INSTITUTO TECNOLÓGICO SUPERIOR DE CALKINÍ

INGENIERIA EN SISTEMAS COMPUTACIONALES

SEPTIMO SEMESTRE

AEB-1055 PROGRAMACIÓN WEB

DOCENTE: DRA. YAQUELINE PECH HUH

Manual Frontend

INTEGRANTES DEL EQUIPO:

- CHAN CHIN CRISMAR DE JESUS MATRICULA: 7612.
- EK MAS DYLAN FERNANDO MATRICULA: 7618.
- KU CENTENO BERENISE ANTONIA MATRICULA: 7628.
- MAY VERGARA ROBERTO CARLOS MATRICULA: 7630.
- UC PERALTA ADIEL EDUARDO MATRICULA: 7666.

GRUPO: “A”

Contenido

Framework

Descripción

Angular es un framework de desarrollo de aplicaciones web de código abierto, mantenido principalmente por Google, que permite construir aplicaciones dinámicas y de alto rendimiento en el lado del cliente. Angular utiliza TypeScript como lenguaje base, proporcionando tipado estático y funcionalidades orientadas a objetos que mejoran la experiencia de desarrollo y la calidad del código.

Este framework sigue una arquitectura basada en componentes, lo que permite dividir la aplicación en unidades reutilizables y modulares, facilitando tanto el desarrollo como el mantenimiento.

Angular también cuenta con un CLI (Command Line Interface) que permite agilizar tareas comunes, como la generación de componentes, servicios, directivas, y la configuración de compilación y despliegue, ofreciendo una estructura consistente en el proyecto. La robustez de Angular, su enfoque modular y su ecosistema extenso lo hacen ideal para construir aplicaciones escalables y bien estructuradas.

Versión:

Framework: Angular

Versión: 16.2.0

Requisitos previos:

- Node.js: Versión 18.20.4
- Angular CLI: Versión 16.2.15
  Comando para instalar: npm install -g @angular/cli
- Editor de texto recomendado: Visual Studio Code.

Dependencias y versiones

1. **@ng-bootstrap/ng-bootstrap**
   - **Versión necesaria**: ^15.1.2
   - **Descripción**: Proporciona componentes de Bootstrap adaptados para Angular, facilitando la integración de elementos de Bootstrap en aplicaciones Angular.
2. **@popperjs/core**
   - **Versión necesaria**: ^2.11.6
   - **Descripción**: Biblioteca utilizada para gestionar y posicionar elementos emergentes en la interfaz, como tooltips y dropdowns.
3. **bootstrap**
   - **Versión necesaria**: ^5.3.3
   - **Descripción**: Framework de CSS popular para crear diseños web responsive y estilizar componentes de interfaz.
4. **bootstrap-icons**
   - **Versión necesaria**: ^1.11.3
   - **Descripción**: Conjunto de íconos SVG diseñados para ser utilizados con Bootstrap, que también funcionan de manera independiente.
5. **jquery**
   - **Versión necesaria**: ^3.7.1
   - **Descripción**: Biblioteca JavaScript para manipulación de elementos DOM y manejo de eventos. Aunque no es nativa de Angular, algunos proyectos la requieren para funciones adicionales.
6. **jspdf**
   - **Versión necesaria**: ^2.5.2

- **Descripción**: Herramienta para generar documentos PDF desde el lado del cliente en JavaScript. Útil para crear reportes o archivos descargables.

7. **ngx-ui-loader**
   - **Versión necesaria**: ^13.0.0
   - **Descripción**: Módulo de Angular para mostrar un cargador en la interfaz, útil para gestionar cargas y mejorar la experiencia de usuario mientras se procesan datos.
8. **pdfmake**
   - **Versión necesaria**: ^0.2.14
   - **Descripción**: Biblioteca de JavaScript que permite la creación de documentos PDF directamente en el navegador, alternativa o complemento a jspdf.
9. **rxjs**
   - **Versión necesaria**: ~7.8.0
   - **Descripción**: Biblioteca para programación reactiva que es esencial en Angular para gestionar flujos de datos asíncronos.
10. **sweetalert2**
    - **Versión necesaria**: ^11.14.0
    - **Descripción**: Biblioteca para mostrar alertas personalizables y con estilo en la aplicación, mejorando la interacción con el usuario.
11. **tslib**
    - **Versión necesaria**: ^2.3.0
    - **Descripción**: Biblioteca auxiliar para TypeScript que optimiza la creación de código reutilizable en aplicaciones.
12. **zone.js**
    - **Versión necesaria**: ~0.13.0

- **Descripción**: Biblioteca utilizada por Angular para gestionar la detección de cambios en la aplicación y la ejecución asíncrona.

13. ng-apexcharts

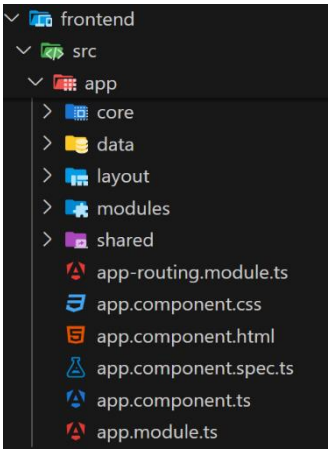
- **Versión necesaria:** ^3.54.1
- **Descripción:** Biblioteca de gráficos de JavaScript que permite la creación de gráficos interactivos y responsivos. Proporciona una variedad de tipos de gráficos como líneas, barras, áreas, pastel, entre otros, facilitando la visualización de datos en aplicaciones web.

Este comando instalará todas las dependencias y sus versiones especificadas:

```
npm install
```

Estructura de carpetas

La estructura del proyecto está organizada de la siguiente manera



app: Contiene la lógica principal de la aplicación. En esta carpeta se encuentran los módulos, componentes, y servicios que forman el núcleo de la aplicación.

core: Incluye elementos centrales de la aplicación.

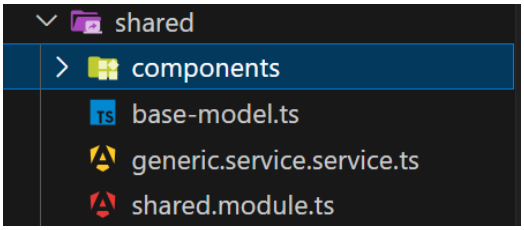
data: Carpeta destinada a manejar los modelos de datos, clases y constantes que representan la información con la que interactúa la aplicación.

layout: Contiene los componentes y módulos relacionados con el diseño o estructura general de la interfaz, como el encabezado, el pie de página, etc.

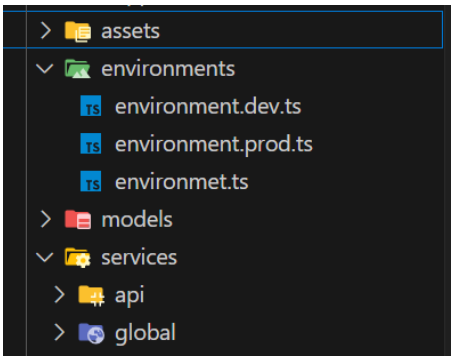
modules: Aquí se alojan los diferentes módulos de la aplicación, cada uno representando una funcionalidad específica.

shared: Incluye elementos que se comparten entre varios módulos o componentes.

- **components**: Contiene los componentes reutilizables que pueden ser utilizados en distintas partes de la aplicación.



- **generic.service.ts**: Servicio genérico que maneja las operaciones CRUD de manera genérica para facilitar el uso de servicios comunes.
- **base-model.ts**: Archivo que define la estructura de un modelo básico que puede ser extendido por otros modelos en la aplicación.



assets: Carpeta donde se almacenan los archivos estáticos, como imágenes, íconos, archivos de estilos, etc.

environments: Almacena configuraciones de entorno para desarrollo, pruebas y producción. Esto permite cambiar la configuración de la aplicación según el ambiente en el que se ejecuta.

models: Carpeta donde se definen los modelos de datos de la aplicación. Estos representan las estructuras de datos con las que trabaja la aplicación.

services: Contiene los servicios de la aplicación que son responsables de conectar la lógica de negocio con las operaciones de datos.

Servicio genérico

El servicio genérico es un servicio que maneja las operaciones básicas de CRUD (create, getAll, update y delete) para cualquier modelo de datos que implemente la interfaz IBaseModel. Esto facilita la reutilización de código, ya que un solo servicio puede manejar múltiples modelos, reduciendo la necesidad de crear servicios específicos para cada entidad.

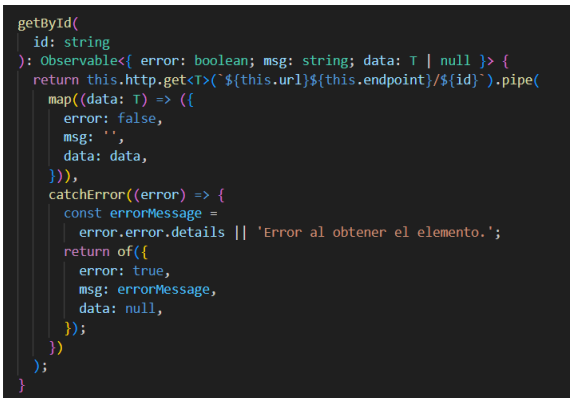
getAll: Obtiene todos los elementos de la entidad especificada.

- **Retorna**: Observable que contiene un objeto con las propiedades error, msg, y data.
- **Manejo de Errores**: Si ocurre un error, devuelve error: true y un mensaje indicando el problema.

```
getAll(): Observable<{
  error: boolean;
  msg: string;
  data: T[] | null;
}> {
  return this.http.get<T[]>(`${this.url}${this.endpoint}`).pipe(
    map((data: T[]) => ({
      error: false,
      msg: 'Datos recuperados con éxito',
      data: data,
    })),
    catchError((error) => {
      console.error('Error fetching data:', error);
      const errorMessage = error.error.details || 'Error del servidor';
      return of({
        error: true,
        msg: errorMessage,
        data: null,
      });
    })
  );
}
```

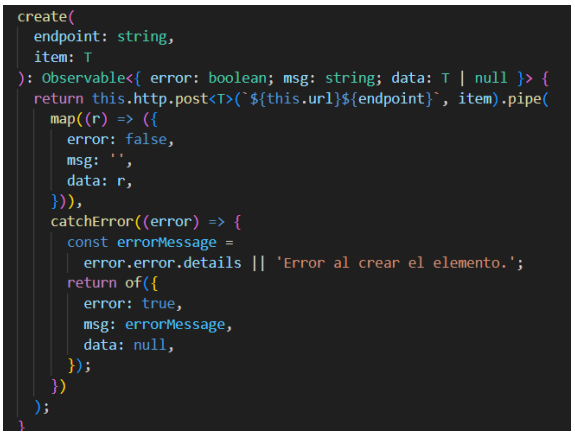
getById: Obtiene un elemento específico por su ID.

- **Argumento**: id (string).
- **Retorna**: Observable que contiene el objeto encontrado o un mensaje de error en caso de fallo.



create: Crea un nuevo elemento en la entidad especificada.

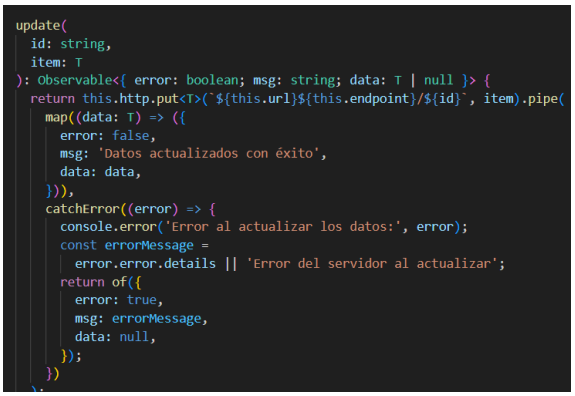
- **Argumentos**:
  - endpoint (string) — el endpoint específico donde se realizará la creación.
  - item (T) — el objeto que se creará.



- **Retorna**: Observable con el objeto creado o un mensaje de error si falla la operación.

update: Actualiza un elemento existente por su ID.

- **Argumentos**:
  - id (string) — el identificador del elemento a actualizar.
  - item (T) — el objeto con los datos actualizados.



- **Retorno**: Observable con el objeto actualizado o un mensaje de error en caso de falla.

delete: Elimina un elemento existente por su ID.

- **Argumento**: id (string) — el identificador del elemento a eliminar.
- **Retorno**: Observable<void>, indicando que la operación de eliminación fue exitosa o un error en caso de fallo.

Creación de archivos

Components

- **Descripción:** Los componentes son las unidades principales de la interfaz en Angular que contienen la lógica de presentación y definen una vista específica en la aplicación. Cada componente tiene una plantilla HTML, una hoja de estilos y un archivo TypeScript que encapsula su lógica. Los componentes ayudan a modularizar la aplicación en partes reutilizables y manejables.
- **Comando de creación:**

  `ng generate component nombre-del-componente`

  Ejemplo:

```
PS C:\Users\crizm\albergueproyecto1\Equipo_5\frontend\src\app\layout> ng generate component examplecomponent
CREATE src/app/layout/examplecomponent/examplecomponent.component.html (31 bytes)
CREATE src/app/layout/examplecomponent/examplecomponent.component.spec.ts (629 bytes)
CREATE src/app/layout/examplecomponent/examplecomponent.component.ts (242 bytes)
CREATE src/app/layout/examplecomponent/examplecomponent.component.css (0 bytes)
UPDATE src/app/app.module.ts (2050 bytes)
```

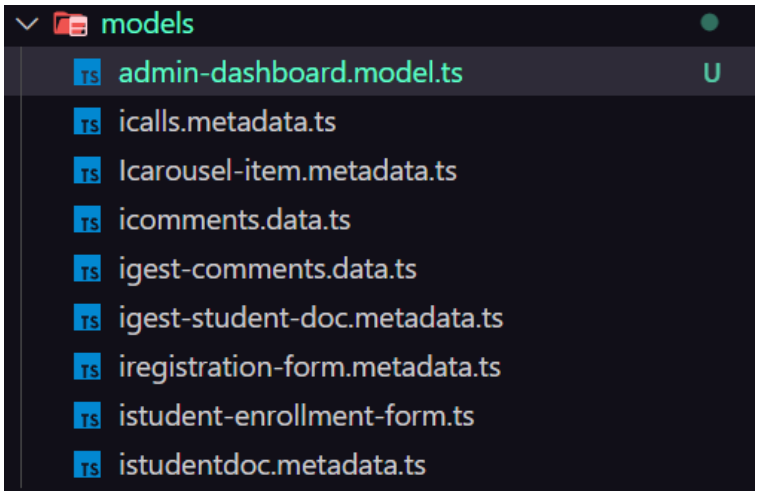
  Se ha creado el componente correctamente que consiste en una carpeta con 4 archivos.

- **ejexamplecomponent.component.ts**: define la lógica y funcionalidad del componente.
- **ejexamplecomponent.component.html**: define la estructura HTML que se mostrará en la interfaz.
- **ejexamplecomponent.component.css** : contiene los estilos específicos del componente.
- **ejexamplecomponent.component.spec.ts**: archivo para pruebas unitarias.

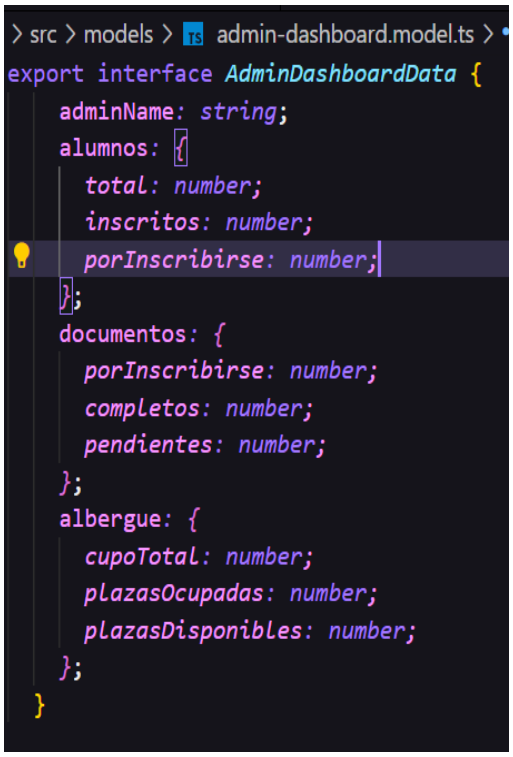
Models

- **Descripción:** Los modelos representan la estructura de datos utilizada en la aplicación. Sirven como una representación de los datos que se manejarán, facilitando la interacción con el backend y la consistencia en el tipo de datos. Los modelos permiten definir la forma de los objetos de datos, lo que ayuda a evitar errores y asegura una correcta tipificación.
- **Creación:** Los modelos se crean manualmente en el proyecto. Como buena práctica, los modelos se organizan en una carpeta llamada models dentro de la estructura del proyecto, en la ruta src/models. En esta carpeta se encuentran los archivos de cada modelo que representan la estructura de datos.

  Y aquí se encuentran los modelos ya creados en el proyecto:



  A continuación, se presentan dos ejemplos


src > models > admin-dashboard.model.ts
export interface AdminDashboardData {
adminName: string;
alumnos: {
total: number;
inscritos: number;
porInscribirse: number;
};
documentos: {
porInscribirse: number;
completos: number;
pendientes: number;
};
albergue: {
cupoTotal: number;
plazasOcupadas: number;
plazasDisponibles: number;
};
}


nd > src > models > icomments.data.ts > ..
import { IBaseModel }
from '@shared/base-model';
export interface IComments
extends IBaseModel {
comment: string;
createdAt: string;
createdBy: string;
}

Services

- **Descripción:** Los servicios manejan la lógica de negocio de la aplicación y permiten la comunicación entre el frontend y el backend. A través de los servicios, Angular puede realizar peticiones HTTP, acceder a datos compartidos y ejecutar funciones complejas que no deberían residir en los componentes. Esto mantiene el código modular y facilita la reutilización.
- **Comando de creación:**
  `ng generate service nombre-del-servicio`

Ejemplo: crearemos un servicio llamado service-example dentro de una carpeta del mismo nomnre, que a su vez estará contenida en la carpeta del proyecto destinada para los servicios.

El comando sería `ng generate service service-example/service-example`

```
PS C:\Users\crizm\albergueproyecto1\Equipo_5\frontend\src\services> ng generate service service-example/service-example
CREATE src/services/service-example/service-example.service.spec.ts (398 bytes)
CREATE src/services/service-example/service-example.service.ts (143 bytes)
```

Así es como se ve un servicio generado:

Este comando ha creado dos archivos:

- **service-example.service.ts**: contiene el código principal del servicio.
- **service-example.service.spec.ts**: archivo para pruebas unitarias.

Y en el archivo service-example.service.ts es donde se escribe el código que comunicará el frontend y backend, porque allí es donde se realizarán las peticiones HTTP al backend.