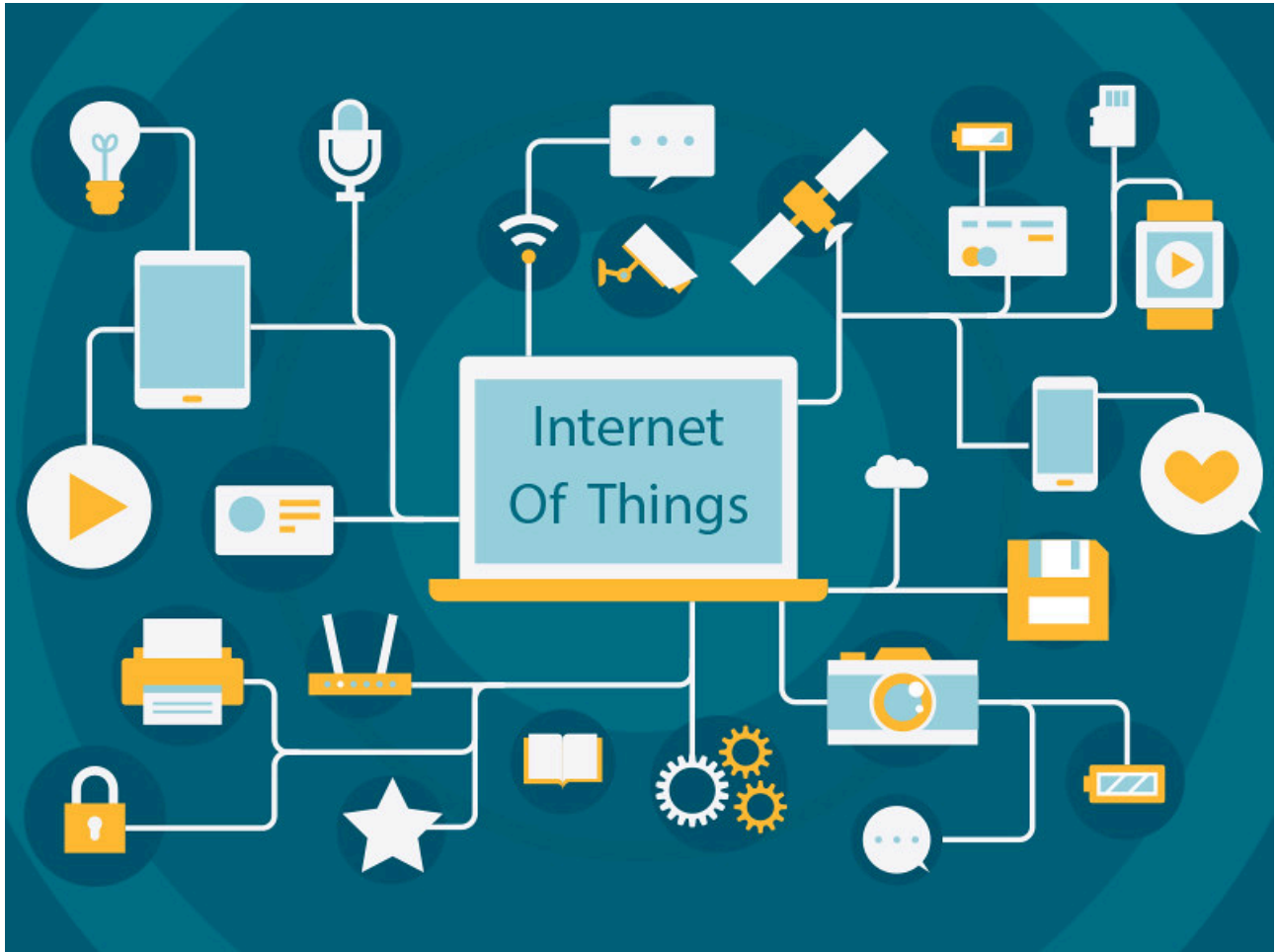


Trabajo de Fin de Grado Superior DAM

DANIEL RODRÍGUEZ VILCHES 2ºDAM



Enlaces:

font [github](#)
arduino [tinkercad](#)

LICENCIA	3
INTRODUCCIÓN	4
TECNOLOGÍAS	4
Detalles técnicos interesantes	5
Problemas	5
Mejoras posibles	5
Conclusiones	5
Bibliografía/Webgrafía	5
Desarrollo	6
Entorno de desarrollo	6
Historias de Usuario	10
Requisitos no funcionales	11
Diagrama Casos de Uso	12
Diagrama de Clase Entidad Relación	12
Estructura Aplicación Angular	13
Imágenes ilustrativas	14
Aclaración final	15
Mockup front	16
Infraestructura de Arduino	16
Componentes	16
Esquema eléctrico	17
Explicación del funcionamiento del sistema Arduino	18
Código y explicación	19
Configuración del hardware	19
Función setup()	20
Función loop()	21
Funciones Auxiliares	22
Conexión con InfluxDB	23
Funcionamiento de la aplicación Angular	24
Estructura	24
Librerías Externas	24
Gráficos	24
Estilos	24
Módulo	25
Conexión con InfluxDB	26

LICENCIA



Reconocimiento-NoComercial-CompartirIgual CC BY-NC-SA

<https://creativecommons.org/>

Autor: Daniel Rodríguez Vilches

Esta licencia permite a otros remezclar, adaptar y desarrollar su trabajo sin fines comerciales, siempre y cuando le den crédito y concedan licencias para sus nuevas creaciones bajo los mismos términos.

INTRODUCCIÓN

El objetivo del proyecto es desarrollar un sistema que mejore la experiencia de los usuarios de una biblioteca.







Para ello se desarrolla una aplicación con información sobre los asientos ocupados, reservados y libres., de tal manera que las personas que deseen ir a la biblioteca tengan una visión clara de la situación de cada sala de la biblioteca y, si lo desean, puedan incluso reservar un asiento.

También se monitoriza la temperatura de cada sala de la biblioteca para que los usuarios tengan esta información clara por si la precisan.

Además se capta el nivel de luminosidad de cada asiento ocupado en la sala. Información que es utilizada para que una luz se ilumine dependiendo de esta.

Todo ello será posible gracias a un sistema sensores y actuadores instalados en la biblioteca.

TECNOLOGÍAS

-  Arduino
 - Plataforma de creación de electrónica de código abierto que se basa en hardware y software libre.
 - [Sitio Web Oficial](#)
-  Angular 16
 - Framework de desarrollo basado en Typescript para la creación de aplicaciones web.
 - [Sitio Web Oficial](#)
-  C++
 - Lenguaje de programación utilizado para programar la lógica de componentes Arduino.
 - [Sitio Web Oficial](#)
-  InfluxDB
 - Plataforma que actúa como gestor de base datos enfocado a guardar datos obtenidos de elementos IOT y monitorización, también ofrece distintas herramientas para poder visualizar estos datos en forma de gráficas y estadísticas.
 - [Sitio Web Oficial](#)
-  Docker
 - Plataforma de código abierto que permite a los desarrolladores crear, desplegar, ejecutar y gestionar contenedores, que son componentes estandarizados y ejecutables que combinan el código fuente de aplicación con las dependencias y las bibliotecas del sistema operativo (SO) necesarias para ejecutar dicho código en cualquier entorno.
 - [Sitio Web Oficial](#)
-  GIT
 - Control de versiones para programas informáticos.
 - [Sitio Web Oficial](#)

Detalles técnicos interesantes

InfluxDB cuenta con su propia API para poder insertar datos desde Arduino como para obtenerlos de su base de datos por lo que no es necesario crear una API para el proyecto.

Para la aplicación frontend Angular se ha construido con el siguiente comando:

ng new tfg-font --package-manager pnpm --no-standalone

Esto quiere decir que:

1. El administrador de paquetes de nodejs, que por lo general suele ser npm, se ha asignado que sea pnpm. El cuál es mucho más rápido y aligera enormemente la tarea de instalación y compilación de librerías para la aplicación.
2. El parámetro --no-standalone quiere decir que los componentes generados no serán autónomos y deberán ser declarados en módulos NgModule existentes en la aplicación.

Esto sigue el enfoque tradicional de Angular, donde los componentes se declaran y se agrupan en módulos para su organización y reutilización.

Es necesario escribir este parámetro a partir de Angular 17 si se quiere trabajar de esta forma.

Problemas

- El sensor de temperatura capta una temperatura inferior a la esperada.
- El shield wifi adquirido no es compatible (aparentemente) con las librerías que ofrece la API de InfluxDB para conectarse con Arduino.
- Los CORS de la aplicación de Angular no permiten la comunicación con InfluxDB.

Mejoras posibles

- Escanear cada sala de la biblioteca para poder trasladar el mapa escaneado al frontend. De esta forma los usuarios pueden visualizar el mapa de la sala en 3D y tener un mejor juicio de qué asientos les interesa en la sala.
- Analizar el tiempo de ocupación en asientos para que los encargados de la limpieza prioricen las zonas donde desinfectar, preparar, etc... . Por ejemplo: si un determinado espacio de la sala ha sido ocupado más tiempo que otro, se priorizará la limpieza en dicha área. Sería de gran utilidad en caso de, por ejemplo, una epidemia.

Conclusiones

Bibliografía/Webgrafía

Desarrollo

Entorno de desarrollo

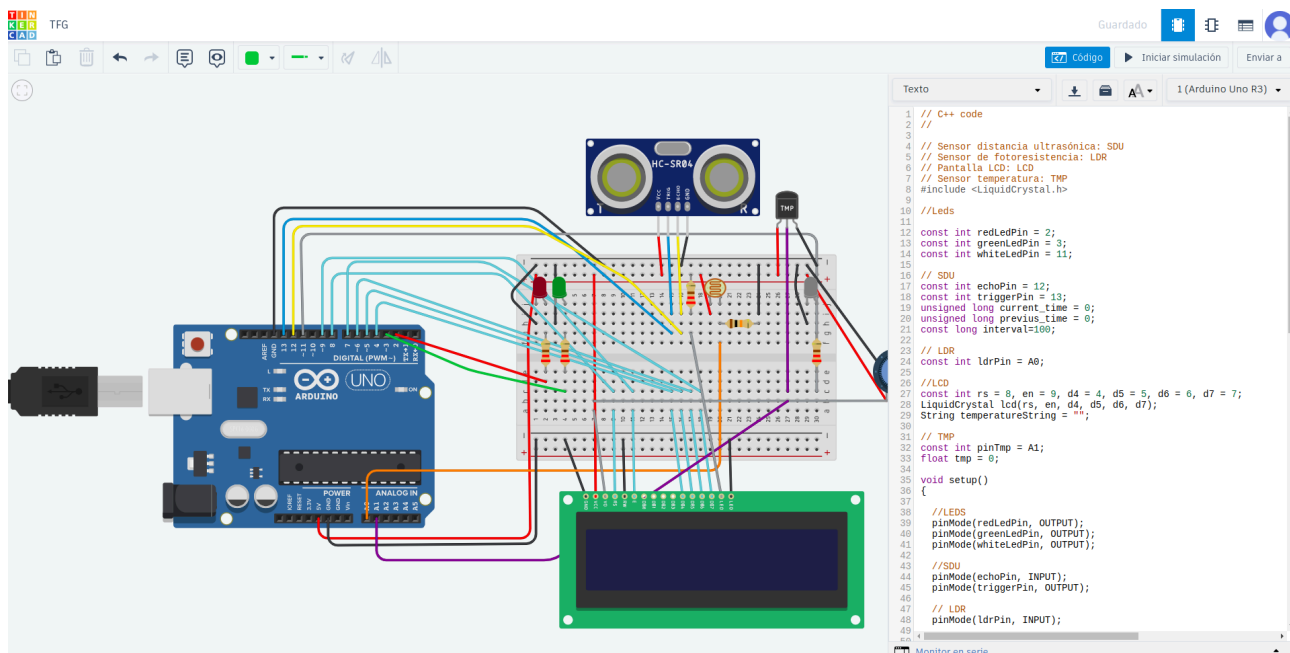
Autodesk Tinkercad:

Es una aplicación online para diseñar proyectos de electrónica en 3D y codificación.

Se ha utilizado en este proyecto para hacer pruebas y diseñar, de la forma más fiel posible a la realidad, la configuración y codificación del Arduino.

Link del proyecto

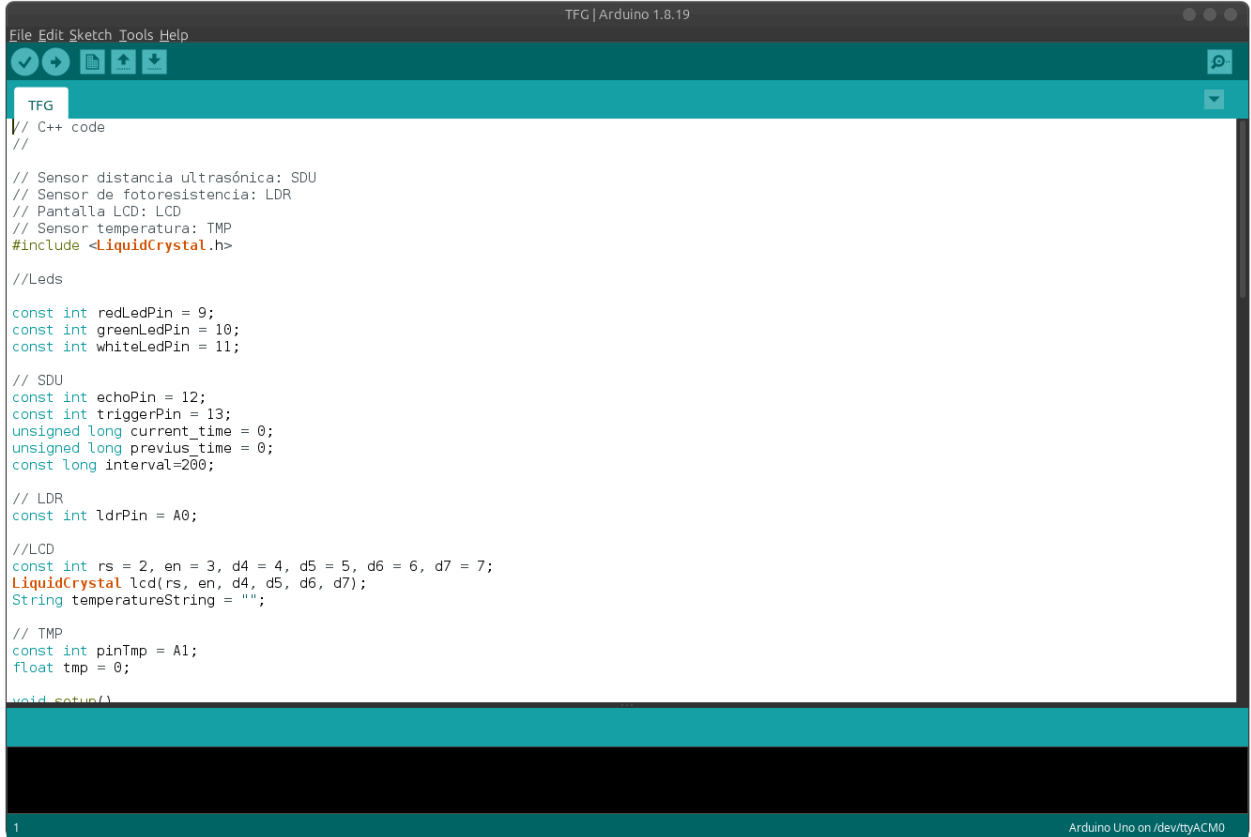
La explicación del diseño se explica en el apartado [Infraestructura de Arduino](#).



Arduino IDE:

Es un editor de código abierto diseñado para desarrollar programas para los componentes de Arduino. Entre las características más útiles de este editor se encuentra la posibilidad de descargar librerías diseñadas para un caso específico dentro de un programa arduino.

En este proyecto se ha utilizado para el desarrollo del programa para el arduino uno.



```
File Edit Sketch Tools Help
TFG | Arduino 1.8.19

// C++ code
//

// Sensor distancia ultrasónica: SDU
// Sensor de fotoresistencia: LDR
// Pantalla LCD: LCD
// Sensor temperatura: TMP
#include <LiquidCrystal.h>

//Leds

const int redLedPin = 9;
const int greenLedPin = 10;
const int whiteLedPin = 11;

// SDU
const int echoPin = 12;
const int triggerPin = 13;
unsigned long current_time = 0;
unsigned long previous_time = 0;
const long interval=200;

// LDR
const int ldrPin = A0;

//LCD
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
String temperatureString = "";

// TMP
const int pinTmp = A1;
float tmp = 0;

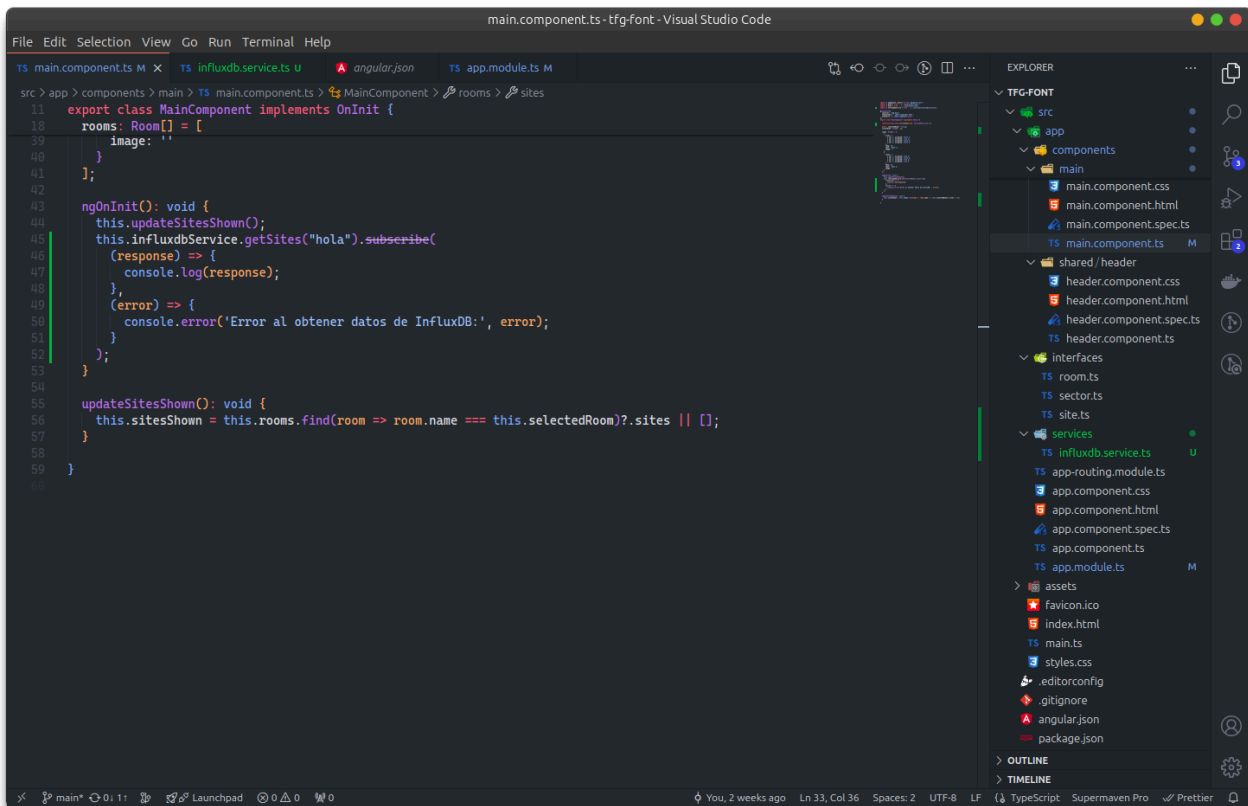
void setup()
```



Visual Studio Code:

Es un editor de código diseñado por Microsoft. Actualmente es el editor de código más utilizado para todo tipo de lenguajes de programación.

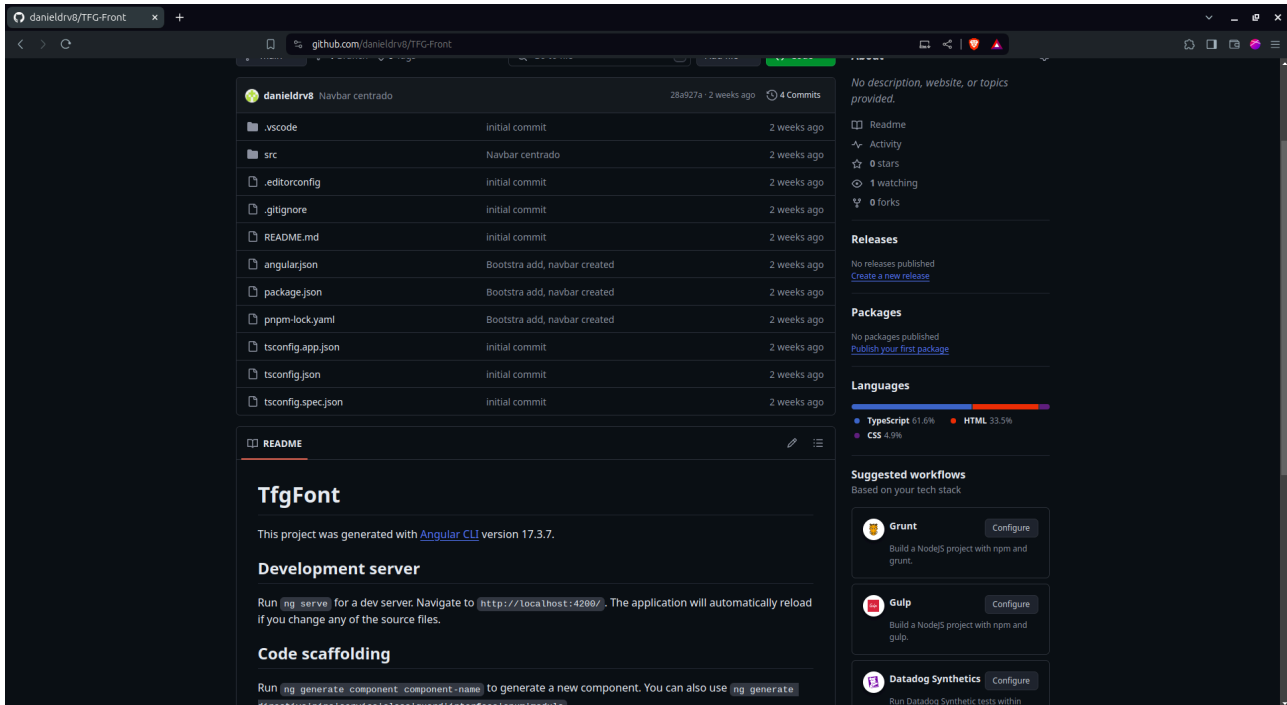
En este caso se ha utilizado para la creación de la aplicación Angular.



GitHub:

Se trata de una plataforma online donde subir repositorios de git y la cual facilita muchísimo el almacenamiento en la nube, la colaboración, etc, de proyectos de desarrollo.

En esta ocasión se ha utilizado github para mantener un mejor control sobre el proyecto y tener una copia de seguridad en la nube del mismo.



Historias de Usuario

Id: 1	Título: Ver temperatura
Descripción: Como Bibliotecario, quiero ver la temperatura de cada sala de la biblioteca	
Estimación: 2	
Prioridad: 1	Dependiente: No

Id: 2	Título: Regulación luz
Descripción: Como Estudiante, quiero la regulación de intensidad automática de una lámpara según la luminosidad	
Estimación: 4	
Prioridad: 1	Dependiente: No

Id: 3	Título: Leds de ocupación
Descripción: Como Estudiante, quiero que una luz verde se encienda en asientos desocupados y que se encienda un led rojo cuando esté ocupado	
Estimación: 2	
Prioridad: 1	Dependiente: No

Id: 4	Título: Vista de ocupación
Descripción: Como Estudiante quiero ver en cualquier dispositivo en tiempo real la ocupación de asientos de la biblioteca Para juzgar si debo o no asistir.	
Estimación: 10	
Prioridad: 1	Dependiente: 3

Id: 5	Título: Media de temperatura
Descripción: Como Bibliotecario quiero ver la media de temperatura, asistencia y luminosidad al mes	

Estimación: 10	
Prioridad: 1	Dependiente: 1, 2, 3, 4

Id: 6	Título: Dividir la información por habitaciones
Descripción: Como Estudiante quiero dividir la información por habitaciones	
Estimación: 2	
Prioridad: 1	Dependiente: 4

Id: 7	Título: Reservar asiento
Descripción: Como Estudiante quiero poder reservar asientos desde la aplicación.	
Estimación: 10	
Prioridad: 1	Dependiente: 4

Requisitos no funcionales

- La aplicación debe estar en funcionamiento las 24 horas del día y tanto sensores como actuadores tienen que estar funcionando, al menos, durante las horas en las que la biblioteca está abierta.
- La información de la aplicación debe ser clara y concisa.

Diagrama Casos de Uso

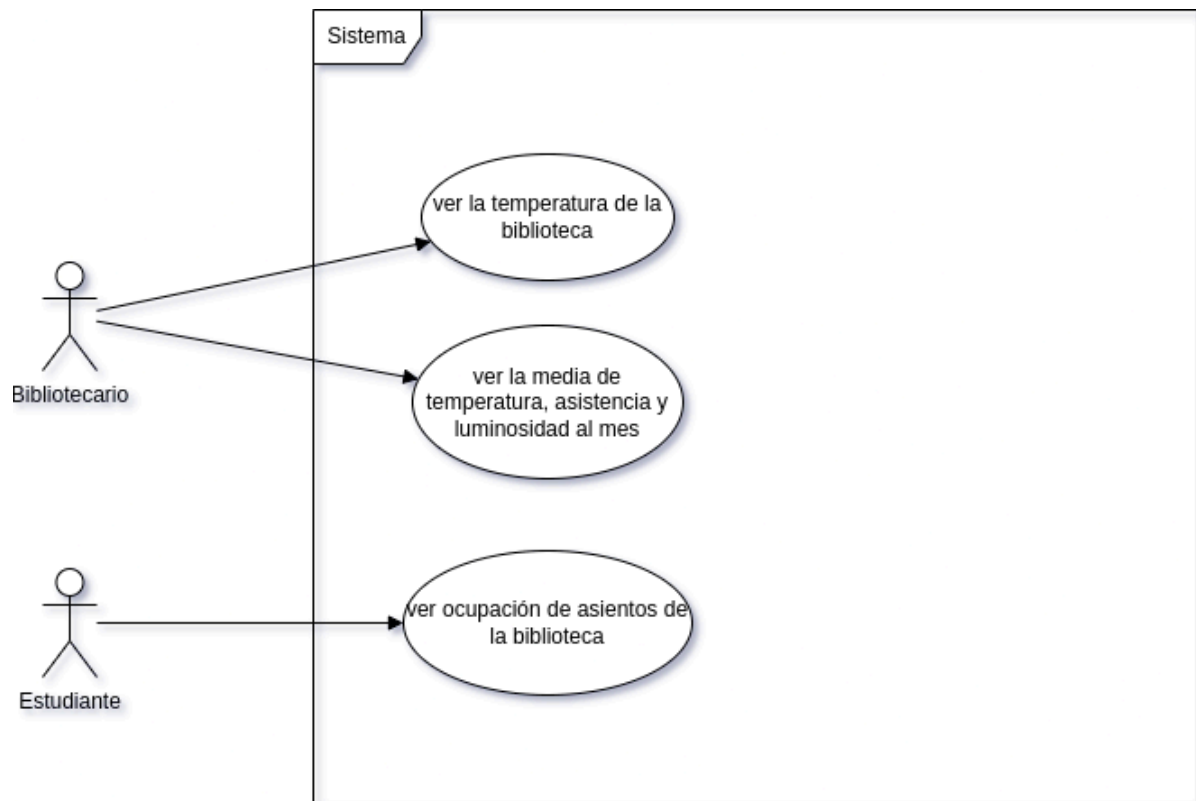
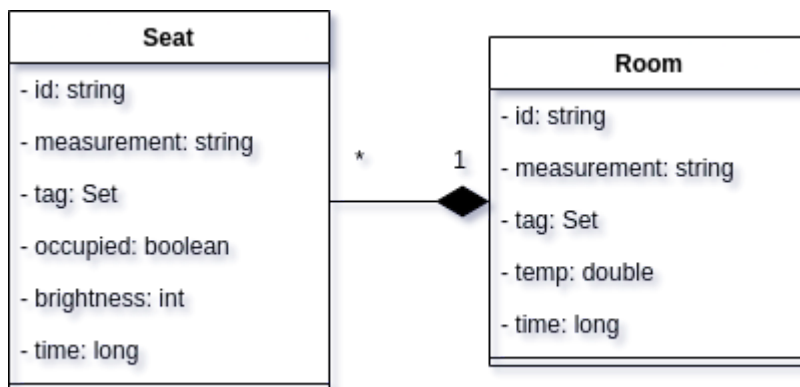


Diagrama de Clase Entidad Relación

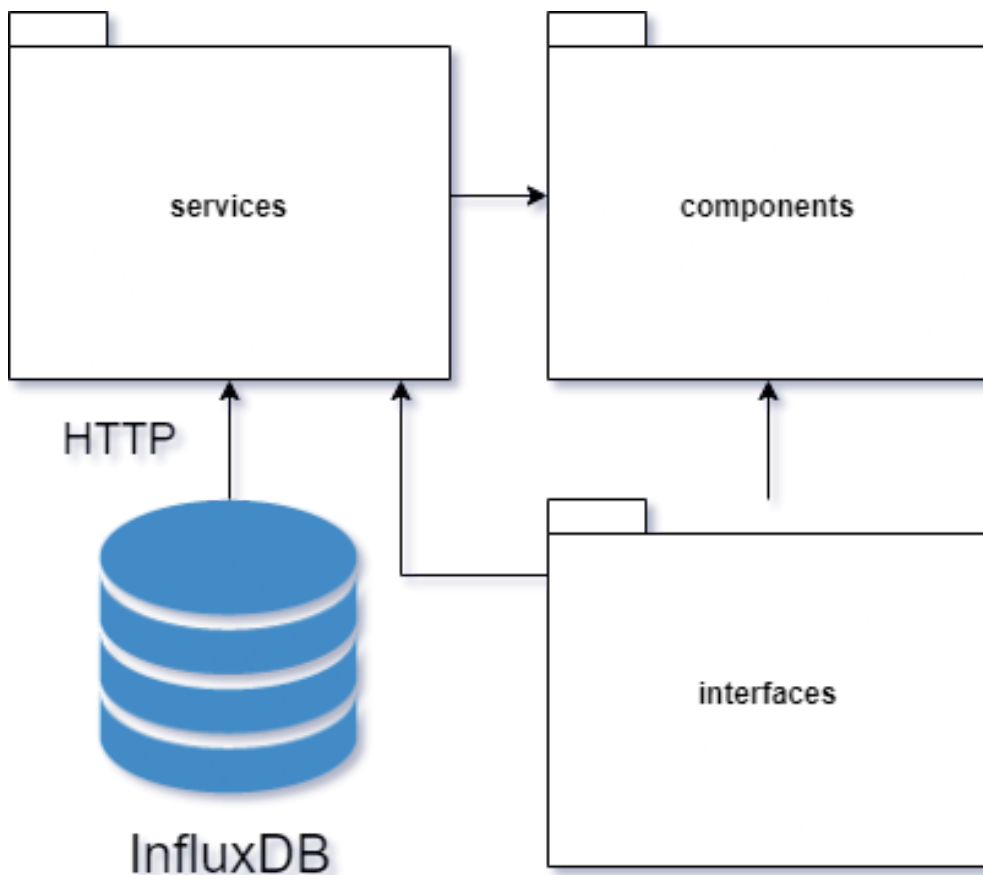


Estructura Aplicación Angular

En este esquema se refleja cómo se estructura la aplicación de Angular. El paquete **services** almacena los servicios encargados de comunicarse con la base de datos de InfluxDB. Estos servicios se inyectarán en los componentes necesarios para que estos puedan hacer

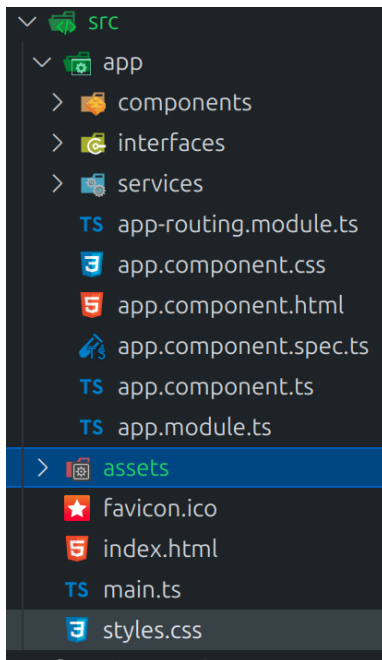
El paquete **componentes** almacena los componentes que conforman la aplicación, contienen la lógica y vistas de la aplicación.

El paquete **interfaces** contiene las interfaces que son los modelos en una aplicación angular. Estos modelos se utilizan tanto en los servicios como en los componentes.

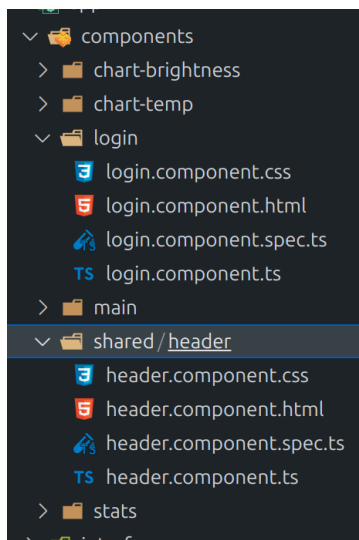


Imágenes ilustrativas

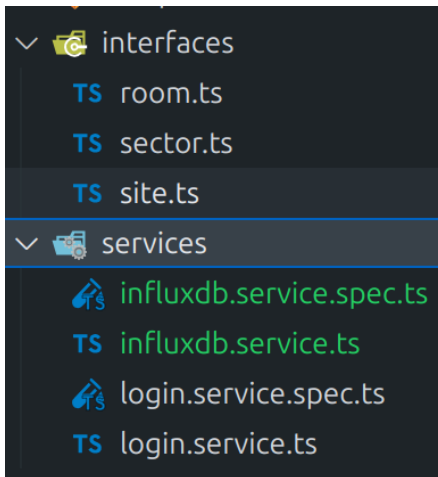
Es posible que ciertos componentes, vistas o servicios no estén presentes en las imágenes de a continuación debido a que fueron sacadas durante el desarrollo de la aplicación.



- Estructura general del proyecto



En esta imagen se aprecia como se estructuran los componentes. En general todos los componentes están al mismo nivel dentro del paquete components excepto algunas excepciones como el componente header que va dentro del paquete shared, puesto que este componente es siempre visible en la aplicación.



Los paquetes interface y services tienen una organización más simple puesto que todos los componentes están al mismo nivel.

Aclaración final

La manera en la que la aplicación se estructura no tiene por que ser la misma, de hecho, suele cambiar dependiendo del tamaño y de la lógica que vaya adoptando el proyecto durante su desarrollo. Aunque siempre es buena práctica tener indicaciones y acuerdos de cómo estructurar los directorios y archivos en un proyecto de desarrollo, sobretodo si en dicho proyecto intervienen varios desarrolladores.

Mockup front

<https://app.moqups.com/eJvVgxs8700kY3axTjIFzqwq1OswH1zl/view/page/ad64222d5>

Infraestructura de Arduino

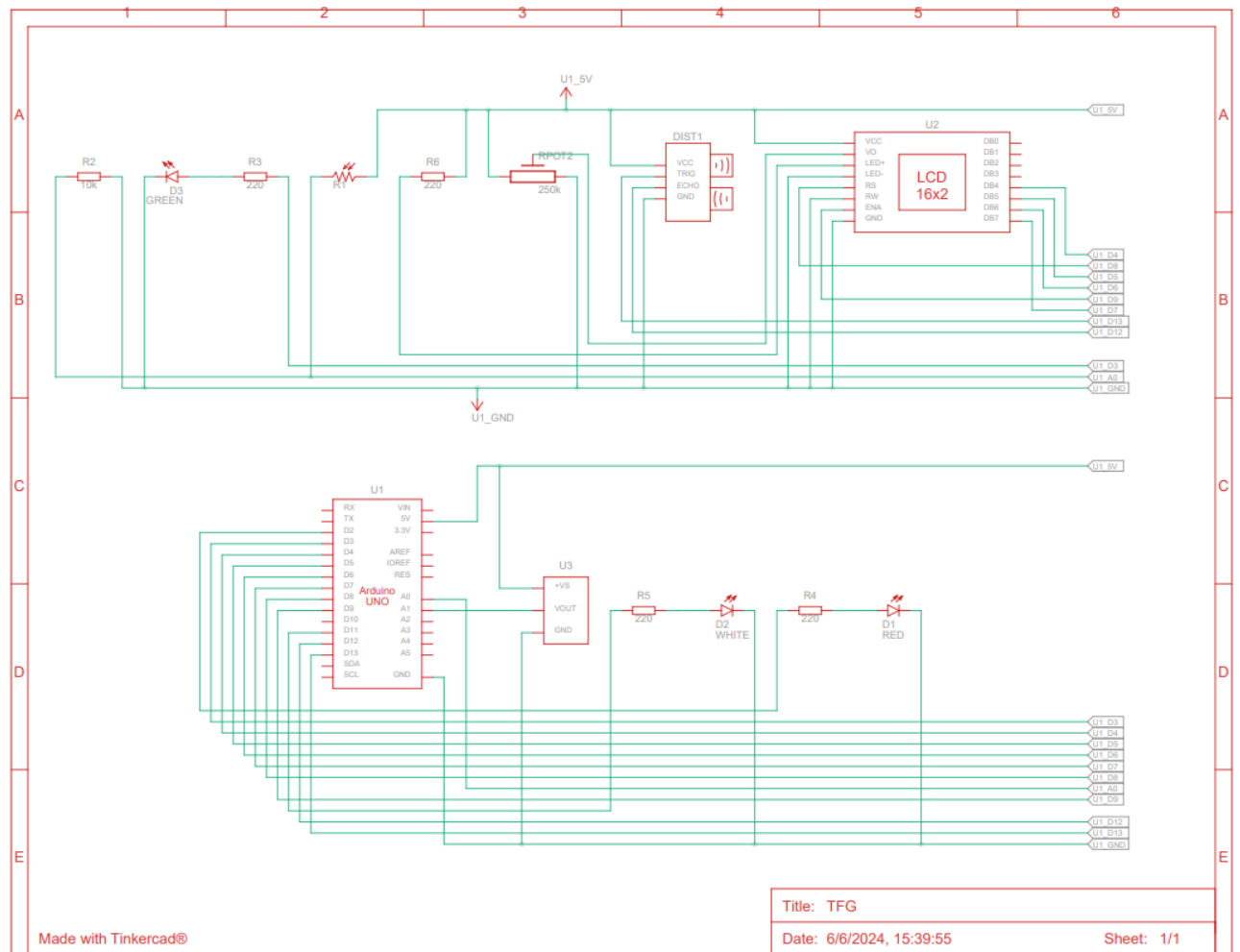
Componentes

Sensores

Actuadores

Nombre	Número
Arduino Uno R3	1
LCD 16 x 2	1
Fotorresistencia	1
Blanco LED	1
Verde LED	1
Sensor de distancia ultrasónico (4 pines)	1
10 kΩ Resistencia	1
220 Ω Resistencia	4
250 kΩ Potenciómetro	1
Rojo LED	1
Sensor de temperatura [TMP36]	1

Esquema eléctrico



Explicación del funcionamiento del sistema Arduino

El sistema es capaz de detectar si una persona se encuentra a una distancia de 2 cm (la distancia se puede configurar según se necesite), es decir, cuando una persona ocupa un asiento. En cuanto el sistema lo detecta, el led verde que estaba encendido, porque no detectaba nada a 2 cm de distancia, se apaga y se enciende el led de color rojo.

Otra funcionalidad del sistema es que es capaz de medir la temperatura, aunque esto es a nivel de sala y no por cada asiento. El sistema cuenta con una pantalla LCD que muestra esta temperatura . Para configurar el nivel de brillo de la pantalla se utiliza un potenciómetro.

También es capaz de medir la luminosidad mediante la Fotorresistencia. En cuanto se detecta que la luminosidad no es lo suficientemente alta, se ilumina un led de color blanco de forma progresiva y dependiendo del nivel de luminosidad.

Código y explicación

Configuración del hardware

- Leds: Se definen los pines para tres LEDs (rojo, verde y blanco) y se configuran como salidas.
- Sensor de Distancia Ultrasónica (SDU): Se configuran los pines para el trigger y el echo del sensor, y se inicializan variables para manejar el tiempo.
- Sensor de Fotorresistencia (LDR): Se configura el pin analógico para la entrada del sensor.
- Pantalla LCD: Se configuran los pines y se inicializa la pantalla LCD.
- Sensor de Temperatura (TMP): Se configura el pin analógico para la entrada del sensor de temperatura.
- Se define los parámetros para la conexión con influxDB

```
#include <LiquidCrystal.h>
#include <InfluxDbClient.h> // Agrega la biblioteca para la conexión a InfluxDB

// Declaración de objetos y variables globales

// Leds
const int redLedPin = 2;
const int greenLedPin = 3;
const int whiteLedPin = 11;

// SDU
const int echoPin = 12;
const int triggerPin = 13;

// LDR
const int ldrPin = A0;

// LCD
const int rs = 8, en = 9, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
String temperatureString = "";

// TMP
const int pinTmp = A1;
float tmp = 0;

// InfluxDB Configuración
#define WIFI_SSID "vodafoneACC8"
#define WIFI_PASSWORD "..."
#define INFLUXDB_URL "http://localhost:8086"
#define INFLUXDB_TOKEN "..."
#define INFLUXDB_ORG "My_Org"
#define INFLUXDB_BUCKET "Biblioteca"
#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"

WiFiMulti wifiMulti;
InfluxDbClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);
Point dataPoint("sensores");

... ..
```

Función setup()

- Inicializa los pines de los LEDs como salidas.
- Configura los pines del SDU.
- Configura el pin del LDR.
- Inicializa la pantalla LCD con 16 columnas y 2 filas.
- Se inicia la conexión con influxdb

```
void setup() {  
  // Inicialización de periféricos y conexiones  
  pinMode(redLedPin, OUTPUT);  
  pinMode(greenLedPin, OUTPUT);  
  pinMode(whiteLedPin, OUTPUT);  
  pinMode(echoPin, INPUT);  
  pinMode(triggerPin, OUTPUT);  
  pinMode(ldrPin, INPUT);  
  lcd.begin(16, 2);  
  
  // Configuración de conexión WiFi  
  WiFi.mode(WIFI_STA);  
  wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);  
  
  // Conexión a InfluxDB  
  timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");  
  if (client.validateConnection()) {  
    Serial.print("Conectado a InfluxDB: ");  
    Serial.println(client.getServerUrl());  
  } else {  
    Serial.print("Fallo en la conexión a InfluxDB: ");  
    Serial.println(client.getLastErrorMessage());  
  }  
}
```

Función loop()

- Medición de Distancia:
 - Genera un pulso en el trigger del SDU.
 - Mide la distancia usando la función `measureDistance()`.
 - Enciende el LED rojo si la distancia es menor a 200 cm; de lo contrario, enciende el LED verde.
- Control de Intensidad del LED Blanco:
- - Lee el nivel de luz ambiente con el LDR.
 - Mapea el valor leído a un rango de brillo (0-255).
 - Ajusta la intensidad del LED blanco con `analogWrite()`.
- Medición y Visualización de la Temperatura:
 - Mide la temperatura usando la función `calculateTemperature()`.
 - Limpia la pantalla LCD y muestra la temperatura en grados Celsius
- Envío de información a InfluxDB

```
void loop() {
  // Medidas de sensores
  triggerPulse();
  int distance = measureDistance();
  int lightLevel = analogRead(ldrPin);
  int brightness = map(lightLevel, 0, 1023, 255, 0);
  tmp = calculateTemperature();

  // Actualización de LEDs y LCD
  digitalWrite(redLedPin, distance < 200);
  digitalWrite(greenLedPin, distance >= 200);
  analogWrite(whiteLedPin, brightness);
  lcd.clear();
  lcd.setCursor(0, 0);
  temperatureString = String(tmp, 1);
  lcd.print("Tmp: ");
  lcd.print(temperatureString);
  lcd.print(" C");

  // Preparación de datos para InfluxDB
  dataPoint.clearFields();
  dataPoint.addField("ocupado", distance < 200);
  dataPoint.addField("brillo", brightness);
  dataPoint.addField("temp", tmp);

  // Envío de datos a InfluxDB
  if (wifiMulti.run() == WL_CONNECTED) {
    if (!client.writePoint(dataPoint)) {
      Serial.print("Error al escribir en InfluxDB: ");
      Serial.println(client.getLastErrorMessage());
    }
  } else {
    Serial.println("Conexión WiFi perdida");
  }

  delay(10000); // Esperar 10 segundos antes de la siguiente lectura
}
```



Funciones Auxiliares

- triggerPulse(): Genera un pulso de 10 microsegundos en el pin del trigger del SDU.
- measureDistance(): Mide la duración del eco y convierte esta duración a distancia en centímetros.
- calculateTemperature(): Lee el valor analógico del sensor de temperatura, lo convierte a voltaje y luego a grados Celsius.

```
void triggerPulse() {
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
}

int measureDistance() {
    triggerPulse(); // Generar pulso
    long duration = pulseIn(echoPin, HIGH); // Medir la duración del eco
    return duration * 0.034 / 2; // Convertir la duración a distancia en cm
}

float calculateTemperature() {
    float temperature = analogRead(pinTmp) * 5.0 / 1023.0; // Convertir lectura en voltaje
    //float resistance = (5.0 * 10000.0) / voltage - 10000.0; // Calcular resistencia del termistor
    temperature -= 0.5; // "Offset voltage", viene en el datasheet, solo aplica para el TMP36
    temperature = temperature * 100; // Convertir a celsius
    return temperature;
}
```

Conexión con InfluxDB

Para ello es necesario incluir la siguiente librería:

```
#include <InfluxDbClient.h>
```

Funcionamiento de la aplicación Angular

Estructura

Se comentó en anteriores apartados: [Estructura Aplicación Angular](#)

Librerías Externas

Gráficos

Se ha utilizado la siguiente librería para mostrar gráficos: **igniteui-angular-charts**.

Para instalarla se ejecuta el siguiente comando: **pnpm install igniteui-angular-charts**.

En el apartado [Módulo](#) se puede apreciar cómo se pueden importar los módulos que sean necesarios de esta librería.

[Enlace al sitio web oficial de la librería.](#)

Estilos

También se ha utilizado bootstrap 5 para los estilos de la aplicación.

El modo de instalación es el siguiente:

1. Se instala en el proyecto con el siguiente comando: **pnpm @ng-bootstrap/ng-bootstrap@next y pnpm i bootstrap bootstrap-icons**
2. Luego se modifica el archivo **angular.json** y se añaden las siguientes líneas:

```
"styles": [  
  "node_modules/bootstrap/scss/bootstrap.scss",  
  "node_modules/bootstrap-icons/font/bootstrap-icons.css",  
  "src/styles.css"  
],  
"scripts": [  
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"  
]
```

[Enlace al sitio web oficial de bootstrap.](#)

Módulo

Para este proyecto en Angular se ha requerido de los siguientes módulos, que se ven reflejados en el archivo app.module.ts. Este archivo administra por defecto los módulos globales de la aplicación Angular.

Resaltar por ejemplo la librería que se ha utilizado para ilustrar datos en gráficos, el cómo se importan los módulos de dicha librería en este archivo:

```
import {
  IgxLegendModule,
  IgxDataChartCoreModule,
  IgxDataChartCategoryCoreModule,
  IgxDataChartCategoryModule,
  IgxDataChartInteractivityModule,
  IgxDataChartVerticalCategoryModule,
  IgxDataChartAnnotationModule,
  IgxCategoryChartModule,
} from 'igniteui-angular-charts';
```

Por una parte se hacen los importes a los módulos que se quieren utilizar en el proyecto.

Y luego se importan dichos módulos para que cualquier componente, servicio o interface pueda usarlo.

Aparte de esta librería externa externa se puede ver como cada componente de la aplicación se declara en este fichero (es posible que algunos componentes falten o sobren en esta imagen puesto que está tomada durante el desarrollo de la aplicación).

```
You, 16 seconds ago | 1 author (You)
@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    MainComponent,
    LoginComponent,
    StatsComponent,
    ChartTempComponent,
    ChartBrightnessComponent,
  ],
  imports: [
    BrowserModule,
    NgbModule,
    AppRoutingModule,
    FormsModule,
    IgxLegendModule,
    IgxDataChartCoreModule,
    IgxDataChartCategoryCoreModule,
    IgxDataChartCategoryModule,
    IgxDataChartInteractivityModule,
    IgxDataChartVerticalCategoryModule,
    IgxDataChartAnnotationModule,
    IgxCategoryChartModule,
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```



Conexión con InfluxDB

Para ello se crea un servicio como el siguiente (dependiendo de lo que necesites):

```
export class InfluxdbService {
  private url = 'https://localhost:8086'; // Reemplaza con tu url
  private token = '2ETqNKKYdv_RGgBvj_tUQnR4rdL4QJGiv7JCg4SR5GmiPcScXdSu-NEAEMqvSjYwYZBjyiNHmjby8BL-_f-uDg=='; // Reemplaza con tu token
  private org = 'My_Org'; // Reemplaza con tu organización
  private bucket = 'bitcoin'; // Reemplaza con tu bucket

  private client: InfluxDB;

  constructor() {
    this.client = new InfluxDB({ url: this.url, token: this.token });
  }

  async getData(): Promise<any> {
    const queryApi = this.client.getQueryApi(this.org);
    const query = `
      from(bucket: "${this.bucket}")
      |> range(start: -12h, stop: now())
      |> filter(fn: (r) => r["_measurement"] == "coindesk")
      |> filter(fn: (r) => r["_field"] == "price")
      |> filter(fn: (r) => r["code"] == "USD")
      |> filter(fn: (r) => r["crypto"] == "bitcoin")
      |> filter(fn: (r) => r["description"] == "United States Dollar")
      |> filter(fn: (r) => r["symbol"] == "$")
    `;

    return new Promise((resolve, reject) => {
      const scatterData: { x: number; y: any; }[] = [];
      const lineBarData: { x: any; y: any; }[] = [];

      queryApi.queryRows(query, {
        next(row, tableMeta) {
          const o = tableMeta.toObject(row);
          scatterData.push({ x: new Date(o['_time']).getHours(), y: o['_value'] });
          lineBarData.push({ x: o['_time'], y: o['_value'] });
        },
        error(error) {
          console.error(error);
          reject(error);
        },
        complete() {
          resolve({ scatterData, lineBarData });
        }
      });
    });
  }
}
```

Luego este servicio se inyecta donde sea necesario.