National College of Ireland

Team Project

Group 1- Technical Report

14th April 2024

2023/2024

Contents

Demo Video | Code Repository | Deployed Website

Executive Summary

This report is an overview of a Technology project called WhatsOn, detailing its development process, challenges encountered, and outcomes achieved. The use of various technologies is highlighted, covering all areas of the project. A detailed use case explaining all the application's proposed functionalities and the results achieved. Challenges in the development process were faced with API integration and security but the project successfully delivered its core features such as dynamic event loading on maps, filtering, and random event functionalities. However, limitations in securing API keys and reliance on external services are acknowledged.

Key points of the report include:

1. Challenges with Meetup API integration, including the need for trial or paid accounts and token expiration issues.

2. Security concerns related to the Google Maps API key, highlighting the risk of exposure.

3. Description of the graphical user interface (GUI) features, such as dynamic pin loading, event differentiation, and responsive design.

4. Testing approach, including black-box, white-box, and unit testing strategies using tools like Selenium, Mocha, and Supertest.

5. Conclusions on achieving project goals despite challenges, emphasizing the valuable learning experience gained.

6. Recommendations for further development, such as using OpenStreetMaps API, enhancing existing functionalities, and developing a native mobile app.

7. Collaboration summary, detailing the tools and platforms used for effective teamwork and project management.

In conclusion, the report underscores the project's achievements, identifies areas for improvement, and emphasizes the importance of continuous learning and adaptation in software development projects.

1.0 Introduction

1.1. Background

We undertook this project because we felt there was a gap in the market for an event discovery application that simply notified users of activities that were available to them. Upon conducting market research we discovered this to be the case. Although there are many event management platforms with ticketing and the ability to create events, there is not a strictly event discovery platform. We felt that there was a need for a simple application with an emphasis on removing the friction of doing activities with an intuitive and visually engaging way to discover events.

We found that other apps employ a traditional list format with layers of pages and planning expected of the user, usually including signing up for an account and searching. We decided to use a single uncluttered and consistent design with events appearing directly on the map itself with only the location permission of the user required. We believed seeing events on a map makes people realise how accessible and local they are to you, instead of a time and date with an address on a website.

We identified barriers such as decision fatigue, anxiety associated with trying new things or the FOMO (Fear of Missing Out) associated with missing things. As such we consciously chose to design our features to encourage spontaneity and action. We limited our application to real-time events that are current and removed them afterwards. We wanted this app to be a welcoming user experience instead of another source of digital anxiety.

To break through indecision and inertia we thought of the Random event feature. If a user can't decide on what to pick, the random button will present an event for them each time it is pressed. This has a slot machine characteristic of the most popular social media sites such as Snapchat, TikTok and Instagram.

We undertook this project because we believed WhatsOn had great potential because of our unique approach to capitalising on current and future trends in the events industry. With a singular vision that permeates the entire application, from colour scheme, layout, font, copy, and features.

1.2. Aims

This project aims to produce an application that satisfies the user needs we identified in our research. To accomplish this we have to need to execute several functionalities in a seamless user experience.

The scope of the project is to develop map-based event discovery web application called WhatsOn. A user need only open the application, and allow location permissions and, at a glance, they will be able to see all the current events going on around them without the need to sign up or search. They can click through to the source site if they are interested. If a user can't decide on what to pick, the random button will present an event for them. Pins on the map will indicate the type of event it is and the distance from the user. If a user wants to see only a certain type of event, they can click on the icon event to see only events that meet that description. Users will have the option to sign up for an account if they would like to follow recurring events or specific organisers.

1.3. Technology

Frontend

HTML- to define the structure and content of the webpage

EJS- easier to manage HTML files and to take in live injection of data such as JSON data

CSS- to style all the HTML content

JavaScript- is used considerably to achieve all use cases. To load and initialize the map, create markers, info windows, grabbing the users location and the event data, etc

Backend

Node.js- backend server used to load the webpage and and domain services such as Render

Render- to deploy website and be validated for a Meetup API key

Express.js- as a Node.js library to handle backend requests

Morgan- check code status, in terms of loading, resources, 404 errors, etc

Bcrypt- library to encrypt user passwords

Crypto- encrypting google maps API key

Axios- webcrawler- makes http calls to server to pull data

Cheerio- to parse and read that data

FS- to write that data to a JSON file

APIs

Google Maps API- for map content

Maps JavaScript API- to load and manipulate map content

Google Cloud Console- to create API keys, map styles, security restrictions, etc

Graphic Design

PhotoPea- icon design

Figma- wireframes and mockups

Pixelmator Pro- icon design

Project Management

Trello- to assign and manage tasks

Microsoft Teams- to communicate and transfer files

OneDrive- to store shared documents and files

1.4. Structure

Included in this document are the overall requirements of the system. From user requirements to technical requirements and use cases, responsibilities of the system in terms of users, data, usability and requirements from the environment to work.

Next the design and architecture are fleshed out through several diagrams and detailed descriptions from higher to lower level. Implementation is showcases some key snippets of code and a discussion on difficulties encountered, solutions to overcome them. Different views of the GUI are the shown demonstrating the fucnitonalit of the system.

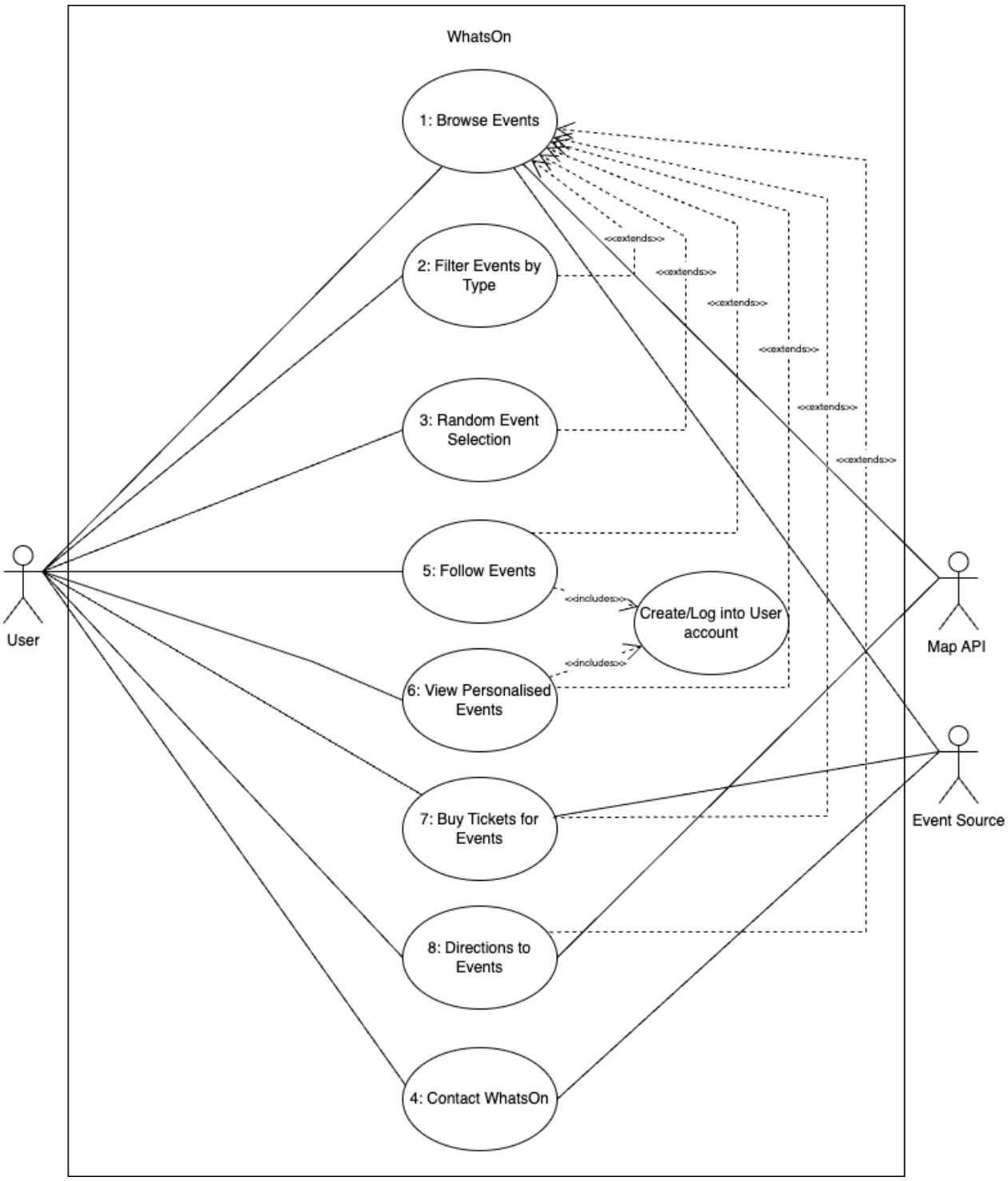
Testing is addressed with suggested test tools, a test plan and black, white and unit testing. Conclusions from the project as a whole with strengths and weaknesses discussed. Further development considers what we would do with additional time and resources and also with the benefit of hindsight. It concludes with detailed project planning and collaboration summaries.

2.0 System

2.1. Requirements

1. **Browse Events:** Enable users to browse events on a map-based user interface.
2. **Filter Events By Type:** Allow users to filter events by type.
3. **Random Event Selection:** Provide users with the ability to randomly select events.
4. **Contact WhatsOn:** Allow users to contact WhatsOn for inquiries or support.
5. **Follow Events:** Enable users to follow recurring events.
6. **View Personalised Events:** Implement personalized event viewing based on user preferences.
7. **Buy Tickets for Events:** Facilitate ticket purchasing for events.
8. **Get Directions to Events:** Provide users with directions to events.

2.1.1. Use Case Diagram



2.1.1.1. Requirement 1: Browse Events

2.1.1.1.1. Description & Priority

Browsing events on a map-based UI is crucial for the app's USP and differentiation from competitors. It forms the core functionality of the app and is essential for user engagement.

Priority: Highest.

2.1.1.1.2. Use Case

Scope

The scope of this use case is for a user to browse events within the WhatsOn app.

Description

This use case describes a user browsing events within the WhatsOn app.

Flow Description

Precondition

1. The user has location services enabled on their device.
2. The user has granted location permission to the WhatsOn App.

Activation

This use case starts when the user launches the WhatsOn app.

Main flow

1. The system retrieves the user's location using GPS/IP geolocation.
2. The system displays a map user interface centred on the user's current location.
3. The system retrieves current events from external API relative to the user's location.
4. The system represents events as pins with an icon type.
5. The user can interact with the map by dragging across the map or zooming in/out with a cursor/finger.

6. The system displays events dynamically based on visible area.
7. The user clicks/taps the event icon.
8. The system displays a pop-up window with event information such as:
    - Event name
    - Event type
    - Event venue/location
    - Event date/time
    - Event description
    - Event host
    - Event accessibility information
    - Event price
9. The user can exit pop-up window by clicking/tapping outside the window.
10. The system closes pop-up window.

Alternate flow

A1: User's location cannot be determined.
1. The system prompts user to enable location services and grant permission to the WhatsOn app.
2. The user enables location services and grants permission to the WhatsOn app.
3. The use case continues at position 2 of the main flow.

A2: User's location cannot be determined persists.
1. The system displays a default view centred around a predefined location.
2. The use case continues at position 3 of the main flow.

Exceptional flow

E1: Error retrieving event data
1. The system notifies the user and prompts them to retry fetching the data.
2. The user clicks/taps the retry button.
3. The use case continues at position 4 of the main flow.

E2: Error retrieving event data persists.
1. The system notifies the user that data is currently unavailable.
2. The system displays a default map user interface.

3. Once the issue is resolved, the use case continues at position 4 of the main flow.

Termination

The user exits the “Browse Events” use case by navigating to a different screen or closing the app.

Postcondition

The system defaults to position 2 of the main flow.

Result

We achieved all the steps of the Main/Alternative Flow with some small alterations. We were not able to use an API but instead used Webcrawler to scrape event data from event sites. Events did not populate dynamically based on the user’s location but instead based on the filters we applied to the webcrawler method. We did achieve error correction for location and event data but it displayed only in the console and not to the user. We did add additional functionality such as event pins not only being differentiated by type but also time status relative to the user.

2.1.1.2. Requirement 2: Filter Events by Type

2.1.1.2.1. Description & Priority

Filtering events based on predefined types enhances user experience by allowing users to narrow down their preferences without the need to search or type. This requirement further differentiates WhatsOn by creating an uncluttered and seamless user experience.

Priority: High.

2.1.1.2.2. Use Case

Scope

The scope of this use case is for a user to filter events to display only a defined type within the WhatsOn app.

Description

This use case describes a user filtering events by type within the WhatsOn app.

Flow Description

Precondition

1. The user has clicked/tapped an event icon.
2. The system displays a pop-up window with event information.
3. Or the user has clicked/tapped the “Legend” Button.
4. The system displays a pop-up window with icon information.

Activation

This use case starts when the user clicks/taps the “Type” button within the event information pop-up window or the “Legend” pop-up window.

Main flow

1. The system exits the pop-up window.
2. The system displays events that match the selected type within a defined range of the user.

3. The user can exit the filter by clicking/tapping on an event to open the event information pop-up window or the “Legend” button to open the “Legend” pop-up window.
4. The user clicks/taps the “Type” button within the event information/legend pop-up window.
5. The system terminates the filter and displays all events within a defined range of the user.

Alternate flow

A1: No events match the selected type from the “Legend” pop-up window.
1. The system notifies the user that no events match the selected type.
2. The system displays all events within a defined range of the user.

Exceptional flow

E1: Error retrieving event-type data.
1. The system notifies the user and prompts them to retry fetching the data.
2. The user clicks/taps the retry button.
3. The use case continues at position 4 of the main flow.

E2: Error retrieving event data type persists.
1. The system notifies the user that event data is currently unavailable.
2. The use case continues at position 6 of the main flow.
3. Once the issue is resolved, the use case continues at position 3 of the main flow.

Termination

The user clicks/taps the “Type” button within the event information/legend pop-up window. The system terminates the filter and displays all events within a defined range of the user.

Postcondition

The system defaults to the “Browsing Events” interface, displaying all events within a defined range of the user.

Result

We achieved all the steps of the Main/Alternative/Exceptional Flow with one large exception which caused all the other exceptions. We decided to filter events only from within the info window as opposed to also filtering from the legend. Due to this, there was no need for any error correction if an event didn't match a type because at least one event of that type would be required to filter from its info window.

2.1.1.3. Requirement 3: Random Event Selection

2.1.1.3.1. Description & Priority

Random event selection is crucial for creating an addictive novelty factor while addressing barriers to spontaneous socialising like decision fatigue and avoidance.

Priority: High.

2.1.1.3.2. Use Case

Scope

The scope of this use case is for a user to randomly select an event within the WhatsOn app.

Description

This use case describes a user randomly selecting an event within the WhatsOn app.

Flow Description

Precondition

1. The user has location services enabled on their device.
2. The user has granted location permission to the WhatsOn App.

Activation

This use case starts when the user clicks/taps the "Random" button within the WhatsOn app.

Main flow

1. The system randomly selects from the available events within a defined range of the user.
2. The system displays a pop-up window with event information such as:
    - Event name
    - Event type
    - Event venue/location
    - Event date/time

- Event description
- Event host
- Event accessibility information
- Event price

3. The user can click/tap the "Random" button again.
4. The system randomly selects another available event within a defined range of the user.
5. The user can exit the pop-up window by clicking/tapping outside the window.
6. The system closes the pop-up window.
7. The system displays all events within a defined range of the user.

Alternate flow

A1: No events available for random selection.
1. The system notifies the user.
2. The System defaults to the “Browse Events” UI, displaying all events within a defined range of the user.

Exceptional flow

E1: Error retrieving event-type data.
1. The system notifies the user and prompts them to retry fetching the data.
2. The user clicks/taps the retry button.
3. The use case continues at position 2 of the main flow.

E2: Error retrieving event data type persists.
1. The system notifies the user that event data is currently unavailable.
2. The use case continues at position 8 of the main flow.
3. Once the issue is resolved, the use case continues at position 2 of the main flow.

Termination

The user exits the “Random Event” use case by clicking/tapping outside the event pop-up window.

Postcondition

The system defaults to the “Browsing Events” requirement interface, displaying all events within a defined range of the user.

Result

We achieved all the steps of the Main/Alternative/Exceptional Flow except notifying the user based on lack of event retrieval. This was decided because this would only occur if events to not appear in the Browse events use case and so it is redundant.

2.1.1.4. Requirement 4: Contact WhatsOn

2.1.1.4.1. Description & Priority

Enabling users to contact WhatsOn directly within the app is essential for gathering user feedback, addressing concerns promptly, and fostering partnerships with events for business opportunities.

Priority: High.

2.1.1.4.2. Use Case

Scope

The scope of this use case is for a user/event to contact WhatsOn for inquiries or support within the WhatsOn app.

Description

This use case describes how a user or event interacts with the contact feature within the WhatsOn app to send inquiries to WhatsOn. We will refer to both actors as user for simplicity.

Flow Description

Precondition

1. The user clicks/taps the "Contact" button on the main UI.
2. The system displays a pop-up window with a form for entering an email address, a text field for the message, and a "Send" button.

Activation

This use case starts when the user/event clicks/taps the "Contact" button on the main UI.

Main flow

1. The user fills out the email address input form and the message text field.
2. The user clicks/taps the "Send" button.
3. The system sends the inquiry/message to the WhatsOn support team.
4. The system displays a confirmation message to the user indicating that the message has been sent successfully.

Alternate flow

A1: User decides not to send message.
1. The user clicks/taps the "Contact" button on the main UI.
2. The system displays a pop-up window with a form for entering an email address, a text field for the message, and a "Send" button.
3. The user closes the contact pop-up window use case by clicking/tapping outside of it to return to the default UI.

Exceptional flow

E1: Error sending message.
1. The system notifies the user and prompts them to retry sending the message.
2. The user clicks/taps the “Send” button.
3. The use case continues at position 5 of the main flow.

E2: Error sending message persists.
1. The system notifies the user that the contact feature is unavailable.
2. The system displays a default map user interface.
3. Once the issue is resolved, the use case continues at position 5 of the main flow.

Termination

The user exits the "Contact WhatsOn" use case by clicking/tapping outside the contact pop-up window to return to the default UI.

Postcondition

The system defaults to position 4 of the main flow.

Result

We achieved all the steps of the Main/Alternative Flow except but did not include notifying the user in the case of an error. However, we did include notifying the user to include a real email address.

2.1.1.5. Requirement 5: Follow Events

2.1.1.5.1. Description & Priority

While useful for users interested in specific recurring events, following events is not essential for all users and can be considered as supplementary functionality.

Priority: Medium.

2.1.1.5.2. Use Case

Scope

The scope of this use case is for a user to follow recurring events within the WhatsOn app.

Description

This use case describes how a user interacts with the "Follow" feature in the event pop-up window to follow recurring events. If the user is logged in to their account the event will be added to their “Following” list. If the user is not logged/ registered they will be prompted to log in/create an account.

Flow Description

Precondition

1. The user may or may not be logged into their WhatsOn account.
2. The user has clicked/tapped an event icon.
3. The system displays a pop-up window with event information.

Activation

This use case starts when a user clicks/taps the "Follow" button within the event information pop-up window for a recurring event.

Main flow

1. If the user is logged in:
    - The system adds the event to the user’s followed events list.

- The user can view/edit their following list by accessing by clicking/tapping the “Following” button.
- The system displays a heart icon above the event when it has been followed.
- The user can click/tap a heart icon to see only followed events.
- The system will display only followed events on the map.
- The user can unfollow an event by clicking/tapping a previously clicked event.
- The system will remove that event from the following list.

2. If the user is not logged in:
    - The system prompts the user to log in/create an account to follow the event.
    - The user clicks/taps on the prompt.
    - The system adds the event to the user’s followed events list.

Alternate flow

A1: The user selects the option to create an account.
1. The use case continues at position 1 of the main flow of Requirement 6: View Personalised events.
2. The use case continues at position 1 of the main flow.

A2: User forgets password for account.
1. The user clicks/taps The use case continues at position 1 of the A2 flow of Requirement 6: View Personalised events.
2. The use case continues at position 1 of the main flow.

A3: No events are recurring.
1. The system will not display the “Follow” event button within the event information pop-up window.

Exceptional flow

E1: Error processing the user's request to log in or create an account.
1. The system notifies the user and prompts them to retry log-in/account creation.
2. The use case continues at position 2 of the alternate flow.

E2: Error processing the user's request to follow the event.

1. The system notifies the user and prompts them to retry following the event.
2. The user clicks/taps the retry button.
3. The use case continues at position 1 of the main flow.

E2: Error processing the user's request to follow the event persists.
1. The system notifies the user that the follow event feature is currently unavailable.
2. The system defaults to event info pop-up window interface.
3. Once the issue is resolved, the use case continues at position 1 of the main flow.

Termination

The user exits the “Follow Events” use case by tapping either clicking/tapping the “Follow” button to unfollow an event or by removing the event from the “Following” pop-up window.

Postcondition

The system defaults to the event pop-up window, displaying the event information. If the user has just followed the event, the system displays a heart icon above the event to indicate it has been followed.

Result

We overall did not achieve the functionality we hoped for in this use case. The initial problem was that there was no easily usable filter from any APIs that we investigated for this feature. When we switched to making our own JSON from webcrawler data we did find a way to achieve a moderate following feature by listing and displaying events from an event organiser. However, we were not able to achieve this functionality to execute from the info window.

As this and the next use case deals with account creation and login I will address what we achieved here. We were able to achieve account creation, login, password verification, email validation and encryption of user emails. Once a user is logged in the window does switch to the following info window view although it does not populate.

We achieved all the steps of the Main/Alternative Flow except but did not include notifying the user in the case of an error. However, we did include notifying the user to include a real email address.

2.1.1.6. Requirement 6: View Personalised Events

2.1.1.6.1. Description & Priority

While personalized events contribute to user satisfaction and engagement, they are not essential for all users and can be considered as additional, optional functionality.

Priority: Medium.

2.1.1.6.2. Use Case

Scope

The scope of this use case is for a user to view events based on their personalised event profile within the WhatsOn app.

Description

This use case describes how a user creates their personalised event profile and views personalised events within the WhatsOn app.

Flow Description

Precondition

1. The user does not have a user account on WhatsOn.

Activation

This use case starts when a user clicks/taps the “Following” button from the main UI without an account.

Main flow

1. The system prompts the user to “Create account” by entering an email address and password.
2. The user enters an email address and password.
3. The system validates the entered information.
4. If the email address is already associated with an existing account, the system prompts the user to log in instead.
5. If the email address is not associated with an existing account, the system creates a new account for the user.

6. The system prompts the user to create their personalised event profile by selecting their preferred event types.
7. The user selects their preferred events.
8. The system saves the user's preferences in their user account.
9. The system retrieves the user's personalised event data.
10. The system displays a star above personalised events on the map.
11. The user can tap/click a star to see only personalised events on the map.

Alternate flow

A1: The user has an account and is not logged in.
1. The user clicks/taps the "Following" button from the main UI.
2. The system prompts the user to enter an email address and password.
3. The user enters an email address and password.
4. The system validates the login by comparing the email address and password to existing accounts.
5. The use case continues at position 10 of the main flow.

A2: User forgets password for account.
1. The system displays a "Forgot Password" button.
2. The user clicks/taps the button.
3. The system prompts the user to enter their email address.
4. The user enters their email address.
5. If the email address is already associated with an existing account:
    - The system sends a password reset link to the user's email address.
    - The user checks their email for the password reset link.
    - The user clicks/taps on the password reset link.
    - The system prompts the user to enter a new password.
    - The user enters a new password.
    - The system updates the user's password.
    - The use case continues at position 10 of the main flow.
3. If the email address is not associated with an existing account:
    - The system prompts the user to create an account instead.
    - The use case continues at position 1 of the main flow.

Exceptional flow

E1: Error creating an account.
1. The system notifies the user and prompts them to retry the account creation process.

2. The user retries the account creation process with a valid email address and password.
3. The use case continues at position 4 of the main flow.

E2: Error creating account persists.
1. The system notifies the user that the create account feature is currently unavailable.
2. The system defaults to the default UI.
3. Once the issue is resolved, the use case continues at position 2 of the main flow.

E3: Error retrieving personalized event data.
1. The system notifies the user that personalized event data is currently unavailable.
2. The system defaults to the default UI displaying all available events.
3. Once the issue is resolved the use case continues at position 10 of the main flow.

Termination

The user exits the “View Personalized Events” use case by logging out of their account.

Postcondition

The system defaults to the main UI interface, displaying all available events. If there are personalized events they will appear with a star icon above them.

Result

We did not achieve the functionality we hoped for in terms of account creation and alternative flows with exception handling as discussed in the prior use case. We did not achieve a user-personalised event profile as part of the user account creation process.

2.1.1.7. Requirement 7: Buy Tickets for Events

2.1.1.7.1. Description & Priority

WhatsOn does not feature ticketing or payment processing. However facilitating ticket purchases contributes to user convenience and satisfaction, making it a valuable feature without being as essential as core functionalities.

Priority: Medium.

2.1.1.7.2. Use Case

Scope

The scope of this use case is for a user to purchase tickets for events displayed on the WhatsOn app.

Description

This use case describes a user viewing an event within the WhatsOn app and being redirected to the event source site to buy tickets.

Flow Description

Precondition

1. The user clicks/taps an event icon from the main UI.
2. The system displays a pop-up window with event information and a source link to the event source site

Activation

This use case starts when the user clicks the “Source” button within the event information pop-up window.

Main flow

1. The system redirects the user to the ticket purchase page on the external event source website.
2. The user can navigate the external website to view event details and purchase tickets for the event.

Alternate flow

A1: The does not buy tickets.
1. The user clicks/taps the “Source” button within an event information pop-up window.
2. The system redirects the user to the ticket purchase page on the external event source website.
3. The user can navigate the external website to view event details and decide not to purchase tickets.

Exceptional flow

E1: Error redirecting to event source site.
1. The system notifies the user and prompts them to retry fetching the data.
2. The user clicks/taps the retry button.
3. The use case continues at position 2 of the main flow.

E2: Error redirecting to event source site persists.
1. The system notifies the user that the external website is currently unavailable.
2. The system defaults to the event info pop-up window.

E3: Error retrieving external website address with event info.
1. The system does not display the “Source” button.

Termination

The user exits the "Buy Tickets for Events" use case by closing the external website window and/or returning to the WhatsOn app.

Postcondition

The system defaults to the event pop-up window, displaying the event information. If the user clicks/taps the "Source" button, the system returns to the event pop-up window after closing the external website browser window.

Result

We achieved all the steps of the Main/Alternative Flow except but did not include Exceptional flow by notifying the user.

2.1.1.8. Requirement 8: Get Directions to Events

2.1.1.8.1. Description & Priority

WhatsOn does not feature native route planning. Linking to a reliable and accurate route planning site such as Google Maps enhances convenience and facilitates event attendance.

Priority: Medium.

2.1.1.8.2. Use Case

Scope

The scope of this use case is for a user to purchase tickets for events displayed on the WhatsOn app.

Description

This use case describes a user viewing an event within the WhatsOn app and being redirected to an event source site a reliable mapping service (Google, Apple Maps) for accurate route planning in different modes of transport such as Walking, Driving, Cycling, Public Transport, etc.

Flow Description

Precondition

1. The user clicks/taps an event icon from the main UI.
2. The system displays a pop-up window with event information and a source link to an external mapping service site.
3. External Geolocation API tracks the event accurately.

Activation

This use case starts when the user clicks the "Directions" button within the event information pop-up window.

Main flow

1. The system retrieves the coordinates of the event venue from the event data.
2. The system launches the selected mapping service (Google Maps, Apple Maps) on the user's device.

3. The system displays the route and directions from the user's current location to the event venue, allowing the user to choose different modes of transport (Walking, Driving, Cycling, Public Transport, etc).

Alternate flow

A1: No directions available for the event.
1.0 The system does not display the “Directions” button.

Exceptional flow

E1: Error redirecting to mapping service.
1. The system notifies the user and prompts them to retry fetching the data.
2. The user clicks/taps the retry button.
3. The use case continues at position 2 of the main flow.

E2: Error redirecting to mapping service persists.
1. The system notifies the user that the external website is currently unavailable.
2. The system defaults to the event info pop-up window.

E3: Error retrieving event venue coordinates.
4. The system does not display the “Directions” button.

Termination

The user exits the "Get Directions to Events" use case by closing the maps application or returning to the WhatsOn app.

Postcondition

The system defaults to the event pop-up window, displaying the event information. If the user clicks/taps the "Directions" button, the system returns to the event pop-up window after closing the maps application.

Result

We again achieved all the steps of the Main/Alternative Flow except but did not include Exceptional flow by notifying the user.

Although we missed some error handling in these use cases, it was often in the code just not displaying to the user. We also added a 404 error page too.

2.1.2. Data Requirements

1. The system shall retrieve event data from external APIs.
   The system shall retrieve event coordinates
2. The system shall store user accounts
3. The system shall store user preferences for events
4. The system shall store user followed events
5. The system shall retrieve event source links
6. The system shall handle user authentication and authorization for contacting WhatsOn/user accounts

2.1.3. User Requirements

1. The system shall provide clear instructions for enabling location services and granting permissions to the app
2. The system shall offer intuitive user interfaces for browsing events, filtering by type, and following events
3. The system shall include error messages and prompts to guide users in case of failed data retrieval or authentication.
4. The system shall provide a seamless experience for users to redirection to external event sources
5. The system shall allow users to access personalized events with a simple account creation processes
6. The system shall encrypt sensitive user information and adherence to data protection regulations
7.

2.1.4. Environmental Requirements

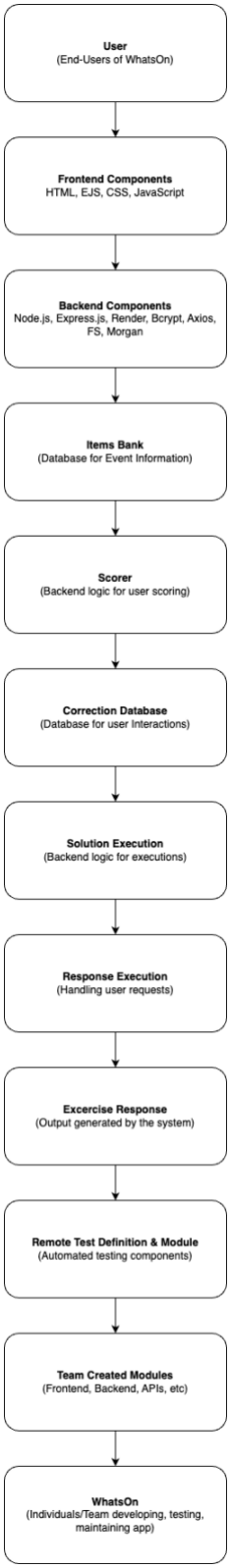
1. The system shall be compatible with mobile devices running iOS and Android operating systems
2. The system shall be compatible with desktop devices running Windows, MacOs and Linux operating systems
3. The system shall require an internet connection for retrieving event data, mapping services, and external event sources
4. The system shall require geolocation services for retrieving user location coordinates
5. The system shall function effectively in areas with different network speeds
6. The system shall adapt to different screen sizes and resolutions
7. The system shall update event data in response changes in location/orientation

2.1.5. Usability Requirements

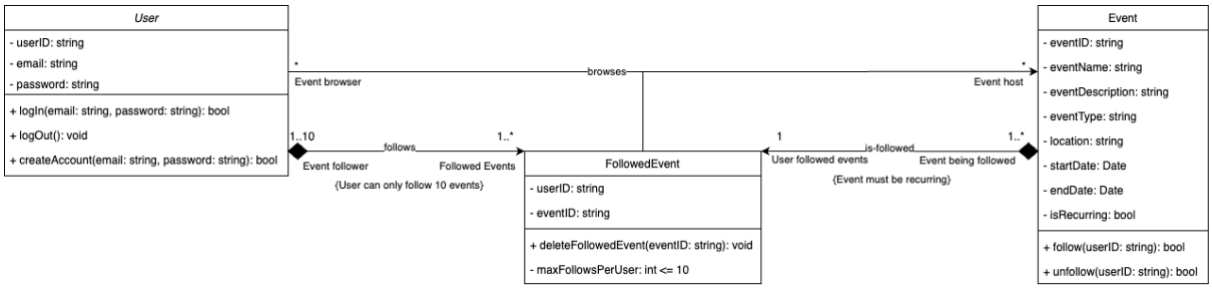
1. The system shall feature intuitive user interfaces and navigation
2. The system shall feature a consistent design for a cohesive and simple user experience
3. The system shall support gestures such as swiping, tapping, and pinching to zoom
4. The system shall minimize cognitive load by presenting information in a structured and easily digestible format
5. The system shall prioritize speed and responsiveness, ensuring quick loading times for event data, maps, and ticketing pages.

2.2. Design & Architecture

2.2.1 System Architecture Diagram



2.2.2 Class Diagram for Use Case Template 2: Follow Events



2.2.2.1 Description

2.2.2.2 Names from the problem domain

Class names **User**, **Event** and **FollowedEvent** reflect entities within the problem domain as described in **Use Case Template 2: Follow Events.**

2.2.2.3 Attributes

User: userID, email, password
Event: eventID, eventName, eventDescription, eventType, location, startDate, endDate, isRecurring
FollowedEvent: userID, eventID

2.2.2.4 Methods

User: logIn(email: string, password: string), logOut(), createAccount(email: string, password: string)
Event: follow(userID: string), unfollow(userID: string)
FollowedEvent: deleteFollowedEvent(eventID: string), maxFollowsPerUser: int <= 10

2.2.2.5 Relationships

User browses Event: A user can browse multiple events and decide to follow or not, each event can be browsed/followed by multiple users.

User to FollowedEvent: A user can follow up to 10 events, and each followed event is associated with at least one user.

Event to FollowedEvent: events, when followed by users, become instances of FollowedEvent.

2.2.2.6 Roles

User: Represents the entity browsing and following events.

Event: Represents the entity being browsed and followed.

FollowedEvent: Represents the association between a user and an event.

2.2.2.7 Navigability

User to Event: a user can browse multiple events as shown by the association line pointing from User to Event.

User to FollowedEvent: a user can access and delete their followed events as indicated by the association line pointing from User to Event.

Event to FollowedEvent: events, when followed by users, become instances of FollowedEvent. This is depicted by the association line pointing from Event to FollowedEvent.

2.2.2.8 Constraints

User to FollowedEvent: a user can only follow up to 10 events, as indicated by the constraint “User can only follow 10 events” under the association line from User to FollowedEvent. Another constraint is that each followed event must be associated with at least 1 user. See Multiplicity section below.

Event to FollowedEvent: events must be recurring to be followed by users, as indicated by the constraint "Event must be recurring" under the association line from Event to FollowedEvent. Another constraint is that each followed event must only be associated with 1 event. See Multiplicity section below.

2.2.2.9 Association Class

The FollowedEvent class serves as an association class, connecting User and Event. It includes additional methods to delete events and limit the amount of events that can be followed.

2.2.2.10 Multiplicity

User to Event: “*” represents a many-to-many relationship, where each user can browse multiple events, and each event can be browsed by multiple users.

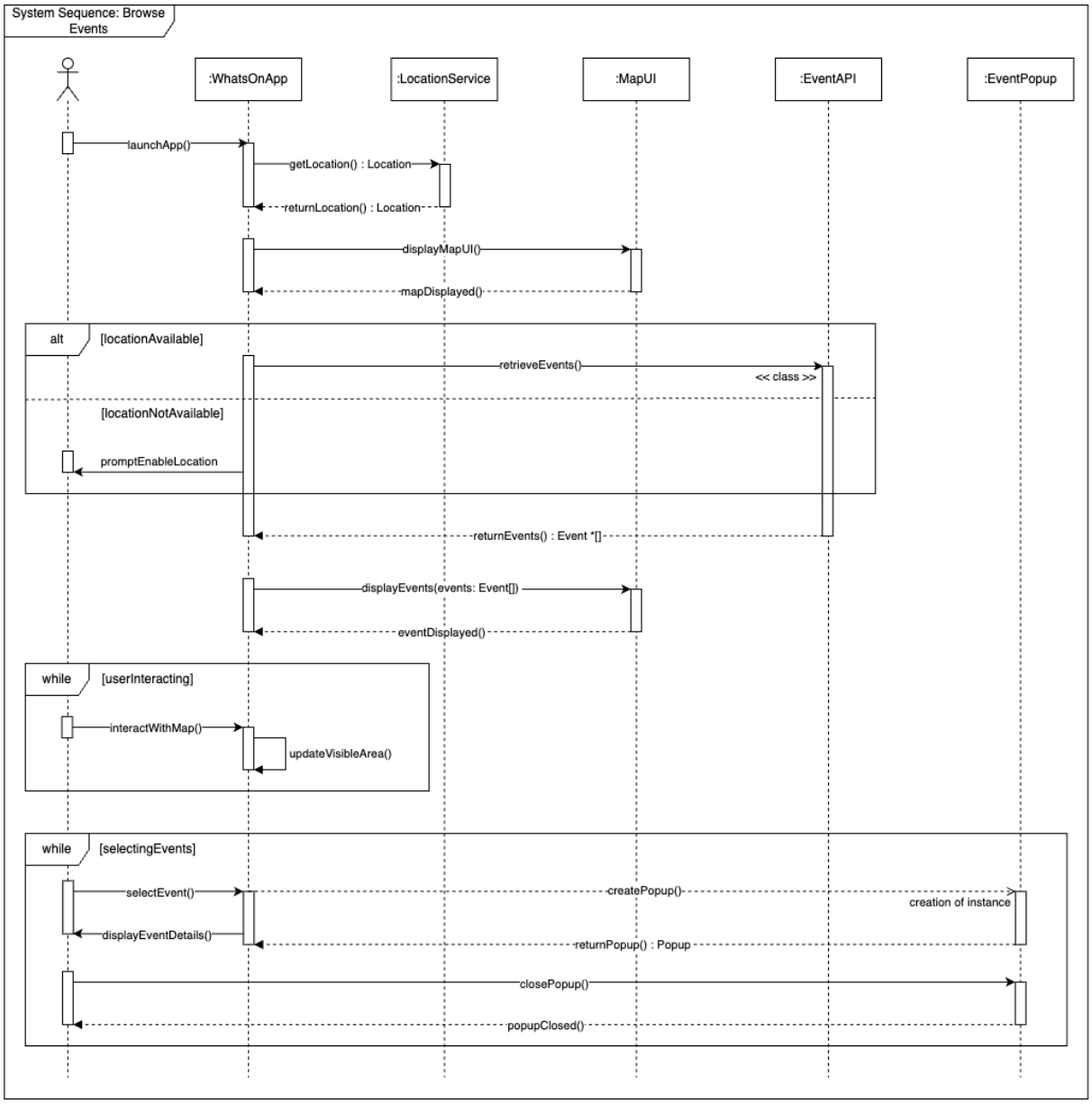
User to FollowedEvent: “1..10” indicates that each user can follow up to 10 events. “1..*” indicates that each followed event must be associated with at least one user.

Event to FollowedEvent: “1..*” indicates that each event can be followed by multiple users. "1" indicates that each followed event is associated with exactly one event.

2.2.2.11 Composition

FollowedEvent cannot exist without both the User and Event Class. As such it is a composition relationship as noted by a closed diamond with both other classes.

2.2.3 Class Diagram for Use Case Template 1: Browse Events



2.2.4 Contract for selectEvent()

Name: selectEvent()

Responsibilities:

• Handle the user's selection of a specific event on the map.

• Create an instance of EventPopup associated with the selected event.

• Communicate with other components (e.g., EventPopup, WhatsOnApp) to manage event details.

Type: System

Pre-conditions:

• The map UI is displayed and ready for interaction.

• The user has selected an event (clicked on an event marker).

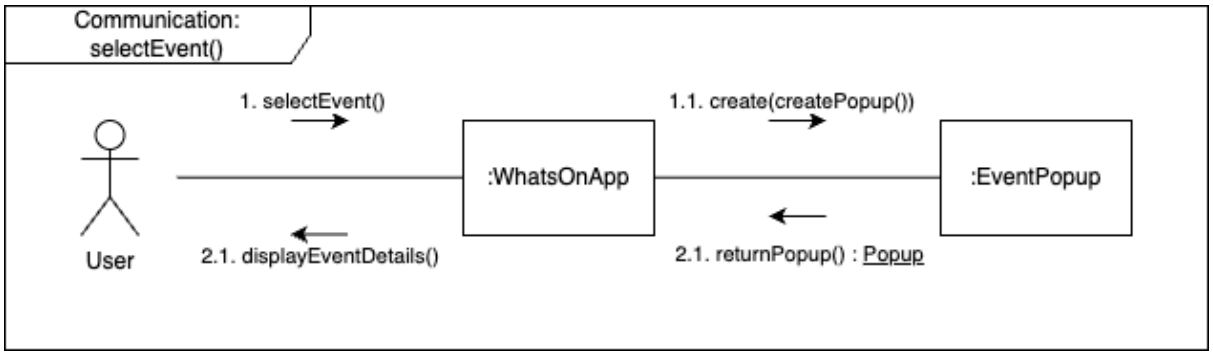
Post-conditions:

• If the user clicks on an event, a new EventPopup is created (instance creation).

• If the user clicks on an event, the new EventPopup is associated with the selected event (association formed).

• If the user clicks on an event, event details are displayed in the new EventPopup.

2.2.4.1Communication Diagram for selectEvent()



1.2.1.1.1 User

• Initiates the event selection by interacting with the map UI.

• Sends a message to the WhatsOnApp.

1.2.1.1.2 WhatsOnApp

• Coordinates the event selection process.

• Manages the flow of messages.

• Communicates with other components.

1.2.1.1.3 EventPopup

• Represents a popup displaying event details.

• Created dynamically when the user selects an event.

• Communicates with the WhatsOnApp to provide event information.

1.2.1.1.1 Design Patterns

1.2.1.1.1.1 Expert

• The WhatsOnApp acts as the expert by handling event selection and creating the EventPopup.

• The WhatsOnApp has the necessary knowledge to manage event-related actions.

1.2.1.1.1.2 Creator

• The WhatsOnApp creates an instance of EventPopup when the user selects an event.

• The WhatsOnApp is responsible for object creation.

1.2.1.1.1.3 Controller

• The WhatsOnApp serves as the controller, coordinating user interactions and managing the event selection process.

• It communicates with other components (EventPopup).

1.3 Implementation

The main algorithms, classes, and functions used:

Function for initializing and loading the map on the frontend using Google Maps API

Loading

```
function loadGoogleMaps(apiKey) {
  const script = document.createElement('script');
  script.src = `https://maps.googleapis.com/maps/api/js?key=${apiKey}&map_ids=13185b1ffbbba3af&callback=initMap`;
  script.async = true;
  document.body.appendChild(script);
}
```

Initialising

```
//  CJ- using callback function "initMap" from Google Map Script API URL
function initMap() {
  // <!-- CJ- targeting "map-div" to load map -->
  map = new google.maps.Map(document.getElementById("map-div"), {
    // CJ- loads map at specific lattitude and longitude
    center: { lat: 53.349076911151386, lng: -6.242441879918039 },
    // CJ- defines the zoom level
    zoom: 14,
    // CJ- mapId is the id of the map created in Google Cloud Platform
    mapId: "13185b1ffbbba3af",
```

Function for fetching event data from the backend server using Axios.

```
import axios from "axios";
import cheerio from "cheerio";
import fs from "fs";

const categories = {
  Charity: ["511", "604", "624", "673", "449"],
  Cultural: ["521", "622", "467"],
  Education: ["405", "593", "436", "546"],
  Entertainment: ["612", "535", "395"],
  Social: ["571", "652", "701"],
  Sports: ["482", "612", "684"],
};      Eoin Fitzsimons, yesterday • Fetching events by type …

async function fetchData() {
  let eventsData = [];

  for (let eventType in categories) {
    for (let categoryId of categories[eventType]) {
      const main_url =
        "https://www.meetup.com/find/?sortField=DATETIME&source=EVENTS&eventType=inPerson&dateRange=tomorrow&location=ie--Dublin&categoryId=" +
        categoryId;

      let eventDetails = "";
```

Function for parsing and processing the retrieved data using Cheerio.

```
const eventPromises = eventLinks.map((eventLink) => {
  return axios
    .get(eventLink)
    .then((response) => {
      const $ = cheerio.load(response.data);
      const imgSrc = $(
        'picture[data-testid="event-description-image"] img'
      ).attr("src");
      const groupLink = $("a#event-group-link").attr("href");
      const scripts = $('script[type="application/ld+json"]');
```

Function for creating markers based on eventType and eventTime

```
//CJ- Dynamic JSON data for markers and info windows
//CJ- Function to get the icon URL based on the event type
function getIconUrl(eventType, eventTime) {
  //CJ- Create a new image object
  var icon = new Image();
  //CJ- Add an onload event listener to the image object
  icon.onload = function () {
    //CJ- The image has been loaded
  };
```

Storing event time category as a variable, splitting into 3 categories

```
// CJ- Create a variable to store the time category
var timeCategory;
// CJ- Get the current time in milliseconds
var currentTime = new Date().getTime();
// CJ- Get the event start time in milliseconds
var eventStartTime = new Date(eventTime).getTime();
// CJ- Calculate the time difference in hours
var timeDifference = (eventStartTime - currentTime) / (1000 * 60 * 60);

// CJ- Determine the time category based on the time difference
if (timeDifference < 8) {
  // CJ- Event is starting in less than 8 hours
  timeCategory = "starting_soonest";
} else if (timeDifference < 16) {
  // CJ- Event is starting in between 8 and 16 hours
  timeCategory = "starting_in_between";
} else {
  // CJ- Event is starting in more than 16 hours
  timeCategory = "starting_furthest_away";
}
```

Switch Statement to swap marker image depending on type and time

```
//CJ- switch statement to return the icon URL based on the event type
switch (eventType) {
  //CJ- case statement for each event type
  case "Charity":
    //CJ- switch statement to return the icon URL based on the time category
    switch (timeCategory) {
      case "starting_soonest":
        return "ChS.png";
      case "starting_in_between":
        return "ChO.png";
      case "starting_furthest_away":      Conor Judge, 13 hours ago • Event M
        return "ChE.png";
    }
```

Array to store events for random button and creations of event marker for each event in the array

```
//CJ- array to store events for random button
let events = [];

//CJ- Multiple Markers with Info Windows from JSON Data
fetch("/eventsData.json")
  //CJ- convert data to JSON
  .then((response) => response.json())
  //CJ- log data to console
  .then((data) => {
    //CJ- loop through data array
    data.forEach((eventData, index) => {
      //CJ- create new marker for event
      const marker = new google.maps.Marker({
        //CJ- set position of marker to event location
        position: { lat: eventData.lat, lng: eventData.lon },
        map,
        //CJ- set title of marker to event name
        title: eventData.eventName,
        icon: {
          url: getIconUrl(eventData.eventType, eventData.eventTime),
          scaledSize: new google.maps.Size(48, 48),
        },
        //CJ- set animation to drop from top of map
        animation: google.maps.Animation.DROP,
        //CJ- add eventType property to marker
        eventType: eventData.eventType,
        groupLink: eventData.groupLink,
      });
```

JSON Info Window content

```
// CJ- Create an info window for the event
const infowindow = new google.maps.InfoWindow({
  //CJ- div class to style info window content
  content: `
    <div class="info-window-content">
      <div class="text-container">
        <h3>${eventData.eventName}</h3>
        <div class="event-type-container">
          <p><strong>Type:</strong></p>
          <button class="toggle-btn" onclick="window.filterEvents('${
            eventData.eventType
          }')">${eventData.eventType}</button>

        </div>
        <p><strong>Group Name:</strong> ${eventData.groupName}</p>
        <p><strong>Start Time:</strong> ${new Date(
          eventData.eventTime
        ).toLocaleString()}</p>
        <p><strong>End Time:</strong> ${new Date(
          eventData.eventEndTime
        ).toLocaleString()}</p>
        <p><strong>Location:</strong> ${
          eventData.locationInfo
        }</p> <!-- Location info -->
        <p>${eventData.eventDetails.substring(
          0,
          100
        )} <!-- Displays the first 100 characters -->
            <span id="extra-text-${
              eventData.eventId
            }" class="extra-text">${eventData.eventDetails.substring(
    100
  )}</span>
            <button id="btn-extra-text-${
              eventData.eventId
            }" class="expand-collapse-btn" onclick="window.expandText('extra-text-${
    eventData.eventId
  }');">Read more</button>
            <button id="collapse-btn-extra-text-${
              eventData.eventId
            }" class="expand-collapse-btn" style="display: none;" onclick="window.collapseText('extra-text-${
    eventData.eventId
  }');">Collapse</button>
        </p>
      </div>
      <div class="image-source-container">
        <img src="${eventData.eventImage}" alt="Event Image">
        <div class="btn-container">
          <a href="https://www.google.com/maps/dir//${
            eventData.lat
          },${
    eventData.lon
  }" target="_blank" class="directions-btn">Directions</a>
          <a href="${
            eventData.eventUrl
          }" target="_blank" class="source-btn">Source</a>
          <a href="#" id="follow-btn-${
            eventData.eventId
          }" onclick="event.preventDefault(); follow(${eventData});" class="follow-btn" style="${
    isLoggedIn ? "display: inline;" : "display: none;"
  }">Follow</a> </div>
      </div>
  `,
});
```

Prior Build- 6 hardcoded markers and info windows were used before

```
// CJ- creates new constant named marker to attach to event listener to open info window
const marker1 = new google.maps.Marker({
  // CJ- sets the location of marker
  position: { lat: 53.34904445277732, lng: -6.2430142587188975 },
  map,
  //CJ- title of marker
  title: "Education: Starting",
  icon: {
    // CJ- location of icon image
    url: "EdS.png",
    // CJ- new Google maps object that defines the size of icon image
    scaledSize: new google.maps.Size(48, 48),
  },
  //CJ- animation that drops pin from top of the map
  animation: google.maps.Animation.DROP,
});

//CJ- new info window object set to open when marker is clicked
const infowindow1 = new google.maps.InfoWindow({
// CJ- div class to style info window content
content: `
    <div class="info-window-content">
    <h3>Education</h3>
    <p>Status: Starting</p>
    <p>Location: National College Of Ireland</p>
    <p>Mayor Street Lower</p>
    <p>Join us for an enlightening evening of knowledge and camaraderie.</p>
    <p>Let's expand our horizons and make lasting memories together!</p>
    </div>
`
});
```

Random function

```
//CJ- Adds a click event listener to the random button to display a random event info window
//CJ- SelectS the button inside the div with class 'randomTest'
// let btn = document.querySelector(".randomTest button");
let btn = document.querySelector("button.item.random");
//CJ- Add a click event listener to the button
btn.addEventListener("click", function () {
  // CJ- Select a random index from the events array to display an event
  let index = Math.floor(Math.random() * events.length);
  // CJ- Close the currently open info window if there is one
  if (currentInfoWindow) {
    currentInfoWindow.close();
  }
  //CJ- Open the info window for the random event
  events[index].infowindow.open(map, events[index].marker);
  //CJ- Set the current info window to the newly opened one
  currentInfoWindow = events[index].infowindow;
  //CJ- Center the map on the location of the random event
  map.setCenter(events[index].marker.getPosition());
});
//CJ- click event listener added to the map to close the info window when the map is clicked
map.addListener("click", function () {
  if (currentInfoWindow) {
    currentInfoWindow.close();
  }
});
```

Filter Function

```
//CJ- Function to filter the events based on the event type
window.filterEvents = function (eventType) {
  //CJ- If the current filter is the same as the clicked event type, remove the filter
  if (currentFilter === eventType) {
    //CJ- Set the current filter to null
    currentFilter = null;
    //CJ- Show all markers
    markers.forEach((marker) => {
      //CJ- Set the marker to be visible
      marker.setVisible(true);
    });
  } else {
    //CJ- If the current filter is not the same as the clicked event type, set the current filter to the clicked event type
    currentFilter = eventType;
    //CJ- Hide all markers that do not match the current filter
    markers.forEach((marker) => {
      //CJ- Set the marker to be visible if the event type matches the current filter
      marker.setVisible(marker.eventType === currentFilter);
    });
  }
};
```

Function for encrypting and decrypting sensitive data such as API keys using Crypto.

```
window.onload = function () {
  const encodedApiKey = window.prompt("Please enter your name:", "");
  // Now you can decode the API key
  fetch("http://localhost:3000/decrypt", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ encryptedApiKey: encodedApiKey }),
  })
    .then((response) => response.json())
    .then((data) => {
      const apiKey = data.decryptedApiKey;
      // Now you can use the decoded API key in your application
      loadGoogleMaps(apiKey);
    })
    .catch((error) => console.error("Error:", error));
};
```

Prior random testing

```
<script>
  function moveCrosshair() {
    // Get the crosshair div
    var crosshair = document.querySelector(".crosshair");

    // Get the size of .map-div
    var mapDiv = document.querySelector(".map-div");
    var mapDivWidth = mapDiv.offsetWidth;
    var mapDivHeight = mapDiv.offsetHeight;

    // Get the height of the navbar
    var navbarHeight = document.querySelector(".nav-bar").offsetHeight;

    // Generate random coordinates within .map-div, below the navbar
    var randomX = Math.floor(Math.random() * mapDivWidth);
    var randomY = Math.floor(
      Math.random() * (mapDivHeight - navbarHeight) + navbarHeight
    );

    // Position the crosshair at the random coordinates
    crosshair.style.left = randomX + "px";
    crosshair.style.top = randomY + "px";

    // Make the crosshair visible
    crosshair.style.visibility = "visible";
  }

  function closeCrosshair() {
    var crosshair = document.querySelector(".crosshair");
    crosshair.style.visibility = "hidden";
  }
</script>
```

This imports the data taker, the data analyser and the way to put the data into another file to make it readable.

```
import axios from "axios";
import cheerio from "cheerio";
import fs from "fs";
```

This is how the categories from meetup are sorted into our own categories, then the data is fetched from the url for every category.

```
const categories = {
  Charity: ["511", "604", "624", "673", "449"],
  Cultural: ["521", "622", "467"],
  Education: ["405", "593", "436", "546"],
  Entertainment: ["612", "535", "395"],
  Social: ["571", "652", "701"],
  Sports: ["482", "612", "684"],
};

async function fetchData() {
  let eventsData = [];

  for (let eventType in categories) {
    for (let categoryId of categories[eventType]) {
      const main_url =

"https://www.meetup.com/find/?sortField=DATETIME&source=EVENTS&eventType=inPer
son&dateRange=tomorrow&location=ie--Dublin&categoryId=" +
        categoryId;
```

This is how the details are taken and sorted, much of the data is in the second script of the page, there are two event links in the main page, one on a photo and one not, we use the one in the photo. Then we take the event details and take any difficult new line code segments and replace them with just a space.

```
    let eventDetails = "";

    await axios.get(main_url).then((response) => {
      const $ = cheerio.load(response.data);
      const eventLinks = $('a[data-event-label="Event card"]:has(img)')
        .map((i, el) => $(el).attr("href"))
        .get();

      const eventPromises = eventLinks.map((eventLink) => {
        return axios
          .get(eventLink)
          .then((response) => {
            const $ = cheerio.load(response.data);
            const imgSrc = $(
              'picture[data-testid="event-description-image"] img'
            ).attr("src");
            const groupLink = $("a#event-group-link").attr("href");
            const scripts = $('script[type="application/ld+json"]');

            if (scripts.length >= 2) {
              const script = $(scripts[1]).html();
              const data = JSON.parse(script);
              const eventDetailsSections = $("div.break-words p");
              eventDetails = eventDetailsSections
                .map((i, el) => $(el).text())
                .get()
                .join(" ")
                .replace(/\n/g, " ");

              eventsData.push({
                groupName: data.organizer
                  ? data.organizer.name
                  : "The organizer property does not exist.",
                eventTime: data.startDate
                  ? data.startDate
                  : "The startDate property does not exist.",
                eventDetails: eventDetails
                  ? eventDetails
                  : "The event details do not exist.",
                lat:
                  data.location && data.location.geo
                    ? data.location.geo.latitude
```

```
      : "The latitude property does not exist.",
  lon:
    data.location && data.location.geo
      ? data.location.geo.longitude
      : "The longitude property does not exist.",
  locationInfo:
    data.location && data.location.address
      ? data.location.address.streetAddress
      : "The streetAddress property does not exist.",
  eventName: data.name
    ? data.name
    : "The name property does not exist.",
  eventEndTime: data.endDate
    ? data.endDate
    : "The endDate property does not exist.",
  eventImage: imgSrc ? imgSrc : "The img src does not exist.",
  eventType: eventType,
  eventUrl: eventLink,
  groupLink: groupLink
    ? groupLink
    : "The group link does not exist.",
```

This is decoding the encrypted key, it's calling the backend where the secret key is kept, meaning the same keys will always get the actual API key, then the load googlemaps url is called with the real apli key banged in there.

```
window.onload = function () {
      const encodedApiKey = window.prompt("Hey Julie. Please enter the Encoded API Key. It's in the 4th page of the Technical Report. Above the Executive Summary:", "");
      // Now you can decode the API key
      fetch("/decrypt", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ encryptedApiKey: encodedApiKey }),
      })
        .then((response) => response.json())
        .then((data) => {
          const apiKey = data.decryptedApiKey;
          // Now you can use the decoded API key in your application
          loadGoogleMaps(apiKey);
        })
        .catch((error) => console.error("Error:", error));
    };

    function loadGoogleMaps(apiKey) {
      const script = document.createElement("script");
      script.src = `https://maps.googleapis.com/maps/api/js?key=${apiKey}&map_ids=13185b1ffbbba3af&callback=initMap`;
      script.async = true;
      document.body.appendChild(script);
    }
```

Contact by email

```
function messageSent() {
    var email = document.forms["contactForm"]["email"].value;
    var message = document.forms["contactForm"]["message"].value;
```

```
    if (email == "") {
      alert("You need to fill the email field!");
      return false;
    } else if (message == "") {
      alert("You need to fill the message field!");
      return false;
    } else {
```

This part of the method reads the value that the user has entered the input fields (email and message) and notifies the user if they have left the left on of the following fields empty and that they need to fill the missing fields in order to send their message.

```
      // Create a new FormData object from the form
      var formData = new FormData(document.getElementById("contactForm"));

      // Use fetch to send the form data to Formspree
      fetch("https://formspree.io/f/mayrlpzk", {
        method: "POST",
        body: formData,
        headers: {
          'Accept': 'application/json'
        },
      })
```

This stops a redirect to formspree's confirmation page and allows us to use our own one for seamless transition and pageloading purposing.

```
        .then((response) => response.json())
        .then((responseJson) => {
          if (responseJson.ok) {
            alert("Your message has successfully been sent to our contact
services!");
            document.getElementById("email").value = "";
            document.getElementById("message").value = "";
          } else {
            alert("There was an error sending your message. Please try
again.");
          }
        })
        .catch((error) => {
          alert("There was an error sending your message. Please try again.");
        });
      // Prevent the form from being submitted normally
      return false;
    }
  }
</script>
```

When a user logs into the following, it will open logged in user modal, if the user fails to log in it will keep the signup/login modal open

```
var isSignupMode = true;
var isLoggedIn = false;
var users = {};
var userEmail = null;
function followingButtonClicked() {
  if (isLoggedIn) {
    // User is logged in, show following
    toggleModal("followingModal", event);
  } else {
    // User is not logged in, show the  signup/login modal
    toggleModal("userModal", event);
  }
}
```

In this method it uses the switch button, it is automatically set to swich to login and first gives the user the choice to sign up, when the user already has a account they will press the switch button to change the form from sign up to login.

```
function switchMode() {
  var form = document.getElementById("authForm");
  var submitButton = document.getElementById("submit");
  var switchButton = document.getElementById("switchMode");

  if (isSignupMode) {
    form.setAttribute("action", "/users/login");
    submitButton.setAttribute("value", "Login");
    switchButton.textContent = "Switch to Sign Up";
  } else {
    form.setAttribute("action", "/users/register");
    submitButton.setAttribute("value", "Sign Up");
    switchButton.textContent = "Switch to Login";
  }

  isSignupMode = !isSignupMode;
}
```

This part of the method will let the users login into their account. It takes the values entered in the input and send it to the backend to check the db to see if the account exists. When the login is successful it will then close all the modals and load a alert notifying the user they have successfully logged in

```
//will work along side with backend code to check if the users log in details are correct and will take the message from the backend and display it to the user onb
  function handleFormSubmit(event) {
    event.preventDefault(); // Prevent the form from submitting normally
    var email = document.getElementById("email").value;
    var password = document.getElementById("password").value;

    if (!isSignupMode) {
      var url = "/users/login";

      fetch(url, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email: email, password: password }),
      })
        .then((response) => response.json())
        .then((data) => {
          if (data.success) {
            alert("Login successful!");
            userEmail = email;
            isLoggedIn = true;
            closeAllModals();
          } else {
            alert(data.message);
          }
        })
        .catch((error) => {
          console.error("Error:", error);
        });
    } else {
```

This follows a similar process to the last method. Takes the values of the input, sends to the back end but instead of reading to the db it will write into the database. Then it will let the customer know they have successfully created an account it will close all the modals that are open.

```
    var url = "/users/register";

    fetch(url, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ email: email, password: password }),
    })
      .then((response) => response.json())
      .then((data) => {
        if (data.success) {
          alert("Account created successfully!");
          isLoggedIn = true;
          closeAllModals();
        } else {
          alert(data.message);
        }
      })
      .catch((error) => {
        console.error("Error:", error);
      });
  }
}
```

Backend Imports

Express – handles all the routing code

Morgan – gives the back feed back to the site in regard to status codes and CRUD

Fetch – is pulling from the other js file that is handling fetching from the meetup website

Fs – is a file writer

Bcrypt – handles encrypting the passwords that are received and it handles reading from the encrypted passwords

Body-parsers – brings the data from the front end to the back

Path – is to direct the file to the right path

FileURL – works the same as path, just handles the functionailty differently

Crypto – encrypts the api key and decrypts the encoded crypted api key

```
import express from "express";
import morgan from "morgan";
import fetchData from "./Data Fetching/fetchData.js";
import fs from "fs";
import bcrypt from "bcrypt";
import bodyParser from "body-parser";
import path from "path";
import { fileURLToPath } from "url";
import crypto from "crypto";
```

Backend – Main Route

This is the main route for the project that loads all the other routes through the front end. This route loads the index page, it loads the fetchData() method whenever the web app is loaded. It will always reload the fetch data method when the page is refreshed

```
app.get("/", async (req, res) => {
  try {
    // Call fetchData() and wait for it to finish
    await fetchData();
    console.log("Data fetched successfully");
    // Rendering the 'index' template
    res.render("index", { title: "Home" });
  } catch (error) {
    console.error(`Error fetching data: ${error}`);
    res.status(500).send("Error fetching data");
  }
});
```

Backend - Register Route

In this method, we use the following imports – express, bcrypt and fs.

The express is handling the routing by using post.
The bcrypt is const salt, and hashpassword. The salt works as an extra layer of security of hex, it is an added set of text to ensure the each encrypted passwords are never the same, if all passwords were the same the encryption would be different for each one. Than the hashed password encrypts the password and then adds the salt within the hash password.

The filecontent is reading from the json file that is stores all of the accounts.

```
app.post("/users/register", async (req, res) => {
  try {
    const salt = await bcrypt.genSalt();
    const hashedPassword = await bcrypt.hash(req.body.password, salt);
    console.log(salt);
    console.log(hashedPassword);
    const newUser = { email: req.body.email, password: hashedPassword };

    let users = [];

    // Read users from the file
    try {
      const fileContent = fs.readFileSync(
        path.join(__dirname, "User Details", "users.json"),
        "utf8"
      );
      users = JSON.parse(fileContent);
      if (!Array.isArray(users)) {
        users = [];
      }
    } catch (error) {
      // If file does not exist, ignore the error
      if (error.code !== "ENOENT") {
        console.error(error);
        res.status(500).json({
          success: false,
          message: "An error occurred while reading the users file.",
        });
        return;
      }
    }
```

It checks to see if the email the customer is using if it has already been used. If the customer email is not used it will make account and write it to the db file(json). If the email is already in use, it will tell the customer that the email is already in use.

```
    // Check if the email is already in use
    const existingUser = users.find((user) => user.email === newUser.email);
    if (existingUser) {
      res
        .status(400)
        .json({ success: false, message: "Email already in use." });
      return;
    }
    // Add the new user
    users.push(newUser);

    // Write the updated users back to the file
    try {
      fs.writeFileSync(
        "./User Details/users.json",
        JSON.stringify(users),
        "utf8"
      );
    } catch (error) {
      console.error(error);
      res.status(500).json({
        success: false,
        message: "An error occurred while writing to the users file.",
      });
      return;
    }
  } catch (error) {
    res.status(500).json({
      success: false,
      message: "An error occurred during registration.",
    });
    console.log(error);
    return;
  }
  res.json({ success: true, message: "Account created successfully." });
});
```

Backend – Route for Login

This reads the db and sees if the account that the user has entered exists and if the account does exist is the password correct.

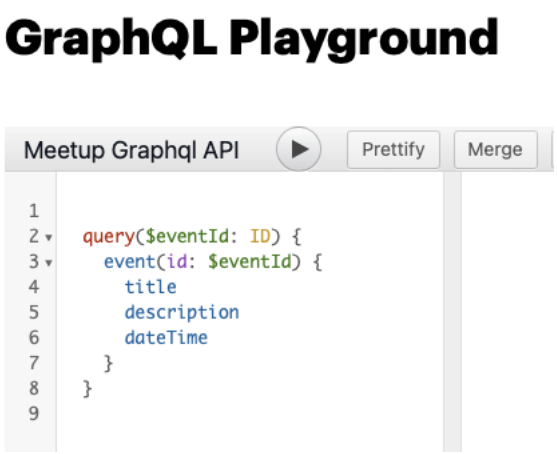
```
app.post("/users/login", async (req, res) => {
  let users;
  try {
    const fileContent = fs.readFileSync(
      path.join(__dirname, "User Details", "users.json"),
      "utf8"
    );
    users = JSON.parse(fileContent);
  } catch (error) {
    console.error(error);
    res.status(500).send("Cannot find users file");
    return;
  }
  const user = users.find((user) => user.email === req.body.email);

  if (user == null) {
    return res.status(400).send("Cannot find user");
  }
  try {
    if (await bcrypt.compare(req.body.password, user.password)) {
      res.json({ success: true });
    } else {
      res.json({ success: false, message: "Password incorrect" });
    }
  } catch {
    res
      .status(500)
      .send({ success: false, message: "An error occured during login" });
  }
});
```

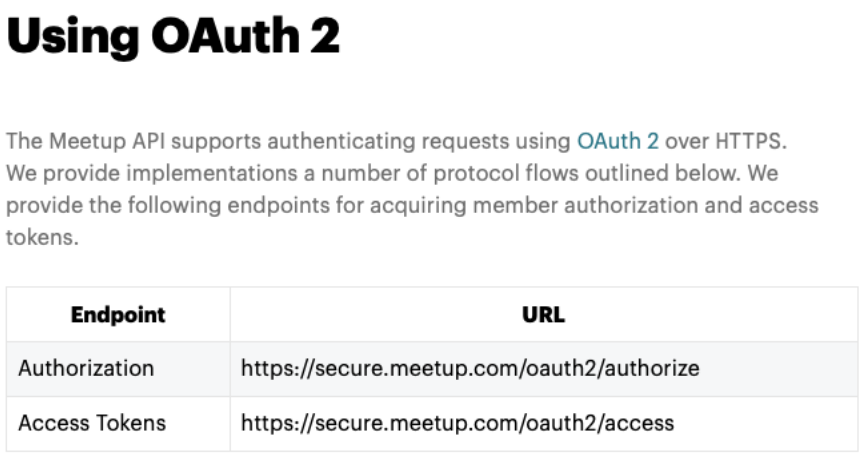
Meetup API

Until the final week of the project, we were Using the Meetup API. This presented many challenges. To get the API key you needed to have a free trial or paid account. You needed to deploy your website and then meetup would grant you the API key. This site needed to have specific functionality also and considerable time was committed to researching that. However, you also need a token to use the key that expires every 30 minutes. This was not clear from research and appeared to be a reasonably straightforward API to use.

Although seemingly straightforward to use at first with this GraphQL playground tool on the meetup site. Once implemented it was clear that it was not straightforward.



Expiring tokens



Authentication

Meetup uses access tokens to authenticate GraphQL requests. If you do not have a token or if your token is expired, please go to OAuth2 Server Flow to create or manage your tokens.

To authenticate the requests, set the **Authorization** HTTP header in your requests.

Header	Value
Authorization	Bearer {YOUR_TOKEN}

Most of the project time was spent trying to implement all the technologies required to use this API. This is evident in the git log as much of the implemented code had to be abandoned when fully realised less than a week before the project ended

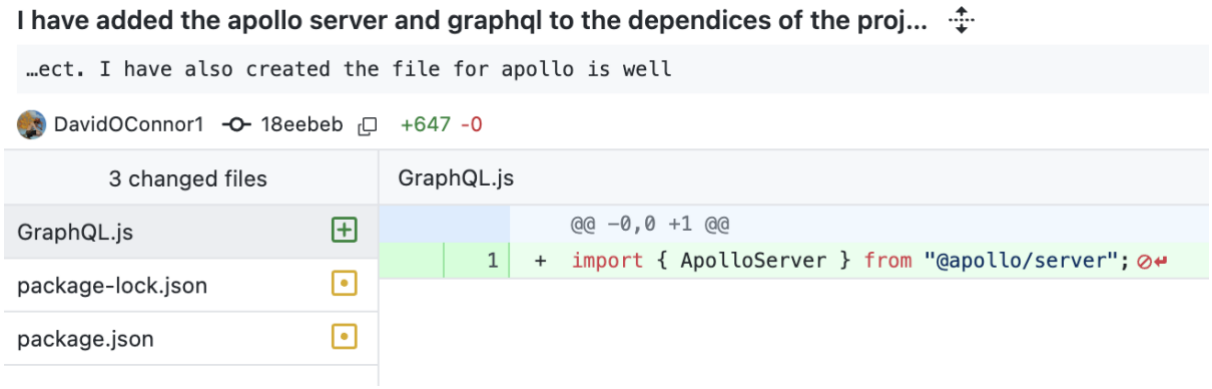
There were requirement to deploy the site on a specific kind of site that allowed live editing of web file. This was to be granted access to the meetup API key. We investigated several very complicated platforms to no avail but found a suitable fit on render

Some of the technologies previously used are below;

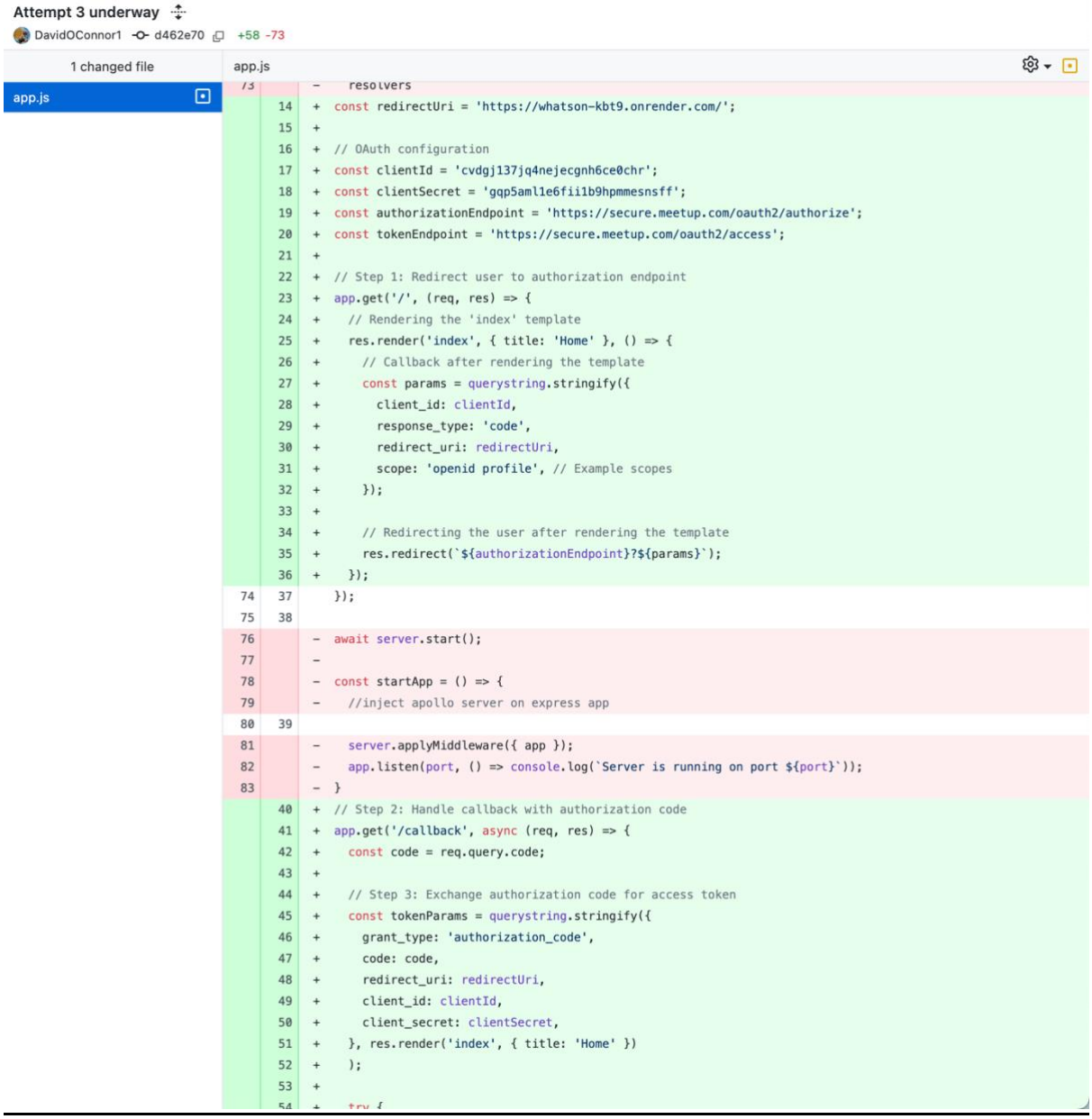
Mongoose

MongoDB

Apollo Server & GraphQL

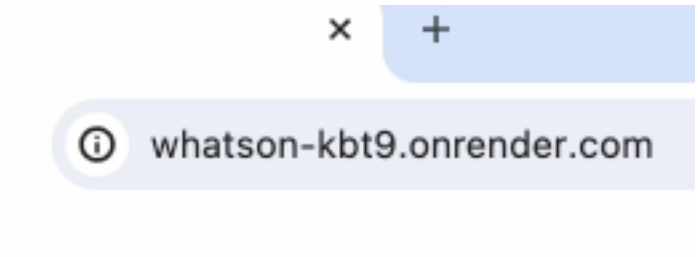


This is the final commit where it was clear we needed to pivot to a webcrawler instead of using the meetup API

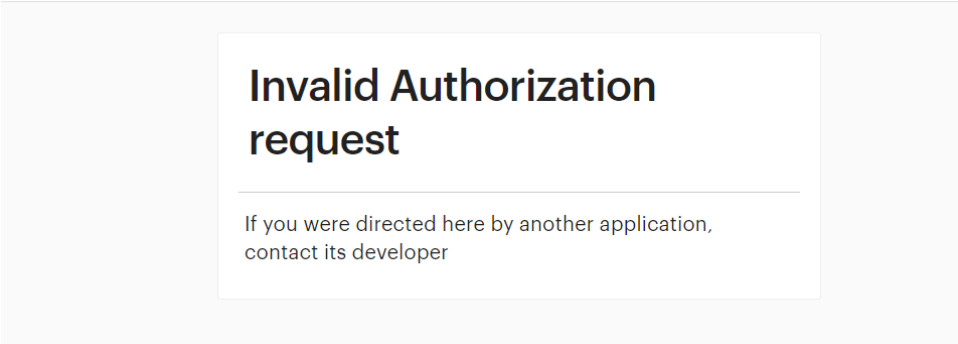


At this point the Website was redirecting to the Meetup website and requiring a user account and login to use it.

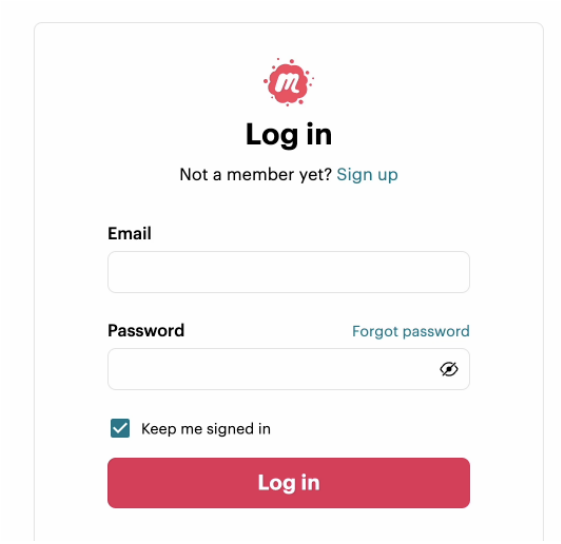
WhatsOns Deployed Site



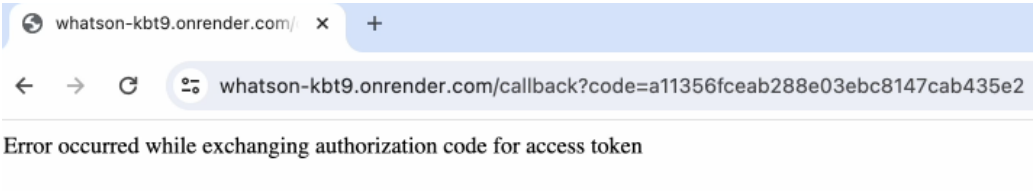
Redirect to Meetup



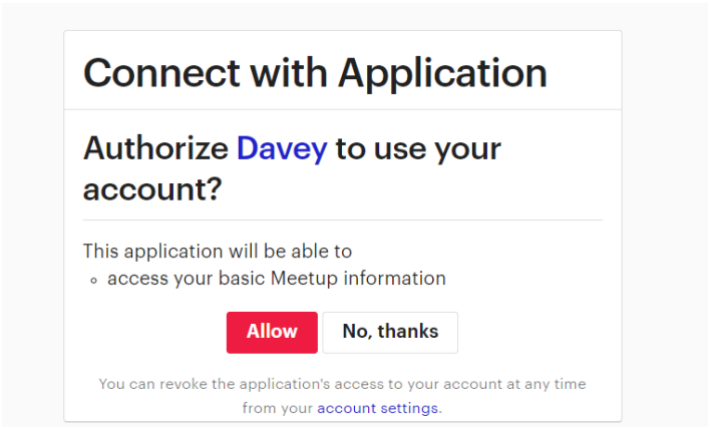
Login



Login Failed



Login "Success"

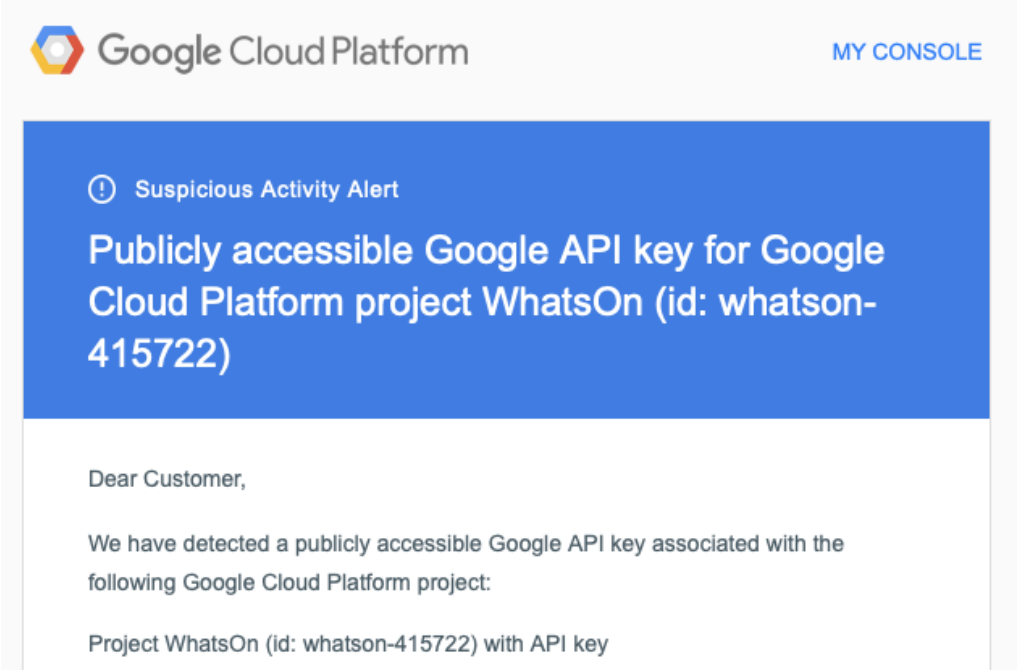


Despite investing 2 months into this approach we had to pivot to using Axios, Cherio and FS to act as a webcrawler to extract, parse and write http calls from this homepage.

https://www.meetup.com/find/?sortField=DATETIME&source=EVENTS&eventType=inPerson&dateRange=tomorrow&location=ie--Dublin&categoryId=

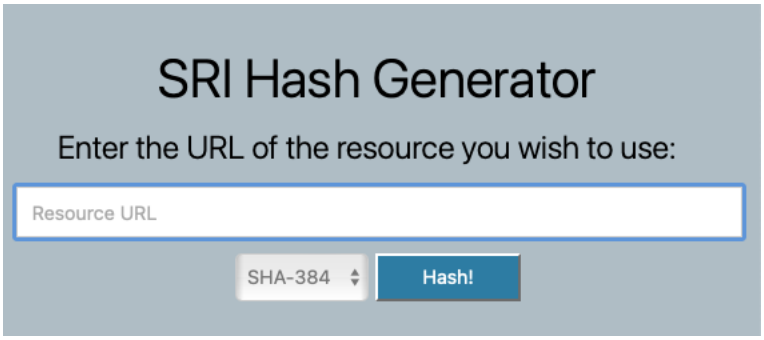
Google Maps API key

Using and setting up google maps was much more straightforward however we ran in to a similar problem with meetup as soon as it deployed.



This API key is linked to a Team Member bank card and is at great risk if it is in a public repository or deployed on a website. Several solutions were proposed to work around this. Using a .env file, storing as a local variable, encrypting it. At the time of writing this report we have not achieved a safe solution and we may be forced to commit an API key that is at risk of hacking a bank account.

Many Solutions were tried



.env file

```
You, 2 days ago | 1 author (You)
1   mapApiKey = "AIzaSyBq9c5JnECXrfMnYkVA2TSyStdrbZ3IQ-w"
```

```
.env  1, M
```

```
<!-- CJ- using env variable to store API key, must be implemented on render website-->
<script> const key = env.mapApiKey; console.log(key); </script>
```

Another Solution

```
<!-- CJ- Solution to hiding key -->
<script>
  const apiKey = process.env.GOOGLE_MAPS_API_KEY;
  const script = document.createElement('script');
  script.src = `https://maps.googleapis.com/maps/api/js?key=${apiKey}&map_ids=13185b1ffbbba3af&callback=initMap`;
  document.body.appendChild(script);
</script>
```

And another

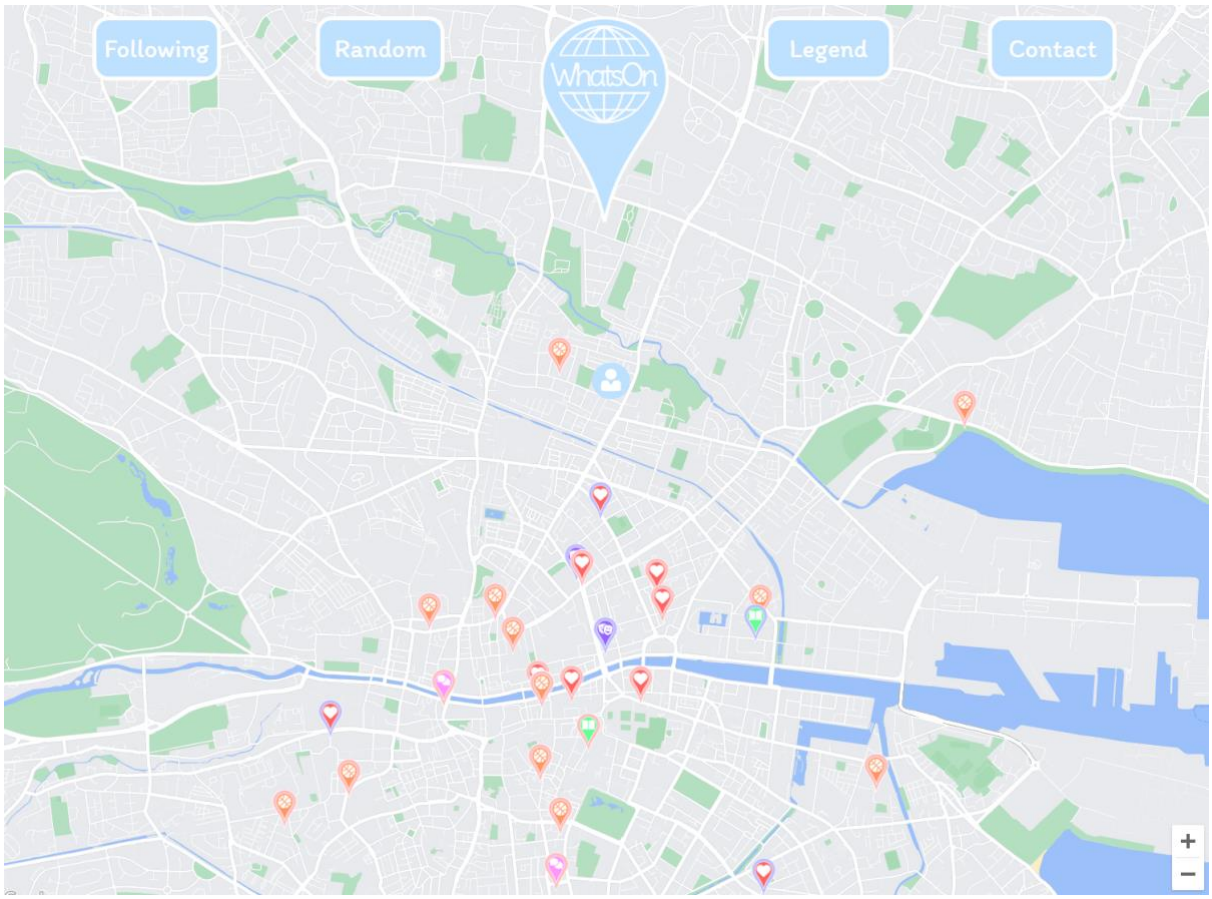
```
fetch('/api-key')
    .then(response => response.json())
    .then(data => {
      // data is the JSON object received from the server
      const apiKey = data.apiKey;
      // Now you can use apiKey in your client-side code
      const script = document.createElement('script');
        script.src = `https://maps.googleapis.com/maps/api/js?key=${AIzaSyClOptqTJtbgqusnHIRbB4RzaS0OsoC2ic}&map_ids=13185b1ffbbba3af&callback=initMap`;
        script.defer = true;
        script.async = true;
        document.head.appendChild(script);
    })
    .catch(error => console.error('Error:', error));
```

Error unacknowledged

```
<!-- CJ- loading map API & Map ID, after script because of callback function in URL that connects to initMap function -->
```

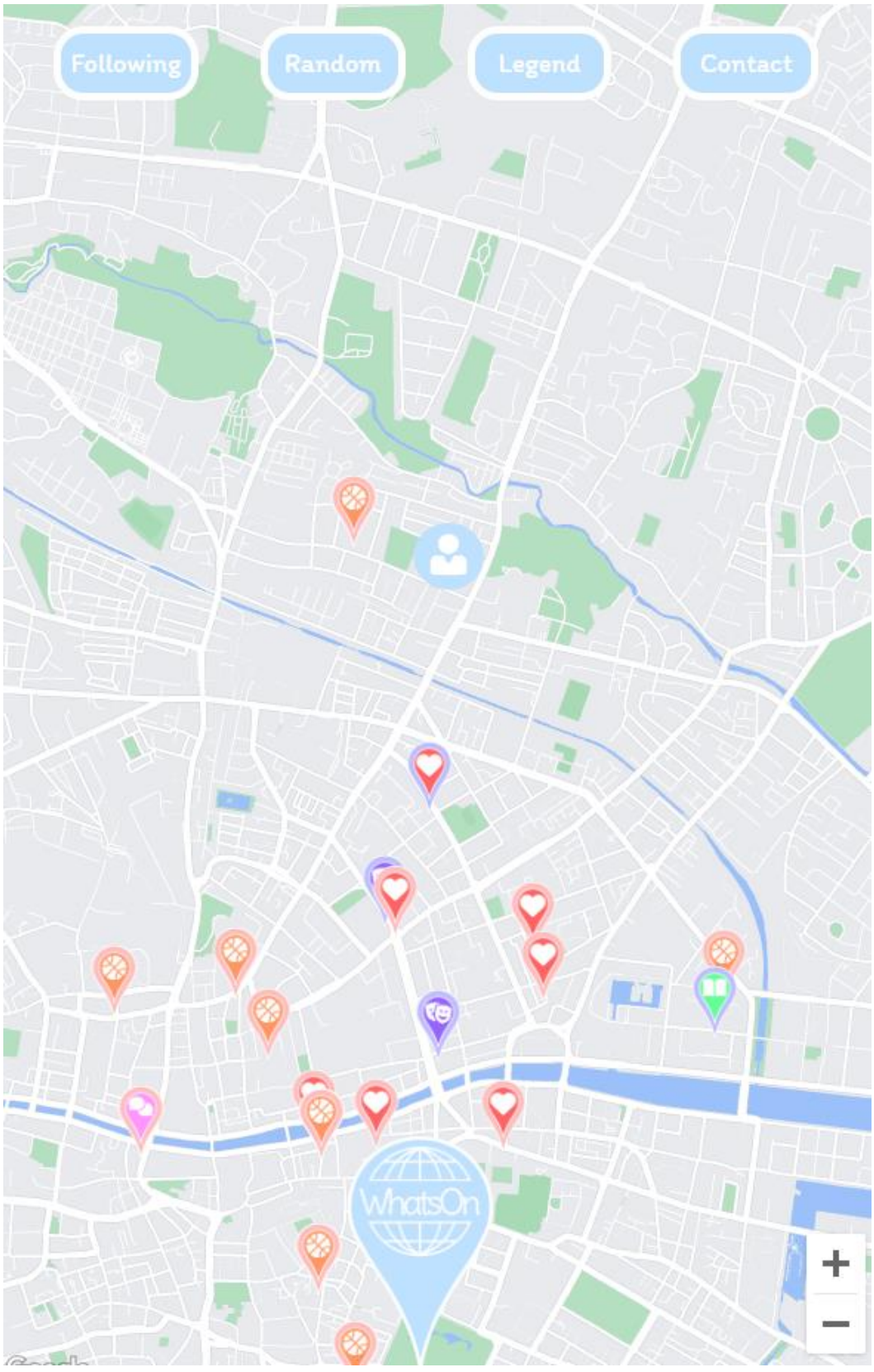
1.4 Graphical User Interface (GUI)

Desktop View



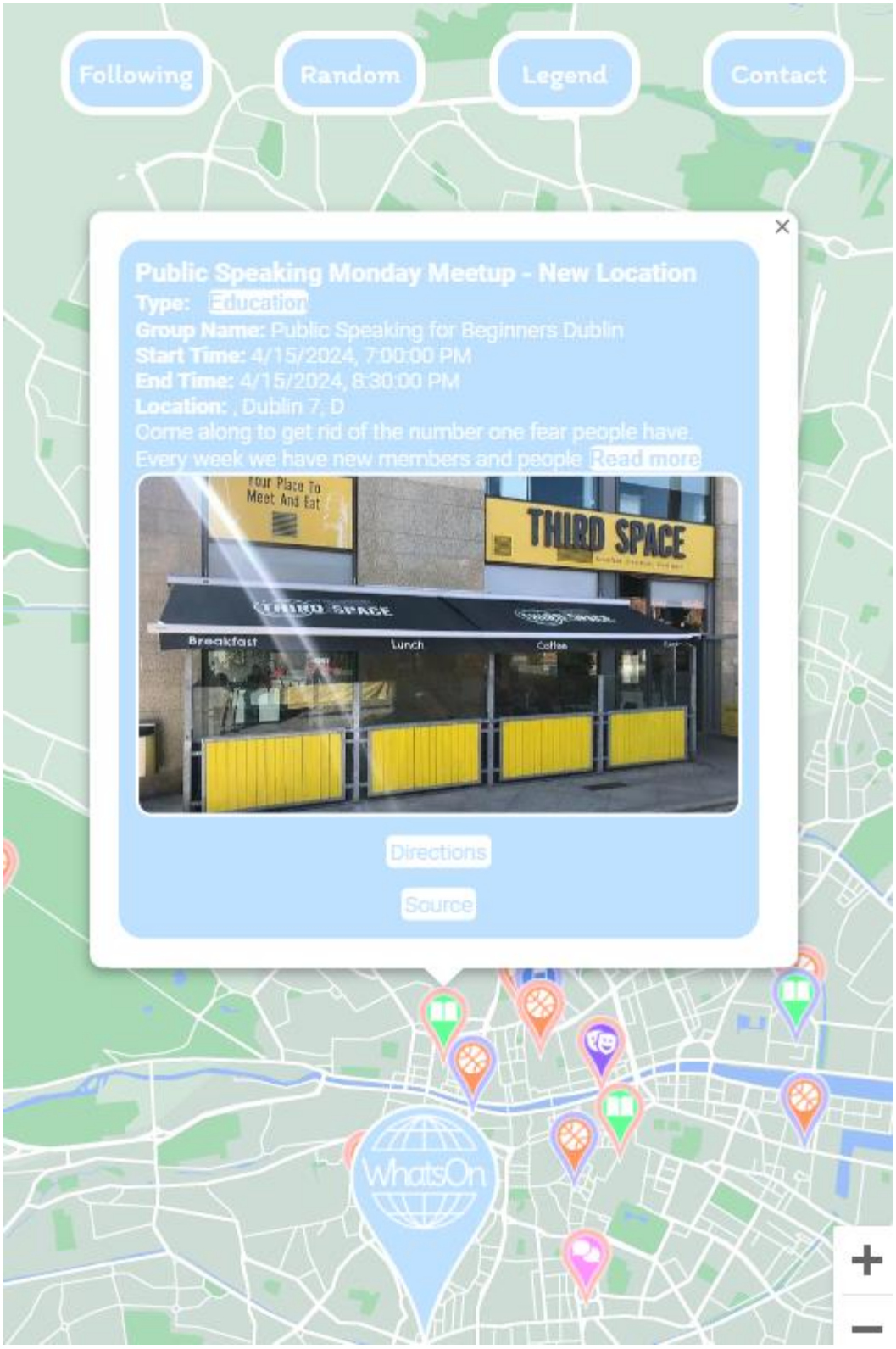
- User pin loads at users location upon app launch
- Pins dynamically load into the map
- Event pins differentiate by event type by icon
- Event pins differentiate by colour ring to indicate how soon they will be starting

Mobile View

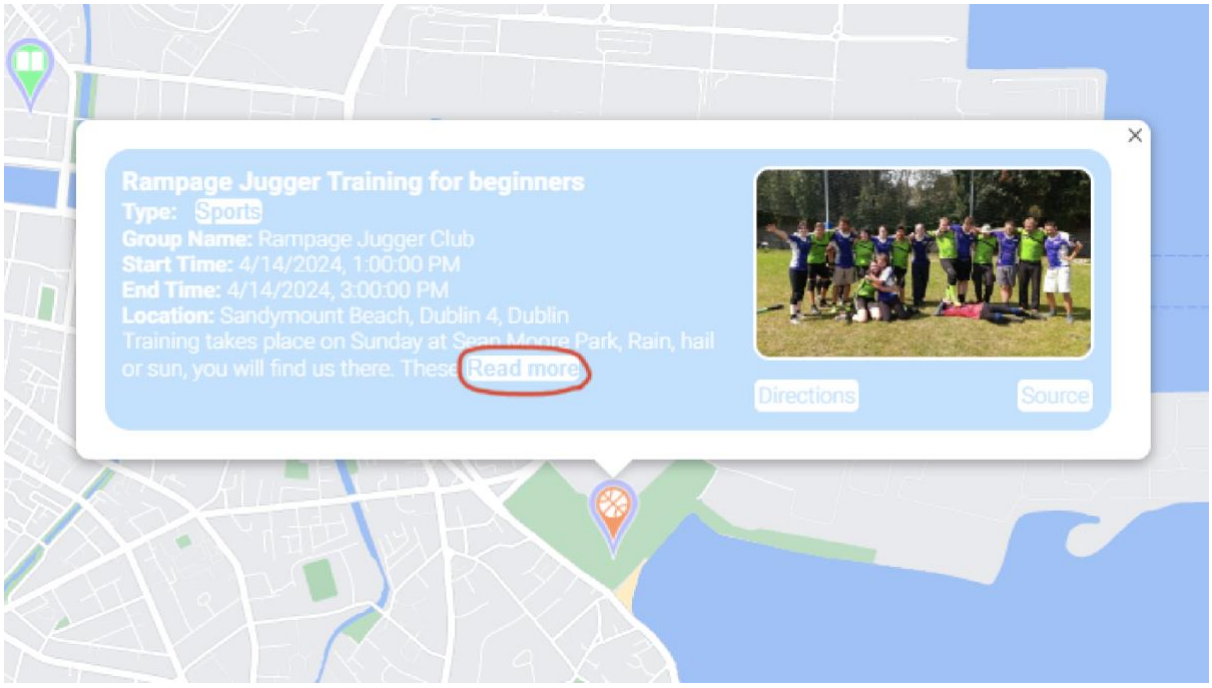


Responsive Design

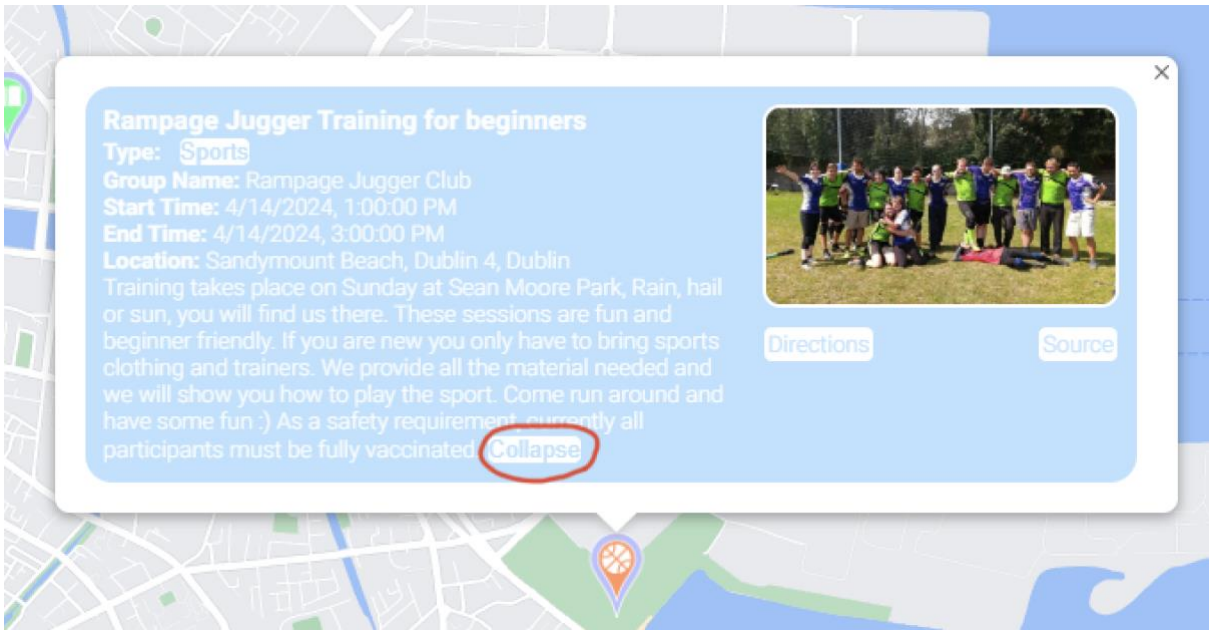
Info windows and other UI elements resize dynamically based on device screen size, aspect ratio



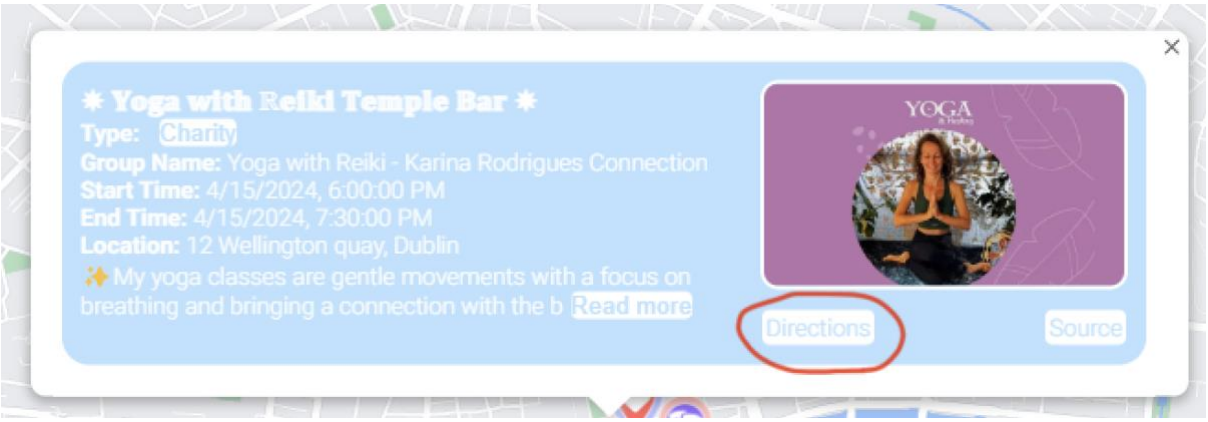
Info Window- Collapsed



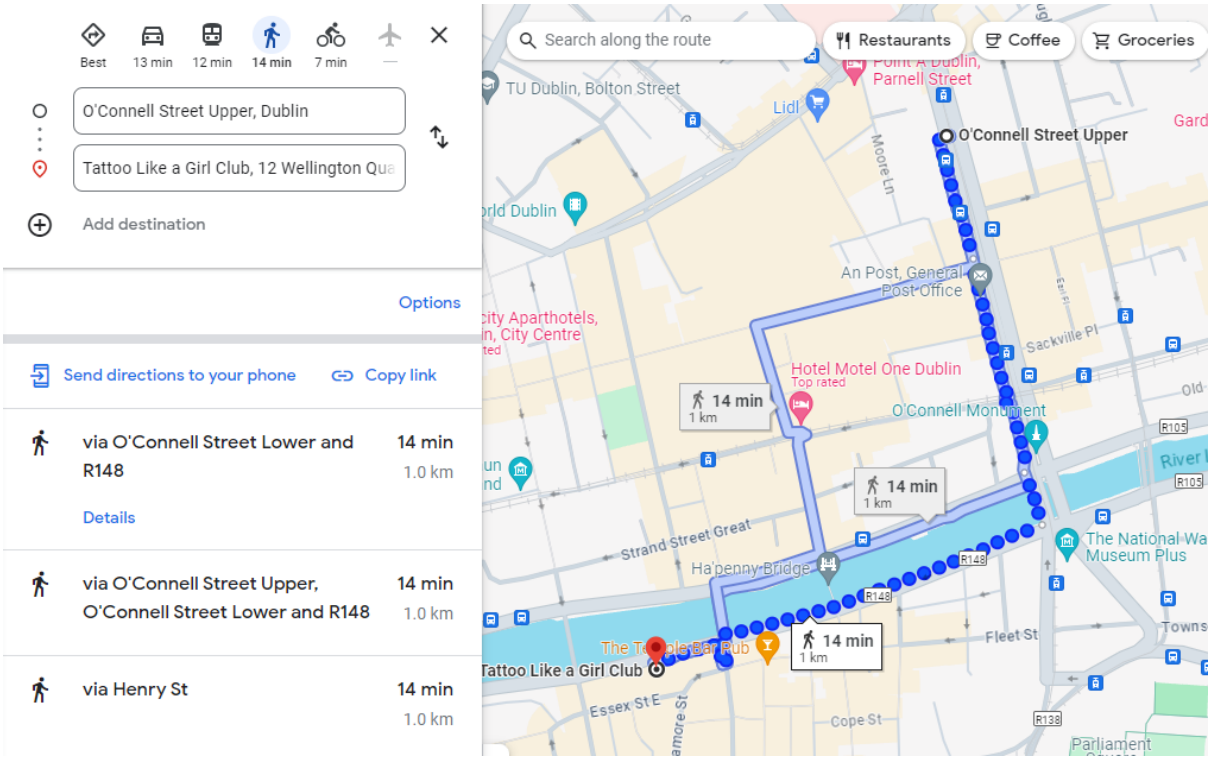
Info Window- Expanded



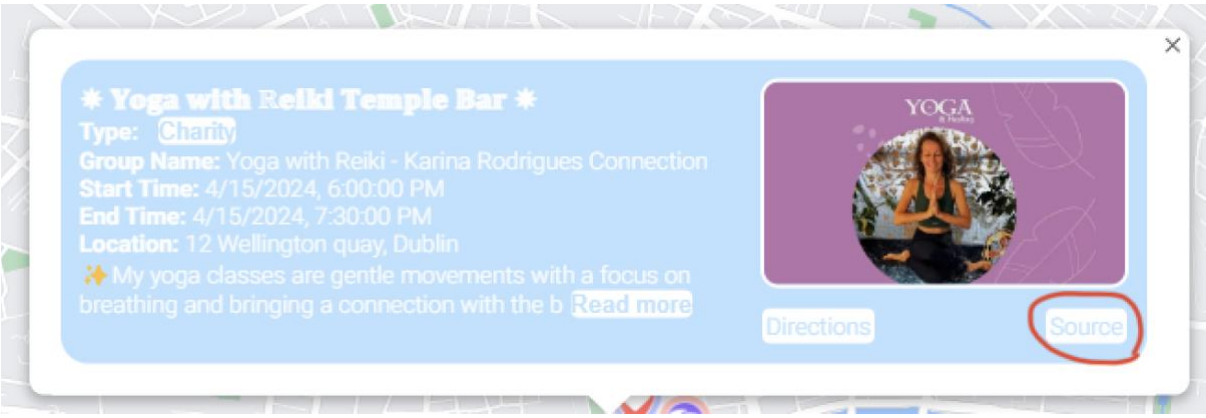
Directions Link to Google Maps Directions



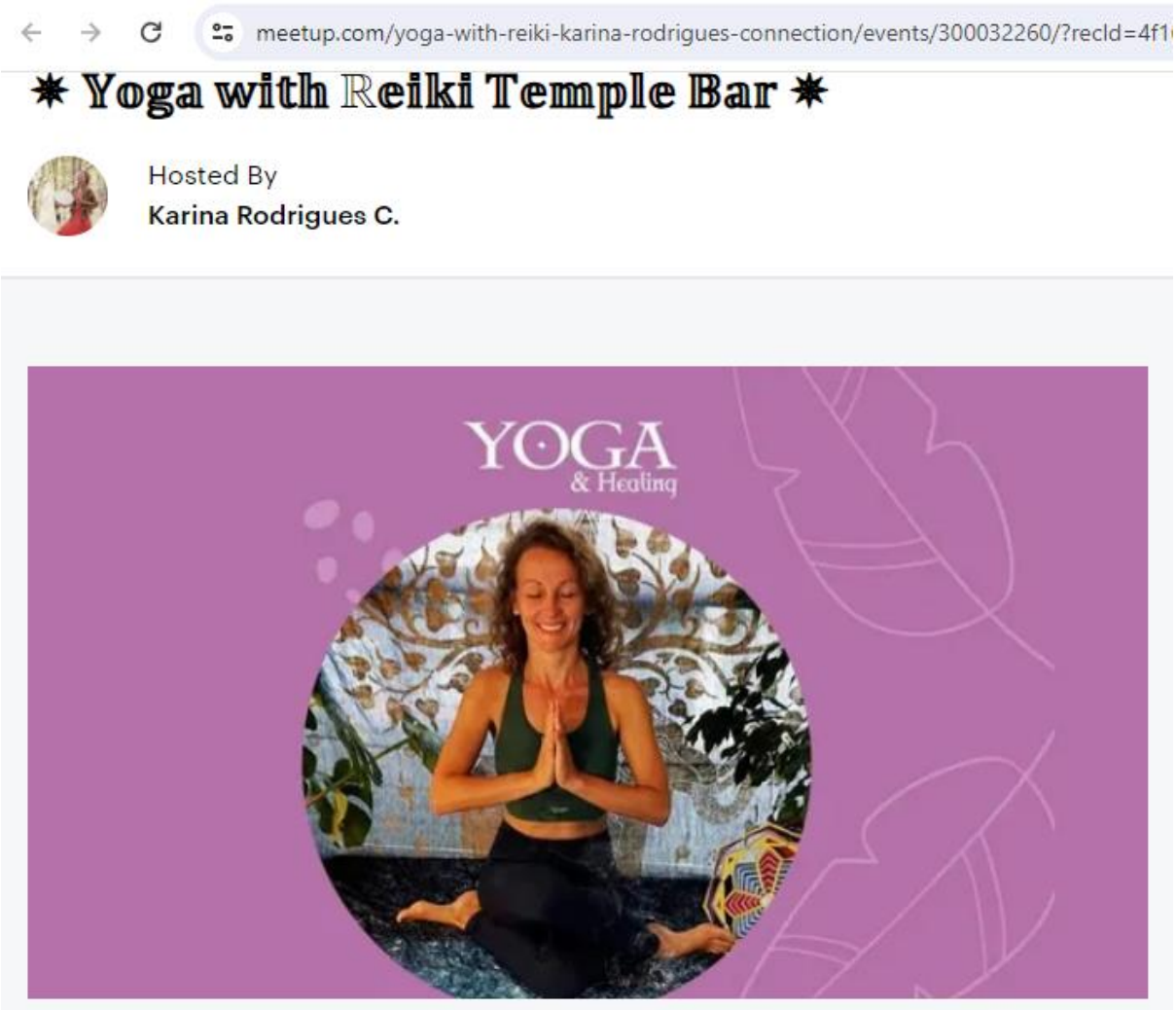
Google Maps Directions



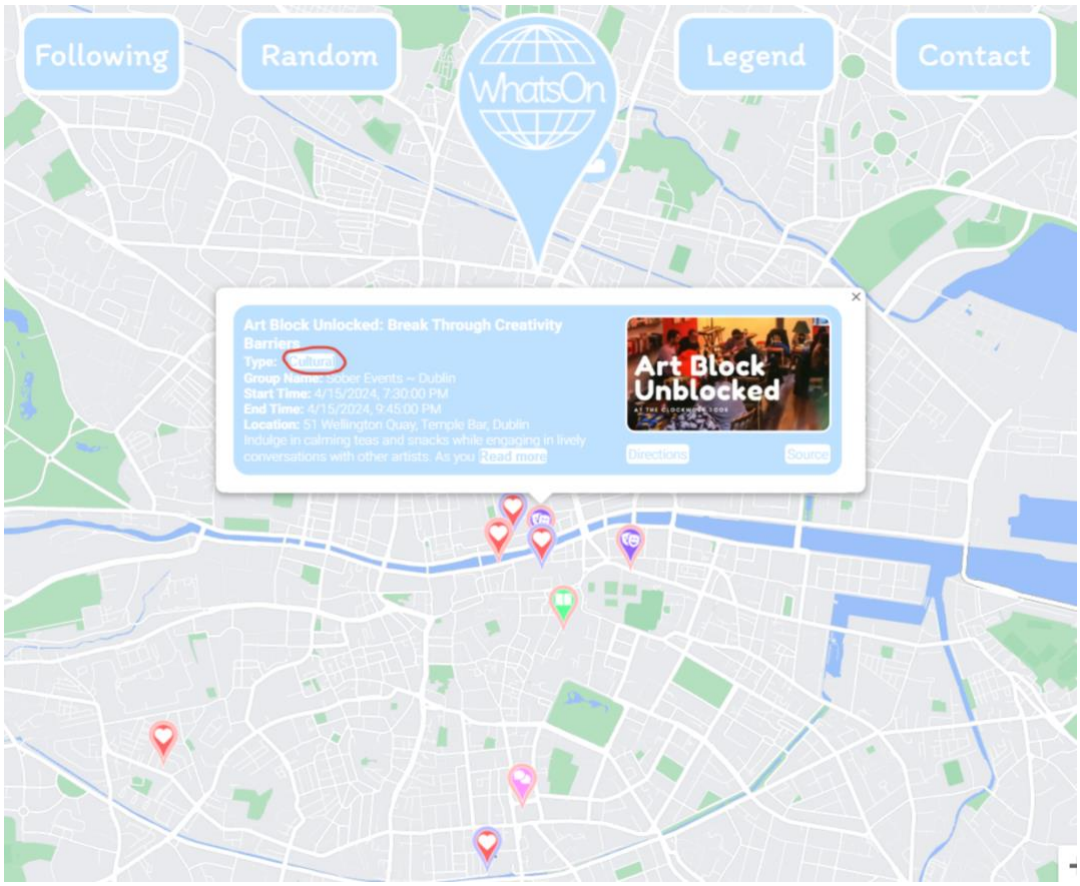
Directions Link to Event Source Site

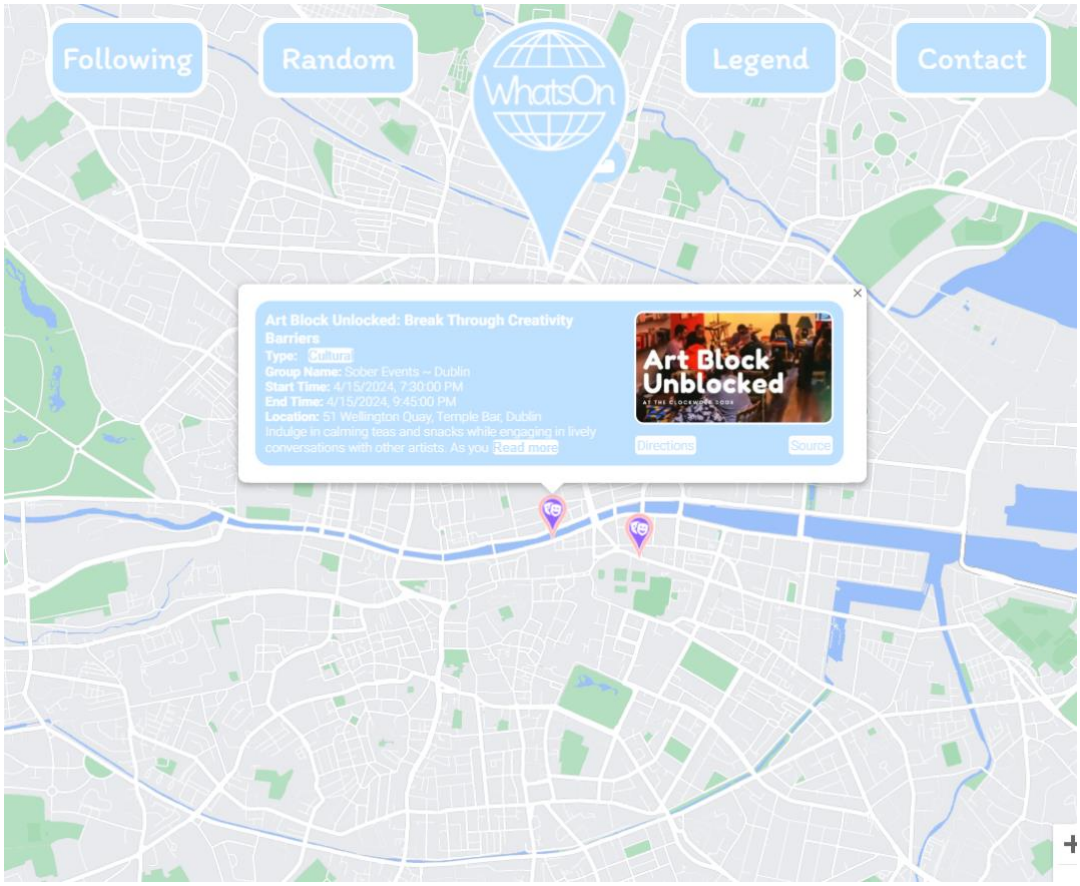


Event Source Site

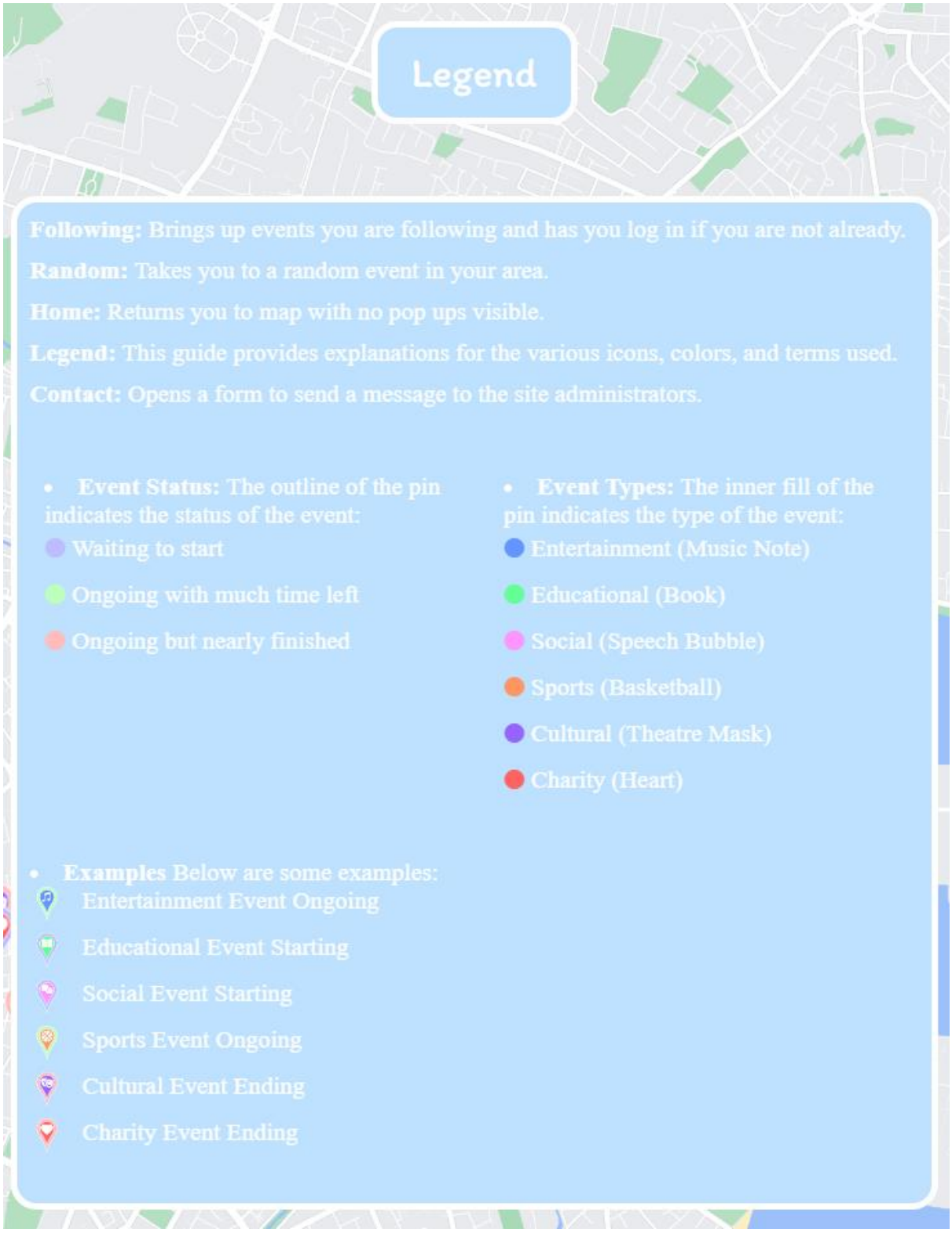


Filter Events By Type

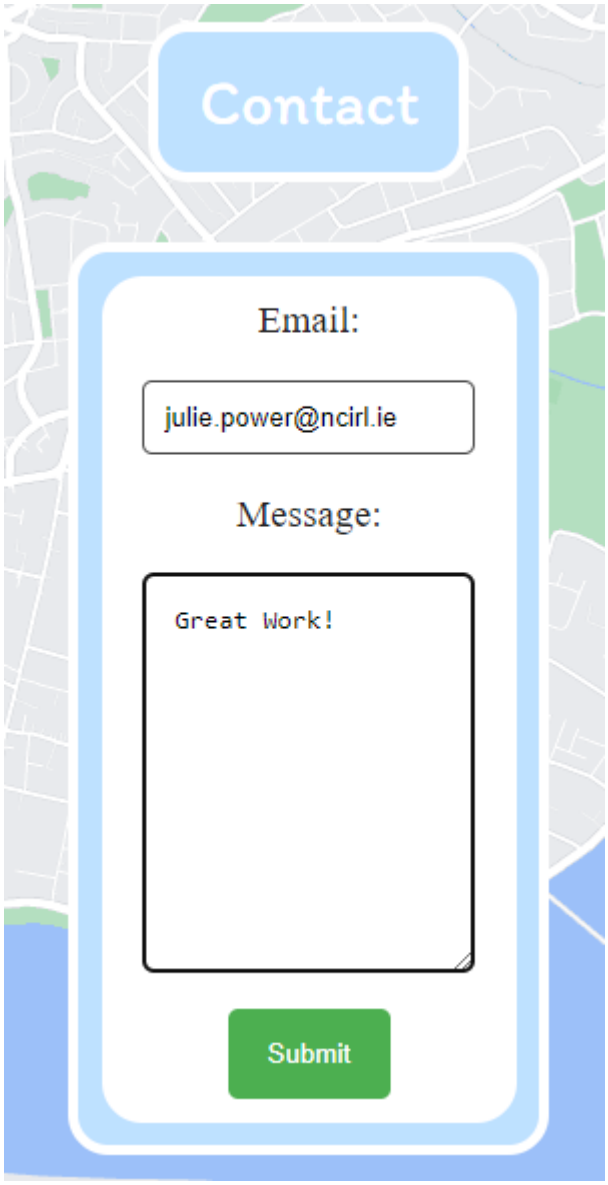




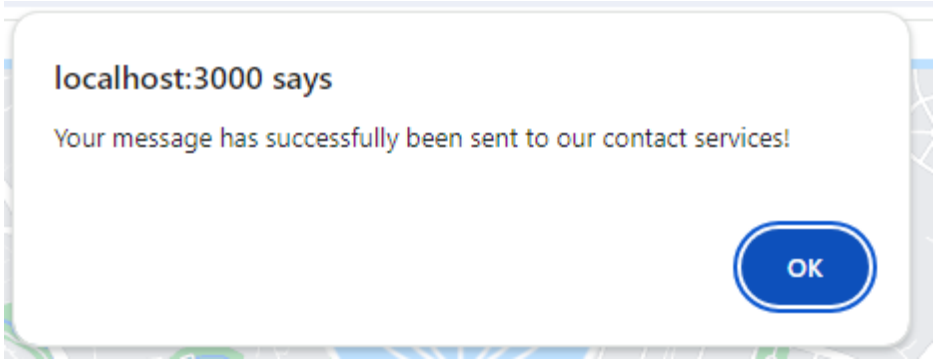
Legend


Legend
Following: Brings up events you are following and has you log in if you are not already.
Random: Takes you to a random event in your area.
Home: Returns you to map with no pop ups visible.
Legend: This guide provides explanations for the various icons, colors, and terms used.
Contact: Opens a form to send a message to the site administrators.
Event Status: The outline of the pin indicates the status of the event:
Waiting to start
Ongoing with much time left
Ongoing but nearly finished
Event Types: The inner fill of the pin indicates the type of the event:
Entertainment (Music Note)
Educational (Book)
Social (Speech Bubble)
Sports (Basketball)
Cultural (Theatre Mask)
Charity (Heart)
Examples Below are some examples:
Entertainment Event Ongoing
Educational Event Starting
Social Event Starting
Sports Event Ongoing
Cultural Event Ending
Charity Event Ending

Contact

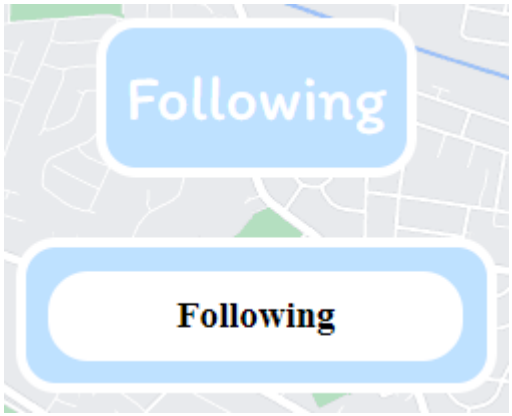


Confirmation Message

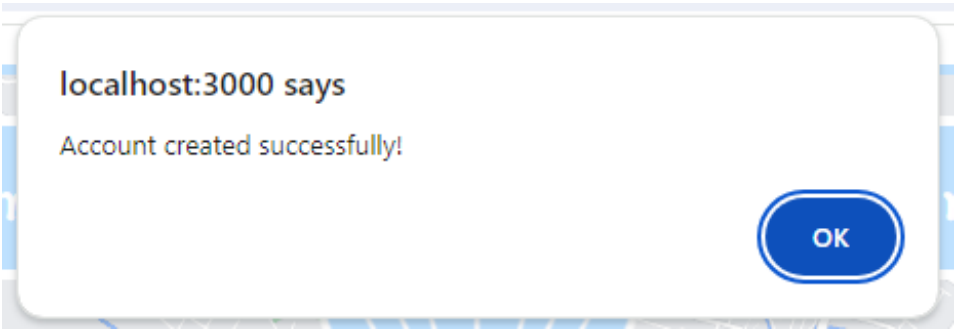


Following- User Account Login/Signup

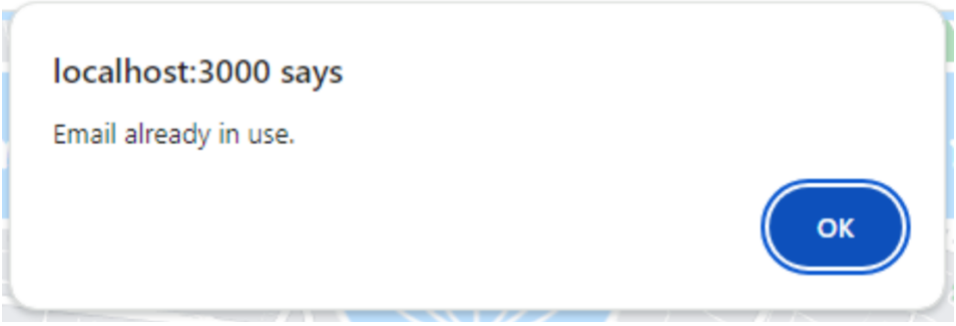
Successful Login Shows Followed Events



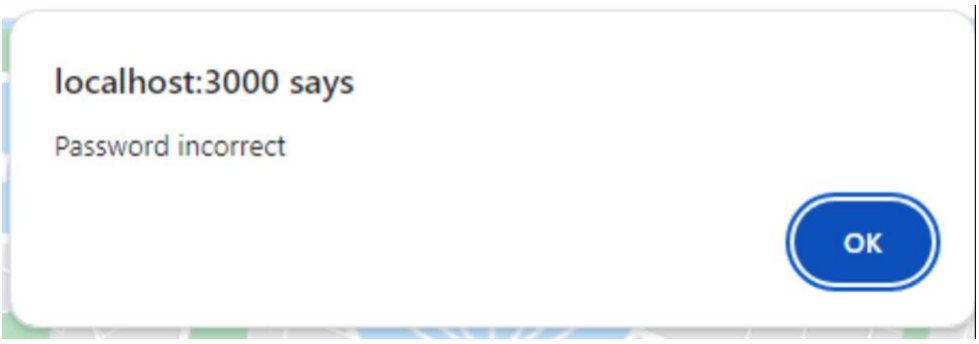
Successful Account Creation



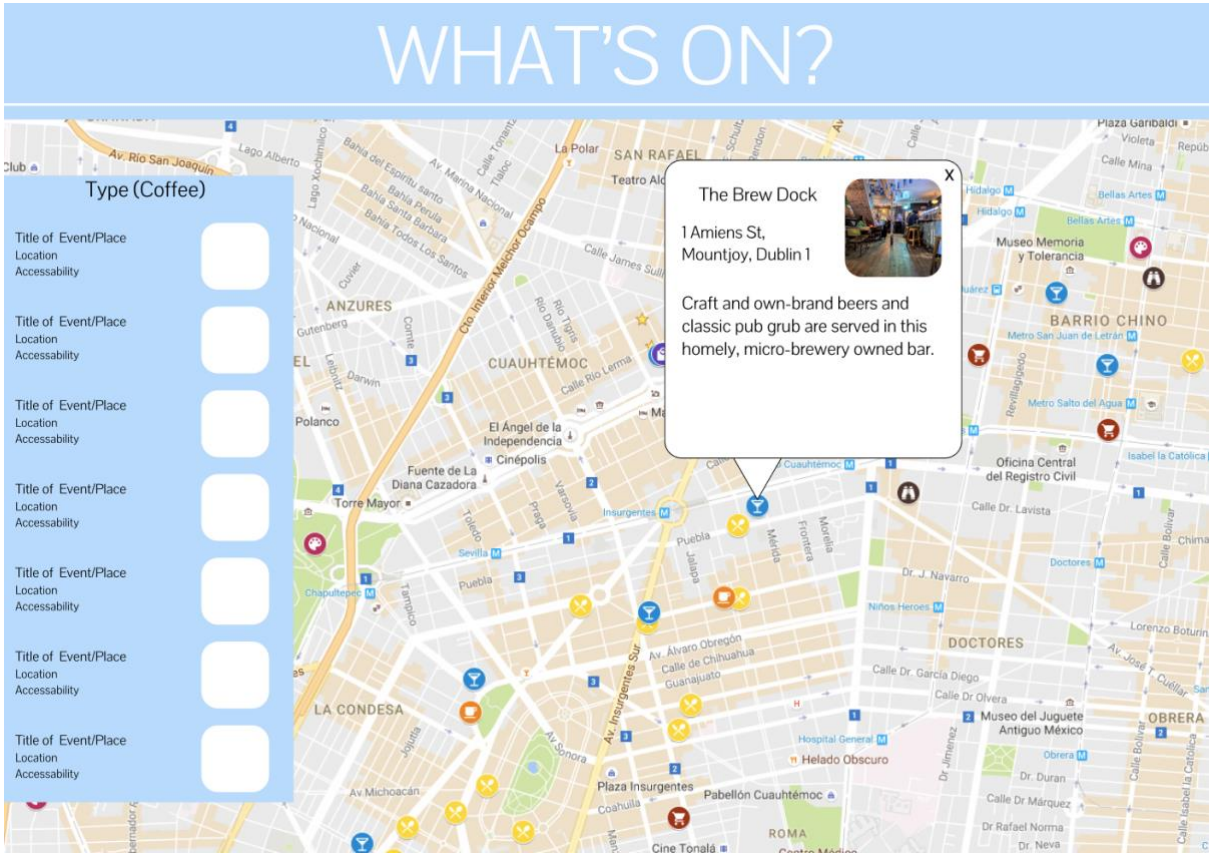
Email Already In Use



Incorrect password



First Mockup



1.5 Testing

2.5.1 Testing Tools

Black-box Testing Tools

Selenium- for testing frontend interactions with the browser

Postman- for testing APIs and verifying functionality

White-box Testing Tools:

Mocha- to test Javascript

Istanbul- for code coverage analysis

Unit Testing Tools

Supertest: for HTTP endpoints and APIs using Node.js

2.5.2 Test Plan

Objectives

1. Validate that all user requirements are met
2. Identify bugs and resolve them
3. Test expected behaviour under various scenarios
4. Assess usability

Scope

1. Browsing events on the map
2. Filtering events by type
3. Random event selection
4. Contacting WhatsOn
5. Following events
6. Viewing personalized events
7. Buying tickets for events
8. Getting directions to events

2.5.3 Black Box Testing

1. Verify that events are displayed on the map relative to the user's location
2. Verify that only events of the selected type are displayed when filtered
3. Verify that a random event is selected
4. Verify that the contact feature delivers messages to the WhatsOn Development team
5. Verify followed events are added to the user's list
6. Verify that events are displayed based on the user's preferences when selected
7. Verify that users are redirected to the event source
8. Verify that users are redirected to external mapping services that capture the events location accurately

2.5.4 White Box Testing

1. Verify that code for event retrieval is fetching data as expected from external APIs
2. Test error handling for event retrieval
3. Test error handling for contacting WhatsOn
4. Verify user authentication for following events
5. Verify that code for filtering events by type handles different event types as expected
6. Test user profiles by ensuring user preferences are saved and applied when prompted

2.5.5 Unit Testing

1. Tests backend for fetching event data from external APIs
2. Test individual functions responsible for filtering events, selecting random events, and user authentication
3. Tests integration between frontend and backend components, such as displaying events on the map

3. Conclusions

Overall we achieved our goal of implementing our use cases. Some such as filter events and random events went exactly as planned, others like Follow Events and View Personalised Events are more of a template for future development. Those two use cases were the lowest priority of all the use cases as they are not a core differentiation from competitors and substitutes.

The strength of the application is that Events do appear dynamically into the map and create the sense of joy we were hoping for with a Map based UI and a very user-friendly and simple design. Despite having to shift to WebCrawler five Days before the deadline we managed to reach all our user requirements in that time.

Events pins appearing by icon type and changing colour based on time also worked out very well. As did the random and filter function. They cycle through events very quickly exactly as intended. We set out to make use of the addictive qualities of low-commitment scrolling and dopamine-spiking novelty and we feel we have achieved that.

Also as we intended, we believe this is a positive use of the attention economy as instead if encouraging isolation, we are encouraging action.

There are however some drawbacks and weaknesses in our application. Principle that we do not have a sure-fire way to secure the Google Maps Api key and protect our group member from potential hacking. Being reliant on other companies services like this leaves us in a vulnerable position unless we were to pivot to open source. The vast majority of our time was spent trying to implement a very sophisticated back end and either load the Meetup API or build our own in a server. As as a consequence we were left with very little usable functionality at the final stage and had to rush day and night to get our application over the line.

It has been a very good learning opportunity to learn many new skills. Almost nothing that we ended up using had any of us ever done before. From map and event APIs, databases, encryption, very sophisticated Javascript functions and even just the use of modals and pop-up windows instead of traditional pages. It created a lot of problems for us to solve and we did manage to solve them.

The time we spent implementing the Meetup API didn't fail because of a technical error, we did manage to make it work, it was a failure however in doing adequate research and not assuming things would be straightforward because they appeared to be so. We also managed to encrypt the map key somewhat although it appears to not be as secure as hoped. We demonstrated excellent project management though and pivoted very quickly to a contingency that we made work.

So ironically in the end, with us, there was a lot of homework, a lot of planning, a lot of stress but we still managed to see WhatsOn.

4. Further Development

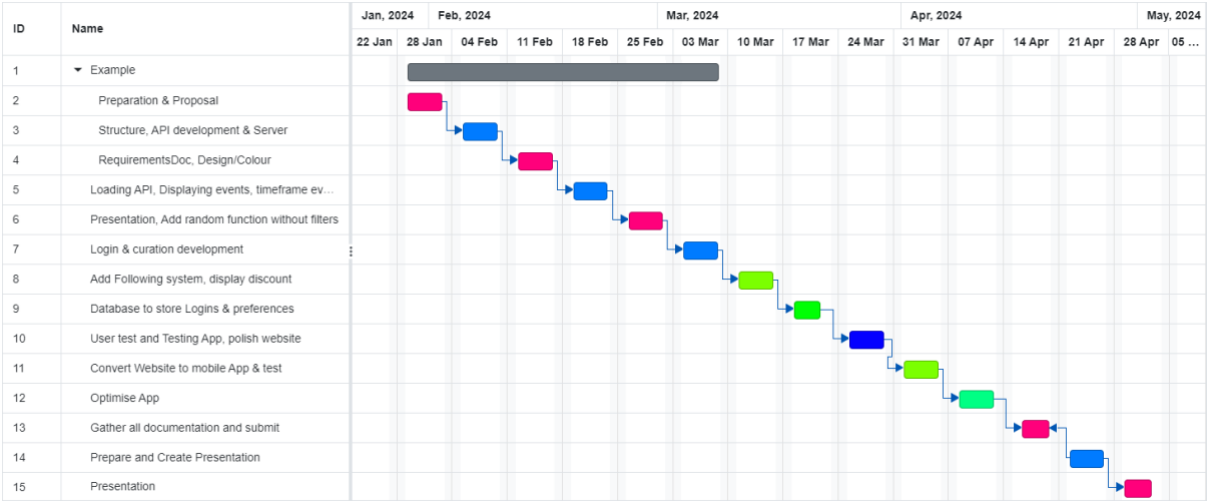
With additional time and resources, we would like to complete all the use cases in their entirety and and fleshing them out in other ways by implementing

- Following events and Follow events by Type full funcitonality
- Error handling is displayed to the user for all scenarios
- User location variable being used to search webcrawler to display in only their area
- More fleshed-out event type categories moving beyond 6
- Better visual contrast for accessibility
- Increasing the speed of the site and implementing more asynchronicity
- Improved responsiveness for mobile versions
- Security solution that does not depend on encryption and password.
- Adding other sites events such as Eventbrite

Beyond these incremental steps we would likely also make bigger choices such as opting for OpenStreetMaps API instead of Google Maps API. As with the Meetup API, APIs that are not Open source and freely available can have the compounding effect of limiting the possibilities of a project such as this. If given the time again we certainly would have opted for the webcrawler solution with OpenStreetMaps or the Leaflet Library Map. This was actually our earliest idea but we instead opted under the mistaken assumption that the more polished and professional options would make things more straightforward. We would add Algorithms to make more informed choices for the user over time, much like Spotify music suggestions. Finally we would make a native android and ios mobile application.

5. Appendices

5.1 Project Plan

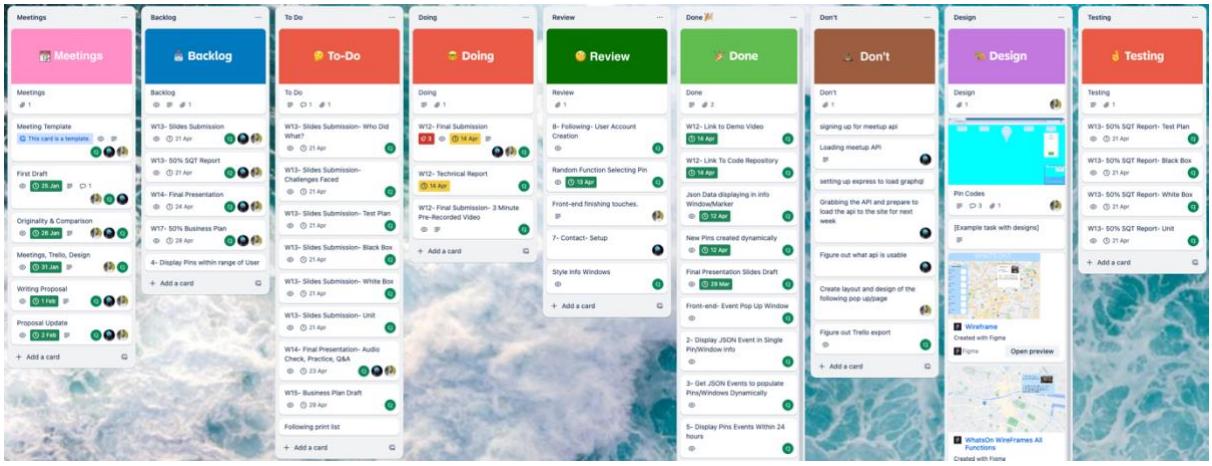


See the Collaboration Summary below for more Project Planning as we used them both

5.2 Collaboration Summary

Trello

Link

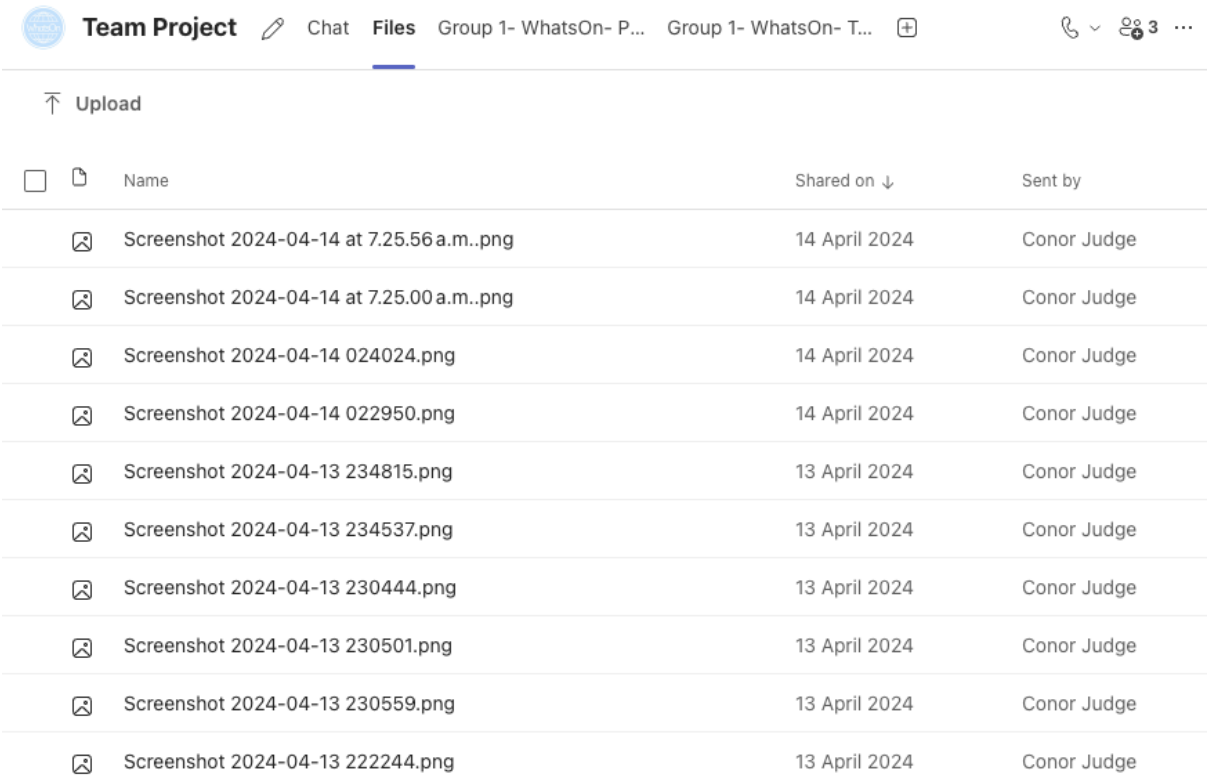


Used for

- Logging meetings
- Creating and assigning tasks
- Moving Tasks from Backlog->To-Do->Doing->Review->Done->Don't->->Design->Testing

We used Trello to log frequent meetings at the beginning of the project to ensure alignment. Tasks would be assigned to team members at the end of each meeting with assigned dates. Tasks would move from the Backlog to Done. We added a Meetings and Done and removed the standard Code Review card as Github was a more useful tool for this task. Don't was used for discarded ideas that no longer were viable. The Trello is public so you can scan through it if you are interested.

Microsoft Teams

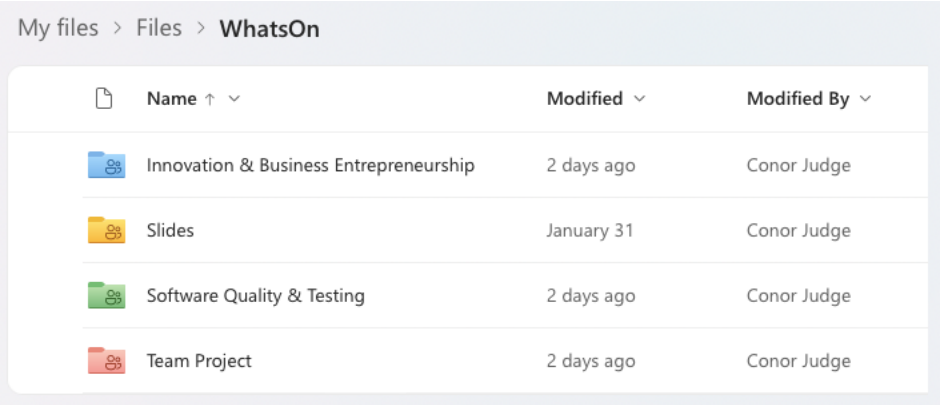


Used for

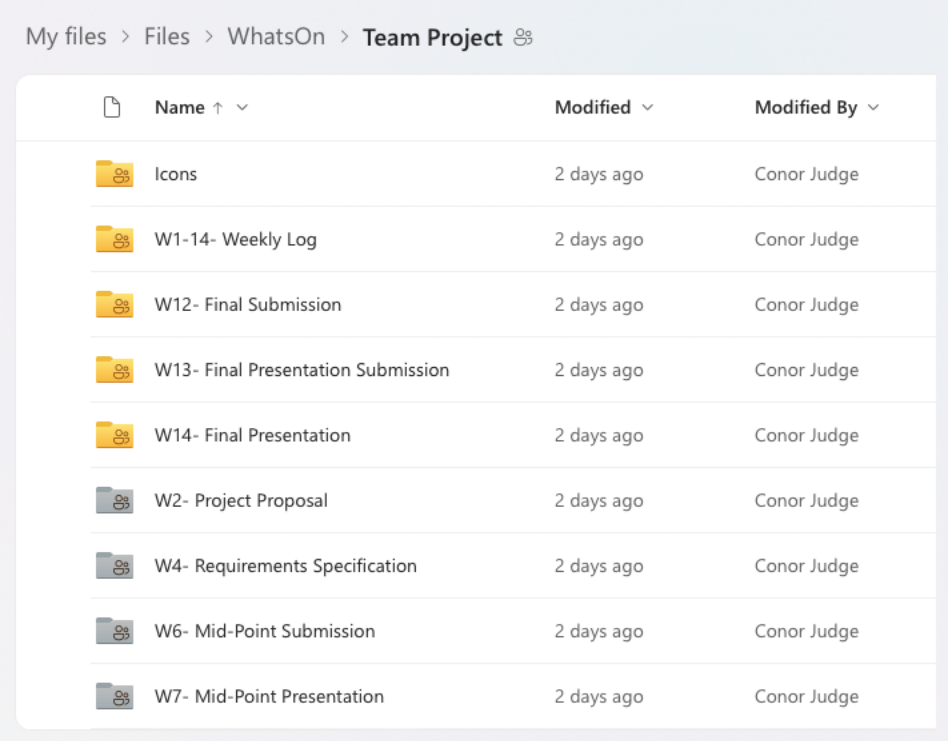
- Remote meetings
- Sharing screens
- Updates over chat
- Scheduling
- File sending
- Collaboration documents like word, powerpoint
- Integrated collaboration on Trello

OneDrive

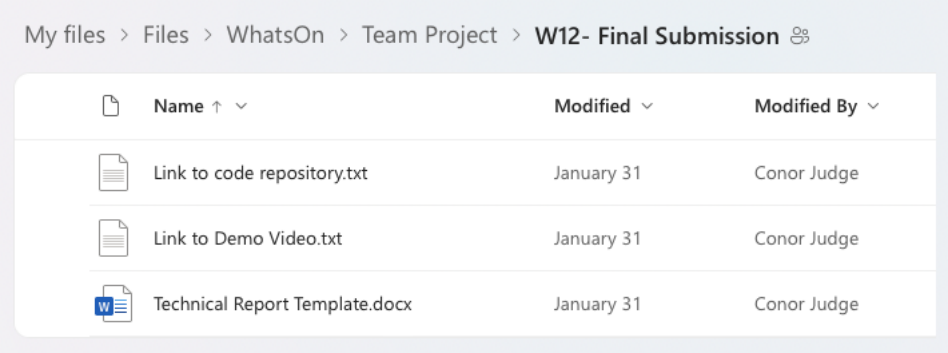
Root Folder For All Modules Concerned



Tracking Deliverables by Weeks, Files and progress by folder colour



Deliverable dates and requirements prepped in Week 2 for entire project

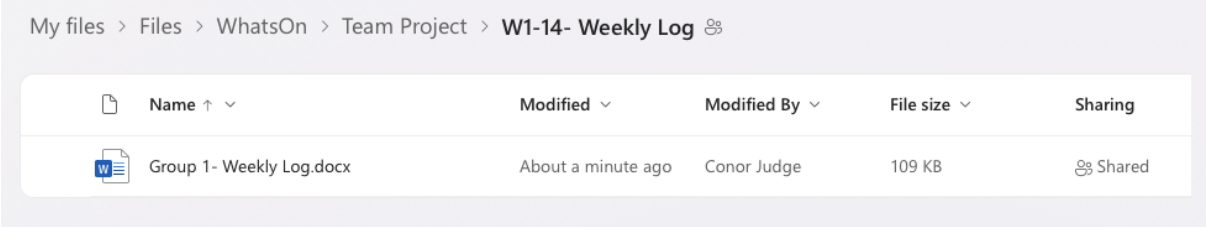


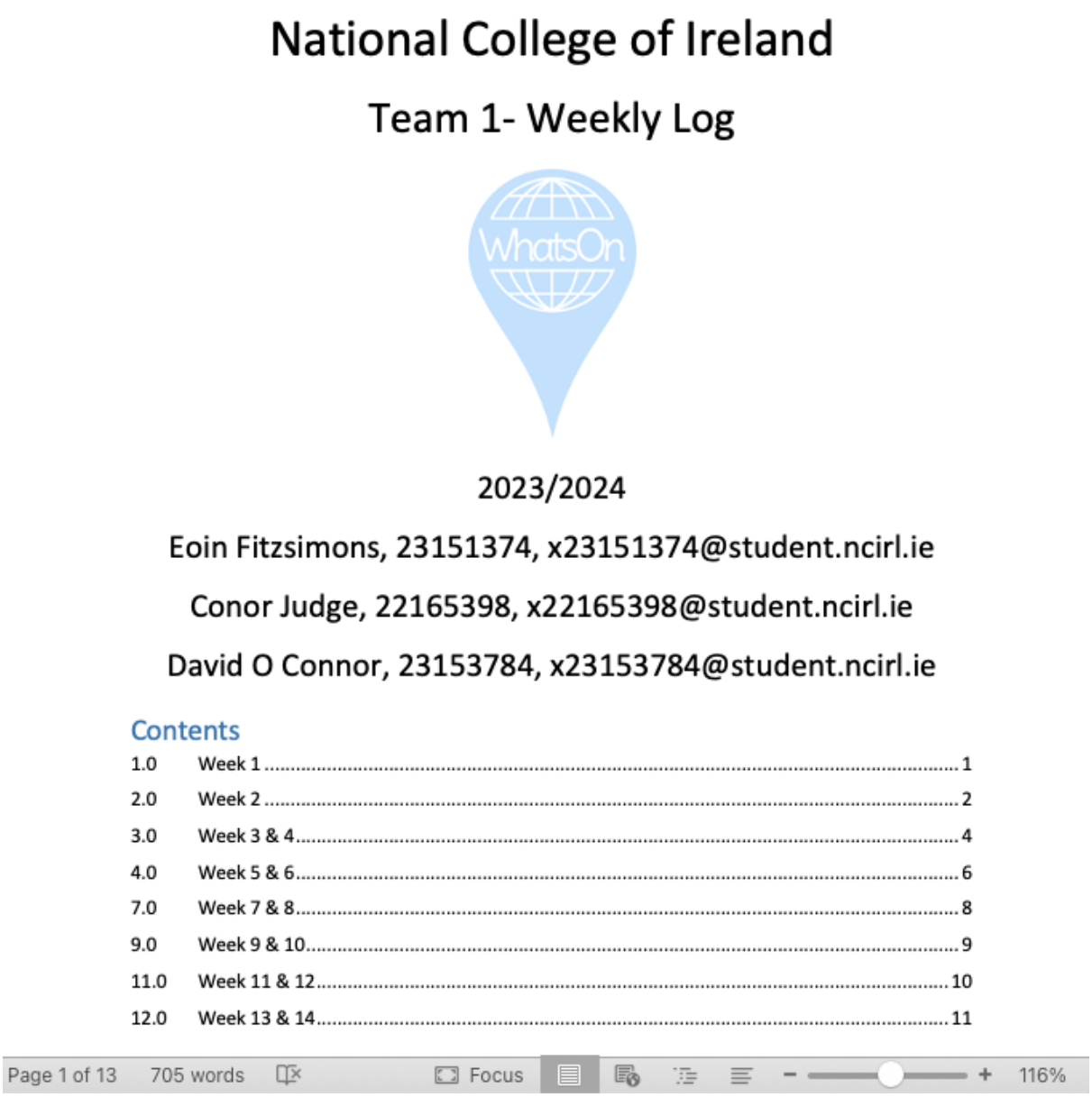
Used For

- Tracking deliverables across 3 modules
- Sharing documents

Shared Weekly Log

- Word Document on shared OneDrive folder
- Enabled all members to log their work
- Encourages transparency and accountability





National College of Ireland

Team 1- Weekly Log

2023/2024

Eoin Fitzsimons, 23151374, x23151374@student.ncirl.ie

Conor Judge, 22165398, x22165398@student.ncirl.ie

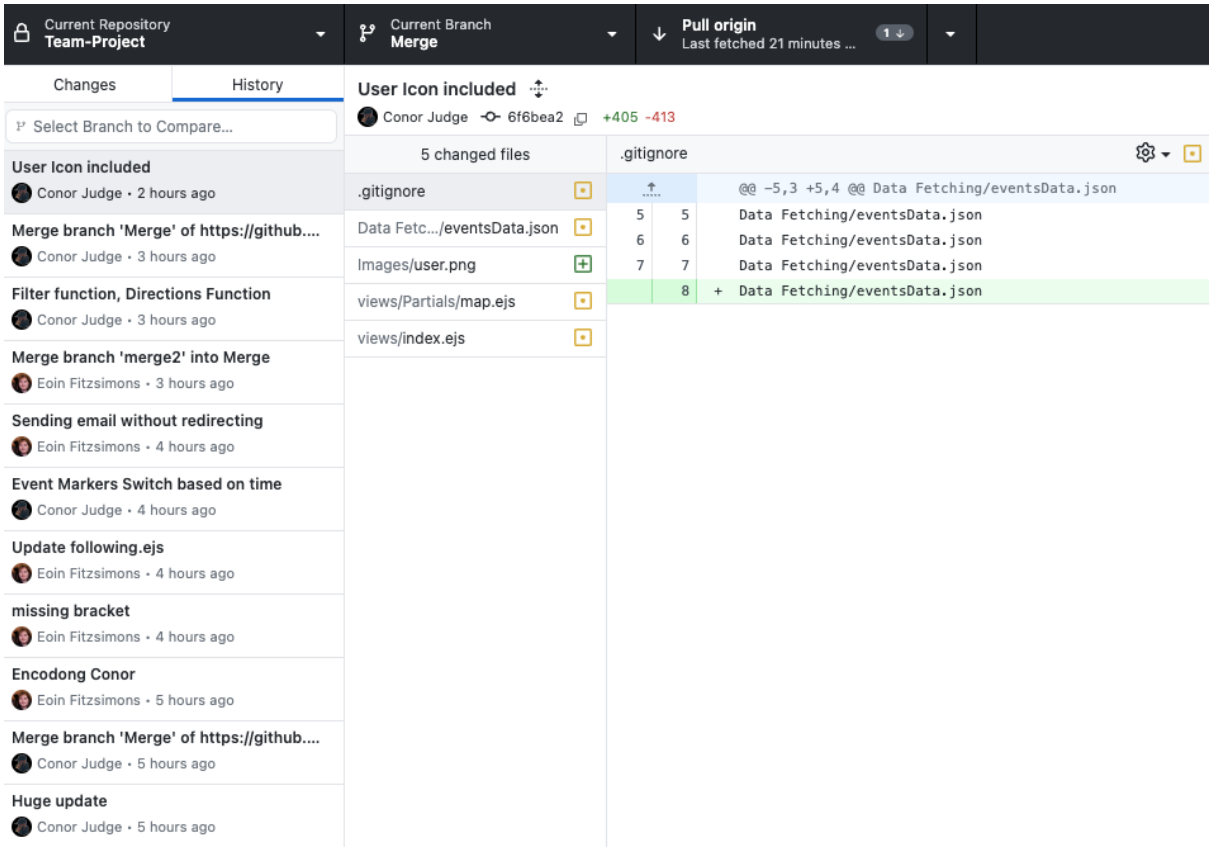
David O Connor, 23153784, x23153784@student.ncirl.ie

Contents

Github Desktop

Used For

- Version Control
- Updating each other on code
- Code Review
- Code History
- Reverting Code



Visual Studio Code

Used For

- Coding the project
- Integration with GitLens extension to see commit comments from Team members