Department of Information Engineering and
Computer Science
University of Trento, Italy

Web Architectures - A.Y. 2022/2023

Assignment 5

Authors:
De Menego Luca

Dec 26, 2022

Contents

1 Introduction

The fifth assignment involves the development of an Angular application that retrieves information about the Scottish Parliament, including its members and parties, from an API and displays this information to the user. In detail, the main page of the application should contain all the Parliament's members with some basic information about them. A separate kind of page, accessible by clicking on a specific member, provides to the user more information about the chosen member: birthday, picture, parties, websites. The application should be deployed on Tomcat.

2 Implementation

The application has been generated with Angular CLI version 15.0.4, and the `src` folder of the final solution is structured in the following way:

app:

components: directory containing all the developed custom components;

enums: exported enumerations (Gender, WebsiteType);

interfaces: directory containing all interfaces representing the API repsonse objects;

services: directory containing the developed services that directly interact with the API;

app.component.ts: main component of the application, containing a page template present in all the pages and the `router-outlet` component;

app-routing.module.ts: file containing the routing configuration module, exposing all available routes;

app-module.ts: main application module, declaring all generated components and setting up dependency injection for some modules.

assets: folder containing the images used by the application;

main.ts: application's starting point;

styles.css: global CSS styles;

index.html: main HTML file in which the application will be injected.

Before diving into the exposed components, it is essential to understand the general structure of the web application. As stated in the requirements, two main pages are available:

- `/members`: main page of the application, showing all the Parliament Members with name, surname, gender, image (if available) and a flag specifying whether the member is currently active or not;
- `/member/:id`: page containing details about a specific member, identified by the ID passed through the URL. This page shows: name, surname, active flag, gender, birthday, parties and websites.

Thanks to the routing module, requests for the root page will automatically forward the user to `/members`:

```
{ path: '', redirectTo: '/members', pathMatch: 'full' }
```

An additional component called `error` has been defined and integrated into the routing module, to which the user will be redirected whenever an invalid path is accessed:

```
{ path: '**', component: ErrorComponent }
```

To make the routing seamlessly work even within a Tomcat deployment, an additional option has been provided to the `RouterModule` configuration: the `useHash` property has been set to `true`. This allows to use the `HashLocationStrategy`, which makes the `Location` service represent its state in the hash fragment of the browser's URL. In fact, without this option any request manually entered by the user into the URL different from `/` would lead to a `404` error.

Another option to solve this problem would be to manually configure the Tomcat's rewrite rules. However the `HashLocationStrategy` seemed the best choice in our case since it allows to make the application work without server-side changes.

Five components have been created for this application:

- **member-base**: a card representing a specific member, showcasing only the most relevant information (name, surname, image, gender, active flag);
- **member-detail**: a card representing a specific member, showcasing more information about him, as requested in the requirements;
- **pagination**: component containing a set of `member-base` components, that implements pagination to limit the number of members shown within a single page. It represents the page `/members`;
- **member**: component containing a single `member-detail`, representing the page `/member/:id`;
- **error**: 404 error page.

All API calls are performed by an injectable service called `MembersService`. This service caches all relevant information gotten from the API into variables, so that no useless additional calls are performed. Apart from functions directly calling and returning the APIs results (`getMembers`, `getMemberParties`, `getParties`, `getWebsites`), the service exposes some functions that automatically join the information. For instance, the function `getMemberDetails(id: number)` returns all basic information about the member, together with his parties and websites. All the HTTP calls are performed via the injected `HttpModule`, and when necessary the results are piped through a map function that changes properties. For instance, when getting the Parliament Members, the map function sets up the birthday object to make it directly usable without additional modification in the UI:

```
this.members = await lastValueFrom(this.http.get<Member[]>('https://.../members')
    .pipe(
        // Convert date strings to the appropriate format for display
        map((response: Member[]) => {
            response.forEach((member: Member) => {
                member.BirthDate = member.BirthDate ? new Date(member.BirthDate)
                    .toLocaleString('en-US', {
                        month: 'long', year: 'numeric', day:'numeric'
                    }) : "";
            });
            return response;
        })
    ));
```

The components can then fetch all needed information by simply getting this service injected and calling the exposed functions from `ngOnInit`.

To style the pages, TailwindCSS has been used, a utility-first CSS framework packed with classes that can be composed to build any design. This framework limits the amount of CSS that must be used, making the code more maintainable and easier to read. TailwindCSS has been integrated by installing the npm packages `tailwindcss`, `postcss`, `autoprefixer` and running `npx tailwindcss init`.

To deploy the project on Tomcat, a `pom.xml` file has been configured that performs the following steps:

1. runs `npm install` to install all necessary dependencies for the application and `npm run build` to compile and build the application for production. This step is performed using `frontend-maven-plugin 1.7.6`;
2. creates a .war file (Web ARchive) from the build artifacts, which is a standard package format used to deploy web applications to a server. This step is performed using `maven-war-plugin 3.3.0`;
3. deploys the .war file to a Tomcat server, using the URL, username, and password provided as properties in the pom.xml file. This step is performed using `tomcat7-maven-plugin 2.2`.

The pom.xml file is an important part of the deployment process because it specifies all the necessary configurations and instructions for building and deploying the application. It tells the build tool (in this case, Apache Maven) what tasks to perform and in what order.

In addition to the steps mentioned above, the pom.xml file also includes other configurations such as the node and npm version. By configuring the pom.xml file in this way, the deployment process can be automated and run with a single command, making it easy to deploy updates and changes to the application without the need for manual intervention. The following command performs all the previously explained steps:

```
mvn clean install tomcat7:undeploy tomcat7:deploy
```

Apart from this kind of deployment, an additional possibility is available: the project can be also deployed on Github Pages. To do this, this project uses `angular-cli-ghpages`, which allows you to automatically build and deploy the application using the command:

```
ng deploy --base-href=/scottish-parliament/
```

The deployment works by creating an additional branch `gh-pages` if not present, where only the build artifacts in `/dist/scottish-parliament` are pushed to the repository. The published version can be found at the following link: Github Pages Deployment.

3 Deployment

The main page of the application showing a list of members can be seen in Figure 1. As we can see, all of them have the active flag set to true: this happens because I decided to give priority to them, by ordering the array of members based on this field. As previously explained, the main component used in this page is the `pagination` component, which shows the members in batches. Figure 2 shows it is possible to move from one page to the other using the *previous* and *next* buttons.

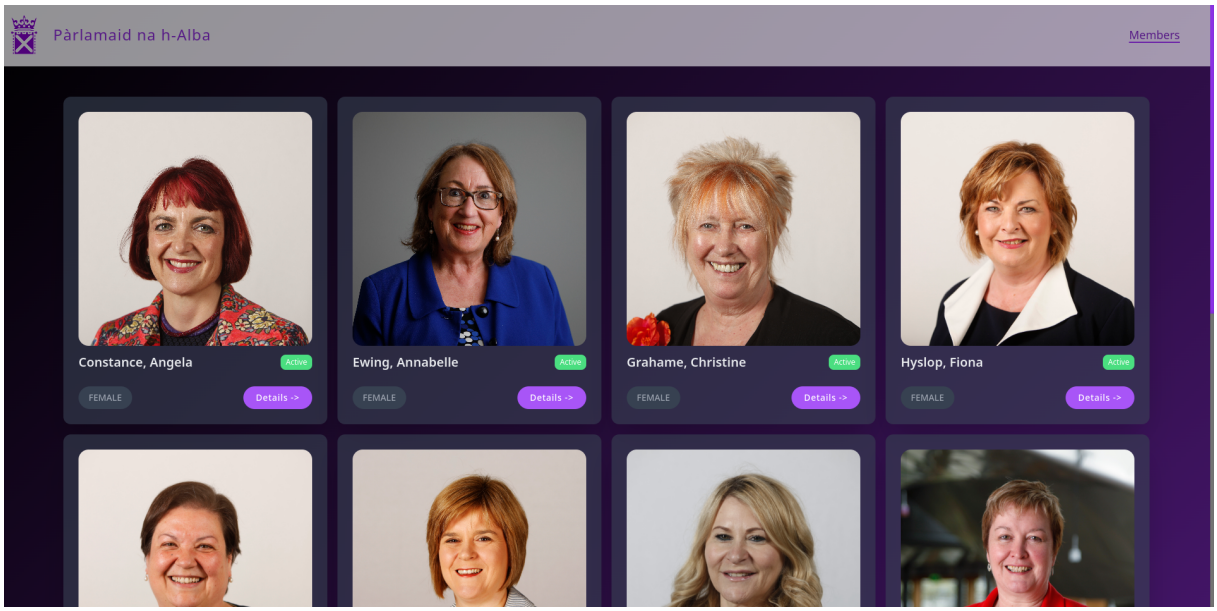

Figure 1: The home page of the web application

Figure 2: Showcasing the pagination compnoent

Details on a specific member can be found by clicking on one card in the members list. This leads the user to a new page that can be seen in Figure 3. This page also shows the parties linked to the member, with the corresponding `fromDate` and `toDate`. Finally, a list of websites related to the member is shown.

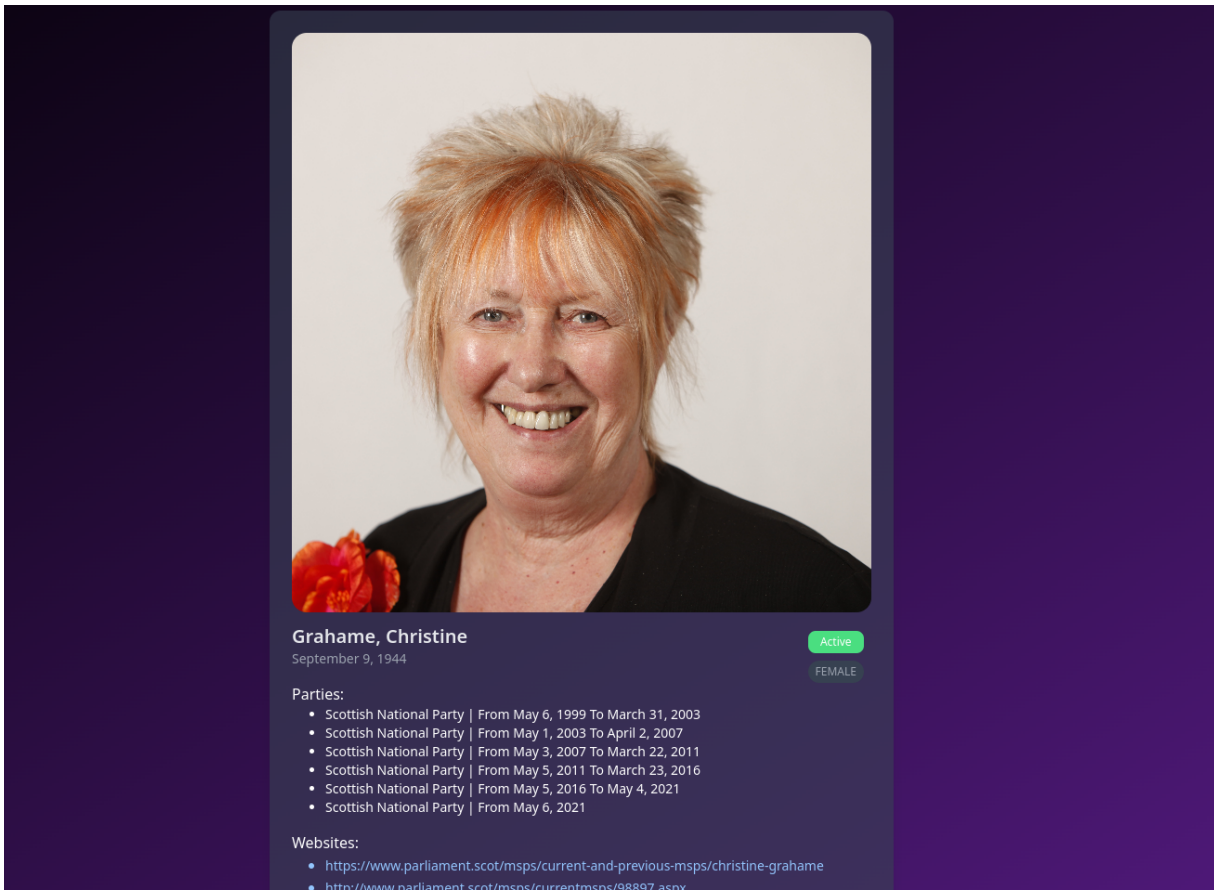

Figure 3: Details of a specific member

The routing module will automatically redirect all requests to wrong URLs to a 404 page (Figure 4). Moreover, if the user tries to search for a specific member by manually modifying the URL (e.g. `/member/123`) and the member does not exist, the user is again redirected to this error page.

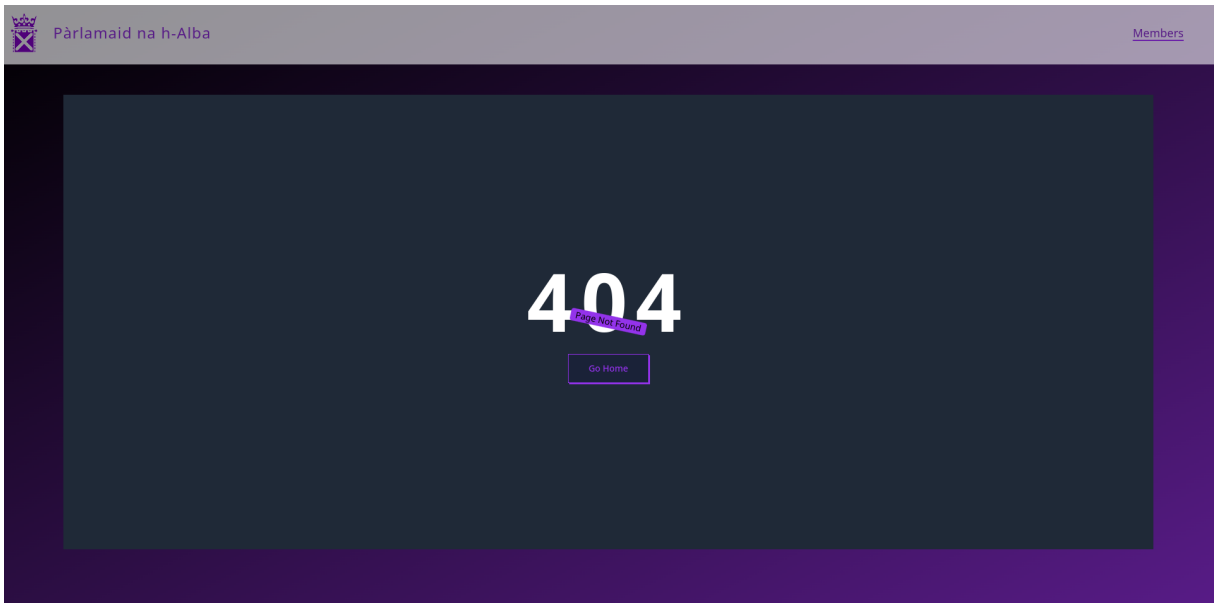

Figure 4: 404 Error page

4 Comments

One of the main problems with the application is that it can be slow to load, particularly when displaying images. This is due to the fact that the images are quite large in size, which can take a long time to download and display.

To partially address this issue, I implemented pagination in the application. This allows the images to be loaded in small batches rather than all at once, which can improve the overall performance. However, the application is still a bit slow, and further optimization may be necessary to fully resolve this issue. A potential solution to consider could include optimizing the size and resolution of the images, but this would require asking for the collaboration of the developers working on the Scottish Parliament API.