Iberdrola

Juan Cotolí San Félix
Héctor Blanco García
Javier Linares Ibáñez

Resumen

Castellano:

En el desarrollo de este proyecto hemos tenido varios objetivos con la finalidad de investigar, aprender y mejorar las distintas tecnologías presentes en el entorno del desarrollo web, además, de poder demostrar los conocimientos aprendidos durante el módulo de Desarrollo de Interfaces.
El primer objetivo fue el de poder utilizar un proyecto de otra asignatura. Este proyecto ya tenía el desarrollo de la base de datos al completo y era una muy buena base para este proyecto.
El segundo objetivo era poder enlazar la base de datos con el proyecto, para ello tuvimos que usar el framework Spring Boot para crear la API que nos permita enlazar ambos proyectos.
El tercer objetivo era desarrollar una web con una experiencia muy cercana a la que tendrías con una aplicación de escritorio, con una respuesta hacía el usuario dinámica.
La principal ventaja de este tipo de desarrollo es la flexibilidad que te da al poder ser utilizado en cualquier dispositivo, incluyendo dispositivos móviles.
Para ello hemos usado Angular como plataforma para el desarrollo del proyecto, ya que nos permitía implementar todo lo que necesitábamos.
Hemos usado HTML y Tailwind CSS para el desarrollo de la interfaz.
Durante todo el proyecto se ha usado Git Hub para el guardado del proyecto, gestión de versiones y aportaciones en grupo.

Valenciano:

En el desenvolupament d'este projecte hem tingut diversos objectius amb la finalitat d'investigar, aprendre i millorar les diferents tecnologies presents a l'entorn del desenvolupament web, a més, de poder demostrar els coneixements apresos durant el mòdul de Desarrollo de Interfaces.
El primer objectiu va ser el de poder utilitzar un projecte d'una altra assignatura. Este projecte ja tenia el desenvolupament de la base de dades al complet i era una molt bona base per a este projecte.
El segon objectiu era poder enllaçar la base de dades amb el projecte, per a això vam haver d'usar el framework Spring Boot per a crear la API que ens permeta enllaçar tots dos projectes.
El tercer objectiu era desenvolupar una web amb una experiència molt pròxima a la qual tindries amb una aplicació d'escriptori, amb una resposta feia l'usuari dinàmica.
El principal avantatge d'esta mena de desenvolupament és la flexibilitat que et dona en poder ser utilitzat en qualsevol dispositiu, incloent-hi dispositius mòbils.
Per a això hem usat Angular com a plataforma per al desenvolupament del projecte, ja que ens permetia implementar tot el que necessitàvem.
Hem usat HTML i Taildwind CSS per al desenvolupament de la interfície.

Durant tot el projecte s'ha usat Git Hub per al guardat del projecte, gestió de versions i aportacions en grup.

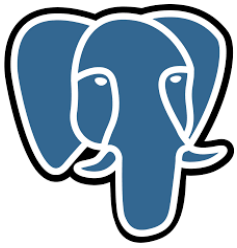
Inglés:

In the development of this project, we have had several objectives with the aim of researching, learning and improving the different technologies present in the web development environment, as well as being able to demonstrate the knowledge learned during the Interface Development module.
The first objective was to be able to use a project from another subject. This project already had the development of the complete database and was a very good basis for this project.
The second objective was to be able to link the database with the project, for this we had to use the Spring Boot framework to create the API that allows us to link both projects.
The third objective was to develop a website with an experience very close to the one you would have with a desktop application, with a dynamic response to the user.
The main advantage of this type of development is the flexibility it gives you as it can be used on any device, including mobile devices.
For this we have used Angular as the platform for the development of the project, as it allowed us to implement everything we needed.
We used HTML and Tailwind CSS for the development of the interface.
Throughout the project, Git Hub has been used for project saving, version management and group contributions.

Tabla de contenido

Herramientas

La herramienta usada para la creación y programación de la base de datos ha sido PostgreSQL. Consideramos que es una herramienta muy potente y además de código abierto, siendo idónea para este proyecto.



Para la conexión de la base de datos con la aplicación, hemos usado Spring Boot. Esta herramienta también es de código abierto.

El uso de Visual Studio Code como herramienta de editor de código ha sido indispensable debido a sus extensiones, estás te permiten poder trabajar mucho más cómodo y de manera más eficiente.

Por último, el uso de Git Hub era preciso para poder trabajar en grupo, además, te permite tener el proyecto de manera redundante y gestionar las distintas versiones.

Módulo de autentificación

Se ha desarrollado un módulo de autentificación para los usuarios, los cuáles deben de introducir obligatoriamente un correo electrónico y una contraseña.
A continuación, se va a explicar los distintos apartados de este módulo:

HTML:

Este es un fragmento de código HTML que representa un formulario de inicio de sesión (login). Aquí hay una explicación detallada de cada parte:

1. Formulario de Inicio de Sesión:
    a. Se utiliza la etiqueta `<form>` para crear un formulario.
    b. Se establece una clase CSS `my-10 space-y-6` en el formulario para aplicar estilos de espaciado vertical.
    c. Se vincula el formulario a un controlador de formulario Angular mediante `[formGroup]="form"`.
    d. Se establece la acción que se ejecutará cuando se envíe el formulario mediante `(ngSubmit)="onSubmit()"`.
2. Encabezado del Formulario:
    a. Se utiliza un elemento `<div>` con la clase CSS `text-center` para centrar el contenido del encabezado.
    b. Se incluye un título "Bienvenido" y se utiliza una clase `text-primary-500` para aplicar estilos específicos al texto.
    c. Se proporciona una descripción adicional debajo del título.
3. Acceso con Google:
    a. Se utiliza un botón que incluye un logotipo de Google y texto "Accede con Google".
    b. Se utiliza la directiva `routerLink="/dashboard"` para dirigir al usuario a la página de inicio del dashboard cuando se hace clic en este botón.
4. Separador "or":
    a. Se utiliza un `<div>` para crear un separador visual entre las opciones de inicio de sesión. Contiene un texto "or".
5. Campos de Usuario y Contraseña:
    a. Se utilizan dos elementos `<input>` para los campos de usuario y contraseña.
    b. Cada campo tiene un `<label>` asociado para describir su propósito.
    c. Se aplican clases y lógica Angular para manejar la validación de los campos.
6. Botón de Recordar Usuario:
    a. Se utiliza un `<input>` de tipo `checkbox` junto con un `<label>` para permitir al usuario seleccionar si desea que su información de inicio de sesión se recuerde.
7. Botón de Acceder:
    a. Se utiliza un botón para enviar el formulario.
    b. Cuando se hace clic en este botón, se ejecutará la función `onSubmit()`.

c. Se aplican clases para estilizar el botón y proporcionar retroalimentación visual cuando el usuario interactúa con él.

Component TS:

Este es un archivo TypeScript que define un componente llamado `SignInComponent`. Aquí está una explicación de cada parte:

1. Imports:
    a. Se importan varios módulos y componentes de Angular necesarios para el funcionamiento del componente. Estos incluyen cosas como el formulario reactivo (`ReactiveFormsModule`), la creación de formularios (`FormBuilder`), el enrutamiento (`Router`), y algunos otros módulos y directivas de Angular.
2. Component Decorator:
    a. El decorador `@Component` se utiliza para configurar el componente. Aquí se especifica el selector del componente, la plantilla HTML asociada y los estilos CSS, junto con algunas opciones adicionales.
    b. La opción `standalone: true` indica que este componente es independiente y no depende de otros componentes específicos para funcionar.
3. Clase del Componente:
    a. La clase `SignInComponent` implementa la interfaz `OnInit`, lo que significa que debe tener un método `ngOnInit()` que se ejecutará cuando se inicie el componente.
    b. Se definen varias propiedades, incluyendo `form`, que representa el formulario del componente, `submitted` para rastrear si el formulario ha sido enviado, y `passwordTextType` para alternar la visibilidad de la contraseña.
4. Constructor
    a. El constructor del componente recibe instancias de `FormBuilder` y `Router` como parámetros, que se utilizan para construir el formulario y para la navegación respectivamente.
5. ngOnInit():
    a. En el método `ngOnInit()`, se inicializa el formulario utilizando `FormBuilder`, especificando los campos `email` y `password`, junto con las validaciones requeridas.
6. get f():
    a. Este es un getter que devuelve los controles del formulario para facilitar el acceso a ellos en la plantilla HTML.
7. togglePasswordTextType():
    a. Este método se utiliza para cambiar el tipo de entrada de la contraseña entre texto visible y asteriscos (para ocultar la contraseña).
8. onSubmit():
    a. Este método se llama cuando se envía el formulario.
    b. Marca el formulario como enviado.
    c. Extrae los valores del formulario para el email y la contraseña.

d. Si el formulario es inválido, se detiene aquí.
e. Si el formulario es válido, navega al inicio (`'/'`) utilizando el servicio de enrutamiento.

Component Spec TS:

Este es un archivo de prueba unitaria (unit test) escrito en Angular utilizando el marco de pruebas `Jasmine`. Aquí está una explicación de cada parte:

1. Imports:
    a. Se importan `ComponentFixture` y `TestBed` desde `@angular/core/testing`. Estos son necesarios para configurar el entorno de prueba y crear componentes para probar.
    b. Se importa el componente `SignInComponent` que se está probando.
2. Describe:
    a. `describe` es una función de Jasmine que define un conjunto de pruebas para un conjunto particular de funcionalidades. En este caso, se describe el componente `SignInComponent` y sus pruebas asociadas.
3. Variables de Componente y Fixture:
    a. Se declaran dos variables: `component`, que representa una instancia del componente `SignInComponent`, y `fixture`, que representa el contenedor que envuelve el componente para realizar pruebas.
4. beforeEach(async):
    a. Este bloque se ejecuta antes de cada prueba, de forma asíncrona.
    b. Utiliza `TestBed.configureTestingModule()` para configurar un módulo de pruebas de Angular.
    c. En este caso, solo se importa el `SignInComponent` en lugar de un módulo completo, ya que este componente no depende de otros componentes o servicios.
    d. `compileComponents()` compila todos los componentes declarados en `TestBed.configureTestingModule()` para su uso en las pruebas.
5. beforeEach():
    a. Este bloque se ejecuta antes de cada prueba.
    b. Utiliza `TestBed.createComponent()` para crear una instancia del componente `SignInComponent`.
    c. Asigna la instancia del componente a la variable `component` y obtiene una referencia al contenedor de pruebas en `fixture`.
6. it('should create'):
    a. Esta es una prueba unitaria individual.
    b. Utiliza la función `it()` de Jasmine para definir una sola prueba.
    c. La cadena `'"should create"'` es una descripción legible de lo que se está probando.
    d. Dentro de la función de prueba, se utiliza `expect()` para definir las expectativas.

e. En este caso, se espera que el componente `SignInComponent` se haya creado exitosamente.
f. `toBeTruthy()` es un matcher de Jasmine que verifica si el valor es verdadero (no nulo ni indefinido). En otras palabras, verifica si `component` no es nulo, lo que significa que se ha creado con éxito.

Routing Module TS:

Este es un módulo de enrutamiento (`AuthRoutingModule`) en Angular que define las rutas para la autenticación en una aplicación. Aquí está una explicación de cada parte:

1. Imports:
    a. Se importan los módulos `NgModule`, `RouterModule` y `Routes` desde `@angular/core` y `@angular/router` respectivamente.
2. Definición de las Rutas:
    a. Se define un array `routes` que contiene las rutas para la autenticación.
    b. Cada ruta tiene un objeto con al menos dos propiedades: `path` y `component`.
    c. `path` especifica la parte de la URL después del dominio que activará esta ruta.
    d. `component` especifica el componente que se renderizará cuando la ruta coincida.
    e. Además de `path` y `component`, se pueden agregar otras propiedades como `data` para pasar datos estáticos a la ruta, como se hace en la ruta `'sign-in'` para proporcionar un `returnUrl`.
3. Lazy Loading:
    a. Este módulo de enrutamiento utiliza el enrutamiento perezoso (lazy loading), lo que significa que los componentes se cargarán bajo demanda.
    b. El componente `AuthComponent` es el componente principal que se carga inicialmente. Luego, sus hijos se cargarán dinámicamente según las rutas coincidan.
4. Redireccionamiento:
    a. Si la ruta no coincide con ninguna de las rutas definidas, se redireccionará a la ruta `'sign-in'`.
5. NgModule Decorator:
    a. Se utiliza el decorador `@NgModule` para configurar el módulo `AuthRoutingModule`.
    b. En la sección `imports`, se importa `RouterModule.forChild(routes)`, lo que configura las rutas definidas en `routes`.
    c. En la sección `exports`, se exporta `RouterModule`, lo que permite que este módulo de enrutamiento sea utilizado por otros módulos de Angular que lo importen.

Módulo de configuración

Se ha desarrollado un módulo de configuración para los usuarios, este permite la manipulación del perfil del usuario.

A continuación, se va a explicar los distintos apartados de este módulo:

HTML de profile:

Este fragmento de código HTML representa un formulario para personalizar el perfil de un usuario. Aquí está una explicación detallada de cada parte:

1. Div con Clase "pt-6":
    a. Este div sirve como contenedor principal del formulario de personalización del perfil.
2. Encabezado:
    a. Dentro del contenedor principal, hay un encabezado con la clase "px-4 sm:px-0", lo que indica el espaciado interno en diferentes tamaños de pantalla.
    b. Contiene un título "Personalizar perfil" con estilo de texto.
3. Información del Perfil:
    a. Justo debajo del encabezado, hay un título secundario "INFORMACIÓN DEL PERFIL".
    b. Este título está en un tamaño de texto más pequeño y tiene un estilo de color gris.
4. Sección de Nombre de Perfil:
    a. Esta sección incluye un campo de entrada de texto para que el usuario ingrese su nombre de perfil.
    b. Se proporciona una descripción debajo del campo de entrada para guiar al usuario sobre lo que debe ingresar.
    c. El campo de entrada tiene un id "first_name" y está marcado como requerido con el atributo "required".
5. Sección de Imagen de Perfil:
    a. Esta sección incluye una imagen de perfil existente junto con un mensaje sobre los formatos de imagen permitidos.
    b. Se presenta una miniatura de la imagen actual del perfil.
    c. Hay un área de carga de archivos donde los usuarios pueden cargar una nueva imagen de perfil.
    d. Esta área de carga incluye un ícono SVG, un mensaje de instrucciones y un campo de entrada de tipo "file" oculto para seleccionar archivos.

Component TS de profile:

Este fragmento de código TypeScript define un componente Angular llamado `ProfileComponent`.

1. Imports:
    a. Se importa el decorador `Component` desde el módulo `@angular/core`, que es necesario para definir un componente en Angular.
2. Component Decorator:
    a. El decorador `@Component` se utiliza para configurar el componente. Aquí se especifica el selector del componente (`selector: 'config-profile'`), que se utilizará para incluir este componente en otras partes de la aplicación.
    b. Se especifica la ruta de la plantilla HTML asociada (`templateUrl: './profile.component.html'`), lo que significa que la plantilla se encuentra en un archivo separado llamado `profile.component.html`.
    c. La opción `standalone: true` indica que este componente es independiente y no depende de otros componentes específicos para funcionar.
3. Clase del Componente:
    a. Se define una clase llamada `ProfileComponent`.
    b. Esta clase está vacía en este fragmento de código, lo que significa que no tiene propiedades ni métodos definidos. Es posible que se agreguen más propiedades y métodos más adelante, dependiendo de las necesidades del componente.

HTML de user:

Este código HTML representa una sección de ajustes de cuenta en una página web. Aquí está una explicación detallada:

1. Encabezados y Secciones:
    a. Se utilizan encabezados (`<h3>` y `<h6>`) para organizar y etiquetar diferentes secciones de ajustes de cuenta.
    b. Las clases CSS como `pt-6`, `mt-10`, `-mb-5`, etc., se utilizan para ajustar el espaciado y la posición de los elementos.
2. Ajustes de Cuenta:
    a. Se muestra información sobre la dirección de correo electrónico y la contraseña del usuario.
    b. Cada sección tiene un botón "Cambiar" que permite al usuario modificar esa información.
3. Preferencias de Contenido e Interfaz:
    a. Se muestran opciones para cambiar preferencias relacionadas con el contenido y la interfaz de usuario, como el idioma del contenido y el tema del sistema.
    b. Cada opción tiene un botón "Cambiar" que, cuando se hace clic, probablemente abrirá un modal o una nueva página para realizar el cambio.

4. Eliminar Cuenta:
    a. Se proporciona una sección para eliminar la cuenta.
    b. Se muestra un botón "ELIMINAR CUENTA" que, al hacer clic, probablemente solicitará confirmación antes de eliminar permanentemente la cuenta.
5. Clases CSS Personalizadas:
    a. Se utilizan varias clases CSS personalizadas para aplicar estilos específicos a los elementos, como colores de texto, fondos y bordes.
6. Directiva Angular `routerLink`:
    a. Se utiliza la directiva `routerLink` en el botón "ELIMINAR CUENTA" para enrutar al usuario a la página de autenticación cuando se hace clic en el botón. Esto está destinado a evitar que el usuario elimine la cuenta accidentalmente, mostrando una confirmación antes de realizar la acción.

Component TS de user:

Este es un componente Angular llamado `UserComponent` que parece estar relacionado con la configuración del usuario. Aquí está la explicación de cada parte:

1. Imports:
    a. `Component`: Este import es necesario para definir el componente.
    b. `OnInit`: Esta interfaz es implementada por la clase del componente, lo que significa que debe tener un método `ngOnInit()` que se ejecutará cuando se inicie el componente.
    c. `ThemeService`: Este import probablemente es un servicio que maneja el tema de la aplicación, posiblemente alternando entre temas claro y oscuro.
2. Component Decorator:
    a. Se utiliza el decorador `@Component` para configurar el componente.
    b. Se especifica el selector del componente como `'config-user'`.
    c. Se proporciona la ubicación de la plantilla HTML asociada al componente.
    d. Se establece la opción `standalone: true`, lo que sugiere que este componente no depende de otros componentes específicos para su funcionamiento.
3. Clase del Componente:
    a. La clase `UserComponent` implementa la interfaz `OnInit`.
    b. Contiene un constructor que inyecta el servicio `ThemeService` como una dependencia. La palabra clave `public` antes del nombre de la variable `themeService` en el constructor indica que el servicio estará disponible para su uso en la plantilla HTML.
    c. Tiene un método `ngOnInit()` vacío, lo que significa que no realiza ninguna acción específica cuando se inicializa el componente.
4. toggleTheme():
    a. Este método se utiliza para cambiar el tema de la aplicación.
    b. Cuando se llama, alterna entre el tema oscuro y claro según el estado actual del servicio `ThemeService`.

c. Esto se logra cambiando la propiedad `theme` del servicio a `'dark'` si el tema actual no es oscuro, y a `'light'` si es oscuro. Esto probablemente activará algún tipo de cambio en la apariencia de la aplicación basado en el tema seleccionado.

Routing Module TS:

Este es un archivo de enrutamiento de Angular (`ConfigRoutingModule`) que define las rutas para el módulo `ConfigModule`. Aquí hay una explicación de cada parte:

1. Imports:
    a. Se importan los módulos necesarios de Angular, `NgModule` y `RouterModule`, junto con la interfaz `Routes` que define la estructura de las rutas.
2. Declaración de rutas:
    a. Se define un arreglo de objetos llamado `routes`, que contiene las rutas para este módulo.
    b. Cada ruta es un objeto que especifica la URL del recurso (`path`) y el componente que se debe renderizar cuando esa URL coincida.
    c. En este caso, solo hay una ruta definida con la URL raíz (`''`) que se asigna al componente `ConfigComponent`.
    d. También se define un niño de ruta vacío que redirige al componente `ConfigComponent` cuando la URL coincide exactamente con la raíz.
3. NgModule Decorator:
    a. Se utiliza el decorador `@NgModule` para configurar el módulo `ConfigRoutingModule`.
    b. En el bloque de configuración del decorador, se especifica la importación de `RouterModule` utilizando el método `forChild()`, pasando las rutas definidas anteriormente.
    c. Se exporta `RouterModule`, lo que permite que las directivas de enrutamiento y las funcionalidades estén disponibles para otros módulos que importen este módulo de enrutamiento.

Módulo de HomePage

HTML de home:

Este es un fragmento de código HTML que probablemente forma parte de una página web o una aplicación web. Aquí está la explicación de cada parte:

1. Contenedor Principal (`<div class="w-full max-h p-4 bg-white dark:bg-night-900">`):
    a. Es un contenedor principal que ocupa todo el ancho disponible (`w-full`) y tiene un relleno interior de 4 unidades (`p-4`).
    b. Tiene un fondo blanco (`bg-white`) en el modo claro y un fondo oscuro (`dark:bg-night-900`) en el modo oscuro.
2. Contenedor con Efecto de Fondo:
    a. Este contenedor se utiliza para crear un efecto visual de fondo.
    b. Tiene una posición relativa (`relative`) y se aísla (`isolate`) para evitar que los estilos hereden desde fuera.
    c. Tiene un relleno en el eje X de 6 unidades (`px-6`) y en el eje Y de 14 unidades (`pt-14`).
    d. En dispositivos grandes (`lg:`), el relleno horizontal se incrementa a 8 unidades (`lg:px-8`).
3. Div con Efecto de Fondo Gradiente:
    a. Este div se coloca absolutamente (`absolute`) en el contenedor principal y se desplaza hacia arriba (`-top-40`) para crear el efecto de fondo.
    b. Se aplica un desenfoque (`blur-3xl`) al contenido del div para crear un efecto visual difuminado.
    c. El div tiene un gradiente de color que va desde un tono rosa (`#ff80b5`) hasta un tono morado (`#9089fc`).
    d. Se define una forma poligonal (`clip-path`) para ajustar el gradiente al diseño deseado.
4. Contenido Principal (`<div class="mx-auto max-w-2xl py-32 sm:py-48 lg:py-56">`):
    a. Este div contiene el contenido principal de la sección.
    b. Está centrado horizontalmente (`mx-auto`) y tiene un ancho máximo de 2xl (extragrande) para adaptarse a diferentes tamaños de pantalla.
    c. Tiene un relleno en el eje Y de 32 unidades (`py-32`), que se incrementa en dispositivos medianos (`sm:`), grandes (`lg:`).
5. Texto y Enlaces:
    a. Hay un enlace con texto "Informate sobre nuestra compañia." que lleva a un recurso desconocido (`#`).
    b. Debajo, hay un título principal (`<h1>`) con un tamaño de fuente grande y un peso de fuente negrita.
    c. Se proporciona un texto de subtítulo (`<p>`) debajo del título principal.
    d. Se muestran dos enlaces de navegación (`<a>`) que probablemente llevan a diferentes partes de la aplicación, como la página de precios y la página de inicio de sesión.
    e. Los enlaces tienen un estilo de botón con un fondo de color índigo y texto blanco, y aplican sombras y efectos de enfoque visuales al ser seleccionados o enfocados.

Component TS de home:

Este es un componente Angular llamado `HomeComponent`, que aparentemente controla la página de inicio (`home`). Aquí está la explicación de cada parte del código:

1. Imports:
    a. `Component` y `OnInit` son decoradores e interfaces de Angular, respectivamente, que se utilizan para definir y manejar componentes.
    b. `ThemeService` es un servicio proporcionado por la aplicación para manejar temas y probablemente se utiliza para cambiar entre un tema oscuro y claro.
    c. `RouterLink` es una directiva que se utiliza para navegar a una ruta en la aplicación cuando se hace clic en un elemento.
2. Component Decorator**:
    a. `@Component` es un decorador que se utiliza para configurar el componente. Aquí se especifica el selector del componente (`'home-home'`), que se utiliza para referirse al componente en la plantilla HTML, y la plantilla HTML asociada (`'./home.component.html'`). También hay una opción adicional `standalone: true`, que indica que este componente es independiente y no depende de otros componentes específicos para funcionar.
3. Clase del Componente:
    a. `HomeComponent` es la clase del componente.
    b. Se inyecta el servicio `ThemeService` en el constructor del componente, lo que permite al componente acceder a las funcionalidades proporcionadas por este servicio.
4. ngOnInit():
    a. `ngOnInit()` es un método del ciclo de vida del componente que se ejecuta cuando el componente se inicia.
    b. En este caso, el método está vacío, lo que significa que no hay acciones específicas que se realicen cuando el componente se inicializa.
5. toggleTheme():
    a. Este es un método definido en el componente.
    b. Cuando se llama, cambia el tema entre oscuro y claro. Lo hace a través del servicio `ThemeService`, modificando la propiedad `theme` del servicio en función de si `isDark` es verdadero o falso. Si el tema es oscuro, se cambia a claro, y viceversa.

Routing Module TS:

Este fragmento de código corresponde a un módulo de enrutamiento (`HomepageRoutingModule`) en una aplicación Angular. Aquí hay una explicación de cada parte:

1. Imports:
    a. `NgModule`: Se importa desde `@angular/core` para definir un módulo de Angular.
    b. `RouterModule` y `Routes`: Se importan desde `@angular/router` para configurar las rutas de la aplicación.
    c. Se importan los componentes `HomepageComponent`, `PricingComponent`, y `HomeComponent`, que se utilizarán como vistas para las diferentes rutas.
2. Definición de Rutas:
    a. Se define una constante `routes` que es un array de objetos de tipo `Routes`.
    b. Cada objeto representa una ruta y tiene dos propiedades principales: `path` y `component`.
    c. El primer objeto especifica que cuando la URL contenga '/prices', se cargará el componente `PricingComponent`.
    d. El segundo objeto especifica que cuando la URL sea la raíz del sitio ('/'), se cargará el componente `HomeComponent`.
3. NgModule Decorator:
    a. El decorador `@NgModule` se utiliza para configurar el módulo.
    b. En la sección `imports`, se importa el módulo de enrutamiento `RouterModule` y se le pasan las rutas definidas anteriormente mediante `forChild(routes)`.
    c. En la sección `exports`, se exporta `RouterModule`, lo que permite que el módulo de enrutamiento sea utilizado por otros módulos de la aplicación.
4. Exportación del Módulo:
    a. El módulo `HomepageRoutingModule` se exporta como parte de la estructura de la aplicación, lo que permite que otros módulos lo importen y utilicen sus configuraciones de enrutamiento.

Módulo de Layout

HTML del bottom-navbar:

Este es un fragmento de código HTML que representa una barra de navegación fija en la parte inferior de la página. Aquí está una explicación de cada parte:

1. Sección de Navegación:
    a. Se utiliza la etiqueta `<section>` para crear una sección de contenido.
    b. Se le aplican varias clases CSS para definir su posición (`fixed bottom-0`), su apariencia (`bg-white/50 shadow backdrop-blur-sm dark:bg-night-600/50`), y su tamaño (`block w-full`).
2. Contenido de la Navegación:
    a. Dentro de la sección, hay un conjunto de botones que representan diferentes opciones de navegación.
    b. Cada botón tiene un ícono y un texto asociado.
    c. Se utilizan clases CSS para aplicar estilos de diseño (`flex justify-between`), tamaño (`w-full`), alineación (`items-center justify-center`), y espaciado (`pt-3 pb-3`).
    d. Se aplican clases de estilos de texto (`text-xs font-semibold text-gray-600 dark:text-night-300`) para el texto dentro de los botones, indicando un tamaño de texto más pequeño, negrita y color de texto diferente dependiendo del tema claro o oscuro (`dark:text-night-300`).
3. Íconos:
    a. Se utilizan íconos SVG dentro de los botones para representar diferentes funciones o secciones de la aplicación.
    b. Los íconos se cargan desde archivos SVG en la carpeta de activos (`assets/icons/heroicons/outline/`) y se especifican mediante la directiva `svg-icon`.
    c. Se aplican clases de tamaño a los íconos (`[svgClass]="'h-5 w-5'"`) para definir su altura y anchura.
4. Texto del Botón:
    a. Cada botón tiene un texto asociado que indica la función o sección representada por el botón.
    b. Se utiliza la etiqueta `<span>` para envolver el texto, y se aplican clases de estilos de texto (`text-xs`) para definir su tamaño.

HTML del footer:

Este código HTML representa una barra de navegación simple que contiene algunos enlaces y texto informativo. Aquí está una explicación detallada:

1. Contenedor de la barra de navegación:
    a. `<div class="flex h-16 items-center bg-white dark:bg-night-700">`: Este `<div>` es el contenedor principal de la barra de navegación. Tiene una altura fija (`h-16`), y los elementos dentro están centrados verticalmente (`items-center`). El fondo es blanco en modo claro (`bg-white`) y gris oscuro en modo oscuro (`dark:bg-night-700`).
2. Contenido de la barra de navegación:
    a. `<div class="flex w-full items-center justify-between px-7 text-[13px] font-medium">`: Este `<div>` contiene el contenido principal de la barra de navegación. Está configurado para ocupar todo el ancho disponible (`w-full`) y tiene un relleno horizontal (`px-7`). El texto dentro de este contenedor tiene un tamaño de 13 píxeles (`text-[13px]`) y se establece como medio (`font-medium`).
3. Información del proyecto:
    a. `<div class="text-gray-400 dark:text-night-300">`: Aquí se muestra información sobre el proyecto. El texto es gris en modo claro (`text-gray-400`) y gris oscuro en modo oscuro (`dark:text-night-300`).
    b. `<span class="mr-2">{{ year }}©</span>`: Esto muestra el año actual (dado por la variable `year`) seguido del símbolo de copyright. El texto tiene un margen derecho de 2 píxeles (`mr-2`).
    c. `<a href="https://github.com/reethfx/IberdrolaApp/" target="_blank" class="font-normal text-gray-800 hover:text-primary-500 dark:text-night-100 dark:hover:text-primary-500">Proyecto Iberdrola</a>`: Este enlace dirige a un proyecto en GitHub llamado "Proyecto Iberdrola". En modo claro, el texto es gris oscuro (`text-gray-800`) y cambia a azul (`hover:text-primary-500`) cuando se pasa el mouse sobre él. En modo oscuro, el color del texto es gris claro (`dark:text-night-100`) y también cambia a azul cuando se pasa el mouse (`dark:hover:text-primary-500`).
4. Enlaces de navegación:
    a. `<ul class="flex text-gray-500 dark:text-night-300">`: Esta lista contiene enlaces adicionales de navegación. Los enlaces están en gris en modo claro (`text-gray-500`) y gris oscuro en modo oscuro (`dark:text-night-300`).
    b. Cada enlace (`<li>`) tiene un enlace (`<a>`) que apunta a diferentes partes del proyecto en GitHub. Estos enlaces tienen el mismo comportamiento de cambio de color al pasar el mouse que el enlace del proyecto.

HTML del navbar:

Este fragmento de código HTML define una barra de navegación que puede cambiar su diseño dependiendo del tamaño de la pantalla. Aquí está la explicación de cada parte:

1. Contenedor Principal (`<div class="relative bg-white dark:bg-night-700">`):
    a. Este div envuelve toda la barra de navegación.
    b. Tiene una clase CSS `relative` para posicionar sus elementos secundarios relativos a él.
    c. Establece un fondo blanco (`bg-white`), que se cambia a un fondo oscuro (`dark:bg-night-700`) en modo oscuro.
2. Contenido de la Barra de Navegación (`<div class="mx-auto px-5">`):
    a. Este div limita el ancho del contenido y agrega un relleno horizontal.
3. Sección de la Barra de Navegación (`<div class="flex items-center justify-between py-3.5 md:justify-start">`):
    a. Esta sección contiene los elementos principales de la barra de navegación y ajusta su disposición dependiendo del tamaño de la pantalla.
4. Botón del Menú de Navegación para Dispositivos Móviles (`<div class="sm:order-1 md:hidden">`):
    a. Este div contiene un botón para abrir el menú de navegación en dispositivos móviles.
    b. El botón tiene un icono de menú y ejecuta la función `toggleMobileMenu()` cuando se hace clic en él.
5. Logotipo (`<div class="flex items-center justify-start sm:order-2 md:mr-10 lg:hidden">`):
    a. Este div muestra el logotipo de la aplicación.
    b. Si el tamaño de la pantalla es grande (lg:hidden), muestra solo el logotipo; de lo contrario, muestra el logotipo junto con el nombre de la aplicación (`Proyecto Iberdrola`).
6. Menú de Navegación para Escritorio (`<div class="hidden space-x-10 sm:order-3 md:flex">`):
    a. Este div contiene el menú de navegación para dispositivos de escritorio.
    b. El menú está oculto en dispositivos móviles (`hidden`) y se muestra en pantallas más grandes (`md:flex`).
7. Menú de Perfil (`<div class="items-center justify-end sm:order-4 md:flex md:flex-1 lg:w-0">`):
    a. Este div muestra el menú de perfil del usuario, que puede contener opciones como "Perfil", "Configuración", etc.
    b. Se ajusta a la derecha de la barra de navegación y ocupa todo el espacio disponible en pantallas grandes.
8. Menú Móvil (`<app-navbar-mobile></app-navbar-mobile>`):
    a. Este componente `app-navbar-mobile` representa el menú de navegación para dispositivos móviles.

b. Se mostrará cuando el usuario haga clic en el botón de menú de navegación para dispositivos móviles.

HTML del sidebar:

Este fragmento de código HTML representa un componente de barra lateral de navegación (`nav`) que puede ocultarse o mostrarse según el estado de `menuService.showSideBar`. Aquí hay una explicación de las partes principales:

1. ngClass:
    a. La directiva `ngClass` se utiliza para aplicar clases condicionalmente en función del valor de `menuService.showSideBar`.
    b. Cuando `menuService.showSideBar` es verdadero, se aplica la clase `w-52 xl:w-64`, lo que determina el ancho de la barra lateral.
    c. Cuando `menuService.showSideBar` es falso, se aplica la clase `w-[70px]`, lo que establece un ancho más estrecho para la barra lateral.
2. Clases CSS:
    a. Se aplican varias clases CSS para estilizar la barra lateral.
    b. `scrollbar-thumb-rounded` y `scrollbar-track-rounded` se utilizan para redondear los bordes del scrollbar.
    c. `hidden` se utiliza para ocultar la barra lateral en dispositivos de pantalla grande (`lg:flex` se encarga de mostrarla en dispositivos grandes).
    d. `bg-white` establece el color de fondo de la barra lateral en blanco.
    e. `dark:bg-night-700` se utiliza para establecer el color de fondo en modo oscuro.
    f. `pt-3` establece el relleno superior de la barra lateral.
3. Contenido de la barra lateral:
    a. La sección `px-4` contiene el contenido principal de la barra lateral.
    b. Incluye el logo de la aplicación, un botón para alternar la visibilidad de la barra lateral, un separador (`<hr>`), y elementos de menú como la opción de cambiar entre temas, cerrar sesión y mostrar la versión de la aplicación.
4. Botón de alternar barra lateral:
    a. El botón en la esquina superior derecha se utiliza para alternar la visibilidad de la barra lateral.
    b. Al hacer clic en el botón, se llama al método `toggleSidebar()`.
5. Elementos del menú:
    a. Los elementos del menú incluyen opciones como cambiar entre temas claro y oscuro, cerrar sesión y mostrar la versión de la aplicación.
    b. Cada opción tiene un icono (`<svg-icon>`) y un enlace o información asociada.
    c. Se aplican estilos de hover para resaltar las opciones cuando se pasa el cursor sobre ellas.

6. Versionado:
    a. Se muestra la versión de la aplicación, y al hacer clic en ella se dirige al usuario al repositorio de GitHub de la aplicación en una nueva pestaña del navegador.

Routing Module TS:

Este fragmento de código pertenece a un módulo de enrutamiento en una aplicación Angular. Aquí está la explicación:

1. Imports:
    a. Se importan los módulos necesarios de Angular para trabajar con el enrutamiento: `NgModule`, `RouterModule` y `Routes`.
2. Definición de Rutas:
    a. Se define un conjunto de rutas en la constante `routes`. Cada ruta es un objeto con propiedades que especifican la URL (`path`) y el componente que se mostrará cuando se navegue a esa URL (`component`).
    b. En este caso, hay dos rutas principales:
    c. `dashboard`: Cuando la URL es `/dashboard`, se carga el componente `LayoutComponent` y se carga de forma dinámica (`lazy loading`) el módulo `DashboardModule`.
    d. `config`: Cuando la URL es `/config`, también se carga el componente `LayoutComponent` y se carga de forma dinámica el módulo `ConfigModule`.
    e. Además, hay dos rutas adicionales:
    f. `''` (ruta vacía): Redirige al usuario automáticamente a la ruta `/dashboard` cuando no se proporciona ninguna URL específica.
    g. `'**'` (cualquier otra ruta): Redirige al usuario a una página de error (`error/404`) cuando se navega a una URL no válida.
3. *NgModule Decorator:
    a. El decorador `@NgModule` se utiliza para configurar el módulo de enrutamiento.
    b. Se especifica que este es un módulo de enrutamiento para un módulo secundario utilizando `RouterModule.forChild(routes)`.
    c. `exports: [RouterModule]` asegura que el módulo `RouterModule` esté disponible para su uso fuera de este módulo.
4. Exportación del Módulo de Enrutamiento**:
    a. Finalmente, el módulo de enrutamiento se exporta como `LayoutRoutingModule`, lo que permite que otros módulos de la aplicación lo importen y lo utilicen en su configuración de enrutamiento.

Módulo de Dashboard

HTML de contratos:

Este fragmento de HTML describe una estructura de página que incluye varios componentes personalizados (probablemente de Angular) para una interfaz de usuario relacionada con contratos. Aquí está la explicación de cada parte:

1. Div Contenedor Principal:
    a. `<div class="mx-auto px-4 py-4 sm:px-8 lg:container">`: Este `div` sirve como un contenedor principal con estilos aplicados para centrar el contenido horizontalmente (`mx-auto`), proporcionar relleno (`px-4 py-4`), y definir diferentes rellenos laterales en pantallas pequeñas (`sm:px-8`).
2. Header de Contratos:
    a. `<db-contrato-header></db-contrato-header>`: Esto es un componente personalizado llamado `db-contrato-header` que representa el encabezado de una sección de contratos. Este componente se inserta directamente en el HTML sin ninguna estructura adicional.
3. Barra de Búsqueda de Contratos:
    a. `<div class="my-5 grid grid-cols-1 lg:grid-cols-3 gap-4">`: Aquí se define un `div` con estilos aplicados para definir un margen vertical (`my-5`) y para establecer un diseño de cuadrícula con una columna en pantallas pequeñas y tres columnas en pantallas grandes (`lg:grid-cols-3`). El espacio entre las celdas de la cuadrícula se establece en `gap-4`.
    b. `<db-contrato-sb class="items-center col-span-1"></db-contrato-sb>`: Parece ser otro componente personalizado llamado `db-contrato-sb` que se coloca dentro de una celda de la cuadrícula (`col-span-1`). Además, se aplica una clase `items-center` para centrar el contenido verticalmente dentro de la celda.
4. Tabla de Contratos:
    a. `<db-contrato-table class="col-span-1"></db-contrato-table>`: Este es otro componente personalizado llamado `db-contrato-table` que se coloca dentro de una celda de la cuadrícula (`col-span-1`).

HTML de database:

Este fragmento de HTML representa una estructura de página que contiene un contenedor principal con algunas secciones anidadas. Aquí está la explicación de cada parte:

1. Contenedor Principal:
    a. Se utiliza un `<div>` con la clase CSS `mx-auto px-4 py-4 sm:px-8 lg:container` para crear un contenedor que se centra horizontalmente en la página y tiene un relleno interno específico en diferentes tamaños de pantalla.
2. Sección del Encabezado (Header):
    a. Hay un comentario HTML `<!-- Header -->`, pero no hay ningún contenido dentro de esta sección en el fragmento proporcionado.

3. Contenedor de la Tabla de Clientes:
    a. Se utiliza un `<div>` con la clase CSS `grid grid-cols-1 gap-4 md:grid-cols-2 lg:grid-cols-2 xl:grid-cols-4` para crear un contenedor con un sistema de cuadrícula flexible que cambia el número de columnas dependiendo del tamaño de la pantalla.
    b. Dentro de este contenedor de cuadrícula, hay otro `<div>` con la clase CSS `md:col-span-2 xl:col-span-3`. Esto define el tamaño de la columna en diferentes tamaños de pantalla. Por ejemplo, en pantallas medianas (`md`), esta columna ocupará 2 columnas, y en pantallas extra grandes (`xl`), ocupará 3 columnas.
    c. Además, hay un atributo `db-cliente-table` en este div. Este atributo podría ser utilizado para identificar este elemento específico y aplicar lógica o estilos adicionales basados en él, aunque esto depende de la implementación del proyecto.

HTML de facturas:

Este fragmento de código HTML representa una estructura de página que consiste en un encabezado y un contenido, con un diseño flexible que se adapta a diferentes tamaños de pantalla.

1. Div contenedor principal:
    a. Tiene clases de diseño `mx-auto` para centrar horizontalmente el contenido.
    b. También tiene clases de relleno (`px-4 py-4 sm:px-8`) que definen los espacios internos del contenedor en diferentes tamaños de pantalla.
    c. Para pantallas grandes (`lg:container`), se utiliza la clase `container`, que define un ancho máximo y centraliza el contenido.
2. Encabezado:
    a. Se incluye un componente llamado `db-factura-header`, que presumiblemente renderizará el encabezado de una factura.
3. Contenido:
    a. Este bloque contiene el contenido principal de la página.
    b. Se organiza en una cuadrícula (`grid`) con una sola columna en dispositivos pequeños y tres columnas en dispositivos grandes.
    c. Se aplica un espacio entre las columnas (`gap-4`) para separarlas.
    d. Se incluye un componente llamado `db-factura-table` que, posiblemente, represente una tabla de detalles de factura.
    e. La clase `col-span-1` se aplica al componente `db-factura-table` para que ocupe una sola columna en dispositivos grandes (una de las tres columnas disponibles).

Routing Module TS:

Este fragmento de código es un módulo de enrutamiento (`DashboardRoutingModule`) en Angular. Este módulo se encarga de definir las rutas para el componente `DashboardComponent` y sus componentes hijos.
Aquí hay una explicación detallada:

1. Imports:
    a. Se importan las clases `NgModule`, `RouterModule` y `Routes` desde el módulo `@angular/core` y `@angular/router`, respectivamente. Estas clases son necesarias para definir y configurar las rutas en Angular.
    b. También se importan los componentes que se utilizarán en las rutas.
2. Definición de Rutas:
    a. Se define una constante `routes` que es un array de objetos `Routes`. Cada objeto representa una ruta específica.
    b. La ruta principal (`path: ''`) está asociada al componente `DashboardComponent`.
    c. Se definen rutas hijas dentro del componente `DashboardComponent` utilizando la propiedad `children`.
    d. Cada ruta tiene una URL (`path`) y un componente asociado (`component`). Algunas rutas tienen parámetros dinámicos en la URL, como `:dni` y `:id`, que se pueden acceder dentro de los componentes asociados.
3. Configuración del Router Module:
    a. Se utiliza el método `forChild()` de `RouterModule` para configurar el módulo de enrutamiento con las rutas definidas anteriormente.
    b. Finalmente, el módulo `DashboardRoutingModule` se exporta para que pueda ser utilizado por otros módulos de Angular.

Trabajos futuros

Consideramos que este proyecto puede ser ampliamente ampliable en un futuro dotando de una mayor cantidad de características y funcionalidades, además, hay preparados módulos para una futura ampliación del mismo.