

**CAB432 Cloud Computing**

**Mashup/Docker Project**

**Submitted by**

**Thanat Chokwijitkul n9234900**

**Tutor: Malcolm Corney**

**Tutorial: Tuesday 4 pm – 6 pm**

**Queensland University of Technology**

**14th September 2015**

Table of Contents

Introduction 3

Mashup Use Cases and Services 4

Technical Description of the Application 6

Server Side Architecture 6

Client Side Architecture 7

Technical Issues 8

Docker 8

Testing and Limitations 10

Unit Testing 10

Limitations 10

Possible Extensions 11

References 12

Appendix 13

# **Introduction**

Place Finder provides users with visual and interactive way of exploring nearby places. The application detects locations of the web visitors and displays some basic information, including an interactive map and weather information in the area. Users can discover the nearby places using some provided buttons labeled with place types/categories specified by the Google Places API. However, other place types/categories still can be searched via a search box.

The primary purpose of this API mashup is to provide users an easy and clear way to explore nearby businesses and services around a specified area using services and data APIs listed as follows:

#### Telize

The Telize API allows users to request a website visitor's geolocation information.

URL: <http://www.telize.com>

#### Open Weather Map

The Open Weather Map API allows users to retrieve the current weather information with various options of parameters, such as city name or geographic coordinates.

URL: <http://openweathermap.org/api>

#### Flickr

The Flickr API allows users to access and retrieve photos from the Flickr photo sharing service.

URL: <https://www.flickr.com/services/api/>

#### Google Maps

The Google Maps JavaScript interface is used to embed Google Maps onto web application.

URL: <https://developers.google.com/maps/?hl=en>

#### Google Places

The Google Places API is a web service that provides information about nearby places within a specified area.

URL: <https://developers.google.com/places/>

#### Foursquare

This API allows access to the Foursquare’s database of location as well as information on venue check in's.

URL: <https://developer.foursquare.com>

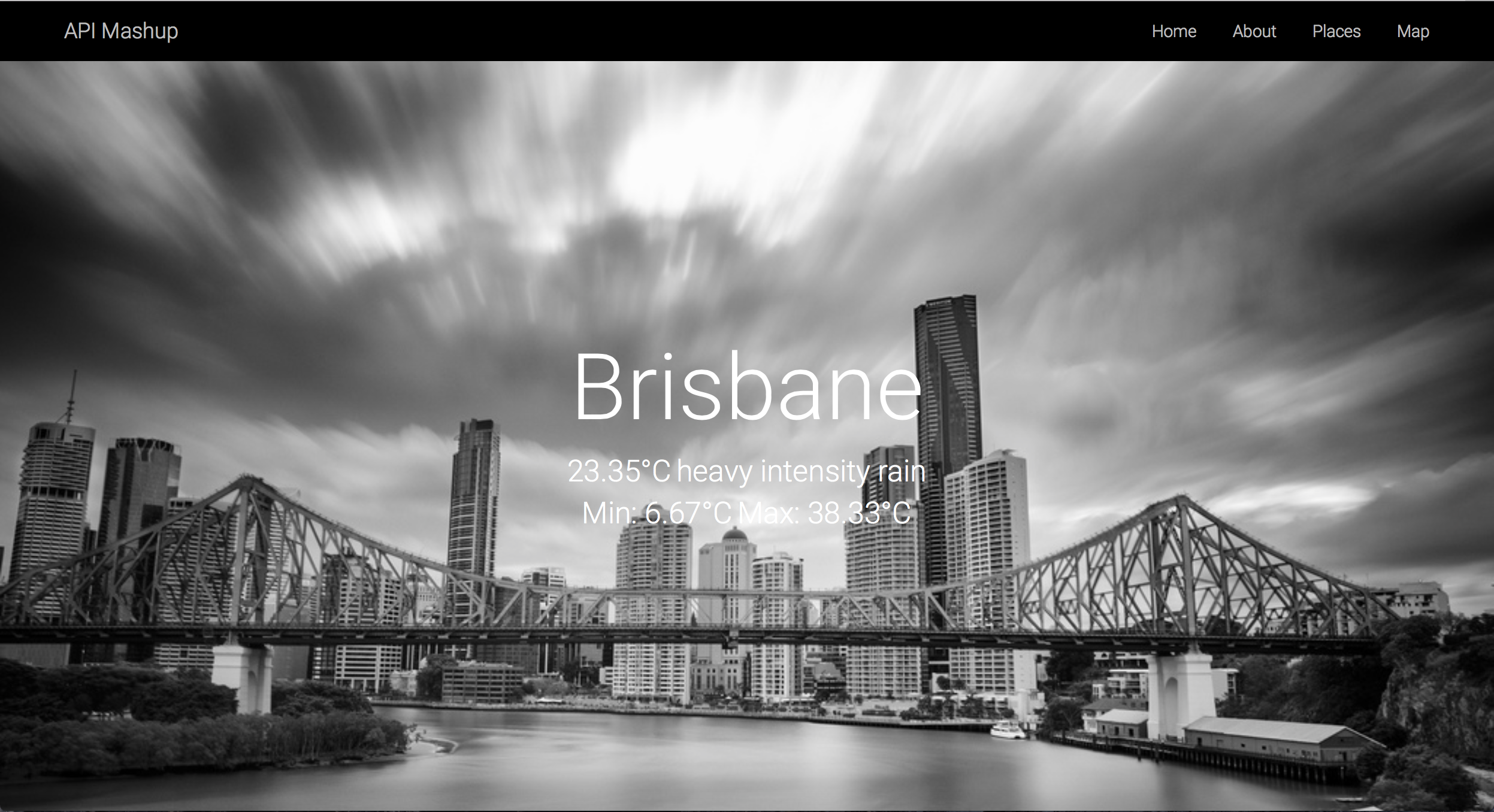
The results displayed on Place Finder are continuously updated by Telize, Open Weather Map, Flickr, Google and Foursquare. By the incorporation of the mentioned APIs, users should be able to find nearby places in a specified area with decent information.

# Mashup Use Cases and Services

This section outlines the use cases supported by the mashup and the service API calls used in the application to satisfy each user story. The term “user” used in each story refers to both tourists and local users within a specified area.

“As a user, I would like the system to be able to detect my location and display the current weather information, such as current temperature and weather description so that I can decide whether I should go out or not.”

When accessing the website, the system will detect where a user is by sending a request to Telize in order to obtain geolocation data (geographic coordinates). Afterwards, the piece of data obtained from Telize will be sent to the Open Weather Map API to get current weather information, and also the Flickr API to get a current location related photo.

**Figure 1**: The screenshot displaying the city name with current weather information.

“As a user, I would like the system to be able to detect my location and display an interactive map so that I can explore the area using that map.”

By using the Google Maps JavaScript interface, an interactive map will be embedded into the webpage. It will automatically ask for the permission to detect and display a user’s current location. If the permission is given, the map should correctly display the user’s location in the Map section.

“As a user, I would like the system to provide buttons labeled with place types/categories, such as restaurant, cafe, bar etc., so that I can find commonly searched businesses and services around my area easily without typing.”

Several buttons labeled with place types/categories specified by the Google Place API are provided in the section, called “Places”. The Places section corporates with the “Map” section. As a result, if a button is clicked, the webpage will automatically scroll down to the Map section and point out all the nearby places according to the specified type/category using markers.

“As a user, I would like to be able to search on the map so that I can find other nearby places using other keywords apart from the provided categories.”

A search box is provided on top of Google Map using the Google Places API JavaScript library. Users can search using any keywords, such as city names, place types/categories, place name etc. Searching via the search box will give some results regardless where users are.

“As a user, I would like the system to display details of each place, such as address, contact number, official website, rating or reviews, so that it can help me decide whether I want to go to that place.”

When searching for nearby places using either the provided buttons or search box, several markers will be pinned down onto the map according to the search results. Each marker contains specific information of a particular place. Therefore, when a marker on the map is clicked, the server will send a request to the Foursquare API to obtain the details of the location pointed by the marker, and then display the data inside the current infowindow. However, if the system cannot get data from Foursquare, it will send another request to get some details at least an address and rating from the Google Places Web Service instead.

# Technical Description of the Application

This section delineates the architecture and technologies used in both client and server sides, along with any technical issues encountered and how the mashup has been developed.

## Server Side Architecture

Node.js and Express (a web development framework for Node.js) are the major technologies used to build the server. The details are listed below:

#### Node.js

Node.js is chosen because it allows developers to build software with JavaScript on the server side. It is event-based rather than thread-based like Apache, and I/O operations in Node.js are asynchronous, which means it can handle incoming requests while another I/O operation is taking place.

URL: <https://nodejs.org>

#### Express

Due to the fact that Node.js itself was not designed to build a website, even though it has some basic HTTP functions, but those functions cannot be efficiently used to build an entire web application. This is where the Express framework takes place. Express is a server-side web and mobile application development framework for Node.js. It is written in JavaScript, which means using Node.js with Express allows developers to create an entire application with JavaScript. The framework itself contains common back-end functions for single-page, multi-page and hybrid mobile and web applications, and also can be used to build APIs (application programming interface).

URL: <http://expressjs.com>

#### Other Modules

This section outlines all the essential node modules used on the server side.

**Request**

The “request” module is used to send all the HTTP requests (API calls) to external servers. It also supports HTTPs and follows redirects by default.

URL: <https://www.npmjs.com/package/request>

**Morgan**

Morgan is a middleware used for logging all the request details onto the terminal console.

URL: <https://www.npmjs.com/package/morgan>

**Embedded JavaScript Template (EJS)**

Even though EJS is not directly used to render any EJS template in the application, it is used as a view engine to render HTML pages instead.

URL: <https://www.npmjs.com/package/ejs>

## Client Side Architecture

In addition to HTML and CSS, JavaScript is the main scripting language used on the client side. The details are as follows:

#### JavaScript

It is widely acknowledged that front-end development is implemented almost exclusively using JavaScript. It enables the enhancement and manipulation of web pages and client browsers. For this API mashup, it is used to manipulate HTML elements with DOM (Document Object Model) and send HTTP requests to the local server with AJAX (Asynchronous JavaScript and XML).

#### Presentation Side

HTML and CSS are used on the presentation side. The design layout of the application is based on the Twitter Bootstrap standard web page layout.

Implementation of the Project

Although the Express framework provides developers with the project generator, this API mashup has been developed from scratch.

Package.json (see Appendix B for package.json) was the first file created to initialise the project. This file contains various the details, such as project name, version, author, scripts, but the most important section, the list of dependencies (node modules) required for the project.

After installing all the node modules, the server-side components, including the Express server (see Appendix C and D for the server configuration) and other user-defined modules (see Appendix E for all the user-defined modules used in the application) were developed. The user-defined modules consist of the module that manage all the static routes, and the module that handles all the HTTP requests (API calls).

This is a single-page web application. Therefore, it only consists of 2 HTML pages, including the main web page and the error-handling page (404). As mentioned in the previous section, the layout of the web page is based on the Twitter Bootstrap standard web page layout.

Then the client-side JavaScript was developed to make the client able to interact with the local server, and manipulate the data obtained from the server.

## Technical Issues

There has been no any technical issue so far, except for some issues about Docker. When installing Docker on the local machine in the practical week 4, Docker was installed with “boot2docker”, which is a tool used to manage Docker and container deployment in the Docker host. However, after strictly following the instruction outlined in the worksheet along with the instruction on the Docker website, it was found that a new command called “docker-machine”, has replaced “boot2docker” in the package downloaded from the official website. After migrating from “boot2docker” to Docker Machine, some subcommands have been altered but not much different from the pre-existing version. However, this is a very small issue and it is the only issue that has been found.

# Docker

Docker is an open-source technology, which makes it easier for the creation and deployment of applications by using containers. While Docker is great at building and sharing software image, it also provides a central repository of disk images (DockerHub), which allows developers to run applications on top of different operating systems, such as Ubuntu, CenOS or Fedora.

For this API mashup project, the application has been deployed via Docker, sitting on top of an Azure Linux virtual machine (Ubuntu 15.04).

In order to deploy the application, a Docker image that contains the software stack needs to be built using Dockerfile, which is a text document that contains all the commands to assemble an image. The Dockerfile (see Appendix F for the complete Dockerfile) for this project consists of commands listed as follows:

Setting the base image to a specified version of Ubuntu (15.04). The FROM command tells Docker which operating system will be used to build an image.

# Set the base image to Ubuntu 15.04

FROM ubuntu:15.04

Identifying the author of the Dockerfile.

# File author / maintainer

MAINTAINER Thanat Chokwijitkul

Running the following command during the build process.

# Download and update packages

RUN apt-get update

# Install basic applications

RUN apt-get install -y nodejs npm

Copying the src folder that contains the source code into the local image.

# Copy the application folder inside the container

COPY ./src /src

Changing directory to the source file directory and install all the dependencies (node modules) using npm install.

# Run the command to install node modules

RUN cd /src; npm install

Telling the container to listen to a specific port (8080) during runtime.

# Expose port

EXPOSE 8080

Using the command CMD to provide defaults for an executing container

# Set the default command to execute when creating a new container

CMD ["nodejs", "/src/server.js"]

To use the Dockerfile to build an image for the mashup application, the command

$ docker build –t alexenriquent/place-finder .

must be used, where alexenriquent is the DockerHub username and place-finder is the application name.

The application will be run using the command

$ docker run –p 8080:8080 –d alexenriquent/place-finder

Using the –p option will map the external port to the internal port, along with using –d option, which will deamonise the application.

However, when establishing a connection between the Docker client (local machine) and the Docker host (Docker in the Azure Linux VM), the command

-tls -H tcp:// cab432-ubuntu.cloudapp.net:2376

must be used to secure the connection. For instance, in order to run the application on the Docker host, the command should be

$ docker -tls -H tcp:// cab432-ubuntu.cloudapp.net:2376

run –p 8080:8080 –d alexenriquent/place-finder

The application can be accessed via <http://cab432-ubuntu.cloudapp.net:8080>.

# Testing and Limitations

This section discusses about the testing framework used in the project and some limitations of the functionality.

## Unit Testing

The unit testing in this project has been implemented using 3 main modules, including Mocha, Should and SuperTest to test all the routes, user-defined modules (API requests) and other functionalities (see Appendix G for a sample unit test case). The details of each module used for testing are listed below:

#### Mocha

Mocha is a JavaScript test framework running on Node.js. It can be used for both synchronous and asynchronous testing.

URL: <https://mochajs.org>

#### SuperTest

SuperTest is a module developed by super-agent. It provides services for testing HTTP but also can be used to test lower level APIs.

URL: <https://www.npmjs.com/package/supertest>

#### Should

Should is a very expressive and readable testing framework for Node.js. Some major advantages are to keep the code organised and to make error messages expressive and helpful.

URL: <https://www.npmjs.com/package/should>

## Limitations

An obvious limitation of the functionality is the data retrieved from Foursquare’s database. Information of some places cannot be found in the company’s database. As a result, if the system cannot find any information of a specified place in the Foursquare’s database, it will send another request to Google Places Web Service (which is different from its API JavaScript library in terms of usability) to obtain at least basic information, such as place name, address or rating, and display these information to users instead.

# Possible Extensions

According to the limitations mentioned in the previous section, one possible extension is to integrate the application with more web services or data APIs, such as Factual or Yelp. This approach can be used as a backup plan, in case that data of a specified place cannot be found in one database, it might be available in another database from a different API endpoint. In addition, existing data obtained from different databases can be used for statistical analysis and optimising user experience.

# References

# Appendix A

#### Brief User Guide

When a user accesses the website, the system will detect where the user is, display the name of the city and also provide basic weather information, including current temperature, maximum and minimum temperatures (all in Celsius).

If scrolling down to the 'Map' section, the map should display the current location (the area where the user is).

For the 'Places' section, several buttons labeled with place types/categories can be used to find nearby businesses or services. After clicking on one button (and the webpage is fully loaded), the webpage will automatically scroll down to the 'Map' section and display all the nearby places according to the specified type/category.

Each marker contains specific information of a particular place. Clicking on a marker will display details, such as address, contact number, official website, rating or reviews.

Note: If the system cannot get any information for a particular place, it means no data can be obtained from the Google Places and Foursquare's database. However, it will display the name of that place by default.

# Appendix B

**package.json**

{

"name": "api-mashup",

"version": "1.0.0",

"description": "The API mashup provides users with visual and interactive way of exploring nearby places.",

"main": "server.js",

"scripts": {

"test": "make test",

"start": "node server.js"

},

"dependencies": {

"ejs": "~2.3.3",

"express": "~4.13.1",

"mocha": "^2.3.1",

"morgan": "^1.6.1",

"request": "^2.61.0",

"should": "^7.1.0",

"supertest": "^1.1.0"

},

"repository": {

"type": "git",

"url": "git+https://alexenriquent@github.com/alexenriquent/api-mashup.git"

},

"keywords": [

"API",

"mashup"

],

"author": "Thanat Chokwijitkul",

"license": "ISC",

"bugs": {

"url": "https://github.com/alexenriquent/api-mashup/issues"

},

"homepage": "https://github.com/alexenriquent/api-mashup#readme"

}

# Appendix C

**server.js**

/\*\* Module dependencies \*/

var http = require('http');

var app = require('./app');

/\*\* Identify port from environment \*/

var port = process.env.PORT || 8080;

/\*\* Create a server and log a message to the console \*/

http.createServer(app).listen(port, function() {

console.log('Listening on port ' + port);

});

# Appendix D

**app.js**

/\*\* Module dependencies \*/

var express = require('express');

var ejs = require('ejs');

var logger = require('morgan');

var routes = require('./routes/index');

var api = require('./routes/api')

/\*\* Create an 'express' object \*/

var app = express();

/\*\* Use Morgan - Log requests to the terminal console \*/

app.use(logger('dev'));

/\*\* Add connection to the 'views' folder for css and javascipt \*/

app.use(express.static(\_\_dirname + '/views'));

/\*\* Add connection to the 'public' folder for html \*/

app.use(express.static(\_\_dirname + '/public'));

/\*\* Set EJS as a templating language with HTML as an extension \*/

app.engine('.html', ejs.\_\_express);

app.set('view engine', 'html');

/\*\* Static route \*/

app.get('/', routes.index);

/\*\* API routes \*/

app.get('/geolocation', api.geolocation);

app.get('/weather/:location', api.weather);

app.get('/photo/:location', api.photo);

app.get('/placeinfo/:location', api.placeInfo);

app.get('/placedata/:id', api.placeData);

app.get('/place/:location', api.place);

/\*\* Export 'app' module \*/

module.exports = app;

# Appendix E

**index.js**

module.exports = {

/\*\* GET index.html \*/

index: function(req, res) {

res.render('index', { });

}

};

**api.js (partial - only the HTTP request to Telize)**

/\*\* Module dependencies \*/

var request = require('request');

/\*\*

\* RESTful API calls

\* @module API

\*/

module.exports = {

/\*\* HTTP request to the Telize API \*/

geolocation: function(req, res) {

var url = 'http://www.telize.com/geoip';

request(url, function(error, response, body) {

if (error) {

return console.log('Error: ', error);

}

if (response.statusCode !== 200) {

return console.log('Invalid status code: ', response.statusCode);

}

var info = JSON.parse(body);

res.send(info);

});

};

# Appendix F

**Dockerfile**

# Dockerfile to build API mashup

# Based on Ubuntu

# Set the base image to Ubuntu 15.04

FROM ubuntu:15.04

# File author / maintainer

MAINTAINER Thanat Chokwijitkul

# Download and update packages

RUN apt-get update

# Install basic applications

RUN apt-get install -y nodejs npm

# Copy the application folder inside the container

COPY ./src /src

# Run the command to install node modules

RUN cd /src; npm install

# Expose port

EXPOSE 8080

# Set the default command to execute when creating a new container

CMD ["nodejs", "/src/server.js"]

# Appendix G

**index.test.js (partial)**

/\*\* Module dependencies \*/

var app = require('../app');

var http = require('http');

var request = require('supertest');

describe('Static Routes', function() {

describe('GET /', function() {

it('GET / should return 200', function(done) {

request(app)

.get('/')

.expect(200, done);

});

});

});