Assignment 5 - Scottish Parliament

Web Architectures - 06/01/2023

Sabin Andone

Introduction

For this assignment it is necessary to create an Angular application that, given the list of members of the Scottish Parliament, lists the data for each parliament member, showing details such as the name of the parliament member, their picture, their birthdate, etc. The webapp shows two pages: the first one is the page that contains the list of all the members while the second one is the page that shows the details of a certain parliament member selected by the user. In order to move between the two pages it has been suggested to use routing.

The list of the parliament members is available at https://data.parliament.scot/api/members. The other data (the member details and their websites) are available at different links, these being the following:

- https://data.parliament.scot/api/memberparties (for member-party relation)
- https://data.parliament.scot/api/parties (for party details)
- https://data.parliament.scot/api/websites (for showing members' websites)

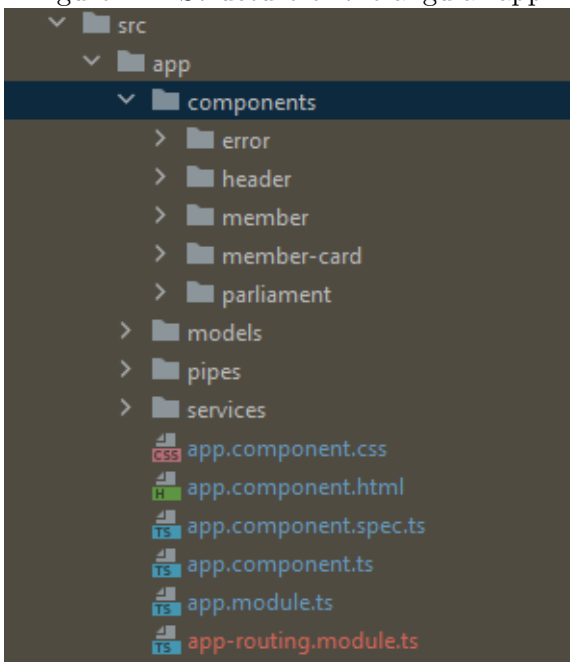
For this assignment it has been suggested to create a service to manage the data coming from these links.

Finally, it is necessary to deploy the developed angular application on Tomcat. It is possible to do so by first building the angular project, having the built artifacts in the dist/ directory and then copy the content of that directory inside a new web app project created with Intellij IDEA. Note that inside the index.html file it is necessary to change the base-ref variable from '/' to '.'.Then it is necessary to create the .war file by creating a "web application: archive" for the "web application:war exploded" artifact and build it.

Implementation

In order to implement the solution for this assignment, it has been necessary to consider different elements that compose the entire application. All these elements together compose the structure of the angular app. Such structure is shown in the following picture:

Figure 2.1: Structure of the angular app



As shown in Figure 2.1 the most relevant parts of this angular app are:

- the components, each one of them being responsible for displaying a certain component of the page;
- the models, i.e. the interfaces used for getting data for the service functions;
- the pipes, useful for changing the format of the input data in another format much more manageable
- the service, necessary for data caching;
- the routing module, included in the app component and necessary to apply routing to the angular app

The first part developed for this project have been the components of the angular app. This app has six components (app component included). The app component is the one responsible for rendering the whole application. The other components are responsible for displaying a part of the page. The html displays two of these components:

- the header;
- the routing output, this being either parliament, member or error.

The header component is the one responsible for rendering the header of the webapp. It contains a navbar with the logo of the Scottish Parliament located on its left and a button located on the right that triggers the loadMembers() function, which redirects the user to the parliament member list page.

The parliament component is the one resposible for displaying the list of all members of the Scottish Parliament. This component retrieves the list with all the members, sorting them alphabetically based upon their surname. This is possible by using the service functions in order to retrieve the data of all members and create a member-card component for each of them through a *ngFor directive. This member-card is a component that displays one Member object within a card that displays its name and its photo (or a placeholder photo, if no personal photo available).

Figure 2.2: Parliament Component class source code

```
export class ParliamentComponent {
  //list of members
  members?: Member[];

  constructor(private memberService: MemberService) { }

  ngOnInit(): void {
    //get the list of members
    this.memberService.getMembers().subscribe( options: (members) => {
      //sort them by surname
      members.sort(function(member1,member2) {
        return ((member1.ParliamentaryName > member2.ParliamentaryName) ? 1 : -1);
      });
      //assign to this.members the sorted list of the members
      this.members = members;
    });
  }
}
```

The member component is the one that displays the person details page of a member given its ID. Such ID is passed as a parameter inside a URL when the user clicks on the name of the member in the parliament member list: if the ID is wrong or no ID is found, then the user is redirected to the error page. If the ID is correct and corresponds to one member, then the MemberComponent retrieves all the details and displays a Member object that has a ParliamentaryName, the BirthDate (if present), the PhotoURL, the party association and the links to their websites.

Figure 2.3: Member Component class source code. The getMemberById, getPartyByMember and getWebsites-ByMember are all three functions that are used to get data to assign to member, party and websites attributes

```
export class MemberComponent implements OnInit{
  member!: Member;
  party!: Party;
  websites!: Website[];

  constructor(
    private memberService: MemberService,
    private route: ActivatedRoute,
    private router: Router
  ) { }

  ngOnInit(): void {
    //get the id from the parameters of the query
    this.route.queryParams.subscribe( options: params => {
      const id = params['id'];
      //if undefined, redirect the user to error page
      //otherwise get the details of the member
      if(id === undefined) {
        this.router.navigate(['/error-page']);
      } else {
        this.getMemberById(id);
        this.getPartyByMember(id);
        this.getWebsitesByMember(id);
      }
    });
```

The error is the component that renders the page in case of errors.

After we create all the components it is necessary to create also the models necessary to for the data caching part of the angular app. These models are defined as interfaces. For this assignment the following models have been created:

- Member, interface used for representing a single member;
- MemberParties, interface used for representing a relation between a member and a party;
- Member, interface used for representing a single party;
- Member, interface used for representing a single website;

Figure 2.4: Member model interface. The other model interfaces have a similar format.

```
export interface Member {
  PersonID: number;
  PhotoURL: string;
  Notes: string;
  BirthDate: string;
  BirthDateIsProtected: boolean;
  ParliamentaryName: string;
  PreferredName: string;
  GenderTypeID: number;
  IsCurrent: boolean;
}
```

Next thing to add to the project are the pipes, which are functions that given an input value they return a transformed value of such input. Two pipes have been defined for this assignment:

- name pipe: used to format each member's name, takes as input a string in format (surname, name) and returns in output a string in format (name surname);
- photo pipe: used to handle empty photoURL fields, takes as input the photoURL and the genderTypeID of the member, and returns the correct placeholder image in case no photoURL has been provided.

Figure 2.5: Name pipe source code.

```
export class NamePipe implements PipeTransform {

  transform(fullname: string) {
    let result = fullname.split( separator: ', ');
    return result[0] + ' ' + result[1];
  }

}
```

Figure 2.6: Photo pipe source code.

```
export class PhotoPipe implements PipeTransform {

  transform(value: string, gender: number) {
    if (value === "") {
      let photoURL;
      if (gender === 1) {
        photoURL = "assets/photo_gender1.png";
      } else if (gender === 2) {
        photoURL = "assets/photo_gender2.jpg";
      }
      return photoURL;
    } else {
      return value;
    }
  }
}
```

For the data caching part of the project it has been suggested to create a service, since they are responsible for accessing the data. In this case, only one service is defined (called member.service.ts) and comes with different getter methods and other methods that handle errors or manipulate responses in order to obtain specific resources. All of these functions returns an Observable<T> object, where T is the type of the model that the method returns. In order to exploit the data manipulation, some RXJS functions have been used, some of them being map and filter.

Figure 2.7: Getter methods of the service.

```
export class MemberService {

  private url = 'https://data.parliament.scot/api/';

  constructor(private http:HttpClient) { }

  getMembers(): Observable<Member[]> {
    return this.http.get<Member[]>(this.url + 'members');
  }

  getMemberByID(id: number): Observable<Member> {
    return this.http.get<Member>(this.url + 'members/' + id)
      .pipe(
        catchError(this.handleError)
      );
  }

  getMemberParties(): Observable<MemberParty[]> {
    return this.http.get<MemberParty[]>(this.url + 'memberparties');
  }

  getPartyByID(id: number): Observable<Party> {
    return this.http.get<Party>(this.url + 'parties/' + id);
  }
```

Figure 2.8: Methods that make use of RXJS functions.

```
getPartyByMemberID(id: number): Observable<Party> {
  return this.getMemberParties().pipe(
    map(memberParties => memberParties.filter(memberParty => memberParty.PersonID == id)),
    switchMap((memberParties) => {
      return this.getPartyByID(memberParties[0].PartyID);
    })
  );
}

getWebsites(): Observable<Website[]> {
  return this.http.get<Website[]>(this.url + 'websites');
}

getWebsitesByMemberID(id: number): Observable<Website[]> {
  return this.getWebsites().pipe(
    map(websites => websites.filter(website => website.PersonID == id))
  );
}
```

Finally, the last element to be added to the project is the routing part. In order to be able to navigate between the list page and the member details page, a routing module has been defined, which defines two main routes:

- a parliament route, which sends to the ParliamentComponent
- a member route, which sends to the MemberComponent

In all other cases (i.e. if the route is empty) the user is redeirected to the parliament route. Moreover, if there is a requested URL that doesn't match any path, the webapp redirects the user to the error page. This is done be defining a wildcard **.

Figure 2.9: App routing module source code

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { MemberComponent } from './components/member/member.component';
import { PageNotFoundComponent } from './components/page-not-found/page-not-found.component';
import { ParliamentComponent } from './components/parliament/parliament.component';

const routes: Routes = [
  { path: 'parliament', component: ParliamentComponent},
  { path: 'member', component: MemberComponent},
  { path: '', redirectTo: 'parliament', pathMatch: 'full'},
  { path: '**', component: PageNotFoundComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes,  config: {useHash: true})],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Deployment

Figure 3.1: The parliament members list page. The user is presented with the list of all members that compose the Scottish Parliament. In order to visualize the details of a certain member, the user must click on their name, which will redirect them to the peron detail page.

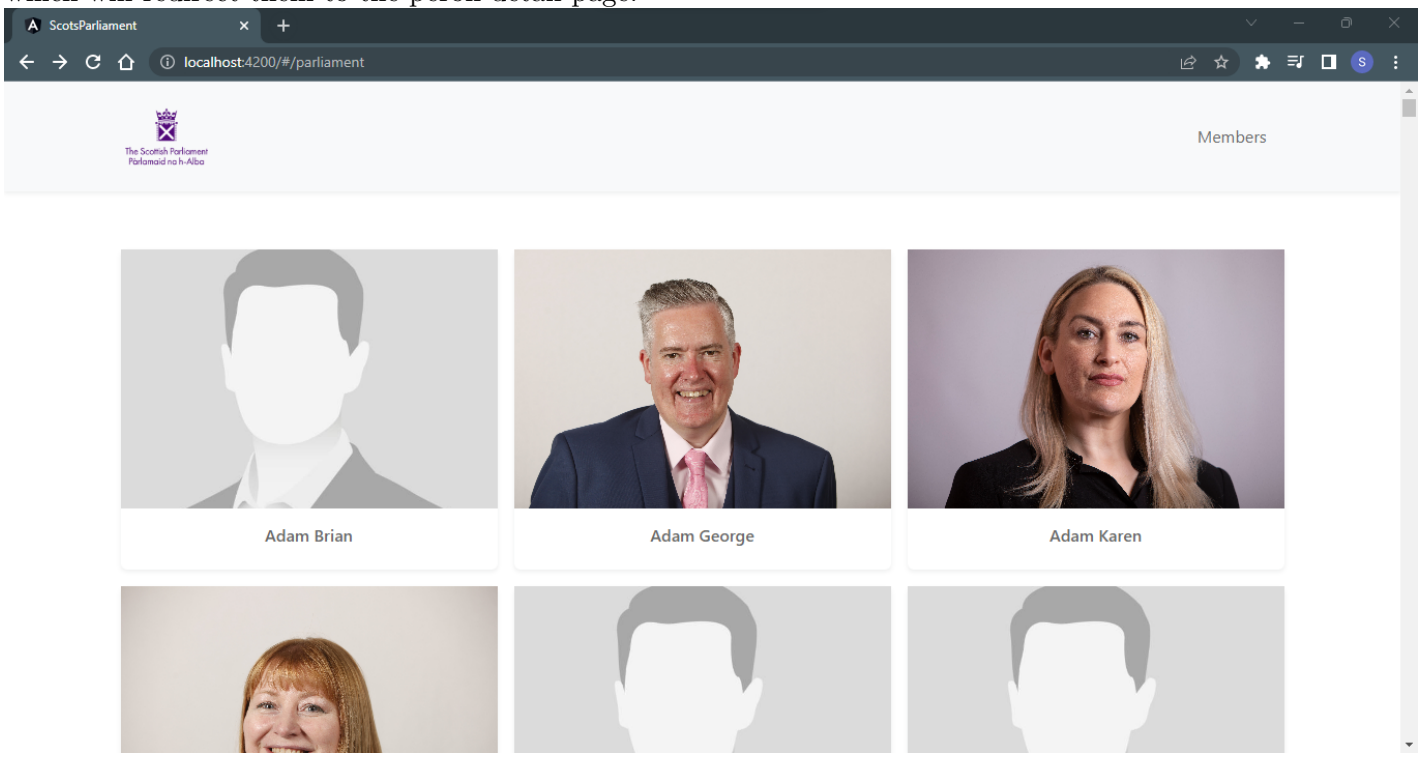


Figure 3.2: The person detail page. The user can visualize the details of the selected member. The page also contains a "Members" link that will redirect the user back to the first page, so s/he can see the list and eventually select another member to visualize.

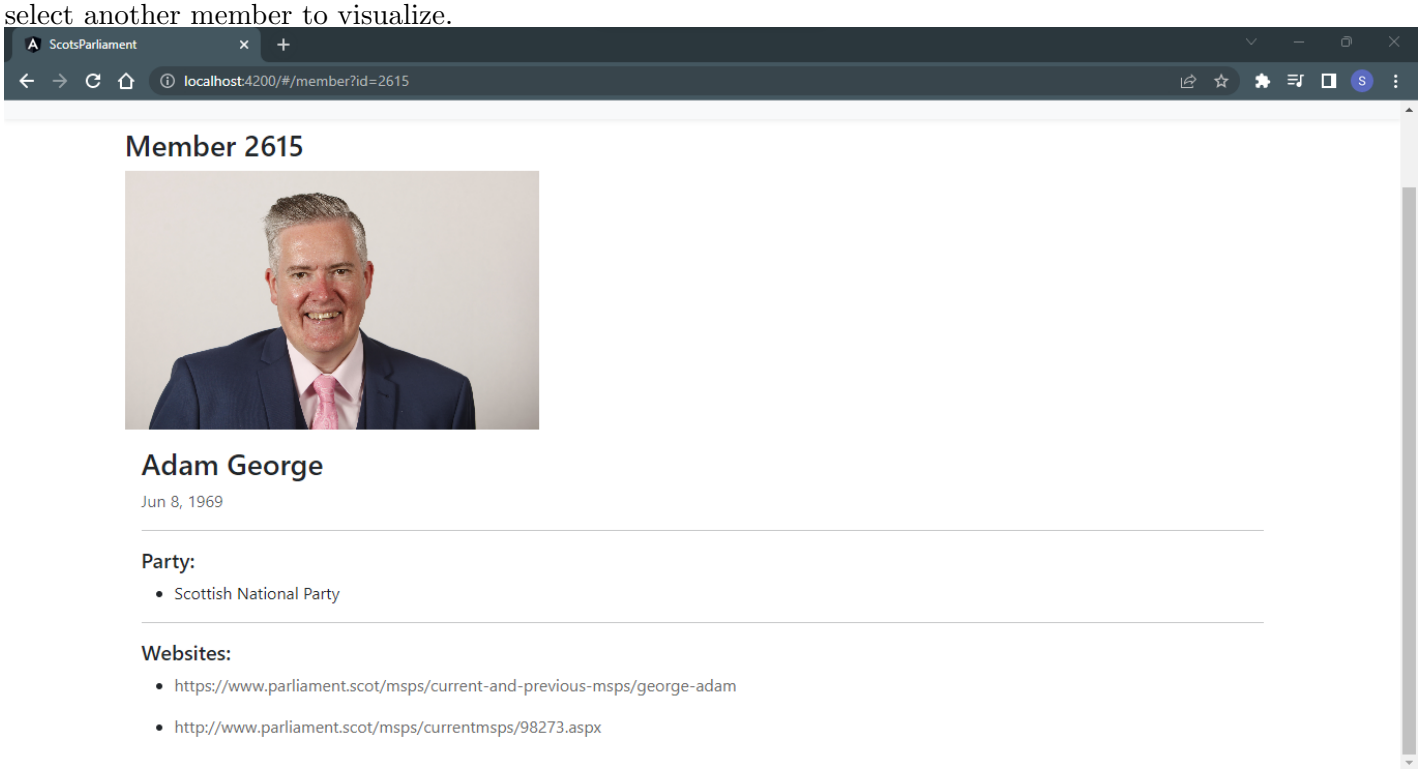
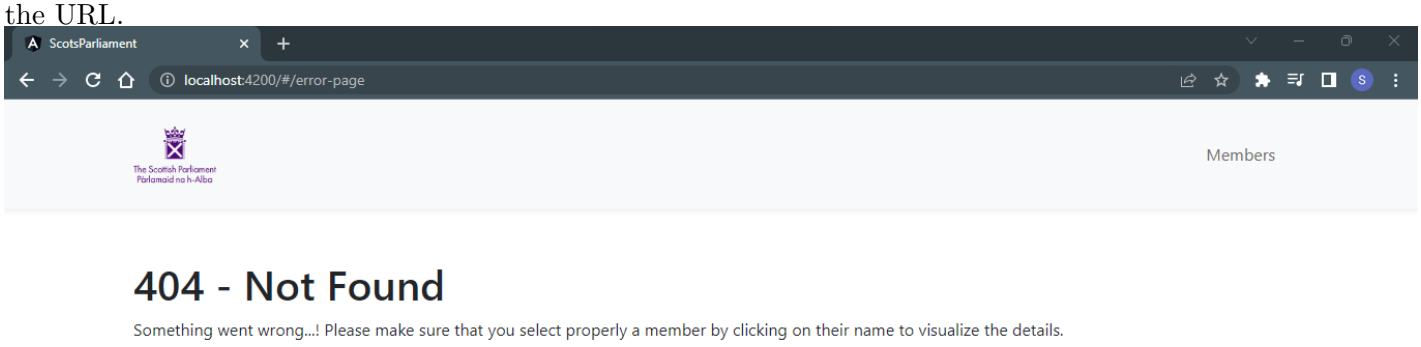


Figure 3.3: The error page. This page appears when the user puts a wrong ID on the URL or no ID is found in the URL.



Comments and problems

Overall the development of this assignment has not been problematic at all. The only issue I got has been the fact that moodle puts a limit of 100MB for each delivered file. Initially, my zip file had a size bigger than 100MB, but that was because I included also all the angular files initially created with ng new command. To solve this issue, I had to deliver only the src directory of my angular app, which still consists of the most important part of the app and can still be used in any angular project by replacing the original src directory of an angular project with the src directory delivered for this assignment.