Luke Pritchard N9460250

CAB432 – Cloud computing

**Cusinear Finda**

Contents

[Introduction 1](#_Toc493541771)

[Use cases 2](#_Toc493541772)

[Services 2](#_Toc493541773)

[Technical Description 3](#_Toc493541774)

[Server-side usage 3](#_Toc493541775)

[Client-side usage 3](#_Toc493541776)

[Test Plan & Results 4](#_Toc493541777)

[Technical Summary 6](#_Toc493541778)

[Possible extensions 7](#_Toc493541779)

[Appendix 8](#_Toc493541780)

[Index page 8](#_Toc493541781)

[Index page part two 10](#_Toc493541782)

[Results Page 11](#_Toc493541783)

[Restaurant details 12](#_Toc493541784)

[Cuisine Images 13](#_Toc493541785)

[Contact page 14](#_Toc493541786)

[Contact form confirmation 15](#_Toc493541787)

[Error Handling 16](#_Toc493541788)

[References 17](#_Toc493541789)

# Introduction

Cusinear Finda is a web application designed to help users find the best and most suitable restraint near them. Users can input their location, whether it be city or street address, select a search radius and then select what type of meals and cuisines they are interested in.

The output produces an aesthetic and easy to interpret list of restaurants, sorted by either cost, distance from user or just random. The list of restaurants shows basic information such as its name, address and rating. From there, the user can select on a restaurant to view it in more depth, such as its graphical location via Google Maps and reviews provided by Zomato.

# Use cases

Cusinear Finda offers a broad range of uses for a variety of users. Some of them are as follows:

1. As a retail worker in a large shopping centre it is sometimes quite hard to decide what to eat on my lunch break, especially with so many options. It would be great if there was something to recommend a good place nearby so I don’t have to waste time on my lunch break looking for a good place.
2. The boys and I need some takeaway that’s near usso we can enjoy the game and have a snack. However poor Jimmy can’t eat spicy food so that’ll have to be excluded.
3. As it’s my 10th anniversary I’d like to take my wife somewhere with a very expensive, 5-star restaurant for dinner.

# Services

**Google Maps API**

The Google Maps API is the first API that is accessed in the web application. This service is used to query the inputted address into the system to return its longitude and latitude which is then parsed to the Zomato API to discover the local restaurants. The Google Maps API is also used on selection of the restaurant as it will provide a graphical map of where the restaurant is located.

**Zomato API**

Zomato is a database of eating establishments all across the world, whether it be formal restaurants to small town cafes. The Zomato API provides access to restaurants in specific vicinity’s, including their address, expense, reviews and cuisines served. This service Is the most used as compared to the other APIs.

The web app uses the Zomato API to locate nearby restaurants, with the specific cuisine and meal type. When the list of suitable restaurants is provided, on selection of the restaurants, the Zomato API is then used to provide additional information such as its address, reviews and expense.

**Twilio API**

The Twilio API is used an extension to the already existing web app, so to speak. The Twilio service is used as a contact form verification. As no web service is generally complete without a contact, if the user wishes to contact the owner of the web app by any means, Twilio will verify the submission of the contact form.

Upon submission of the contact form the Twilio API verifies the sent form by sending both the submitter and the receiver a text message to say the form has been successfully posted. This API is not a service used to support the return of restaurants however as an extension to API usage.

**Flickr API**

The Flickr service is used purely for image grabbing. In the web application, at the time users are selecting the cuisines they would like, they are also allowed to ‘preview’ the cuisines. This ‘preview’ extracts the top 10 images from Flickr under that cuisine. This allows the user to see what might be expected in their meal when choosing this cuisine. It is also helpful if the user has never heard of a cuisines and would like to know more about it.

# Technical Description

## Server-side usage

The web server prominently runs the Express JS web framework which sits atop of Node JS. Along with the Express JS framework, multiple JS packages from NPM are multiple node packages also used to maintain the service. These packages include:

* Express (web framework)
* Http (server-side http requests)
* @google/maps (NPM package for Google Maps API)
* Body-parser (NPM package for parsing middleware)
* Url (url parsing and resolution)
* Querystring (url percent-encoding on url string)
* Request (server-side http requests)
* Fs (input/output of html files)
* Https (server-side https requests)
* Twilio (NPM package for Twilio API)
* Zomato (NPM package for Zomato API)

## Client-side usage

With the application predominately being written on the server-side, there is little to do on the client side. HTML5 and CSS3 represents the user interface on the client side, even though this is later manipulated on the server side.

JavaScript has also been used on the client side of the application for form verification. If the form has an invalid parameter (as seen in Figure 1) then the user will not be allowed to post the form for the client side. Figure 1 also shows how client side programming has been used to create a step-by-step interface of how the application works.

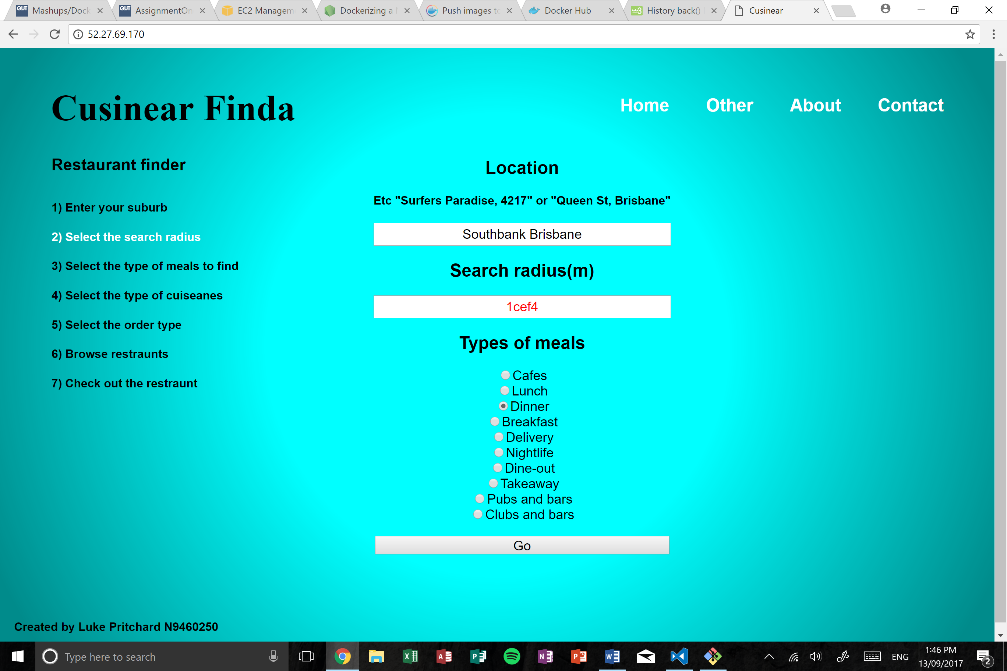


Figure - Client-side programming for form validation and step-by-step user interface

## Test Plan & Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Action | Expected | Appendix | Result |
| 1 | Hover over header links in top right hand corner | Mouse changes and text highlights black | Index page | Pass |
| 2 | On select of ‘Location’ | ‘Step 1’ highlights a white colour in left corner | Index page | Pass |
| 3 | On select of ‘Search radius’ | ‘Step 2’ highlights a white colour in left corner | Index page | Pass |
| 4 | On input of valid integer in ‘Search radius’ | Submit button becomes activated so form can be submitted | Index page | Pass |
| 5 | On input of an invalid integer in ‘Search radius’ | Search radius text highlights red, submit remains deactivated | Index page | Pass |
| 6 | On selection of a ‘cuisine type’ | ‘Step 3’ highlights a white colour in left corner | Index page | Pass |
| 7 | On submission of index form | Redirects to part two of the form with the parameters parsed in URL | Index page | Pass |
| 8 | On selection of ‘type of cuisine’ | ‘Step 4’ highlights a white colour in left corner | Index page part 2 | Pass |
| 9 | On selection of ‘sort by’ | ‘Step 5’ highlights a white colour in left corner | Index page part 2 | Pass |
| 10 | On select of a cuisine (not selecting radio box) | Redirects to new page with that cuisine showing images returned from Flickr | Index page part 2 | Pass |
| 11 | On load of the results page | ‘Step 6’ highlights a white colour in left corner | Results page | Pass |
| 12 | On load of the results page | Name, location and rating of top 20 results found are shown | Results page | Pass |
| 13 | On load of the results page with no results found | Application redirects to error page stating no results found | Results page | Pass |
| 14 | On hover of restaurant name | Restaurant name highlights in white and mouse changes to click mouse | Results page | Pass |
| 15 | On selection of restaurant name | Application redirects to a page where all the restraint information is shown | Results page | Pass |
| 16 | On load of restaurant page | ‘Step 7’ highlights a white colour in left corner | Restaurant Details | Pass |
| 17 | On load of restaurant page | Restaurant name, location, cost and rating is displayed | Restaurant Details | Pass |
| 18 | On load of restaurant page | Geographical location provided by Google Maps API is displayed | Restaurant Details | Pass |
| 19 | On load of restaurant page | Five most recent reviews of the restraint are displayed | Restaurant Details | Pass |
| 20 | On load of restaurant page with no reviews | The page will prompt no reviews have been submitted | Restaurant Details | Pass |
| 21 | On click of restaurant name | Redirects to original source on Zomato | Restaurant Details | Pass |
| 22 | On click of ‘about’ page | Page returns text of what the page is about | About page | Pass |
| 23 | On click of ‘other’ page | Page returns author, author image and subject headers of QUT | Other page | Pass |
| 24 | On click of ‘contact’ page | Page returns a basic contact form containing 4 elements. | Contact page | Pass |
| 25 | On input of invalid integer in ‘mobile’ | Mobile number will highlight red and form will not be able to submit | Contact page | Pass |
| 26 | On submission of contact form | Application redirects to a ‘Successful completion’ page | Contact page | Pass |
| 27 | On submission of contact form | Application sends a confirmation message to the host of the app | Contact page | Pass |
| 28 | On submission of contact form | Application sends confirmation message to the user of the app | Contact page | Pass |
| 29 | On request of an invalid URL | Application redirects to an error page stating ‘invalid html’ | Error handling | Pass |
| 30 | On submission of an invalid location in index | Application redirects to an error page stating, ‘bad location’ | Index page | Pass |

## Technical Summary

As stated previously, the application is mainly wrote using sever side JavaScript, with little client side. The server prominently runs the Express JS Web Framework along with the help of many other packages to maintain the service.

The application is designed to both be aesthetically pleasing to the user, yet still runs a reliable, speedy user interface that doesn’t stall when excessive data is requested. Formed by HTML, CSS, JavaScript and NodeJS, the application server is contained in a Docker container hosted by Amazon AWS on a Ubuntu Virtual Machine.

The application is predominately operated using the parsing of results through the URL, with the use of the ‘URL’ and ‘Querystring’ packages provided by NPM. For each time the user loads on to another page, the NodeJS file opens the default unfinished html (index.html) and depending on the page, will write the ending to this html file. This has been designed this way so repetitive code is not apparent in the NodeJS file.

The application will be deployed on an Ubuntu VM as provided by Amazon AWS. The application is built using a docker image and then deployed on this VM. The dockerfile is quite basic, preventing as many errors as possible. The dockerfile firstly copies all local files on to the virtual machine. Following this all the required packages for the server to run is installed. This is mainly the installation of NodeJS and npm. Following this, the apps dependent services such as express, Twilio and so on are installed. Below is the list of packages that are installed in this stage Finally, the applications ‘package.json’ is updated and the port is opened.

* Express (web framework)
* Http (server-side http requests)
* @google/maps (NPM package for Google Maps API)
* Body-parser (NPM package for parsing middleware)
* Url (url parsing and resolution)
* Querystring (url percent-encoding on url string)
* Request (server-side http requests)
* Fs (input/output of html files)
* Https (server-side https requests)
* Twilio (NPM package for Twilio API)
* Zomato (NPM package for Zomato API)

The Dockerfile exposes port 80 on the container. As this is the default port for all traffic, after specifying the servers IP address in the URL we will not need to follow this with the port allocation. Once the image is created, the container was developed. This container was demonized to ensure that it continually ran on the server.

# Possible extensions

As any application, improvements and extensions could always be inherited. If the allocated time to the web application was greater these extensions can definitely be implemented on a practical level. Some of the improvements and extensions are as follows:

**Links to website**

Right now, at the outcome of the web app the restaurant shows much information including address, a geographical location and even reviews. However, the one thing that is missing due to system limitations by the Zomato API is actual links to the restaurants websites.

However even though the Zomato API does not provide these address, the Google Search API could always be used to return the websites address and make it accessible by a click on this web application. Both the restaurant name and address would be queried, and if a link exists to that particular query the Google Search API would return it. This has not been implemented due to time constraints.

**Directions to the restaurant**

This was an extension that was considered in implementing to the application, if enough time was remainder. The Google Maps API provides a geographical search which can be used to, at the click of a button, get your exact coordinates. With these coordinates, the Google Maps API could find the exact distance and a set of directions to the chosen restaurant.

This extension could be implemented even without the user’s exact geographical position, however because some location inputs may be quite vague (such as ‘Brisbane, Australia’), it was decided not to implement this extension.

**Auto-fill location search**

This extension is a little tougher to implement than the others, but is still feasible. This extension looks into the problem of users putting in very vague locations and not getting the result they expected as it defaulted to another location elsewhere in the world.

As the user enters the address or location, the Google Maps API would be used to create a drop-down list of possible auto fill addresses. This extension would be the most time consuming to implement, thus was not considered as part of the application however still remains a possible extension.

# Appendix

## Index page



3

1

2

**(1) User application guide**

The user application guide is designed to help the user navigate the application. Whenever the user selects an input or checkbox, the user application guide will update and highlight where the user is at in the system. This guide remains stationary for the whole use of the application to allow ease of use.

**(2) Initial input fields**

The initial input fields gather the most valued information to operate the application – the user’s location, search radius and meal type. These values will then be passed on to the next page (see index page part two) which will contribute to the result of the application. The user will not be allowed to progress to the next page until all values are satisfied (see below for accepted inputs for each box).

Location – The location box is a text box which allows any characters to be inputted. The input box prompts for a location to be inputted. There is no specific type of location to be inputted, whether it be just the city or even an street address. The more descriptive the location, the more accurate the results will be that are soon to come.

On submission, the location is parsed through the Google Maps API. The parsing of the location returns the locations longitude and latitude, which is then later used to parse to the Zomato API to find restaurants within the radius of that longitude and latitude.

Search radius – The search radius is also an input text box, however unlike the location the search radius only allows positive integers. If the input is not a valid integer, the user will not be able to progress through the web application. The text box will also highlight in a red colour if the integer is not valid.

This validation is supported by JavaScript on the client side, validating both when the input changes or the user selects away from the input box.

Types of meals – The types of meals is a selection of radio buttons. Since the selection is made of radio buttons only one selection can be made. The default selection is dinner, however the user can obviously change this. The system has a default selection to prevent the user from not choosing a type of meal.

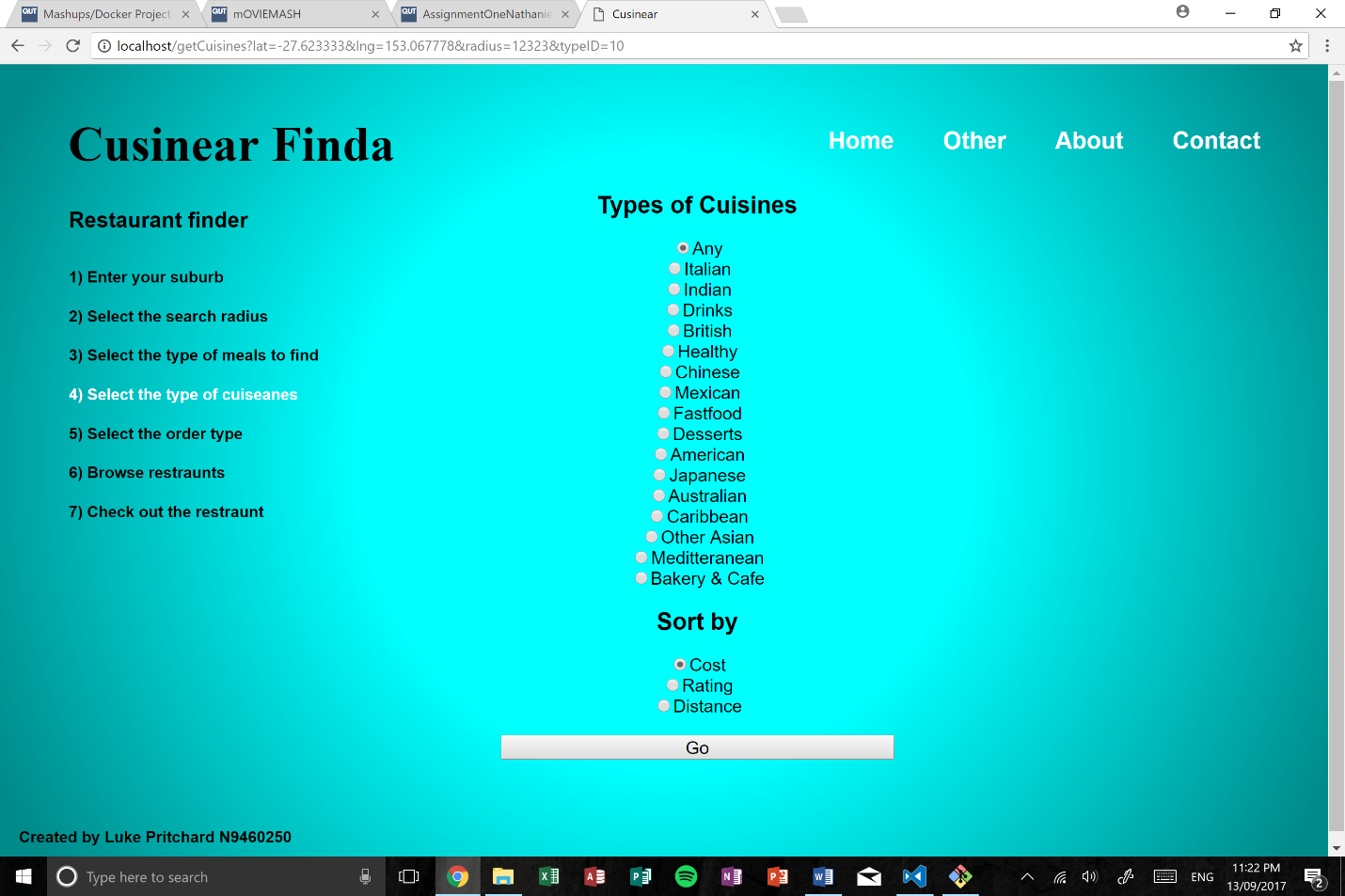
On validation the initial input fields from the server-side code, the ‘go’ submit button will switch from its initial deactivated stage to being activated. This allows the user to then progress through the web application.

**(3) User Headers**

The user headers are there to help the user navigate through the web application if necessary. There are four user headers:

1. Home – the landing page to the web application
2. Other – other details about the web application (mainly about the author and credits)
3. About – the page that explains the application in more depth
4. Contact – a contact page that allows users to contact the site administrator. See section ‘contact’ in the appendix for more details of this page and how it implements the Twilio API with SMS validation.

## Index page part two



1

2

**(1) Parsed URL**

After submission of the first page, the web application will redirect to last part of the form. The details from the previous page will be parsed in the URL. The URL displays the information as follows:

1. Lat: Is the latitude of the user’s location cited from the Google Maps API
2. Lng: Is the longitude of the user’s location cited from the Google Maps API
3. Radius: Is the search radius as per the users request on the previous page
4. typeID: Is the Zomato unique identification number for the chosen meal type

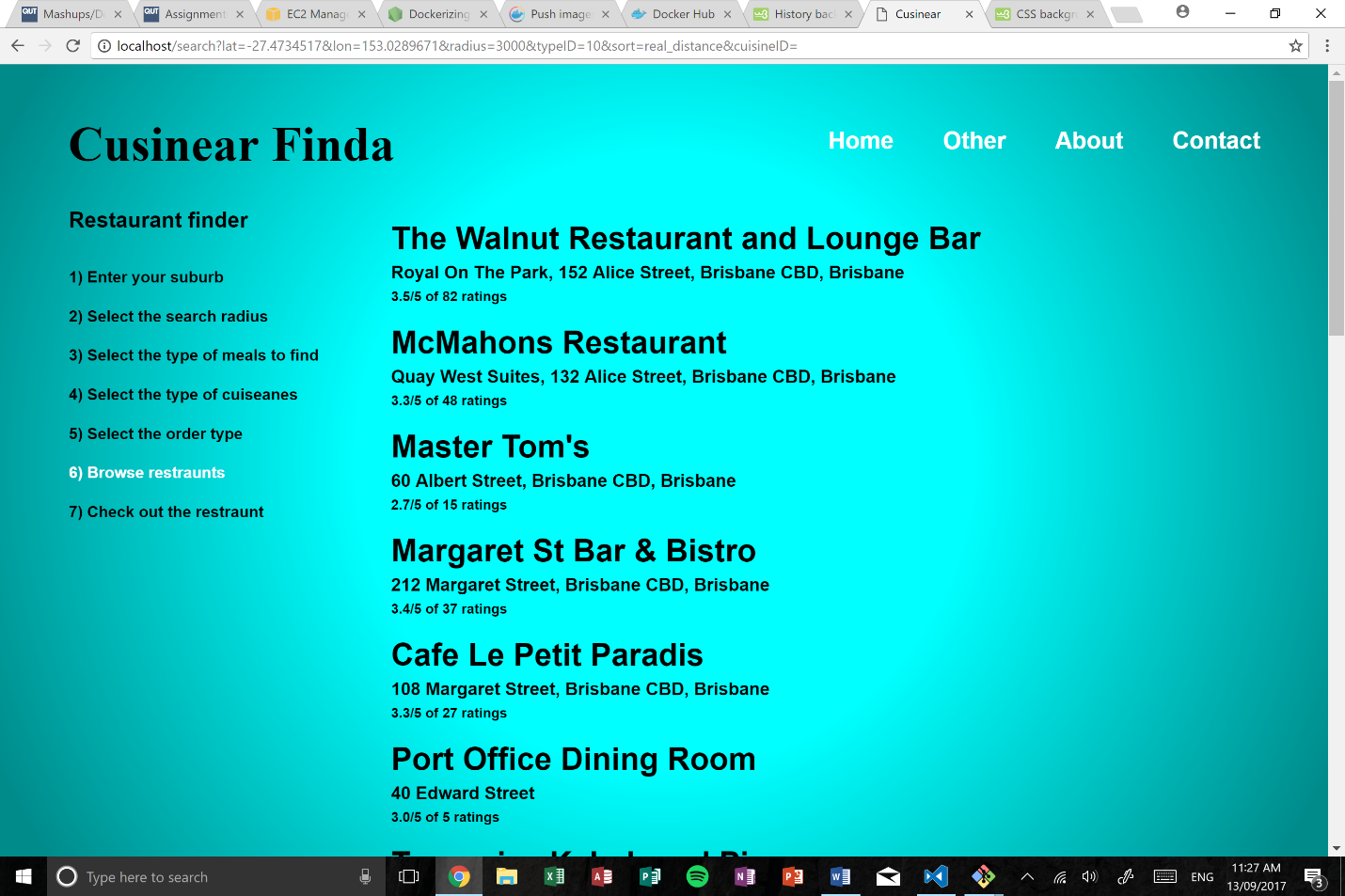
**(2) Secondary input fields**

The secondary input fields are just the second part to the web application which requires user input data; they type of cuisines the user would like and the ‘sort by’.

Type of cuisines – the cuisine types is a radio button input of a various types of cuisines that are found locally. Users can either select one specific cuisine or all of them. The default selection is ‘any’. When a user hovers over the cuisine it will highlight, prompting the user to click it. On click of the cuisine the user will be redirected to another page that displays the top 10 images returned from the Flickr API about that cuisine. This is designed to give users an idea of what they might expect in that cuisine type, especially if they have not heard of it before. Please see the ‘Cuisine images’ section below for a preview.

Sort by – The sort by is pretty self-explanatory. There are three selections to sort the results from: cost, rating or distance. The default selection is cost. Like the type of cuisines, the sort by is a radio button list which only allows a user to make one selection.

## Results Page



1

2

**(1) Parsed URL**

After submission of the previous page, the web application will redirect to the results page. The details from the previous page will be parsed in the URL, alongside the latitude, longitude, radius and type ID from the previous URL. The URL will additionally parse the additional information

1. sort: the Zomato unique ID for sort by.
2. cuisineID: the Zomato unique ID for cuisine types.

**(2) Results**

This page is the main page to the web application, it is the page that retrieves the results from the Zomato API and displays them in an aesthetic manner. So that the page is not overloaded, only the top 20 results will be displayed. As this is only the show of results, not all the information on each resulted restaurant is shown, only its vital information such as the restaurant’s name, address and rating.

When the user hovers over the restaurant name it will highlight to prompt the user to select it. If the user selects the restaurant they will be redirected to a page where all the restaurants information will be displayed. More about this can be found below in the section of ‘Restaurant details’

## Restaurant details



4

3

2

1

**(1)** **Parsed URL**

The URL parses only the Zomato unique restaurant ID. This ID is then queried back to Zomato to retrieve all that specific restaurants details as shown below

**(2)** **Main details**

The main details of the restaurant are retrieved from the Zomato API. Within these main details, several pieces of information are gathered from the API.

1. Restaurant name – The trading name of the restaurant
2. Restaurant address – The physical address of the restaurant
3. Cost – The average cost for 2 people to dine at this restaurant
4. Summary of votes – A summary of the number of votes, rating and star-score.

**(3) Geographical location**

Using the details provided from the main details section, these details can be queried using the Google Maps API. This API provides a service where users can interactively view the physical address of the restaurants location. This increases ease of use to the web application and also allows users to get a gauge of where the restaurant is and how far from them.

**(4) Reviews**

From the same Zomato API we are able to produce reviews of the restaurant. For ease of use, only the five most recent reviews are shown.

## Cuisine Images

This page of the web application produces display images of a chosen cuisine. The images are retrieved through the Flickr API. So that the page is not overloaded with data, only the top 10 results from the Flickr API are shown.

The user can also click on the image to view the original sourced image from the Flickr API.

## Contact page



1

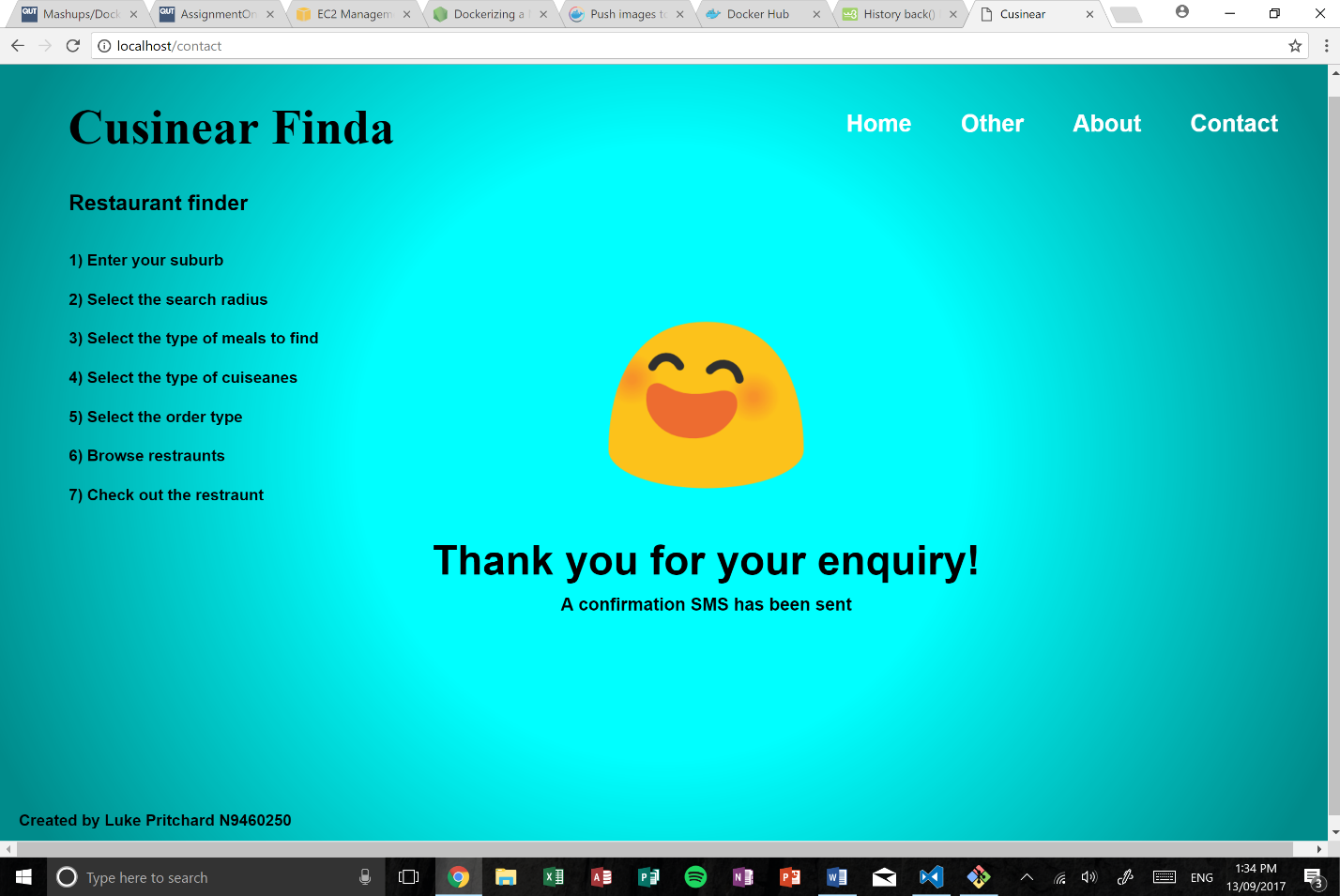
**(1) Contact form**

The contact form is a basic form that will be found on most web application. The contact form here a more unique from others as it carries a SMS confirmation supported by the Twilio API. There are four main parts to the form:

1. Name – the users name, or preferred name. Can be any character
2. Mobile number – the users contact number, can only be an integer
3. Query type – the summary type of the users message
4. Message – the users message, in more depth

The form is validated using client-side programming, if the parameters of the form are not adequate the submit button will not be active. On successful submission of the contact form the user will be directed to a page which confirms their submission and sends the confirmation SMS message. Please see the following section for this.

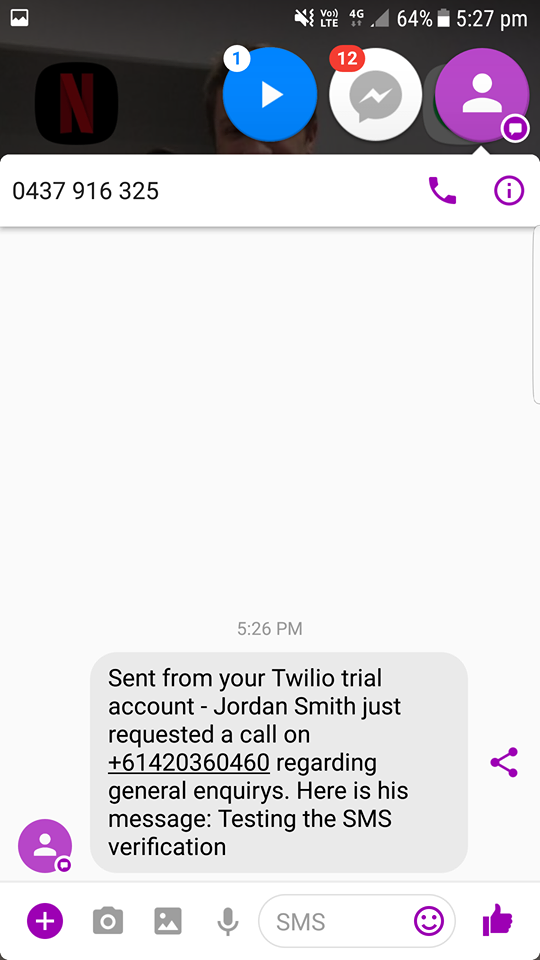
## Contact form confirmation



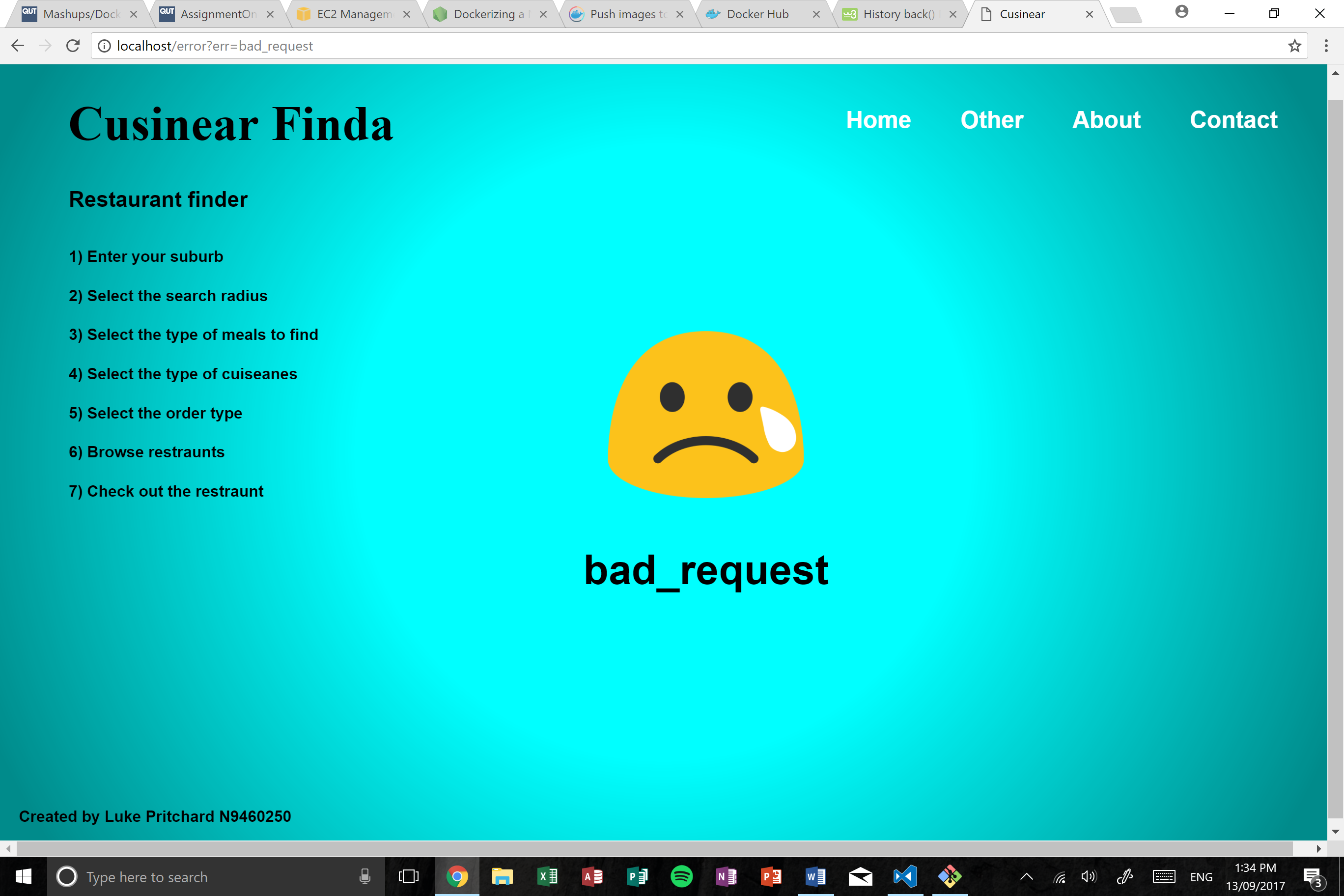
On completion of the contact form and if successful, the app will redirect to a conformation page. If an error occurs with the submission then the app will redirect to the error page, explain the error as shown in the next section.

Using the Twilio API the application uses SMS verification. On successful completion two SMS messages will be sent – One to the user and the other to the applications owner as a heads up.

The first message sent goes the user. The message is just a confirmation stating ‘*Thank you for your response, we aim to respond in 2 business days’*. The second message sends itself to the owner of the application with the parameters that the app user inputted. A copy of this message can be found in the figure to the right.



## Error Handling



The error handling page is a page dedicated to displaying the output caused by errors in the web application. For example, if the user attempted to input an invalid URL, the app would redirect the user to the error page where the error displayed would be ‘bad\_request’.

Other error handling such as invalid form submission, invalid location or if the application returns a 0-result page, then the application would redirect to the error handling page.

## References

*Docker, 2017, What is Docker?, accessed 24 of August 2017, from*

*https://www.docker.com/whatisdocker*

*Zomato, 2017, Zomato API, accessed 27 of August 2017, from*

*https://developers.zomato.com/api*

*Google, 2015, Google geocoding API, accessed 27 of August 2017, from*

*https://developers.google.com/maps/documentation/geocoding/intro*

*Google, 2015, Google maps JavaScript API, accessed 27 of August 2017, from*

*https://developers.google.com/maps/documentation/javascript/*

*Flickr, 2017, Flickr image search API, accessed 29 of August 2017, from*

*https://www.flickr.com/services/api/*

*Twilio, 2017, Twilio SMS API, accessed 3 of September 2017, from*

*https://www.twilio.com/docs/api/rest*