Web Architectures
Assignment 5

Antonio Pio Padalino

December 26, 2022

Contents

1 Introduction

1.1 The project goal

The assignment concerns the development of an angular application which displays information about the members of the Scottish Parliament, using the API provided by the parliament official open data portal[4]. The application provides both a page listing all members with names and picture (if available), and individual pages showing, for a selected individual, their name, picture, birth date, the party/parties to which they belongs or have belonged in the past, and links to their websites. Finally, the application will be deployed to a Tomcat web server.

1.2 The project structure

To achieve the assignment goal, 2 main angular components have been used, both coming with their template `.html` file and the main logic in TypeScript:

- `all-members`, responsible for displaying a list of all Parliament members. The component has a method called `resolveMembers` that retrieves an array of all members information and assigns it to the `allMembers` property of the component. The property is then used to render the list of members in an HTML template associated with the component.
- `member-detail`, which is responsible for displaying detailed information about a Parliament member on a web page. The component obtains the information it needs to display by making HTTP requests to `ParliamentScot`.

The `.html` template files for both component are responsible for the presentation side:

- `all-members.component.html` is responsible for displaying a list of members as a series of cards, with each card showing the name, photo, and a button that links to a page displaying more detailed information about the person, and shows a loading message while the data is being retrieved.
- `member-detail.component.html` is responsible for displaying detailed information about a member, including the member's name, photo, birth date, parties, and websites collected inside a card, and again shows a loading message while the data is being retrieved.

The application also makes use of 2 services:

- `parliament-scot.service.ts` is responsible for fetching information about members, list of parties they have been in, parties names, and the members' websites.
- `cache.service.ts` is responsible for all the caching used by the components.

More on each section and their specific functioning will be discussed in the Methods section.

2 Methods

Before discussing of each component, here is a brief overview of their interactions.

The user can navigates to the Angular app in their web browser, which causes the root component, `AppComponent`, to be displayed. It has a router outlet where it displays the components specified in the routes defined in `app-routing.module.ts`.

If the user navigates to the route for displaying the list of *all members*, the `AllMembersComponent` is displayed in the router outlet, then the component makes a HTTP request[8] to the `MembersService` to retrieve the data for the members, and then binds the retrieved data to the template. The `all-members.component.html` at this point displays a list of cards, each containing a button that links to the route for that member detail.

If the user clicks on the button for a *member*, the router navigates to the route for the selected id[10], displaying more detailed information about the member and the `MemberDetailComponent` is displayed in the router outlet. The component retrieves the data it needs to display from the `ParliamentScot` and binds the data to the template, displaying detailed information about the member.

2.1 Implementation

Retrieving information

This code defines a service[2] called `ParliamentScot` that can be injected into other Angular components or services. The service makes HTTP requests to the Scottish Parliament's API to retrieve data on members, parties, and websites. The service defines four properties, each of which is a string representing the URL of a specific endpoint in the API. The service also has a constructor that takes in an `HttpClient` object, which is used to make the HTTP requests.

The service has five methods, each of which makes an HTTP request to the corresponding endpoint and returns the data as an Observable[9].

- `getMembers` fetch the whole list of members;
- `getMember` fetch information on a single member, given their id;
- `getMemberParties` fetch information on all the parties a member has been in, given their id;
- `getParties` fetch the complete list of parties with their ids;
- `getWebsites` fetch information on all websites available for all members.

In some cases, like for `getWebsites`, it might have been more appropriate to retrieve just the websites for a given member, instead of fetching the whole list and doing the filtering later. But the APIs provided by the Parliament website only allow for the `<api-url>/:id` syntax, where `id` is not the id of a member, but the id of the unique website.

The all-members component

The Angular component `AllMembersComponent` is responsible for displaying a list of all members on a web page. The component implements the `ngOnInit` callback method which is invoked when the component is initialized. In this case, the method calls the `resolveMembers` method, which attempts to retrieve the data for the members from the cache or using the APIs provided.

The `resolveMembers` method basically wraps the `getMembers` method provided by the service `ParliamentScotService` to make use of the caching service (more on that will be discussed later) and avoid unnecessary API calls. Inside the function, the data is processed by iterating over each element and creating a new object for each member with the relevant data. Those objects are then pushed into the `allMembers` array, ready to be displayed by the template.

The member-detail component

The `member-detail` is responsible for displaying a card with more detailed information on a chosen member. The component has a number of fields that store information about the person, such as their name, birth date, photo, but also parties they have been in with corresponding dates, and links to any personal or parliamentary websites they may have.

The component has a constructor that injects a number of services that are used to retrieve data about the member and their parties. These services include the `ActivatedRoute` service, which is used to retrieve the id of the member from the route parameters, and the `DatePipe` service, which is used to format the member's birth date as a string.

The component also has a number of methods which - similarly to the previous case - are used to retrieve data from the API and store it in the component's fields.

- The `resolveMember` method uses the `ParliamentScot` service to retrieve the basic information about the member, such as their name, birth date, and photo.
- The `resolveMemberParties` method uses the same service to retrieve information about the parties the member has belonged to, their ids, and the dates the member was a member of those.
- The `resolveParties()` method uses the again the same service to retrieve a list of all parties ids and their corresponding names, and it is used in tandem with the previous method to make possible for the application to display the actual names of the parties instead of their id. The method also deals with scenarios in which a member is listed as being in the same party for consecutive time windows: in such case "contiguous" time windows are merged into one single chunk, so that if a person has been in parties, say, `[1,1,1,2]` with *from* dates `[t1,t2,t3,t4]` and *until* dates `[t2,t3,t4,null]`, the final result will be that said member has been just in parties `[1,2]` with from dates `[t1,t4]` and until dates `[t3,"present"]`.
- The `resolveWebsites()` method, lastly, uses the service to retrieve a list of available websites for a given member. In particular, upon inspection of the JSON response obtained from *https://data.parliament.scot/api/websites*, it appears that for each member at most 3 websites are listed, each represented by a different `WebSiteTypeID`. The first two URLs, redirect to the official government page of the Parliament member, and the main difference mostly resides in the web page being of a previous member or a current one (as also highlighted by `/current-and-previous-msps/` or `/currentmsps/` inside the two URLs). But if both are available for a same user, the pages they redirect to are identical. The 3rd URL instead refers to the Parliament member's personal website. Having said that, the logic for this section is to choose at most 2 links to display, one for a *"government website"* and one for a *"personal website"*: if both the first 2 are available, the first one is preferred, arbitrarily.

The component has an `ngOnInit()` method that is called when the component is initialized: it invokes all the `resolve-` methods described so to prepare the template for the presentation of all information.

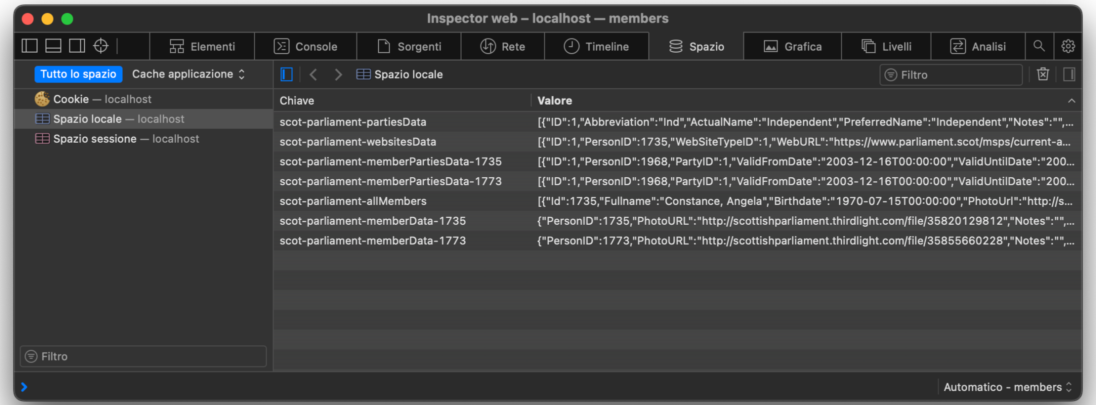
Caching service

The Angular app also implements a caching service[3] to store some of the data fetched from the Scot Parliament APIs, so to avoid unnecessary calls in the future. The service exposes methods for interacting with the browser's local storage[5]: a static *get* method that takes a key as input and returns the corresponding value from local storage if it exists, and a static *set* method that takes a key and a value as input and stores the value in local storage under the specified key. The *get* and *set* methods both prefix the keys with a static string `scot-parliament-` to prevent key collisions with other data stored in local storage.

The service is used in both the all-members component and the member-detail component.

- In the first component, the whole list of members with info needed to display is cached altogether;
- In the second one, the cached information regards name, birth date and photo URL information, the list of a parties, the list of parties ids and names and the list of websites of all members. Note that the first two are member-specific, and their key in the local cache is hence followed by their `id`, while the last two are generic data which can be fetched once and reused indefinitely.

The caching service is used inside the `resolve-` methods, which "wrap" those exposed by the service `parliament-scot.service.ts`. The basic logic of such methods consists in checking if the data they need is already stored in the cache using the `CacheService.get()` method, then if the data is found in the cache, the functions return the cached data immediately without making a network request, else a request to retrieve the data is made, and the retrieved data is stored in the cache using the `CacheService.set` method before returning the data. Overall, this allows for faster loading times if a member-detail page has been already visited once, and also a faster navigation in the all-member page, for which an API request will be most probably done just once, when the users launch the app.



Chiave	Valore
scot-parliament-partiesData	[{"ID":1,"Abbreviation":"Ind","ActualName":"Independent","PreferredName":"Independent","Notes":"",...
scot-parliament-websitesData	[{"ID":1,"PersonID":1735,"WebSiteTypeID":1,"WebURL":"https://www.parliament.scot/msps/current-a...
scot-parliament-memberPartiesData-1735	[{"ID":1,"PersonID":1968,"PartyID":1,"ValidFromDate":"2003-12-16T00:00:00","ValidUntilDate":"200...
scot-parliament-memberPartiesData-1773	[{"ID":1,"PersonID":1968,"PartyID":1,"ValidFromDate":"2003-12-16T00:00:00","ValidUntilDate":"200...
scot-parliament-allMembers	[{"Id":1735,"Fullname":"Constance, Angela","Birthdate":"1970-07-15T00:00:00","PhotoUrl":"http://s...
scot-parliament-memberData-1735	{"PersonID":1735,"PhotoURL":"http://scottishparliament.thirdlight.com/file/35820129812","Notes":"",...
scot-parliament-memberData-1773	{"PersonID":1773,"PhotoURL":"http://scottishparliament.thirdlight.com/file/35855660228","Notes":"",...

Figure 1: The content of the local storage of the browser cache after navigating first on the `all-members` page and then on the `member-detail` pages for members #1735 and #1773

Some notes on the presentation

- `*ngIf`[7] clauses are frequently used across all templates to make sure the properties that are trying to be accessed are ready;
- a `isLoading` ng-template is used in both templates to accounts for values that are still unavailable. Also another ng-template, `noWebsites`, is used to account for scenarios in which no websites are available for a given member;
- the clause `*ngIf="memberLoaded && partiesLoaded && websitesLoaded` in the `div` containing the `member-detail` card is to make sure every information displayed inside the card has been correctly received before showing the final result. Otherwise, the results would be a card visibly loading the different information over time, in an unpleasant view. In this way the presentation of a card is seamless (and still quite fast). The conditions the `*ngIf` clause checks refer to boolean properties set to true in the `member-detail` component when each chunk of information has been handled correctly.
- regarding photo URL in both templates, two `*ngIf` clauses check for the URL availability and if needed display a generic user photo[1]. Also, the `onerror` event is used to handle cases in which the photo URL is actually available, but some error is encountered while getting the picture (as for Parliament member *Richard Lochhead*).

2.2 Deployment

The `app-routing.module.ts` has been set so that

- the route `/members` triggers the component `AllMembersComponent`
- the route `/member/:id` triggers the component `MemberDetailComponent`
- an empty route redirect to the `/members`, to show the whole members list as default page

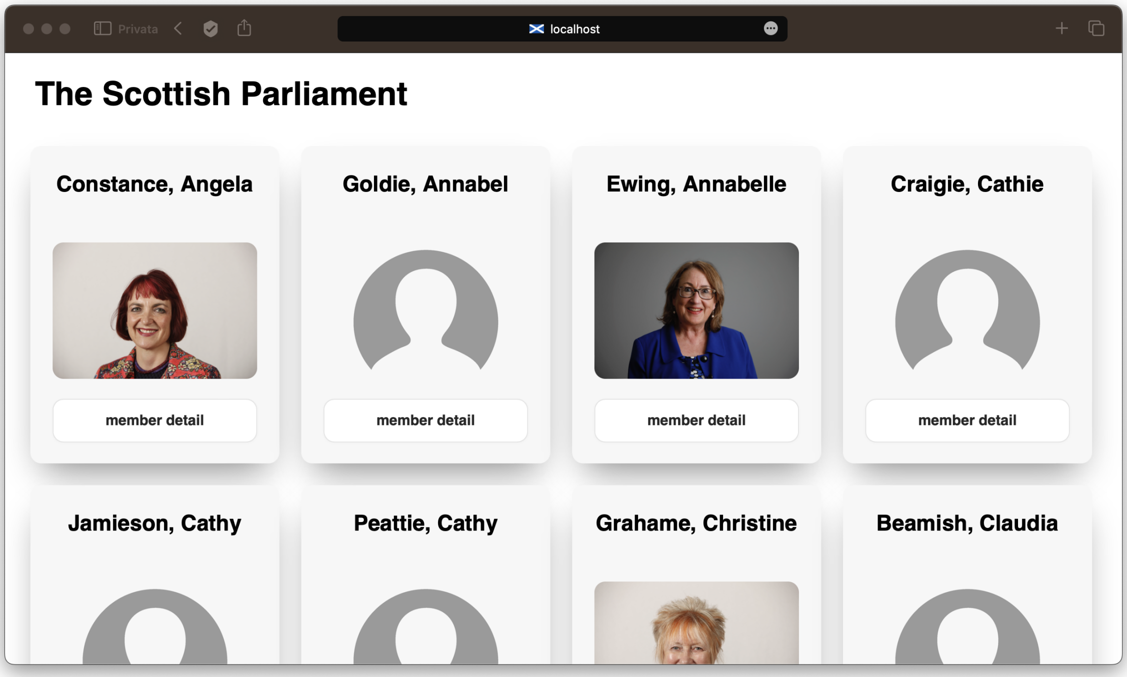


Figure 2: The presentation of the `all-members` component. All Parliament members are displayed in grid view, each card containing a button routing to the all member component with the respective `id`

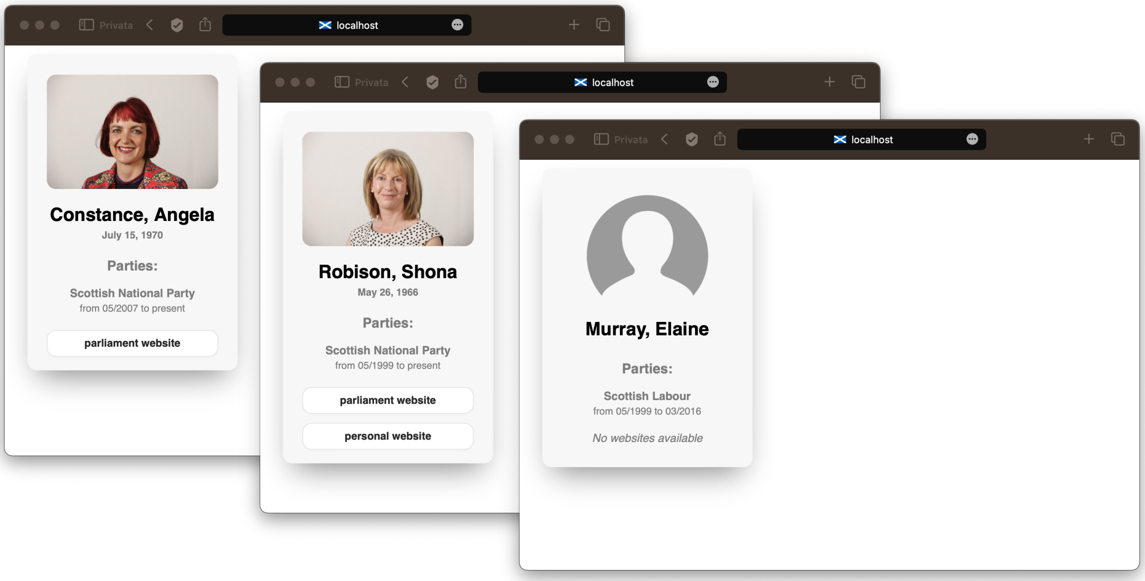


Figure 3: The presentation of the `member-detail` component. A card for the Parliament member with the `id` specified in the URL is displayed, together with the member's information and buttons to all available websites.

After developing and debugging the Angular application using the Angular CLI server (with `ng serve`), the app has been built for production with `ng build -configuration=production`, and the content of the `dist/scot-parliament` directory has been copied into `/src/main/webapp` inside a newly created IntelliJ project. The result and behavior of the app is the same as it was tested previously on the Angular CLI server.

3 Comments and notes

3.1 Deployment issues

Resources not found on launch

During the deployment phase an issue, which caused a 404 error when trying to access the app, was encountered. The problem was solved[6] thanks to a couple of adjustments.

- The issue was handled first by modifying the `index.html` file of the built application at line 4, which is now `<base href="./scot-parliament">` instead of `<base href="/scot-parliament">`, making sure the URL always points to the correct sources.
- The issue still persisted on page refresh inside the web app. To completely solve the problem, a `web.xml` file was created into the `src/WEB-INF` directory of the Angular application, with:

```
<web-app>
    <error-page>
        <error-code>404</error-code>
        <location>/index.html</location>
    </error-page>
</web-app>

```

  and the `angular.json` file in the project root was also modified so that the field `"assets"` now also contains `"src/WEB-INF"`. The point, in this case, is to direct the user to the `index.html` in the correct location on 404 errors. At this point the application was re-built and deployed, and both issues were solved.

3.2 Possible improvements

Improving the caching system

A caching service has been implemented and discussed to avoid unnecessary repeated API requests. However, it is important to keep in mind that cached data can become stale over time. For example, if the data in the database or network resource changes, the cached data will no longer be up-to-date. In this case, it is necessary to invalidate the cache and fetch fresh data from the source.

There are a few reasons why it is important to invalidate the cache from time to time:

- *Accurate data*: If the cache is not invalidated, the application will continue to serve stale data to the user, which may not be accurate or up-to-date.
- *Performance*: if the cache is not invalidated and the data in the source changes frequently, the cache may end up storing a large amount of stale data that is never used. This can waste resources and potentially decrease the overall performance of the application.
- *Consistency*: if multiple users are accessing the same data and the cache is not invalidated, each user may see different versions of the data depending on when they last accessed it.

To invalidate the cache, a mechanism that is able to detect when the data in the source has changed, and clear the corresponding entries from the cache may be implemented. There could be several ways to do this, such as using cache expiration times, cache tagging, or cache invalidation callbacks. The specific approach of course depends on the nature of the data we are working on, and it would be helpful to know how frequently they are updated.

References

[1] *Angular 2 - show placeholder image if img src is not valid.* https://stackoverflow.com/questions/36026428/angular2-show-placeholder-image-if-img-src-is-not-valid.

[2] angular.io. *Angular services tutorial.* https://angular.io/tutorial/tour-of-heroes/toh-pt4.

[3] betterprogramming.pub. *How to create a caching service for Angular.* https://betterprogramming.pub/how-to-create-a-caching-service-for-angular-bfad6cbe82b0.

[4] data.parliament.scot. *the Scottish Parliament - Open Data portal.* https://data.parliament.scot/api/members.

[5] levelup.gitconnected.com. *Simple caching with local storage.* https://levelup.gitconnected.com/simple-caching-with-local-storage-bbb02dd12d7a.

[6] stackoverflow.com. *404 error on refresh for angular deployed on tomcat server.* https://stackoverflow.com/questions/47366792/404-error-on-refresh-for-angularv4-deployed-on-tomcat-server.

[7] ultimatecourses.com. *Angular ngIf-else-then.* https://ultimatecourses.com/blog/angular-ngif-else-then.

[8] www.tektutorialshub.com. *Angular HTTP Get example using httpclient.* https://www.tektutorialshub.com/angular/angular-http-get-example-using-httpclient/.

[9] www.tektutorialshub.com. *Angular observable tutorial using rxjs.* https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/.

[10] www.tektutorialshub.com. *Angular passing optional query parameters to route.* https://www.tektutorialshub.com/angular/angular-passing-optional-query-parameters-to-route/.