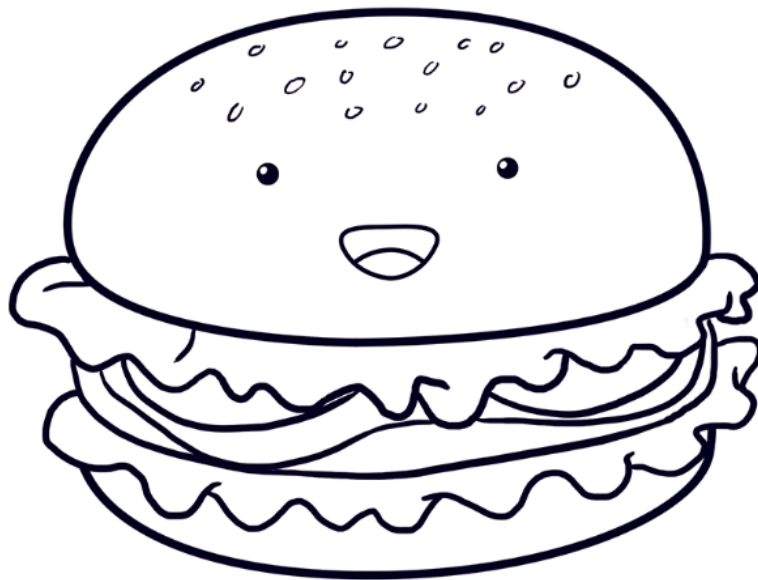


CAB432 Cloud Computing Assignment 1

Christian Neilsen – n9137874

# In or Out?



*Helping you make your food decisions*

# Contents

Introduction .....	4
RapidAPI .....	4
Spoonacular Food and Recipe API.....	5
Zomato API .....	5
Google JavaScript Maps API .....	5
Use Cases .....	7
Use Case A .....	7
Use Case B/C .....	9
Use Case D .....	12
Technical Breakdown.....	14
Client .....	14
Server.....	14
PUG .....	15
Difficulties .....	16
Lack of Web Development Experience.....	16
API Selection.....	16
Docker .....	17
Extensions.....	18
Automatic Cuisine/M meal Tagging for Untagged Recipes.....	18
Use of Front-End Framework.....	18
Shopping List Functionality .....	18
‘Did You Mean’ Search Recommendations .....	19
Testing.....	20

Error Handling .....	20
Test Cases .....	22
Appendix A – User Guide.....	24
Appendix B – Testing Screenshots .....	27
Appendix C – Docker Install Guide .....	35

## Introduction

'In or Out' aims to provide a combination of services that lets a user quickly find delicious looking meals, and then gives them the tools and information to help them decide whether they want to eat in, or out. There are two primary routes a user can take using the app, the first of these being a directed search and the second being a randomised search for users having trouble finding food inspiration.

For the direct search the Spoonacular Food/Recipe, Zomato and Google Maps JavaScript APIs will be used, users can search for a type of food or cuisine, and then get a list of results via images with attributed meal names. Upon selecting the food they want, the user will be taken to the results page for that food item. This results page will display recipe information such as: cooking/prep time, necessary ingredients and cooking instructions. Alongside the recipe information, related Zomato restaurant details will be displayed through a filterable list and markers pinned on a Google Map.

For the randomised search, the end points will be the same, however rather than being shown a list of recipes based on a search the user will instead be shown a carousel of randomly selected recipes, showing them a wider range of meal ideas and helping them decide on what food they are interested in. If nothing looks good to the user they can also generate a new set of random recipes to look at.

## RapidAPI

<https://rapidapi.com/>

The RapidAPI marketplace provided middleman Nodejs functions to make calls to Zomato APIs, and also provided Nodejs request code for Spoonacular endpoints. The middleman functions were provided by npm installing 'rapidapi-connect' and then calling the specific API functions for each API taken from the RapidAPI website.

## Spoonacular Food and Recipe API

<https://spoonacular.com/food-api>

The Spoonacular API provides a robust set of callable methods and API endpoints categorised into 4 separate APIs (Ingredients, Recipes, Food Products, Menu Items). Use of the Spoonacular API for the app focuses on making calls to the Recipes API.

The methods/endpoints used for the app are: the **GET Search Recipes** method with the `‘/recipes/search’` endpoint, to find recipes based on a search query, and the **GET Recipe Information** method with the `‘/recipes/RecipeID/information’` endpoint to extract more information on a particular recipe. The **GET Get Random Recipes** method was used for the random recipe search to populate the carousel with recipe information.

## Zomato API

<https://developers.zomato.com/api>

The Zomato API was chosen over the Yelp API to provide restaurant information due to the higher level of specificity allowed for using Zomato (more cuisine categories to allow for more customizable searching). The Zomato API is used to provide restaurant data using either the tagged cuisines for a recipe (if available) or the recipe name as a search query. Zomato API also provides a basic form of location searching to get different results for different city queries.

The methods used for the app are: **POST searchCity** with user input as the search query to get the most relevant city ID, **POST getCities** to get the city name related to the city ID for displaying purposes, **POST getCuisines** to get the list of cuisine IDs available in a given city for matching against a recipes tagged cuisines, and finally **POST search** to retrieve a list of restaurants for a given city and filtered for the specified cuisines.

## Google JavaScript Maps API

<https://developers.google.com/maps/documentation/javascript/tutorial>

The Google JavaScript Maps API was chosen for its well explained and accessible documentation. The API will be used to map relevant restaurant markers on a map in the recipe result page.

All of the functions and API calls for the Google Maps API will be made on the client side, with limited processing of data also being performed on the client side to allow for markers to be placed, removed and filtered based on UI actions from the user. Descriptions of the mapping functions used will be provided later in this report.

# Use Cases

## Use Case A

*Primarily uses Spoonacular, also uses Zomato and Google Maps*

As a user, I want to be able to **search for food/recipes and get a list of results**. After selecting a recipe that appeals to me I want to be able to see more information about the recipe such as **cooking/prep time, ingredients and cooking instructions** so that I can try and replicate the recipe myself. I also want to get a list of related restaurants in case the prep/cooking time is too much of a commitment for me.

This user will navigate to the index page of 'In or Out' with the intent to search for a recipe or type of food/cuisine. Using the search bar they will enter what their search query and click submit (This will call the Spoonacular GET Search Recipes method):

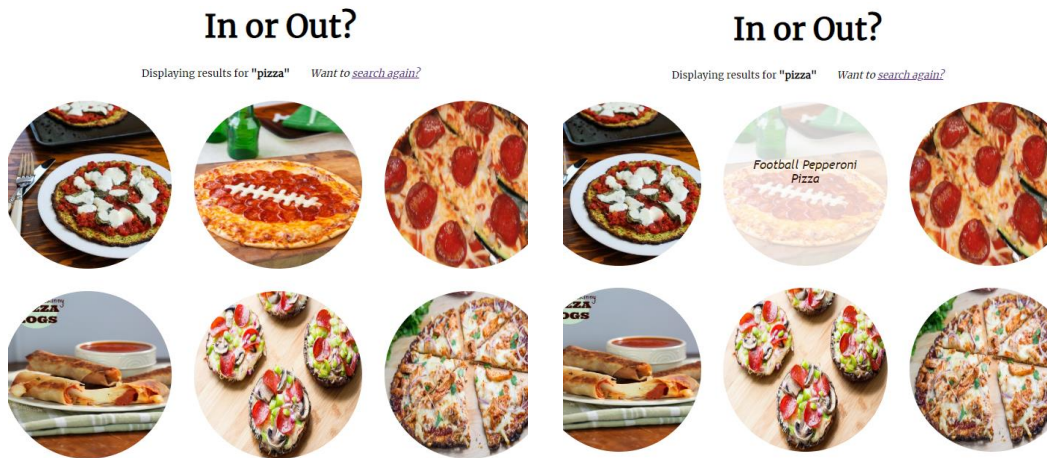
## In or Out?

Submit

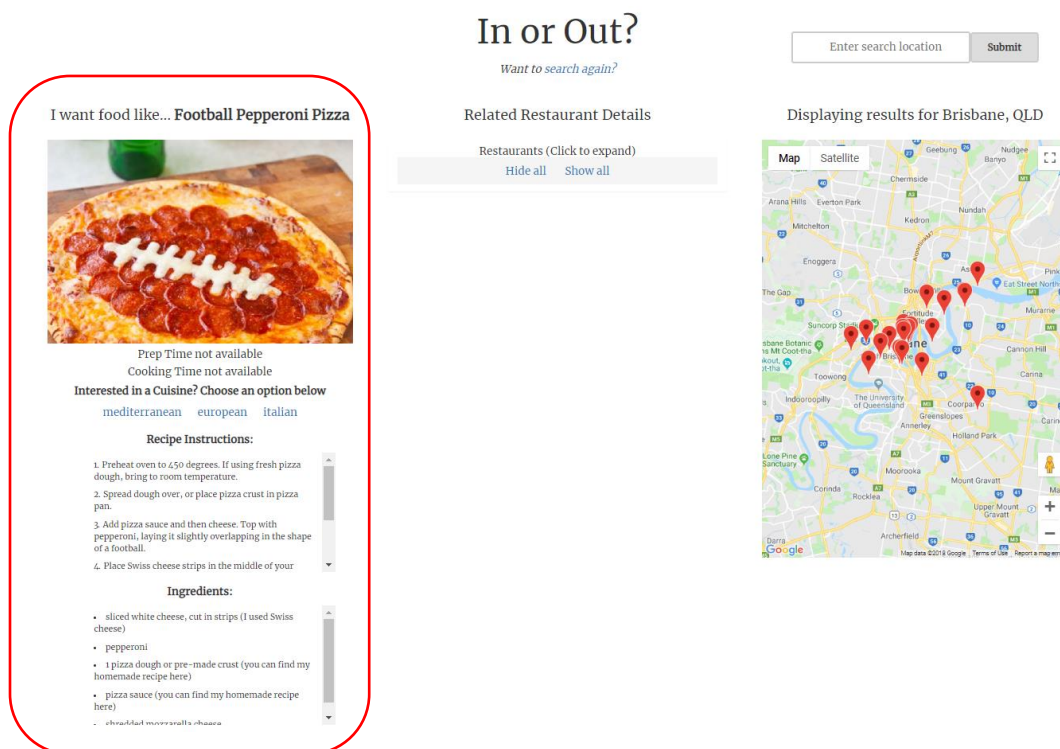
Can't Decide?



For example, after a user searches for 'pizza' the following results will be displayed, and the user can hover their mouse over an image to get the recipe name (hmmm, football pizza 🤔):



After clicking on the football pizza the user will be taken to the food results page where they will shown the relevant **cooking/prep time, ingredients and cooking instructions for their selected recipe**. From this page they will also be able to look at relevant restaurants if they wish (This will call the Spoonacular GET Recipe Information method, as well as the four Zomato POST methods, depending on if the user wants to change the location):





## Use Case B/C

Both use Spoonacular and Zomato, Case C uses Google Maps

As a user, I want to be able to **find related restaurant data** for a recipe selected from a list. After this:

- *Case B:* I want to be able to see these related restaurants and their information in a list so that I can decide on where I want to eat Out.
- *Case C:* I want to be able to **see these related restaurants on a map** so that I visualize how far away they are from me. I also want to be able to **filter the restaurant markers shown on the map by cuisine selection**, with the option to **hide/show all markers or select only specific restaurants to display their markers**.


For use cases B and C and users will search for a recipe/cuisine using the index just like in use case A.

For **Use Case B**, after selecting a recipe and being taken to the food result page, the user will be able to click on the header of the restaurant details page, expanding the table and displaying relevant restaurant details:

### In or Out?

*Want to search again?*

I want food like... **Spicy Korean Cucumber Kimchi Refrigerator Pickles**



Prep Time not available  
Cooking Time not available

Interested in a Cuisine? Choose an option below

asian korean

**Recipe Instructions:**

1. Trim ends of cucumbers; cut into spears.
2. Place in a large bowl and toss with salt, chili powder, ginger, garlic, soy sauce, and scallions. Put into glass jars and add water. Cover tightly and let sit on the kitchen counter for 24 hours. Refrigerate.
3. Serve after a few days. The longer you refrigerate them, the more flavor the pickles will have.

#### Related Restaurant Details

**Restaurants (Click to expand)**

**Bird's Nest Yakitori**

Cuisines: Asian, Japanese, Tapas

User Ratings: 4.5 - Excellent

Address: Shop 5, 220 Melbourne Street, South Brisbane, Brisbane

**Sushi Kotobuki**

Cuisines: Asian, Japanese, Sushi

User Ratings: 4.5 - Excellent

Address: 53 Lytton Road, Kangaroo Point, Brisbane

**Madtongsan II**

Cuisines: Korean

User Ratings: 4.4 - Very Good

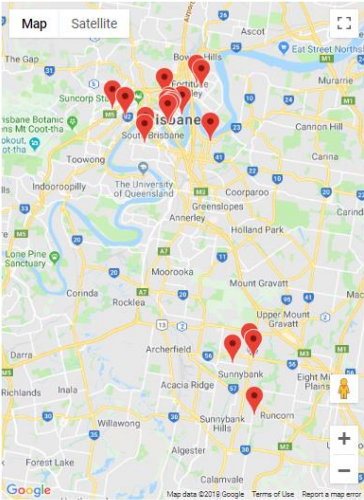
Address: Shop 1, 85 Elizabeth Street, Brisbane CBD, Brisbane

**The Bun Mobile**

Cuisines: Asian, Burger

User Ratings: 4.5 - Excellent

Displaying results for Brisbane, QLD

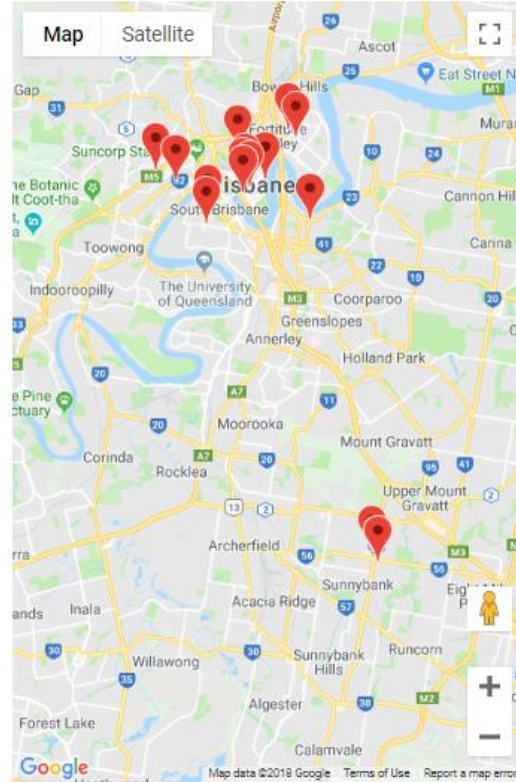


For **Use Case C**, a user can filter map results by clicking on a cuisine tag on the recipe information column at the left of the food result page. The figure below shows the result of a user clicking on the 'korean' and 'asian' tags for a recipe with related restaurants tagged with both 'asian' and 'korean' food:

Displaying results for Brisbane, QLD  
Filtering for **korean** food



Displaying results for Brisbane, QLD  
Filtering for **asian** food

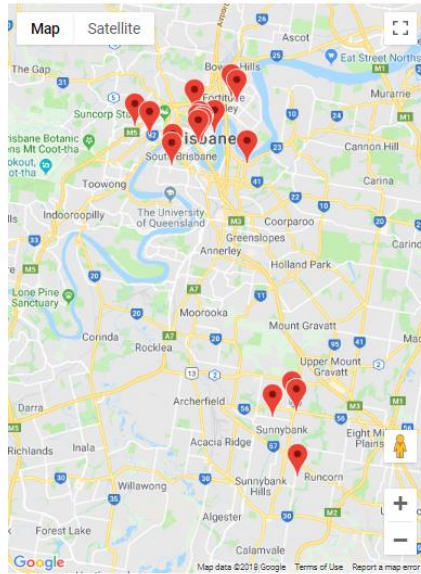


Additionally, for **Use Case C**, a user can click on the 'Hide all' or 'Show all' text in the restaurant details footer to hide/show all of the restaurant markers, with the hide all click removing all markers along with the maps current scope:

Restaurants (Click to expand)

[Hide all](#) [Show all](#)

Displaying results for Brisbane, QLD



Restaurants (Click to expand)

[Hide all](#) [Show all](#)

Displaying results for Brisbane, QLD



Finally, for **Use Case C**, after hiding all of the restaurant markers, the user can click on individual restaurant names to add that restaurant's marker to the map and automatically scope around it. The figure below shows the results after a user hides all map markers, then clicks on two restaurants from the list:

Related Restaurant Details

Restaurants (Click to expand)

[Bird's Nest Yakitori](#)

Cuisines: Asian, Japanese, Tapas

User Ratings: 4.5 - Excellent

Address: Shop 5, 220 Melbourne Street, South Brisbane, Brisbane

[Sushi Kotobuki](#)

Cuisines: Asian, Japanese, Sushi

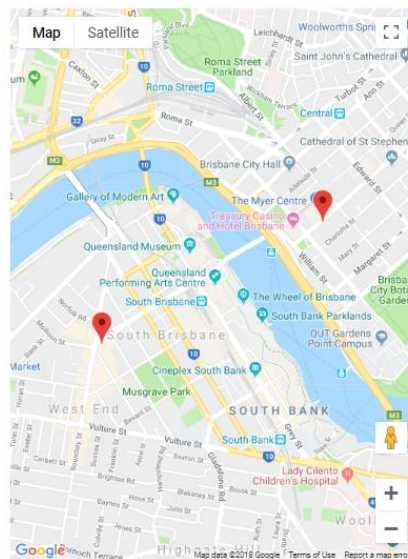
User Ratings: 4.5 - Excellent

Address: 53 Lytton Road, Kangaroo Point, Brisbane

[Madtongsan II](#)

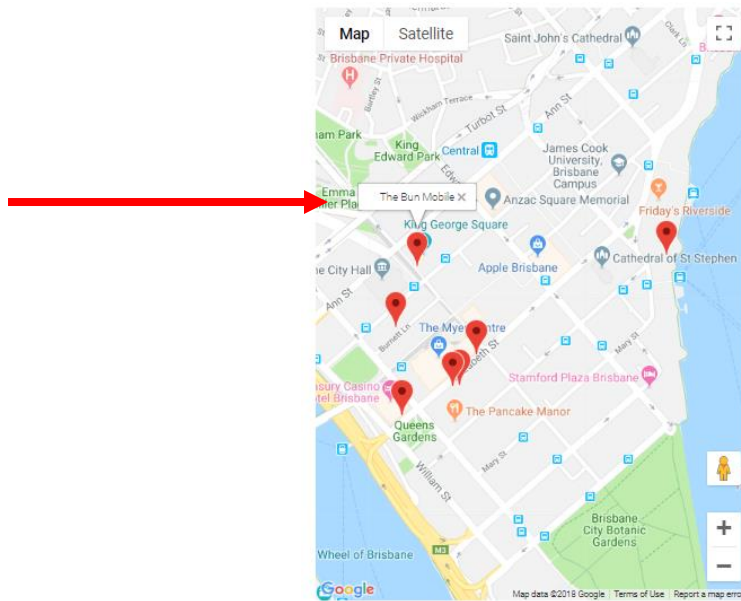


Displaying results for Brisbane, QLD





When a user hovers their mouse cursor over a map marker the restaurant name for that marker will show up, showing the user where each restaurant is located:



## Use Case D

*Primarily uses Spoonacular, but also uses Zomato and Google Maps*

As an indecisive user, I want to be able to **see a diverse range of food without requiring me to search for anything specific**. I want to be able to easily make a choice for what sort of food/cuisine I want to eat, **and then from here I want to see how to cook my selected recipe OR to see related restaurants in my area**.

For **Use Case D** the user will navigate to the index page, and upon deciding that they can't make up their mind on what kind of food they want, they will click the 'Can't decide' link. After clicking this link they will be redirected to the random food result page, where they will be able to view multiple random food items in a carousel layout. The user will be able to click to the right or left of the image to change to another food item, and if they find something they like they will click the image center and be taken to the previously described food result page, where they can view the recipe information and related restaurants. The following screenshots show an example of this:

The user navigates to the random food results page, and after scrolling through the results they find something that fits the bill.

## In or Out?

See anything you like ; )



[Click here for more random results](#)


[Or go back home](#)

They **click on the image** to get the recipe information and then find out that there's actually a **nearby Mexican restaurant with great reviews!**

## In or Out?

[Want to search again?](#)

I want food like... **Black Bean Quesadillas with Garlicky Shishito Peppers**



Prep Time: 10 mins  
Cooking Time: 15 mins

Interested in a Cuisine? Choose an option below

[mexican](#)

**Recipe Instructions:**

1. Prepare the peppers by cutting into 1/4" thick slices and removing the stem.
2. Heat a large pan or griddle over medium heat.
3. Add olive oil followed by the peppers. Saut until the pepper pieces are browning and beginning to blister, 5 or so minutes.
4. Add the garlic to the pan and cook for 1 to 2

### Related Restaurant Details

Restaurants (Click to expand)

**Billy Kart Kitchen**

Cuisines: Modern Australian, Mexican

User Ratings: 4.4 - Very Good

Address: 1 Eric Crescent, Annerley

**Guzman y Gomez**

Cuisines: Mexican

User Ratings: 4.5 - Excellent

Address: Emporium, Shop 31, 1000 Ann Street, Fortitude Valley, Brisbane

**La Quinta Mexican Cafe & Bar**

Cuisines: Mexican, Tex-Mex


User Ratings: 4.1 - Very Good

Address: Shop 1, 189 Oxford Street, Bulimba, Brisbane

**Pepe's Mexican Restaurant**

Cuisines: Mexican, Tex-Mex

Displaying results for Brisbane, QLD



# Technical Breakdown

## Client

The client initially makes a request to the Nodejs server, and is served the default 'index.pug' Index page using express. From here the user will have the option to either make a request to search for food (via making a HTTP POST request to the server) or to get a number of randomly selected recipes (redirecting to the random food page where the server makes necessary API calls to grab data).

When the user clicks on a recipe image in the food search page a hyperlink will redirect them to the food results page with the 'result' query being set to the chosen recipes ID, and a default 'loc' location query of Brisbane (although it isn't the best use of resources, location selection is performed in the food results page alone for the purpose of simplifying the UI).

The location search bar in the food results page is simply used to alter the query sent in the Zomato API request by the server.

Due to the use of Google Maps Javascript API to display restaurant locations, it was necessary to have a sizeable amount of scripting in the food results 'foodres.pug' file, however the majority of data processing is performed by the server in the 'index.js' file.

## Server

The server responds to requests from the client by making extended requests to the Spoonacular and Zomato APIs. If a successful request is made the responses from the APIs are then stored in JavaScript variables and processing/handling of the data is done, with the resulting processed variables then being pushed to the relevant pug views via the express app.render() function.

The npm packages used to run the app are: express (used to host the app and to route js files), path (to create static directory and set links), pug (used as the view engine in the place of HTML), rapidapi-connect (used to make Zomato API requests) and unirest (used to make HTTP requests to the Spoonacular API endpoints).

In trying to make use of sound coding practice/design patterns I made the choice to provide a 'separation of concerns' with the apps codebase by employing express routing. By using the following syntax, I was able to separate each major function/body of code and isolate it in a separate JavaScript file:

```
let foodSearch = require('./foodsearch');  
app.use('/', foodSearch);
```

In the above example, the './foodsearch' requirement refers to a Javascript file, with the app.use function appended the file to the previously defined static directory. In the 'foodsearch.js' file a router object was defined with 'const router = express.Router()' and then this object was used to serve the pages function using 'router.get()' before exporting with 'module.exports = router'. This allowed me to greatly clean up my code base while keeping all functionality, leaving the 'index.js' file to act primarily as the Server. Another benefit of this approach is the modularisation of code and easier testing later down the line.

## **PUG**

The Pug templating engine was chosen for use over HTML for rendering the apps views for a number of reasons:

- Simple indent-styled syntax which greatly reduces amount of front-end code needed
- Ability to compile into a JavaScript function using rendered variables as arguments (allows for easier access to data parsed from the Server).
- Full integration with Express.

Pug was used to manage layouts, respond directly to Boolean cases to provide dynamic views and to render popups and Google Maps API responses. Development was made much easier due to Pugs Server-side JavaScript compilation, which allowed JSON data to be parsed directly from the Server and then displayed accordingly. This also reduced the amount of client-side processing required.

## Difficulties

### Lack of Web Development Experience

The primary difficulty I encountered was a near complete lack of experience in web development, having never created a website or used JavaScript, Node.js or many other technologies used throughout the assignment.

The first hurdle for me in starting development was understanding the basic structure of a website and its source code, resulting in a lot of mis-directed references when using express and trying to access locally stored images. Luckily the large amount of educational content online, especially content regarding JavaScript's asynchronous functionality, helped me greatly in achieving most of the functionality I set out for myself.

### API Selection

With my original proposed API's (Yelp instead of Zomato) it proved harder and harder to achieve the desired app function when using Yelp's APIs. A major factor in the successful operation of 'In or Out' is the availability of tagged 'cuisines' to make requests and get relevant results. When comparing the number and specificity of 'cuisines' available for searching using Yelp and Zomato, Zomato was the clear winner. Because of this, gears were shifted a few weeks into development and the app was repurposed to work with Zomato's API instead of Yelp's.

Other key difficulties arising from API selection was the lack of tagged cuisines on certain recipes returned by Spoonacular requests. With no option to specify that cuisine tags were required and with development time running out a quick fix had to be implemented in the case of a selected recipe having no assigned cuisine. This resulted in no Zomato restaurant or Google Map data being displayed in these cases. An attempt was made to search for restaurants using the recipe name however the results this produced were often poor, so this was scrapped last minute. If more time was available, a form of search parsing (possibly using NLTK or other machine learning) to extract key food information, would have been tested to try and provide better results and usability in these cases. Another issue came from running out of Zomato API requests on the day before submission while I was trying to test the app. Luckily I had a previously stored JSON file that I used as dummy data to continue.

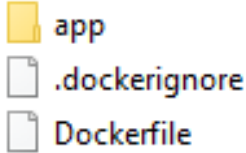


# Docker

When dockerising the application I decided that for my purposes the simpler the process was, the better it would be. Because of this I chose to simply create a docker image of the app using a Dockerfile rather than Docker composing. For my deployment I decided to run my Dockerfile one level above the 'app' directory, with the following commands making up my Dockerfile:

```
FROM node:8

WORKDIR /usr/src/app
COPY /app/package*.json ./
RUN npm install
COPY /app/. .
EXPOSE 80
CMD [ "npm", "start" ]
```



Although it is not the latest stable version, Node.js version 8.0.0 was the chosen image for building the docker image. The main reason for this is the support and stability of v8.0.0 and the fact that it worked without hiccups with all required dependencies for the app. As the node:8 base model comes with Node.js and NPM installed, step 3 was used to copy over the package.json file and app dependencies to the WORKDIR before these dependencies were installed in step 4 using 'npm install'. Copying the json file over in step 3 was done before copying the rest of the files in step 5 (COPY ..) due to dockers installation layers, which would keep track of the installed dependencies in the case of a drop in connection (preventing a full reinstall from being required). After bundling the app's source code in step 5 port 80 is set to be exposed on the local machine, and finally the command 'npm start' is run to start the Node.js server using the 'start' command found in the the apps 'package.json' file. A simple '.dockerignore' file is stored next to the Dockerfile with the contents:

```
node_modules
npm-debug.log
```

This ignore file prevents docker from trying to bring over these (often large) folders/files when creating an image.

## **Extensions**

### **Automatic Cuisine/M meal Tagging for Untagged Recipes**

A big issue throughout development of 'In or Out' was the previously mentioned lack of cuisine tags on some recipes. As using cuisine tags to request relevant restaurant data was a key part of the app, the fact that Spoonacular did not have a request parameter that only returned results with cuisine tags made things difficult.

For one build I manually filtered out non-tagged results but it over complicated the process and led to a large amount of results (almost all drinks/dessert items) being filtered out completely, which may not be in the best interest of the someone trying to use the app for Use Case A (just trying to get information about a specific recipe with the OPTION to look at related restaurant information).

If I had more development time I would either use some form of NLTK/machine learning to estimate cuisine tags for untagged recipes OR possibly use Google Vision API to get tags based on the recipe's image (processing a picture of a coffee might return a 'coffee' or 'café' tag etc.).

### **Use of Front-End Framework**

If I started a similar project from scratch I would look into using some form of front-end framework such as Bootstrap in order to speed up development and programming of the UI. With my near complete lack of Web Dev knowledge, I spent a large amount of time early on trying to search for and to perfect a CSS styling or JS animation that I wanted to use. It was only later on that I discovered the use of Bootstrap (some of which I used to develop the app). I tried my best to make the app display correctly for multiple screen resolutions but using a framework would make its presentation and display more stable while reducing noisy/unnecessary CSS code.

### **Shopping List Functionality**

As stated in my initial proposal, I wanted to give the user the option to choose whether they wanted to search for relevant grocery stores (i.e. nearby Asian grocers for 'Asian' tagged recipes) however I focused entirely on restaurant data towards the end of development. If I

had more time I would try working on integrating grocery store searching and possible shopping list functionality, to help balance the 'Should I eat in or out' factor of the app.

### **'Did You Mean' Search Recommendations**

Similar to Googles 'did you mean' function, in the future I would make the app recommend potential fixes to search queries (food or location) that are likely to not give any results, rather than letting a user type in a string with typos (e.g. 'frdie chigen') initially and returning no results.

# Testing

## Error Handling

All of the Spoonacular and Zomato API requests made make use of anonymous call back functions to handle errors. Spoonacular requests were made and handled in the form shown by the following snippet:

```
uniREST.get( REQUEST_ENDPOINT_AND_PARAMETERS )
    .header( REQUEST_HEADERS )
    .end(function (result) {

// ----- ERROR HANDLING ----- //

    if (result.status !== 200) {
        console.log('Error for specific case, Status code: ' + result.status);
        res.render('index.pug')
    }
    // Rest of code processing request //
    }
```

The index page was chosen as the redirection page for most failed requests with necessary information being logged to the console. This was done to be non-obtrusive from a user's point of view (rather than just crashing the app or taking to dead-end status code page).

Likewise, the following snippet shows the method of processing Zomato API requests:

```
rapid.call(SEARCH_METHOD, {Search parameters})
    .on('success', (payload)=>{

        console.log('success');
        // Rest of code processing request //

    }).on('error', (payload)=>{

// ----- ERROR HANDLING ----- //

        console.log('Error doing SEARCH_METHOD');
        res.render('index.pug');

    });
```

In addition to the request handling code shown above, separate code was written to handle special cases, such as when a user submits an invalid search query e.g. ('sad89234hsdf'). The following code snippet was used to test for the case where no location suggestions were returned aka and invalid search was made:

```
if (payload.result.location_suggestions.length > 0) {  
  // DO THESE THINGS //  
} else {  
  console.log('Invalid Location Search');  
  console.log(foodItem);  
  res.render('noresults.pug', {foodItem: foodItem, location:  
location});  
}
```

On the client side pug was used to loop and perform conditional tests on variables passed to it from the server via res.render(). For example:

```
if prepTime !== undefined  
  h3 Prep Time: #{prepTime} mins  
else  
  h3 Prep Time not available
```

Or checking a Boolean value, simply:

```
if anyCuisine
```

The various test cases used on the app and their results can be seen on the following page

## Test Cases

After the final version of the app was ready to submit I performed a series of manual tests checking all of the desired UI functionality. The following table shows the test cases that were run in order to test the functionality and error handling of the app:

Task	Expected Outcome	Result	Screenshots (Appendix B)
Enter app address into browser	Display the index/home page	PASS	1
Click on Random Food Button	Display random food results	PASS	2
Click on 'more random results'	Display a new set of results	PASS	-
From random page click home link	Return back to index/home	PASS	-
Click on recipe image from random search page	Display correct food results page	PASS	-
Click on random recipe carousel (left, right icons)	Slide to the next recipe/food item	PASS	-
Search for food	Display relevant search results	PASS	3
Search for food (invalid search)	Display 'no results' page	PASS	4
Click 'search again' from no results page	Return back to index/home	PASS	-
Hover mouse over recipe image	Display recipe name	PASS	5
Click on recipe image	Display correct food results page	PASS	6

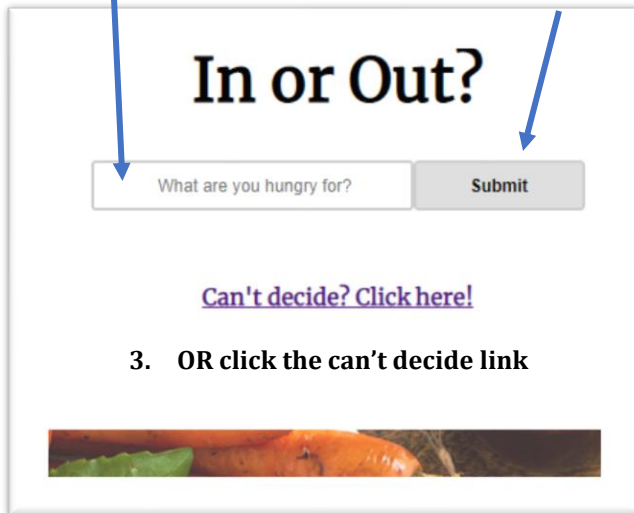
From results page click 'search again'	Return back to index/home	PASS	-
Click on foods cuisine option (if multiple)	Filter map based on cuisine	PASS	7
Click on second cuisine option	Wipes previous markers, shows new markers on map	PASS	-
Click on restaurants info table header	Expand/minimize the restaurants info list	PASS	8
Click on hide all	Wipe markers, scope map to original bounds ('Australia')	PASS	9
Click on show all	Show all restaurant map markers, scope to their bounds	PASS	-
Click on specific restaurant name after wiping map	Adds marker and scopes map around it and any new added markers	PASS	10
Hover over marker	Show restaurant name	PASS	10
Enter in valid search location	Pulls new Zomato data and shows new restaurants on map	PASS	11, 12
Enter in invalid search location	Load invalid search location page	PASS	13, 14
Get recipe with no cuisine	Don't display the map, remove search bar	PASS	15

## Appendix A – User Guide

The following section will show you a simple user guide for 'In or Out':

1. Enter a search term

2. Hit enter or click submit



3. OR click the can't decide link

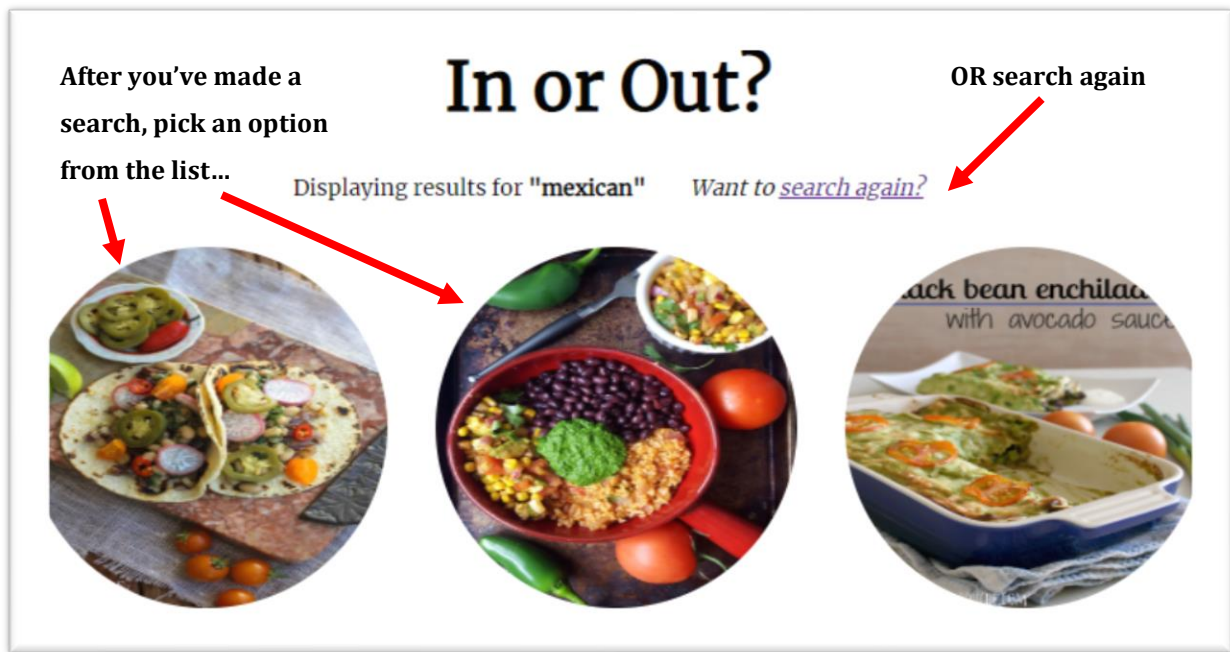
**In or Out?**  
See anything you like ; )  
If you couldn't decide, click left/right of the image to navigate and find something you like, then click it centre on



OR Click here  
for more  
random results

OR Click  
here to go  
back home





Now you can look at the recipe information...

Or look at relevant restaurant info by expanding the table

Or change the search destination

## In or Out?

[Want to search again?](#)

Enter search location

I want food like... **Mojo Pork Tacos with Mexican Rice**

Prep Time not available  
Cooking Time not available

Interested in a Cuisine? Choose an option below

[mexican](#)

Recipe Instructions:

1. Combine the juice from the fresh oranges, orange juice, onion, garlic, cumin and oregano and a blender and pureed until smooth. Set over night. Pork: Season the pork on all sides with salt and pepper or the optional spice blend.

Related Restaurant Details

Restaurants (Click to expand)

[Hide all](#) [Show all](#)

Displaying results for Brisbane, QLD

Or filter the results shown on the map by clicking a cuisine

## Related Restaurant Details

Restaurants (Click to expand)

### Billy Kart Kitchen

Cuisines: Modern Australian, Mexican

User Ratings: 4.4 - Very Good

Address: 1 Eric Crescent, Annerley

### Guzman y Gomez



Address: Shop 1, 189 Oxford Street, Bulimba, Brisbane

### Pepe's Mexican Restaurant

Cuisines: Mexican, Tex-Mex

User Ratings: 4.2 - Very Good

Address: 184 Enoggera Road, Newmarket, Brisbane

### El Torito

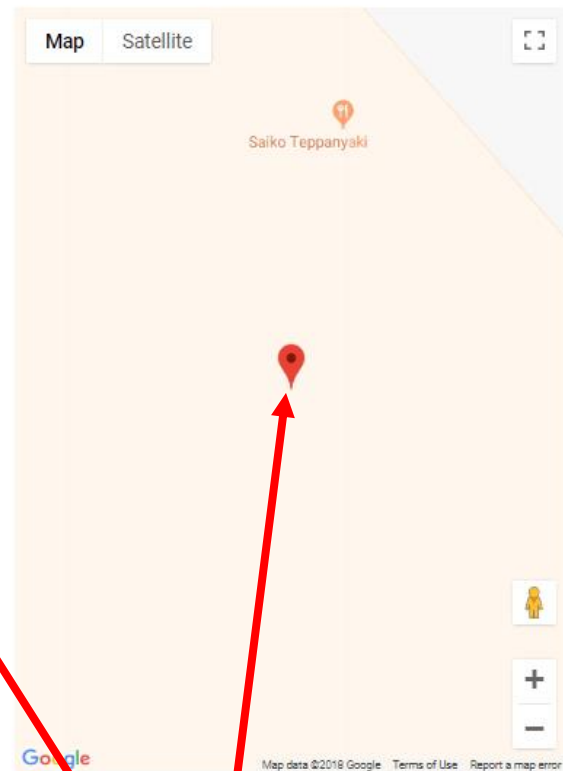
Cuisines: Mexican

User Ratings: 4.0 - Very Good

[Hide all](#) [Show all](#)

You can also show/hide  
all restaurant markers

## Displaying results for Brisbane, QLD



Or click on a restaurant  
name and show specific  
markers (after hiding all  
previous markers)

## Appendix B – Testing Screenshots

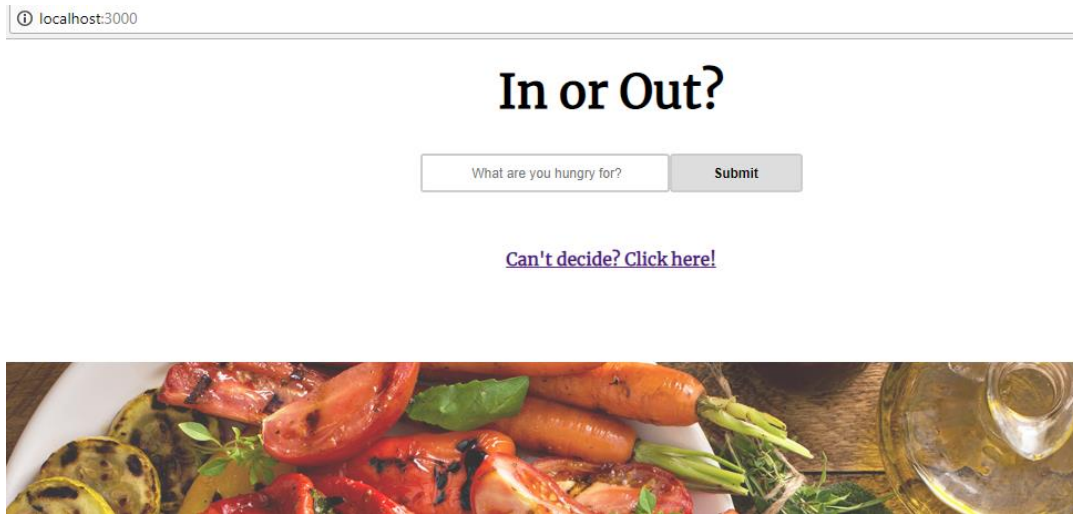


Figure 1 - Test figure 1



[Click here for more random results](#)  
[Or go back home](#)

Figure 2 - Test figure 2

# In or Out?

Displaying results for "pasta"    Want to [search again?](#)



Figure 3 - Test figure 3

# In or Out?

No results found for "sdasdubasjhdasjhd"    Want to [search again?](#)



Figure 4 - Test figure 4




Figure 5 - Test figure 5

# In or Out?

Want to search again?

I want food like... **Crock-pot Ravioli Casserole**



Prep Time not available  
Cooking Time not available

Interested in a Cuisine? Choose an option below

[mediterranean](#) [european](#) [italian](#)

## Related Restaurant Details

Restaurants (Click to expand)

[Hide all](#) [Show all](#)

Displaying results for Brisbane, QLD




Figure 6 - Test figure 6



Displaying results for Brisbane, QLD  
Filtering for **mediterranean** food

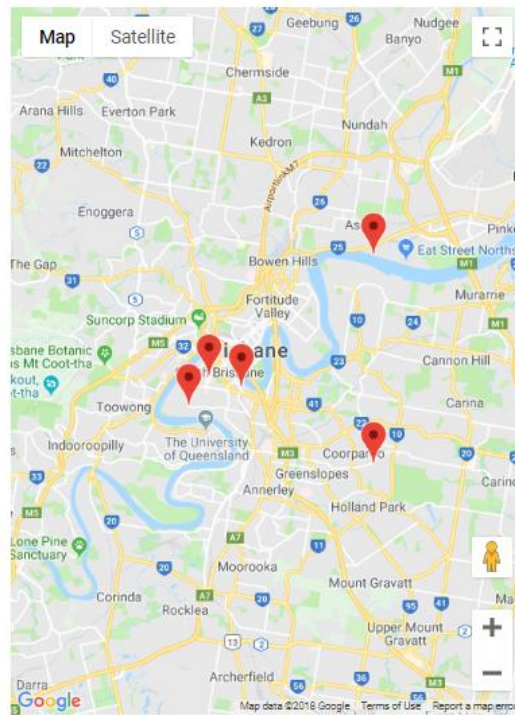


Figure 7 - Test figure 7

## Related Restaurant Details

Restaurants (Click to expand)

### Little Greek Taverna

Cuisines: Mediterranean, Greek, Salad

User Ratings: 4.5 - Excellent

Address: Shop 5, 1 Browning Street, South Brisbane, Brisbane

### Jamie's Italian

Cuisines: Italian, Salad

User Ratings: 3.9 - Good

Address: 237 Edward Street, Brisbane CBD, Brisbane

### Beccofino

Cuisines: Italian, Pizza

Figure 8 - Test figure 8



Figure 9 - Test figure 9



Figure 10 - Test figure 10

Sydney

Submit

Figure 11 - Test figure 11


In or Out?

Want to *search again?*

Enter search location

Submit

I want food like... **Crock-pot Ravioli Casserole**



Prep Time not available

Cooking Time not available

Interested in a Cuisine? Choose an option below

mediterranean

european

italian

Recipe Instructions:

1. Brown ground beef with onion and garlic. Put in crock-pot and add sauce, tomatoes and seasonings. Cook for 6-7 hours on low.

2. Add the last 4 ingredients during the last 30 minutes of cooking and turn crock-pot to high. I added the fresh spinach, pasta, parmesan and 1 cup of the mozzarella and I mixed it all up really well. Then I added the last 1/2 cup of mozzarella to melt on the top.

Ingredients:

• 1 (15 oz.) can tomato sauce

• 1 can stewed tomatoes

Related Restaurant Details

Restaurants (Click to expand)

Restaurant Hubert

Cuisines: French, European

User Ratings: 4.9 - Excellent

Address: 15 Bligh Street, CBD, Sydney

Jamie's Italian

Cuisines: Italian

User Ratings: 4.1 - Very Good

Address: 107 Pitt Street, CBD, Sydney

Vapiano

Cuisines: Italian, Pizza

User Ratings: 4.0 - Very Good

Address: 77-79 Corner King And York Street, CBD, Sydney CBD

Est.

Cuisines: Modern Australian, European

User Ratings: 4.8 - Excellent

Address: Level 1, Establishment, 252 George Street, CBD, Sydney

The Italian Bowl

Displaying results for Sydney, NSW

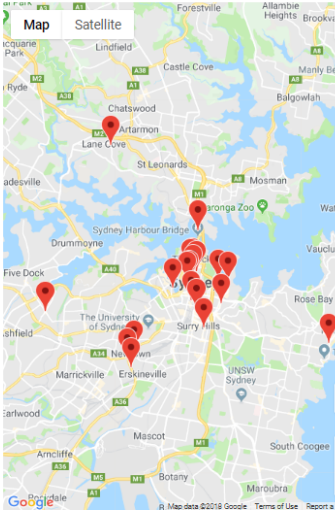


Figure 12 - Test figure 12



Figure 13 - Test figure 13

# No results found ; (

No location results found for **hdashkjsadhjkasd**    Want to [search again?](#)

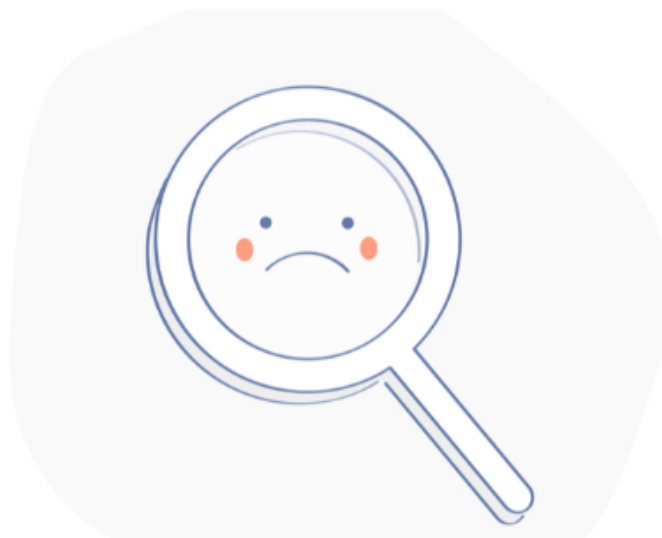


Figure 14 - Test figure 14

# In or Out?

*Want to search again?*

I like the look of... **One-Pan Pasta**



Prep Time: 15 mins

Cooking Time: 5 mins

**Recipe Instructions:**

## Ingredients:

- 2 sprigs basil, plus torn leaves for garnish
- Coarse salt and freshly ground pepper
- 4 cloves garlic, thinly sliced
- 12 ounces cherry or grape tomatoes, halved or quartered if large
- 12 ounces linguine
- 2 tablespoons extra-virgin olive oil, plus more for serving
- 1 onion, thinly sliced (about 2 cups)

[No cuisine tagged was provided for your search, want to search again?](#)

Figure 15 - Test figure 15

## Appendix C – Docker Install Guide

Using the Dockerfile structure shown in the Docker section, the following steps were used on an AWS EC2 Compute machine running Ubuntu OS v16.0.4 (LTS) in order to dockerise the application.

After booting up the instance, a new directory was created and navigated to, while all necessary source files were transferred over using FileZilla. After this docker was installed on the machine and the following commands were used to build a docker image of 'In or Out' and to run a container with this image:

Running docker build in the directory containing the Dockerfile:

```
sudo docker build -t foodapp .
```

Running the build app using docker run, matching the VM's exposed port(3000) to the app servers exposed port(80):

```
sudo docker run --name apptest -p 3000:80 -i -t foodapp
```

After running this command, the server successfully logged that it was running on port 80. Using the IPv4 address of the VM and the VM's expose port, the app was navigated to for use:

```
{IPv4ADDRESS:PORT} -> enter into web browser.
```

After testing the dockerised app, the final release was push to Dockerhub for future use.

The image can be pulled by entering:

```
sudo docker pull kurisu93/inorout
```