Software Engineering Group Project

COMP2002 / G52GRP: Final Report

Project information:

Project title: Novel Ways of Using NCC Open Data Resources

Project sponsor: Peer-Olaf Siebers

Academic supervisor: Peer-Olaf Siebers

Team information:

Team number: 16

(Saamiya, Aqeel, 20378737, psysa11)

(Alice, Thornton, 20287635, psyat9)

(David, Laming, 20357978, psydl4)

(Maria, Constantin, 20370023, psymc8)

(Kuang, Li, 20377034, psykl7)

(Rahul, Ravi, 20389281, psyrr5)

(Roan, Hutcheson, 20361598, psyrh11)

Documentation (links):

Trello board:

- Main Report: https://trello.com/b/OKBlcCsr/main-report
- Software Manual: https://trello.com/b/sqU74ek5/software-manual
- User Manual: https://trello.com/b/vqpLg9nZ/user-manual

Code repository: https://projects.cs.nott.ac.uk/comp2002/2022-2023/team16_project

Document repository: https://projects.cs.nott.ac.uk/comp2002/2022-2023/team16_project/-/tree/main/Documentation

NCC Open Data: https://www.opendatanottingham.org.uk/

Software Manual

Introduction

The NCC Open Data for Nottingham is a valuable resource with an abundance of underrepresented datasets and vital information about the city. Our project has made the most of every resource available to us, as demonstrated in the following section. Our team was tasked with creating a robust system that could consume real-time datasets and produce a mobile-friendly app as mentioned in the project brief. This requirement led us to explore our own creative ideas, resulting in an app with various entry modes, real-time data updates using a pipeline, and additional features obtained through web scraping of other datasets to enhance the existing app.

The app we have uncovered holds immense potential, and our project aims to efficiently and thoroughly explore its capabilities.

As a part of a project, we wanted to provide a way for users to navigate around Nottingham and see what is in their immediate area. This was developed into a map page to see what is around the user and a directions page to navigate between 2 points. We also wanted a way for users to learn more about the city in a fun and interactive way. To fulfil this, we created various quizzes, a word search game and a memory match game. Making the system usable by a wide variety of users was important to use so we implemented 2 separate user modes to change parts of the app such as colours to have higher contrast in the kid's mode map. Some other features we implemented were a login system, saving favourite locations on the map, and a page to search for places to eat.

Installation Guide

Cloning Project

Creates a linked copy of the Git repository that will continue to synchronise with the target repository

1. Open the preferred terminal and navigate to the desired destination for the project

2. Type the following command

```
git clone https://projects.cs.nott.ac.uk/comp2002/2022-2023/team16_project.git
```

Download Source Code Zip

1. Download the source code as a zip file

2. Extract the zip to desired destination

Code Architecture

Firebase Authentication

Firebase Authentication documentation: [Firebase Authentication (google.com)]

The signup and login credentials of the user are stored using the Firebase Authentication service, using email and password based authentication.

To login, users authenticate with Firebase using their email and password.

Actor	General User
Use Case	Register
Description	For users to register an account using an email and password
Precondition	User in on signup page
Postcondition	If the use case was successful, the user is now registered onto the Firebase Authentication system. If not, the system state is unchanged.
Primary Path	1) System displays signup page 2) User enters email and password, and selects a mode for the app 3) User clicks "SIGN UP" button 4) System verifies that all textboxes have been filled out and are valid 5) System successfully registers user in Firebase Authentication 5) System displays home page based on user's mode
Alternative Paths	4) User does not fill in all fields or fields are invalid - System displays error message - Use case resumes at step 1

Actor	General User
Use Case	Login
Description	For users to login to the system using their email and password
Precondition	User in on signup page
Postcondition	If the use case was successful, the user is now logged onto the app. If not, the system state is unchanged.

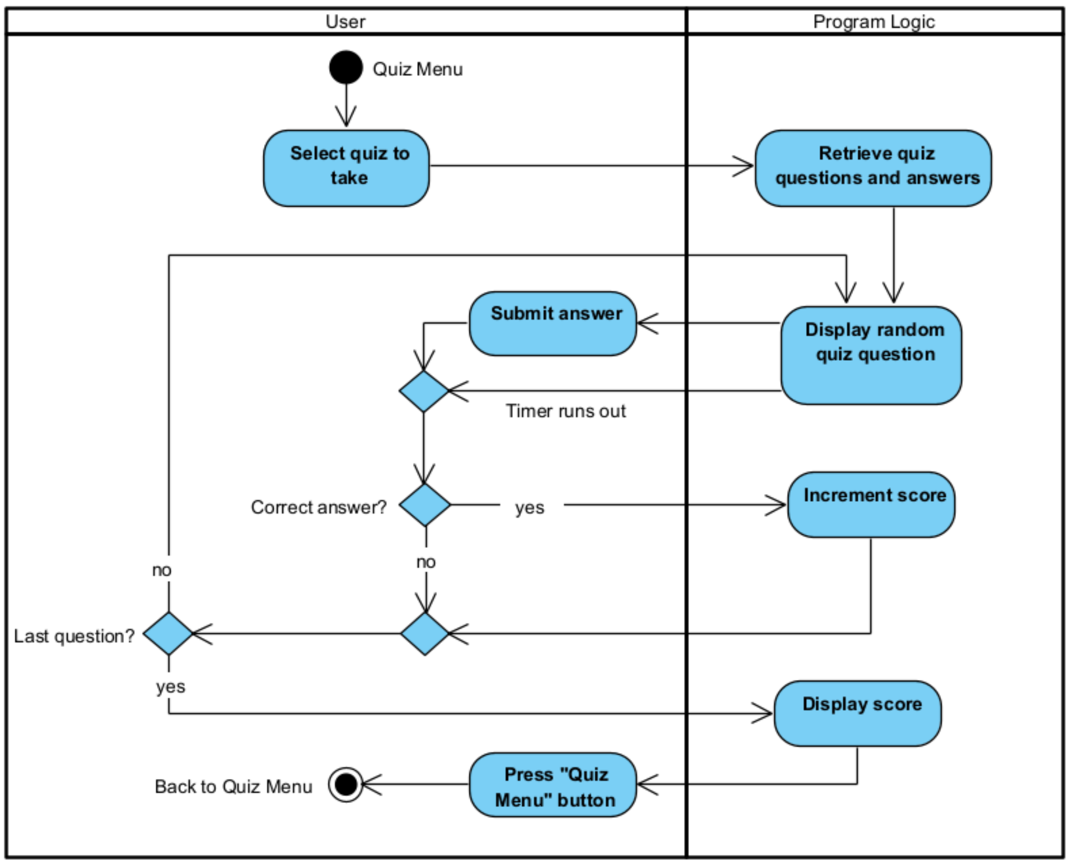
Primary Path	1) System displays login page 2) User enters email and password 3) User clicks "LOG IN" button 4) System validates user's email and password and logs them into the system 5) System displays home page based on user's mode
Alternative Paths	4) User enters invalid email / password - System notifies user that email / password is invalid - Use case resumes at step 1

System does not specify which of either the email or password is incorrect, this is to prevent brute force attacks if the email is confirmed to be registered in the system.

Quizzes

For each quiz category the question data and the answer data (with corresponding bool relating to its correctness) is stored in a local JSON file (e.g. *"landmarks.json"*).

This data is retrieved within the corresponding HTML page for the selected quiz (e.g. *"landmarks.html"*) via a jQuery AJAX call. The *"quiz.js"* script then utilises this data to initialise the quiz.



Activity diagram for quiz functionality

Upon window load, the *startTimer()* function is called to initialise the timer for the quiz.

The *startTimer()* function uses DOM manipulation to change the number representing the seconds remaining for the user to answer the question. Every second the timer is decremented, and if the timer is less than zero the *next_question()* function is called.

The *update_question(id)* function retrieves a random question from a shuffled question array, then updates the HTML using DOM manipulation to display the question. The potential answers are shuffled and displayed in a similar manner. The score displayed is also updated to represent the current score of the user.

The answer options have an attached event listener to check for mouse clicks. Once a user selects an option the corresponding boolean value linked to the answer is stored (i.e. whether the answer is right or wrong) and the *next_question()* function is called.

The *next_question()* function increments the user's score if the correct answer is selected. If the number of questions is less than the total then *resetTimer()* is called and the *update_question(id)* function is called with an incremented id (represents the shuffled question array index). Else, if the number of questions exceeds the total then the score as well as a button leading to the quiz menu is displayed via DOM manipulation.

The *resetTimer()* function resets the countdown timer for the quiz, i.e. set it to 10 to allow the user 10 seconds to answer the question.

Memory match game

Actor	Player of Game
Use Case	“Play Memory Match game”
Primary Path	1) Player Opens Game Page 2) Player Clicks on “Memory Match Game” 3) Player clicks on first card 4) Player clicks on second card 5) Match is made and player scores one point 6) All matches are made and game ends before time ends 7) The player is shown a page with their full score
Alternative Path(s)	**1.1 Match is not made** 5) Cards are flipped 6) Timer Runs out 7) The player is shown a page with a score less than the full amount (4 for kids and 6 for normal)

The code's structure is rooted in the use case illustrated above. It seamlessly combines HTML, CSS, and Javascript, with the latter being the driving force that manipulates the objects on the page based on user actions. The Javascript code is extensively commented to ensure easy comprehension of its functionality. To delve into the specifics, the bottom timer employs the "startTimer" function, which is also reused in the quiz and word search. This function relies on intervals and parsers to format the time display. The Memory Match game comprises several auxiliary functions. The "flipCard" function, executed upon clicking the first and second cards, reveals the images underneath and subsequently triggers the "checkForMatch" function. Step 5 involves checking if the card IDs match, and if so, calling the "disableCards" function to remove the event listeners from the cards, mark them as matched, increase the match count, end the game when all matches are made, and reset the board's state. In contrast, if the cards do not match, the "unflipCards" function is called, which removes the "flip" attribute, changes the displayed images back to their original state, and calls the "resetBoard" function. The game ends when either the timer runs out or all matches are made. At this point, all card attributes are removed, and the total score is displayed, along with a "game over" message and a "game menu" button. Finally, to add an element of randomness to the game, the card IDs are randomised using an array of numbers at the program's outset.

Places to eat

Actor	A User in Normal Mode
Use Case	“Using the places to eat page(filtering, searching and scrolling)”
Primary Path	1) User clicks on places to eat page 2) User scrolls through the available places to eat on the page 3) User searches on page for a restaurant 4) The searched for item is loaded on page 5) User clicks on a cuisine filter 6) The selected cuisine of restaurants come up on page 7)
Alternative Path(s)	**1.1 When a user searches for item that is not there :** 5) The page loads the message “sorry no contents found”

The structure of the code is designed according to the use case illustrated in Figure ?. It comprises three components: retrieving a list of available restaurants for the user to browse through, applying filters to modify the displayed content, and a search bar that enables users to locate restaurants containing specific search terms. The initial step involves using JavaScript to access Firebase to load the content onto the screen. The code first checks if the user is authenticated, then proceeds to extract data from each entry key, such as cuisine, image source, and title, which are stored as variables. Next, buttons are added to the "food-list" empty list on the HTML page using JavaScript and styled with CSS which are then displayed as shown in step 2. The second part of the code involves cloning the current "food-list" to create a new list for filtering. The original "food-list" is then cleared, and the cloned list is searched for specific keywords, which are then appended to the list according to whatever cuisine was chosen in step 5. Lastly, the search bar utilises similar code but searches for specific search terms entered in the search bar to identify relevant items which is done in step 3.

As of now, the code has not been extended to accommodate the alternative path 1.1

Lightbulb

Actor	Normal Mode User or Kids Mode User
Use case	“Click on Light Bulb”

Primary Path	1) User clicks on landmark 2) User clicks on lightbulb 3) Use views the fun fact that shows up

The code follows the use case presented in the figure above, where a lightbulb is added to the landmark blurb using a CSS style that includes pictures of lightbulbs. The lightbulb includes an "onclick" function that is triggered in step 2. This function uses different CSS styles to "toggleclass" the lightbulb when clicked, making it appear as though it is "lighting up". When the bulb is "lit up", the landmark blurb displays a fun fact, as illustrated in step 3. This is accomplished by selecting the "fun fact" element from the HTML description and setting it to empty when the lightbulb is not lit. When the lightbulb is lit, the function searches through the firebase of landmarks to check if a fun fact is available. If a fun fact is present, it is displayed on the page.

Favoriting Landmarks

The HTML for each landmark's information popup is dynamically added upon the initialisation of the map in the *"kids-map.js"* script within the *createLandmarkPins()* function. The heart icon image displayed (either empty or filled-in) is determined by whether the landmark ID is present in the specific user's favourites branch within the Firebase Realtime database.

If the landmark is favorited then the class "heart-full" is attached to the HTML favourite container div element, else the class "heart-empty" is attached. The heart image is displayed within the div element via the CSS background image within *"favourite-style.css"* corresponding to the attached class.

Actor	General User
Use Case	Add Landmark to Favourites
Description	For users to add landmarks to their favourites list
Primary Path	1) User navigates to map page 2) User clicks on landmark icon to open information popup 3) User clicks on empty heart icon 4) System stores landmark ID in user's favourites branch in the database 5) Heart icon changes to a filled-in heart
Alternative Paths	3) User clicks on full heart icon - System removes landmark ID from user’s favourites branch in the database - Heart icon changes to an empty heart

The HTML for the favourite container div contains an “onclick” attribute which transfers control to the *favourite()* function within *"kids-map.js"* (step 3). This function adds the selected location's key (corresponding to the key within the landmark_buildings branch) to the user's favourites branch in the database (step 4). The "heart-empty" and "heart-full" classes are then toggled to switch the heart image (step 5).

Favourites Page

Actor	Users in both modes
Use Case	View your favourites
Description	View your favourite locations on the favourites page
Primary Path	1) Click on the profile icon 2) Select “Favourites” from the drop-down menu 3) View the with your favourite locations

The HTML page includes a single div that shows the list of favourite locations. To create the list, the "favourites.js" script is used, which connects to the database and retrieves the ID of all favourite locations from the currently logged-in user. In order to find any matches, the script iterates through the database and creates a dynamic list of the user’s favourite locations.

Map Page

Actor	Normal Mode User or Kids Mode User
Use case	“Find places of interest ”
Primary Path	1. User clicks opens map screen 2. User finds marker related to a place of interest 3. User clicks marker to read name and any additional info
Alternative paths	1.5)User filters out places of interest they do not want to see on the map

The html for the map page is either in src/main/resources/static/KidsModeScreens/map.html or src/main/resources/static/normalModeScreens/mapNormal.html. These contain various div elements that make up the page. The ‘map’ div contains the google maps js map object. The ‘legend’ div contains images of the types of points of interest that the user can toggle on and off to be shown on the map. The state of this toggle is shown by the image being slightly transparent. The ‘centre location’ div is a button that pans the map screen to the user's location if the location is enabled. The ‘directions’ div is a button that opens up the directions page.

The javascript for the map page (in both kids mode and normal mode) is found at src/main/resources/static/KidsModeScreens/kids-map.js.
The main functions to note are:

▪ initMap() - Initialises the map screen and serves as the main function.

▪ createPins(), createLandmarkPins(), createTramPins() - Functions that get relevant data based on parameters from firebase and return an array of google maps markers. The pins from createLandmarkPins() have extra text, the favourites feature and a fun fact in a related infowindow. createTramPins() creates pins for trams and buses. These pins then populate the map.

▪ createUserLocationPin() - Uses the browsers geolocation api to get the users location and create a pin if enabled.

Transport Links

Actor	Normal Mode User
Use case	“Find modes of public transport ”
Primary Path	1. User clicks opens map screen 2. User enables bus/ tram filter 3. User finds a mode of transport near them 4. User looks at the top of the screen to see how far away it is
Alternative paths	N/A

When the user is on the map page, pins can be shown that represent locations to catch public transport. The methods of transport themselves are only loaded within 2km of the user's location in order to reduce loading times. They are also filtered out by default to further reduce loading times. The ‘info div’ in src/main/resources/static/normalModeScreens/mapNormal.html is a text box at the top of the screen that says where the closest tram and bus stops are if the location is enabled.

The javascript for this feature (only in normal mode) is found at src/main/resources/static/KidsModeScreens/kids-map.js.
The main function to note is:

▪ updateShownMarkersLocation() - Calculates which markers to display based on the user's location. Also creates the info box at the top of the page and displays it with the closest transport links. This is re-run every 60 seconds as if the user is moving through the city while using the app, the markers will need to update to the new location.

Directions Page and Map Link

Actor	Normal Mode User
Use case	“Click on Pin to go to Directions Page ”
Primary Path	1. User clicks “directions” on the map screen. 2. User is taken to the directions.html page. 3. User enters the origin location into the text box. 4. User enters the destination location into a text box.

	5. User Clicks calculate button.
Alternative paths	N/A

When the user is on the map page, a pin button is loaded on the bottom left of the map overlay. Here the user can click on the pin and get redirected to the directions page, which is only accessible through this button. On this page the user is presented with a smaller map, and above it two text boxes and two buttons. The respectively labelled buttons can take you back to the map screen or calculate and display the car directions between the two locations input into the text boxes.

The javascript for this feature (only in normal mode) is found at src/main/resources/static/KidsModeScreens/kids-map.js. The main function to note is:
▪ initMap() - Uses innerHTML and javascript in order to create a button in the bottom left of the map overlay which will redirect the user to the directions page.

src/main/resources/static/KidsModeScreens/dirPlan.js. The main functions to note are:

▪ initMap() - Uses javascript to first create new google directions and service renderers, then it creates the parameters for the map to be displayed at the bottom of the screen. The calculateAndDisplayRoute() function is then called in order to calculate the directions for the user input locations. Finally, the event listeners are applied to the buttons so that they can be interactable.

▪ calculateAndDisplayRoute(directionsService, directionsRenderer) - Uses javascript to retrieve the users input of the origin and destination locations. The google place API is then used on the inputs to find the coordinates of the users inputs. Those latitude and longitude values are then used to show the driving directions for the user on the map at the bottom of the screen.

Maintainability and Extensibility

Making the web application manageable and expandable was one of our key objectives. To accomplish this, we divided appropriate code into smaller, more reusable modules (i.e. “firebase.js”, “logout.js”, etc.). This approach not only made the code more organised but also made it easier to maintain and debug.

In addition to modularizing the code, our approach to organising the data enables a top-down methodology when it comes to the Firebase database. We have achieved this by breaking down the system into its subsystems, consisting of the user data, transportation links and landmark information. By using this approach, we have ensured that the database meets the requirements of the system while also ensuring that the components are well-structured and easily manageable.

Furthermore, we heavily documented and commented on the code base. This further reinforces our objectives by facilitating maintenance and improving code readability. In addition to this, a coding convention has been imposed to preserve consistency across the entire codebase. This helped in making the code more readable and easier to understand.

Additionally, we adapted the SOLID principles, which is a set of design principles aimed at making software more maintainable, scalable, and easy to understand, to make them suitable for front-end development. The SOLID principles helped to ensure that the code was modular, easily extensible, and had a low coupling between different components by ensuring single-responsibility scripts and interface segregation, demonstrated so in the implementation of key features such as the map, quizzes and games.

Data Storage

Database Setup

This section of the manual briefly explains the relevance of the initial setup for the Firebase database. Each CSV file from the NCC Open Data platform was cleaned and processed. After processing, a javascript program was written to upload them to the Firebase real-time database. This assisted in the initial testing of the application and the development of the data pipeline.

Pseudocode

```
Initialise connection to firebase
dataArray <- data for 1 table // stored in JSON format
ref <- reference // reference string for the data table

loop through the dataArray
      push data to the firebase
end loop
```

Setup Details

Language: Javascript (Node.js v18.14.2)

Dependencies: initializeApp from “firebase/app”
getDatabase, push, ref from “firebase/database”

Variables: const firebaseConfig = {..};
const app = initializeApp {firebaseConfig};
const db = getDatabase(app);

Places To Eat (Server Side)

The server side code for the place-to-eat prepares the information about restaurants around a certain user location to be displayed on the application. The information is extracted from the web using the Yelp API. Once extracted, the relevant data is then pushed to the Firebase database for further processing.

Overall Pseudocode (describes the function M2 partially)

```
location <- [latitude, longitude] // or name a city (E.g., Nottingham)
numRes <- y // set a number of restaurants (max 50)
radius <- x // set a radius of x metres
```

```
rawData <- get from url // get request to a web url
data <- load data // convert to json
data2Push <- convertData // prepare to push to the Firebase
push2Firebase(data2Push) // push data to Firebase database
```

Setup Details

Language: Python 3 (3.11.2)

Dependencies: import json, requests, firebase_admin
from firebase_admin import credentials, db

Variables: credentialsObject = credentials.certificate(path_to_certificate)
defaultDatabase = firebase_admin.initializeApp(credentialsObject,'databaseURL': url)
url = yelp_api_url , api_key = yelp_api_key
headers = { 'Authorization': 'Bearer' {api_key} }

Functions

Name	Input	Output	Pseudocode

filter	dict	dict	`If input dictionary is empty, then return`
			`resDict <- initialise new, empty dictionary`
			`Copy as is : name, image url, latitude,longitude`
			`Fix duplicates in cuisine data`
			`return new dictionary`

convert2JSON	dict	[list]	`Initialise new empty list`
			`Iterate through input dictionary`
			`      add each element from input dict to list`
			`return list`

push2Firebase	[dict],str	None	`ref <- set Firebase reference`
			`If db contains nothing at reference 'ref', then`
			`        Iterate and add json data items`

M2 (Main Function , pseudocode provided above)
Input(s) : str, list of float or str , int, int, str
Output(s) : None

Data Pipeline

The data pipeline is a server-side system that regularly looks for changes in the NCC Open Data platform and updates data accordingly to ensure that users are viewing the latest data.
It comprises a series of functions that depend on each other to allow this to work as expected.

Setup Details

Language: Python 3 (3.11.2)

Dependencies: import csv, requests, firebase_admin
from firebase_admin import credentials, db

```
Variables: credentialsObject = credentials.certificate(path_to_certificate)
           defaultDatabase = firebase_admin.initializeApp(credentialsObject,’databaseURL’: url)
           dbTables = [..] # database tables to update
           schemaDict = {...} # schema dictionary to translate raw and processed schema
           databaseDict = {...} # dictionary of urls to the NCC Open Data platform
```

Function Table

Name	Input(s)	Output(s)
Sub-Helper		
cleanFile	[list]	[list]
getSchema	[list]	list
readNClean	str	[list]
findCombineDiff	[list],[list]	[list]
convert2JSON	[list]	str
getDataFromDict	dict	list
Helper		
getData	str,dict	[list]
push2Firebase	str	None
removeFromFirebase	str	[list]
updateData	[dict],[dict],str	None
Main		
M1	str,dict,dict	None

Pseudocode

Helper Functions

cleanFile
```
loop through input data
      if data item has a space or is empty, then remove item
      end if
end loop
return data
```

getSchema
```
return data item at index 0
```

readNClean
```
fileData <- read input file
cleanedData <- apply cleanFile to fileData

if schema of fileData is equal to schema of cleanedData, then return cleanedData
else return fileData
end if
```

findCombineDiff
```
d1NotInd2 <- initialise new empty list // data found in d1 but not in d2
d2NotInd1 <- initialise new empty list // data found in d2 but not in d1

loop through all items in dataset1
      if item is not in dataset2, then, add item to d1NotInd2
      end if
end loop
loop through all items in dataset2
      if item is not in dataset1, then, add item to d2NotInd1
      end if
end loop

diff <- d1NotInd2 // avoid duplication of difference data
loop through all items in d2NotInd1
      if item is not in diff, then add the item to diff
      end if
end loop

result <- dataset2
loop through all items in diff
      if the item is not in result, then add item to result
      end if
end loop
return result
```

convert2JSON
```
Initialise new empty list
Iterate through input dictionary
      add each element from input dict to list
return list
```

getDataFromDict

```
resultantList <- initialise new empty list
loop through all items in the input dictionary
      Add item value to the emptyList
end loop
return resultantlist
```

getData

```
url <- dictionary value at input keyword
data <- get text data from url
cleanedData <- clean data using csv operations
data <- convert cleanedData into a list
return data
```

push2Firebase

```
ref <- set Firebase reference
If db contains nothing at reference ‘ref’, then
        Iterate and add json data items
```

removeFromFirebase

```
ref <- set Firebase reference
data <- empty list
if the database table doesn't exist at ref, then return error
else:
      data <- get schema and add it to data
      data <- add rest of the data from Firebase
      delete ref // removes the table from Firebase
end if
return data
```

updateData

```
apply removeFromFirebase on the input title
apply push2Firebase on the input title with input json data
```

M1: Main Function

M1

```
oldData <- apply removeFromFirebase on input database table
oldDataSchema <- apply getSchema on oldData

if oldData is empty, then display error and return None

newRawData <- apply getData on input database table (with reference to the database
dictionary)

schemaForFilter <- translate newRawData schema using oldDataSchema against the
schemaDict

newFilteredData <- apply schema filter on newRawData

newData <- apply findCombineDiff on newFilteredData and oldData

remove any empty elements from newData
insert oldDataSchema into newData at index 0

data2Push <- apply convert2JSON on newData
apply push2Firebase on data2Push
```

Testing

Cypress

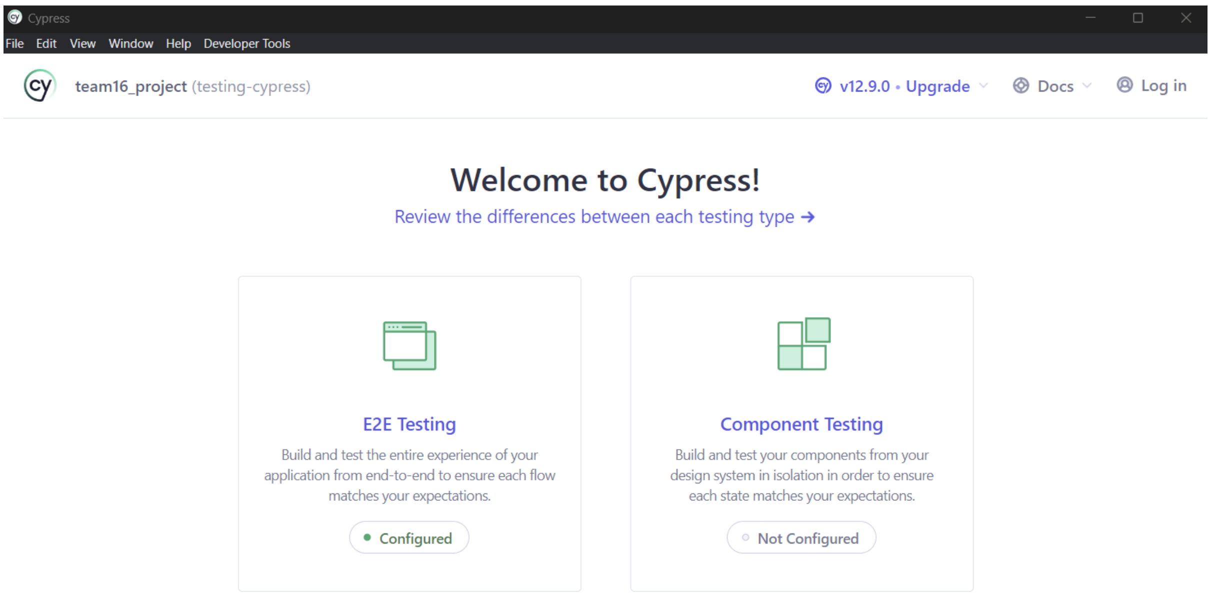
Cypress is a JavaScript-based end-to-end testing framework.
Utilised Cypress to write end-to-end (E2E) tests which can be tested across multiple browsers.
These E2E tests visit the application in a browser and perform actions via the UI.

Running Tests

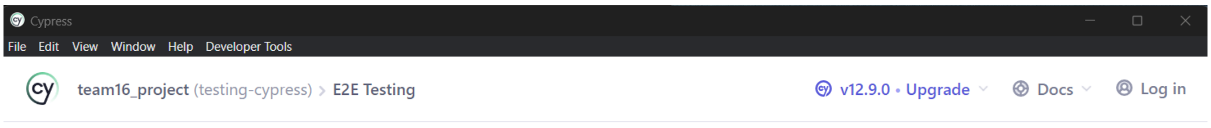
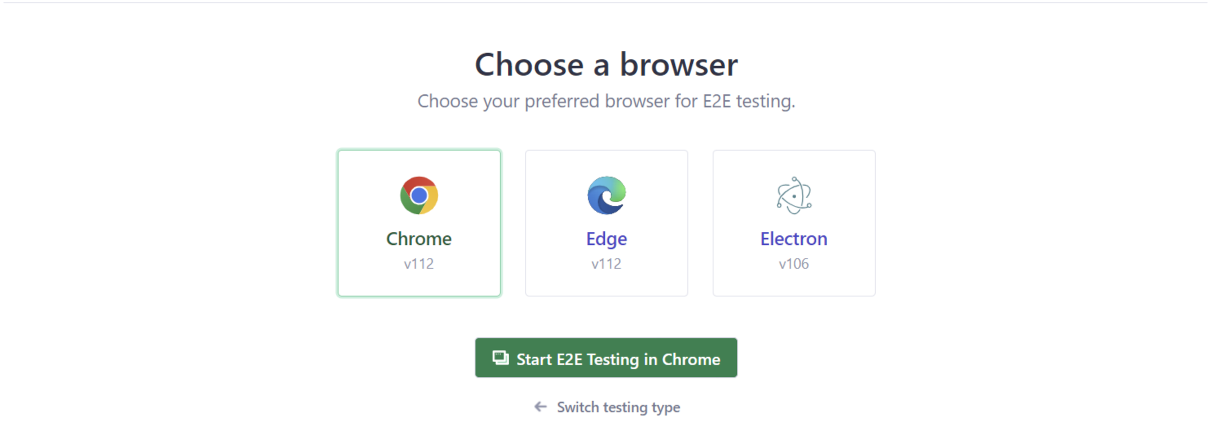
- Ensure Node.js is installed
- Navigate to the project's root directory
- Type the following command into the terminal to open Cypress

```
npx cypress open
```

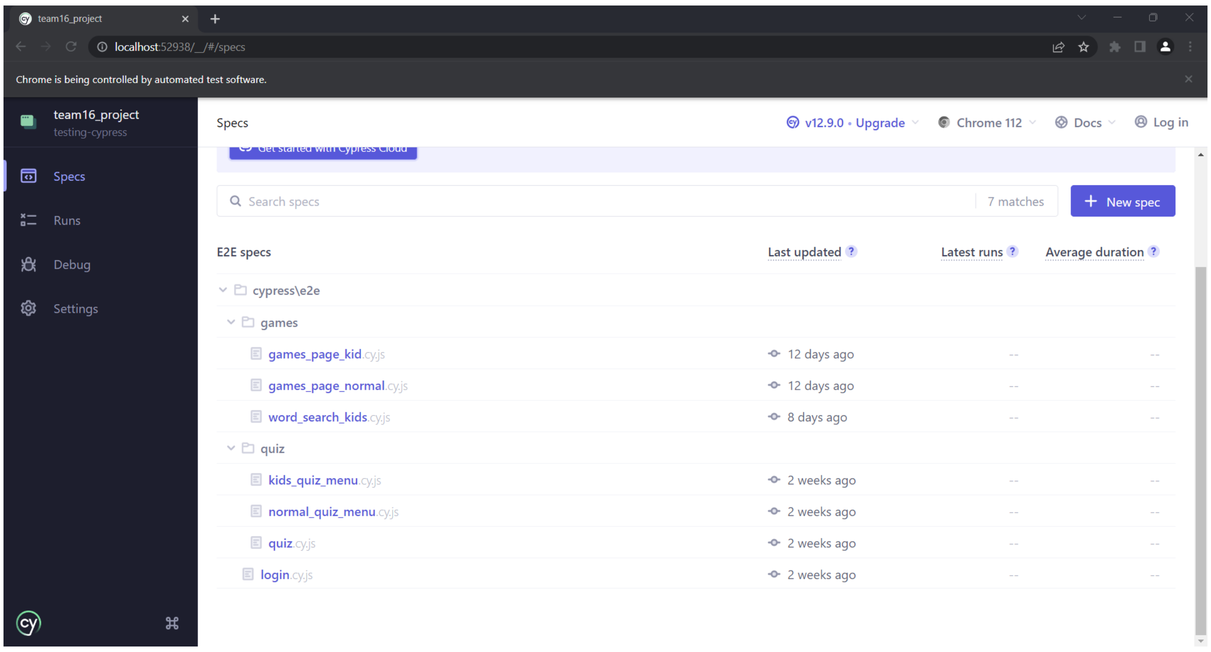
- Select "E2E Testing" option


Cypress
File Edit View Window Help Developer Tools
team16_project (testing-cypress)
v12.9.0 • Upgrade
Docs
Log in
Welcome to Cypress!
Review the differences between each testing type →
E2E Testing
Build and test the entire experience of your application from end-to-end to ensure each flow matches your expectations.
Configured
Component Testing
Build and test your components from your design system in isolation in order to ensure each state matches your expectations.
Not Configured

- Choose your preferred browser, e.g. Chrome, Edge, etc.
- Select "Start E2E Testing"

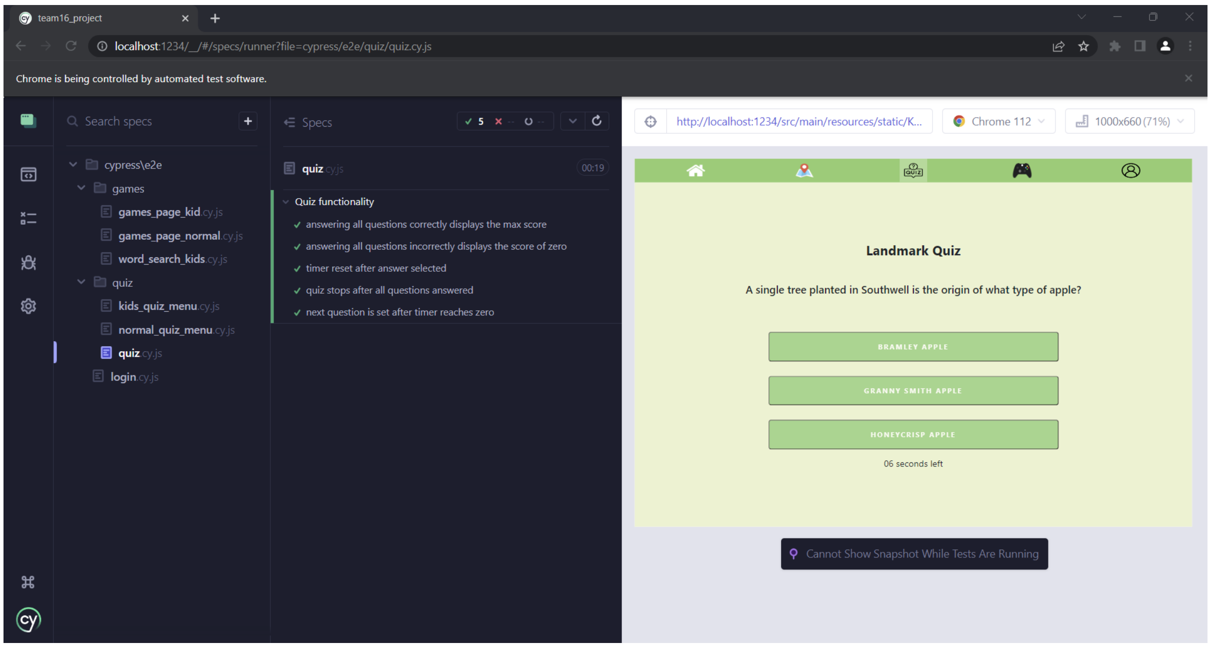


Cypress
File Edit View Window Help Developer Tools
team16_project (testing-cypress) > E2E Testing
v12.9.0 • Upgrade
Docs
Log in
Choose a browser
Choose your preferred browser for E2E testing.
Chrome v112
Edge v112
Electron v106
Start E2E Testing in Chrome
Switch testing type

▪ Select the spec you wish to run the tests for, e.g. quiz.cy.js



E2E specs	Last updated	Latest runs	Average duration
cypress\e2e			
games			
games_page_kid.cy.js	12 days ago	--	--
games_page_normal.cy.js	12 days ago	--	--
word_search_kids.cy.js	8 days ago	--	--
quiz			
kids_quiz_menu.cy.js	2 weeks ago	--	--
normal_quiz_menu.cy.js	2 weeks ago	--	--
quiz.cy.js	2 weeks ago	--	--
login.cy.js	2 weeks ago	--	--

▪ Tests should automatically run in specified browser


team16_project
localhost:1234/__/#/specs/runner?file=cypress/e2e/quiz/quiz.cy.js
Search specs
Specs
✓ 5 ✕ -- ↻ --

Chrome is being controlled by automated test software.

```
cypress\e2e
  games
    games_page_kid.cy.js
    games_page_normal.cy.js
    word_search_kids.cy.js
  quiz
    kids_quiz_menu.cy.js
    normal_quiz_menu.cy.js
    quiz.cy.js
  login.cy.js
```

```
quiz.cy.js                                                      00:19
Quiz functionality
  ✓ answering all questions correctly displays the max score
  ✓ answering all questions incorrectly displays the score of zero
  ✓ timer reset after answer selected
  ✓ quiz stops after all questions answered
  ✓ next question is set after timer reaches zero
```

Landmark Quiz

A single tree planted in Southwell is the origin of what type of apple?

06 seconds left

Cannot Show Snapshot While Tests Are Running

Test Plans

Test Plan: Index Page

ID	Description	Action	Expected Result	Status	Changes Done
1	Blank email and password fields do not redirect user to home page	Press "LOG IN" button	Remain on index page	Pass	None
2	Verify if a user can login with an invalid email and a valid password	Enter invalid email Enter valid password Press "LOG IN" button	Remain on index page	Pass	None
3	Valid username and password redirects user to home page	Enter valid email and password Press "LOG IN" button	Redirected to home page	Pass	None
4	Verify "Create an account" functionality	Press "Create an account" button	Redirected to signup page	Pass	None

Test Plan: Normal Mode Quiz Menu

ID	Description	Action	Expected Result	Status	Changes Done
1	Ensure "Landmarks" button redirects user to landmarks quiz	Press "Landmarks" button	Redirected to normal mode landmarks quiz	Pass	None
2	Ensure "Historical Facts" button redirects user to historical facts quiz	Press "Historical Facts" button	Redirected to normal mode historical facts quiz	Pass	None
3	Ensure "Religious Spaces" button redirects user to religious spaces quiz	Press "Religious Spaces" button	Redirected to normal mode religious spaces quiz	Pass	None

Test Plan: Kids Mode Quiz Menu

ID	Description	Action	Expected Result	Status	Changes Done
1	Ensure "Landmarks" button redirects user to landmarks quiz	Press "Landmarks" button	Redirected to kids mode landmarks quiz	Pass	None
2	Ensure "Historical Facts" button redirects user to landmarks quiz	Press "Historical Facts" button	Redirected to kids mode historical facts quiz	Pass	None
3	Ensure "Religious Spaces" button redirects user to landmarks quiz	Press "Religious Spaces" button	Redirected to kids mode religious spaces quiz	Pass	None

Test Plan: Quiz Functionality

ID	Description	Action	Expected Result	Status	Changes Done
1	Ensure that answering all questions correctly results in a max score	Select all correct answers for quiz	Text contains the string "10 out of 10"	Pass	None
2	Ensure that answering all questions incorrectly results in a score of zero	Complete quiz	Text contains the string "0 out of 10"	Pass	None
3	Ensure quiz timer is reset after answer selected	Select any answer	Timer is reset to correct duration (i.e. 10 secs)	Pass	None

4	Verify quiz stops after all questions answered	Answer all quiz questions	quiz_complete is set to true	Pass	None
5	Ensure next question is set after timer reaches zero	Allow timer to countdown to zero	question string set to a new question	Pass	None

Test Plan: Kids Games Page

ID	Description	Action	Expected Result	Status	Changes Done
1	Ensure the “Word Search” button directs to the word search Game	Click on “Word Search” Button	Redirected to Word Search Page	Pass	None
2	Ensure the “Memory Match” button directs to the memory match button	Click on “Memory Match” Button.	Redirected to Memory Match page	Pass	None

Test Plan: Normal Games Page

ID	Description	Action	Expected Result	Status	Changes done
1	Ensure the “Word Search” button directs to the word search Game	Click on “Word Search” Button	Redirected to Word Search Page	Pass	None
2	Ensure the “Memory Match” button directs to the memory match button	Click on “Memory Match” Button.	Redirected to Memory Match page	Pass	None

Test Plan: Word Search Functionality (Normal Mode)

ID	Description	Action	Expected Result	Status	Changes done
1	Ensure that entering all correct answers results in a max score	Enter all correct answers	Text contains “13 out of 13”	Fail Passed on 11/04/2023	Fixed the incorrect incrementation
2	Ensure game ends after timer is up	Allow timer countdown to zero	Game_complete set to true	Pass	None
3	Ensure all wrong answers results in a zero score	Enter all wrong answers	Text contains “0 out of 13”	Pass	None
4	Ensure score is within possible bounds	Complete word search	correctAnswers is between 0 and 13 (inclusive).	Pass	None

Test Plan: Word Search Functionality (Kids Mode)

ID	**Description**	**Action**	**Expected Result**	**Status**	**Changes done**
1	Ensure that entering all correct answers results in a max score	Enter all correct answers	Text contains “9 out of 9”	Fail Passed on 11/04/2023	Fixed the incorrect incrementation
2	Ensure game ends after timer is up	Allow timer countdown to zero	Game_complete set to true	Pass	None
3	Ensure all wrong answers results in a zero score	Enter all wrong answers	Text contains “0 out of 9”	Pass	None
4	Ensure score is within possible bounds	Complete word search	correctAnswers is between 0 and 9 (inclusive).	Pass	None

Test Plan : Memory Match Kids

ID	Description	Action	Expected Result	Status	Changes done
1	Ensure cards that do not match are flipped to their original state	Flipping unmatched cards	Flipped back to their original state	Pass	None
2	Ensure that cards that do match stay unflipped	Flipping matched cards	Remaining unflipped	Pass	None
3	Ensures the game ends after timer has stopped	Wait for timer to finish	Game_complete set to true, redirected back to end page	Pass	None
4	Game ends after all matches have been made	Make all correct matches	Score contains “4 out of 4”	Pass	None
5	Ensure the score of the is inclusive of the kids mode range	Finish game	Score contains a number inclusive of 0 to 4	Pass	None

Test Plan : Memory Match Normal

ID	Description	Action	Expected Result	Status	Changes done
1	Ensure cards that do not match are flipped to their original state	Flipping unmatched cards	Flipped back to their original state	Pass	None
2	Ensure that cards that do match stay unflipped	Flipping matched cards	Remaining unflipped	Pass	None
3	Ensures the game ends after timer has stopped	Wait for timer to finish	Game_complete set to true, redirected back to end page	Pass	None

4	Game ends after all matches have been made	Make all correct matches	Score contains “4 out of 6”	Pass	None
5	Ensure the score of the is inclusive of the normal mode range	Finish game	Score contains a number inclusive of 0 to 6	Pass	None

Test Plan: Favourites Page (Normal Mode)

ID	Description	Action	Expected Result	Status	Changes done
1	A list of buttons appears on screen with the landmark names	No action	All the user favourites appear on the page	Pass	None

Test Plan: Favourites Page (Kids Mode)

ID	Description	Action	Expected Result	Status	Changes done
1	A list of buttons appears on screen with the landmark names	No action	All the user favourites appear on the page	Pass	None

Drop-down menu normal mode

ID	Description	Action	Expected Result	Status	Changes done
1	Profile page	Click on Profile button	The user is redirected to the profile page	Pass	None
2	Favourites page	Click Favourites button	The user is redirected to the Favourite Locations page	Pass	None
3	Logout	Click Logout button	The user is being logged out and redirected to the login page	Pass	None

Drop-down menu kids’ mode

ID	Description	Action	Expected Result	Status
1	Profile page	Click on Profile button	The user is redirected to the profile page	Pass
2	Favourites page	Click Favourites button	The user is redirected to the Favourite Locations page	Pass
3	Logout	Click Logout button	The user is being logged out and redirected to the login page	Pass

Places to eat page

I D	Description	Action	Expected Result	Statu s
1	Filter check	Click on a filter	The page should show a list of restaurants corresponding to the cuisine selected in the filter	Pass
2	Search bar use	Use the search bar to look for a type of cuisine	The page should show a list of restaurants corresponding to the cuisine typed in the search bar	Pass
3	Filter check advanced	Select multiple filters	Page should show a list of restaurants corresponding to the cuisine selected in the filters	

Test Plan: Map (Kids and Normal Mode)

ID	Description	Action	Expected Result	Status	Change s done
1	Move to user location	Click on the user location button	The screen should pan to the users location	Pass	None
2	Marker infobox	Click on a marker on the map	The markers related infobox should be visible and the name and extra info be correct	Pass	None
3	Filter	Click an image on the legend	All markers of the type of image clicked should be hidden/shown from the map	Pass	None
4	User location pin	Open the map page	A marker is placed at the users location accurately.	Pass	None

Test Plan: Map (Normal Mode)

ID	Description	Action	Expected Result	Status	Changes done
1	Close transport links	Open maps page	Information about close transport links should be accurate	Pass	None
2	Bus and tram pins	Click on a bus or tram pin	Infobox contains the correct name	Pass	None

Test Plan : Data Pipeline

ID	Function	Test Data	Expected Result	Status	Changes Done
1	cleanFile	[[0,1,"Hello",""],[2,3,"Bye","Hi"],[10,20,"e","i"]]	[[2,3,"Bye","Hi"],[10,20,"e","i"]]	Passed	None
2	cleanFile	[[0,1,"Hello",""],[2,3,"","Hi"],[10,20,"e","i"]]	[[10,20,"e","i"]]	Passed	None
3	cleanFile	[[],["",2,3]]	[[]]	Passed	None
4	getSchema	[['A','B','C','D'],[0,1,2,3],[2,3,4,5]]	['A','B','C','D']	Passed	None
5	readNClean	"rNC_testData1.csv"	[['A','B','C','D'],[1,2,3,4],[2,3,6,7],[5,6,7,8]]	Passed	None
6	readNClean	"rNC_testData2.csv"	[['A','B','C',' '],[2,3,4,5],[4,5,6,1]]	Passed	None
7	findCombineDiff	[['A','B','C','D'],[1,4,3,2],[4,5,1,3]] [['A','B','C','D'],[1,4,3,2],[10,5,1,3]]	[['A','B','C','D'],[1,4,3,2],[4,5,1,3],[10,5,1,3]]	Passed	None
8	findCombineDiff	[['A','B','C','D'],[1,4,3,2],[4,5,1,3]] [['A','B','C','D'],[1,4,0,2],[4,5,1,3]]	[['A','B','C','D'],[1,4,0,2],[4,5,1,3],[1,4,3,2]]	Passed	None
9	convert2JSON	[['A','B','C','D'],[1,4,3,2],[4,5,1,3]]	[{'A':1,'B':4,'C':3,'D':2},{'A':4,'B':5,'C':1,'D':3}]	Passed	None
10	convert2JSON	[['A','B','C','D'],[1,4,3,2],[4,0,500,3]]	[{'A':1,'B':4,'C':3,'D':2},{'A':4,'B':0,'C':500,'D':3}]	Passed	None
11	getDataFromDict	{'A':1,'B':2,'C':3,'D':4}	[1,2,3,4]	Passed	None
12	getData	'markets' databaseDict	Anything unequal to None	Passed	None
13	getData	‘bus_tram_lanes’ databaseDict	Anything unequal to None	Passed	None
14	push2Firebase	[{"A":1,"B":2,"C":3,"D":4},{"A":12,"	Visual Confirmation	Passed	None

		B":22,"C":32,"D": 24}] 'p2F_testData1_1'			
15	removeFromFirebase	'p2F_testData1_1'	Visual Confirmation and Function returns Not None	Passed	None
16	removeFromFirebase	'unknown_doesnt_exist'	Display error message Nothing returned	Passed	None

Test Plan: Places To Eat - Server Side

ID	Function	Test Data	Expected Result	Status	Changes Done
1	filter	{'id': 'Xm7psilJ_gogxFNVgi5S9A', 'alias': 'world-service-restaurant-nottingham', 'name': 'World Service Restaurant', 'image_url': 'https://s3-media4.fl.yelpcdn.com/bphoto/p-qHkLcFpBn_9hiru_RiKA/o.jpg', 'is_closed': False, 'url': 'https://www.yelp.com/biz/world-service-restaurant-nottingham?adjust_creative=rKAm3DvC_1-tTGr2WpqoKQ&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=rKAm3DvC_1-tTGr2WpqoKQ', 'review_count': 8, 'categories': [{'alias': 'british', 'title': 'British'}], 'rating': 4.0, 'coordinates': {'latitude': 52.9508874560959, 'longitude': -1.15253448486328}, 'transactions': [], 'price': '£££', 'location': {'address1': 'Newdigate House', 'address2': 'Castlegate', 'address3': '', 'city': 'Nottingham', 'zip_code': 'NG1 6AF', 'country': 'GB', 'state': 'NGM', 'display_address': ['Newdigate House', 'Castlegate', 'Nottingham NG1 6AF', 'United Kingdom']}, 'phone': '+441158475587', 'display_phone': '+44 115 847 5587', 'distance': 1761.188029846324}	{'name': 'World Service Restaurant', 'image_url': 'https://s3-media4.fl.yelpcdn.com/bphoto/p-qHkLcFpBn_9hiru_RiKA/o.jpg', 'longitude': -1.15253448486328, 'latitude': 52.9508874560959, 'cuisine': 'British'}	Pass	None
2	filter	{'id': '', 'alias': 'world-service-restaurant-nottingham', 'name': 'Indian Restaurant', 'image_url': 'https://s3-media4.fl.yelpcdn.com/bphoto/p-qHkLcFpBn_9hiru_RiKA/o.jpg', 'is_closed': False, 'url': 'https://www.yelp.com/biz/world-service-restaurant-nottingham?adjust_creative=rKAm3DvC_1-	{'name': 'Indian Restaurant', 'image_url': 'https://s3-	Pass	None

		tTGr2WpqoKQ&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=rKAm3DvC_1-tTGr2WpqoKQ', 'review_count': 3, 'categories': [{'alias': 'indian', 'title': 'Indian'}], 'rating': 4.0, 'coordinates': {'latitude': 52.9508874560959, 'longitude': -1.15253448486328}, 'transactions': [], 'price': '£££', 'location': {'address1': 'Newdigate House', 'address2': 'Castlegate', 'address3': '', 'city': 'Nottingham', 'zip_code': 'NG1 6AF', 'country': 'GB', 'state': 'NGM', 'display_address': ['Newdigate House', 'Castlegate', 'Nottingham NG1 6AF', 'United Kingdom']}, 'phone': '+441158475587', 'display_phone': '+44 115 847 5587', 'distance': 1761.188029846324}	media4.fl.yelpcdn.com/bphoto/p-qHkLcFpBn_9hiru_RiKA/o.jpg', 'longitude': -1.152534484863 28, 'latitude': 52.950887456095 9, 'cuisine': 'Indian'}		

Known Issues

Due to time constraints, we were not able to completely implement the (favourites page button) redirection. However, it has been partially implemented for the normal mode of the application because there were problems in accessing global variables between scripts.