Queensland University of Technology

CAB432 - Assignment 1
Mashup Project

Marat (Matt) Sadykov n9312706

Tutor: Jacob Marks marksj@qut.edu.au

September 15, 2018

Contents

1 Introduction

The aim of this project to provide users with some comfortable environment to plan their journeys. This planner will use one of the following API[1]:

- Google Maps
- Expedian with Flightstats combined for one purpose.
- Webcam

A user will be able to choose a destination, view closes and cheapest flight information and makes a decision based on a picture from realtime webcameras (Maybe it is not as pretty, but it will give right feeling regarding destination). First and the important part is an interactive map, provided by Maps services. It will allow a user to change view between satellite and geographic, zoom in to view details and help to extract geographic locations. Next is the webcam, which produces some images or short video streams on some of the sights of the area. Moreover, the Main part is quick information on the closest flight regarding price, time and journey. The first experiment version located in the Appendix.

1.1. World Map

The foundation for weekend planer environment is continental Map. Gladly, there is a high chance of finding a right Map provider. Two global websites, whose work offers an API for use cases are Google Maps and Yandex. If first is the global giant, who existed in the worldwide market for a while, second has the most popularity in Russian. Both of them will serve the primary purpose very well: establish a right environment for integrating other API. Besides, both services provide an excellent API to extend the scope of the current project.

As a result, the Google Map *Google Maps Platform* (2018) API provided a good environment, with a way presented in Figure 1.1. To locate a particular area, the server has to receive a centre point of the map with longitude and latitude. Besides, API provides a good drawing environment, which allows places some images on top of the map, draw markers or lines from one location, to another. As a scope of this unit, a working environment will be stick to Australia.

[1]Names contain hyperlinks to the resources. Feel free to click

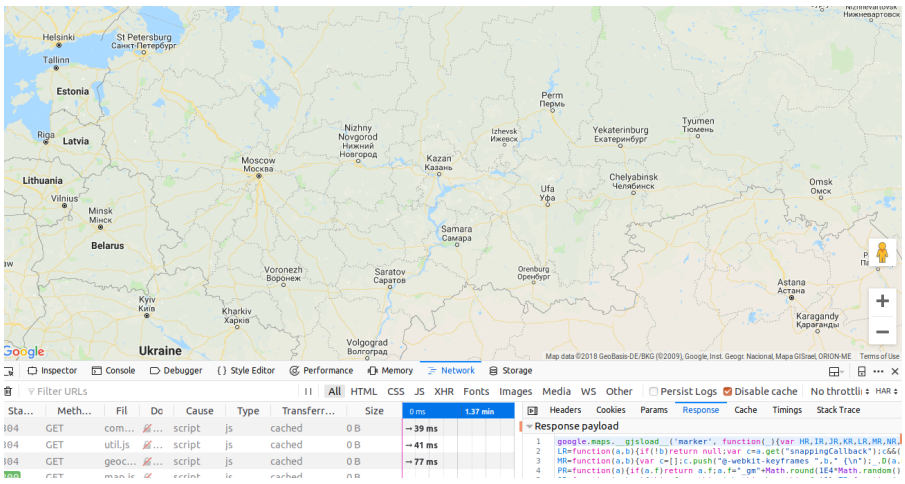


Figure 1.1: Google Maps API at 54 54 longitude and latitude.

1.2. Flight Management

In the process of gather information about the possible flight tickets, the project has potential problems, regards to context. Since the beginning of the project, the request for API keys may take longer than expected. Such services like Expedia *Expedia API Documentation* (2018) provides some flight information, which may be used to gather closest flight information about time, prices, airports. Obtaining this information may become a trickier process. For example, if there will be no alternatives discovered, the best solution in this situation will be parsing web page with flight information. However, this approach goes beyond the scope of the assignment, and it may prove itself as a good, but not easy, alternative.

As a result, plans were achieved using Flightstat *FlightStats Developer Center Home* (2018) API in combination with Expedia. In order to produce a request which can be handled by flight API, there was a need in creating a request in particular form. In this situation, Expedia was able to provide absolute returns based on the name of the city, as an example, the code below represents two predefined messages for requests. In some situation, during the development process, Expedia refused to give the response, that is why the code contains some predefined cities, to avoid error possibility and not interrupt the user. Figure 1.2 provides an example of the flight from Main City[2] to the destination.

```
1            function def_(name) {
2
3                if (name == 'brisbane') {
4                    return {
5                        displayName: '<B>Brisbane</B>, Queensland, AU',
6                        fullName: 'Brisbane (and vicinity), Queensland, Australia',
7                        lastSearchName: 'Brisbane, Queensland, Australia',
8                        shortName: '<B>Brisbane</B>, Queensland, AU',
9                        coordinates: {lat: '-27.458819',  long: '153.103613'},
10                       airport: 'BNE'
11                   };
12               }
13
14               if (name == 'sydney') {
15                   return {
16                       displayName: '<B>Sydney</B>, New South Wales, AU',
17                       fullName: 'Sydney (and vicinity), New South Wales, Australia',
18                       lastSearchName: 'Sydney, New South Wales, Australia',
```

[2]The one which user picked as starting point

```
19                          shortName: '<B>Sydney</B>, New South Wales, AU',
20                          coordinates: {lat:  '-33.86757', long: '151.20844'},
21                          airport: 'SYD'
22                  };
23                  ...
24                  ...
25              }
```

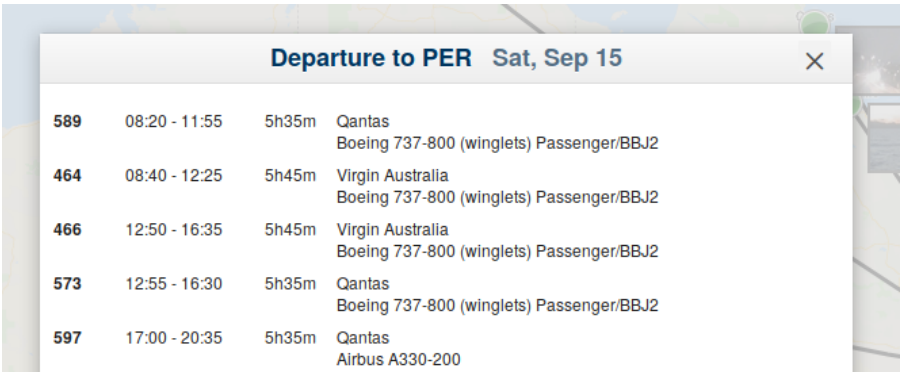


Figure 1.2: Departure flights from Brisbane to Perth.

1.3. Location Explorer

To gather information unusually and present in on top of maps webcam service will be used. Generally, it contains Library of open web cameras located around the world. Instead of gathering all possible sources, webcam includes all of them in one place. It will allow extracting at least images, which were made very recent to a user request. An example of how data can be collected is displayed in Figure 2.3.

As a result of hard work with Webcams *webcams.travel API* (2018) API, it was possible to integrate a real-time video playthrough, fresh within 5-15 minutes. In case if need, streams can be switched to Day, Month or Year in a minute. Video Players also includes information about the provider, link for connection and script for embed, in addition to full-screen view. Figure 1.3 refers to their view [3].

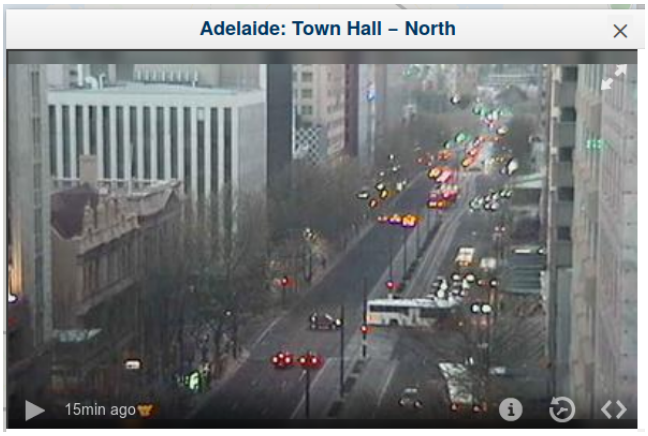


Figure 1.3: Departure flights from Brisbane to Perth.

[3]Nothing inspires more than eye dropping over someone's life or an entire city.

2 Use Cases

2.1. Trip Planner

The main expected advantage of this service is plan trips to another city on one Map, instead of planning on the official website, comparing each ticket one by one. This comparison gives user possibility select option flexibly, and choose several places and time.

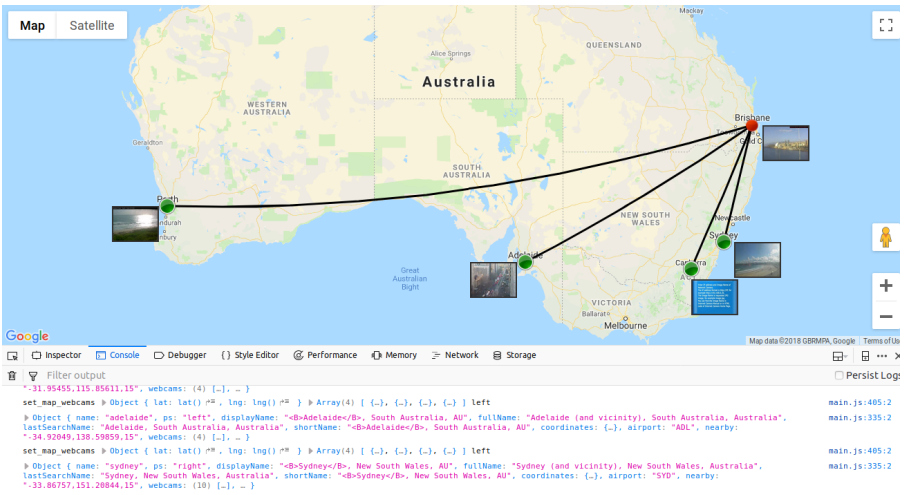
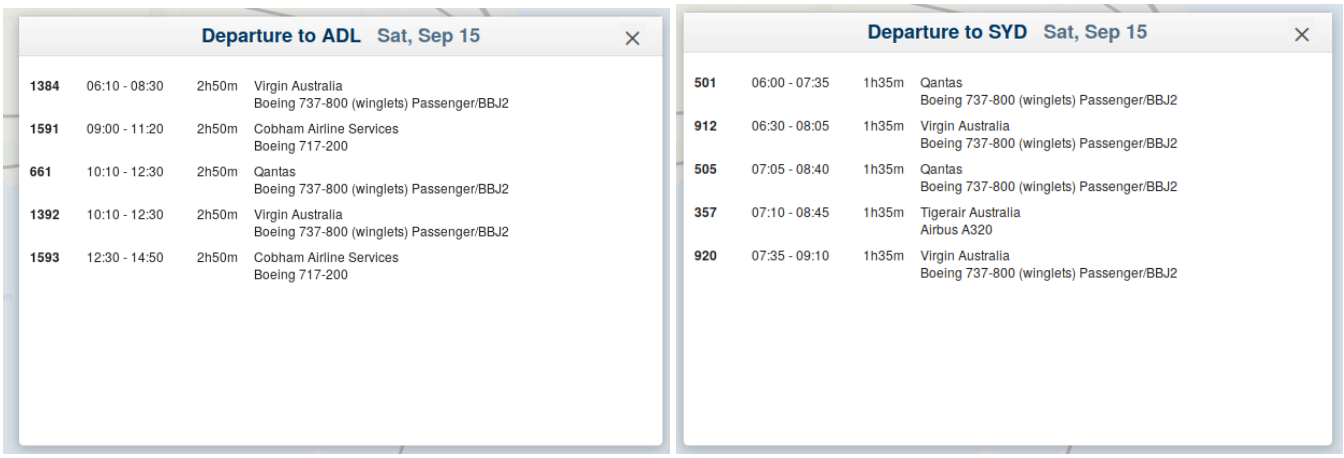


Figure 2.1: Departure flights from Brisbane to Perth.



(a) Flights to Adelaide. *(b) Flights to Sydney.*

Figure 2.2: Choosing most efficient flight.

2.2. Exploring area with cameras

Web cameras spread around cities all other the place. Some of them may contain tedious traffic exploration, another some beautiful views. Comparing to high-quality images on other services, webcam takes real-time shoots. The original intention is to allow the user to plan short weekend trips [4].

[4]There is no better way to make a decision rather take a small pick into a city of destination.

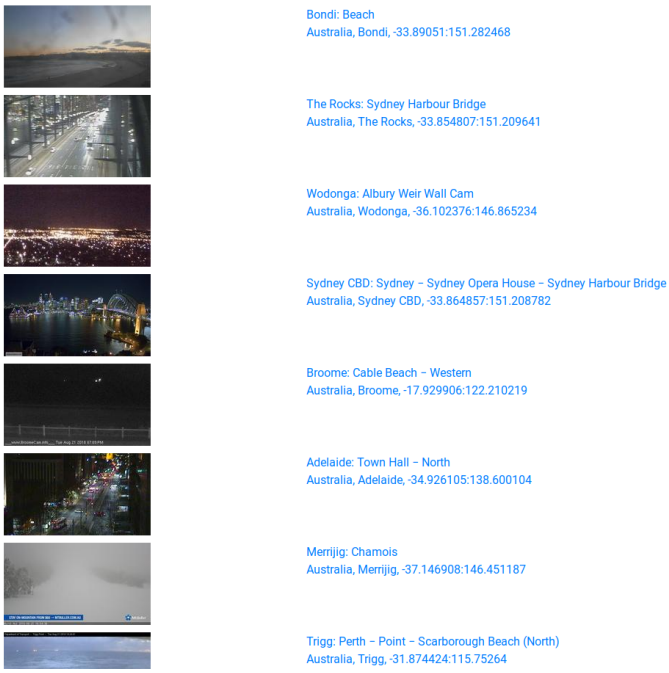
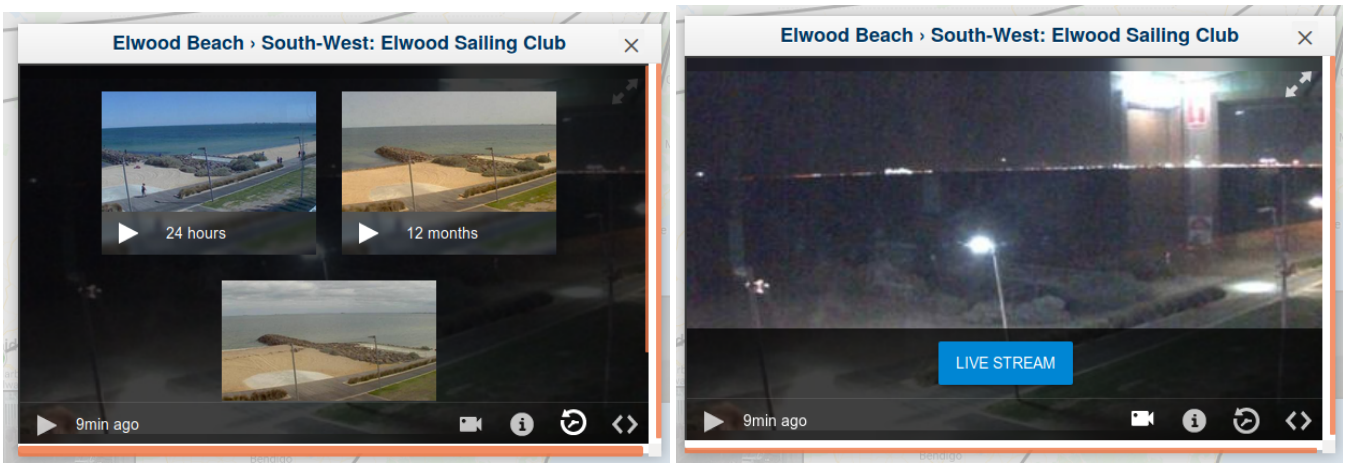


Figure 2.3: First test print with API use.

By choosing one cities, he can visualise video of current situation in predefined area.



(a) Choose from server. *(b) Watch live stream.*

Figure 2.4: Webcam in Melbourne

2.3. Exploring area with Google Maps

The usage of Google Maps allowed several map manipulations. One of them is changing overall map view to satellite. With zooming, the map gets added with extra details such as street names, or locations of interest, including shops, restaurants and more. Besides, Google provided a street view, which means a person can explore areas as a living virtual being. Some people may find this feature suitable for planning.

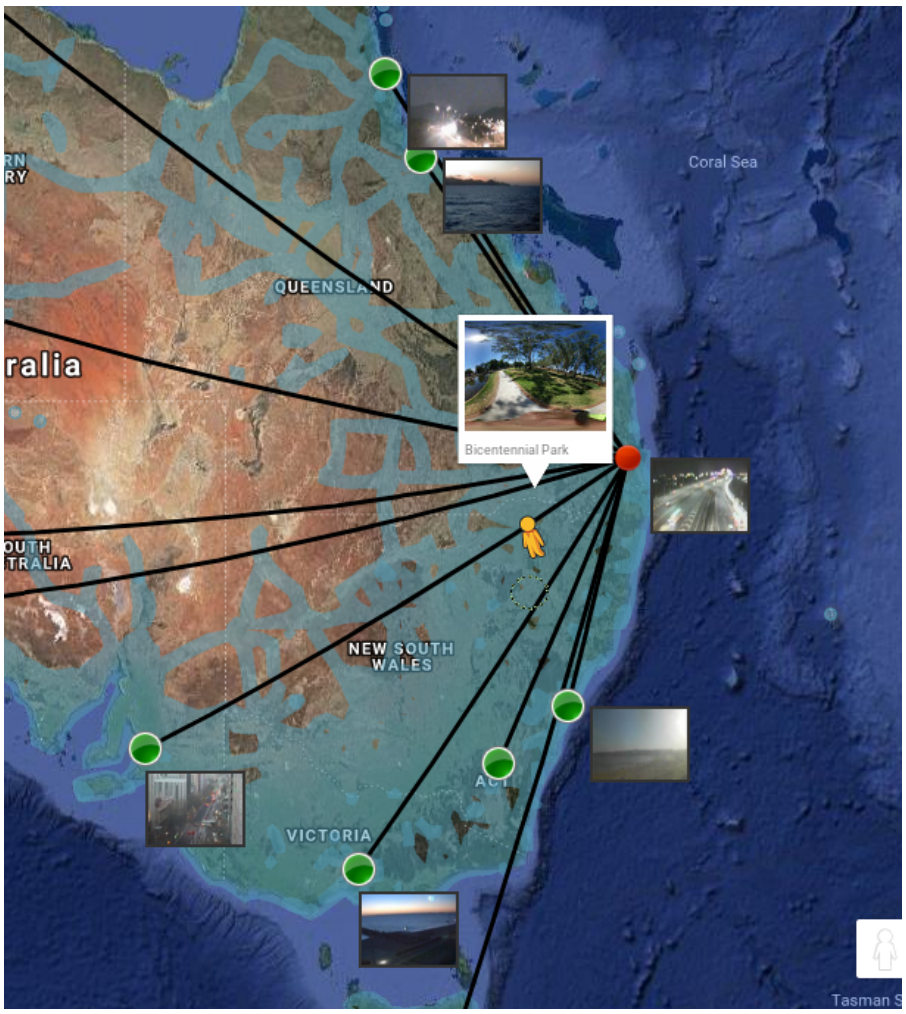


Figure 2.5: Pick location to explore with Google.

2.4. Weekend planner

In addition to flight management and camera exploration, some **the Maps services** can be used to make decisions on sight. For example, explore the streets a destination in a particular city, without moving to another Map view.
In addition to this stage of the project, final goal after implementing the server side of the project is to allow the possibility of integration this web application into Linux GNOME Desktop environment. With this opportunity, the user will be free from the browser and will be able to decide weekday workflow. It will allow storing some information of the user on the computer, instead of some server login/password way.

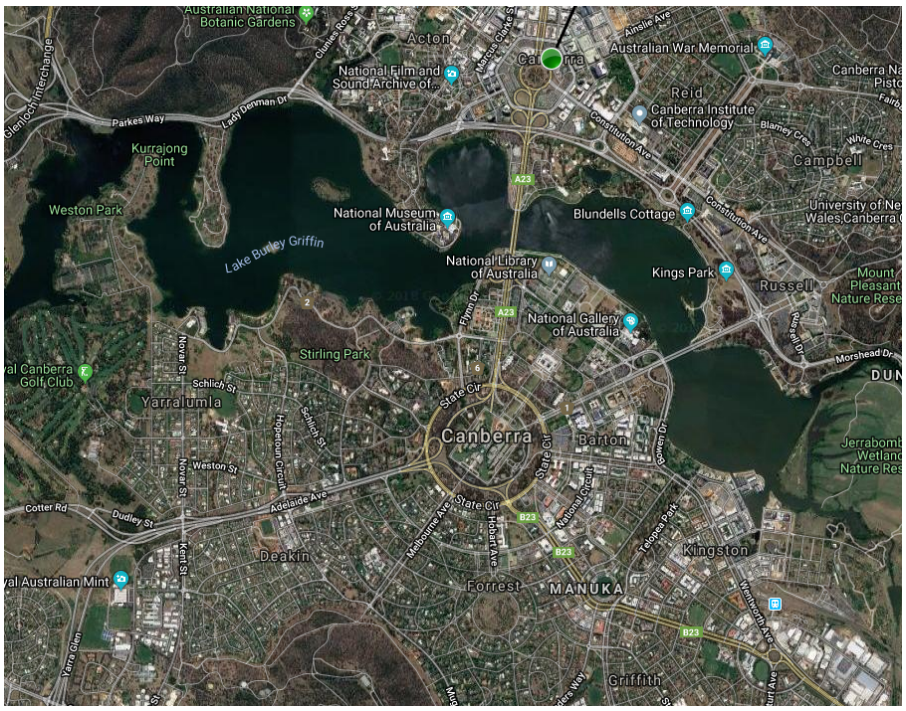


Figure 2.6: Google satellite view on locations of interests.

3 Technical Breakdown

The image below 3.1 represents the first and last structure breakdown of the current project. Arrow indicates application communication on client site, then dot arrows indicate web request and response. The blue connection between API and cloud services are constant communication to provide service.

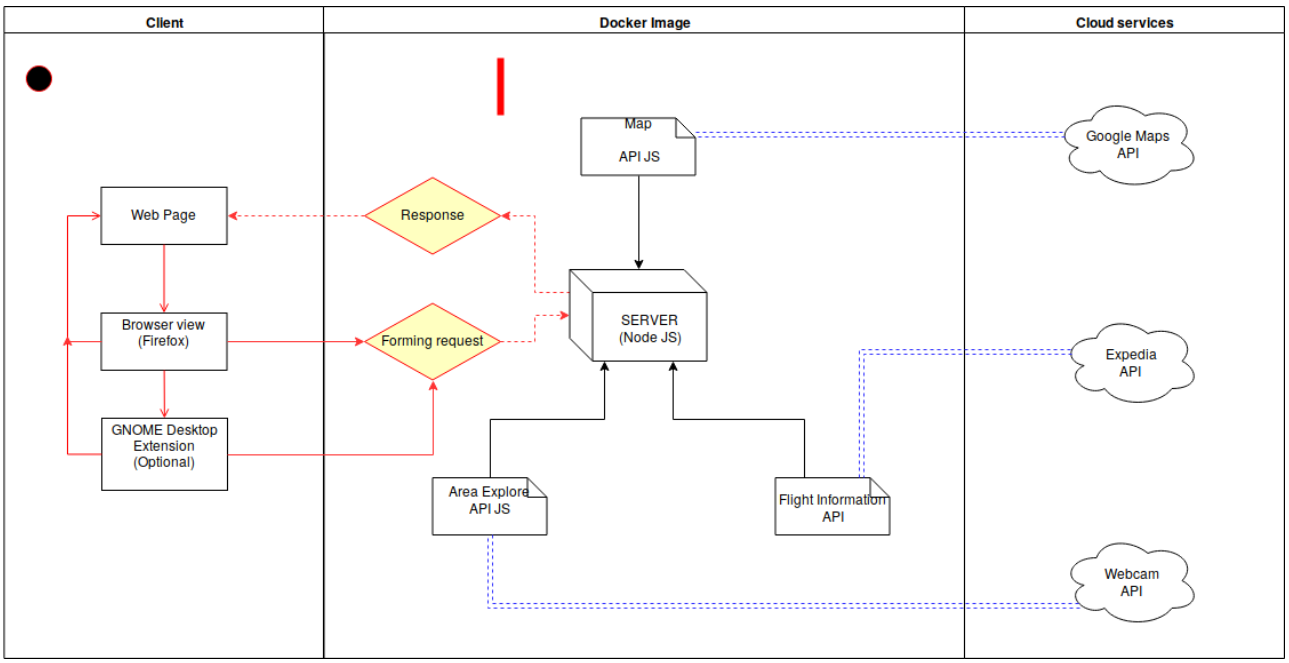


Figure 3.1: First structure Breakdown.

4 Client

At the loading page stage, a client is come up with Google Loaded Map, with predefined cities and their airports. First initial travelling location preallocated as Brisbane. Also, the user gets provided with the first images from the webcam, from which it is possible to view the full video. They get linked to dot markers of the cities in Australia. Those templates are build using Embedded JavaScript (.ejs) located in views directory. Besides, the similar window is applied for error message. After picking locations on the map, they get handled by JQuery and processed on the server side. To process each city to extract information about airports locations a way of sending asynchronous requests was used McMahon (2018). It allowed to speed up the process of drawing the web page.

After choosing a location to get flights from, the client will send the location of choice to server and server will form a proper request to flights management and return in proper view. Closest available flights within a day will be displayed. The user is also able to change the central city of departure, to any other with radio buttons.

As soon as the user click any buttons which shows a pop window, another fading layout to keep out of interaction with everything else.

All request are send using GET requests, this way it is better to focus all effort on the server side and keep the client out of complicated code[5]

5 Server

Server processes requests from the client, provides a first map page, and additional information per request. It also sends 404 page, if for some reason Map gets unavailable. In addition to error messages, if the server is in the process for extracting information, the user receives loading screen. As prior of tests, loading did not take longer than 3 seconds.

Server process requests from a user by adapting them into proper one to send with API posts. Upon their feedback, they are brought to visible form and presented on the client side. Regarding the flights - this strategy works well. However, video feedback from webcam gets processed much similarly. Instead of downloading a video to the server and showing to a user, the client receives a link to video stream straight from webcam and plays inside the browser. Such an approach removes need in filtering callbacks, does not store unnecessary unused information and saves the user from loading time.
All API information, the configuration for the server, API requests for each service and their keys get stored inside the app folder. Upon need, they get called from js files. This way the user will not be able to get access to some private server information.
File architecture in the project presented below. The app folder contains API used for requests, Views - pages for drawing and Public - A view for a browser with JQuery and CSS files.

[5]Except async.js - I found no better way to approach it.

6 Docker

All instructions how to use Docker file attached in the README.md file with the application and report.
Generally, Docker container used for simple deployment of the project any type server based machine. For the sake of time for building, the container gets uploaded to DockerHub repository. In order to download and deploy it, following command is enough [6]:

```
docker run --name Assignment1 -p 8000:3000 -i -d -t smugglersmr/weekends-pl
```

Docker build file uses a node v8 as a base ground for the project. After installing additional application are going to be installed to simplify the debugging process. After moving all files, main port 80 going to be exposed for connection. After it, every request addressed to port 8000 will be reallocated to Docker container. After running image, there is no need for additional commands to start.

```
# Set initial image to ubuntu
FROM node:8
# Author
LABEL maintainer="Smuggler"

# Updating
RUN apt update && apt install -y \
build-essential \
curl \
dialog \
git \
mc \
net-tools \
tar \
vim \
wget

# copy the app
ADD /app /app

# expose
EXPOSE 80

# Default directory
WORKDIR /app
RUN npm install
CMD [ "npm", "start" ]
```

7 Difficulties

This section is much harder to write than it appears, and the reason behind it is apparent. At first, you are trying to approach the difficult problem, and when you are finding solutions and it works, you starting questions yourself what was so difficult in the first place.

For particularly this king of Cloud Computing I found it challenging to convert coordinates into latitude and longitude to apply them for Map. And certain problems with it. Most of the

[6] Nickname of the user has nothing personal. Another simple way to remember logins combined with the past.

time I used projects which were already created, this one was the first which I developed from nothing. Maybe it could be helpful to use better templates like a pug, but I learned about it too late. API like Maps had some issues with the building, which was hard to debug. As a result, the solution behind it was adding a small time delay. Those kinds of problems not easy to approach when you can't form a proper request to Uncle Google for a solution.
I interfered with a problem passing certain information from one file to another. Understanding the logic which deriving yourself brings a new perspective for the code approach.

Getting a price list from flights API was not an easy task, none had a will to share this opportunity freely. It could be another useful implementation to allow a user to buy those tickets or calculate prices on the way. In combinations of hotels, journey planner and other useful API, this project could become a far exciting practice that I planned initially.

8 Future work

8.1. Gnome Integration

The idea to use Gnome integration considered to be irrelevant as no additional marks are going to be provided for trials. However the idea and desire still remain, there is still a hope for future projects.
This could be another experience with Linux programming. Gnome on Linux is a GUI desktop environment, which supports different kinds of extensions. One of then is OpenWeather. The idea was to design a similar extension to use in Gnome environment, then there will be no use in Web Browser. All operation could be performed in a startup environment. Those extensions not protected from bugs and crashes, but it definitely will be an experience.

8.2. More features

Some additional options such as the ability to order those tickets, reliable saving information in the Computer memory with Gnome Extension, not in some web server. Besides, adding map explorer with some good locations of interests, hotel, food and journal for travels. Those options could make this extension much useful and attract more interest around people. Also, it could become more practical regarding usage.

9 Tests

Task	**Outcome**	Result
Error screen	error.ejs applied	DONE
Build Map with cities	Google Maps Overlay	DONE
Cities choosing	Array with cities and markers	DONE
Arrange flights	Expedian in combination with Flightstat	DONE
Set cameras around cities	Webcames produces video stream	DONE
Set Restrictions from user	Server/Client approach	DONE
Safety measures for API falling	Predefined cities. No Video storage.	Sort Of
Asynchronic requests handeling	async.js integrated	DONE
Docker establishment	Dockerfile initialised	DONE
Usage of the Docker container	Container creted and stored in Hub	DONE
GitHub and DockerHub	Both created as safety measure	DONE
Latex Proposal and Report	Proposal and Report complete	DONE

An example of test written for getting information from API is provided below.

```
smuggler@Xana:/mnt/149CB8E99CB8C710/QUT/cab432/assgn1/app$ node test.js
-------test------------

webcams.show
offset: 0
limit: 10
total: 1
{ id: '1216143079',
  status: 'active',
  title: 'Pinkenba: Greenpeace, The Greenpeace ship Esperanza',
  location:
   { city: 'Pinkenba',
     region: 'Queensland',
     region_code: 'AU.04',
     country: 'Australia',
     country_code: 'AU',
     continent: 'Oceania',
     continent_code: 'OC',
     latitude: -27.443731,
     longitude: 153.10899,
     timezone: 'Australia/Brisbane',
     wikipedia: 'https://en.wikipedia.org/wiki/Pinkenba%2C_Queensland' },
  player:
   { live: { available: false, embed: '' },
     day:
      { available: false,
        link: 'https://www.lookr.com/lookout/1216143079#action-play-day',
        embed: 'https://api.lookr.com/embed/player/1216143079/day' },
     month:
      { available: true,
        link: 'https://www.lookr.com/lookout/1216143079#action-play-month',
        embed: 'https://api.lookr.com/embed/player/1216143079/month' },
     year:
      { available: true,
        link: 'https://www.lookr.com/lookout/1216143079#action-play-year',
        embed: 'https://api.lookr.com/embed/player/1216143079/year' },
     lifetime:
      { available: true,
        link: 'https://www.lookr.com/lookout/1216143079#action-play-lifetime',
        embed: 'https://api.lookr.com/embed/player/1216143079/lifetime' } } }
smuggler@Xana:/mnt/149CB8E99CB8C710/QUT/cab432/assgn1/app$
```

Figure 9.1: First test implementation.

References

Expedia API Documentation. (2018). Retrieved 2018-09-15, from https://hackathon.expedia.com/docs/public/api/Flights%20Overview/

FlightStats Developer Center Home. (2018). Retrieved 2018-09-15, from https://developer.flightstats.com/

Google Maps Platform. (2018). Retrieved 2018-09-15, from https://developers.google.com/maps/documentation/

McMahon, C. (2018, September). *Async utilities for node and the browser. Contribute to caolan/async development by creating an account on GitHub.* Retrieved 2018-09-15, from https://github.com/caolan/async (original-date: 2010-06-01T21:01:30Z)

webcams.travel API. (2018). Retrieved 2018-09-15, from https://developers.webcams.travel/#webcams/list

10 Appendix

Figure 10.1: First test implementation.

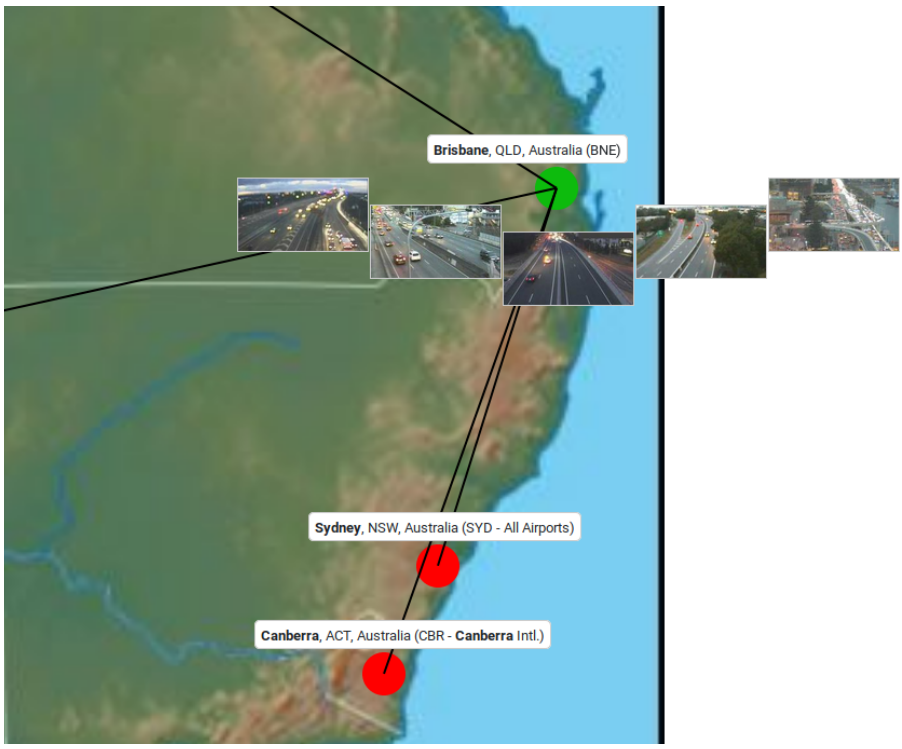
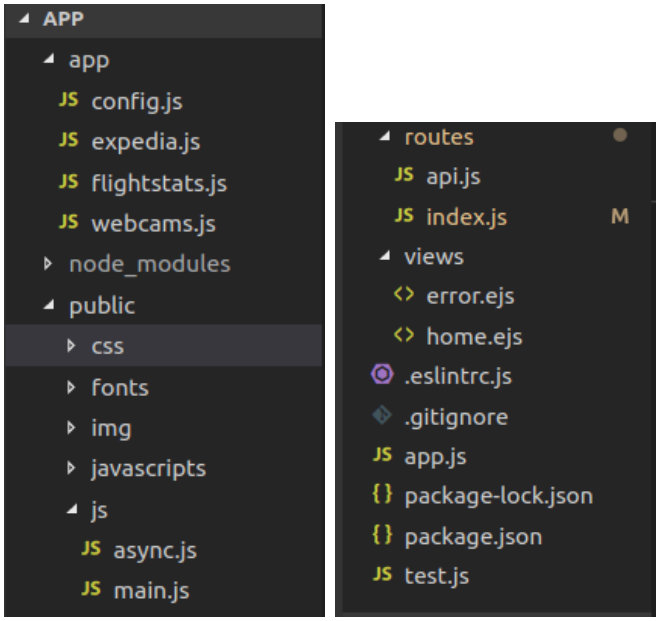


Figure 10.2: First version of overview.



(a) First part.

(b) Second part.

Figure 10.3: File structure

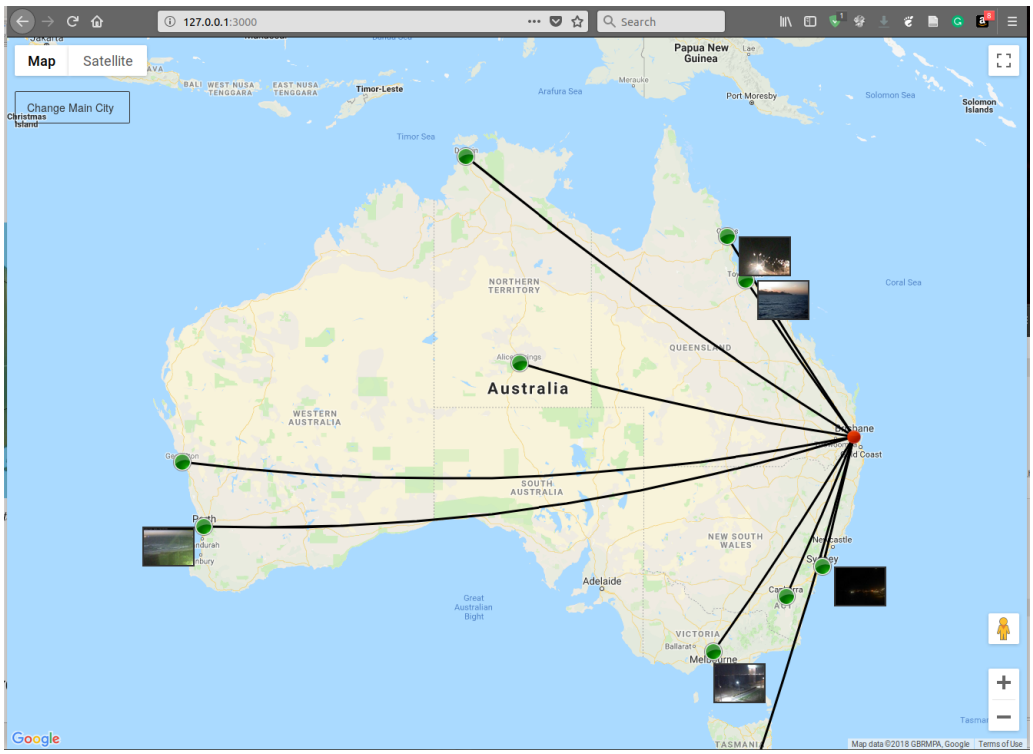


Figure 10.4: First test implementation.

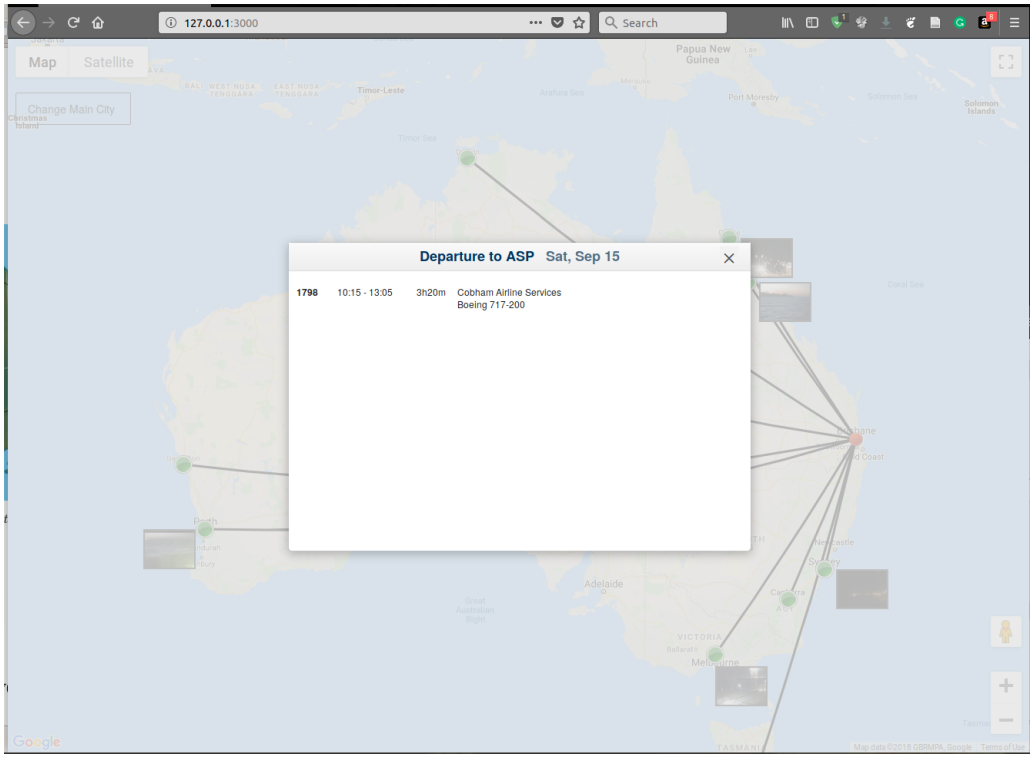


Figure 10.5: First test implementation.

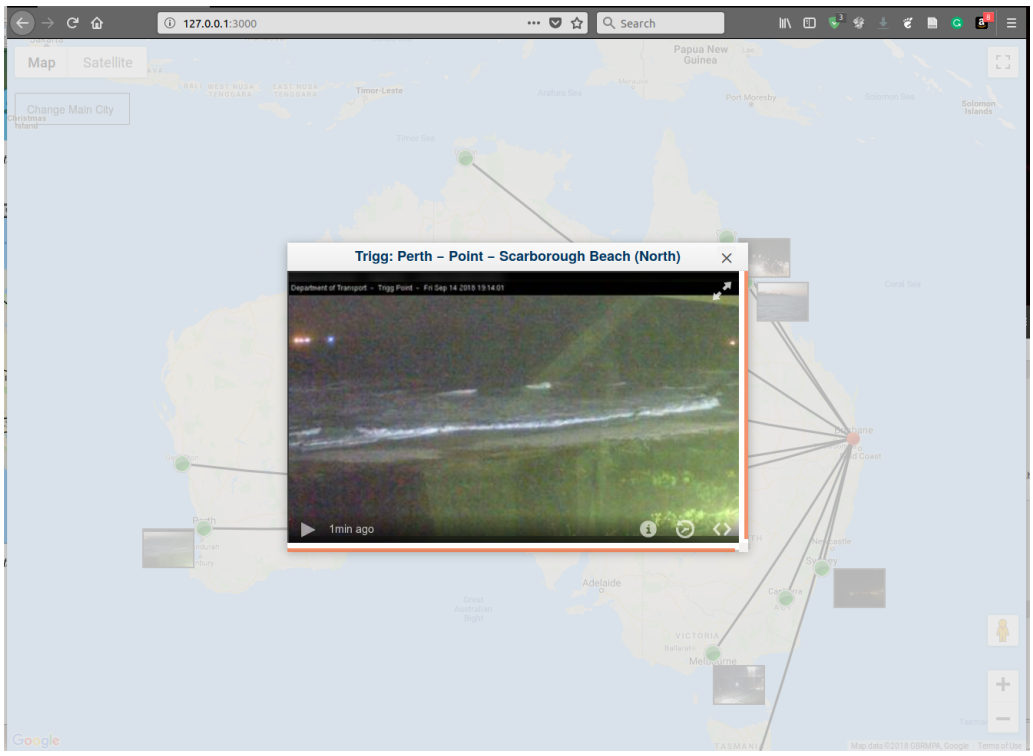


Figure 10.6: First test implementation.

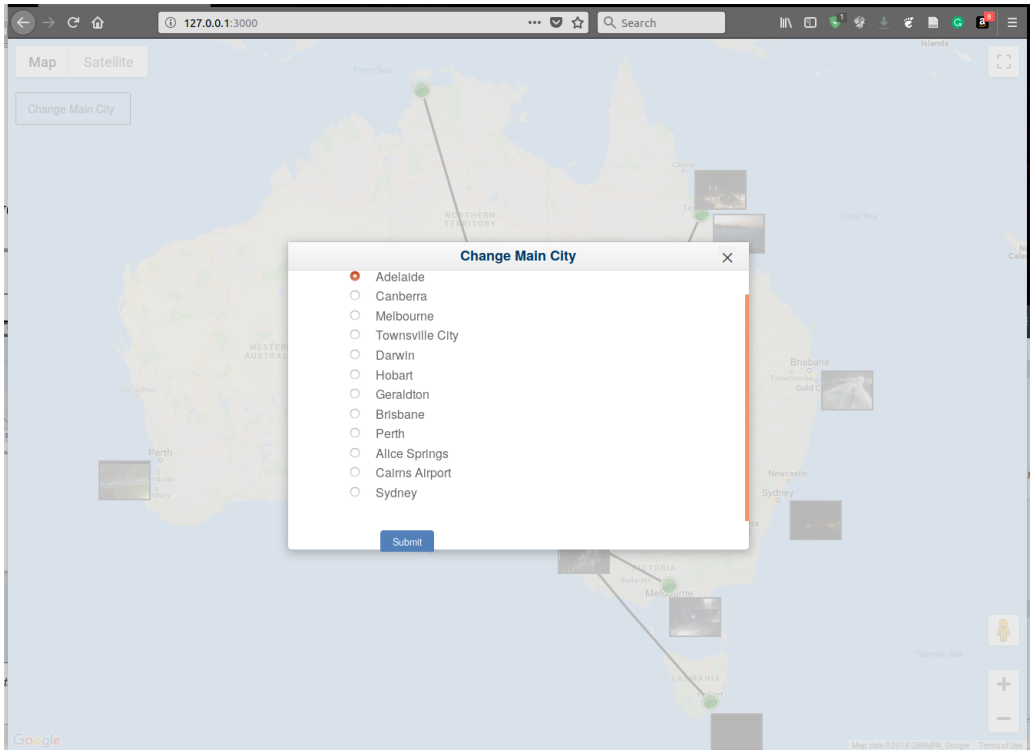


Figure 10.7: First test implementation.