-ui-dashboard](#)

También en la organización de la pagina me he fijado en proyectos de reservas de polideportivos como puede ser el siguiente:

[polideportivo](#)

He utilizado componentes de:

<https://material.angular.io/>

<https://sweetalert2.github.io/>

Y a la hora de buscar duda he utilizado mucho:

[stackoverflow](#)

[chatgpt](#)

Enlaces a mis repositorios de github:

<https://github.com/gilcahs/GestionDeportivaBack.git>

<https://github.com/gilcahs/GestionDeportivaFront.git>

Mi pagina desplegada en netlify:

<https://startling-lebkuchen-663c2b.netlify.app/>