Chapter 1

Front-End

This chapter discusses the front-end. We start by presenting the technology stack in section 1. After that we describe the most important control flows in the front-end in the section Design.

1 Technology Stack

This section devotes on section to each technology on the front-ends stack. For each technology we will describe what it does and why we opted to use this technology in favour of other options.

1.1 AngularJS

AngularJS is a web application framework by Google for creating dynamic web applications.

AngularJS depends on JQuery, thus we had to include that to. We have also used JQuery in some places for the removing and adding of classes where to resulted in neater code than using Angular directives.

We chose to use AngularJS since one of us had some experience with it. We considered Backbone as an alternative but dropped it for its flexibility, which may be great if you are an experiences web developer and know exactly what you want, but would have been too much for us. Furthermore it is advised to add a framework on top Backbone which would have added to the learning curve for Backbone [4].

Apart from the previous experience the fact that AngularJS was specifically developed for single page applications is what made us decide in favour of AngularjS.

One of the downsides of AngularJS is that its learning curve, after learning the basic features is quite steep. The documentation is rife with Angular-specific terms which makes it hard to read.

Extensions

We have used some extensions to Angular. We list them and shortly describe their functionality.

ngRoute provides routing and deep-linking services and directives for angular applications [3]. We use it to make it possible to store a link to a specific part of our application.

ngFacebook provides an interface between AngularJS and the Facebook API. We have adapted the source of this service in some places to make it better suited to our purposes and to keep it consistent with `ngLinkedIn`.

ngLinkedIn is for LinkedIn what `ngFacebook` is for Facebook. We have also adapted its source code to extend its functionality and to keep it consistent with `ngFacebook`.

angular-md5 is a service that computes MD5 hashes. We use it to hash the Facebook and LinkedIn identifier of our users, so that we can find multiple searches of one user without storing the actual user.

1.2 Bootstrap

Bootstrap is a framework by Twitter which places emphasis on responsive design.

Although there are alternatives to Boostrap, i.e. Zimit, InK or Pure, we did not really consider them since we were both familiar with Bootstrap and felt that we had enough new technologies to worry about. We did however use a different theme from `http://bootswatch.com` to avoid the distinctive Bootstrap look and feel. We add Font Awesome for the icons it provides.

1.3 UI Bootstrap

UI Bootstrap is an Angular version of the JavaScript part of Bootstrap. As far as we could find there are no alternatives other than building the directives yourself.

In the end we had to write the carousel at the login window, ourselves since the custom buttons we wanted were not supported by UI Bootstrap.

1.4 Dangle

"Dangle is a set of AngularJS directives that provide common visualizations based on D3."[2]

We have used it to create the pie charts shown on the statistics page, after practically rewriting the directive. The directives provided by Dangle did not handle long labels, and pie charts with a lot of pieces very well. To solve this we

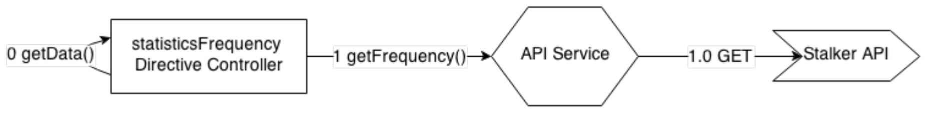

Figure 1.1: A schematic overview of the control flow of requesting the statistics and showing them to the user.

have removed all code from the pie chart directive that added labels and added a separate legend. The one upside of Dangle was that it expected its input in the same format our map-reduce-operations returned it.

In hindsight we should have chosen something other than Dangle, however we did not look further since Dangle played nice with AngularJS.

1.5 RequireJS

RequireJS is a JavaScript file and module loader that is optimized for in-browser use. Neither one of us had any experience with anything like it and since RequireJS was mentioned during the lectures we chose to use it. We used the project structure suggested by Brad Green [1].

2 Design

In this section we discuss the control flow in the front end upon certain actions of the user. To avoid a lengthy report we will only discuss the flow of successful cases.

2.1 View Statistics

The control flow when a user views statistics is presented in Figure 1.1. We have defined a directive that displays a pie-chart for a statistic. When the element defined by this directive is loaded the method `getData()` is called without inference from the user.

2.2 Log In

As mentioned earlier users have to log in to a social network before being allowed to stalk somebody on that network. Since we store the ID of the user and we have not implemented an api call that adds for example the LinkedIn identification to a user that we have already stored with its Facebook identification.

Figure 1.2 presents the flow of control when a users logs in.

The `loggedIn`, and its not shown equivalent `loggedOut`, event ensure that the `searchController` knows that a user is logged in. The `Facebook Service`, `ngFacebook`, is discussed section 1.1 on page 1.

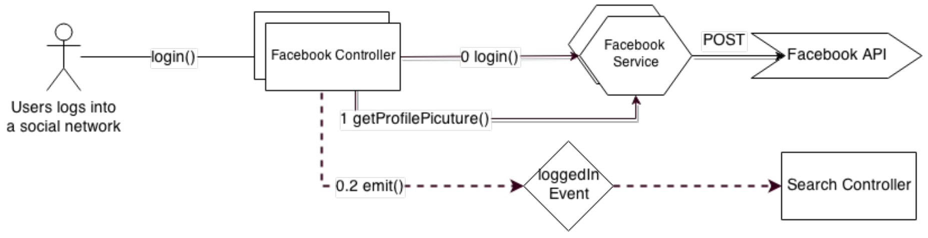

Figure 1.2: A schematic overview of the control flow when a user logs in.

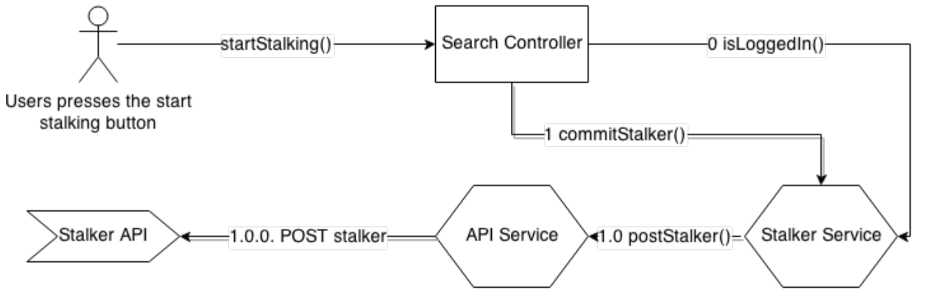

Figure 1.3: A schematic overview of the control flow when a presses the 'start stalking' button.

Logging out of social network is comparable to logging into it. Figure 1.2 uses Facebook, the application works in exactly the same way for e.g. LinkedIn.

2.3 Start Stalking

Figure 1.3 shows what happens in the front end when the indicates that he has logged in into all social networks he wishes to use.

If the POST to the `Stalker API` fails no error is reported to the user, since it does not matter to the user if his information is added to our database.

2.4 Stalk

Figure 1.4 presents the flow of control when a user searches for somebody on the social networks that he has logged in to.

The `Stalker Service` contains all information on the stalker. The `API Service` contains all calls to our own API, we have chosen for a separate service so that we only have to store the address of the API in one place. The Search Service broadcasts the search to the different social media controllers, in Figure 1.4 Facebook is shown as an example. Each social network controller does a search call by calling some function in the matching service that interacts with the actual API of that social network.

We had to use a broadcast instead of an emit as in section 2.2 since we had

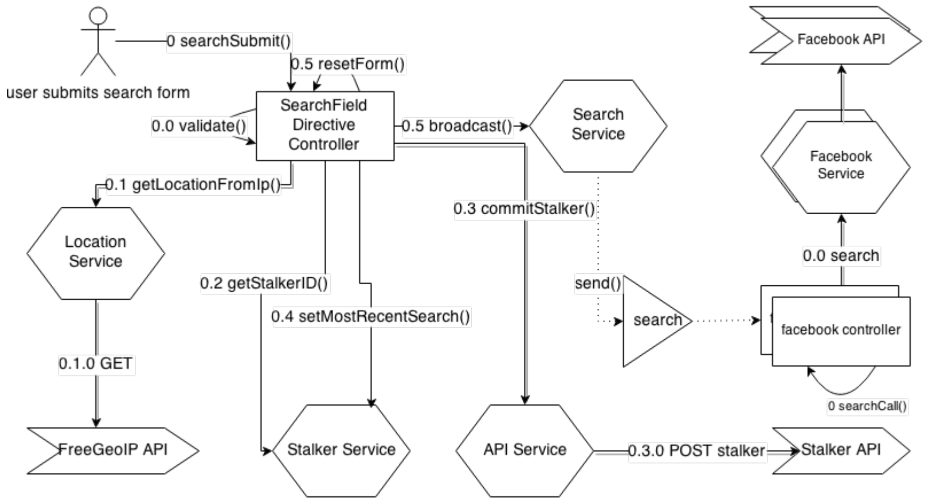

Figure 1.4: A schematic overview of the control flow when a user stalks somebody.

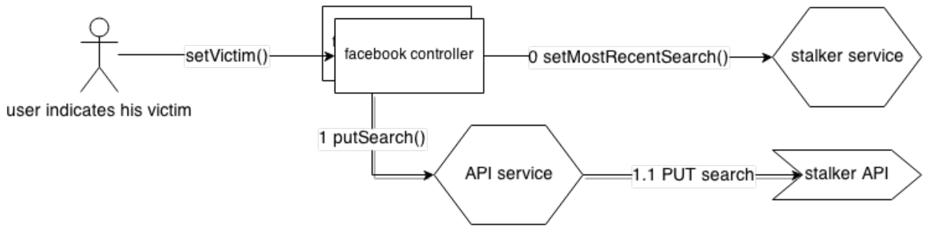

Figure 1.5: A schematic overview of the control flow when a user selects somebody as his victim.

to down in the scope hierarchy instead of up.

We added the start stalking button to force our users to first login to all social networks they wanted to use before actually stalking somebody. This was one of the concessions we had to make due to time constraints, since this button ensures that we can post complete stalkers, instead of having to update a stalker every time he adds a social network.

2.5 Select Victim

Figure 1.5 shows the control flow in the front end when a user indicates that one of the found results was his victim. Even tough it does not matter to the user if the addition of his victim to our database has been reported we report success, and failure, since the user has no feedback on his actions otherwise. Once again we have chosen Facebook as an example, but the flow of control would be the same with a different social network.

Bibliography

[1] Shyam Seshadri Brad Green. *AngularJS*. O'Reilly Media, 2013.

[2] *dangle.js*. URL: `https://fullscale.co.dangle/`.

[3] *ngRoute*. URL: `https://docs.angularjs.org/api/ngRoute`.

[4] Sebastian Porto. *A Comparison of Angular, Backbone, CanJS and Ember*. URL: `http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/`.