Traffic Alerts

Project for the course Design of Dynamic Web Systems 2014

by

Martin Bumba and Sara Fatih

Project repository: https://github.com/mbumba/traffic-alerts

Demo: http://cloud-25.skelabb.ltu.se

API example code: https://github.com/mbumba/traffic-alerts/tree/master/api-example

API example demo: http://cloud-25.skelabb.ltu.se/api-example/

Introduction

Usually when we take a road, we are not notified about the problems that may stop us from continuing in that road. That's when the idea of our project came. Actually, when a user gets to our application's website, they can take pictures of different locations on the road when there is a traffic jam, work (construction, reparation…), a natural disaster, or anything that might stop a driver from continuing in that road. The pictures are taken by the application with GPS coordinates and a note describing the picture. Then, it is uploaded to the server side and in real time distributed to the clients those are connected to the application. All the alerts are represented in a Map and displayed for the clients. The main goal behind this application is to raise awareness about road problems among the users. That way, drivers can avoid going through some path just by consulting this application.

How the system can be used

When the user types the URL of our application to his web browser, they directed to a page that displays a button to login via Facebook. When the user is logged successfully, they are asked to approve to share their camera and their actual position(geolocation). The map is then displayed along with a menu bar where the user can: refresh the map, publish a new alert or log out. If the user click on "New Alert", a window shows up, where the user can take a picture since the application accesses the camera stream. The user can either do that or browse for a photo in their computer or smartphone. The position of the alert can be either specified via position sensor(GPS) by clicking on GPS, or the user can specify position by clicking on map. The user then sets the expiration date of the alert, the icon (on of four icons) and a note to describe the alert. The user then clicks on "Publish alert" or "close" to close window.

When the new alert is published, a message is displayed stating that a new alert is published, they can click on that message to see information about that alert, and the marker is added to the map.

In addition to this, the user can navigate on the map and see the markers of the other alerts. When the user clicks on a marker, they can see information about that alert: publication date, expiration date, the picture, the note, and the percentage representing the progress of time since the publication of the alert, compared to the expiration date. The user can like the alert by clicking on the button "Like". If the connected user is the one who published that alert, then they can remove it from the map by clicking on the marker of that alert and clicking on Remove.

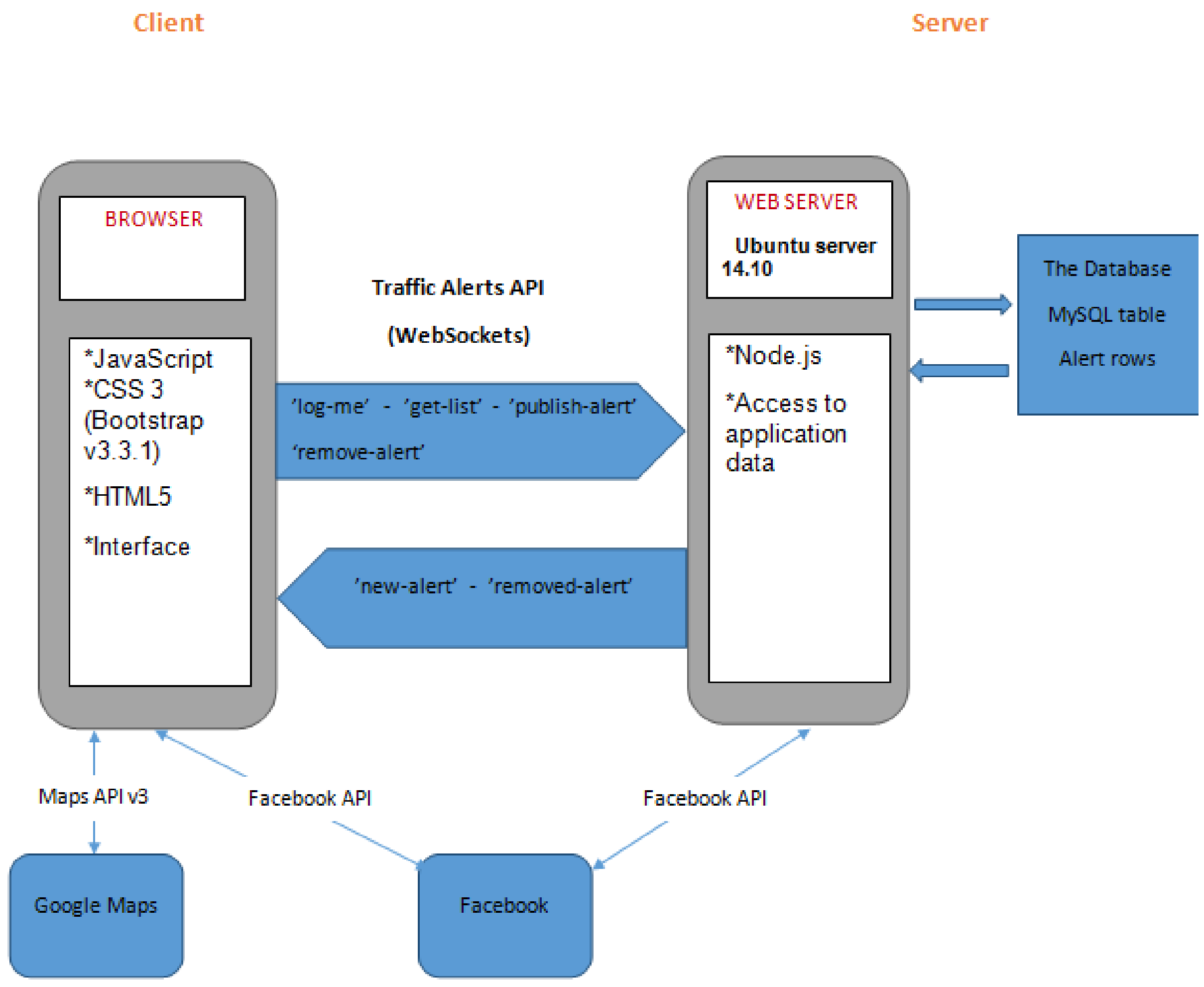
The user can as well refresh the map, and logout via the menu bar on top of the page.

The design and architecture

Our web application is made from a server side, a client side and communication *API*. The web browser shows our client side application's interface to the clients. We used *HTML5, CSS3*, and *JavaScript*, as well as *Bootstrap* for many features, among which the responsive feature of the application.

The server side is running on *Ubuntu server 14.10* and we used *Node.js* as a platform for it.

We used *WebSockets* as a way of communication between the server and the client side. Our propossed *API* is specially based on *Socket.IO* library. We specified the commands "log-me", "get-list", "publish-alert" and "remove-alert" are sent from the client side to the server side while the two commands "new-alert" and "removed-alert" can be sent from the server side to the client side. Every command is explained below in the *API part*.



The API of the application (Traffic Alerts API)

In our application we are using real-time communication for the purpose of publishing new alert and notifying connected clients about alerts in real-time. For this reason we based our API on *WebSockets*, more specifically on *Socket.IO* library which simplifies work with *WebSockets*. In the following tables, are defined all allowed commands which can be emitted in both directions as well as their parameters for every expression.

For using this API to communicate with our server you need at first to define the socket variable, as such:

```
var socket = new io("http://cloud-25.skelabb.ltu.se:80");
```

Client → Server:

Command can be easily emitted to server like:

```
socket.emit(command,[parameter1,parameter2,...]);
```

Socket command	Parameters	Parameters of callback function
"log-me"	*facebookUserToken: valid Facebook user token for some FB application *callback(result,error)	*result: true or null *error: message or null
"get-list"	*callback(result,error)	*result: array of (object) alert or null *error: (string) message or null
"publish-alert"	*image - (string) Base64 encoded JPEG image data *lat: (float) latitude *lng: (float) longitude *icon: (int between 1-4) image type *note: (string) text for note *expires : (Date) expires date *callback(result,error)	*result: (object) alert or null *error: (string) message or null

"remove-alert"	*id: (int) id of alert *callback(result,error)	*result: id of the alert that the user wants to delete *error: (string) message or null

Server → Client:

Command from server can be on client side received like:

```
socket.on(command, callback_function(parameter));
```

Socket command	Parameters
"new-alert"	*row: the entire row of the (object) alert
"removed-alert"	*id: the id of the (object) alert that was removed

Simple example how to use *API* has been published in *Github* repository. Use it to learn how to use this *API* and what are the requirements:
https://github.com/mbumba/traffic-alerts/tree/master/api-example

Used technologies

Node.js: a new platform

We chose **Node.js** as a platform for the server-side because it is a technology we have never used before. Therefore, working with it would be a practical learning experience. It uses JavaScript as well, which makes it easier for us since we are already familiar with it.

HTML5: using the camera API and WebSockets

Concerning the client-side, we have chosen *HTML5* for many reasons. First, it allows us to create communication between the client and the page through WebSockets. Also for multimedia purposes. It allowed us to use, manipulate, and store an image from the computer's camera as well as a mobile device.

Twitter Bootstrap: Responsive features (CSS3)

Bootstrap was also a technology that we made use of, because it helps us make our application responsive since it will be mainly used in mobiles and tablets.

Google Maps JavaScript API v3

We used this *API* as a basis for the main map of our application.

Facebook SDK

We also used this to enable users to connect via Facebook as well as liking alerts.

HTML5-technologies we have used

WebSockets

We made use of the *WebSockets* as a way to transfer data between the client and the server side in real-time. Our designed API for this communication is based specially on *Socket.IO* library. API is described in section above.

Video element

For the showing data from camera directly in web browser we are using new *HTML5* element <video>.

Local Storage

This data persists even after you navigate away from the web site, close your browser tab, exit your browser. We used *LocalStorage* to store the last longitude and latitude of the client's geolocation. Because in next user session we can directly center map to this location and then wait on current location from web browser.

FileReader

The *FileReader* interface provides a number of methods that can be used to read either File objects. These methods are all asynchronous which means that your program will not stall whilst a file is being read. We used the *FileReader API* for uploaded pictures of alerts. In case the user doesn't make use of the camera, they can upload pictures from their computer or from their phone.

getUserMedia

We used this *API* as a method to access external device data: the camera video stream to take the picture of alerts.

Geolocation

We used geolocation purpose of *HTML5* for showing user location on map, for centering map to it's location and for option how to easy get current location for new alert.

Software parts and their interactions

The client-side & the server-side description and interactions

The first page that is shown in the browser for the user is the Facebook login page. When the user clicks on "Login" application give control to *Facebook SDK* for purpose of login user.

The *Facebook SDK* provided us with client-side functionalities such as enabling to use Facebook Login to make it easier for users to sign up on the site. This, as well as enabling the social plugins: like and share button.

After the user is successfully logged in, they can see the navigation bar up to the right which contains the “Refresh Map”, “New alert”, and “Logout” buttons.

The main client-side code is written in both the *HTML* file *index.html* where are specified main *HTML* elements and JavaScripts imports and the main JavaScript file *app.js* which is under the public file.

Through the JavaScript, when a user is successfully authenticated by Facebook, the server connection is established. Afterwards, the "log-me" command is emitted with the Facebook user access token. After successful authentication on server side is called another function for initializing the components of the page. This function sets the style of the html components of the second page that the user is supposed to see after they are logged through Facebook and our server. In the process is also loaded the map is loaded.

When the user clicks on the “new Alert” button, the model for the new alert is opened. This model contains the elements for taking a picture (form-take-photo), selecting files for a picture(for uploading pictures in case the camera doesn’t work, or if the user has already a picture stored), setting the expiration date, the note that describes the alert, as well as the icon. The user can publish the alert afterwards. The command "publish-alert" is emitted to the server side, and the alert is added to the database.

After the alert is added, the callback function is sent to the user who published the alert with the row of the alert that was added, and the command "new-alert" is sent to the other clients. In addition to this, the alert is added to the database.

The user has the choice to delete an alert that has already been published by them. Actually, there is a small pencil near the alerts that were published by the same user that is connected. This allows a user to recognize their own alerts in case they want to delete them. When the user clicks on “Delete”, the callback function is sent to the user who deleted the alert with the row of the alert that was deleted, and the command "removed-alert" is sent to the other clients.

The files structure

The our server’s configuration file is written in the *config.js* file under traffic-alerts root folder. The server JavaScript code is written in the *app.js* file inside the same folder. While the client JavaScript code is in a file with the same name (*app.js*) under *traffic-alerts/public/js/app.js.*

Basically, our file structure is as follows: the server side files are under traffic-alerts folder, they are: *config.js, functions.js, app.js*, and *traffic-alerts.sql*. While the client side files and folders are under: *traffic-alerts/public*. The folders in this case are classified as *css, fonts, images, js*, and *uploads*. And the main html file is in the public folder, and it is called *index.html.*

Security considerations in our application

In our application we are authenticating users by passing *FB user access token* to server. It means user, who wants to use this application is first of all invited to login to *FB* and then he has to agree that the application will use basic information about him. After that, the user is successfully authorized by *FB* and *FB* generates user access token.This access token is then emitted to our application server and checked there. If it is valid, the user is also authenticated on server and in this point, the application can emit commands to server.

In this project we also focused on solving *OWASP TOP 10*. It is a list of the 10 Most Critical Web Application Security Risks.

A1 - Injection

Injection in this application can happen only in the server side where it is communicating with *MySQL*. For communication with *MySQL* we are using the node-mysql library, which automatically performs escaping in our usage. Also we are validating every variable which is going to server.

A2 – Broken Authentication and Session Management

This one can be a problem of our application in this moment, because we are using *HTTP* (unsecured) and *WebSockets* (unsecured). So anybody on network path can steal for example *FB user access token* when we transfer that by *WebSockets* to server. So the solution of this problem is to change these protocols to their secured versions. *HTTP* to *HTTPS* and *WebSockets* to *WebSockets Secured* (*WSS*). This will prevent this problem, because every data travelling from client to server and reversely will be encrypted. We are not using these secured versions, because verified certificate for this purposes can costs a lot of money.

A3 – Cross-Site Scripting (XSS)

We solved the problem of XSS, because we are validating and escaping every user input variable. For example for note in traffic alert we are replacing "<, >, &" and " with HTML entities before we save it to the database. We are using for validating and escaping very useful library *validator.js* which is same for both client and server side.

A4 - Insecure Direct Object References

We solved this problem, because we are not using direct object references. I.e. when we remove traffic alert by parameter id we check if user and *Facebook APP* who wants to remove this alert with specific id are owners of this alert. If not, the removing of this alert is forbidden.

A5 - Security Misconfiguration

We used most recent versions of all used libraries and applications and also we are updating Ubuntu server. For *MySQL* we made special user, which can manage only database *traffic_alerts* and can use only commands: SELECT, UPDATE and DELETE.

A6 - Sensitive Data Exposure

We are not storing and exposing sensitive data like passwords or credit cards numbers. We are only storing in database insensitive parameters such as *FB user id* who published an alert. And even this parameter we not passing straight from server to client. Instead of *FB user id* we are passing in this communication *boolean* if connected user is owner of the alert or not.

A7 - Missing Function Level Access Control

In this time we are not using administration for managing traffic alerts. But if we will use it in future we will focus on this problem too.

A8 - Cross-Site Request Forgery (CSRF)

This problem cannot easily happen in this application. Because our application is not based on *HTTP* requests. So somebody who wants to use our server side, needs to use our *TrafficAlerts API* and in that have to at first authenticate user by emitting command with *FB user access token* (generated for some *FB APP*). And after successfully authentication he can emit commands which can influence only alerts which created same *FB APP* for which was token generated.

A9 - Using Components with Known Vulnerabilities

We used for this application only known libraries which are checked by a lot of users around the world and we also used the most recent versions of these libraries which were available when we developed this application.

A10 - Unvalidated Redirects and Forwards

We prevented this risk, because we are not using redirects.

References to the material we have used

We used many libraries as the base for our application.

Client-side used libraries

Bootstrap v3.3.0

Bootstrap is *CSS* and *JS* library and it helps us make our application responsive since it will be mainly used in mobiles and tablets.

Available on: http://getbootstrap.com/

jQuery

Because Bootstrap JavaScripts and some other libraries used in client side are based on jQuery, we also import this library to our project.
Available on: http://jquery.com/

DateTime Picker

We used this for the alert's datetime that the users specify when they want to publish a new alert. We chose this because it is easier for users to select the day and time with the datetime picker.
Available on: http://www.malot.fr/bootstrap-datetimepicker/

Bootstrap - File Input

We used this for better design of input file field, because *Bootstrap* doesn't improve this field.
Available on: http://gregpike.net/demos/bootstrap-file-input/demo.html

Moment.JS

Moment.js is a free and open source *JS* library for simplify work with date and time in *JS*. The library is a wrapper for the Date object. We used it for the formatting date and time for published and expired properties of alerts.
Available on: http://momentjs.com/

Facebook SDK for JavaScript

We used this for communicating with Facebook. To allow clients to login via Facebook and Like alerts.
Available on: https://developers.facebook.com/docs/javascript?locale=cs_CZ

Google Maps JavaScript API v3:

Used for the main map of our application and it's functions like map infobox (window).
Available on: https://developers.google.com/maps/documentation/javascript/

Both client and server libraries

Socket.IO

We made use of this library for simplified work with *WebSockets*
Available on: http://socket.io/

validator.js

We used this library for its functionalities to validate values. We used for example *validator.isFloat()* and *validator.isIn()* in addition to other ones. It was to check and validate user input values in our application on both server and client side.

Available on: https://github.com/chriso/validator.js

Server side libraries

Node.js

We used *Node.js* as a platform for our server side.
Available on: http://nodejs.org/

express

We used express as web application framework for Node.js that provides a robust set of features for web and mobile applications.
Available on: http://expressjs.com/

image-type

We used image-type to detect the image type of a Buffer/Uint8Array to check whether the type of a picture is jpeg.
Available on: https://github.com/sindresorhus/image-type

node-cron

We used node-cron to check expired alerts every minute. When the present time surpasses the expiry date of an alert, then it is removed from the server.
Available on: http://github.com/ncb000gt/node-cron.git

NodeJS Library for Facebook

We are using communication with Facebook also on server side. Here we are checking with this library whether FB user access token incomming in command *log-in* is valid.
Available on: https://github.com/Thuzi/facebook-node-sdk/

node-mysql

This library is node.js driver for MySQL.We are using that for communicating with MySQL server.
Available on: https://github.com/felixge/node-mysql

Discussion and evaluation

We encountered many problems during the development of our application.

The application actually works fine in *Firefox* and *Chrome* on the computer as well as the mobile phone, however the camera doesn’t work on the Iphone’s browser *Safari*. Therefore, we decided to use also normal file input. Users can choose image in their device and then this picture is processed by *FileReader.*

We also faced a lot issues with the *DateTime picker*. At first, we used a library but it didn't work in *Safari*. Then, we tried another one, but we needed to add the cascading sheet. Eventually, we used the *Bootstrap datetimepicker*, which works well now.

Also, when a user likes an alert's picture by clicking on the Facebook Like button, there is a large window that shows up, which we believe is useless. Therefore, we cropped the div that shows it.

Another issue is that the users of our application cannot share an alert with Facebook, they can only like it. Because in our application's *URLs*, because we are changing only hash part of *URL* (behind the character "#") which doesn't let us share alerts.

We implemented the history feature in our application. When a user clicks on an alert on the map, then clicks on another one, and another one, then if they click on the "previous" button of the browser, the previous alert will show up. Therefore, we implemented the history feature only for alerts as we believe it is not necessary for something else than alerts.

We also used a progress bar for each alert. The progress bar shows the progress of time from the publishing date of the alert until its expiration date.

We encountered a problem with the core of browsers. The scrolling bar of the window of the alert doesn't show up on mobile devices as well as on *Safari*.

Installation instructions

Check the *Github* page: https://github.com/mbumba/traffic-alerts for the installation instructions.

Future work

If a developer want to continue working on our project, they will need to learn many things about our application. They should read the application's API that is in the table we created, as well as the documentation to understand the whole concept of Traffic Alerts.

Many additions can be brought to our web applications. The developer can create a user administration or administration of alerts. Also can add some search or filtering features to frontend.

They can also extend the variety of alerts' types and their icons. For example, adding auroras or any kind of important alert. They can add some features as well to the alerts.