Web Architectures
assignment 4

Scottish Parliament - Angular

Sebastiano Chiari - 220527
sebastiano.chiari@studenti.unitn.it

November 27, 2021

1 Introduction

The aim of this assignment is to create an Angular application that, given the Scottish Parliament Open Data, lists the parliament members (showing their name and photo): each name is linked to an in detail page about that member, reporting its name, its picture (if available), the party to which the parliamentary belongs and its websites (if available).

2 Implementation

2.1 Components

Components are the main building block for Angular applications.

2.1.1 `app`

The `app` component is responsible for rendering the whole application. Inside the HTML template, it displays the `header` component and the routing output, which will be one of the other three components, `parliament`, `member` or `page-not-found`.

2.1.2 `header`

The `header` component is the one responsible for rendering the header of the web application. It contains a navbar with the logo of the Scottish parliament on the left and a "Members" button on the right side.

The "Members" button, when clicked, triggers the `loadMembers()` function, which allow the user to navigates through the Angular routing mechanism to the homepage (the parliament one).

2.1.3 `parliament`

The `parliament` component is responsible for displaying the compelling list of all members of the Scottish parliament. This component retrieves through the corresponding `member.service` 2.4 function the list with all the members and sorts it out in alphabetically order based upon their surname. Then, through an `*ngFor` directive, one `member-card` component is created for each member within the array, passing to the component the corresponding `Member` object through property binding.

2.1.4 `member-card`

The `member-card` component receives as input a `Member` object and displays a card with the clickable member name (which redirects to the member's personal page) and its photo: if no photo is available, a placeholder image is used (2.2).

2.1.5 `member`

The `member` component displays the personal page of a member given its ID. The ID is passed within a query param in the URL: if no query param can be found, the component redirects to the error page. Otherwise, the `MemberComponent` retrieves all the information through the provided `member.service` functions.

If the given ID do not correspond to any member (thus, the service function returns an error object), the component redirects to the error page. On the other hand, if the request returns successfully a member, this object is displayed: the `ParliamentaryName` field, the `BirthDate` (if not protected) and the `PhotoURL` are formatted through pipes (2.2). Then, party association and websites are fetched through services too 2.4.

Within the HTML template, a check if performed over the `BirthDateIsProtected` entry: if it's `true`, then no birth date is displayed. Otherwise, it is presented the content of the `BirthDate` entry. Websites are displayed through an `*ngFor` directive.

2.1.6 `page-not-found`

This component is used to render an error page if the user requests an out-of-scope route or in case of an HTTP error response from a service call (for example, if the given member ID is not found).

2.2 Pipes

Pipes are simple functions to use in template expressions to accept an input value and return a transformed value.

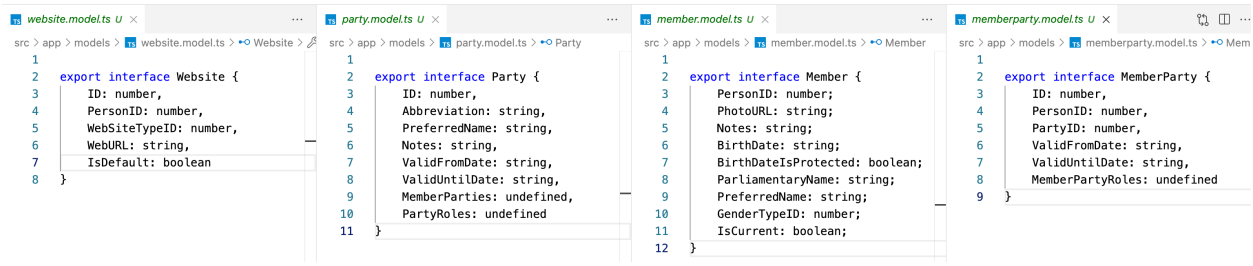
In this project, together with the built-in `DatePipe`, two custom pipes has been implemented:

- `NameFormatterPipe`, used to format each member's name, takes as input a `<surname, name>` string and returns a `<name surname>` string.
- `PhotoFormatterPipe`, used to deal with empty photoURL fields, takes as input the `member.PhotoURL` field and as parameter the `member.GenderTypeID`, returning the corresponding placeholder image if no photoURL has been provided.

2.3 Models

In order to improve readability and maintainability, different models has been created to mimic each response object from the API calls:

- `Member`, interface for a single member
- `MemberParties`, interface for a single association member-party
- `Party`, interface for a single party
- `Website`, interafce for a single website



2.4 Services

Within the Angular architecture, components shouldn't fetch or save data directly. Data access is delegated to services.

In this application, it is provided only one service, `member.service.ts`, with multiple functions: **direct API calls** (such as `getMembers()` or `getMemberGivenID(id)`) request single endpoints, while **complex API calls** (such as `getPartyGivenMember(id)`, returning the party which a given member represents) are more structured functions that manipulate responses in order to return specific resources.

Each function returns an `Observable<T>` object, where `T` depends on the model (2.3) the API returns. Data manipulation is done exploting some `RXJS` functions, such as `filter` or `map`.

2.5 Routing

As it can be seen from the `app-routing.module.ts` file, due main routes are defined:

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

The emtpy route redirects to the parliament one. More, a wildcard `**` has been created in order to catch all the requested URL that doesn't match any of the paths earlier in the list: it sends the user to the `PageNotFoundComponent`.

3 Comments and notes

A possible improvement can be implementing a lazy loading structure for the parliament page: since the list can be very huge and lots of elements (and images) have to be fetched, downloaded and displayed into one single page, this would improve the overall performances and the user experience.

3.1 Tomcat deployment

As suggested from the course forum, in order to solve Tomcat routing problems, Angular was instructed to use the `hash location strategy`: the URL will be composed in such a way (ex. `localhost:8080/#/ScotsParliament/parliament`) that what follows the `#` symbol

will be discarded by the server and will be processed only by the client (in our case, our Angular application). This allows to build a *single page application*, where there is only one page requested from the server (the root of the application) while all the other pages are just changes to the hash fragment which the client application deals with.

4 Results

In the following section, there are screenshots of the running application with descriptions that documents the various steps and scenarios.

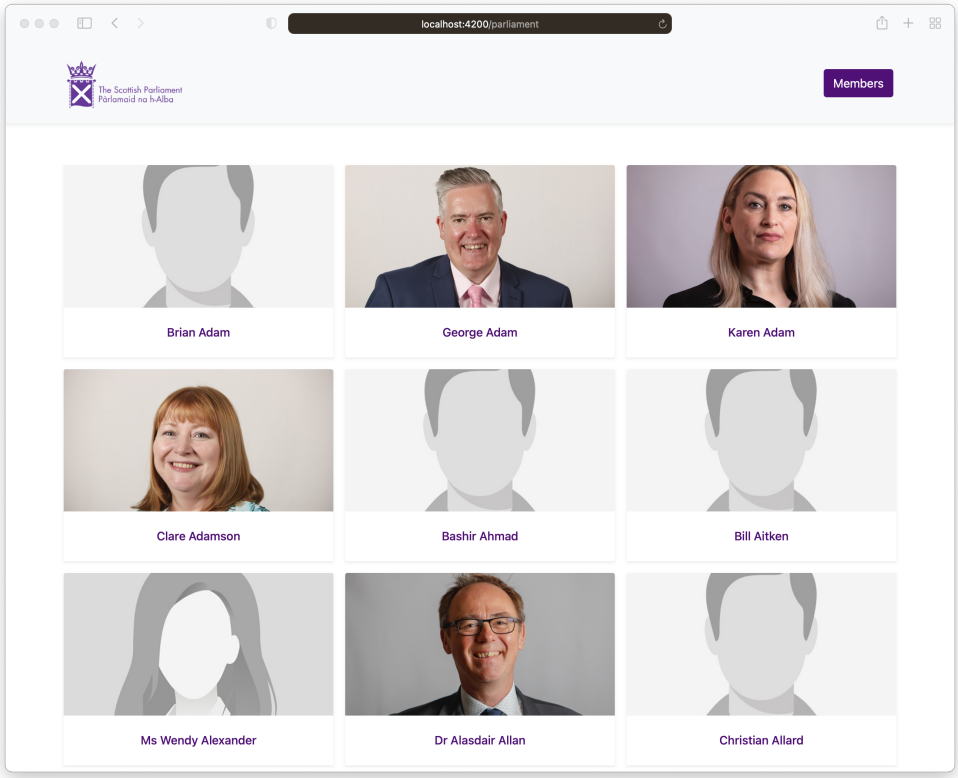


Figure 1: All parliamentary members displayed in one page

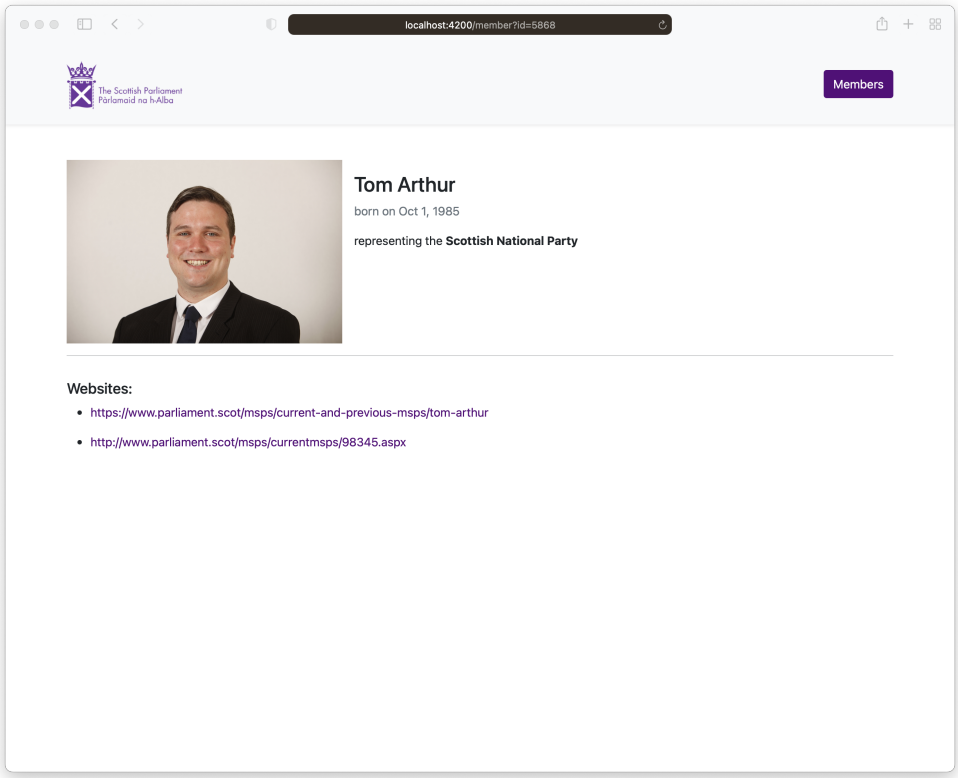


Figure 2: Single member page