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/

Change Section

- Added “Introduction” to “Project Background”
- Removed “UML diagrams” and “Our requirements” subsections. Their content has been integrated throughout the “Requirements and Critical Analysis” section
- Added “Analysis”, “Requirements and Specifications” and “Design” subsections to “Requirements and Critical Analysis”
- Renamed “Initial Software Implementation” to “Software Implementation and Testing”
- Moved “How do we use version control to our benefit” subsection from “Initial Software Implementation” to “Project Management”
- Added new “Front-end Implementation” subsection to “Initial Software Implementation”
- Removed “How have we integrated the front and back end?” subsection from “Initial Software Implementation”

Final Report

PROJECT BACKGROUND

Introduction

We have been made aware that the platform “NCC Open Data” is largely unused, with the most frequent tags being “transport” or “services”, we have established that an application could be made to provide the public of Nottingham with fruitful advice and guidance about the city from the information derived from the datasets. By creating an interactive educational app namely called “Adventurer's Guide to Nottingham,” we have utilised the data to create something innovative and unique. Our team expanded upon the requirements to design an inclusive app that appeals to all ages and those new to the city and familiar. We offer the user the option to traverse through the app by choosing a “mode” i.e. kids mode or normal mode, with each being heavily discerned from the other.

Motivations for the Project

As students ourselves, we take a personal interest in this app, as most of our ideas stem from our needs and wants when it comes to interacting with the city. We wanted a multi-functional app, an app that provided the means for navigation and entertainment without having to switch from one app to another, and one that was unique to the city of Nottingham and personalised to its features.

How do we compare to other products in the market?

With the nature of the app being so flexible, it has allowed us to make almost the entirety of it our additions. By offering two versions of the app, we made it appeal to all the archetypes of Nottingham. The usage of NCC open data plays a large role in supporting the data that is displayed by our app. While some data in our app may already be available in our competitors’ applications, we plan on using data that is underutilised in the current market.

Another key aim is to present the data in a fresh and engaging way that tells the user more about Nottingham. Each point of interest (landmarks, shops, etc.) within the map displays more information about the location upon click, removing the need for external searches outside the app for information.

Additionally, with one of our main goals being to showcase Nottingham as a popular destination and encourage people to visit, we decided to implement games and quizzes to provide a fun way to share knowledge about the city, this sets us apart from other travel apps whose main purpose is to provide navigation and booking.

REQUIREMENTS AND CRITICAL ANALYSIS

Analysis

Throughout the project lifecycle, we maintained a backlog of user stories **(Figure 1)** for our agile development. This allowed us to theorise what a variety of users would require in our app, and

categorise which off the bat would take more time to implement based on difficulty. To our benefit, this developed more confidence in designing the prototype later on.

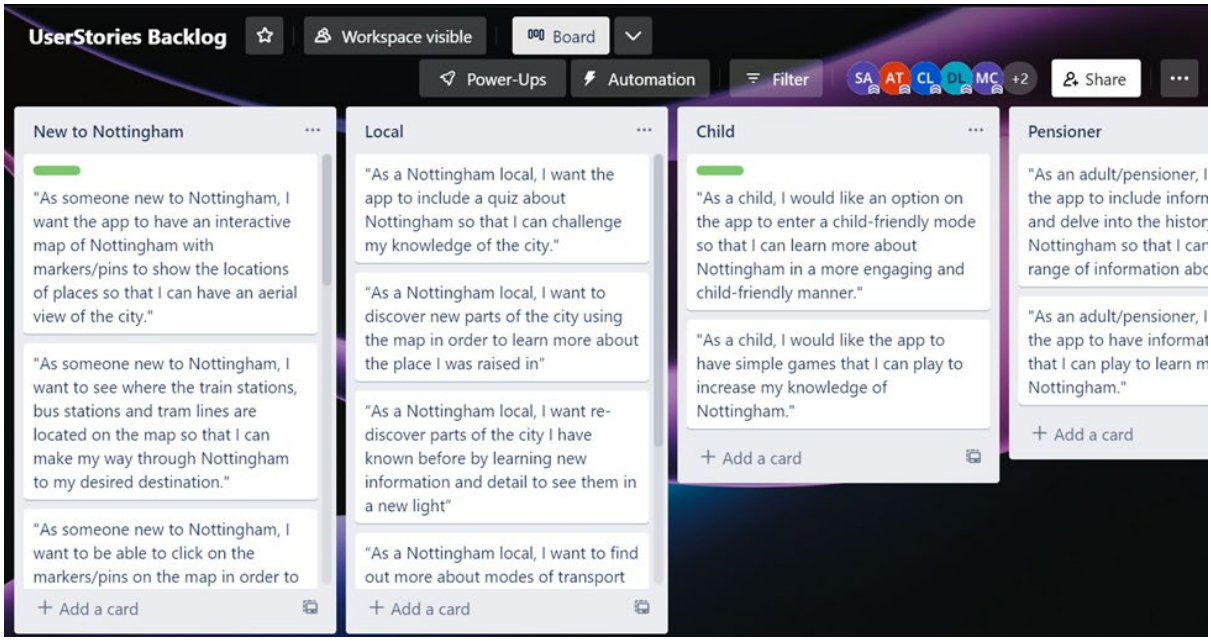

Figure 1

Requirements and Specifications

Req No	KISS VERSION 1	ALPHA VERSION 2	BETA VERSION 3
1	1.1) Different versions of the app - Kid's mode and Normal mode	2.1) Landmarks are issued with information for users to read	3.1) Pipeline made to update data as made available
2	1.2) Ability to choose on screen which version to proceed with	2.2) Transport information is easily available for users to access	3.2) Filters to see certain types of landmarks or places to eat.
3	1.3) Optimum use of data available on the hub	2.3) Discerning the kids' mode and normal mode. Kids mode being more colourful and simpler while the normal mode being more information dense	3.3) Use an external data source to develop a "places to eat" option
4	1.4) Map for the users to follow for navigation	2.4) Quizzes available to play with major landmarks Note: simpler quiz for the kids' mode and a more complex version for the normal mode	3.4) Educational games available to play for the kids' version and normal mode, kids' version being more interactive and normal mode being more educational - Both being easy to play
5	1.5) Setup database with all the data we would use e.g., landmarks	2.5) Emoticon icons available for different purposes of the app e.g., heart for favouriting, lightbulb for random facts or pop ups	3.5) Ability to share on social media to increase awareness of the app
6	1.6) Data is preserved, mode preferences etc	2.6) Create a legend for the map in order to quickly identify the types of landmarks available to visit.	3.6) Include opening hours for landmarks which require tickets to enter
7	1.7) Easy to maintain and extend		3.7) Develop an adaptable interface for different devices
8			3.8) Recommendations available for users based on previous data collected

FUNCTIONAL REQUIREMENTS ●
NON-FUNCTIONAL REQUIREMENTS ●

When creating the app we had a timeline in mind. First, to implement the KISS (Keep It Super Simple) version of the app followed by the alpha and beta versions. The KISS requirements were vital to the initialisation of the app, we produced the skeleton of the app in semester one and carried on with the creative features on a solid base for a well-rounded user experience in semester two, for the alpha and beta versions of the app. Our main priority was to use data from the NCC platform as effectively as possible **(req 1.3),** this was also further stressed by our sponsor.

For the KISS version of the app, we implemented an information-dense map with pins corresponding to the data (i.e. latitude and longitude of landmarks) from the NCC open data platform **(req 1.4 and 1.5)**.

After successfully releasing the KISS version of our app, we then proceeded with the alpha version. The alpha release grazed upon all aspects of the KISS version but with further extensions to the map and the inclusion of quizzes, which heightened the app experience. For example, the alpha release added transport links **(req 2.2)** to the map, and quizzes with categories “Landmarks”, “Historical Facts”, and “Religious Spaces” **(req 2.4)**.

For our final beta release, we polished the style of the app, ensured the UI was mobile-friendly **(req 3.7)**, added a variety of games, such as words search and memory match **(req 3.4)**, added a restaurants page to the normal mode **(req 3.3)**, and dynamically updated the data in the database, via a pipeline, when it was made available on the NCC platform **(req 3.1)**.

What changed?

Having reached the halfway point in the project, the team took some time to reflect upon our initial requirements and improve upon them for the future of the project. To streamline the development process the team decided on removing **req 2.2**. This is due to the data from the NCC platform on bus stop locations having unintelligible numbers for the respective location of each stop. Furthermore, the pins from the bus stops would be over 800 pins which could clutter the map. To bypass this dropped requirement, the directions API has been implemented so that users can receive information on how to get to any of the landmarks.

Req No	Alpha 2	Beta 3
1	2.1) Use the map for navigation, via Directions API	3.1) Ability to share on social media to increase awareness of the app
2	2.2) Save the mode for the user	3.2) Pipeline to update the data as it updates
3	2.3) save filters used by the user on landmarks	3.3) Recommendations available for users on previous data collected
4	2.4) Landmarks are issues with information read	3.4) Include opening hours for landmarks
5	2.5) Filters to see certain types of landmarks	3.5) Games for Kids' Mode
6	2.6) Discerning the kids' mode and normal mode, kids' mode is more colourful and normal mode is more information-dense	3.6) Games for normal mode
7	2.7) Emoticon lightbulb	3.7) Use an external source for places to eat
8	2.8) Emoticon heart, when favourited, should be added to a favourites page	3.8) Filters to see places to eat
9	2.9) Make a profile for the user	
10	2.10) Making quizzes for each mode	
11	2.11) Transport information available, make a symbol indicating transport for user, the popup will then show the user the nearest bus, tram, train, and railway stations from them	
12	2.12) Request location from the user to use for transportation purposes	

FUNCTIONAL REQUIREMENTS
NON-FUNCTIONAL REQUIREMENTS

To store the users' selected mode (e.g. kids mode, normal mode), we determined that user credentials would need to be stored **(req 2.2, 2.9)**.

Following the KISS release of our application, we created a user survey to appraise our app. From this, we re-evaluated and re-prioritised our requirements.

The largest distinction made between our app and our competitor was its interactivity in the form of games and quizzes. The majority (75%) were interested in these features with only a few people saying that they were interested in more practical features only.

The majority of users seemed to be excited at the prospect of using an app that used city-themed games and quizzes, this motivated us to reprioritize that to a higher importance than before **(req 2.5, 2.6 and 1.10)**.

The most requested features were: the ability to navigate using directions on the map **(req 2.1)**, searching for places to eat **(req 3.8)**, and facts about landmarks in Nottingham **(req 2.4)**. Because of this, we prioritised the development of these features within the app.

Having a kids/normal mode means that our app targets a large demographic of people. Receiving a response from each demographic meant that we obtained a better idea of how to differentiate the kids mode from the normal mode **(req 2.6)**.

Additionally, when the users who were not interested in the entertainment portion of the app were asked as to what their preference was, it geared towards having an app with more practical features, hence we decided to implement those features as such in parallel (map directions, filtering for landmarks and nearest transport links).

Question 7 of the survey brought forth a satisfactory rating of an average of 3.5 stars for the look and feel of the app, due to this we refocused on making the app more polished throughout the project, by changing backgrounds, adding sound effects, and making a greater effort in discerning kids and adult mode **(req 2.6)**.

When enforcing the results of question 9, we approached it with a trickle-down approach, we first implemented and perfected the more preferred options such as quizzes, directions and mode distinction and then went on to develop the games, places to eat and landmark blurbs.

Our quizzes were based on the most popular options: landmarks, religion and history.

From the survey, it was determined that respondents would like to be able to favourite locations for ease of future access **(req 2.8)**.

Furthermore, older respondents displayed a preference for information (facts, blurbs, etc.) and younger respondents displayed a preference for activities (interactive, games, etc.), so normal mode was developed to be more informative and kids mode more activity-based **(req 2.4, 2.6, 2.7).**

Most respondents permitted the use of their GPS location to show their position on the map and be directed to their desired destination **(req 2.12, 2.1, 2.11).**

Finally, within the survey we asked participants to select their age bracket from a broad range of ages, we can utilise this data to identify what users from different age brackets would want in the app, allowing us to create more specialised modes in the future.

Design

The use case in **Figure 2** is a diagrammatic view of the specifications. The diagram displays the interaction between the different users and the system. The “new to Nottingham” and “local to Nottingham” actors both inherit from a “general user” actor, as both do the same tasks but have different motives for doing so. For example, someone new may use the quiz (ref to spec 2.3) option to learn more about the city whilst someone who is native to the city may use it to test existing knowledge.

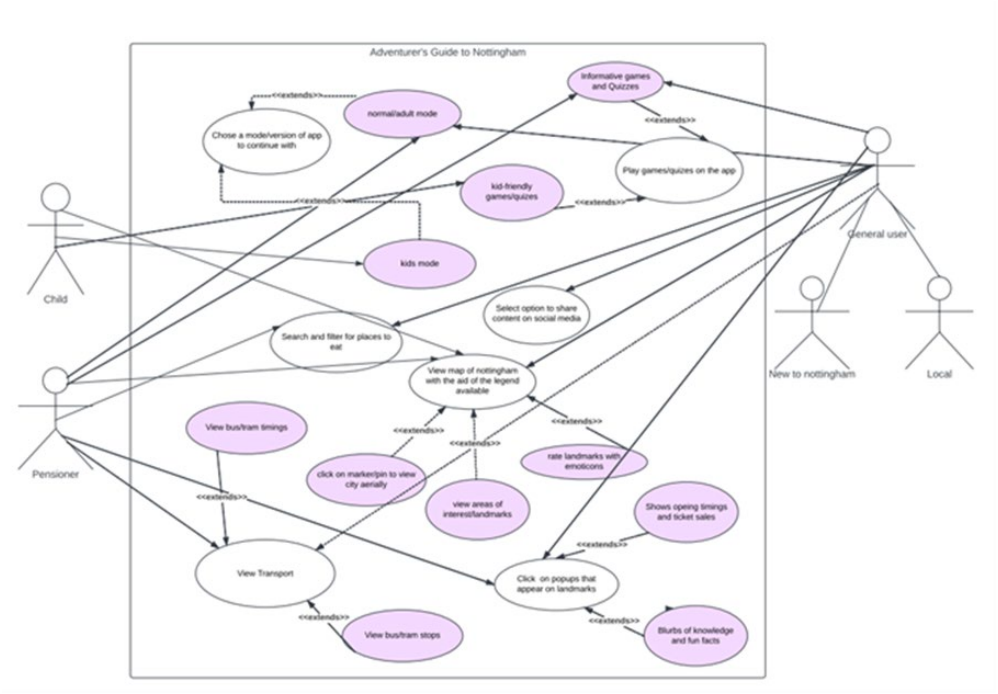

Figure 2

Within both normal mode and kids mode users can complete quizzes about Nottingham's landmarks, history, and religion. An activity diagram for the general flow of a quiz is shown in **Figure 3**.

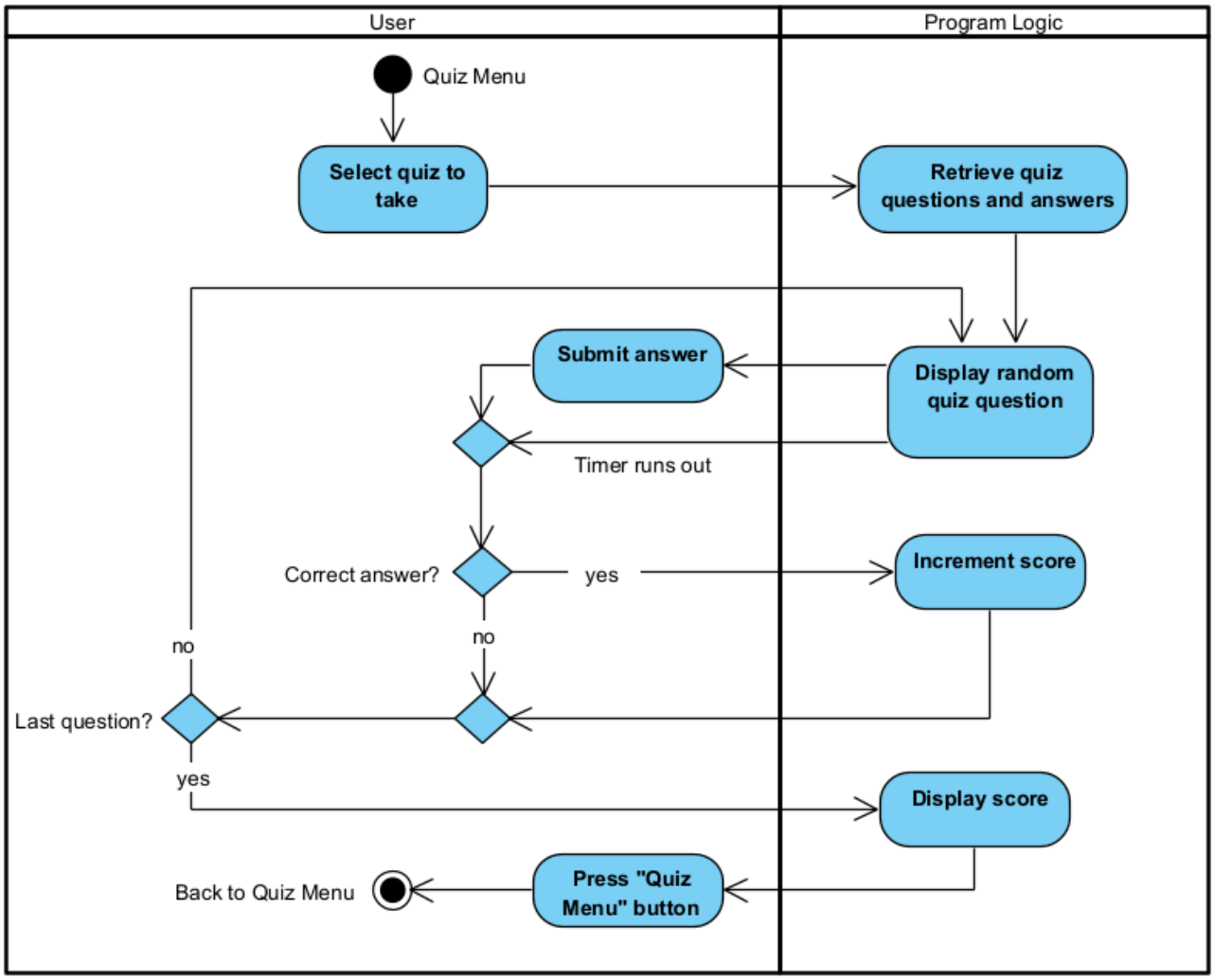

Figure 3

The map page contains pins showing the locations of landmarks, shops, community centres, car parks, etc. Users can press a landmark icon to display information about the landmark. Furthermore, users can add landmarks to their favourites list by pressing the heart icon available within the landmark information display. An activity diagram for the process of adding a landmark to a user's favourites list is shown in **Figure 4.**.

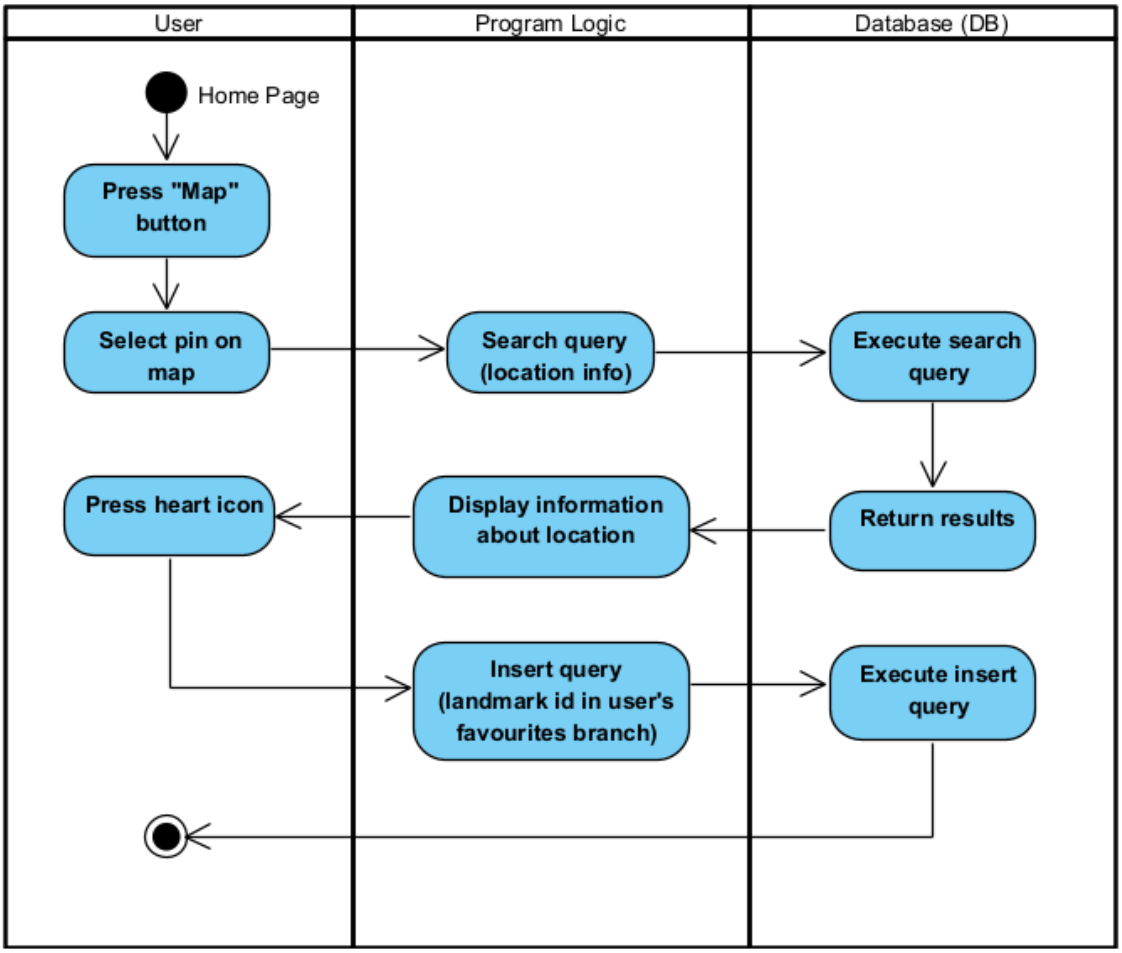
User
Program Logic
Database (DB)
Home Page
Press "Map" button
Select pin on map
Search query (location info)
Execute search query
Press heart icon
Display information about location
Return results
Insert query (landmark id in user's favourites branch)
Execute insert query

Figure 4

To fully grasp and validate the user requirements, several sequence diagrams have been designed to help envision the app's users in various runtime scenarios. Figures **5, 6, and 7** distinctly demonstrate the inter-system events, process concurrency, and the message exchange between the system and the user. In each case, the system consists of three lifelines: the GUI, server, and database. These subsystems form the backbone of the application, which further reinforces our dedication to implementing our system design principles, which are maintainability and extensibility **(req 1.7).**

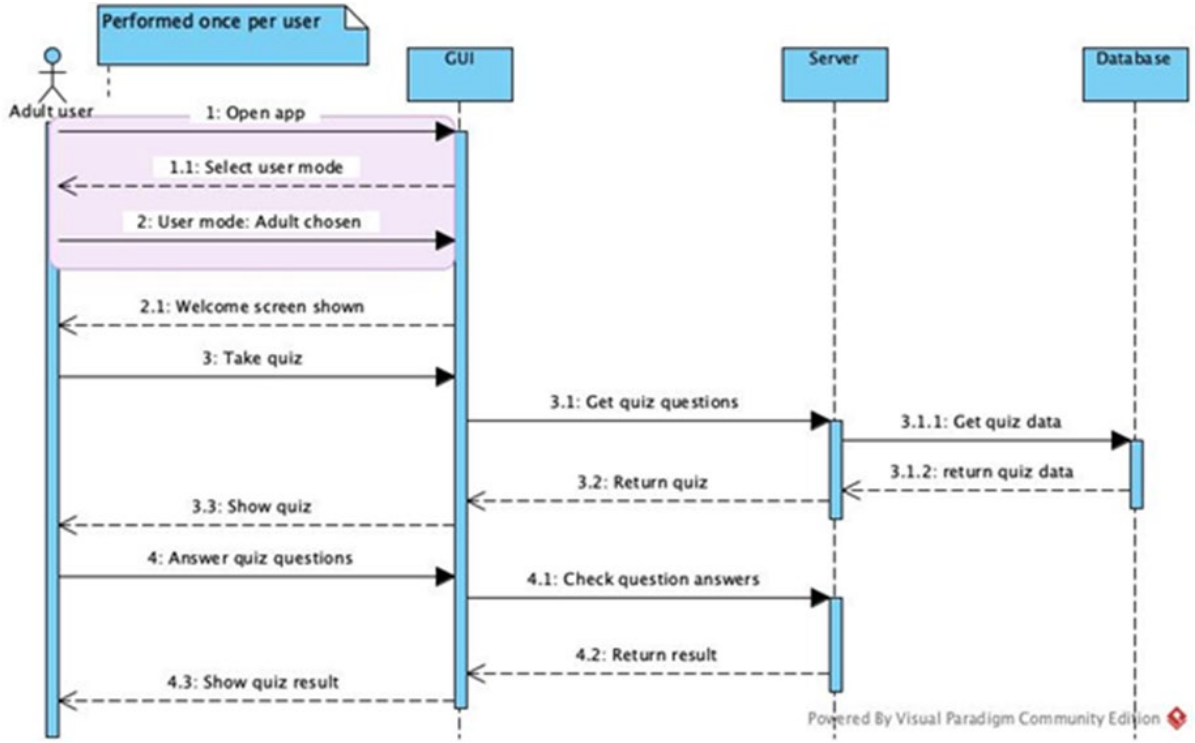

Figure 5

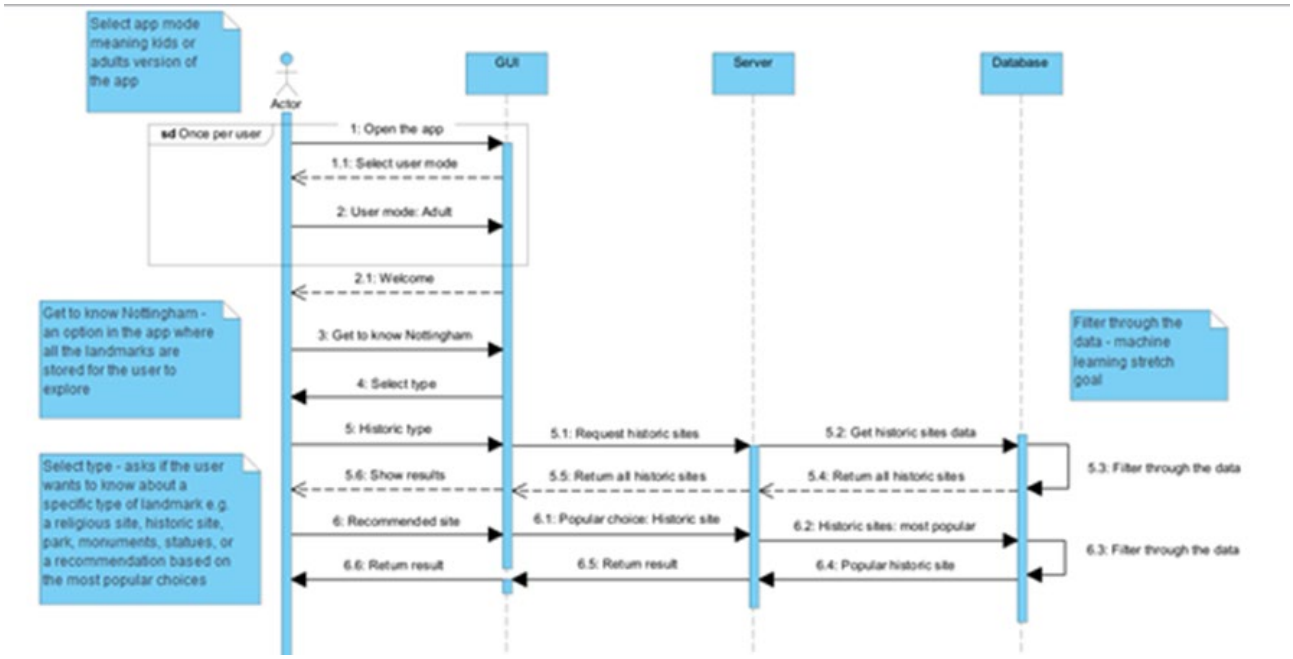

Figure 6

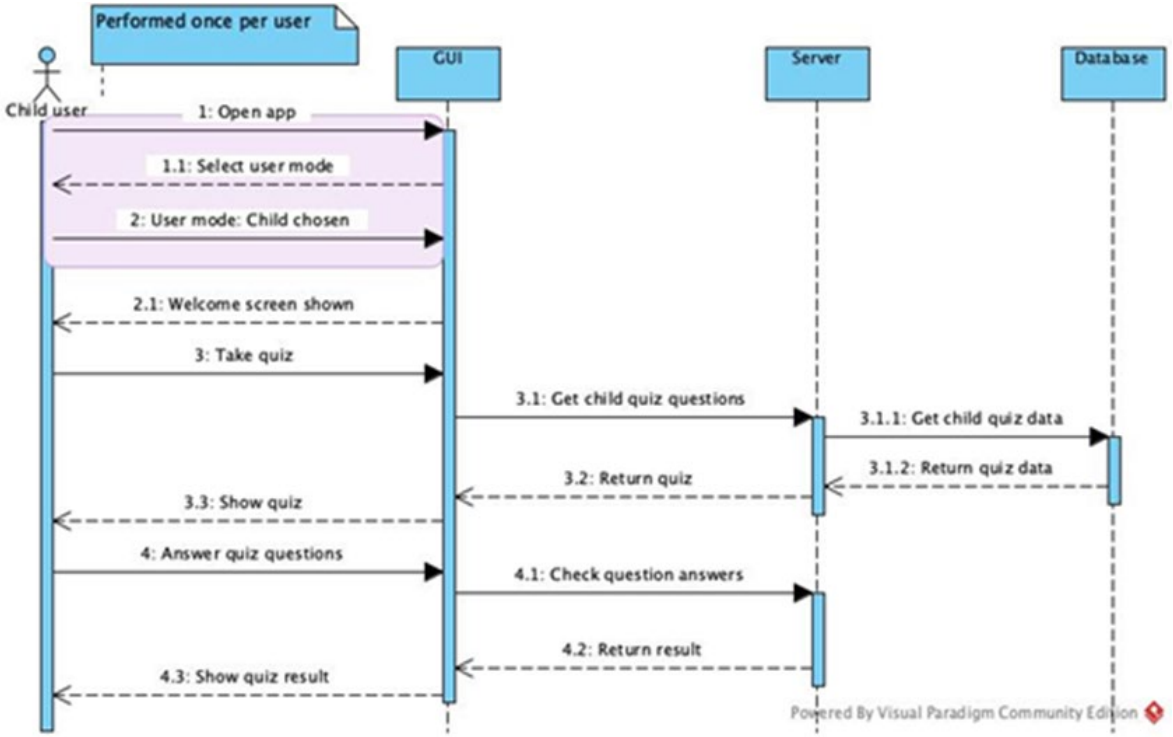

Figure 7

For semester one, we designed a low-fidelity prototype for the app interface **(Figure 8)** corresponding to the KISS requirements. This lo-fi prototype enabled us to develop the design of our UI through iterative design, where different group suggestions in a short space of time prompted innovation and improvement on our initial idea.

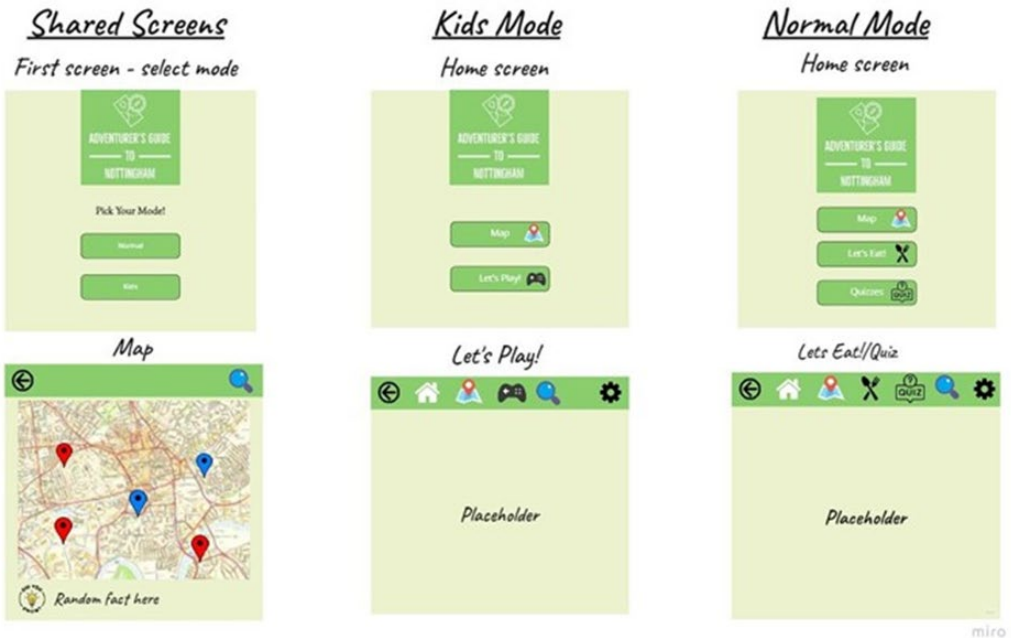

Figure 8

Following the revision of the requirements in semester 2, we proceeded to reflect these changes onto the existing low-fidelity prototype **(Figure 9)**. The most considerable changes have been including a signup/login page, a favourites page, and a user profile page which includes settings where the user can change their app mode.

All of these additions serve as an extension to the current framework, thus enforcing the extensibility and maintainability design features of the project.

Figure 9

SOFTWARE IMPLEMENTATION AND TESTING

Our aim

By utilising information from the Nottingham City Council Open Data website, our project aims to demonstrate the potential advantages of leveraging publicly accessible data sources. Our app successfully offers navigation information and specialised information divided into categories for both modes in an informed manner that appeals to a wide audience. To enhance the user experience, we added the ability for users to create credentials and save their favourite locations. Additionally, to guarantee that our users receive the most recent information, we expanded the app to update data in real-time via a pipeline to the NCC Open Data platform.

Front-end implementation

For the front-end implementation of our app, it was preferable to use web languages such as HTML, CSS, and JavaScript since our front-end group has plentiful experience using them. Our team also decided to utilise the Bootstrap CSS framework as it provides responsive, mobile-friendly web development. We have also modified our used Bootstrap templates (i.e., navigation bar and buttons) with our own CSS stylesheets to personalise the design to fit our preferences.

User credentials

Saving user details via credentials was an unexpected new requirement which proved to be beneficial. By saving user details we can store the user's selected mode which is then retrieved from the database when users log in. Additionally, this allowed us to provide the feature of changing modes. This addition proved to be extremely useful for parents, who want a dual-account to share with their children.

For the login and signup purposes of our app, we decided to utilise the Firebase Authentication service to authenticate users with their chosen email and password. Additionally, this service enables us to easily extend the app to include more sign-in methods in the future, i.e. Google Sign-in, Facebook Login, etc.

Map

The map for the normal and kids mode is implemented using the Google Maps JavaScript API as it provides a way to display interactive customisable maps which we believe are more engaging to the user. This API also allowed us to maintain more control over the mapping experience. For instance, this allowed us to centre the map on Nottingham with a custom zoom value and add our own pins using the NCC data, with unique icons for different locations corresponding to the tables in the database, i.e., Landmarks, shops, car parks, etc. Additionally, we set up a legend for the map to help users understand what different icons represent.

Favourites page

Saving the favourite landmarks of our users was possible by adding a heart icon to the landmark blurb which, upon clicking, saved the landmark id to the user's database branch. This is further utilised in the favourites page, where users click on a location title to be redirected back to the marker on the map.

This particular feature further elevates the project as it allows the user to personalise the app, increasing its reusability.

The user's favorited locations are retrieved via JSON queries from the database, allowing us to dynamically represent favorited locations with a filled-in heart in the location blurb.

The 'favourite' button is implemented using a CSS background image of a heart alongside HTML class toggling to change the appearance of the heart from empty (when not favorited) to filled in (when favorited). When a user selects the heart icon, the location's id is either added to the user's favourite list in the database or removed.

Quizzes

Our strategy to implement the quizzes was to include three quiz categories, one for each topic AGN is based on: landmarks, history and religion in Nottingham. Before the physical implementation, the team conducted research and collected information on these topics to give an accurate representation of the city. After creating the questions for both modes and creating separate HTML pages, a built-in 10 seconds countdown timer was added for each question to increase complexity.

For the implementation of the quizzes, our team determined that it was more appropriate to store the quiz question and answer options in JSON files as the data is complex and unchanging. This also makes it easy to add more questions in the future. Furthermore, we utilised jQuery to retrieve the JSON data using an AJAX call. Following this, we used the HTML DOM in the quiz JavaScript to dynamically access and update the content of HTML elements in the quiz page, i.e. randomly change the question & answer options, and change to the final score page upon completion of quiz.

The answer option buttons each have event listeners which trigger the next question to load and increase the user's score if the correct answer is selected.

Word Search Game

The word search was tackled in a similar way to the quizzes, much of the code was reused and sculpted to fit the requirements of the word search.

The user is prompted to input words under a specified time limit, the user input is then iterated through a JSON file of answers and if correct, the score integer is incremented. The JSON file retrieval and use was reused from the quiz implementation, this reduced the amount of time needed to create this game and maintained the same practicality as the quizzes. The javascript code additionally was tweaked to not allow the user to enter the same attempt twice. This was done by storing the user's attempts into an array and checking afterwards if the array contained the user's answer.

After the word search ends, the user’s score is displayed and the option to return to the game menu provided.

To further add on about the differentiation of the game in terms of normal and kids mode, the game was made harder for normal as fewer total attempts were given to answer, whilst the kids' mode had the opposite.

Memory Match Game

A memory match game is a classic card game, the game involves a deck of cards, which are usually laid out face down in a grid pattern. The objective of the game is to find pairs of matching cards by flipping them over and remembering their locations on the game board. In our case, the game has a theme where each card features the logo of the app and depicts famous places in Nottingham. To enhance the game's difficulty, a timer and point system were incorporated. Additionally, the game is designed to have distinct modes for kids and adults, with the former having fewer total matches.

After completing the word search, we proceeded to implement the memory match game as our second and final game for the app. The process followed a similar pattern as that of the word search and quizzes, where an HTML page was created with stacks of cards, each assigned a unique ID that could be modified dynamically using JavaScript.

To keep the game consistent with the app's design, the timer used on the quizzes page was reused to determine when the game ended. Once the game ended, the user was given the option to return to the game menu and their score was displayed by updating the existing HTML elements with the necessary information.

The JavaScript code for the memory match game utilised event listeners for each card, which allowed the program to determine whether or not a match had been made by checking the card's unique ID using a helper function. If a match was found, the two cards were flipped over to reveal the location of Nottingham and were marked as "matched" by adding an attribute to both cards. Conversely, if the cards did not match, they were flipped back down and marked as "flip”.

Places to Eat

The places to eat page was created using a database that was web scraped (discussed further later) and fulfilled our requirement of using another data source besides the NCC open data website. Also according to the popular feedback of our first survey, it turns out the places to eat page was one of the popular preferences of users. Due to this, we decided incorporating this feature was important for ultimate user satisfaction.

However, with this feature being one of our latter ones, we encountered quite a few unexpected delays in creating this feature. The end product is one with all completed aspects of a listed feature page but we were unable to integrate a page that was accustomed to user location, which will be discussed further later.

The actual page consists of three parts, the actual places to eat which contain the image of the restaurant, the title of the place and the cuisine. All of this information is listed in the Firebase database under “placesToEat”, where the sub-information of the restaurant is listed under a nested format. The page was made by iterating through the Firebase and creating buttons dynamically using JS and styling using CSS. The second part of the page consists of the “filter” where the user

is allowed to pick between multiple cuisines. When clicked, the page shows the cuisine of choice to the user. This was also done with javascript as the “food-list” element on the html page is manipulated to show only the buttons that contain the cuisine of choice. Finally, the search bar of the page is made with the same code, as the search bar is entered with text and upon the “enter” button is clicked the “food-list” only shows the places that satiate the search.

User location

Since our app included a map and various other navigation features, getting a user's location was extremely important. This has allowed us to more effectively convey information to the user by initially showing them information about their location. This will provide a better user experience by not making them have to search the app to find the information they require. As well as this, showing the nearest transport links is only possible by having the user’s location. We implemented getting the users' location by using the geolocation API available for all modern browsers on mobile and desktop. This allows users to opt out of location services if they wish. However, the app's functionality as of now depends on the user location for the ultimate user experience.

Filters

When creating our map, we quickly encountered the challenge of having too much information to display to the user on screen. One way we chose to combat this was to enable filters for different types of landmarks to show on the screen. This way, users can find the specific types of historical monuments more quickly. To do this, when clicking on the legend, the landmarks will show and hide based on a user's click, and the types of landmarks currently being displayed can be seen in the legend.

Transport Links

One of the main ways people commute in the city is via public transport so we thought it was very important to include a way for people to find methods of public transport near them. What ended up being implemented was for the user to see only the tram and bus stops in a 2km radius around their location. This is because there were far too many bus stops to be loaded all at once, causing slow loading of more than 5 seconds. As well as this, the close transport links are filtered out on page load to increase the clarity of the markers on the map.

Another part of this feature that was added was the ability to see how far the closest transport link is in an infobox at the top of the page. This will help the user figure out what method of transport would be most convenient to get to. As well as this, it will allow them to see which bus or tram stop is closest to them if they need to get to one quickly.

A previous implementation of the close transport links feature was on a separate page within the app with added GUI elements like a 2km radius circle showing the range transport links would be shown and markers to display the closest tram and bus stops clearly. However, this page was removed in order for the map screen to be more cohesive instead of having 2 pages for it. In the future, more types of transport could be added to the map (train, rental e-scooters) giving a dedicated close transport links page more purpose.

Directions API

At first when we wanted to implement the public transport locations to the map, we instantly knew that adding a directions page to our application would be crucial. We discussed it briefly and decided to use the google directions API, a powerful tool that can be used to retrieve directions and distance information between two or more points. Initially the implementation of the directions API was through a simple directions page which would take input of an origin and a destination address and return the route on map. The implementation of this page was fairly straightforward after using the provided information on how to use the API to learn its intricacies. Further forward in the development it was decided that the implementation would be accessed through to the main map, and work as a button add-on for the pins. This was then a slightly more difficult uptake as it required the app updating the user's location frequently in order to add an accurate set of directions to a user. This however was remedied by changing from buttons on every pin, to one button on the overlay which redirects the user to the directions page. After implementing this new method, a back button was added to the directions page so that it was easy for the user to return to the map page.

Changes

Firstly, we decided as a team that a static word search was best for that time and it was discussed to instantiate a dynamic word search created upon user refresh, a feature which was axed later on. The static word search was created using an online word search generator tool, and two were made for both modes.

As we developed the map, it quickly became overloaded with features and symbols that some users may have found difficult to read or use. This led to the development of making the map screen look different for types of users (both children and adults). Some features such as car parks aren't useful to display for children, so removing them makes the user experience feel more personal and suitable for them. The map colours also look different and more vibrant in kids' mode compared to normal mode, which children will enjoy more.

Challenges

Unexpected problems were encountered during the implementation of the favourite button. The import for the Firebase utilities needs to be implemented as a module, but the JavaScript for the favourites cannot be a module due to its dynamic nature. Additionally, modules in JavaScript run before anything else and the scope of its variables is local to the module so we were unable to access the required Firebase functions. To solve this, we attached the Firebase utilities to the window to use them outside of the module.

While working on the project, we encountered occasional difficulties in assigning tasks appropriately, which could be attributed to the planning we had done far in advance. One instance of this would be our lightbulb feature, which includes clicking on a lightbulb and having a fun fact appear in the blurb of a landmark. This task was originally allocated to Team 2, but after finishing the favourites page and creating the "heart" button, it was better suited to Team 1.

The method to create this icon was very similar to the heart icon. The bulb was created with a CSS class and had a function which performs the "upon click" action. The function in turn would iterate through the landmark database, locate matching keys of the landmark clicked and display the fun fact.

When it came to the initial web scraping of the app, we encountered an issue in retrieving the data in the first place. Our original idea was to retrieve the data in real-time from an API. This allowed us to completely forgo the need for a data pipeline to update the data of places to eat. Unfortunately with this, we encountered an error when attempting to load the results on the webpage, a “CORS” issue. After much time attempting to make it work, it was mutually decided to stick with our firebase approach but decided that if the project goes further, using the previous method would be more efficient to do.

Regardless, using the Firebase method of instead collecting data and storing it in the Firebase and then iterating through helped with the cohesion of the app, as all the data we have for the app is stored in one place.

Back-end implementation

Initial Developments

For the KISS version of the application, we thought it was best to implement the backend as a server utilising a REST API. This would allow the front end to retrieve the NCC open data without having to worry about the implementation of the database or develop the app with a serverless architecture which no members of the team had experience with. Additionally, we decided to implement a REST API as opposed to another API such as GraphQL since it is simpler to get functioning for the high-fidelity prototype.

To create the server, we used Java as well as the Spring Boot framework - this decision was reached due to all members having a great deal of Java experience and Spring Boot being one of the most popular frameworks for building REST APIs in Java.

Furthermore, we chose to utilise the University’s MYSQL servers to host our relational database since all team members had experience using these servers and SQL. On top of that, this approach is a quick and inexpensive way to prototype the app. Currently, the data stored in our database is from a collection of many of the different datasets available in the NCC open data, such as tram stops, landmarks, and leisure centres.

Our software's alpha and beta releases have been tremendously productive because they allowed us to fully incorporate a number of important features. In particular, we have integrated a user login system successfully, enabling users to create accounts and access personalised content, and we have added user-friendly features like filters, showing the accessibility of transport links, and a crucial component, the places to eat page.

Places to Eat Data Source

“Places to eat” is a feature that allows the user to view a brief list of restaurants surrounding their current location. It also displays the cuisine information and map markers for each restaurant so that tourists/travellers can make informed choices. This utilises the Yelp API for getting restaurant information based on user location. 2 approaches were trialled for the feature. The first approach uses the google maps API to get several restaurants in a specific radius of the current user location. This worked well, it returned relevant information about each restaurant. However, the cuisine for every restaurant wasn’t a data point. Therefore, a switch was made and the Yelp API is used. In addition to the cuisine, the Yelp API can accept more parameters and return an image URL. The overall benefit of switching the data source was relatively high.

The feature is implemented as a set of functions in Python (More about this in the software manual). Each function is responsible for 1 stage of data processing. E.gThe “filter” function takes information about a certain restaurant as an input dictionary and then outputs a dictionary containing the relevant information needed by the app. Apart from functions, there are some data structures which are configurations for the API. E.g. the URL, api_key and headers are written to initialise a connection with the API and “params” specifies the constraints with which the API needs to return data.

Keeping in mind the problems with the original ‘places to eat’ implementation, we have decided to implement a scaled-down version. There are only 2 places to eat tables in the database currently. One is a list of restaurants from A32 (Computer Science Laboratory) and the other is a list of restaurants in and around Nottingham. However, this is a temporary argument due to the limited availability of time and resources.

Changes

The university’s servers are very secure, making accessing the database hosted on the MySQL server very problematic. As a team, we have decided in consensus that migrating the database from MySQL to Firebase is paramount for the project. Firebase allows public access to the database regardless of user location whilst allowing developers to maintain the database regularly and with ease e.g. removing one data point would need a particular SQL query, whereas Firebase has a user-interface setup for developers to interactively edit the database on the web.

Web scraping is an industry-wide process that automates mundane tasks throughout different applications and programs. Implementing web-scraping in the data pipeline and places-to-eat feature would benefit the project immensely as the database can be updated regularly to reflect changes in the data hosted on the NCC OpenData platform. Furthermore, from the user survey, users have indicated that having “places to eat” is crucial to improving their experience visiting Nottingham. Web-scraping would allow us to provide this service by scraping the vast array of data on restaurants and eateries hosted on google maps and displaying the results in a well-ordered manner on the application.

A change to the map design was to limit the map area to Nottingham only. This allowed us to ensure our users would more easily be able to see the relevant content they were looking for and not get lost.

Challenges

A significant technical challenge we faced early on was that the data from the NCC website proved to need adapting as it contained multiple columns of redundant information such as “body name” and “create_dat”, these contained repetitive information that had to be removed entirely before proceeding with the conversion to MySQL. Furthermore, some of the files are available in an XML format rather than CSV, and some files only contain nothing and easting while others have longitude and latitude. Neither is a negative but acts as a setback when there is a lack of consistency between the files.

Initially, it was planned to dynamically update the tables the team has chosen, using the keyword search as a web scraping technique. However, this posed problems which needed a more complex design of error handling which our team wasn’t experienced in. Therefore, we decided that we would only update data in a set number of relevant tables from the NCC Open Data platform as the other tables could be included in a future version of the application if required. Although to plan

around this, we have also implemented the pipeline in a way that allows easy extension to more database tables. The majority of the data pipeline was a straightforward implementation assisted by the useful structure of the NCC Open Data platform. For instance, consistency in field names across all chosen tables made designing update methods simple as implementing a filter became a simple task.

The places to eat page was planned to incorporate an elaborate system of reading/writing to/from the Firebase real-time database. There should have been a script in Python listening for a change in the user location table in the Firebase. If there was a change then, the relevant data would be written to the database. At the front end, there would've been a function listening for a change in the pushed database table, once there is a change, it would've read the updated table and displayed the results on the app accordingly. This approach would work in theory, however, due to scalability and concurrency issues, it was not followed through. Firstly, if there were a large number of users then, that would result in one table of places to eat per user which is an inefficient use of resources. Secondly, if a large number of users were trying to access the feature at the same time could cause a significant subset of them to wait for a period which isn't very practical when immediate results are needed.

Implementation testing

Our initial approach during the implementation of the KISS version was conducting defect testing for the back end to ensure the software produces the correct results. We had confirmation that everything was running smoothly, and the tests were passing with the expected output. After conducting unit testing, we planned on implementing integration testing to guarantee the functionality of the front and back end as a group. We had the intention of following up with unit and integration testing after every version to verify the software's performance.

Due to changes in our design, the migration of our database, and the addition of several key features, our testing approaches have changed. Originally when searching for a testing framework for our app we decided to use the Jest framework for unit testing. However, it quickly became apparent that with our highly dynamic JavaScript and HTML, it would prove difficult to test in a modularised manner.

From further research into testing frameworks for web applications, we concluded that the Cypress framework best matched our requirements, as it allowed us to verify that the software functions correctly in real-world scenarios through the use of end-to-end (E2E) testing. Furthermore, we could test the more dynamic aspects of our project, e.g. quiz and game functionality, more effectively.

Thus, we utilised the Cypress framework to write E2E tests which can be tested across different browsers consequently ensuring browser cross-compatibility. The test plans for our web application are available in the "Testing" section of the software manual.

PROJECT MANAGEMENT

Tools Used

Our team utilises a wide variety of project management tools to ensure we are following the timeline outlined for the project and to guarantee that all team members are on the same page in

terms of the work to be done. However, our main way of communicating tasks is through Trello which is managed by our team leader**.**

This is updated throughout the week as a task is completed. Alongside that, we have a Trello for user stories and the project roadmap which includes brainstorming points for the project's timeline. These are later organised into a Gantt chart format for a strict period allotted to each task **(Figures 10 and 11)**. Furthermore, these are sorted into git milestones and issues by our git lead.

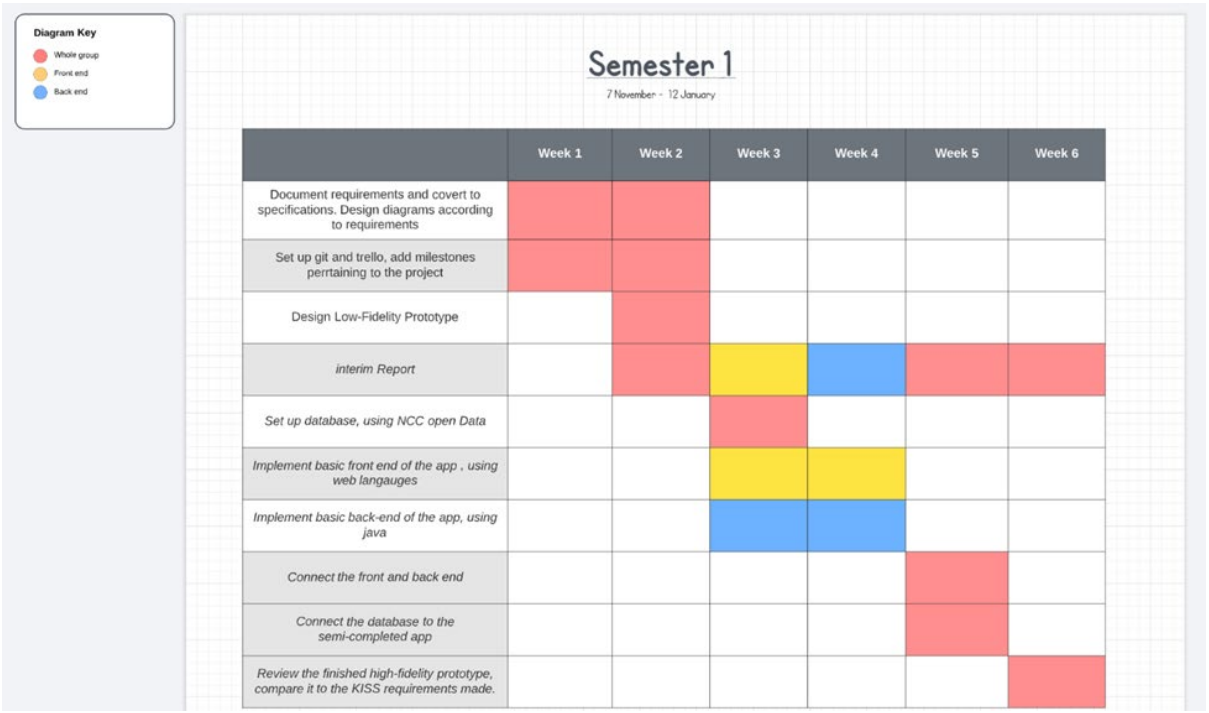

Figure 10 - semester 1

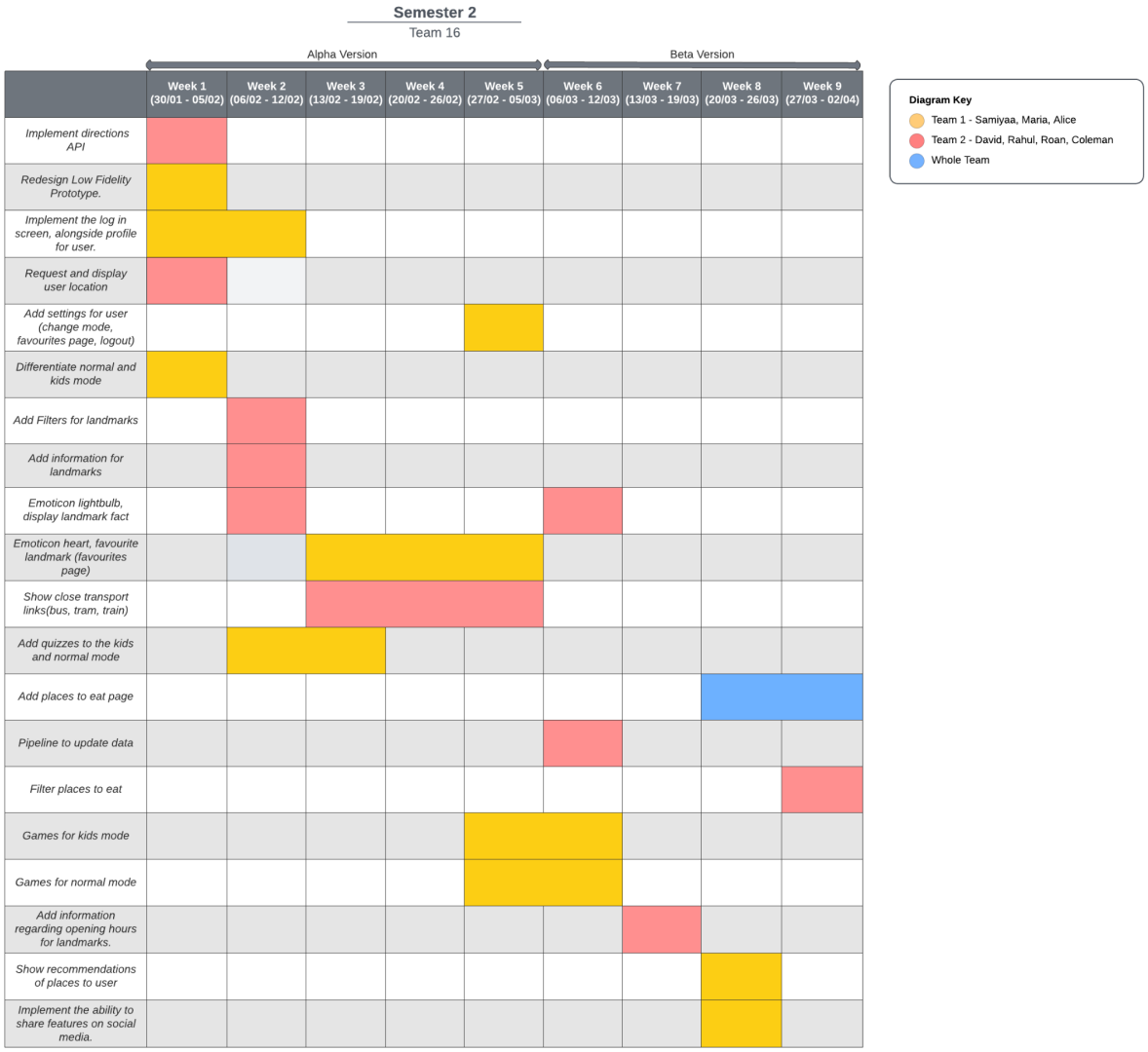

Figure 11 - semester 2

For daily check-ups, regular stand-up meetings, and sprints, we as a group use a discord server that is categorised into channels sorted by relevance. All group resources are uploaded there before approval by the sponsor, after which they are uploaded to the team's google drive.

How do we use version control to our benefit?

Designing our branching model strategy for Git was the next step towards the start of development. After conducting research, two models that might be of interest have been identified, these being GitLab flow and GitHub flow. **Figure 12** presents a combination of these branching models, which have been tailored specifically for our project. This model suggests a "main" branch, a separate "develop" branch with supporting "feature" branches, and "maintenance".

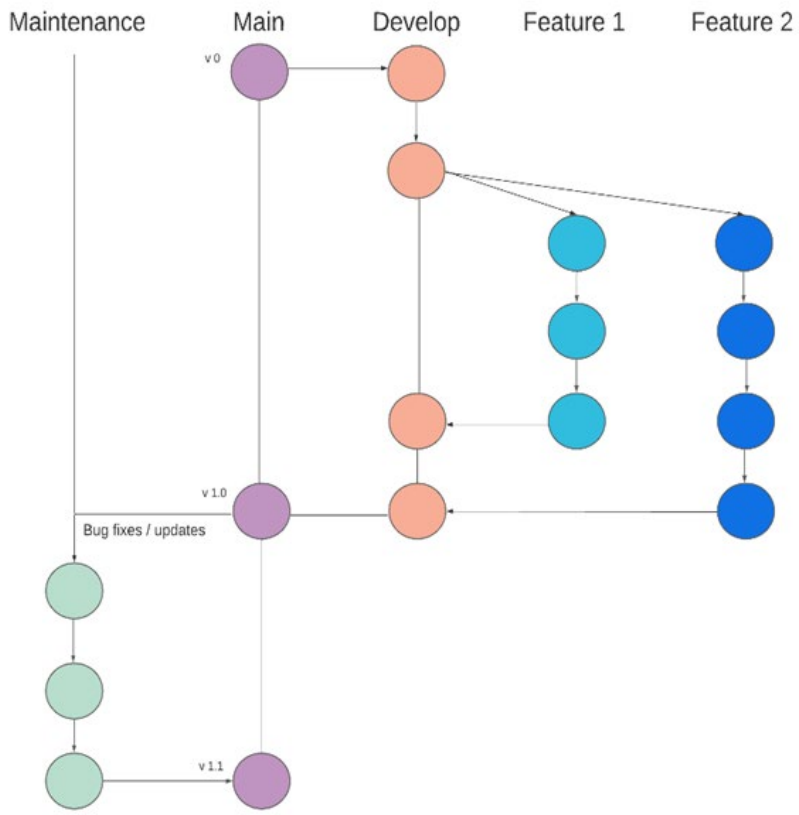

Figure 12

“Main” is reserved for code that is ready for release, while “develop” is used for storing the work in progress. The feature branches are being used by each sub-team to support concurrent, parallel development. As “maintenance” suggests, this branch keeps track of fixes and changes to the code that is ready for release.

Our merging approach consists of assigning other team members to review and approve the changes to be made before pushing. We have utilised GitLab to support the project development since it allows file transfer back and forth and allows us to manage version history.

The upcoming development plan is ready on git in the form of milestones. We plan on implementing the main features of the application which include transport information, landmarks, quizzes, and games.

How does our team work together?

All team members maintained a professional attitude when approaching the project, shown through our communication, commitment, and discipline. Communication for the project is done through a discord server as well as constant and consistent group meetings to help plan the next steps in this group collaboration, a similar approach to last semester. Interactions with the sponsor are also planned before deciding what to discuss to ensure the meeting is as productive as possible. All members show commitment to the project by being punctual to group meetings and meeting deadlines for tasks set within the group. This shows how we are an autonomous group as we can plan and carry out our tasks independently.

Different members of the team have different strengths and weaknesses, and to compensate for this we have implemented cross-training to share our knowledge. Within each subgroup, members have communicated their understanding through code reviews which have the additional benefit of assuring the quality of the code.

The front-end subgroup has frequent meetings in which discussions about design ideas and implementation details of “in-progress” features. Some members have more experience with front-end languages and frameworks and aided other members in planning and implementing their designated tasks. As a consequence of this, by the end of the beta release, every member was confident in their abilities to write in HTML, CSS, JavaScript, etc.

Adhering to our agile software engineering principle, we adopted a flexible approach when allocating tasks, regardless of whether team members specialised in backend or frontend development. By integrating both front-end and back-end skills, this approach not only saved time but also allowed team members to specialise in their assigned features, resulting in a higher quality of work. Consequently, this eliminated the need for detailed documentation and allowed team members to work more efficiently.

Compared to the previous semester, we improved our task allocation strategy, resulting in faster turnaround times and higher-quality work. This semester, we adopted a different method where both sub-teams updated a shared document with details about the methods used in their features and any challenges encountered during development. In team sprint meetings, we discussed this document to identify problems and brainstorm solutions to improve our processes. Additionally, if a feature was divided among two or more members, we heavily documented the code to ensure quick understanding, including detailed explanations of any used functions **(Figure 13)**.

```
/**
 * Creates a Google Maps marker for each item in a table in the
database. Uses longitude, latitude and name field of database entity.
 * @param {*} url api endpoint url
 * @param {*} icon image to be displayed for the marker
 * @param map Google maps map object
 * @returns Array of Google Maps marker objects
 */
```

Figure 13

Progress made so far

According to our Gantt chart, by the end of the first semester, we aligned with what it portrayed. We did stumble across a few hiccups and not all our KISS requirements were executed to the level we had thought at the beginning of the term. However, our team expected some ideas for the app to be axed due to time constraints. Nevertheless, by planning to use what we have learned thus far we hope to fast-track our project for the next phase of our plan.

During the development of the alpha version, our team was on track with no major issues according to the Gantt chart. We easily reached our deadlines and satisfactorily deployed critical features. However, by the end of the beta version, our team encountered significant setbacks that threatened to derail our progress. Despite our best efforts, it seemed like we could not resolve some critical bugs and issues preventing us from completing an essential feature. As a result, we considered axing this feature to ensure we can meet our overall project deadline. While we were

disappointed by this setback, we were still committed to creating a high-quality product. We overcame these major issues by looking for alternative options and making debugging a team objective, actions which we took when encountering the “CORS” error.

PRELIMINARY REFLECTION

Challenges and lessons learnt

No project comes without its challenges and the volume of work for this project and other coursework, consequently led to our development taking longer than anticipated. According to the Semester 1 Gantt chart, especially towards the end, we fumbled with the front and back-end connection due to a communication error. However, this situation did not take long to remedy since as a group we listed down tasks to be completed in priority order and were able to complete them with high-level collaboration, this made an immediate impact as the progress continued to be on track.

A concern identified in the development process was the lack of communication/understanding between the front and back-end groups. Moving forwards, we planned to put in place more meetings between the sub-groups and utilise skill transfer so there is more understanding from both ends of how the other works to combine future implementations more effectively and seamlessly.

During the second semester, we had occasional absences from members of the team. This affected the delivery of certain features at the time planned on the Gantt chart. In order to deal with this effectively, we distributed the remaining work between present members according to our strengths. This allowed us to complete the remaining work as fast as possible. In the future, we will ensure that more time than we need is allocated to certain features, to guarantee that features are more likely to be completed on time if anything unexpected happens.

CONCLUSION

Retrospective

By the end of the first semester, we had successfully designed a high-fidelity prototype integrating HTML, CSS, and JavaScript as our front-end languages with Java as our back-end language. We as a group were able to utilise the NCC open data datasets by utilising the coordinates provided to produce markers.

It is important to emphasise the importance of the Nottingham City Council Open Data platform and its detrimental function in our projects’ development and other fields. Information about transport, housing, attractions, events, education, public health, crime data, and more is available

on the NCC Open Data website. The site can be helpful for many different fields and categories because it has access to such a plethora of data. Researchers, analysts, and developers may all benefit from the data to learn more about many elements of the city, whether it be to boost tourism, create better public health policies, or create new software and tools for the benefit of the city.

After extensively working with the NCC Open Data website, our team has suggested a few improvements that could be done to the platform to make it more useful. Updating the data more frequently would be one way the website could be more helpful for example if all data on the website was provided in the form of an API, it would prove to be highly advantageous for developers as it would eliminate the need for a pipeline.While gathering datasets for the travel app, the team noticed that some of the data sets have not had their frequency of update determined, while other data sets are hardly being updated, thus calling into question the accuracy of the data.

Reducing the amount of redundant data in the data sets would also be beneficial. Fields such as BODY, create_dat and such cannot be used to deduct relevant information, and filtering through the data can be tedious.

In addition to this, creating documentation on how to access and use the data would also be a good way to improve the platform. At some point through the refinement of the data, the team ran into an issue where the data sets were stored in different file formats, thus making it difficult to extract the necessary information.

However, despite the technical and communication challenges we faced as a team, we managed to overcome them and move on to the next version.

The external source of data used for the places to eat page is the Yelp API. It proved to be very useful for the app as it consists of relevant details that users require. Additionally, there is an abundance of data in Yelp’s database making it a long-term solution for extending the places to eat page. Furthermore, this shows that the application is flexible to extract and use data from any source. In the future, we can utilise any other APIs to extend our games,quizzes and any other features we would like to add.

During the planning of requirements, we as a team decided to implement an alpha version of the app in order to test and ensure the basic functionalities of the software are working as they are intended to do so and also to quickly identify and fix issues and bugs in our software. This is the main reason for us placing more emphasis on functional requirements when implementing the alpha version as we wanted to make sure the basic functionality of the app was working as intended. When planning the beta version we included more non-functional requirements such as the ability for users of our app to share it on social media to increase awareness of the app.

There were many challenges during the transition from alpha version to the beta version. The main challenge was to make the app more stable and less susceptible to crashes and bugs, and overall making the app more reliable to use. One of the requirements for the beta version was to update the data for the app as it updates, hence making the app more reliable for our other requirements, such as showing the user opening times of different venues. Considering that we are also implementing an app, it was very challenging in ensuring our app runs on different platforms and devices

As the development of the project came to an end, we are pleased to report that our team has completed all of its project-related goals. Implementing a "Kids Mode," which is significantly different from the usual mode and provides a unique experience for younger users, is one of our

key accomplishments. In addition to this, we have successfully implemented all the key features we set out to develop which are: a custom map, landmark information, user credentials, quizzes, games, user location tracking, transport links, favourites page, places to eat page, and filters for major functions. To ensure that our software can be readily updated and developed in the future, we have also maintained the project's core principles, which are maintainability and extensibility. Moreover, we have successfully implemented a data pipeline for the NCC Open Data platform, allowing the web application to function based on current data.

Overall, the team has achieved a great deal and has successfully met all the goals we set out to accomplish and look forward to further enhancing our product in the future.

Future Plan

Our team has various ideas on how the app will be developed in the future. One of the main objectives is implementing a "Guest Mode" that enables users to use some of the app's functions without registering an account. Additionally, we intend to make the app available for advertising, enabling companies and establishments in Nottingham to market their goods and services to our users. This will benefit Nottingham by providing users helpful information about nearby services, thereby generating additional revenue streams for tourism.

We also have plans to create a chatbot that utilises machine learning to provide personalised recommendations on features, and help users navigate the app. As our app grows in size, it would be beneficial to enhance our security measures. Including a "forgot password" option and sending confirmation emails when users create their accounts can significantly improve the overall safety and security of the app.

Moreover, for an enhanced experience in both modes, it would be desirable to include additional games and levels in our app.

In conclusion, the app has the potential to be customised for other cities such as Lincoln and Manchester by utilising the open data websites available for each city. To take it a step further, the app could be expanded internationally with an "Adventurer's Guide to the World."