Vrije Universiteit Brussel

Faculty of Science and Bio-engineering

Project report Web Technologies

Celebrity Stalker

2015

Arnau Dillen

Toon Riem

Lei Jin

23/12/2015

Table of Contents

TABLE OF CONTENTS **2**

INTRODUCTION **3**

ARCHITECTURE **4**

TECHNOLOGIES **6**

FUNCTIONS **9**

- HOME PAGE 9
- REGISTERING AND LOGGING-IN/-OUT 9
- USER PROFILE 10
- MAP OPERATIONS 11
- CELEBRITY PROFILE 13

CONCLUSION **14**

Introduction

This document will describe the web application Celebrity Stalker and it’s functionality in depth .

Let’s start with a short description of the application first.

This application is used to track celebrity’s whereabouts by giving users the possibility to add a marker on a map when they spot a celebrity. If you have spotted some famous person at some location, you can click on this location on the map that is under the map section on the site. You can then give some information about this person and the location where you spotted him/her. This will add a spot event to the database that other users can see. If a user wants to see where celebrities have been spotted around a certain location, they can see this on the same map. When the user clicks on one of these markers they will be shown an info window with the information about this spot event. There is also an option to see a list of celebrities that have been spotted around the users current location, with the closest one first. If the user wants more information about a spotted celebrity they can click on their name in this list to go to their profile. This celebrity profile will show a picture of that celebrity and give some information about them through the summary paragraph from their Wikipedia page that is fetched on the profile page.

The rest of this report will be structured as follows.

In a first part an overview of the architecture of the web application will be wholly presented.

Then the technologies used for this application will be summarized.

Following this the full functionality will be thoroughly discussed. For every important functional aspect in the application a detailed explanation will be given about the functionality and its implementation.

Finally a short conclusion will be given about the project and the application. More specific, the goals that have been reached or not will be reviewed in detail.

Architecture

The architecture for this web app is pretty simple since this application is not a very extensive one.

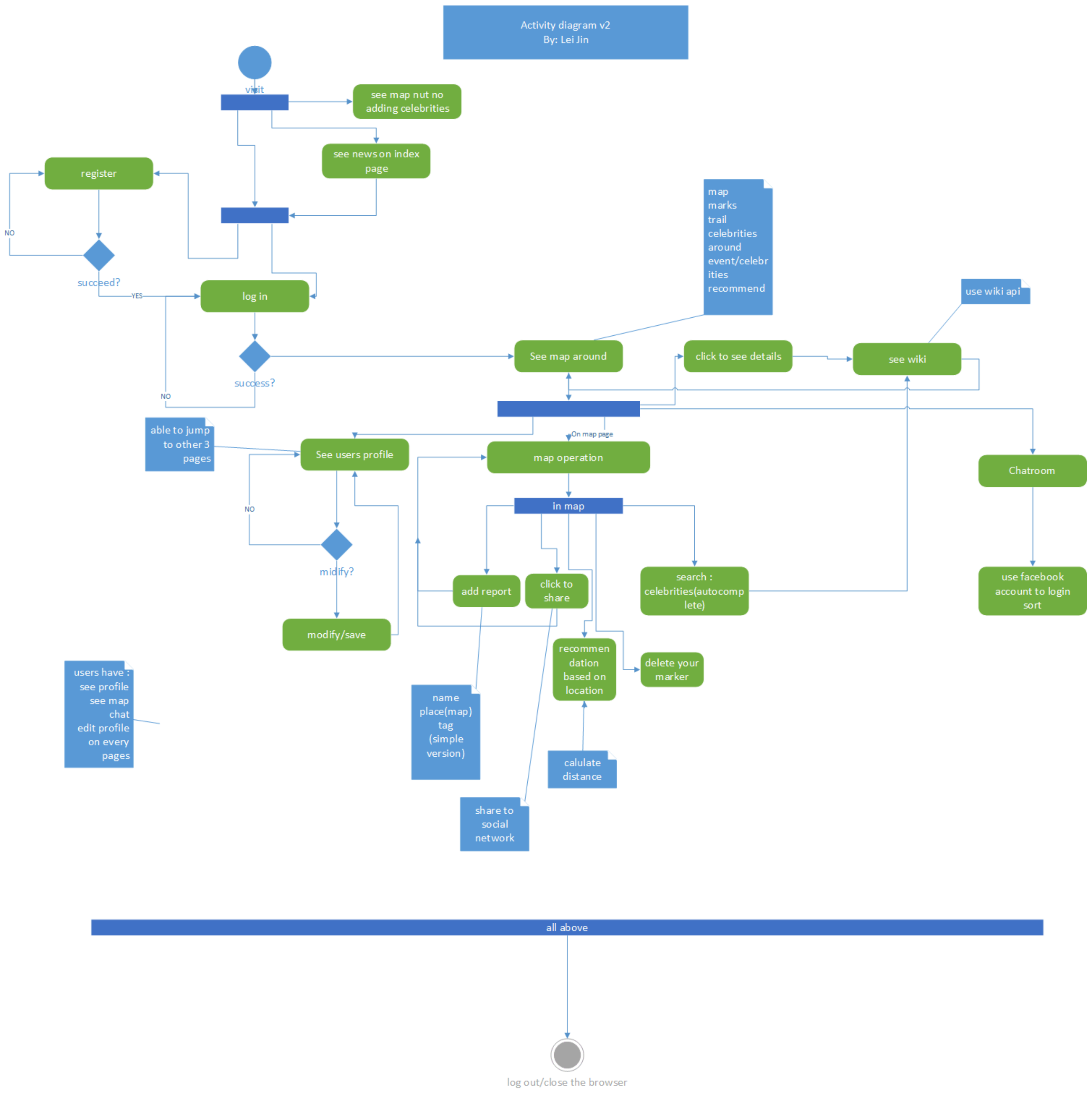
The code that is needed to run on the server can be found in a single main.js file. This file contains all the code that is needed for routing and using the database that the application uses. The reason this code is pretty simple is because most tasks get taken care of by the framework. This framework is Node.js together with Express. Express does the work of initializing our server for us, only needing simple input.

On the database side we used SQLite to manage a simple database containing tables for managing users and spot events on the map. The functions used for managing this database are also found in the main.js file. This is also where SQLite is initialized and the reference to the database is kept.

The views for the different pages are managed through the Handlebars templating engine. This allows us to have a main view that contains the code that is common for all the pages. Only page specific scripts and mark-up need to be specified in the pages file. Required scripts are already loaded for every page this way.

We also make use of the body parser module for Node.js to be able to support JSON encoded bodies and URL encoded bodies. Different bodies can also be supported with this. Also included is a cookie parser module to be able to make use of cookies on the website.

The following activity diagram illustrates this architecture:



Activity diagram v2
By: Lei Jin
visit
see map nut no adding celebrities
see news on index page
register
NO
succeed?
YES
log in
success?
NO
map marks trail celebrities around event/celebrities recommend
use wiki api
See map around
click to see details
see wiki
able to jump to other 3 pages
See users profile
On map page
map operation
Chatroom
in map
NO
midify?
add report
click to share
search : celebrities(autocomplete)
use facebook account to login sort
modify/save
recommendation based on location
delete your marker
users have : see profile see map chat edit profile on every pages
name place(map) tag (simple version)
calulate distance
share to social network
all above
log out/close the browser

Figure 1: Application activity diagram

Now we will go in more detail about the technologies employed in the application. Most have already been mentioned, but will be presented in a more structured manner.

Technologies

Framework: Node.js + Express.js

Thanks to Node's modular nature, adding new function to our application was pretty easy. There are modules for almost everything you might need in your application, with a choice between several options for most. Together with Express Node becomes a very powerful web application framework. Like mentioned before Express takes care of most setup and management tasks for your application. Because of this, we only needed to call on the setup functions for express and provide some simple functions for routing. Routing was very simple, some pages only needing to be sent the render command. It surely is a recommendation for web developers working with JavaScript on the server side.

Node: https://nodejs.org/en/

Express: http://expressjs.com/en/index.html

Database: SQLite

As mentioned in the previous paragraph, Node.js lets you easily get any resources you might need for your application. This is also true for SQLite, it can be simply incorporated in any project through the npm command Node.js provides. SQLite itself wasn't the simplest module to use though**.** It is an alright database engine, but is not very easy to use. Especially for the sqlite3 module we used, which isn't very extensively documented and can be confusing at times.

Sqlite3: https://www.npmjs.com/package/sqlite3

Templating engine: Handlebars

We made use of the Handlebars templating engine to easily manage our web-page elements. This engine gives us the possibility to specify a main view that is common across all the pages, which is useful to include the same nav bar on every page for example. Thanks to this all the scripts which are not page specific can also be included once in our main view. It also lets us define and use handlebars expression which come in very handy when having to pass a variable from the server to the rendered page, especially for structured data. It also can check if a user is logged-in (with {{#if user}}) so this came in very handy.

We would definitely recommend using a templating engine for web applications.

Handlebars: http://handlebarsjs.com/

We used the express variant: https://github.com/ericf/express-handlebars

Styling: Bootstrap

For the layout and styling of our website we made use of the Bootstrap framework. This would be a must have for any website that wants to be responsive, since Bootstrap takes a mobile first approach thanks to the viewport concept. Designing your site is very easy with this, you don't have to write lots of CSS code to style your pages. Many different theme's, styling functionality's, components (like nav bars) and JavaScript functions are available. When making our site we just needed to choose a theme and add the components we wanted where necessary. All these different functionality's are also very well documented and easy to use. Any styling you might want for a page can be quickly implemented with this.

Bootstrap: http://getbootstrap.com/

Authentication: Passport.js

To authenticate our users we made use of the Passport.js middleware. It provides us with many different options for authentication strategies and is relatively simple to use. We only made use of the local authentication strategy that checks a login attempt against the user database. But if we want to implement different strategies, it is only a matter of including the required passport requirement. This is a very useful module when you want to have many different ways for users to authenticate themselves.

Passport.js: http://passportjs.org/

JavaScript library: jQuery

This is an obvious choice for any web application. The jQuery library provides an extensive, easy to use API for JavaScript on the browser side. With jQuery we can easily access and manipulate the different DOM elements on a page. But most useful is the ajax API that lets you easily make requests and handle the retrieved content. The library also provides lots of useful plugins for more specific functionalities.

jQuery: http://jquery.com/

Social API's: Facebook and Twitter

To give our users the possibility to share content from our site on social media, we made use of the Facebook an Twitter SDK's. These give us the possibility to easily add a Facebook like and share, or tweet button anywhere on our site. These provide API's that are pretty easy to use and well documented, though it can sometimes be a pain to show one of these buttons. Facebook comments were also added to the site this way. There are also more functionalities with Facebook especially, but these could be easily added, since the SDK scripts have already been included.

Facebook SDK: https://developers.facebook.com/docs/javascript

Twitter SDK: https://dev.twitter.com/web/overview

Information API: Wikipedia

When a user wants to look at some celebrity's profile, the picture and information are fetched from their Wikipedia page. No information about celebrities is stored in our database. That way, when a celebrity is spotted in the application for the first time, no new entry about the celebrity needs to be added to the database. The information and picture are retrieved from Wikipedia and are always correct and up to date this way. This is not an easy to use API though as you need to know what you're looking for in the API.

Wikipedia API: https://www.mediawiki.org/wiki/API:Main_page

Map API: Google Maps

The map functionalities this web application uses are handled by the google maps API. All map operations, like spotting a celebrity and showing markers for spotted celebrities on the map, are done through this API. It's a very good API and is very well documented by google. The API offers lots of possibilities we didn't even use.

Google Maps API: https://developers.google.com/maps/web/

Functions

Home page

The home page is basically one page with different sections, like Home, Map and Profile. On this page we made use of some page-scrolling JavaScript-code. If an element in the main bar is clicked and it's also on the home page, it will go to that section in a nice scrolling animation. For this, we needed the jQuery Easing plugin.

For the registering-, log-in- and edit-screen we didn't make a different page, instead we used modals to show a little box where you can enter you data and verify it.

The page itself only contains a slideshow about featured celebrities and the welcome message, which is addressed to the user if they are logged in.

Registering and logging-in/-out

For the database, we used the sqlite3-middleware. This is basically simple SQLite-middleware that made making a database and adding new tables and entries really simple. When the server is started, it checks if a CSdatabase.db-file is present, and if this is the case it will use this one and otherwise it will make a new .db-file with two tables: map and users.

As explained above, we use modals as register- and log-in-screen. For registering, the user has to enter a username, email and two times a password. After checking on the client if everything is filled in and long enough, the information is send to the server. Here, it is checked with the database if the username and e-mail are still available, and if the two passwords match. If this is the case, the information is added to the database and the new user is successfully registered. Otherwise, the home screen is rendered again with a list of errors and here the modal is automatically shown again with the correct information filled in, and some errors behind the boxes in the form that caused the errors.

The logging-in is a little different. First of all, a user has to enter his username and password. After the basic checks on the client side like with registering, the information is sent to the server. Here, we made use of the Passport middleware. Different ways of authentication (like local, facebook, twitter, etc.) can be used to check if the user is logged in. We only use the local authentication to check with the database if the filled-in information is correct. If this is not the case, the home page will be loaded again with the sign-in-modal shown and an error displayed. Otherwise, the user will be logged in and a cookie will be saved in the session to keep track of the logged-in user. This will cause the {{#if user}}-test in the handlebars to be true, and the user-id or username can be read inside the handlebars with {{user.id}} and {{user.username}}. As soon as a user is logged in, there will be a profile- and log-out-button in the main menu. The log-out button also uses passport to end the user session.

At first it was a bit tricky to implement because of the asynchronous nature of Node.js. The database-operations wouldn't have finished before the rest of the function was completed, and this caused the log-in to redirect immediately to the failure-redirect. But then we got more used to the callback-function method where a callback was given to the function that will be called on the result of the database- search.

User profile

When a user is logged in, he can view his profile on the home-page. Here, he can see all his information and edit it. The edit button displays a modal, like with the registering and logging in. Here, the user can modify the parts he wants to change and leave the others blank. There's a check to see if certain fields of the form are long enough, but not every field is required like with the registering and logging in. The information is then sent to the server and if there was an e-mail or new password given, there will be a check to see if the e-mail is available and if the passwords match. In case of failure, the user will be redirected to the page with the edit-modal and the errors shown. Otherwise the information will be edited and the user can see his new information in the profile menu.

For editing we also had to consider the asynchronicity of Node.js, like with the callback-function of the logging-in.

Map operations

The core of Celebrity stalker focuses on the operations on the map.
We used the Google Map API to show the map and added a series of functions to it.
There are 2 modes:

1. Visitor: when people do not log in, they can just locate their positions, see celebrities reported by other people (blue stars stand for one report), search for celebrities, find nearest celebrities and check celebrities' profiles and share or like spot events that are posted to a social network like Facebook. However they have no authority to add or delete any information.

2. User mode: users who registered and logged in have all the rights that visitors have and they can add new reports and delete markers they added.

We used the basic Google Maps API and added different markers, info windows, distance calculation and a search box to it, so this map has plenty of functions to fulfil our requirement.
The techniques are as follows:

1. When a page starts to load, we use HTML5's new feature for geolocation to locate the user's position and use the latitude and longitude as the centre of the google map. The initial zoom of the Map is about 16 because it was found suitable for the application's needs.

2. When the map is loaded we use AJAX($.getJSON) to require the existing markers information from the Node.js server which will get the data from the database. It will return a JSON formatted object array. (because the data is small, we decided to load all the data from the database at once) We use a for-loop function to decode it and show every existing celebrity-markers on the map.

   The structure of the marker is as follows:

```
Marker[]=google.maps.Marker({
Position:
Map:
Icon:
Id:
Userid:
Title:
Animation:
From:
Distance:
Percentage:
})
```

Userid is used for recognizing the user who logged in from the other users.

3. The map and markers should have listeners in order to react to user’s behaviours. When a user clicks the marker there should be an info window containing the information on the presence of the celebrities(HTML codes for the info window). In order to be more user friendly, we made the map avoid showing two or more info windows at the same time.

4. Once the user logs in they can see their own markers(red stars) and add new markers by clicking the map and fill in a very simple form. Other people can see these markers. That is how user can view and create information. To make it more user friendly, once users click another place on the map, the existing info window should disappear. We added a lots of logical code to realize this goal and the goal in (3).

5. We use Userid and if-condition to judge whether the markers are made by the user himself; if yes, he/she can delete them and the marker will disappear on the map. That is how users can modify information.

6. Distributed search. Because of Node.js’s single process feature, we think that searching should be done by clients rather than servers. People can click the Search button to see the recommendations. We designed our recommendation system. It is based on location. It will always show you the nearest and top 10 celebrities around you and the distance between you and celebrities. When a user searches for a certain celebrity, there is an autocomplete function. Because we have got all the data from the database when the page is loaded, we can use client search instead of server search, for the purpose of releasing server stress and making browsers as distributed calculation part. In this method we only need to

use jQuery for auto - completion rather than AJAX. The list of the result is sorted by matching degree and distance.

7. When users get all the searching results they can click to see the profile of a certain celebrity; the information is provided by Wikipedia which means we use external web service for retrieving information. This is discussed in the next section. We also used the HTML5's <article></article> tag to organize the profile.

Celebrity Profile

When a certain celebrity has been spotted at least once, users can have a look at their profile page on the website. The particularity about these profiles is that they do not use any information from the database. The content on the page is retrieved through the Wikipedia API. The retrieved assets are the celebrity's profile picture and the introduction section from the Wikipedia article. These assets are acquired by making an AJAX call using jQuery's getJSON() function. The necessary data is then extracted from the JSON response and put in the appropriate field on the page. Using this technique saves us database space and diminishes the load on our server.

The celebrity's Facebook page was also going to be embedded in the page using the Facebook SDK, but this didn't get included due to a lack of time to do this neatly. It is only a matter of parsing the celebrity's name and including the code you get from the Facebook documentation.

In essence, when the user clicks on a celebrity's name, their full name is passed to the page through a Handlebars variable. The script on this page then takes care of the rest for us.

Conclusion

We didn’t get to making the most extensive web application for this project, but where able to put in most of everything we wanted to make. This means the application doesn’t have that much functionality, but does what it needs to do. Developing for the web wasn’t the easiest task for us, since none of us had any experience working with these technologies. Our knowledge had to come from what we learned in the lessons and exercises, or from what we could find on the web. We managed to make a working application though and where lucky in our choices of technologies, since none of us had any prior knowledge about them. All the requirements for the projects should have been met under the following forms.

Functional requirements met by our website:

- Users can register, log in, log out. Validation Check.
- After logging in, users can see and edit his/her profiles.
- On the map users can see celebrity markers generated by himself or others, when they click a certain marker, info windows show the details. They can add create new markers and fill a simple table to generate a new celebrity report when they are not in the system. Users can also delete the markers they created.
- Both visitors and logged-in users can search for a certain celebrity here, we designed our own algorithm to optimize the results.
- To make it more social, we did plenty of work.
  - On the map, users can see which user reported it.
  - User can “like” it and share the report to other social networks
  - There is a chat room(comment) page, users can chat there which give them a place to share their feelings and news.
- Use recommendation system based on the location. When users search or just click “search“ button, they will see 10 nearest hot celebrities sorted by matching degree and distance.
- Wikipedia is used on the profile of celebrities.

Technical Requirements met:

- AJAX: It is used in sending data to the server and getting data about information on celebrities from the server in the background.
- Form validation: It is used in the register and edit user profile parts.
- CSS: It is used in every page through bootstrap and inline styles.
- HTML5: Geolocation for the map; some new tags like <article></article> for showing celebrity profiles; local storage
- Web service: Use Wikipedia to provide the celebrities' profiles
- Published content: Users can create and delete celebrities' markers on the map.
- Mobility: Adapt to the mobile phone screen well. UI changed when the screen is smaller than a particular size.
- Google maps: Using Google Maps API the crucial operation is on the map.

Finally we hope you enjoy our Celebrity Stalker and spot lots of different celebrities for us.