Department of Information Engineering and Computer Science
Fintech
University of Trento

Web Architectures
Simone Bianchin

Assignment 5
6 January 2023

Index

The case

A little sum-up of the assignment.

1. Create an Angular project that fetches multiple data from the official APIs of Scottish.
2. Display the member of the Parliament in a list of cards. The click on a card will redirect to details of that parliament member.
3. The details page shows the PersonalID, Name and Surname, Born Date, Memberships, and Websites of the member concerned.

My implementation

Components list

1. toolbar: shows the title, the official logo and has a home button. It's shared between the two other components.
2. member-list: contains a CSS grid that displays the member's images and names with a material card.
3. member-details: show the concerned member details.

Recycling the Toolbar definition has prove the potential of a component pattern.

SCSS I decided to use SCSS to use his nest potential. It is indeed possible to refer the nested HTML childs of an element that implements the root class.

Listing 1: SCSS.

```
.members-list {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  padding: 1em 10px 1em 10px;
  column-gap: 10px;
  row-gap: 1em;

  mat-card {
    max-width: 400px;
    background: white;

    img {
      display: block;
      max-width: 100%;
      height: 200px;
      object-fit: cover;
      width: 100%;
    }
  }
}
```

Services

1. APIService

Since the APIs are pretty simple I created only one service that handles all the fetch calls. In the following listing, we can see the first example. No external libraries have been use in this process, so with the embedded JavaScript fetch function the endpoint is called and the response is parsed directly into the *json* constant.
An eventual HTTP bad response is handled by checking the *ok* attribute and evenutally the promise is rejected. Since APIs calls are asynchronous all the functions are marked as async to implement the *await* operator. At the end of the function the fallback image, in the event that there is no official photo, referring to two static images based on the member's gender.

Listing 2: Members Fetch.

```
async fetchMembers(): Promise<Member[]> {
    if(this.members) return this.members;
    return new Promise(async (resolve, reject) => {
      const response = await fetch("https://data.parliament.scot/api/members");
      if (!response.ok) reject()

      const json = await response.json();
      json.map((member: any) => {
        const image = new Image();
        image.onload = () => {}
        image.onerror = () => {
          const err = new Image();
          member.PhotoURL = "assets/" + (member.GenderTypeID === 2 ? "woman.jpg" : "man.png")
        }
```

```
        image.src = member.PhotoURL;
      })
      resolve(json as Member[]);
    })
  }
```

Caches To avoid useless HTTP calls, since the data is always requested in bulk, I've implemented a cache layer into the *APIService.ts*. It's very simple, at the start of each fetch I check if a class attribute has been filled before.

Listing 3: Cache Layer into APIService.ts

```
async fetchPartiesMembership(personID: number): Promise<[Party[], Membership[]]> {
    if(!this.parties) {
      const response = await fetch("https://data.parliament.scot/api/parties");
      this.parties = await response.json() as Party[];
    }

    if(!this.memberParties) {
      const response = await fetch("https://data.parliament.scot/api/memberparties");
      this.memberParties = await response.json() as Membership[];
    }

    return [
      this.parties,
      this.memberParties
        .filter(membership => membership.PersonID === personID)
        .sort((a, b) => {
          if (moment(a.ValidFromDate).isBefore(b.ValidFromDate)) return -1;
          if (moment(a.ValidFromDate).isAfter(b.ValidFromDate)) return 1;
          return 0;
        })
    ]
  }
```

Routing The routing implemented with Angular went pretty straightforward. It has been necessary to define the routes into the app module. The details page route defines a *memberId* as a prop, needed to filter the API data.

Listing 4: Cache Layer into APIService.ts

```
RouterModule.forRoot([
      {path: '', component: MembersListComponent},
      {path: 'member/:memberId', component: MemberDetailsComponent},
    ]),
```

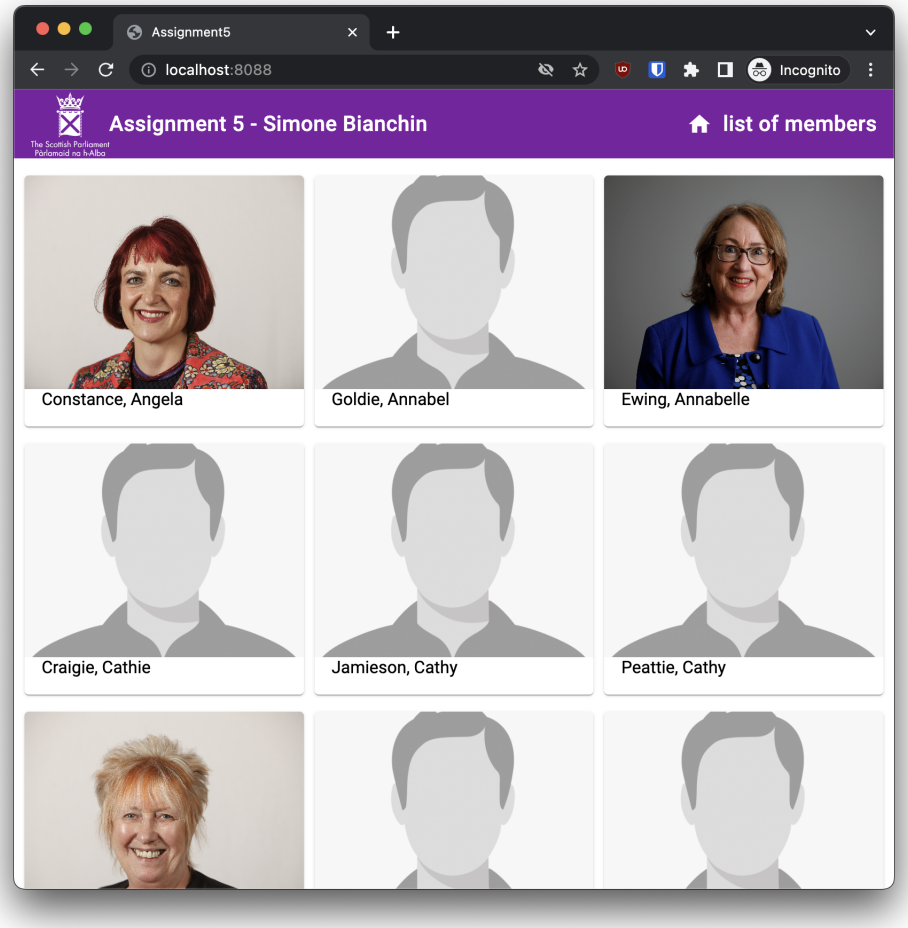
The navigation has been implemented with the *[routeLink]* attribute.

Listing 5: Cache Layer into APIService.ts

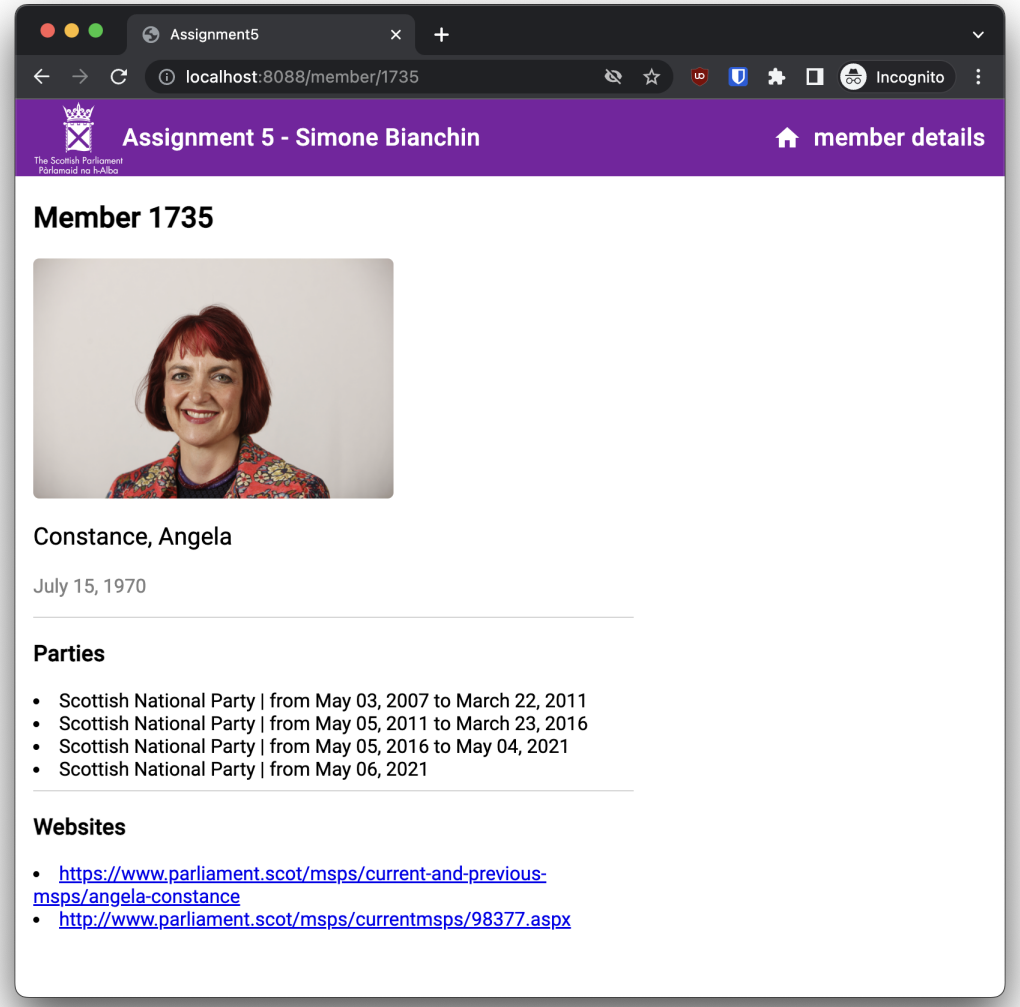
```
<img
      mat-card-image
      src="{{member.PhotoURL}}"
      alt="Photo␣of␣{{member.PreferredName}}"
      [routerLink]="['/member',␣member.PersonID]"
    >
```

App running

Here some screenshots of the app running.



Members list



Member details

Comments

Angular seems a very solid framework, built to be perfect for complex web apps. Also Typestrict has shown a strong potential by inferring strong types to variables. No problem came up during the development, so it has been very enjoyable.