

**ÍNDICE**

[1. Introducción 3](#_Toc199735387)

[2 Contenido de la aplicación 3](#_Toc199735388)

[4. Desarrollo del Backend 3](#_Toc199735389)

[5. Desarrollo del Frontend 4](#_Toc199735390)

[5.1 Instalación del entorno y dependencias 4](#_Toc199735391)

[5.1.1 Angular 4](#_Toc199735392)

[5.1.2 Bootstrap 4](#_Toc199735393)

[5.1.3 Angular calendar 5](#_Toc199735394)

[5.1.4 Simulación de una API (Mock) 6](#_Toc199735395)

[5.2. Explicación de la estructura 7](#_Toc199735396)

[5.3 Diseño de la interfaz 8](#_Toc199735397)

[5.3.1 Pantallas 8](#_Toc199735398)

[5.3.2 Colores 10](#_Toc199735399)

[5.3.2 Iconografía 11](#_Toc199735400)

[6. Demostración de la aplicación 12](#_Toc199735401)

[7. Bibliografía y anexos 12](#_Toc199735402)

# Introducción

Esta aplicación surge a partir de la necesidad de la empresa por una plataforma para reservar y gestionar las vacaciones de los empleados de esta. Esta tarea se nos encomendó a **Miguel Hernández Monllor**, estudiante de prácticas de **DAW** del I.E.S. Paco Molla y a mí, **Roberto Rodes Luchoro**, estudiante de **DAM** del I.E.S Mare Nostrum por nuestro tutor y project manager **Jairo Moreno Villaroel** el cual nos ha ido guiando y dando pautas en el desarrollo de la aplicación. Cabe destacar que para el desarrollo frontend hemos contado de la ayuda de **Kevin Tipantiza**. Se nos encomendó primeramente desarrollar el diagrama de Entidad-Relación, diagrama de casos de uso y diagrama de clases

# Descripción del proyecto

Este proyecto se va a basar en una aplicación de **reserva** de **vacaciones** con **autenticación, token de inicio de sesión JWT** y **roles con permisos** en la que si el usuario va a poder tener dos tipos de roles: **administrador** y **usuario**.

El **rol** del **usuario** va a poder **registrar** sus **vacaciones**, **ver** las que tiene asignadas, **editar** las que tiene además de también poder **borrarlas**.

El **rol** de **administrador** por otra parte va a poder realizar todas estas acciones con sus vacaciones o las de otros usuarios **(tiene el control total)** además de poder ver las vacaciones de todo el mundo, y filtrar la vista de las vacaciones del usuario que elija mediante una lista por nombre y apellidos, y cuando quiera puede quitar los filtros y volver a verlos a todos.

También se ha **contemplado** que cada usuario al **registrarse** además de añadir su **nombre** y **apellidos** pueda elegir un **código** de **color** para que así se **pinten** de ese **color** cuando **aparezcan** en el **calendario**, dando un código de **color único** a cada **uno** de **ellos**.

# Análisis de requisitos

Los **requisitos funcionales** de software hacen referencia a aquellas **acciones** o **funciones** que lleva a cabo la **aplicación** para cumplir con un **objetivo**. A continuación, se definirán los requisitos funcionales de la aplicación.

## Requisitos funcionales

|  |  |
| --- | --- |
| Requisito funcional | Descripción |
| Iniciar sesión | La pantalla inicial de la aplicación contará con acceso autenticado por roles, diferenciando entre administradores y usuarios. |
| Gestión de vacaciones | Permite realizar un CRUD de la entidad vacación. |
| Cerrar sesión | El sistema cerrará la sesión del usuario activo. |

Tabla 1. Requisitos funcionales de la aplicación. Fuente: propia.

## Requisitos no funcionales

|  |  |
| --- | --- |
| Requisito funcional | Descripción |
| Seguridad | El sistema asignará funciones según el rol del usuario, restringiendo el acceso a las que no le correspondan, salvo en el caso del administrador, que tendrá control total. Además, los datos sensibles serán protegidos mediante cifrado. |
| Escalabilidad | La arquitectura deberá permitir incorporar nuevas funcionalidades y un aumento progresivo del número de usuarios sin pérdida de rendimiento. |
| Facilidad de mantenimiento | El código debe estar bien estructurado, modularizado y documentado para permitir una rápida localización de errores y una evolución ágil del sistema. |
| Accesibilidad | La interfaz deberá ser clara y operativa para usuarios con distintos niveles de experiencia digital, incluyendo diseño adaptado y legible en todos los dispositivos. |
| Fiabilidad | La aplicación debe operar sin interrupciones prolongadas, gestionando adecuadamente errores del sistema y asegurando la persistencia de los datos críticos. |

Tabla 2. Requisitos no funcionales. Fuente: propia.

## Metodología de desarrollo ágil: SCRUM

**SCRUM** es una metodología ágil que organiza el trabajo en ciclos cortos llamados **sprints**, fomentando la colaboración, la entrega continua y la adaptación al cambio.

Fue **elegida** porque **permite** responder **rápidamente** a **cambios**, **mejorar** el producto de **forma iterativa** y mantener una **buena comunicación** dentro del **equipo**.

# 4. Diseño

## Diseño de datos

### Diseño lógico-relacional

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

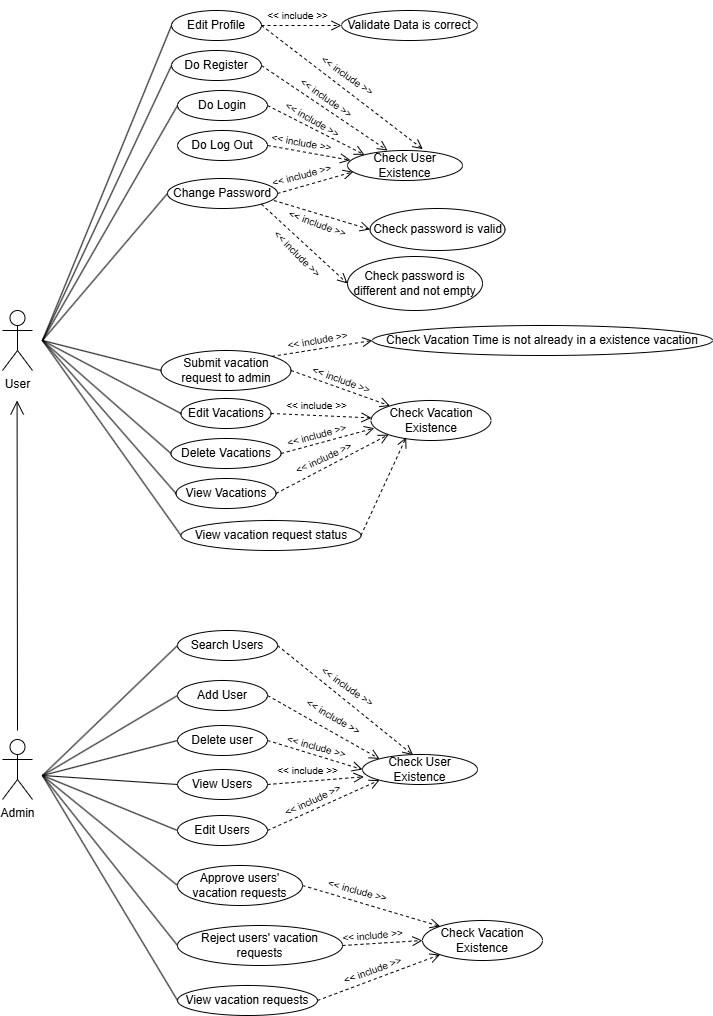
## Diseño funcional

### Diagrama de clases.

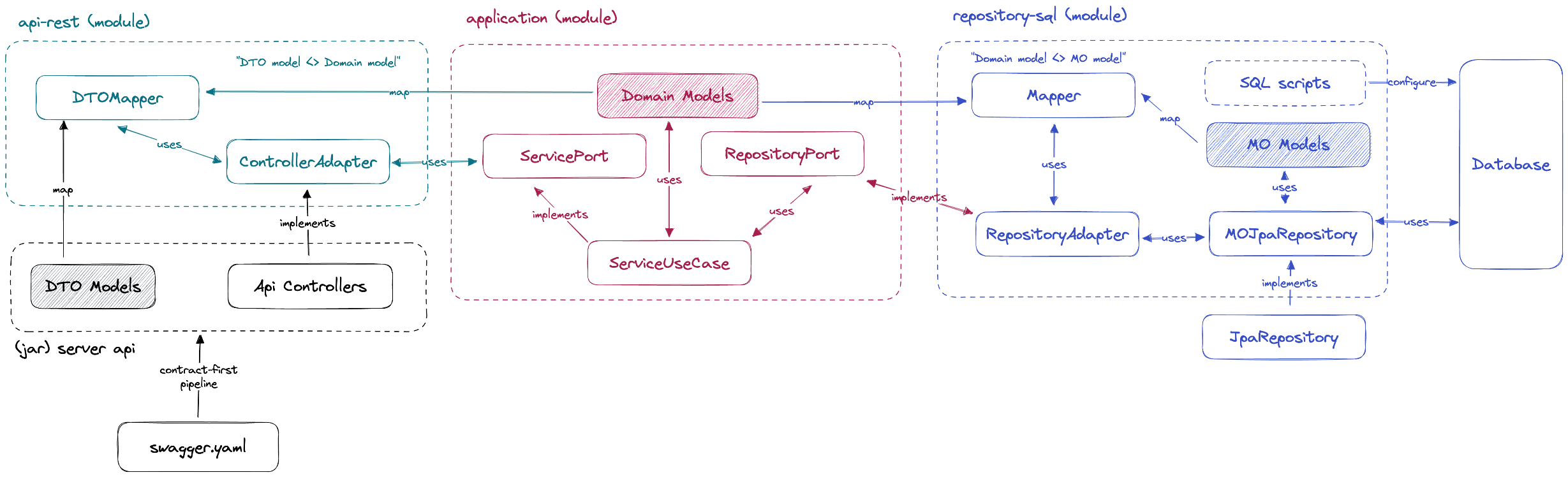
Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

### Diagrama de casos de uso



### Arquitectura de la aplicación: Hexagonal



# Desarrollo del Backend

El desarrollo del **Backend** se ha realizado todo con el **framework** de **Spring Boot** con el lenguaje de programación **JAVA**. Se ha realizado la documentación con la especificación de **openapi** en **swagger** y un plugin para generar los DTOs y las interfaces de los controladores de la aplicación además de para documentar la **API**.

La estructura de módulos y de carpetas que se ha seguido en el proyecto es esta:

Texto

El contenido generado por IA puede ser incorrecto.

Cabe destacar que el paquete de domino se ubica dentro del módulo de aplicación.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

El ejecutable de la aplicación se encuentra en el módulo boot, justo aquí:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

## Explicación de la estructura de módulos:

* **Módulo de aplicación:** Es aquel modulo el cual es independiente de todos y todos dependen de él. Contiene la lógica de negocio además de contener las clases de dominio, los servicios y los contratos que deberán de seguir los repositorios.
* **Módulo de dominio:** es el paquete que contiene las clases modelo que utiliza el módulo de la aplicación.
* **Modulo driven:** es el modulo que contiene todo aquello relacionado a la conexión a la base de datos y al repositorio.
* **Modulo driving:** es el módulo que expone los endpoints de la API-Rest mediante la comunicación HTTP.

## Base de datos: PostgreSQL

Es la base de datos que se ha elegido para este proyecto.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

## Spring JPA

Facilita el acceso a bases de datos mediante una abstracción sobre JPA e Hibernate usando repositorios.

Texto

El contenido generado por IA puede ser incorrecto.

## Mapstruct

Genera automáticamente código para mapear entre objetos (DTOs y entidades)

Texto

El contenido generado por IA puede ser incorrecto.

## Swagger Annotations

Para documentar y dar información excplicativa los endpoints REST:

Texto

El contenido generado por IA puede ser incorrecto.

## Spring Starter Validations

Proporciona validaciones automáticas en clases y entradas usando anotaciones como @NotBlank, @Email, etc.

Texto

El contenido generado por IA puede ser incorrecto.

## Spring Security

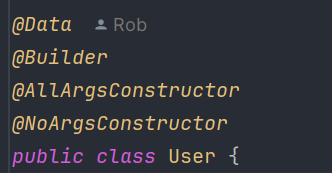
Añade seguridad a la aplicación, incluyendo autenticación, autorización basada en tokens y protección contra ataques comunes.

Texto

El contenido generado por IA puede ser incorrecto.

## Proyect Lombok

Reduce el código repetitivo en Java generando getters, setters, constructores, etc., mediante anotaciones.



## Slf4j

Es una interfaz de logging que permite usar diferentes implementaciones como Logback o Log4j

Texto

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

## JWT

Para generar el token de usuario al iniciar sesión en la aplicación y brindar mayor seguridad a la aplicación.

Texto

El contenido generado por IA puede ser incorrecto.

## Flyway

Permite versionar y automatizar los cambios en la estructura de la base de datos, haciendo tanto dmls como ddls.

Texto

El contenido generado por IA puede ser incorrecto.

# 6. Desarrollo del Frontend

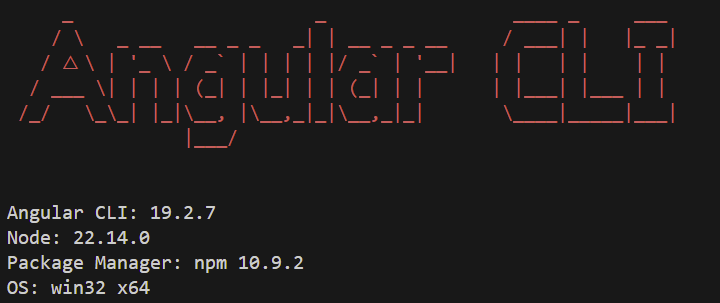
El desarrollo del Frontend lo hemos realizado todo con el framework de **Angular en su versión 19**, un framework moderno para crear aplicaciones web. Una de las principales ventajas de Angular es que permite desarrollar lo que se conoce como una *Single Page Application* (SPA), o "**aplicación de una sola página**". Esto significa que, **en lugar de recargar** completamente la página cada vez que el usuario navega por la aplicación, **se cargan solo las partes necesarias**.

El caso es que con él podemos crear una interfaz dinámica, mantenible y preparada para escalar en el futuro.

## 6.1 Instalación del entorno y dependencias

### 6.1.1 Angular

Para poder utilizar y disponer de la última versión de Angular (19 en este caso) globalmente tenemos que ejecutar en un terminal el comando **npm install -g @angular/cli** y una vez instalado se comprueba que lo tenemos instalado con **ng version**.



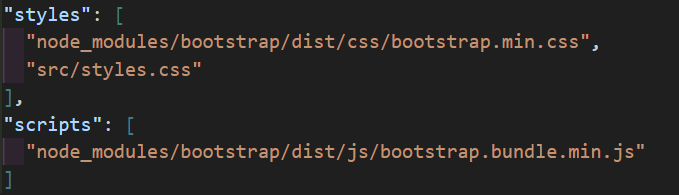
Después para utilizar el servidor de Angular se escribe el comando **ng serve –o** y por defecto se nos abrirá en la pestaña de nuestro navegador en el puerto 4200 la aplicación en modo desarrollo.

### 6.1.2 Bootstrap

En esta aplicación se ha decidido utilizar bootstrap para prescindir de todos los archivos css (menos el global para crear estilos y clases personalizadas).  
Para poder utilizarlo tenemos que añadir la dependencia requerida por Bootstrap con el comando **npm install bootstrap@latest --save.**

Una vez hecho esto tenemos que **configurarlo en Angular,** hay que agregar en el fichero **angular.json** el fichero CSS a nuestro proyecto.

En la sección “**styles**” del json hay que añadir esta linea y en la parte de “**scripts**” esta otra (dentro de build):

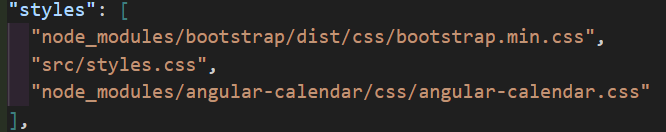


### 6.1.3 Angular calendar

Para integrar la librería de angular-calendar, seguí los pasos recomendados por su documentación oficial. Utilice el siguiente comando para instalar la librería y su dependencia necesaria para el manejo de fechas:

**npm install --save angular-calendar date-fns**

Después falta configurar los estilos globales CSS del calendario, lo podemos hacer en el archivo global css o en **angular.json** que como ya añadimos dentro la linea de bootstrap, la añadimos justo debajo:



Y por último importar el módulo de **CalendarModule** en la aplicación, **dentro de main.ts**, además de incluir el módulo **provideAnimations** para las animaciones del calendario, he incluido los **importProvidersFrom** para más tarde incluir un provider para hacer pruebas simulando una API.

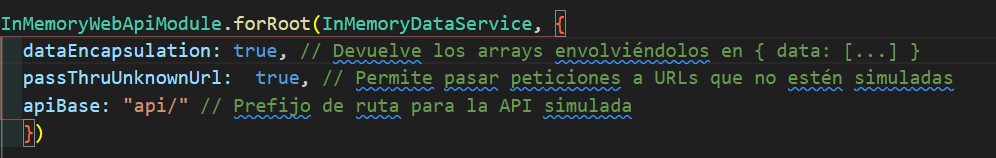


### 5.1.4 Simulación de una API (Mock)

También durante el desarrollo de la aplicación, se ha estado utilizando la dependencia de **InMemoryWebApiModule** del paquete angular-in-memory-web-api con el fin de simular una API RESTful y poder trabajar con datos sin necesidad de un backend real.

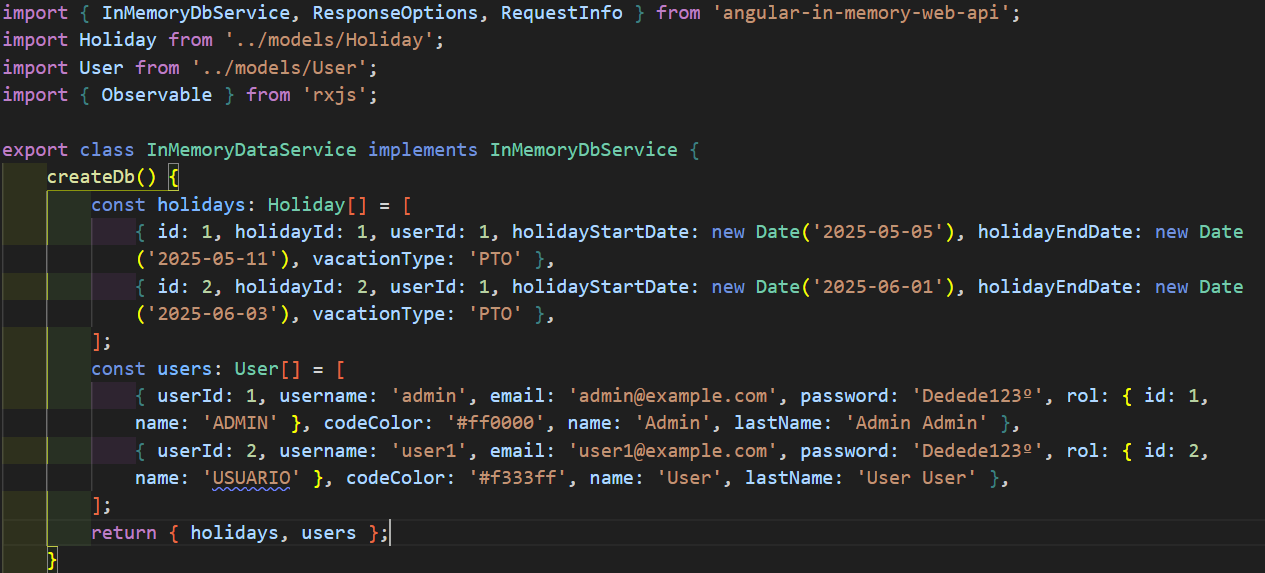
Esta técnica **permite interceptar las peticiones HTTP** realizadas por los servicios de Angular (HttpClient) **y devolver respuestas simuladas**, facilitando el desarrollo y pruebas del frontend.

Dentro de los importProviders:



Después de añadir todas las dependencias necesarias, **me he creado un servicio que simula una base de datos** en memoria utilizando InMemoryWebApiModule donde **se definen los datos de ejemplo (usuarios y vacaciones)** y se personalizan algunas **peticiones** como **GET** con filtros, **POST** para simular el login y la generación automática de IDs por ejemplo.

Algunos objetos en la BD simulada para poder utilizar (localhost:4200/api/users o /holidays)



## 6.2. Explicación de la estructura

Toda la aplicación se encuentra dentro del directorio src, y está organizada principalmente en dos partes:

La carpeta app, que contiene toda la lógica, vistas y estructura de la aplicación.

La carpeta environments, que gestiona las URLs de las APIs según el entorno.

**Desglose por carpetas (app)**

* **app.component:** Contiene el componente principal de la aplicación, el cual incluye el navbar y un <router-outlet> para renderizar las rutas de la aplicación.
* **components:** Aquí se agrupan todas las pantallas principales de la aplicación (login, registro, calendario, perfil, estructura, etc.), organizadas en subcarpetas según funcionalidad. También contiene el módulo vacation, dividido por operaciones como crear, editar o eliminar vacaciones.
* **errors:** Contiene componentes específicos para manejar errores comunes: 403 Forbidden y 404 Not Found, que se muestran cuando el usuario no tiene permisos o accede a una ruta inexistente.
* **guards:** Aquí se encuentran los guards utilizados para restringir el acceso a ciertas rutas, actualmente a usuarios autenticados, y en el futuro también a administradores.
* **models:** Define las interfaces y estructuras de datos utilizadas en la aplicación, lo cual permite evitar el uso de tipos any y facilita el tipado fuerte en TypeScript.
* **services:** En esta carpeta reside toda la lógica que se comunica con el backend (o con la API simulada). Los servicios se inyectan en los componentes y son responsables de manejar peticiones HTTP.
* **shared:** Contiene recursos reutilizables, como constantes globales y loaders para organizar mejor la carga de rutas en el enrutamiento, manteniendo el código más limpio y legible.
* **app.routes.ts:** Donde están todas las rutas de la aplicación con sus guards aplicados.

## 6.3 Diseño de la interfaz

Antes de desarrollar la interfaz, se utilizó **Lucidspark**, una herramienta similar a Figma pero con menos funcionalidades, para diseñar de forma visual cómo se vería la interfaz.

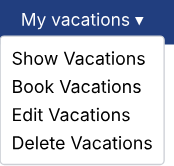
Empezamos a decidir como sería el navbar y las pantallas de login / registro asemejando a como están actualmente

### 6.3.1 Pantallas

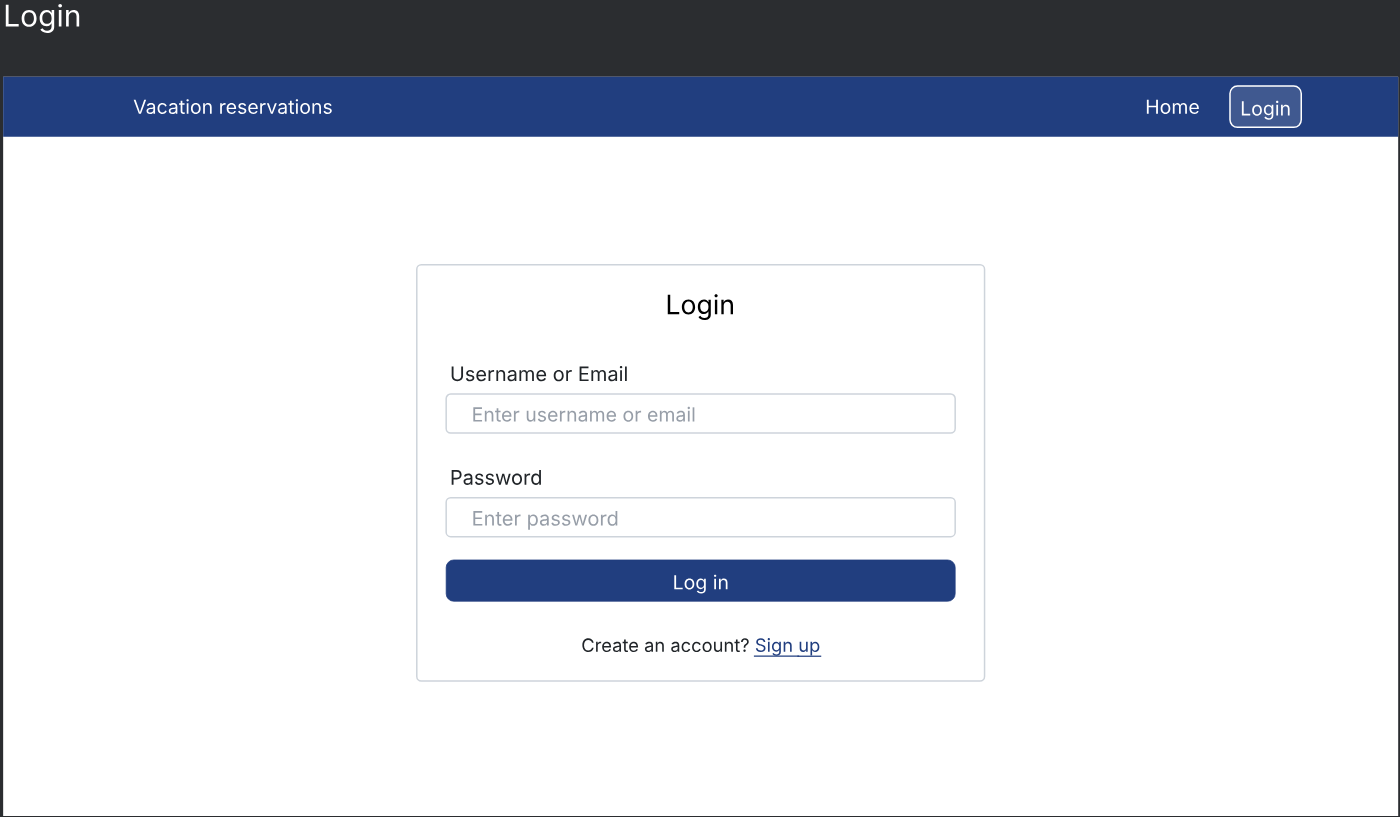
**Navbar**

Se pensó inicialmente realizar estos navbar para cuando se estuviese autenticado o no.

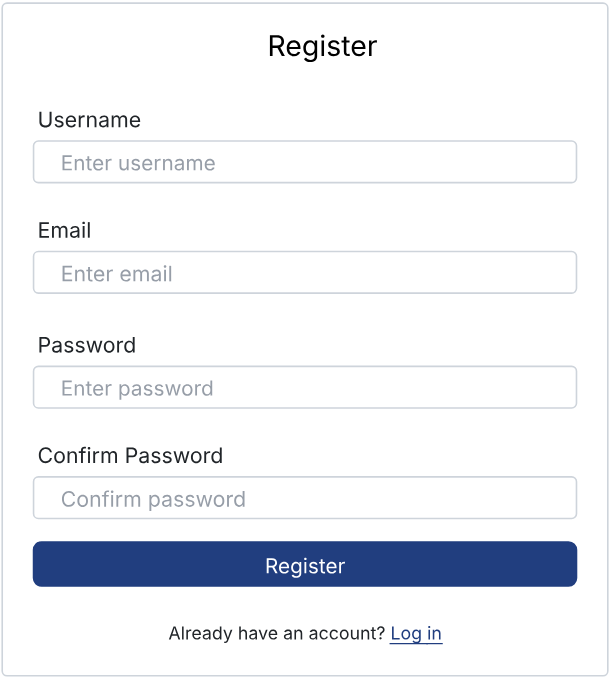




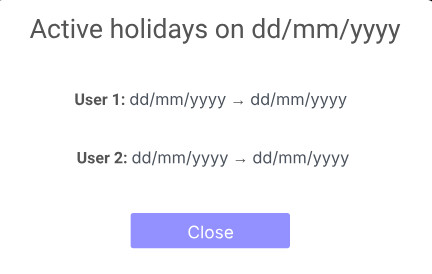
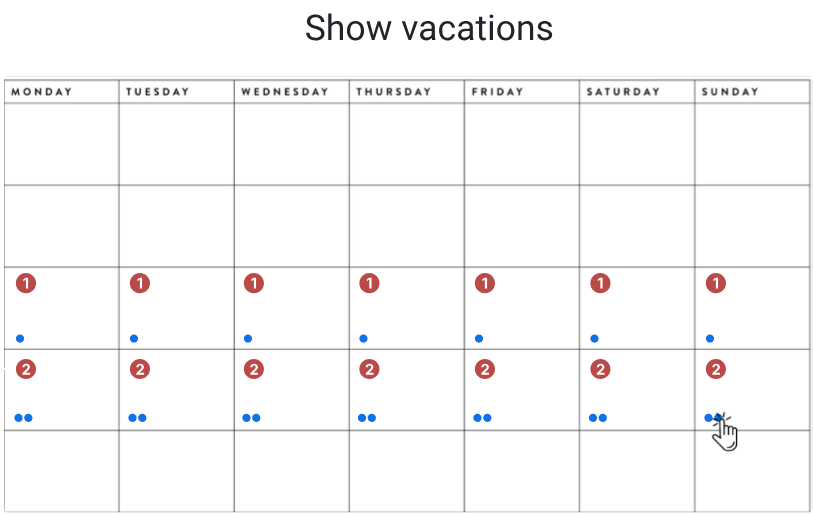
**Login:**



**Registro**



**Calendario (modo ver vacaciones)**



### 6.3.2 Colores

Hemos optado por estos 3 colores al final ya que representan los colores de la empresa de GFT.

Para backgrounds grandes como lo són el navbar y los botones hemos aplicado el color **#213e7f**

Para eventos como pasar el ratón por encima de algunos elementos se ha optado por utilizar el **#405990** que es como la versión con más opacidad del color anterior.

Y el otro color **#153A7B** se ha utilizado para marcar algunos bordes en partes como menus



### 6.3.2 Iconografía

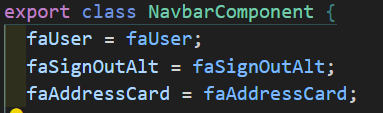
En esta aplicación se han utilizado iconos para **mejorar la usabilidad** **y** **la identificación visual** de algunas secciones, así se representa de manera más intuitiva. Los iconos han sido implementados utilizando **Font Awesome.**

Se ha instalado su dependencia con: **npm install** @fortawesome/angular-fontawesome @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons

Y ya luego se implementan en los componentes que nos hagan falta, importando su módulo.



Y para utilizarlos en la plantilla html los definimos en el TS.



**Iconos utilizados**



* Perfil → 
* Sign Out → 
* Usuario →
* Ojo passwd →