

Laboratorio 6 - Blue Prints parte 3

Andrés Felipe Arias Ajiaco

Cesar David Amaya Gómez

Johan Sebastián Gracia Martínez

Sebastián David Blanco Rodríguez

Universidad Escuela Colombiana de ingeniería Julio Gravito

Arquitecturas de software

Ing. Javier Iván Toquica Barrera

16 de marzo de 2024

Introducción

La evolución tecnológica ha transformado la manera en que desarrollamos aplicaciones web, llevándonos a adoptar arquitecturas más eficientes y escalables. En este contexto, Angular ha emergido como una de las herramientas más populares para la creación de proyectos frontend, gracias a su robustez y facilidad de uso. Sin embargo, el verdadero potencial de una aplicación radica en su capacidad para interactuar con servicios externos, lo que nos lleva a considerar las API REST como elementos fundamentales.

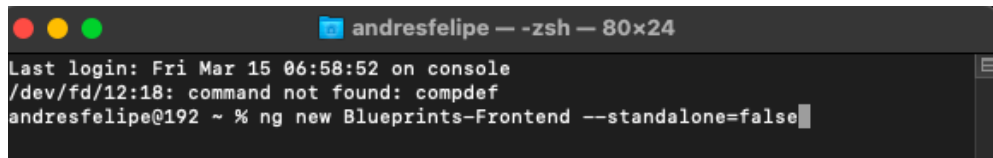
En este laboratorio, abordaremos el proceso de creación de un proyecto frontend en Angular diseñado específicamente para la gestión de planos arquitectónicos. El objetivo es desarrollar una aplicación que permita visualizar y consultar los diversos planos creados por un autor en particular.

Para potenciar la funcionalidad de la aplicación, estableceremos conexión con un backend que proporciona una API REST. Esta API nos permitirá almacenar, recuperar y actualizar los planos arquitectónicos de manera segura y eficiente. Exploraremos cómo configurar Angular para interactuar con esta API, gestionar las solicitudes HTTP y presentar los datos obtenidos de manera coherente y atractiva en nuestra interfaz de usuario.

A lo largo de este laboratorio, descubriremos cómo integrar Angular con las capacidades de una API REST, proporcionando una experiencia fluida y eficiente para la gestión de planos arquitectónicos.

Desarrollo del Laboratorio

1. Creamos el proyecto en angular con el comando mostrado, usamos `--standalone=false`, con el fin de que se cree con el archivo `module.ts`, para poder agregar los componentes que se creen

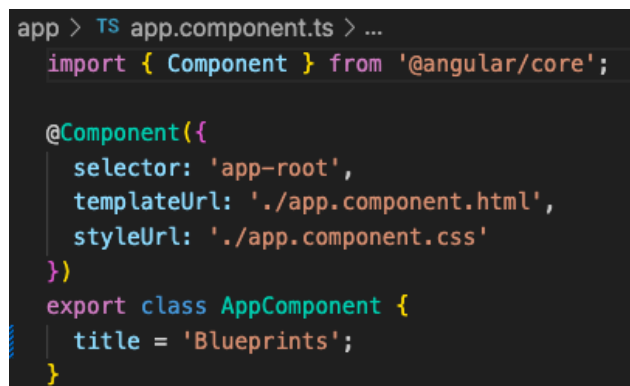


```
andresfelipe — zsh — 80x24
Last login: Fri Mar 15 06:58:52 on console
/dev/fd/12:18: command not found: compdef
andresfelipe@192 ~ % ng new Blueprints-Frontend --standalone=false
```

2. Después de crear el proyecto modificamos el HTML en el archivo `app.component.html` el cual es el que se mostrara en <http://localhost:4200/>, por donde corre angular, también modificamos el archivo `app.component.ts` en el cual se encuentra el título del proyecto y le ponemos “Blueprints”.



```
app > <> app.component.html >
<h1>{{title}}</h1>
```



```
app > TS app.component.ts > ...
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Blueprints';
}
```

3. Entramos a la terminal en la raíz donde se encuentra el proyecto frontend y ejecutamos el comando `ng serve` para compilar el proyecto Angular y ejecutar un servidor de desarrollo local, entramos a <http://localhost:4200/> para verificar la modificación del HTML.



Blueprints

4. Posterior a esto vamos a la terminal y ejecutamos el comando `ng g class blueprint` para crear una clase llamada blueprint en la cual se almacenarán los atributos jSon que obtendremos de las peticiones HTTP.

```
TS blueprint.service.spec.ts  U
TS blueprint.service.ts      U
TS blueprint.spec.ts         U
TS blueprint.ts              U
```

- Al crear la clase obtendremos estos 4 archivos los cuales:

blueprint.service.spec.ts: Este archivo contiene las especificaciones de la unidad para el servicio Blueprint. Se utiliza para probar el servicio Blueprint para asegurarse de que funciona correctamente.

blueprint.service.ts: Este archivo contiene la implementación del servicio Blueprint. Define las funciones y propiedades que se utilizan para crear e interactuar con Blueprints.

blueprint.spec.ts: Este archivo contiene las especificaciones de la unidad para el módulo Blueprint. Se utiliza para probar el módulo Blueprint para asegurarse de que se carga correctamente y que exporta las funciones y propiedades correctas.

blueprint.ts: Este archivo contiene la implementación del módulo Blueprint. Define las funciones y propiedades que se utilizan para crear e interactuar con Blueprints.

- En el archivo blueprint.ts definiremos los atributos que vamos a almacenar:

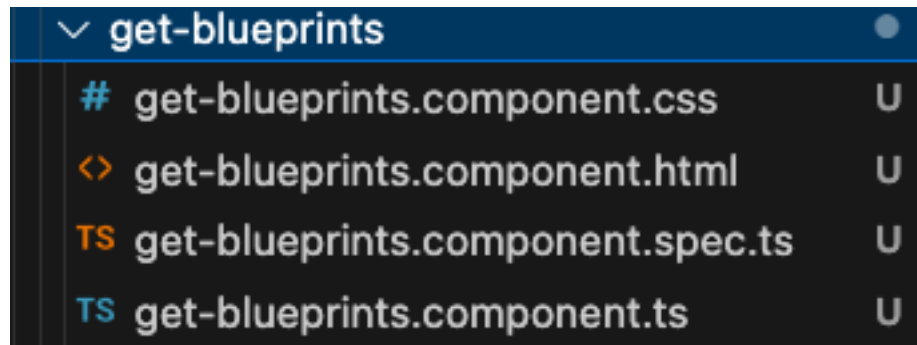
```
app > TS blueprint.ts > ...
export class Blueprint {

  name : string;
  author : string;
  amountOfPoints : number;
  points: { x: number, y: number }[];

  constructor() {
    this.name = '';
    this.author = '';
    this.amountOfPoints = 0;
    this.points = [];
  }
}
```

5. Procederemos a crear el componente mediante el cual podremos obtener los planos del autor requerido. para esto usaremos el comando `ng g c get-blueprints` de esta manera Angular generará automáticamente los archivos necesarios para el componente, incluyendo un archivo TypeScript para la lógica del componente, un archivo HTML para la plantilla del componente, un archivo de estilos CSS (o SCSS,

si está configurado así), así como también actualizará automáticamente el archivo de módulo de Angular para que el componente sea reconocido y pueda ser utilizado en la aplicación.



- Lo primero que realizaremos es verificar que el componente se haya agregado correctamente al archivo app.module.ts, con el fin de asegurarnos de que esté correctamente registrado en el módulo principal de la aplicación. Esto nos permitirá utilizar el componente en nuestra aplicación Angular y garantizar su correcto funcionamiento dentro del contexto del proyecto.

```
@NgModule({
  declarations: [
    AppComponent,
    GetBlueprintsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Observamos que el componente se encuentra ya en el archivo, específicamente en la parte de declarations.

- Posterior a esto modificaremos el HTML del componente y en el archivo app.component.html agregaremos `<router-outlet></router-outlet>` el cual mostrara diferentes partes de la página según la dirección web en la que te encuentres. Por ejemplo, cuando vayas a la página de inicio, se mostrará un componente específico, y cuando vayas a otra sección, se mostrará otro componente diferente, teniendo en cuenta que se debe modificar la ruta de acceso según el componente esto lo haremos en app-routing.module.ts.

```
app > <> app.component.html > router-outlet
<h1>{{title}}</h1>
<router-outlet></router-outlet>
```

```
app > TS app-routing.module.ts > ...
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { GetBlueprintsComponent } from './get-blueprints/get-blueprints.component';

const routes: Routes = [
  {path: 'index.html', component: GetBlueprintsComponent},
  {path: '', redirectTo: 'index.html', pathMatch: 'full'}
];
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Blueprints</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <form (ngSubmit)="onSubmit()">
      <label for="author">Author:</label>
      <input type="text" id="author" [(ngModel)]="author" name="author">
      <input type="submit" value="Get Blueprints">
    </form>

    <div *ngIf="flagDiv">
      <h2>{{ name }}</h2>
    </div>

    <div style="margin-top: 25px" *ngIf="flagDiv">
      <table class="container" *ngIf="blueprints && blueprints.length > 0">
        <thead>
          <tr>
            <th>Name</th>
            <th>Amount of Points</th>
            <th>Open</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let blueprint of blueprints">
            <td>{{blueprint.name}}</td>
            <td>{{blueprint.amountOfPoints}}</td>
            <td>
              <button id="button" (click)="openBlueprint(blueprint.name)">Open</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
```

6. Posterior a esto realizaremos los diferentes servicios con los cuales podremos hacer las peticiones HTTP al servidor y obtener los datos jSon que necesitamos mostrar en el frontend.

- Modificaremos el archivo blueprint.service.ts y crearemos dos métodos en los cuales se obtendrá todos los planos del autor dado y un plano en específico según su nombre:

```
export class BlueprintService {
  //Base URL
  private URL = "http://localhost:8080/blueprints";

  constructor(private httpClient: HttpClient) {}

  //Service to Get all the blueprints by author
  getBlueprintsByAuthor(author: string): Observable<Blueprint[]> {
    return this.httpClient.get<Blueprint[]>(`${this.URL}/${author}`).pipe(
      catchError(error => {
        console.log('Error al obtener los blueprints del autor:', error);
        return throwError(error);
      })
    );
  }

  //Service to Get a blueprint with the name of this and author's name
  getBlueprintsByNameAndAuthor(author: string, nameBlueprint: string): Observable<Blueprint> {
    return this.httpClient.get<Blueprint>(`${this.URL}/${author}/${nameBlueprint}`).pipe(
      catchError(error => {
        console.log('Error al obtener el blueprint del autor:', error);
        return throwError(error);
      })
    );
  }
}
```

7. Modificaremos el archivo get-blueprints.component.ts, el cual es el archivo de TypeScript asociado al componente get-blueprints. En este archivo, realizaremos cambios en la lógica del componente, como la definición de propiedades, métodos y la interacción con servicios o la lógica de negocio de la aplicación. Además, este se relaciona directamente con su propio HTML, que sería get-blueprints.component.html

- Definiremos las variables necesarias en las que almacenaremos los diferentes datos para mostrarlos adecuadamente, como: la lista de blueprints del autor, su nombre, cantidad de puntos, nombre del plano específico y el plano en particular, etc.

```
export class GetBlueprintsComponent {  
  
  blueprints: Blueprint[];  
  flagDiv: Boolean = false;  
  author: string;  
  amountOfPoints: number;  
  name: string;  
  blueprintName: string;  
  blueprint: Blueprint;  
  
  constructor(private blueprintServices: BlueprintService) {  
  
  }  
}
```

- Teniendo los servicios creados y las variables en las que almacenaremos los datos realizaremos los métodos del componente que hará uso de estos servicios

```
//get all blueprint's with the name author's  
private getBlueprintsByAuthor() {  
  try {  
    this.blueprintServices.getBlueprintsByAuthor(this.author).subscribe(data => {  
      this.blueprints = data;  
      this.setName();  
      this.setAmountOfPoints();  
      this.flagDiv = true;  
    });  
  } catch (error) {  
    console.log('Error en getBlueprintsByAuthor:', error);  
  }  
}  
  
//Get the blueprint with the blueprint's name and author's name  
private getBlueprintByNameAndAuthor() {  
  try {  
    this.blueprintServices.getBlueprintsByNameAndAuthor(this.author, this.blueprintName).subscribe(data => {  
      this.blueprint = data;  
      this.drawBlueprint(this.blueprint);  
    });  
  } catch (error) {  
    console.log('Error en getBlueprintByNameAndAuthor:', error);  
  }  
}
```

- Realizamos métodos para definir el valor correcto de las diferentes etiquetas del HTML del componente:

```
//Set author name
private setName(){
  this.name = this.author + "'s" + " " + "blueprints";
}

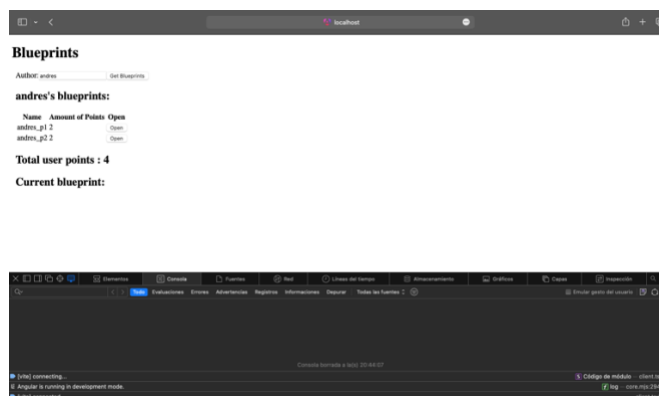
//Set amount of total Amount of points
private setAmountOfPoints(){
  this.amountOfPoints = this.blueprints.reduce((total, blueprint) => total + blueprint.amountOfPoints, 0);
}
```

- Construimos los métodos que se ejecutarán al enviar el formulario y oprimir el botón, los cuales están configurados para ejecutar los métodos que utilizan los servicios, que a su vez realizan las peticiones al servidor.

```
//Method to send the petition for getting blueprints
onSubmit(){
  if(this.author !== undefined && this.author !== ""){
    this.getBlueprintsByAuthor();
  }
}

//Method to open the blueprint
openBlueprint(name: string) {
  this.blueprintName = name;
  this.getBlueprintByNameandAuthor();
}
```

- Verificamos su funcionamiento ejecutando el comando `ng serve` en la raíz del proyecto frontend y `mvn spring-boot:run` en el backend.



Observamos su correcto funcionamiento, teniendo en cuenta que en la consola no hubo errores.

- Procedemos a realizar el método que dibuja el canvas del blueprint especifico al oprimir el botón Open, el cual tiene un id de identificación dentro del HTML.

```
private drawBlueprint(blueprint: Blueprint | null | undefined) {
  if (!blueprint) {
    console.error('El plano es nulo o indefinido');
    return;
  }

  const canvas = document.getElementById('Canvas') as HTMLCanvasElement;
  if (!canvas) {
    console.error('No se pudo encontrar el elemento canvas');
    return;
  }

  const context = canvas.getContext('2d');
  if (!context) {
    console.error('No se pudo obtener el contexto 2D del canvas');
    return;
  }

  //Clean the canvas before to draw
  context.clearRect(0, 0, canvas.width, canvas.height);

  //Fix the style of the lines
  context.strokeStyle = 'red';
  context.lineWidth = 2;

  //Draw the line
  for (let i = 0; i < blueprint.points.length - 1; i++) {
    const startPoint = blueprint.points[i];
    const endPoint = blueprint.points[i + 1];
    context.beginPath();
    context.moveTo(startPoint.x, startPoint.y);
    context.lineTo(endPoint.x, endPoint.y);
    context.stroke();
  }
}
```

- Posterior a esto agregamos estilos css para mejorar la visibilidad del frontend y probamos la aplicación.

```
form {
  display: flex;
  flex-wrap: nowrap;
  margin-left: 15px;
  margin-top: 30px;
  padding: 15px;
  border: 1px solid #ccc;
  border-radius: 5px;
  width: 40%;
}

label {
  display: inline-block;
  margin-left: 20px;
  margin-bottom: 5px;
  font-style: italic;
  font-size: 25px;
  flex-basis: 10%;
}

input[type="text"] {
  display: inline-block;
  margin-bottom: 5px;
  margin-right: auto;
  margin-left: 50px;
  width: auto;
  border: 1px solid #ccc;
  border-radius: 5px;
  box-shadow: 0px 0px 5px 0px rgba(0,0,0,0.1);
  flex-basis: 20%;
}

input[type="submit"] {
  display: inline-block;
```

Author:

andres

Get Blueprints

andres's blueprints:

Name	Amount of Points	Open
andres_p1	2	<button>Open</button>
andres_p2	2	<button>Open</button>

Total user points : 4

Al dar click en el botón open se actualiza el plano actual y se pinta el segmento dentro del canva.



Conclusiones

El uso de Angular junto con peticiones a un API REST implica una comunicación dinámica entre la interfaz de usuario y el servidor. Esto permite que la aplicación web acceda y presente datos en tiempo real, lo que mejora la experiencia del usuario al proporcionar información actualizada de forma instantánea.

Angular ofrece facilidades para abstraer la lógica de red y las peticiones HTTP a través de su módulo HttpClient. Esto permite que los desarrolladores trabajen con peticiones HTTP de manera más sencilla y estructurada, encapsulando la complejidad de la comunicación con el servidor y facilitando el mantenimiento del código.

Bibliografía

- <http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/>
- <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
- https://www.youtube.com/watch?v=o_HV_FCs-Z0&t=1952s