STALKER

Web & Cloud Computing

Rick van Veen (s1883933)
Laura Baakman (s1869140)

November 2, 2014

Introduction

This report accompanies the project for the course Web & Cloud Computing, that we have called STALKER. The main functionality our application STALKERis that it allows a user, the stalker, to request information on another person, the victim, on several social networks at once. Currently we support Facebook and LinkedIn, however due to the modularity of our application adding another social network should be easy. Unfortunately all social networks we tried require that a registered user log in before they allow you to search their user database.

The application also offers users the ability to view anonymous statistics on other users of the website, for example one can see from which countries the most searches originate.

In the next chapter we discuss the technology stack and control flow of the front-end. Chapter 2 does the same for the back-end. The last chapter gives an evaluation of the project.

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 [10].

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 [9]. 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, see figure 1.2 on page 5, 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 have removed all code from the pie chart directive that added labels and added a separate legend, see figure 1.7 on page 7. 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 will 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 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 1.1 presents the flow of control when a users logs in. The views that are presented to the user before and after logging in are presented in figure 1.2.

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 2.

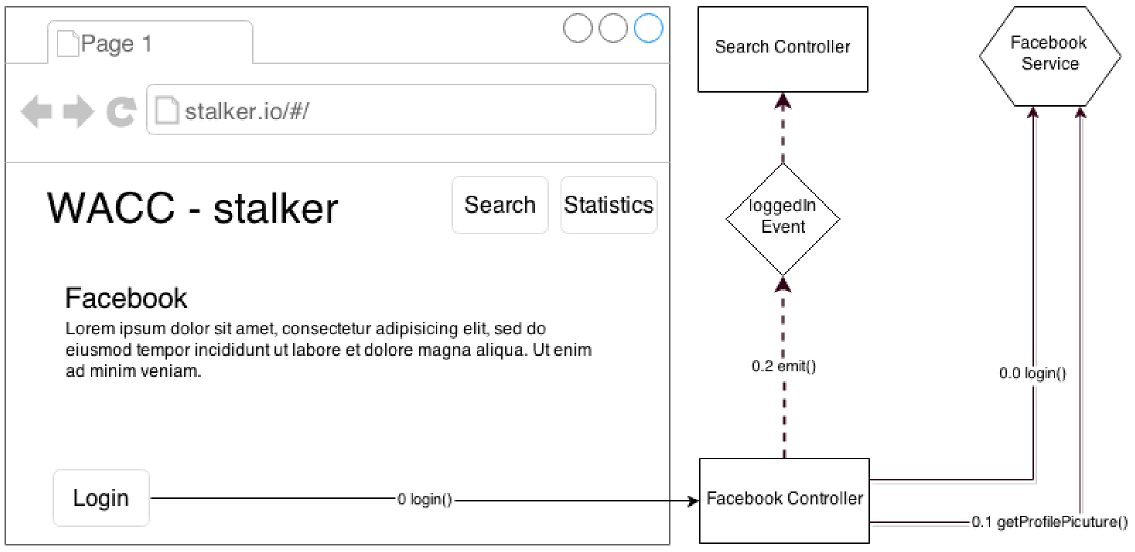


Figure 1.1: A schematic overview of the control flow when a user logs in.

Logging out of social network is comparable to logging into it. Figure 1.1 uses Facebook, the application works in exactly the same way for e.g. LinkedIn.

2.2 Start Stalking

Figure 1.3 shows what happens in the front end when the button 'start stalking', see figure 1.2, is pressed.

2.3 Stalk

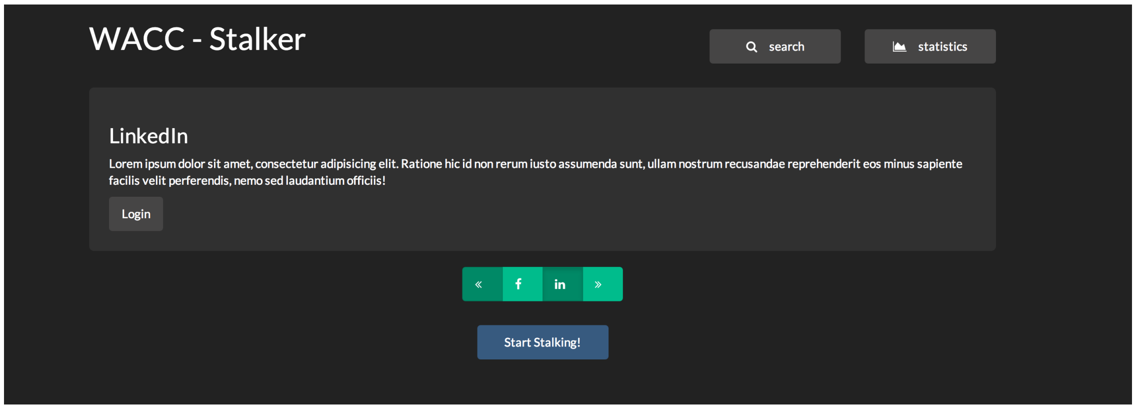
Figure 1.4 presents the flow of control when a user searches for somebody on the social networks that he has logged in to. The views presented to the user are shown in figure 1.5.

Begeleidend verhaaltje schrijven

2.4 View Statistics

The control flow when a user views statistics is presented in figure 1.6. Figure 1.7 shows how the statistics are presented to the user.

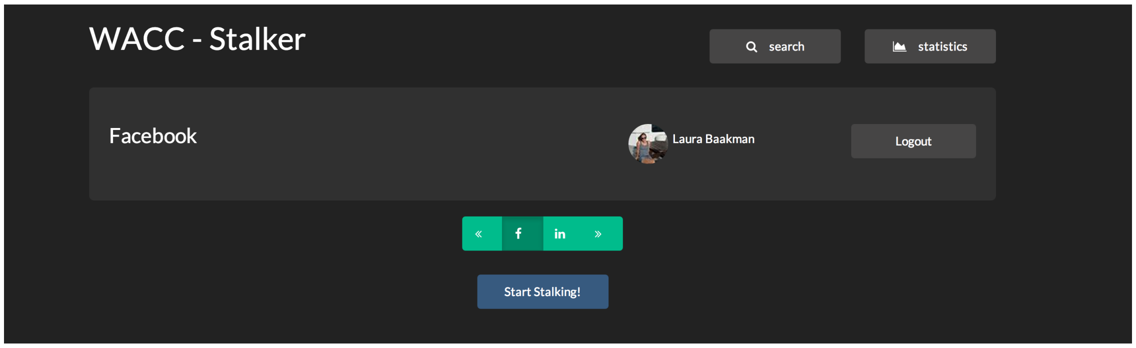
Begeleidend verhaaltje schrijven


WACC - Stalker

LinkedIn

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ratione hic id non rerum iusto assumenda sunt, ullam nostrum recusandae reprehenderit eos minus sapiente facilis velit perferendis, nemo sed laudantium officiis!

(a) The view when the user is not logged into LinkedIn.



(b) The view when the user is logged into Facebook.

Figure 1.2: Screen shots of the application when b the user still has not yet logged into LinkedIn and a the user has logged in into Facebook.

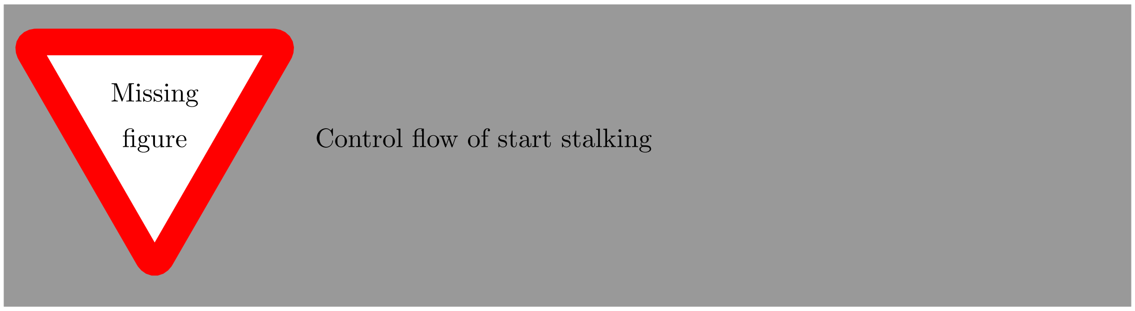


Figure 1.3: A schematic overview of the control flow when a presses the 'start stalking' button.

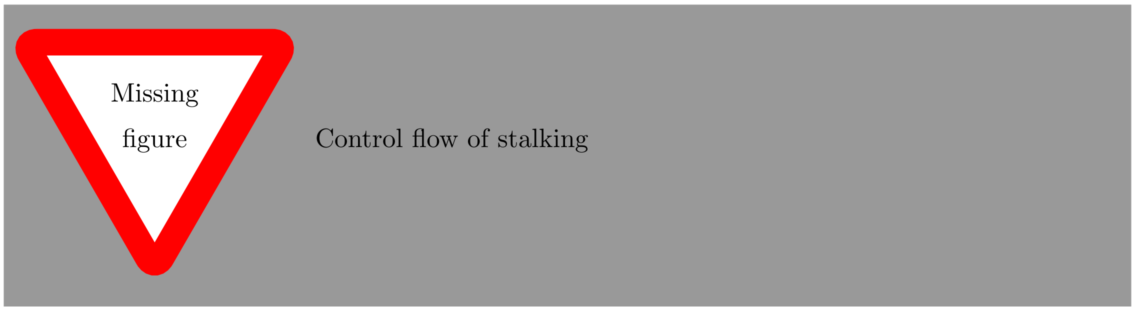
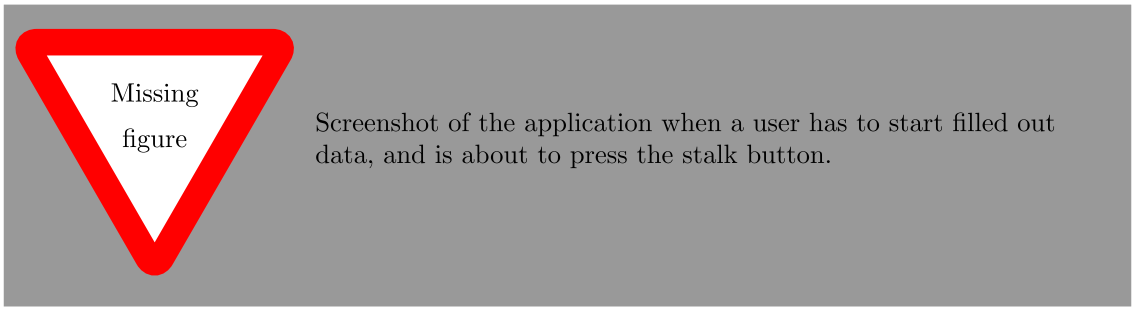
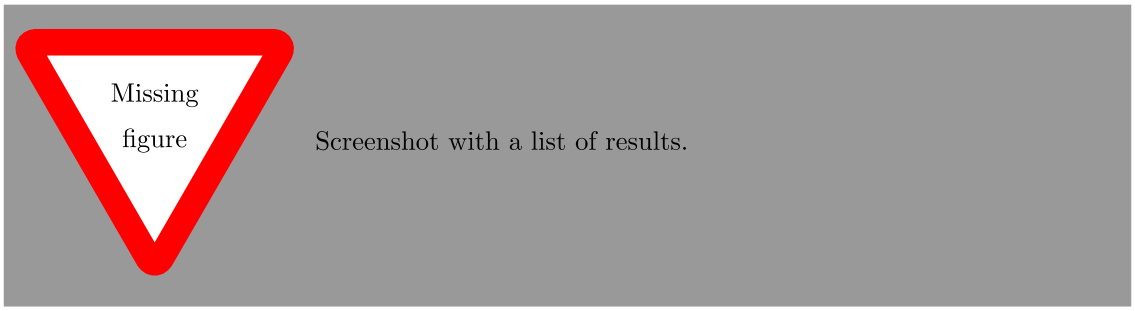


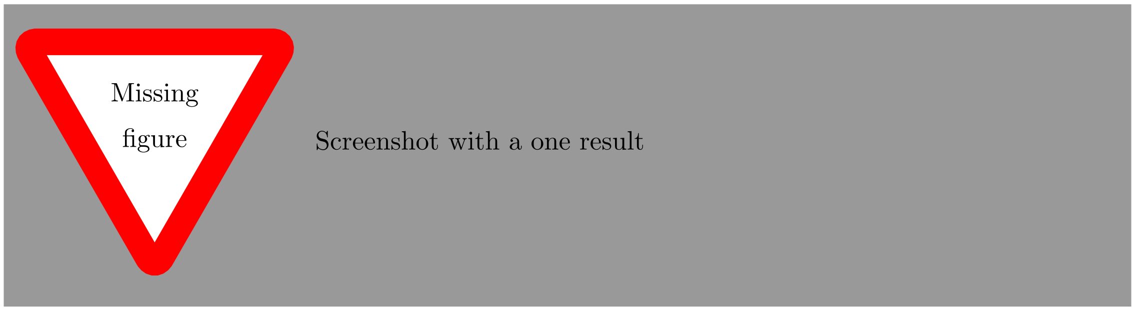
Figure 1.4: A schematic overview of the control flow when a user stalks somebody.


Missing figure
Screenshot of the application when a user has to start filled out data, and is about to press the stalk button.

(a) The view when the user is about to press the stalk button.



(b) The view when all victims are found.



(c) The view with the details of one victim.

Figure 1.5: Screen shots of the application when a the user is about to press the stalk button, b the list of all victims, c the details of one victim.

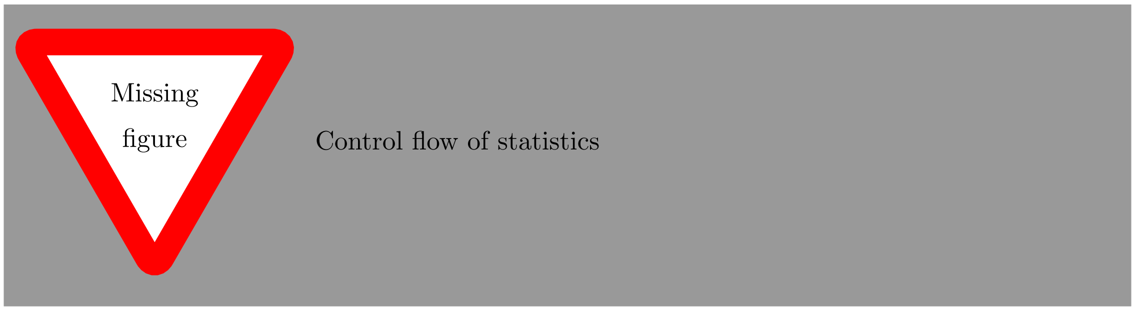


Figure 1.6: A schematic overview of the control flow of requesting the statistics and showing them to the user.

Figure 1.7: The view when the user has requested statistics.

Chapter 2

Back-End

This chapter discusses the back-end. This includes the REST API plus the database storing all the statistics data. We will first discuss the technologies that where used to build these component then how together these technologies and components form the back-end.

1 Technology Stack

In this section we will describes as in section 1 what technologies where used and why we decided to used these above others.

1.1 Flask

Flask [3] is a python micro framework for the web. Flask is based on Werkzeug [12] and Jinja2 [7]. We have used the Flask framework as a base, but mostly used it's extensions described below. We choose Flask for our back-end because one of us already had some experience with Flask and both of us had worked (a little bit) with python before.

Extensions

To make our development easier we decided to use already existing solutions for building a REST API using Flask and MongoDb (subsection 1.2). In this section we will describe the used extensions.

Flask-Restful s an extension for Flask that adds support for quickly build- ing REST APIs. It is a lightweight abstraction that can work together with existing ORM/libraries. Because Flask-RESTful [5] encourages best practices with minimal setup we were able to build a qualitatively good REST with reasonable ease.

Flask-MongoKit is and extension that builds on the Python Pymongo extension [11] and Flask. MongoKit adds an easy way to define MongoDb database models/documents [4].

1.2 MongoDB

MongoDB (from"humongous") is an open-source, document based, NoSQL database with a lot of cool features, we list a few of these that were important for the choice for MongoDB over other NoSQL database solutions [8].

Replication & High Availability A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. Because of this feature and the requirement for

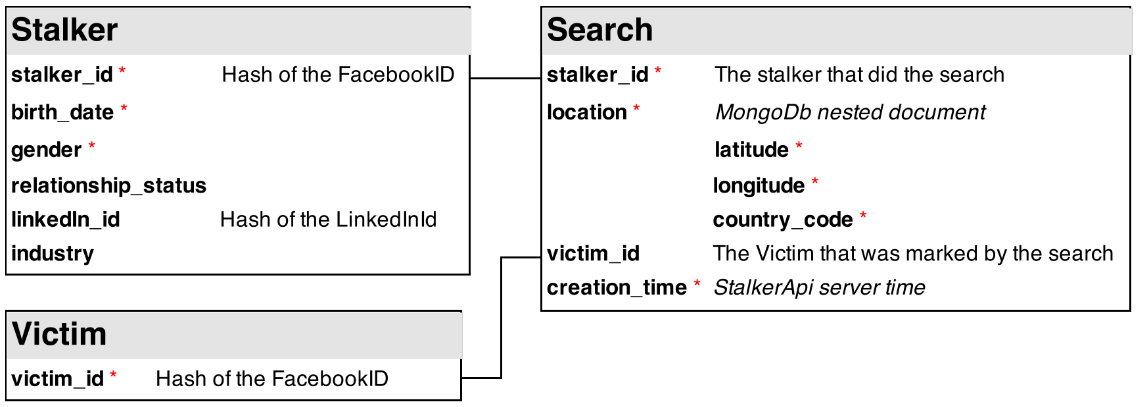


Figure 2.1: A overview of the database models. The attributes that are required and should be always available are marked with the red *. Not all information is available from the social media API's.

the project to be fault tolerant we found this to be a positive point for choosing MongoDB. The actual implementation of this feature in our project will be discussed in section 2.1.

Map/Reduce MongoDB provides an easy way to use map/reduce functionality. The map and reduce function can be written in JavaScript. In the Flask back-end it is very easy to apply these function to the database. This functionality was a requirement for the project and will be discussed in section 2.1.

1.3 HAProxy

From the website of HAProxy [6]. HAProxy is a free, very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for very high traffic web sites and powers quite a number of the world's most visited ones. Over the years it has become the de-facto standard opensource load balancer, is now shipped with most mainstream Linux distributions, and is often deployed by default in cloud platforms.

HAProxy is a load balancing solution pointed out by one of the student assistants and we chose to use this because it is according to their website the default solution for any professional cloud platform and beside fairly easy to set up. We will discuss HAProxy further in section 2.2.

2 Design

This section describes the design of the back-end of the STALKER project.

2.1 Database

Because API calls to the social media API's are done in the frontend (see section 2) the back-end only has to worry about the statistics. We will first describe the database models. Below a list of descriptions of the used terminology is provided. In figure 2.1 we give a more detailed overview of the data models.

Stalker A person who uses the STALKER application to search other people.

Victim A person who is being searches and is marked by the Stalker as the person he/she wanted to find.

Search This stores the link between a stalker, search and a victim.

The statistics are generated with the map reduce methods described in section 2.1.

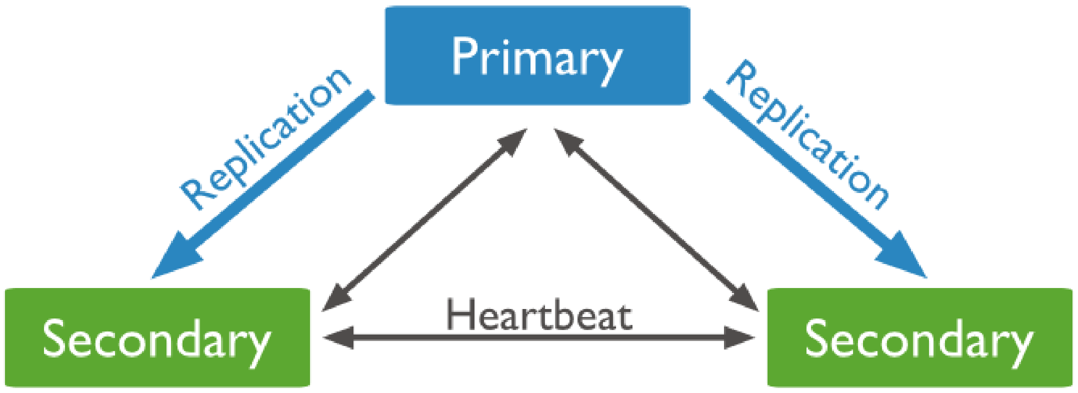


Figure 2.2: Replication implementation MongoDB

Map Reduce

Map-reduce is a paradigm invented by google and is inspired by the functional programming functions map and reduce. The map-reduce method thus consists of a map function which performs filtering and sorting of the data. The reduce method then performs a summary operations and in our case reduces the data to some nice statistic.

Example of map-reduce in our project. When the front-end requests a statistic e.g. the number of searches done per location. The map function will filter the data stored in the collection that stores the searches and will emit per search every location with a count of one. The reduce function will then reduce this list of locations into a list of unique locations. The reduce function does this by merging (taking the sum of) the counts when encountering double locations, this has as result a list of unique location and the number of searches done from those locations. This statistic is then shown in the front-end see figure 1.7.

Replication

This section describes a way MongoDB handles replication, see the image shown in figure 2.2. All write calls from the REST API (see subsection 2.2) to the database are done to the primary MongoDb instance. This data is then replicated to the secondary instances (That run on different machines). Read operation can be load balanced to any of the instances.

All instances know of the other instances and send heartbeat message to each other to ensure this. Whenever the other detect the primary stopped running they decide which one of them becomes the new primary (the first secondary that receives a majority votes becomes the new primary). When you have three database instances you can handle one faulty instance, so if another one stops the communication between the database and REST Api is gone.

We chose for three databases because this number was according to the MongoDB website more than enough replication for most problems. Plus this number of processes was still manageable on our development environments.

2.2 REST API

This section describes the part of the back-end that implements all the server calls and is the part that does the communication with the database(s). We chose to build a REST Api because it is a stateless architecture which gives it good performance and scalability properties. It is very easy to run multiple instances of a REST Api, because it does not know anything about the previous requests and does not store a state.

Stalker REST API Specification		
Resource + methods	*Input*	*Output*
/searches		
GET		Returns: A list of JSON Search objects
POST	Input: JSON Search object*	Returns: The id of the created Search object
/search/<string:id>		
PUT	Input: JSON object with victim_id and in the url a search id	Info: Updates a search object
/stalkers		
GET		Returns: A list of JSON Stalker objects
POST	Input: JSON Stalker object*	
/victims		
GET		Returns: A list of JSON victim objects
POST	Input: JSON vicim object*	
/statistics/location/frequency		
GET	Info: Counts searches per locationa	Returns: A list of JSON location frequency objects
/statistics/relationship/frequency		
GET	Info: Counts the relationships of stalkers who search	Returns: A list of JSON relationship frequency objects
/statistics/gender/relationship/frequency/<string:gender>		
GET	Input: Male/Female	Returns: A list of JSON gender relationship objects
/statistics/gender/location/frequency		
GET	Info: Males/Females followed by a list of locations	Returns: A list of JSON gender location objects

Load balancing

The REST Api can be run on different machine and load balanced using HAProxy. This proces is best described using an image, this image is included in figure 2.3. HAProxy knows about all the api servers and when it gets a request for the REST service it will send the request to one of the three (in our case) REST services running.

If one of the REST service stops this is not a problem for HAProxy and it will keep sending the requests to the other machines. Because the services are REST services it doesn't matter if first a create POST for a search was done on server x and the update of that same search was done to another server y. This illustrates the power of the stateless web services.

2.3 Overview

When we combine all the back-end elements. We get an image like the image in figure 2.4

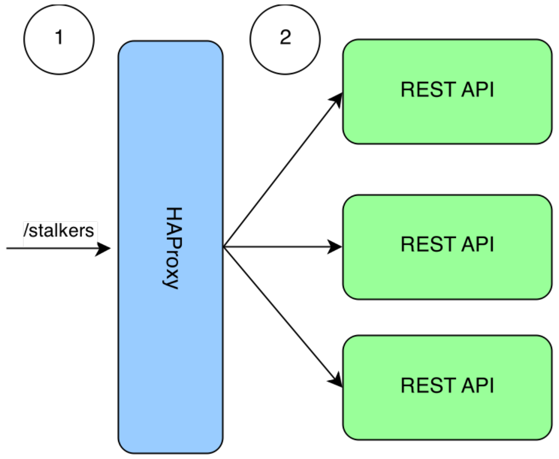


Figure 2.3: This image shows the structure of what HAProxy does.

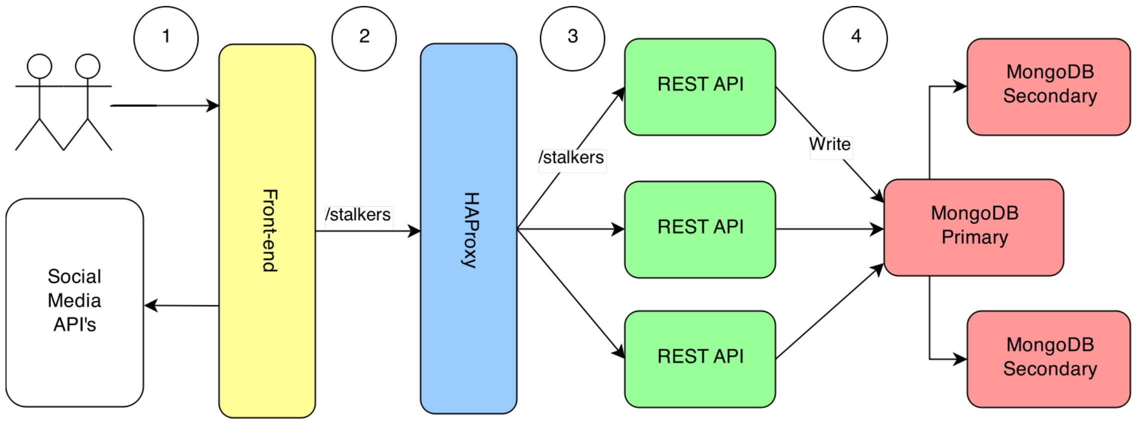


Figure 2.4: A schematic overview of the control flow when a user logs in.

Chapter 3

Evaluation

1 Frontend

Integrate searching and logging in into social networks.

2 Backend

- Security
- Check on data → idempotent post actions

Bibliography

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

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

[3] *Flask*. URL: `http://flask.pocoo.org`.

[4] *Flask-MongoKit*. URL: `https://pythonhosted.org/Flask-MongoKit/`.

[5] *Flask-RESTful*. URL: `http://flask-restful.readthedocs.org/en/latest/`.

[6] *HAProxy*. URL: `http://www.haproxy.org`.

[7] *Jinja2*. URL: `http://jinja.pocoo.org/docs/dev/`.

[8] *MongoDb*. URL: `http://www.mongodb.org`.

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

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

[11] *PyMongo*. URL: `http://api.mongodb.org/python/current/`.

[12] *Werkzeug*. URL: `http://werkzeug.pocoo.org`.