UNIVERSITY OF TRENTO

ASSIGNMENT 4

Vinci Nicolò 220229

nicolo.vinci@studenti.unitn.it

Web Architectures 2021/2022

28 November 2021

Contents

1 Introduction

It needs to develop a Single Page Application (SPA) with Angular that shows the Scottish Parliament members. The SPA should have an header and a footer bar with the Scottish Parliament logo and it shows the members list. The members list is available at *https://data.parliament.scot/api/members*. For each member, his photo and name has to be shown embedded in a card. After that, each card can be clicked in order to change the status of the SPA showing other member details such as:

- The birthdate.
- Parties to which the parliamentary belongs. They are avaialable at *https://data.parliament.scot/api/memberparties* and *https://data.parliament.scot/api/parties*.
- Her/his websites.

The entire SPA is developed in the Angular framework exploiting components, services and routing. At the end, a war file of the SPA is created and deployed in Apache Tomcat web server.

2 Description

In the *src* folder you can find the application itself:

- app folder: where the app is developed.
- assets folder: inside there is an *images* folder which contains the static images used for the application.
- environment folder: to decide whether the application should be deployed in production mode or not.
- index.html: where the application is booted.
- main.ts: where the application is booted.
- polyfills.ts: it contains the code needed to run the application among all possible browsers.
- styles.css: it contains global styles for the application.

The *app* folder contains:

- card-member folder: it contains the component to show the member details when a card is clicked from the member lists.
- error folder: it contains the component to show an error when a user is trying to reach a non-existent state of the app via the URL.
- footer folder: it contains the component to show the footer bar.
- header folder: it contains the component to show the header bar.
- list-members folder: it contains the component to show the member lists with cards. Any card can be clicked in order to see the corresponding *card-member* component.
- models folder: it contains the interfaces needed to map any non-typed JSON obtained from an http get request to a strongly typed response.
- pipe folder: it contains a custom pipe called *customDate* to convert a string in a formatted date.
- services folder: it contains the *ParliamentService* to perform the http get request to retrieve the member lists, a single member, the member parties, a party name and any website associated to a member. Then, there is also a *CacheService* where any http get response is stored after the first request.
- app.components.css: it contains the global style for the app component.
- app.components.html: where all the other components will be shown.

- app.components.ts: it contains the definition of the app component.
- app.modules.ts: it contains the definition of the app module. There are the components declaration, the imports, the providers and the bootstrap component.
- app.routes.ts: it maps the application states to the corresponding URL(s).

3 App

The application functioning is described in this section.

3.1 Deployment

The application needs to be deployed in the Apache Tomcat web server. First of all, the Angular application is compiled with the command `ng build` launched inside the Angular project folder. It compiles the entire Angular app into the output directory *dist/angularProject*, where *angularProject* is the name for this specific Angular application.

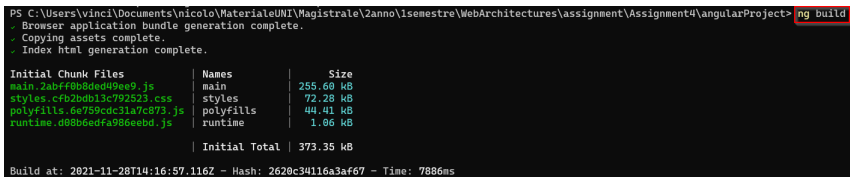

Figure 1: ng build command

The base href has to be changed before creating the jar file. Go into */dist/angularProject/index.html* and change the base href from `"/"` to `"."`.

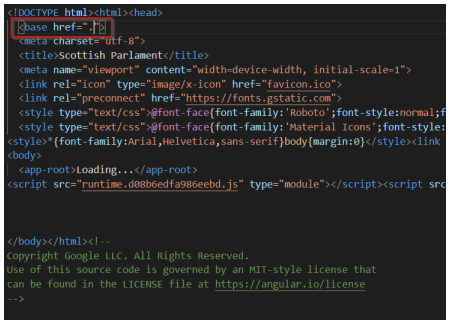

Figure 2: Changing href

Then, the command `jar cvf angularProject.war .` should be launched inside the folder *dist/angularProject*. It produces a file war called *angularProject.war* to be deployed in Tomcat.

Figure 3: jar command

The war file can be copied inside the *webapps* folder of Tomcat and it will be automatically unpacked in a *angularProject* folder.

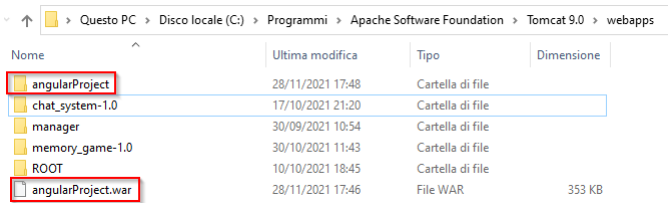

Figure 4: Webapps Tomcat folder

Now, the Angular project can be seen in the Tomcat web application manager at *http://localhost:8080/manager/html* and it can be started clicking on the link.

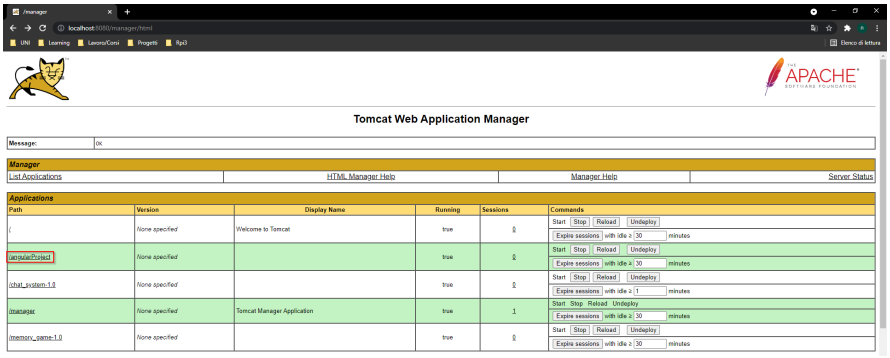

Figure 5: Tomcat web application manager

3.2 Services

The services are the core of the application, because they retrieve the data that will be displayed by components. Services are located in the *services* folder. There are two different services:

- CacheService: it is injected in *app.module.ts* in order to be injected before the *ParliamentService*. It is used to store any response obtained by the *ParliamentService*. When the *ParliamentService* performs an http get request for the first time, it stores the response in the *CacheService*. So, the next time the *ParliamentService* will retrieve the response directly from the *CacheService* without performing the http get request. In this way, the application is faster. Another method to achieve the caching is to exploit the rxjs stream with the operator `shareReplay(1)` which emits the last stream value to any new subscriber. An example is commented in the *ParliamentService*:

```
/*
//Cache with shareReplay
public getMembers(): Observable<IMember[]> {
  return this.http.get<IMember[]>(this.urlGetMembers)
    .pipe(
      map((response) => {
        let members:IMember[]=[];
        response.forEach(element => {
          members.push(this.adjustMember(element));
        })
        return members;
      }),
      catchError((error) => {
        console.error(error);
        throw error;
      }),
    shareReplay(1)
  );
}
*/
```

Figure 6: Caching with shareReplay

- ParliamentService: it is injected in *ListMembersComponent* and *CardMemberComponent*. Instead, in the *ParliamentService* the observable HttpClient and the *CacheService* are injected in the constructor to perform http get requests and to cache http responses. There are six different methods:

  1. getMembers(): it returns an observable of type *IMember[]* which represents the members list. Firstly, it checks if the members list is available in the *CacheService*, otherwise it retrieves the members list subscribing to the HttpClient observable. If it so, it stores the members list in the *CacheService*.
  2. getMemberById(memberId): it returns an observable of type *IMember* considering his/her parliamentary id. Firstly, it checks if the member is present in the *CacheService* exploiting the entire members list, otherwise it retrieves the member subscribing to the HttpClient observable.
  3. getMemberPartiesById(memberId): it returns an observable of type *IMemberParties[]*. It retrieves the member parties list from his/her parliamentary id. Firstly, it checks if the member parties list is present in the *CacheService* exploiting all the members parties, otherwise it retrieves the member parties subscribing to the HttpClient observable. If it so, it stores all the member parties in the *CacheService*.
  4. getPartyNameById(partyId): it returns an observable of type *string*. It retrieves the party name from its party id. Firstly, it checks if the party name is present in the *CacheService* exploiting all the parties, otherwise it retrieves the party name subscribing to the HttpClient observable. If it so, it stores all the parties in the *CacheService*.

5. getWebsitesById(memberId): it returns an observable of type *IWebsite[]*. It retrieves the member websites list from his/her parliamentary id. Firstly, it checks if the websites list is present in the *CacheService* exploiting all the websites, otherwise it retrieves the websites list subscribing to the HttpClient observable. If it so, it stores all the websites in the *CacheService*.
6. adjustParlament(parlament): it is a private method and returns an *IParlament*. It takes as parameter an *IParlament* and it manipulates the *ParliamentaryName* and the *PhotoUrl*.

3.3 Models

Four interfaces have been developed in the *models* folder to map the non-typed JSON http response to a strongly typed response:

- IMember: it can be found in *member-interface.ts*. It defines a member.
- IMemberParties: it can be found in *member-parties-interface.ts*. It defines the member party type adding an optional attributed called *PartyName* that represents the party name retrieved from the party id.
- IParty: it can be found in *party-interface.ts*. It defines a party.
- IWebsites: it can be found in *website-interface.ts*. It defines a website.

3.4 Pipe

A custom pipe has been developed in order to display a string that represents a date in a custom format. It is called *CustomDatePipe* and it can be found in the *pipe* folder. The pipe takes two optional string parameters. If the `add` parameter is null, it returns only the date formatted as *MMMM d, y*. If both parameters are not null, it returns *add+" "+MMMM d,y*. This is useful to print the belonging period to a party. Indeed, the pipe can print *From May 5,2016* or *To May 4, 2021*.

```
@Pipe({name: 'customDate'})
export class CustomDatePipe implements PipeTransform {
  constructor(private datepipe:DatePipe){}

  transform(value?:string, add?:string): any {
    if(add==null) {
      return this.datepipe.transform(value, 'MMMM d, y');
    }
    if(value!=null) {
      return add+" "+this.datepipe.transform(value, 'MMMM d, y');
    } else {
      return "";
    }
  }
}
```

Figure 7: CustomDatePipe

3.5 Members list

When the link is clicked, the application will show the members list as reported in figure 8.

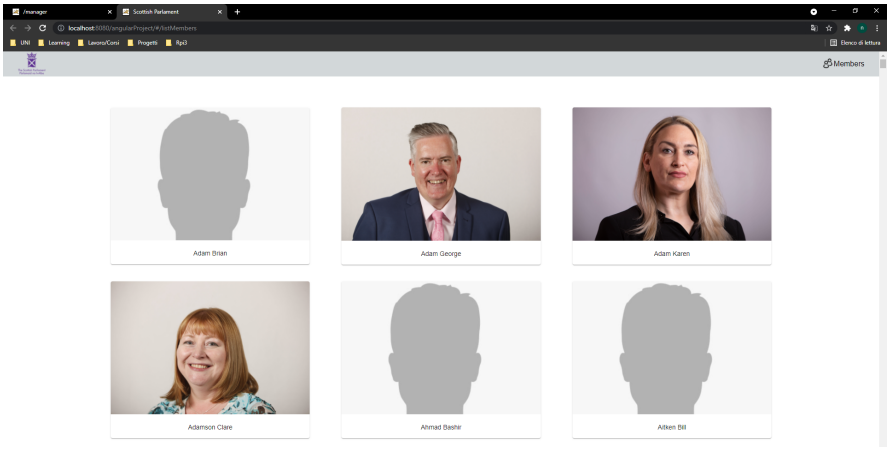

Figure 8: Members list

At the top of the page there is the header bar component. It is a simple bar with the Scottish Parliament logo on the left and a link called *Members* on the right. If the link is clicked, it reports the application in the state at the URL *http://localhost:8080/angularProject/#/listMembers*.

Figure 9: Header bar

At the bottom of the page there is the footer bar component. It is simpler than the header bar, because it has only the Scottish Parliament logo on the left and the copyright on the right.

Figure 10: Footer bar

Then, there is the *ListMembersComponent* that shows the members list. In *list-members.component.ts*, the component performs a subscription to the *ParliamentService* in the method ngOnInit() in order to retrieve the members list. The *ParliamentService* is injected in the constructor and the method `getParlaments()` is used to get all the members. Services are explained in the subsection 3.2. Then, the response is stored in a list called `_members` of type *IMember* and it is sorted by the field *ParliamentaryName*.

```
export class ListMembersComponent implements OnInit {

  private _members:IMember[];

  constructor(private parliamentService:ParliamentService, private router:Router) {
    this._members=[];
  }

  ngOnInit(): void {
    this.parliamentService.getMembers()
      .subscribe(
        {
          next: (members) => {
            this._members=members;
            this._members.sort((a:IMember,b:IMember)=>a.ParliamentaryName.localeCompare(b.ParliamentaryName));
          },
          error: (error) => {
            console.log(error);
          }
        }
    )


  }
```

Figure 11: Subscription to retrieve the members list

The private list `_members` can be reachable from the outside thanks to the get method defined in the *list-members.component.ts*.

```
public get members(): IMember[] {
  return this._members;
}
```

Figure 12: Getter for members list

Members are shown in a grid imported from Angular material. The grid is designed in order to shown three members per raw. Any member in the `members` list is shown in a *mat-card*, thanks to *ngFor* directive. Any *mat-card* is clickable and shows the member's photo and the member's name. Also, *mat-card* tag is imported from Angular material. The html code with the various Angular bindings can be found in *list-parlament.component.html*.

```
<div class="mat-custom">
  <mat-grid-list cols="3" rowHeight="4:3">
    <mat-grid-tile *ngFor="let member of members">
      <mat-card (click)="cardClicked(member.PersonID)">
        <img mat-card-image [src]="member.PhotoURL" alt="member-photo">
        <mat-card-content>
          {{member.ParliamentaryName}}
        </mat-card-content>
      </mat-card>
    </mat-grid-tile>
  </mat-grid-list>
</div>
```

Figure 13: Html and Angular bindings for members list

3.6 Member card

In *list-members.component.ts*, the router service is injected in the constructor and it is used to navigate to a the new state from the method *cardClicked(id)*.

When a card is clicked the application state changes calling the method *cardClicked(id)* and passing to it the corresponding parliamentary id. Then, the id is passed to the *CardMemberComponent* as query parameter in the URL.

```
public cardClicked(id:number) {
  this.router.navigate(['cardMember', id]);
}
```

Figure 14: cardClicked method

The method exploits the router service to navigate through the possible states. Routes are defined in the *app.routes.ts* file.

```
export const appRoutes: Routes = [
  { path: "listMembers", component: ListMembersComponent },
  { path: "cardMember/:id", component: CardMemberComponent },
  { path: "", redirectTo: "listMembers", pathMatch: "full" },
  { path: "**", component: ErrorComponent }
];
```

Figure 15: Routes

So, the member details appears when a member card from the member list is clicked.

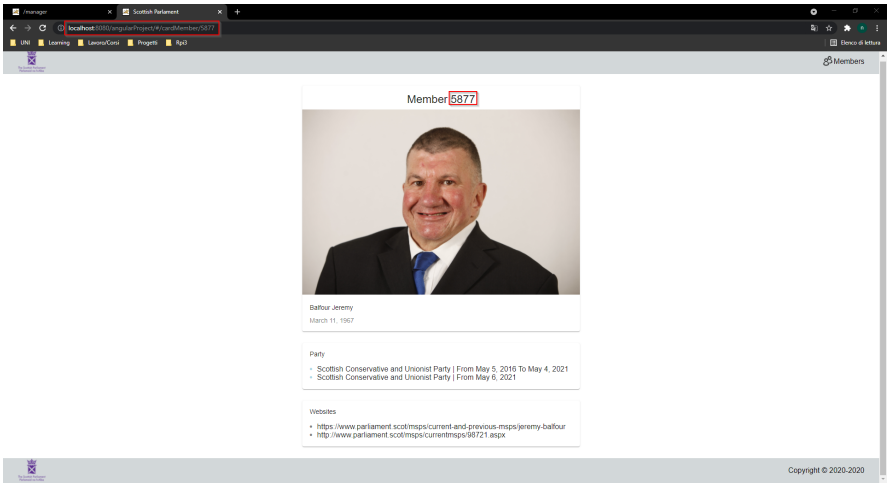

Figure 16: Member details of Bailfour Jeremy

Details are developed in three different *mat-card* in *card-member.component.html*:

- First *mat-card*: it shows the member id, photo, name and birthdate. The birthdate is manipulated by the custom pipe explained in the subsection 3.4.

- Second *mat-card*: it shows the member parties. The party name and the period are shown. The two dates of the period are manipulated by the custom pipe explained in the subsection 3.4.
- Third *mat-card*: it shows the member websites, if available.

Figure 17: Three mat-card tags for member details

There are different observable subscriptions in the ngOnInit() defined in *card-member.component.ts* to retrieve the data shown by html code with Angular bindings. First of all, the *ParliamentService* and the *ActivatedRoute* services are injected in the *CardMemberComponent* constructor. Then, the parliamentary id is obtained from the query id in the URL subscribing to the *ActivatedRoute* observable.

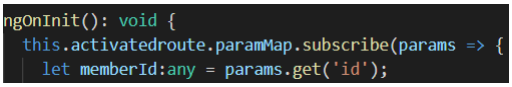

Figure 18: Retrieve parliamentary id from URL

A single member is retrieved subscribing to the method *getMemberById(memberId)* of the *ParliamentService*. The response is stored in a private optional attribute called `_member` of type *IMember*.

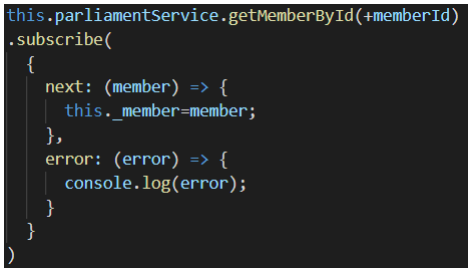

Figure 19: Retrieve a single member by parliamentary id

The member parties are retrieved by subscribing to the method *getMember-PartiesById(memberId)* of the *ParliamentService*. However, in each member party there is only the party id and not the party name. So, a party name is obtained by subscribing to the method *getPartyNameById(partyId)* of the

ParliamentService for each member party . Member parties with the corresponding party name are stored in the private list called `_memberParties` of type *IMemberParties[]*.

```
this.parliamentService.getMemberPartiesById(+memberId)
.subscribe(
  {
    next: (memberParties) => {
      this._memberParties=memberParties;
      this._memberParties.forEach(element => {
        this.parliamentService.getPartyNameById(element.PartyID)
          .subscribe(
            {
              next: (partyName) => {
                element.PartyName=partyName;
              },
              error: (error) => {
                console.log(error);
              }
            }
          )
      })
    },
    error: (error) => {
      console.log(error);
    }
  }
)
```

Figure 20: Retrieve a member parties by parliamentary id and parties name by party id

At the end, websites are retrieved subscribing to the *getWebsitesById(memberId)* method of the *ParliamentService*. Response is stored in the private list called `_websites` of type *IWebsite[]*.

```
this.parliamentService.getWebsitesById(+memberId)
.subscribe(
  {
    next: (websites) => {
      this._websites=websites;
    },
    error: (error) => {
      console.log(error);
    }
  }
)
```

Figure 21: Retrieve a websites by parliamentary id

Private attributes such as `_member`, `_memberParties` and `_websites` in the *CardMemberComponent* are exposed by getter methods.

```
public get member(): IMember | undefined {
  return this._member;
}

public get memberParties(): IMemberParties[] {
  return this._memberParties;
}

public get websites(): IWebsite[] {
  return this._websites;
}
```

Figure 22: Getter methods in CardMemberComponent

4 Useful links

Some links followed to develop the project are reported in this section.

- Angular material grid: `https://www.concretepage.com/angular-material/angular-material-grid-list`.
- Angular material grid: `https://www.educba.com/angular-material-responsive-grid/`.
- Angular routing, query parameters in URL: `https://www.tektutorialshub.com/angular/angular-passing-parameters-to-route`.
- Angular material card: `https://material.angular.io/components/card/overview`.
- Angular Date pipe: `https://angular.io/api/common/DatePipe`.
- Angular custom pipe: `https://www.ngdevelop.tech/angular/pipes/`.
- Rxjs shareReplay: `https://www.learnrxjs.io/learn-rxjs/operators/multicasting/sharereplay`.