

National College of Ireland

Bachelor of Science (Honours) in Computing

Software Development

2023/2024

Jade Fogarty

X20395471

X20395471@student.ncirl.ie

Event Point

Technical Report

Contents

| | |
|--|----|
| Table of Figures..... | 3 |
| Executive Summary..... | 4 |
| 1.0 Introduction | 5 |
| 1.1. Background..... | 5 |
| 1.2. Aims | 5 |
| 1.3. Technology | 6 |
| 1.4. Structure..... | 6 |
| 2.0 System..... | 7 |
| 2.1. Requirements | 7 |
| 2.1.1. Functional Requirements | 7 |
| 2.1.1.1. Use Case Diagram..... | 7 |
| 2.1.1.2. Requirement 1: View events | 8 |
| 2.1.1.3. Requirement 2: Login / Create an Account..... | 9 |
| 2.1.1.4. Requirement 3: Event information | 12 |
| 2.1.1.5. Requirement 4: User profile..... | 13 |
| 2.1.1.6. Requirement 5: User account | 15 |
| 2.1.1.7. Requirement 6: Custom user lists | 18 |
| 2.1.1.8. Requirement 7: User friends | 20 |
| 2.1.1.9. Requirement 8: Collaborative list | 22 |
| 2.1.2. Data Requirements | 23 |
| 2.1.3. User Requirements..... | 24 |
| 2.1.4. Security Requirements | 24 |
| 2.1.5. Usability Requirements | 24 |
| 2.1.6. Maintainability Requirements..... | 24 |
| 2.1.7. Robustness Requirements | 24 |
| 2.1.8. Availability Requirements | 24 |
| 2.1.9. Performance Requirements | 25 |
| 2.2. Design & Architecture | 25 |
| 2.3. Implementation | 27 |
| 2.4. Graphical User Interface (GUI) | 35 |
| 2.5. Testing | 38 |
| 3.0 Conclusions | 41 |
| 4.0 Further Development or Research | 42 |

| | | |
|--------|---|----|
| 5.0 | References | 43 |
| | References | 43 |
| 6.0 | Appendices..... | 43 |
| 6.1. | Project Proposal..... | 43 |
| 6.1.1. | Objectives | 44 |
| 6.1.2. | Background | 44 |
| 6.1.3. | State of the Art | 44 |
| 6.1.4. | Technical Approach..... | 45 |
| 6.1.5. | Technical Details | 45 |
| 6.1.6. | Special Resources Required | 46 |
| 6.1.7. | Project Plan | 46 |
| 6.1.8. | Testing..... | 48 |
| 6.2. | Ethics Approval Application (only if required)..... | 49 |
| 6.3. | Reflective Journals..... | 49 |
| | October | 49 |
| | November | 50 |
| | December..... | 52 |
| | January | 53 |
| | February..... | 54 |
| | March | 55 |
| | April..... | 57 |
| 6.4. | Invention Disclosure Form..... | 58 |
| 6.5. | Additional Functional Requirements..... | 62 |

Table of Figures

| | |
|--|----|
| Figure 1 Use Case Diagram of main system | 7 |
| Figure 2 Use Case Diagram for View Events [EP001] | 8 |
| Figure 3 Use Case Diagram for Login / Create an account [EP002] | 10 |
| Figure 4 Use Case Diagram for Event information [EP003] | 12 |
| Figure 5 Use Case Diagram for User profile [EP004] | 14 |
| Figure 6 Use Case Diagram for User account [EP005] | 16 |
| Figure 7 Use Case Diagram for Custom user lists [EP006] | 18 |
| Figure 8 Use Case Diagram for User friends [EP007] | 20 |
| Figure 9 Use Case Diagram for Collaborative list [EP008] | 22 |
| Figure 10 Architectural Diagram for Express backend server | 25 |
| Figure 11 Architectural Diagram for React frontend | 25 |
| Figure 12 MongoDB Collections structure | 26 |
| Figure 13 Class Diagram for Express backend server | 27 |
| Figure 14 Code snippet 1: events model | 28 |
| Figure 15 Code snippet 2: api call | 28 |
| Figure 16 Code snippet 2: api call cont. | 29 |
| Figure 17 Code snippet 3: database check | 29 |
| Figure 18 Code snippet 4: api response processing | 30 |
| Figure 19 Code snippet 5: store event data | 31 |
| Figure 20 Code snippet 6: get events for ViewEvents component | 31 |
| Figure 21 Code snippet 7: axios frontend middleware | 32 |
| Figure 22 Code snippet 8: getEvents function | 32 |
| Figure 23 Code snippet 9: /events route | 32 |
| Figure 24 Code snippet 10: GetAllEvents function | 32 |
| Figure 25 Code snippet 11: QueryAllEvents function | 33 |
| Figure 26 Code snippet 12: SendFriendRequest component | 33 |
| Figure 27 Code snippet 13: sendFriendRequest call to server | 34 |
| Figure 28 Code snippet 14: SendFriendRequest in friend_controller.js | 34 |
| Figure 29 Code snippet 15: NewFriendRequest query | 34 |
| Figure 30: Event Point design | 35 |
| Figure 31 GUI 1: ViewEvents [EP001] | 35 |
| Figure 32 GUI 1: View Events[EP001] cont. | 36 |
| Figure 33 GUI 2: SearchUsers | 36 |
| Figure 34 GUI 3: ViewProfile (User profile [EP004] | 37 |
| Figure 35 GUI 4: EventInformation [EP002] | 37 |
| Figure 36 GUI 5: UserAccount [EP005] | 37 |
| Figure 37 GUI 6: AddToList | 38 |
| Figure 38 Tests directory | 38 |
| Figure 39 AccountComponents.spec.ts overview | 38 |
| Figure 40 Jest console error | 39 |
| Figure 41 Test results 1 | 39 |
| Figure 42 Test results 2 | 40 |
| Figure 43 Playwright Test report | 40 |
| Figure 44 Playwright Test report for EventsComponents.spec.ts | 40 |
| Figure 45 Lighthouse report - locally ran | 41 |
| Figure 46 Lighthouse report - deployed application | 41 |
| Figure 47 API research spreadsheet | 42 |

Executive Summary

This report looks to provide a comprehensive and detailed outline of the development process of my final project titled Event Point. It highlights the technologies and tools used to build the web application throughout the various stages of planning and development. These technologies (Express.js, React, MongoDB, Node.js, GitLab, and CircleCI) will be used to create a full stack web application utilizing the MVC architecture. The report identifies and outlines the functional requirements in detail using Use Cases and supporting diagrams for a visual representation of the flow of the application and also addresses the nonfunctional requirements expected of this project.

The architectural design of the system is detailed and the role each component is outlined further when the implementation of some key functionalities is broken down and described in a step-by-step process. This give some key insights into the various functions which are called as the user interacts with the web application. Due to the MVC architecture of the system, each component is separated enabling code reusability and best practices. These processes are supported with screenshots, code snippets, and diagrams as a visual representation of the system.

The conclusions of this report identify and address the limitations of this application before acknowledging the potential for future development further on. After analysing and reflecting on the overall project, the conclusions of this report have found that Event Point enhances the event going experience for its users and provides an engaging platform for those who wish to personalise their interactions with events and enhance their social interactions with other event goers.

1.0 Introduction

1.1. Background

I undertook this project to address some of the issues users face when attending or interacting with events. Due to the high volume of ticketing vendors, and platforms promoting their events, users are forced to browse each vendors website in order to find an event they are interested in. Additionally, an event can be easily missed when browsing a high number of websites if a specific one is not viewed. This web application looks to solve this problem by creating an aggregated list of events from vendors and displaying them in the one, convenient location.

Another issue event goers often face when planning to attend events surrounds the social aspect of it. Often people may see an event they are interested in but don't know anyone planning to attend which demotivates them from going. By allowing users to browse profiles and view custom lists they are able to interact with users similar interests who may be attending the same event. This social aspect enhances user's experience with the web application and event going in general.

An additional issue faced by event goers surrounds the purchase of resale tickets. Due to the high demand for specific events, tickets are difficult to obtain at the time of the general sale through the official ticketing vendors platform. Once users look to purchase resale tickets this puts them at risk of falling victim to a scam. Untrustworthy parties can sell the same ticket to multiple different buyers, once a buyer goes to the event, they will be denied entry if they are not the first person with the ticket to enter.

There is also a risk as both a buyer and a seller of a resale ticket when transferring the ticket or payment. Either party can refuse to fulfil their end of the agreement once they have received the payment or ticket. This project looks to reduce the risk for everyone involved in this transaction by acting as a mediator. Once a buyer and a seller are established on the platform the web application will broker the transaction, making it secure for both parties involved. Additional features to address these issues which has potential to be implemented in the future development of this project are outlined in this report.

1.2. Aims

The aim of this project is to develop a full stack web application which prioritises user experience when browsing, planning, and attending events. The project looks to implement a number of functionalities to achieve this goal and provide users with an enhanced event going experience. These features include aggregated event listings, creating an account, adding events to custom user lists, interacting with friends, and sharing collaborative lists with other users.

The aggregated event listings provide users with convenient location to browse a wide variety of events from different ticketing vendors. This reduces the amount of site searching users must do to find events they are interested in. It also reduces the likelihood of missing out on an event as they are in the one location and won't be missed or overlooked if a user doesn't visit a specific ticketing site.

Enabling users to create an account and use features such as creating custom event lists, add friends, and create collaborative lists increases the user's engagement with the web application and allows them to personalise their event going experience by customising their profiles and adding a social aspect through interacting friends or other users.

These functionalities will be implemented by developing a backend Express server, a MongoDB database, and a React frontend, which is outlined in more detail throughout this report.

1.3. Technology

The backend server is developed using an Express server framework to interact with both the frontend and the database. The server utilises the Mongoose library to create the necessary schemas, collections and perform queries on the MongoDB database. The Axios library is also imported to handle requests and responses to/from the frontend when processing data while also making external API calls to gather the event data to be saved to the database.

MongoDB is the database of choice in this project as it has flexible schema structures, handles JSON data successfully, and integrates well with Node and Express servers. From my experiences utilising this throughout my internship, I found this to be the more favourable option in comparison to MySQL. It also has different tools to use when handling the data locally and in a remote environment. In early stages of development, I used the locally ran MongoDB Compass as an efficient way to handling the data. As the project moved to the late stages of development, I created a deployed database using MongoDB Atlas.

The frontend is created using the React framework and implements Bootstrap to ensure a responsive GUI along with my custom HTML and CSS components. It also uses Axios middleware to handle all requests and responses to the backend Express framework. Separating the frontend and the backend allows me to develop a full stack web application using the MVC architecture. This also allows each components purpose to be clearly defined and enables reusability of components within each framework.

Additionally, JavaScript is the main programming language this web application is written in alongside HTML, and CSS.

1.4. Structure

Section 1 provides the contextual overview of this report and answers questions such as: what does this project involve? why it is being undertaken? and how will it be developed?

Section 2 highlights the requirements specifications of this project. Each functional requirement is listed by priority and contains a use case and a diagram, outlining details the flow within the system. The structure of the application along with details of implementation and testing is also outlined in this section.

Section 3 outlines the conclusions of the project. Here, the project will be analysed for its benefits, drawbacks, and limitations which have been encountered throughout the planning and developing stages.

Section 4 addresses the ways in which the project can be further developed if provided with the time and resources. This will outline the potential for future growth outside of the time constraints of the course.

Section 5 includes any referenced material cited within this report.

Section 6 is the appendix, containing additional documents and information to the report.

2.0 System

2.1. Requirements

The web application's layout and functionalities should allow all users to easily navigate through different pages and interact with its features with little to no errors. Users who wish to access features such as Custom User Lists, Add Friends, Book Ticket, and Secure Ticket Transfer will have the option to Login or Create an Account. Existing users should be able to enter their details and access their account easily. New users should have the option to Create an account and access additional features.

2.1.1. Functional Requirements

The functional requirements outlined below, explain the way in which a user can interact with the web application. The order in which they are listed, are relevant to both their priority in the overall system, and also the predicted order in which a user will move through the system (with slight variations as users are faced with different options).

2.1.1.1. Use Case Diagram

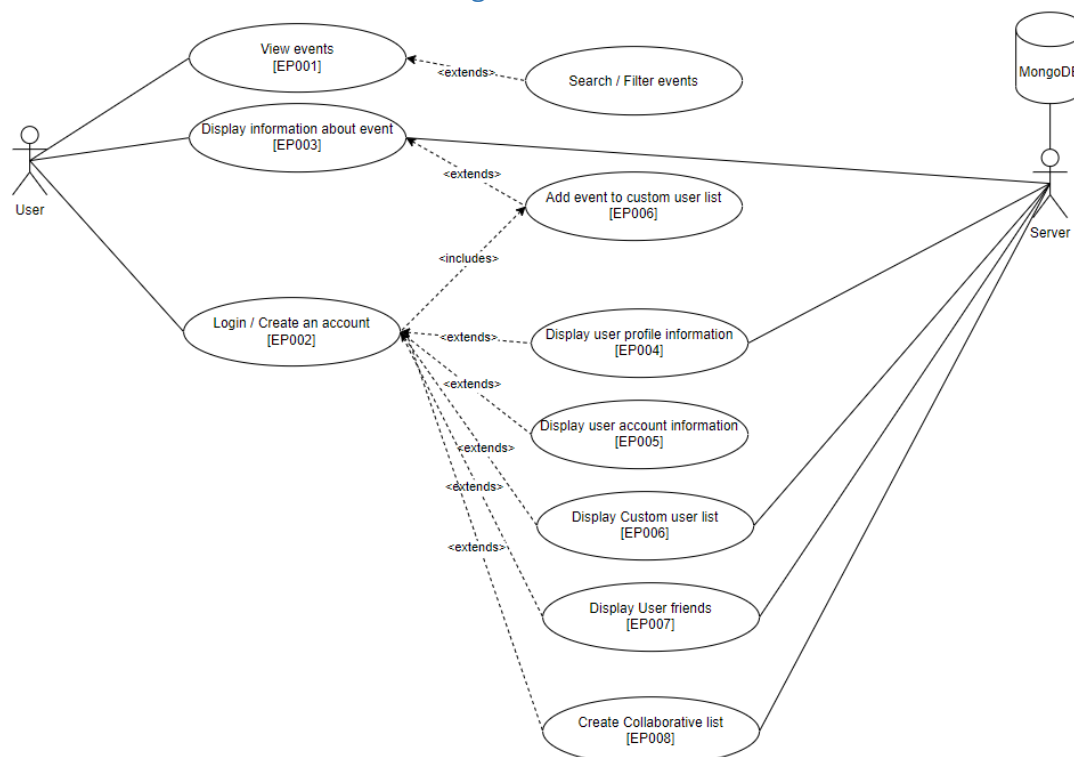


Figure 1 Use Case Diagram of main system

2.1.1.2. Requirement 1: View events

Description & Priority

The user will be presented with an aggregated list of upcoming events from different platforms (APIs for Ticketmaster, DataThistle, and Skiddle). This is the main functionality of the web application and will be required to implement additional features such as Create list, Add Friend, and Collaborative List.

Priority: High

Use Case

View Events

[EP001]

Scope

The scope of this use case is to enable users to view a large list of upcoming events.

Description

This use case provides users with a wide list of upcoming events, which will then allow users to view more information for specific event and add selected events to custom lists linked to the user's account which they can share with friends.

Use Case Diagram

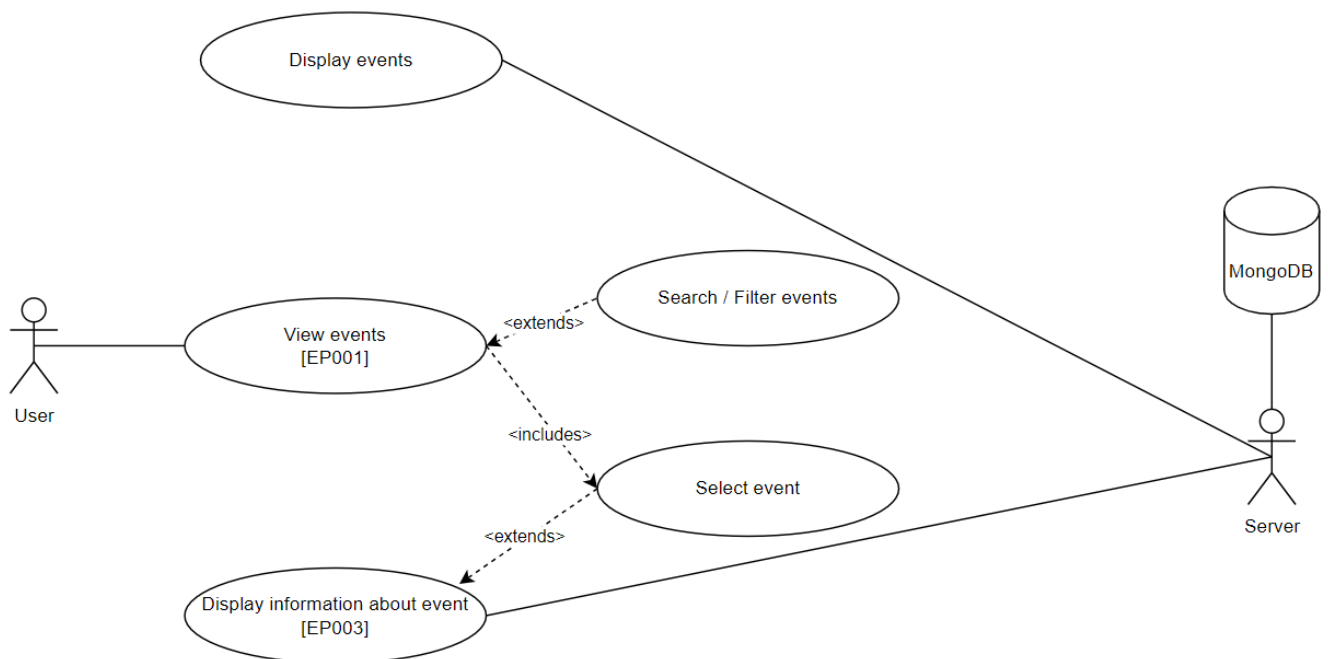


Figure 2 Use Case Diagram for View Events [EP001]

Flow Description

Precondition

The web application is running.

Activation

This use case starts when a user has opened the web application.

Main flow

1. The web application loads.
2. The system presents an aggregated list of upcoming events to the user.
3. The user selects an event to view (A1) (A2).
4. The system presents the user with more information about the selected event.

Alternate flow**A1 : Search Events**

1. The user searches the event listings using the search bar.
2. The system takes this user input.
3. The system refines the events being displayed (E1).
4. The system presents the user with an updated list of events.
5. Continue from step 3 of the main flow.

A2 : Filter Events

1. The user filters the event listings using the dropdown filter menus.
2. The system notes this user input.
3. The system refines the events being displayed (E1).
4. The system presents the user with an updated list of events.
5. Continue from step 3 of the main flow.

Exceptional flow**E1 : No events found**

1. The system has no events matching the applied search / filter.
2. The system displays an error message stating there are no events found.
3. The user clears the search / filter inputs.
4. The use case returns to step 3 of the main flow.

Termination

The system presents the next web page (event information)

Post condition

The system goes into a wait state.

2.1.1.3. Requirement 2: Login / Create an Account**Description & Priority**

User will create an account or log in to an existing account. This step will be required for users to access additional functionalities of the web application. These include Add friends, Create lists, and Collaborative Lists.

Priority: High

Use Case

Log in / Create an account

[EP002]

Scope

The scope of this use case is to allow users to login / create an account to the web application.

Description

This use case describes the process in which users will access their account. If the user does not have an account, they will have the option to create a new account and access additional features of the web application.

Use Case Diagram

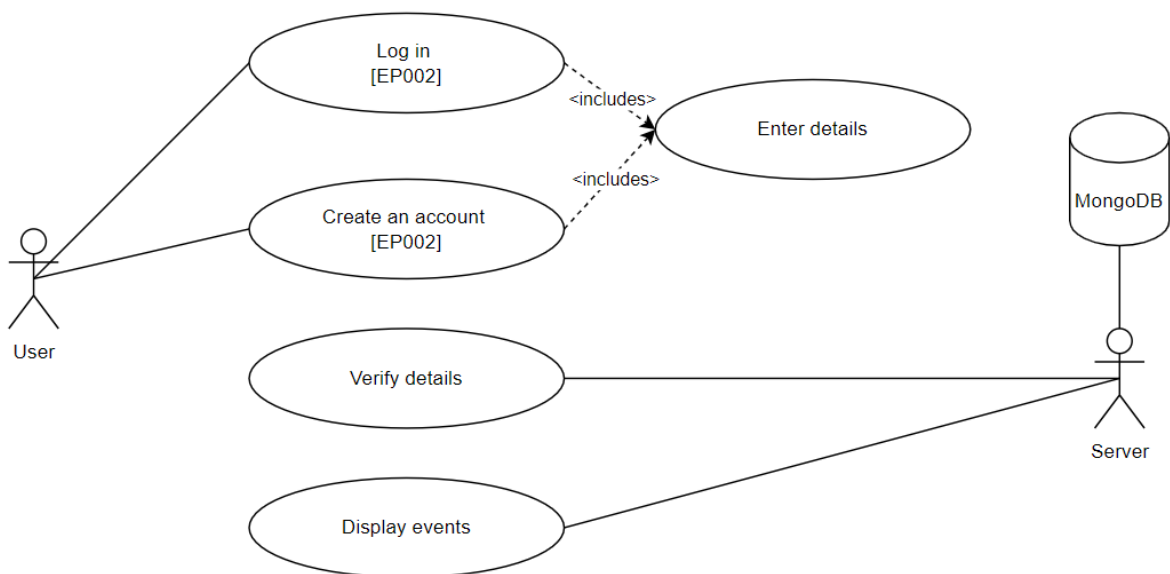


Figure 3 Use Case Diagram for Login / Create an account [EP002]

Flow Description

Precondition

The server is running and accessible to users.

The web application is open on the landing page.

Activation

This use case starts when a user opens the Log in / create an account page.

Main flow

1. The web application loads.
2. The user clicks 'Login' button.
3. Log in page is open.
4. The user enters log in details (E1) (E2) (A1).
5. The system verifies the details entered.
6. The details the user entered are correct.
7. The next page loads (View Events).

Alternate flow

A1 : Create an account

1. The Create an account page is open.
2. The user enters details to create an account.
3. The system verifies the details entered (E3) (E4).
4. The system creates an account.
5. Continue from step 7 of the main flow.

Exceptional flow

E1 : Incorrect details

1. The system verifies the details entered.
2. The details the user entered are incorrect.
3. The system displays an error message stating the details entered are incorrect.
4. Return to step 4 of the main flow.

E2: No account

1. The system verifies the details entered.
2. The system has no record of these details.
3. The system displays an error message stating there is no account found linked to these details.
4. The user clicks the 'Create an account' button.
5. Alternate flow is triggered (A1).

E3: Account already exists

1. The system already has a record of these details (username or email).
2. The system displays an error message stating there is an account linked to these details.
3. The user clicks the 'Login' button.
4. Continue from step 4 of the main flow

E4 : Weak password

1. The user enters a password that doesn't meet the minimum requirements.
2. The system displays an error message to the user stating the requirements needed.
3. The user enters a suitable password.

4. Return to step 5 of the main flow.

Termination

The system presents the next page (View Events).

Post condition

The system goes into a wait state.

2.1.1.4. Requirement 3: Event information

Description & Priority

Once the user has selected a specific event the system will present the event information to the user. From here the user will have the option to interact with the selected event. They will have ability to read information about the event, and save the selected event to a Custom List.

Priority: High

Use Case

Event information

[EP003]

Scope

The scope of this use case is to provide event information to the user and enable them to further interact with the event through the web application.

Description

This use case describes the processes a user may follow after they have selected a specific event they wish to view more information about.

Use Case Diagram

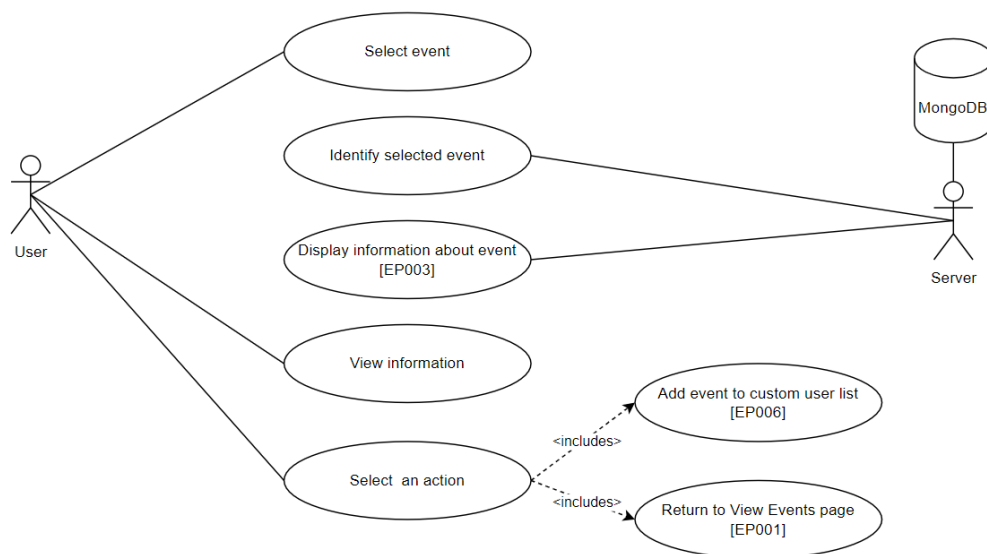


Figure 4 Use Case Diagram for Event information [EP003]

Flow Description

Precondition

The server is running and accessible to users.

The web application is on a specified event page.

Activation

This use case starts when a user selects an event.

Main flow

1. The system identifies the selected event.
2. The system presents the event information to the user.
3. The user clicks 'Add to list' button (A1)
4. The system presents the user's lists (E1).
5. The user selects the list to add the event to.
6. The user returns to the event listings page.

Alternate flow

A1: No action

1. The user chooses to take no action with the selected event.
2. Return to step 5 of the main flow.

Exceptional flow

E1: No lists

1. The user has no lists associated with their account.
2. The user clicks the 'Create List' button.
3. The user follows the steps of [EP006] main flow to complete this action.

Termination

The system presents the events listings page.

Post condition

The system goes into a wait state.

2.1.1.5. Requirement 4: User profile

Description & Priority

Allows users to view their profile which can be displayed to other users (if their profile visibility is set to public). Created custom lists and added friends associated with the user will be displayed here.

Priority: High

Use Case

User profile

[EP004]

Scope

The scope of this use case is to enable users to view their profile which will be visible on the web application to other users. They will also be able to view their custom lists and friends from this feature.

Description

This use case describes the user profile display feature of the web application.

Use Case Diagram

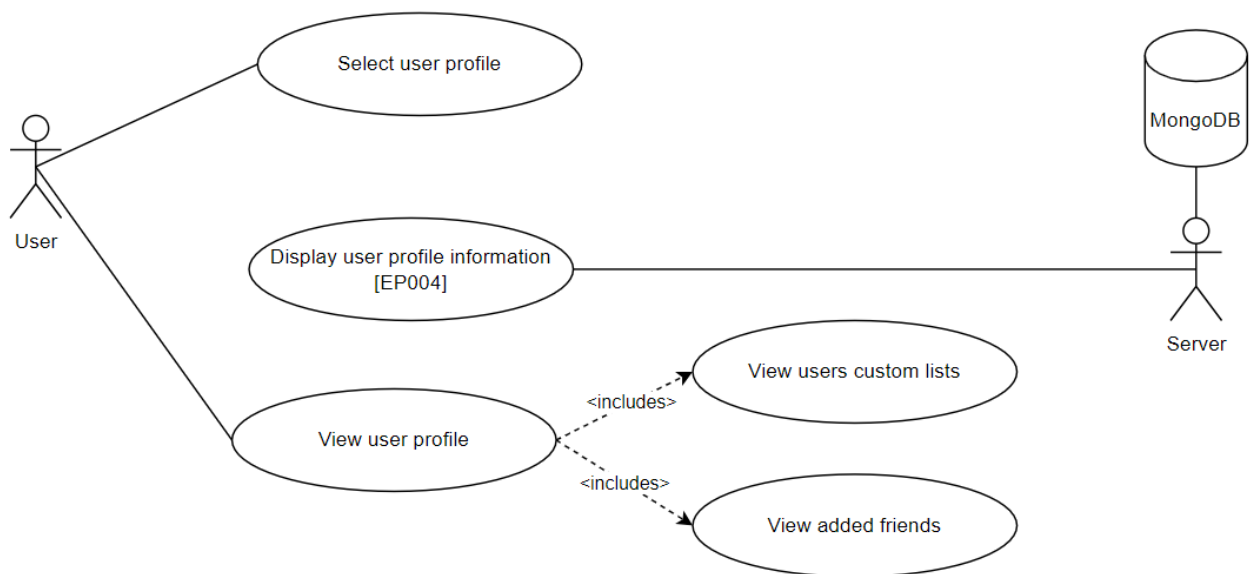


Figure 5 Use Case Diagram for User profile [EP004]

Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user clicks the 'Profile' button in the header dropdown.

Main flow

1. The user clicks the 'Profile' button.
2. The system displays the profile information to the user.
3. The user clicks a Custom List displayed on their account (A1) (E1).
4. The system displays the Custom List information.
5. The user returns to the profile page.

Alternate flow

A1 : User friends

1. The user clicks a Friend's profile displayed on their account (E2).
2. The system displays the profile of the selected friend.
3. Continue from step 5 of the main flow.

Exceptional flow

E1 : No lists found

1. The system has no available Custom Lists to display.
2. The system displays an error message saying there are no lists found.
3. Return to step 5 of the main flow.

E2 : No friends found

1. The system has no added friends to display.
2. The system displays an error message saying there were no friends found.
3. Return to step 5 of the main flow.

Termination

The system presents the user profile page.

Post condition

The system goes into a wait state.

2.1.1.6. Requirement 5: User account

Description & Priority

This is an essential feature to allow users to view and manage their account information. User profile information will be updated from here as changes are made. Users will have the option to set their profile visibility to public or private. Users will also have the option to delete their account in this functionality.

Priority: High

Use Case

User account

[EP005]

Scope

The scope of this use case is to enable users to manage their profile and account details.

Description

This use case allows users to view, edit and delete their account information. If the username is updated in this section, it will be displayed accordingly in the user profile feature.

Use Case Diagram

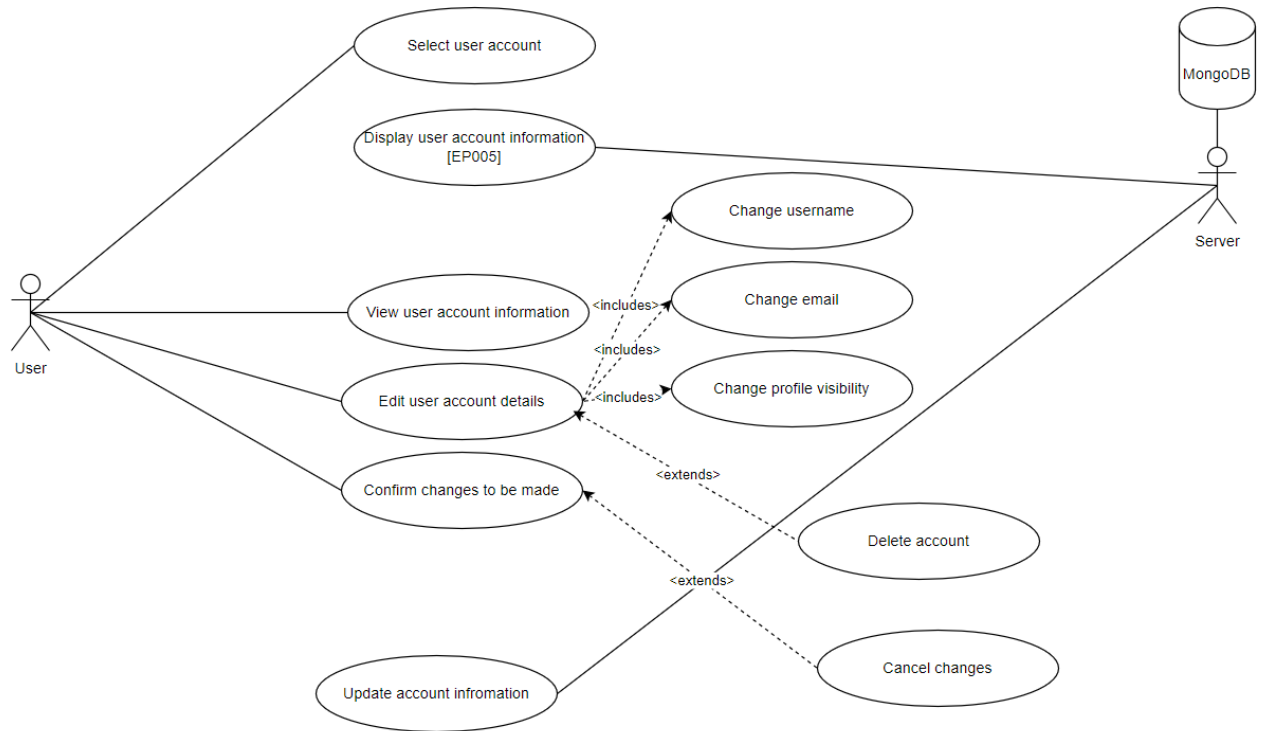


Figure 6 Use Case Diagram for User account [EP005]

Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user selects 'Account'.

Main flow

1. The system loads the User account information page.
2. The system presents the user's account information.
3. The user clicks 'Edit' button.
4. The user changes their username. (A1) (A2) (A3) (A4).
5. The user clicks 'save' (A6) (E1) (E2) (E3).
6. The system saves the update made by the user.
7. The user is redirected back to the User account information page.
8. The system displays the updated information.

Alternate flow

A1 : Change email

1. The user changes their email.
2. Return to step 5 of the main flow.

A2 : Change profile visibility

1. The user changes their profile visibility.
2. Return to step 5 of the main flow.

A3 : Delete account

1. The user clicks 'delete' button.
2. The system displays a prompt to the user to confirm the deletion of their account.
3. The user clicks 'confirm' (A4).
4. The system removes the users account information from the system.
5. User is redirected to the landing page.

A4 : Cancel change

1. The user clicks 'cancel'.
2. The user is redirected back to the Account page.

Exceptional flow

E1 : Username already taken

1. The user enters a username that is already taken.
2. The system displays an error message to the user stating the field must be unique.
3. The user enters a different username.
4. Return to step 5 of the main flow.

E2 : Email already taken

1. The user enters an email that is already taken.
2. The system displays an error message to the user stating the field must be unique.
3. The user enters a different email.
4. Return to step 5 of the main flow.

E3 : Username invalid

1. The user enters a username that doesn't meet the requirements.
2. The system displays an error message to the user stating the requirements needed.
3. The user enters a suitable username.
4. Return to step 5 of the main flow.

Termination

The system presents the user account information page.

Post condition

The system goes into a wait state.

2.1.1.7. Requirement 6: Custom user lists

Description & Priority

This requirement allows users who have made an account to create custom lists to save and store events of their choice. The user will have the option to add collaborators to the lists, allowing them to add events to a shared space with friends. This will enhance the event planning experience for users and add a social aspect to the web application.

Priority: Medium

Use Case

Custom user lists

[EP006]

Scope

The scope of this use case is to allow users to create custom event lists.

Description

This use case describes the functionality to enable users to create custom lists in which they can add events and display on their profile. Collaborative list creation with other users will also be optional.

Use Case Diagram

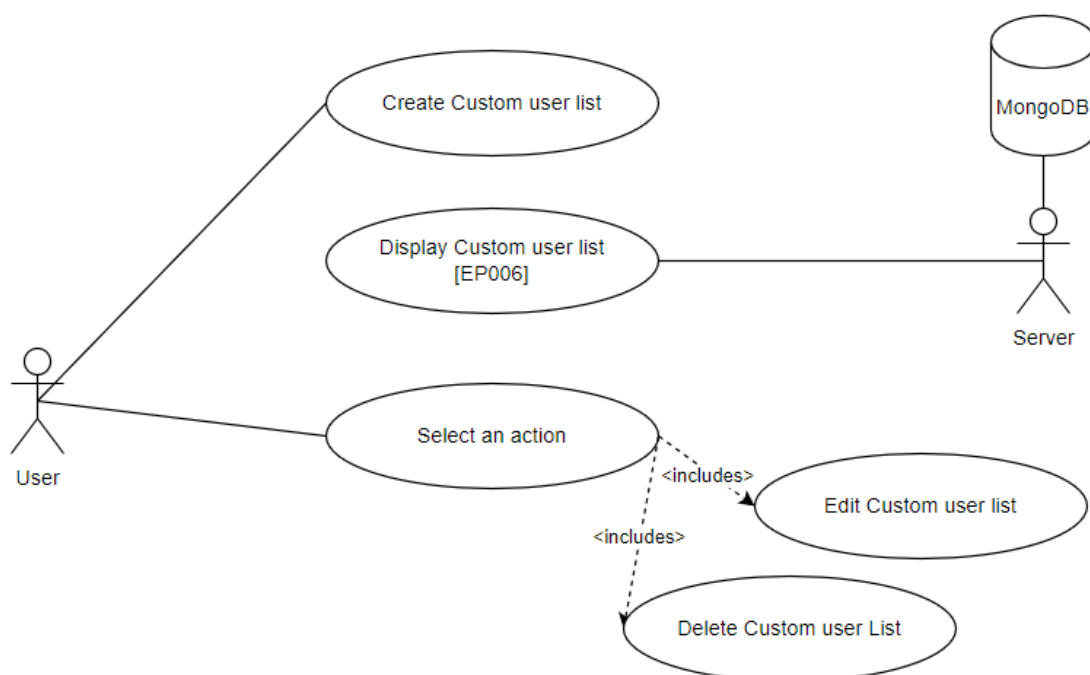


Figure 7 Use Case Diagram for Custom user lists [EP006]

Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user clicks the 'create list' button in the user profile page.

Main flow

1. The system loads the 'create custom list' page.
2. The user enters the details of the list they wish to create.
3. The user clicks 'create' (A1).
4. The system verifies the details entered.
5. The system saves the list details to the database.
6. The system presents the list details to the user (A2).
7. The user clicks 'edit' button.
8. The user changes the list name (A3) (A4).
9. The user clicks 'save' button (A1).
10. The system saves the update made by the user.
11. The page is refreshed.
12. The system displays the updated information

Alternate flow

A1 : Cancel

1. The user clicks 'cancel'.
2. The system returns to the previous page.

A2 : Add event to list

1. Steps are outlined in the main flow of [EP002].

A3 : Change list visibility

1. The user clicks checkbox to change the visibility.
2. Continue from step 9 of main flow.

A1 : Delete list

1. The user clicks 'delete' button.
2. The system prompts the user to confirm the list deletion.
3. The user clicks 'confirm' (A1).
4. The system removes the list from the database.
5. The system redirects to the user profile page.

Termination

The system presents the next page.

Post condition

The system goes into a wait state.

2.1.1.8. Requirement 7: User friends

Description & Priority

Allows the user to add friends enabling them to create collaborative lists together. Once a user has created an account, they can use the search bar to search other user's profiles by username if their profile visibility is set to public. This functionality is necessary to allow users to create collaborative lists.

Priority: Medium

Use Case

User friends

[EP007]

Scope

The scope of this use case is to allow users to add friends by username.

Description

This use case describes the process to add a user as a friend which will enable them to interact with the collaborative list functionality.

Use Case Diagram

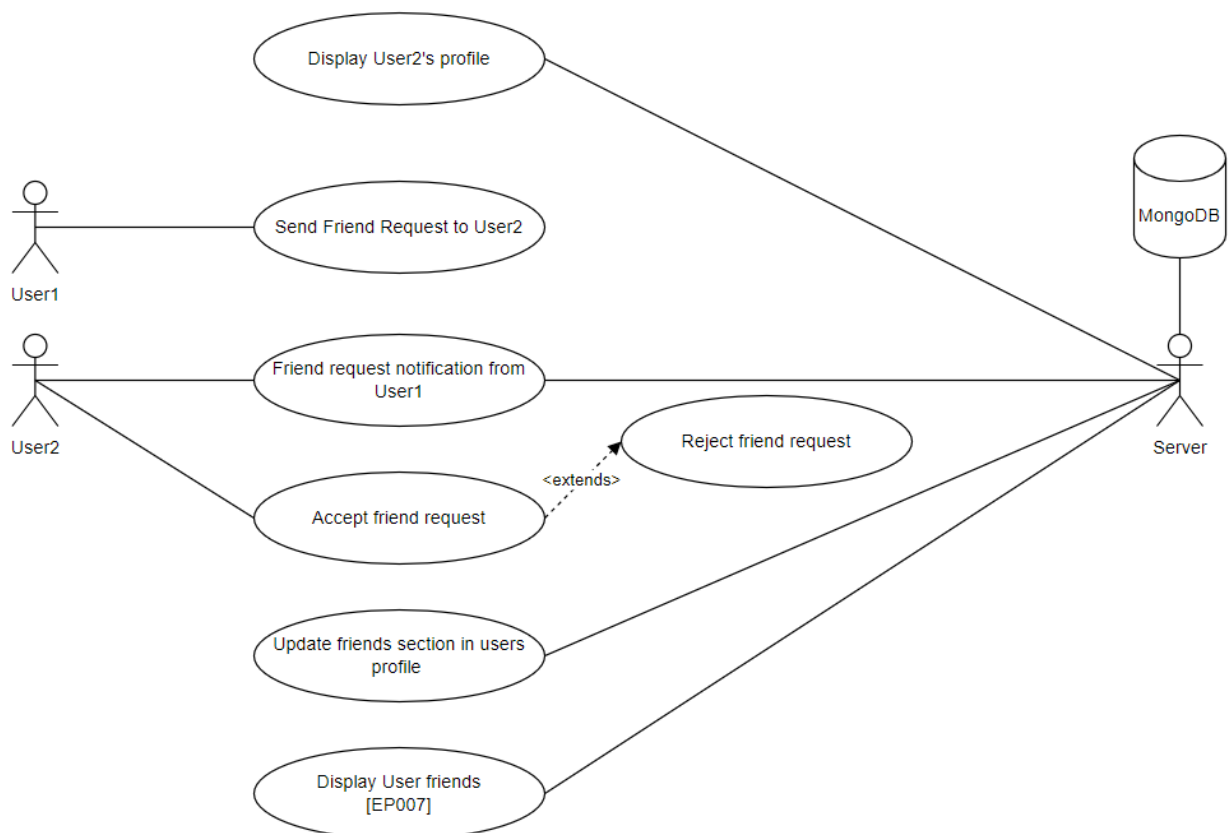


Figure 8 Use Case Diagram for User friends [EP007]

Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user views the profile of another user.

Main flow

1. The system displays a user's profile (user2).
2. The user (user1) clicks the 'send friend request' button (E1) (A4).
3. The system displays a dialog (to user1) saying a friend request has been sent (to user2).
4. The system sends a notification to user2 about the friend request from user1 (A1) (A2).
5. The system goes into a wait state.

Alternate flow

A1 : Accept request

5. User2 accepts the friend request (A2).
6. The system notifies user1 that their request has been accepted.
7. The system updates friends section in both users' profiles.
8. Return to step 5 of the main flow.

A2: Reject request

5. User2 denies the friend request from User1(A2).
6. The system removes user1's pending request.
7. Return to step 5 of the main flow.

A3: Remove friend

1. The user clicks the 'remove friend' button.
2. The system removes both user's details from each user's account.
3. Return to step 1 of main flow.

A4: Don't add friend

1. The user doesn't send a friend request.
2. Return to step 5 of main flow.

Exceptional flow

E1 : Users are already friends

1. The system presents the 'remove friend' button.
2. The user returns to the previous page (A3).
3. Return to step 5 of the main flow.

Termination

The system presents the next web page.

Post condition

The system goes into a wait state.

2.1.1.9. Requirement 8: Collaborative list

Description & Priority

This feature allows a user to add collaborators to their lists enabling both users to have full access to the shared list and its details. From here users will have the ability to observe updates as they interact with lists such as adding events and editing the details. This will add a social aspect to the web application and enhance the event planning and going experience.

Priority: Medium

Use Case

Collaborative list

[EP008]

Scope

The scope of this use case is to allow users to add friends to a collaborative list.

Description

This use case describes the functionality enabling users to add friends and utilize a shared collaborative list to edit, and add events.

Use Case Diagram

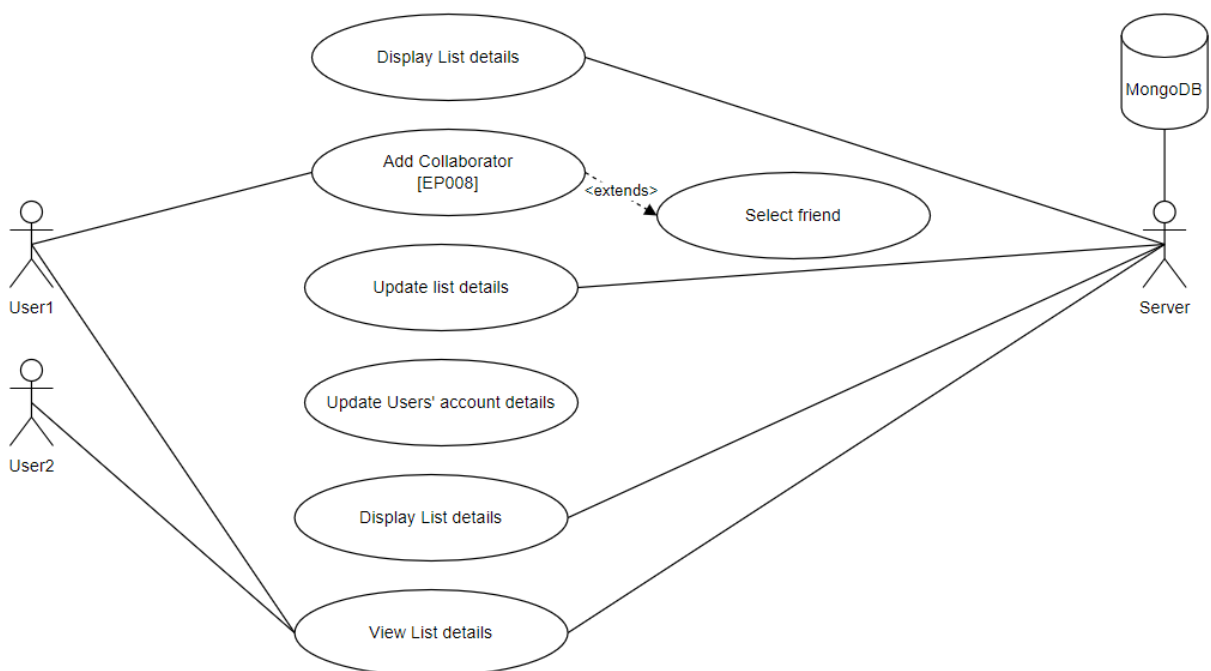


Figure 9 Use Case Diagram for Collaborative list [EP008]

Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

User1 and User2 are friends and listed in friends' section of profile.

Activation

This use case starts when a user opens a list.

Main flow

1. The system displays information about selected list.
2. The user clicks 'add collaborator' button.
3. The system displays the list of user' friends (E1).
4. The user clicks on a friend to add as a collaborator.
5. The system updates the list's collaborators field to include the selected user.
6. The system updates the selected user's lists field to include the added list.
7. The system presents the updated list details.

Exceptional flow

E1 : No friends found

4. The system presents a message stating there are no friends found.
5. The system redirects to the selected event information page.

Termination

The system presents the next web page.

Post condition

The system goes into a wait state.

2.1.2. Data Requirements

The web application will require users to enter a username, email address, and password in order to create an account. The app will handle this data with considerations to both security and privacy of the user. Guidelines of the GDPR will be taken into consideration and users' data will not undergo any further processing by parties outside of the scope of this project. The web app will be secure to best protect user's data, implementing IP whitelisting on the deployed MongoDB database and utilising the Bcrypt library for password hashing and salting in the Express server.

2.1.3. User Requirements

Users will be required to have some knowledge of using technology and connecting to the internet when using web applications. This web application looks to be as inclusive as possible by providing a user friendly GUI.

2.1.4. Security Requirements

User's data will be stored in a secure MongoDB database which will incorporate security measures to best protect the user's data. this includes IP whitelisting, password hashing and salting. Input validation measures will be in place to prevent cyberattacks and security tests will be carried out on the web application to identify vulnerabilities. Users will need the correct email and password to access their account. To ensure passwords are secure, they will need to meet the minimum requirements for a strong password (upper and lowercase letters, numbers, special symbols, etc). If a security breach does occur, measures will be put in place to handle this issue and users will be notified.

2.1.5. Usability Requirements

The web application's graphical user interface (GUI) will incorporate a consistent design throughout all pages within the system with the use of reusable React components. An established layout, colour palette, and fonts (outlined in the design & architecture section) will be used throughout the application to ensure consistency. The GUI looks to be inclusive to all types of users. This will be achieved by ensuring each functionality is usable to users with any/little technical experience, each component will be accessible through the tab button, and alternative text will be provided where necessary.

2.1.6. Maintainability Requirements

Using frameworks such as Express and React will allow this project to maintain good version control over imported libraries and dependencies throughout development and in the future. Npm allows libraries to be easily updates and is well documented to highlight any compatibility issues which may arise. The project utilises GitLab as version control and the code will be commented throughout the application to ensure that it is easy to interpret by other developers.

2.1.7. Robustness Requirements

The web app will perform as expected with minimal to no errors. For example, if the user enters an input which is not the expected input, an error message will display specifying the issue and allowing them to try again.

2.1.8. Availability Requirements

The web application will incorporate a responsive GUI allowing it to be accessed and readable on any size of device. The web app looks to be accessible on a wide range of browsers and will incorporate measures to be compatible with major browsers.

2.1.9. Performance Requirements

The web application should run quickly and efficiently when carrying out requests from the user and providing a response. When a user interacts with the web app, requests between the backend server and the frontend server should be fulfilled quickly with error handling to account for unexpected inputs.

2.2. Design & Architecture

This full stack web application implements the MVC architecture to structure the backend server.

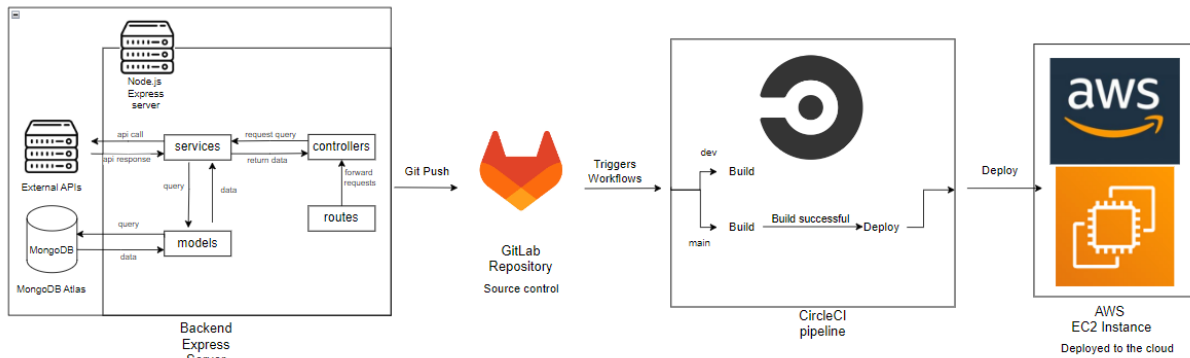


Figure 10 Architectural Diagram for Express backend server

Outlined above is the full structure of the Express backend server. When a code change is pushed from the local Git repository to the remote GitLab repository a workflow is triggered to start the CircleCI pipeline. The dev branch will carry out a build job of the project. Once the main branch is merged this will also undergo the build job, after the build is successful the code changes will automatically be deployed to an EC2 AWS instance. Here the pm2 library will clone the repository, install the dependencies, and deploy the server to the cloud.

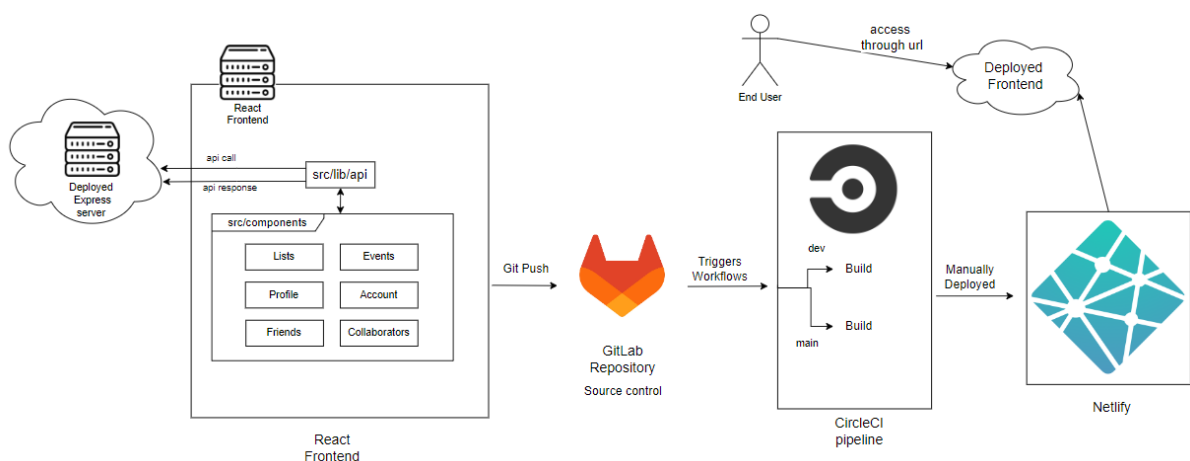


Figure 11 Architectural Diagram for React frontend

Here the structure of the React frontend is illustrated. Once the backend Express server is deployed and available through the URL the React src/lib/api Axios middleware is able to communicate back and forth with the express endpoints. The React components use this

middleware to call the appropriate functions depending on the user's interactions with the GUI. Similarly, once a code change is pushed to the GitLab repository, the CircleCI workflows are triggered. Unlike the Express backend server, there is no automated deployment in this scenario. Both dev and main branches go through the build jobs and then the frontend can be deployed manually using Netlify. From here, the user can access the deployed application in the browser.

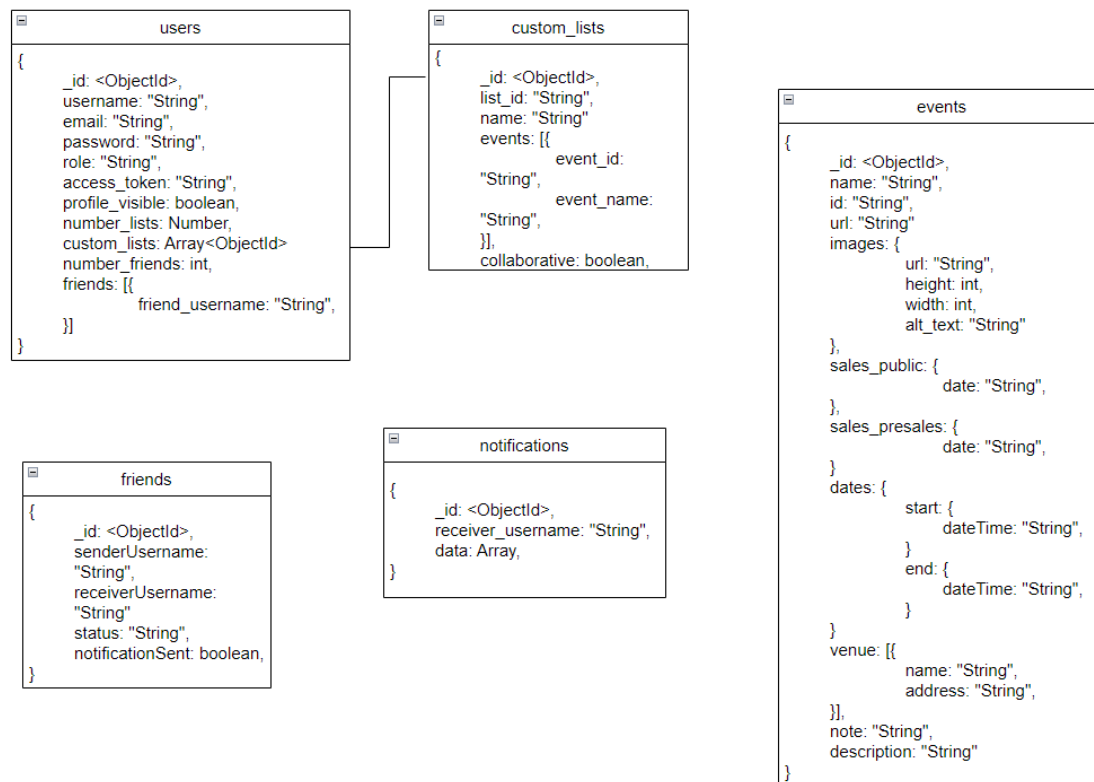


Figure 12 MongoDB Collections structure

Outlined in the diagram above is the structure of the collections used in the projects database. The MongoDB database collection definitions is located in the models folder of the Express backend server. These collections are to store information surrounding events, users, custom lists, friend requests and notifications. I chose MongoDB as the database for this project as it can be easily integrated with the backend server using the Mongoose library. My level of experience with MongoDB is less than that of MySQL, however during my work placement I was able to familiarise myself with the key features that made this the favourable choice. Originally, I had planned to primarily use embedded sub-documents for fields such as custom_lists and friends, however as development of the functional requirements progressed, I had to adapt the structure to focus more on a normalized data structure. By adding an array of usernames for the friends field in the users collection, information on each user can be easily retrieved because each username in the database must be unique.

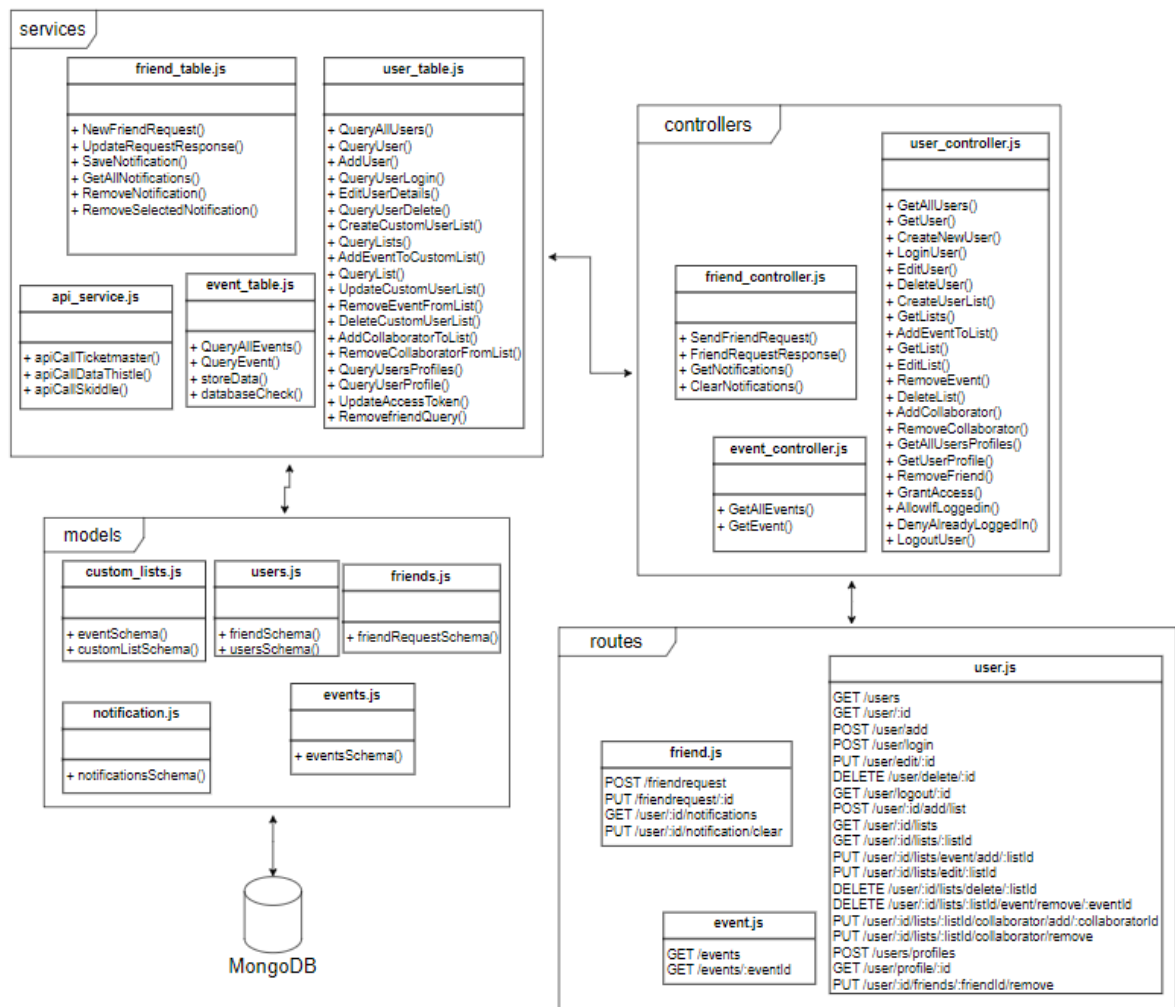


Figure 13 Class Diagram for Express backend server

Illustrated above is a more in depth look at the Express backend server, including its file structure and the ways in which the MVC folders interact with each other.

2.3. Implementation

Because this is a full stack web application with both the frontend and backend deployed, this section will focus on the flow of events which take place as the user interacts with the web application. The features that will be outlined in detail are two of the highest complexity functionalities that this project implements. All other functionalities follow the same sequence of events when carrying out their tasks.

View events [EP001]

Once the Express server is running and deployed, the server will make several API calls to the external APIs and populate the events collection in the MongoDB database.

The following code snippets outline the functions used to call the APIs, handle the responses, and store the data in the database (each piece of code is explained in the comments:

server/models/events.js

This is the initial collection definition in which the events data will be stored.

```

1  import mongoose from 'mongoose';
2  const { Schema } = mongoose;
3
4  //schema maps to a collection
5  const eventsSchema = new Schema({
6    name: String,
7    id: String,
8    url: String,
9    images: Schema.Types.Mixed,
10   sales_public: Schema.Types.Mixed,
11   sales_presales: Schema.Types.Mixed,
12   dates: Schema.Types.Mixed,
13   venue: Array,
14   note: String,
15   description: Schema.Types.Mixed,
16 });
17
18 //create a new model named 'events' using the above schema
19 const events = mongoose.model('events', eventsSchema);
20
21 export default events;
22
23

```

Figure 14 Code snippet 1: events model

server/server.js.

Before calling the APIs, the server runs a function to check if the collection is already populated with event data. If it is the server does not call the APIs. This was added to reduce the number of API calls being made during development. On every code change the server would restart and the APIs would be called. Adding this check reduced the number of unnecessary calls being made and allowed development to be carried out more efficiently. To securely store the API keys the package dotevn was used. The API keys are saved in the environmental variables in the CircleCI project and are called into the project during the deployment job on the main branch.

```

31
32 //function to handle the data received from the api call
33 const handleData = async () => {
34   try {
35     //check if 'events' database collection is populated before making an api call
36     //this reduces the number of api calls when server refreshes
37     let check = await databaseCheck();
38     if (check) {
39       console.log('Events collection populated. Skip API call and data handling.');

```

Figure 15 Code snippet 2: api call

The URLs are stored in arrays to allow the Promise.all function to be run on each URL, mapping each response to a variable. Once all calls have completed, the variables from each API response are combined and stored in the events collection.

```

59
60 //call each api url and map the responses
61 const apiResponsesTicketmaster = await Promise.all(apiUrlsTicketmaster.map(url => apiCallTicketmaster(url)));
62 const apiResponsesDatathistle = await Promise.all(apiUrlsDatathistle.map(url => apiCallDatathistle(url)));
63 const apiResponsesSkiddle = await Promise.all(apiUrlsSkiddle.map(url => apiCallSkiddle(url)));
64
65 //concatenate the responses
66 const tmData = apiResponsesDatathistle.reduce((acc, data) => acc.concat(data), []);
67 const dtData = apiResponsesTicketmaster.reduce((acc, data) => acc.concat(data), []);
68 const skData = apiResponsesSkiddle.reduce((acc, data) => acc.concat(data), []);
69
70 //combine and concatenate the responses from each api
71 const combinedData = [
72   ...tmData,
73   ...dtData,
74   ...skData
75 ];
76
77 //store the api data in the database
78 await storeData(combinedData);
79
80 } catch (error) {
81   console.error('Error handling data: ', error.message);
82 }
83 };
84
85 handleData();
86

```

Figure 16 Code snippet 2: api call cont.

server/services/event_table.js

```

66
67 //checks if the database collection 'events' is populated
68 export const databaseCheck = async () => {
69   const existingData = await events.find();
70   let populated = false;
71   if (existingData.length > 0) {
72     populated = true;
73   }
74   else {
75     populated = false;
76   }
77   return populated;
78 };
79

```

Figure 17 Code snippet 3: database check

server/services/api_service.js

Because each API response is different the responses are processed individually to ensure the data being stored in the events collection consistent. The Ticketmaster response undergoes the least amount of processing in comparison to the DataThistle and Skiddle responses.

```

77
78 //function to make the api call to skiddle
79 export const apiCallSkiddle = async (url) => {
80   try {
81     //call api
82     const response = await axios.get(url);
83     const jsonData = response.data;
84     //console.log("Response body:", jsonData);
85
86     const formattedData = jsonData.results.map(event => ({
87       name: event.eventname,
88       id: event.id,
89       url: event.ticketUrl ? event.ticketUrl : null, //could to event.link to go to skiddle event page
90       images: [{
91         url: event.largeimageurl ? event.largeimageurl : null,
92       }],
93       sales: {
94         public: null, //not applicable with this API
95         presales: null, //not applicable with this API
96       },
97       dates: {
98         start: {
99           dateTime: event.startdate
100         },
101         end: {
102           dateTime: event.enddate
103         }
104       },
105       _embedded: {
106         venues: [{
107           name: event.venue.name,
108           address: event.venue.address,
109         }],
110       },
111       pleaseNote: null, //not applicable with this API
112       description: event.description,
113     }));
114     //console.log('Formatted Data:', formattedData);
115     return formattedData;
116   } catch (error) {
117     throw new Error(`Error with Skiddle API call: ${error.message}`)
118   }
119 }

```

Figure 18 Code snippet 4: api response processing

server/services/event_table.js

In this function the combined and processed API responses are lastly checked for duplicate event names in order to prevent two of the same event from being stored. Here the data array is mapped through and each unique event name is saved while adding the event object to an array. If an event with the same name is found it is not added to the array.

```

25
26 //function to store the api data in the database
27 export const storeData = async (data) => {
28   //console.log("data being stored:", data)
29   try {
30     //prevent events with the same name being stored
31     const uniqueEventData = [];
32     const uniqueNames = new Set();
33
34     //map api response data to name checks
35     data.forEach(event => {
36       if (!uniqueNames.has(event.name)) {
37         uniqueNames.add(event.name);
38         uniqueEventData.push(event);
39       }
40     });
41
42     //map the filtered api response data to events collection
43     const eventsData = uniqueEventData.map(event => {
44       return {
45         name: event.name,
46         id: event.id,
47         url: event.url,
48         images: event.images[0] ? event.images[0] : null,
49         sales_public: event.sales.public,
50         sales_presales: event.sales.presales,
51         dates: event.dates,
52         venue: event._embedded.venues[0],
53         note: event.pleaseNote,
54         description: event.description ? event.description : null,
55       };
56     });
57
58     //console.log("store the following data:", eventsData)
59     //create method to insert all mapped data
60     await events.create(eventsData);
61     console.log(eventsData);
62   } catch (error) {
63     throw new Error(`Error storing data: ${error.message}`);
64   }
65 };

```

Figure 19 Code snippet 5: store event data

When the frontend React project connects to the Express backend server the GET /events endpoint is called and the response is displayed to the user in the GUI.

The functions which are called are outlined below in sequential order.

react-frontend/src/components/ViewEvents.jsx

```

18
19 //fetch the list of events when the component mounts
20 useEffect(() => {
21   fetchEvents();
22 }, []);
23
24 //function to fetch the list of eventa from the API
25 const fetchEvents = async () => {
26   try {
27     const response = await getEvents();
28     //console.log(response.data)
29     setEvents(response.data);
30   } catch (error) {
31     console.error(error);
32   }
33 };

```

Figure 20 Code snippet 6: get events for ViewEvents component

react-frontend/src/lib/api/frontend.js

The axios middleware configured for all requests being sent to the Express backend server.

```
1 import applyCaseMiddleware from "axios-case-converter";
2 import axios from "axios";
3
4 const options = {
5   ignoreHeaders: true,
6 };
7
8 const frontend = applyCaseMiddleware(
9   axios.create({
10     baseURL: "https://3.253.123.101:3333/",
11     headers: {
12       'Content-Type': 'application/json',
13     },
14   }),
15   options
16 );
17
18 export default frontend;
```

Figure 21 Code snippet 7: axios frontend middleware

react-frontend/src/lib/api/backend-server.js

```
1 import frontend from './frontend';
2 import { getCookie } from '../cookieConfig';
3
4 //gets events from server
5 export const getEvents = () => {
6   return frontend.get('/events');
7 };
8
```

Figure 22 Code snippet 8: getEvents function

server/routes/event.js

The previous function sends a GET request to the /events endpoint

```
8
9 router.get('/events', GetAllEvents);
10
```

Figure 23 Code snippet 9: /events route

server/controllers/event_controller.js

The controller calls a service to perform the query to the database and returns its data which is sent to the frontend through a JSON response.

```
5
6 //GET /events
7 export const GetAllEvents = async (req, res, next) => {
8   try{
9     //query database in event_table.js
10    const eventsList = await QueryAllEvents();
11
12    // return list of events in json
13    return res.json(eventsList);
14  } catch (error) {
15    console.log(error);
16    return res.status(500).json({ error: "Backend server error. Location: event_controller, GetAllEvents function" });
17  }
18 }
```

Figure 24 Code snippet 10: GetAllEvents function

server/services/event_table.js

```
5
6 export const QueryAllEvents = async (req, res) => {
7   const allEvents = await mongoose.model('events').find().exec();
8   return allEvents;
9 }
```

Figure 25 Code snippet 11: QueryAllEvents function

Once the React frontend receives this JSON response it is presented to the user through the GUI highlighted in the next section.

User friends [EP007]

Once a user has created an account, they have full access to all functionalities the web application has to offer. To add a friend User1 begins the process by clicking the SearchUsers button located in the header. Here they can browse through the profiles of all users who have their account set to public. The function to display the profiles differs than the one used to view account details. Sensitive fields such as password, and access_token is not returned to the frontend for security reasons. When User1 clicks on the profile of User2 they can click the 'send friend request' button to begin the process of adding them as a friend. From here, if User2 is also logged in they will receive a notification in real time through the socket.io package. If User2 is not logged in the request will be stored in the notifications collection and be presented the next time they login. They can choose to accept or reject the request and the sender (User1) will be notified of their response. If accepted, both Users' profiles and account details will be updated, listing the newly added user in their friends list. Outlined below are the functions involved in the process of sending a friend request.

react-frontend/src/components/SendFriendRequest.jsx

```
4
5 function SendFriendRequest({ receiver_username }) {
6
7   const handleClick = async () => {
8     try {
9       console.log("handle click function called")
10      const username = getCookie('username')
11      const accessToken = getCookie('access_token')
12
13      console.log(receiver_username)
14      if (!!receiver_username) {
15        const friendRequestData = { senderUsername: username, receiverUsername: receiver_username }
16        const response = await sendFriendRequest(username, accessToken, friendRequestData)
17        // console.log("response", response)
18        if (response.status == 200) {
19          if (response.data.success == true) {
20            console.log(response.data.data)
21            window.alert(`a friend request was sent to ${receiver_username}`)
22          }
23        }
24      }
25    } catch (error) {
26      console.log(error)
27    }
28  }
29 }
```

Figure 26 Code snippet 12: SendFriendRequest component

react-frontend/src/lib/api/backend-server.js

```
304
305 export const sendFriendRequest = (username, accessToken, friendRequestData) => {
306   console.log("sendFriendRequest called");
307   try {
308     return frontend.post(`/friendrequest`, friendRequestData, {headers: {
309       'x-access-token': accessToken,
310     }});
311   } catch (error) {
312     console.error('Error sending friend request:', error);
313     throw error;
314   }
315 }
```

Figure 27 Code snippet 13: sendFriendRequest call to server

server/controllers/friend_controller.js

```
17
18 //POST /friendrequest
19 export const SendFriendRequest = async (req, res, next) => {
20   try{
21     console.log("SendFriendRequest function called")
22     // console.log(req.body)
23
24     const senderUsername = req.body.sender_username
25     const receiverUsername = req.body.receiver_username
26     const friendRequest = await NewFriendRequest(senderUsername, receiverUsername)
27     const notificationData = [
28       friendRequest.data.senderUsername,
29       friendRequest.data.receiverUsername,
30       friendRequest.data.status,
31       friendRequest.data.notificationSent,
32       friendRequest.data._id,
33       friendRequest.data.__v
34     ];
35     // console.log(notificationData)
36
37     const io = req.io;
38     // io.emit('friendRequestReceived', senderUsername)
39     // io.to(receiverUsername).emit('friendRequestReceived', notificationData);
40
41     //check to find if the receiver of the friend request is online to send notification through the socket
42     const receiverSocket = connectedUsers.get(receiverUsername);
43     if (receiverSocket) {
44       //receiver is online
45       receiverSocket.emit('friendRequestReceived', notificationData);
46     } else {
47       //receiver is not online save notification to database
48       await SaveNotification(receiverUsername, notificationData);
49     }
50
51     return res.json(friendRequest)
52   } catch (error) {
53     console.log(error);
54     return res.status(500).json({ error: "Backend server error. Location: friend_controller, SendFriendRequest function" });
55   }
56 }
57 }
```

Figure 28 Code snippet 14: SendFriendRequest in friend_controller.js

server/services/friend_table.js

```
7
8 export const NewFriendRequest = async (senderUsername, receiverUsername) => {
9   console.log("NewFriendRequest query called")
10   let result = "";
11   try {
12     const newRequest = new friendRequests({ senderUsername, receiverUsername, status: "pending", notificationSent: true })
13     await newRequest.save();
14     // console.log("New Request created");
15
16     result = { success: true, message: "New friend request created", data: newRequest }
17
18     return result
19   } catch (error) {
20     console.error("Error creating new friend request", error.message);
21     return { success: false, message: "Error creating new friend request", error: error.message };
22   }
23 }
24 }
```

Figure 29 Code snippet 15: NewFriendRequest query

2.4. Graphical User Interface (GUI)

Before styling the components in the frontend React project I created a logo, selected a colour palette, and fonts which is used throughout the pages in the frontend GUI. This will help establish a consistent design throughout the project and create an identifiable brand for the Event Point project.

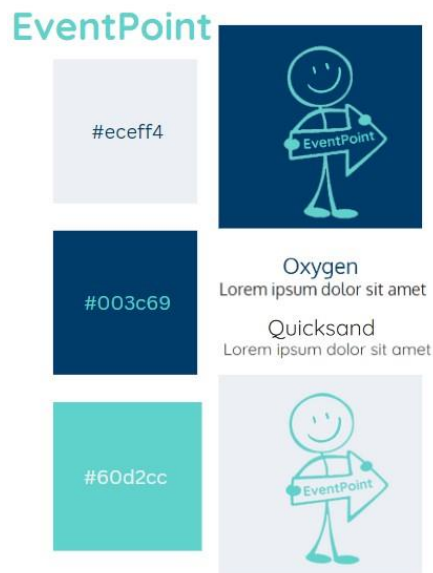


Figure 30: Event Point design

This is the home page of the frontend server which is the landing page presented to users when they first visit the site. View events [EP001]. It presents a list of events in the main content section of the page. From here users will have the option to view more details about a specific event or refine their search by using the search bar and filters above.

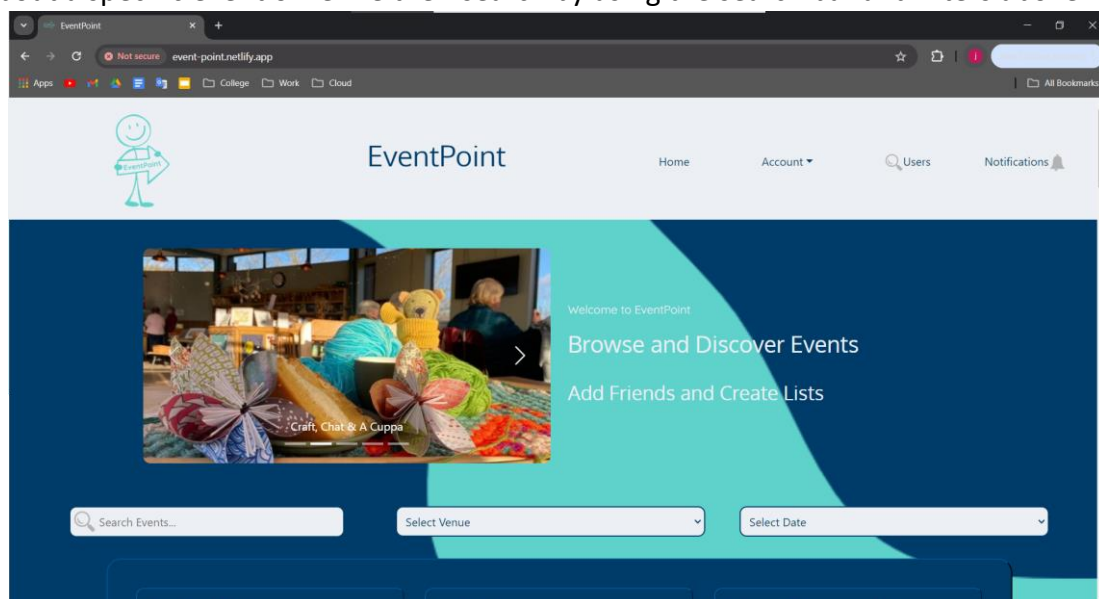


Figure 31 GUI 1: ViewEvents [EP001]

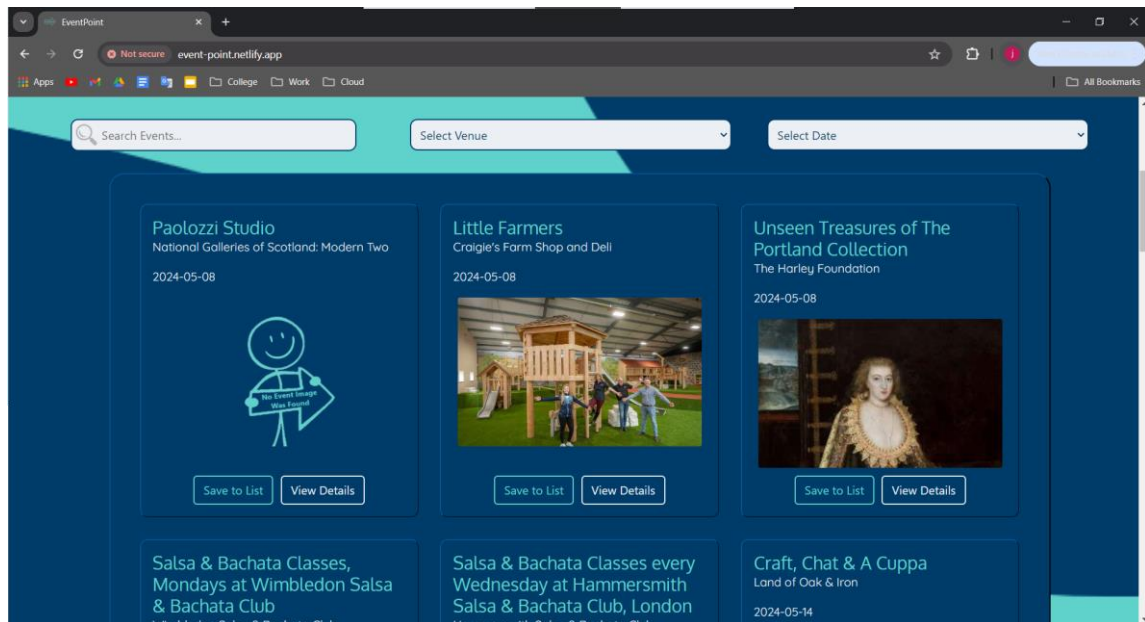


Figure 32 GUI 1: View Events[EP001] cont.

User friends [EP008]

This page allows users to view profiles of other users who have created an account and are logged in. From here they select a user and send them a friend request.

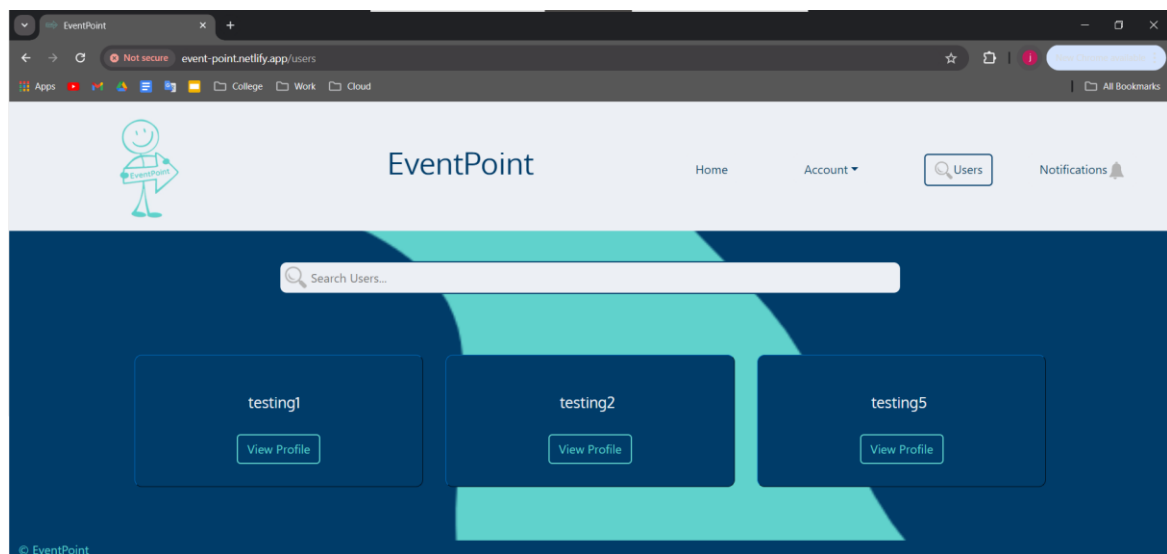


Figure 33 GUI 2: SearchUsers

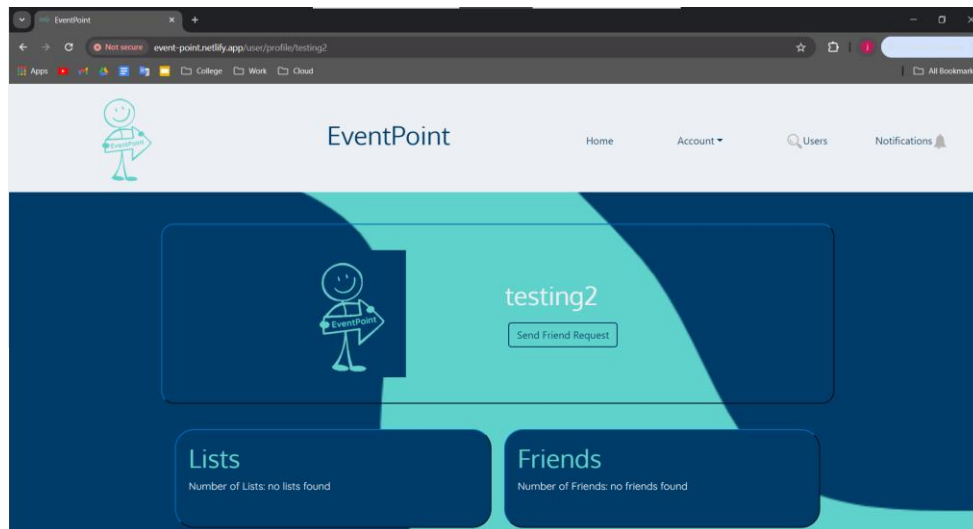


Figure 34 GUI 3: ViewProfile (User profile [EP004])

Additional GUI pages:

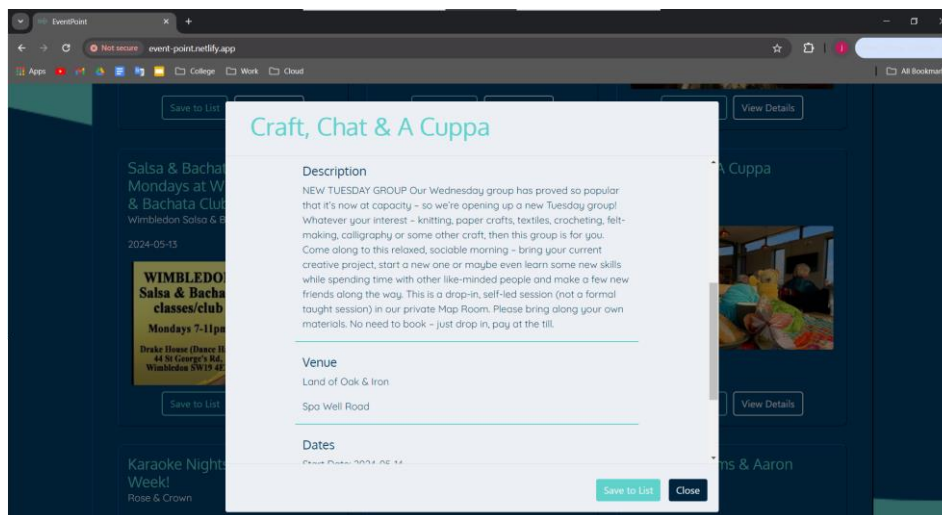


Figure 35 GUI 4: EventInformation [EP002]

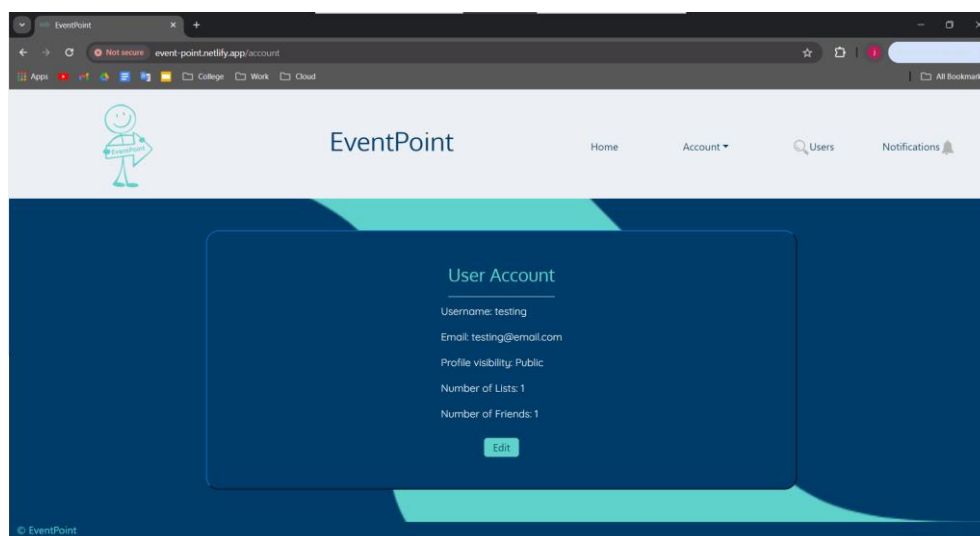


Figure 36 GUI 5: UserAccount [EP005]

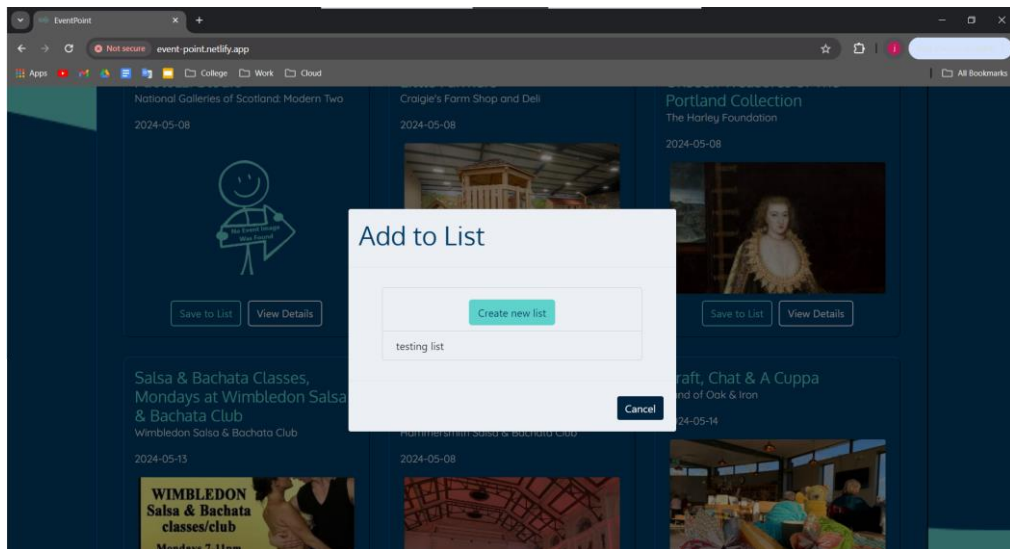


Figure 37 GUI 6: AddToList

2.5. Testing

The tests carried out on this project include Component and Integration testing performed on the locally ran version of the React frontend during the development of this project. Illustrated below is the full directory containing each functionalities test cases and a brief overview of the test cases within the AccountComponents.spec.ts file.

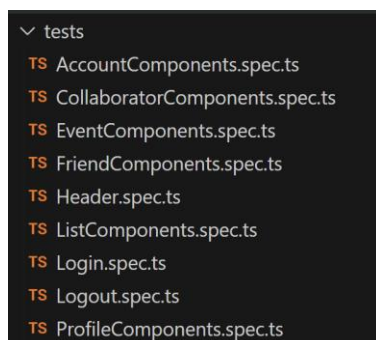


Figure 38 Tests directory

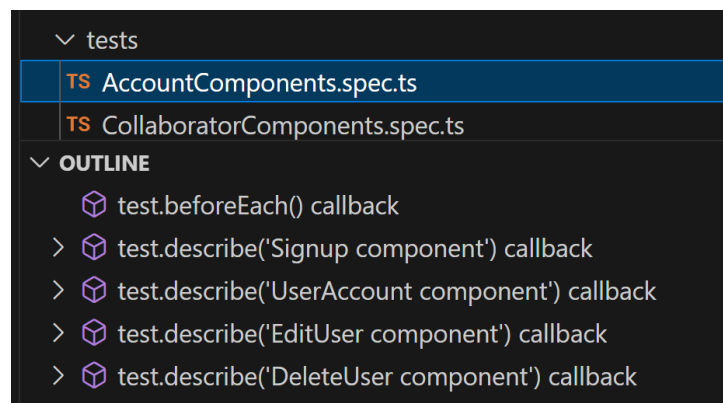


Figure 39 AccountComponents.spec.ts overview

The Playwright extension and library was installed to the React frontend. Originally I had planned to implement libraries like Jest and Mocha to perform these tests however due to the configuration of my project I faced issues integrating Jest and Mocha with my components. This was due to errors involving the ECMAScript Modules (ESM) syntax, which uses import/export statements.

```

PS C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend> npm test
> react-frontend@0.1.0 test
> jest

console.error
Error: Uncaught [ReferenceError: React is not defined]
    at reportException (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:66:
24)
    at innerInvokeEventListeners (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:66:
353:9)
    at invokeEventListeners (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:286:
3)
    at HTMLUnknownElementImpl._dispatch (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:239:9)
    at HTMLUnknownElementImpl.dispatchEvent (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:104:17)
    at HTMLUnknownElement.dispatchEvent (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\jest-runtime\script-errors.js:241:34)
    at Object.invokeGuardedCallbackDev (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\react-dom\cjs\react-dom.development.js:4213:
16)
    at invokeGuardedCallback (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\react-dom\cjs\react-dom.development.js:4277:31)
    at beginWork$1 (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\react-dom\cjs\react-dom.development.js:27451:7)
    at performUnitOfWork (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\react-dom\cjs\react-dom.development.js:26568:12)
    at workLoop$1 (C:\Users\jader\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend\node_modules\react-dom\cjs\react-dom.development.js:26466:15)

```

Figure 40 Jest console error

After researching alternative testing libraries and installing the Playwright extension I no longer faced any of the issues encountered when trying to implement Jest and Mocha. The Playwright library utilizes browser contexts to allow the UI of the application to be interreacted with during the test cases. It also offers tools to select which browsers the application will be tested on such as Chromium, Webkit, and Firefox. These test cases can be run directly in the Command Line using the following commands in the root directory of the project:

```

npx playwright test --workers=1 --project=chromium
npx playwright test --workers=1 --project=webkit
npx playwright test --workers=1 --project=firefox

```

The tests for each browser are run one at a time because of the speed at which the test cases are run. This minimises the chances of a test case failing. If a failure does occur the individual test within the component can be rerun easily. The numbers of workers are set to 1 due to Playwrights parallel execution of tests. This implements a more linear structure for each component test and allows them to be run one at a time, further preventing the likelihood of a test failing and also being more similar to how a user interacts with the project in a real world environment. These runtime issues were discovered through multiple reruns of the full tests folder. Outlined below are the results from an early run of tests.

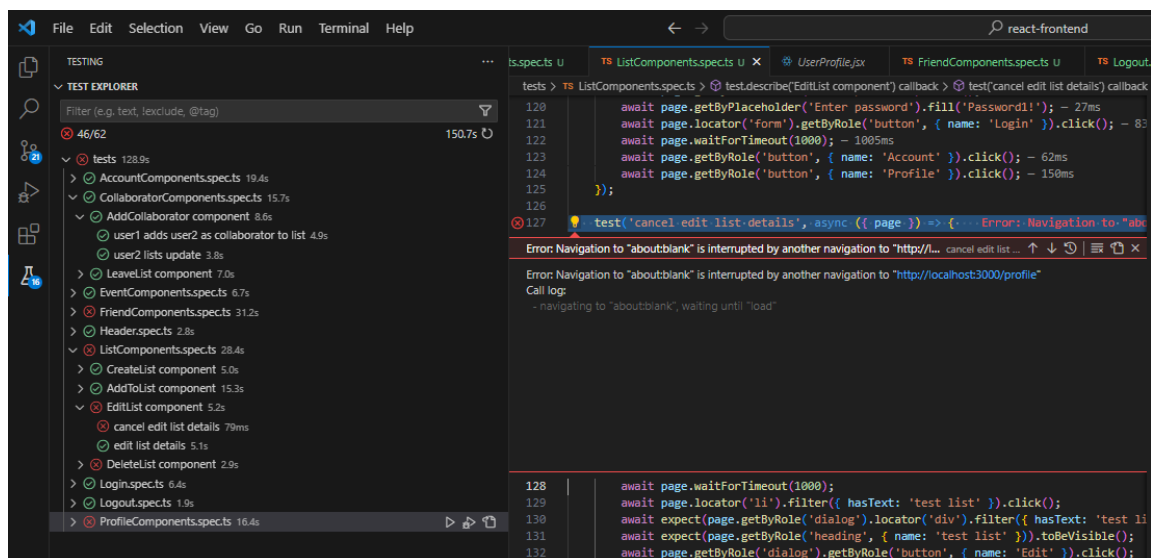


Figure 41 Test results 1

After adjusting the execution speeds by using functions such as `.waitForTimeout(1000)` I was able to successfully run testing on each component with no failures. The playwright library also generates a report of the results which can be seen below.

```
PS C:\Users\jadel\OneDrive - National College of Ireland\Documents\college\year 4\Final Project\event-point\react-frontend> npx playwright test --workers=1 --project=chromium
Running 60 tests using 1 worker
Slow test file: [chromium] > ListComponents.spec.ts (47.2s)
Slow test file: [chromium] > AccountComponents.spec.ts (38.5s)
Slow test file: [chromium] > FriendComponents.spec.ts (34.6s)
Slow test file: [chromium] > CollaboratorComponents.spec.ts (21.8s)
Consider splitting slow test files to speed up parallel execution
60 passed (3.3m)

To open last HTML report run:
npx playwright show-report
```

Figure 42 Test results 2

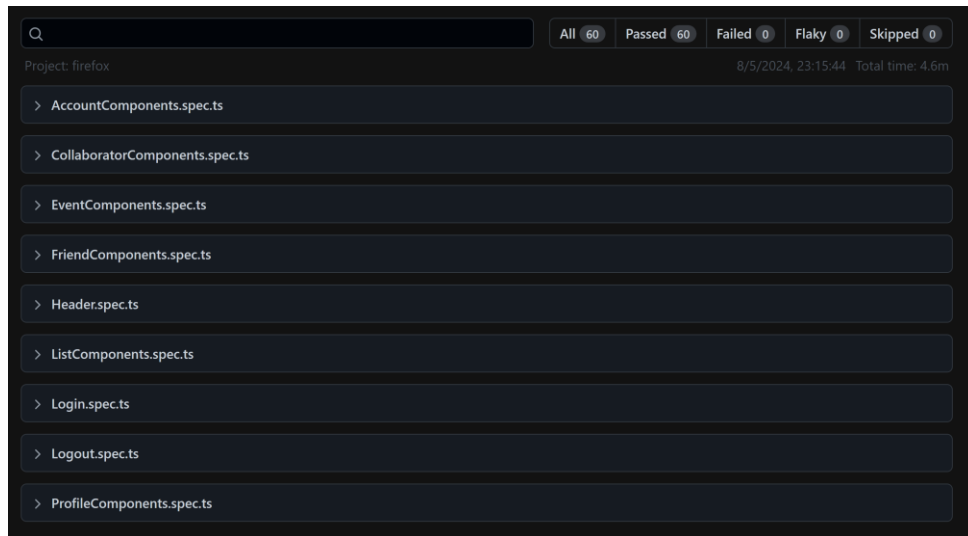


Figure 43 Playwright Test report

| | |
|--|------|
| EventComponents.spec.ts | |
| ✓ ViewEvents component › renders landing page | 2.5s |
| EventComponents.spec.ts:11 | |
| ✓ ViewEvents component › filters events by search input | 5.2s |
| EventComponents.spec.ts:15 | |
| ✓ ViewEvents component › filters events by venue dropdown | 2.7s |
| EventComponents.spec.ts:24 | |
| ✓ ViewEvents component › filters events by date dropdown | 2.4s |
| EventComponents.spec.ts:31 | |
| ✓ ViewEvents component › shows no events when no events match the search filters | 1.8s |
| EventComponents.spec.ts:38 | |
| ✓ EventInformation component › renders event information modal when events are shown | 2.1s |
| EventComponents.spec.ts:47 | |

Figure 44 Playwright Test report for EventsComponents.spec.ts

Using Lighthouse in the developers window of Chrome, metrics such as Performance, Accessibility, Best Practices and SEO are tested for both the locally ran and deployed React frontend. Testing in both environments allows the results to be compared and areas of concern to be easily identified.

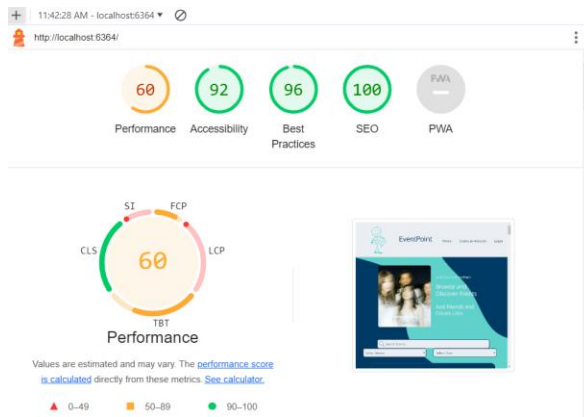


Figure 45 Lighthouse report - locally ran

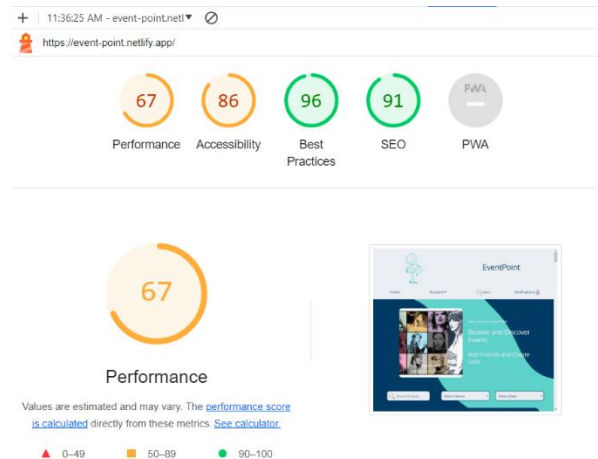


Figure 46 Lighthouse report - deployed application

3.0 Conclusions

This project shows many advantages while also recognising its limitations. There is good potential for the project to evolve and develop in the future, which is outlined in more detail in the next section of the report.

An advantage of this project is the convenience it provides to users. Having events from multiple different ticketing vendors in the one location greatly reduces the number of sites a user must browse to view events, it also reduces the chances of a user missing an event due to the fact that they have not visited a specific site.

Another advantage of this site is being able to access the events listings and details without having to create an account. This is an optional feature giving users the flexibility and convenience of not having to create an account. This makes it accessible to all, without users being obligated to sign up to browse the aggregated events listings. However, a disadvantage to this is the restrictions in additional features which are imposed on those who do not create an account. This is an unavoidable issue as the restricted features require users to be authenticated for security and functional reasons. These features include the Friends, Custom Lists, and Collaborative Lists. This will enable the project to run more effectively with minimal issues and complications.

The project relies on the data obtained through API calls. This means the limitations of APIs are correlated with the limitations of this project. Due to the minimal number of APIs available regarding events in Ireland this will impose limitations on the events being displayed to users looking for events in Ireland.

To address this, the project has expanded its location of event listings to the UK and Ireland. This greatly increases the range and variety of events available for users to browse while also allowing users to discover events they may not have considered if the location was more focused to Ireland.

Additionally, due to the SSL certificate associated with the Express backend used during deployment, the number of browsers which can render the full web application and its

features is limited and works on a case by case basis. As of writing this report the full application is available on Chrome and Firefox.

Lastly, the collaborative lists feature is a good way to allow users to interact with friends and engage with the web application more. The social aspect of attending events is a major factor in many people's decisions when planning and attending events. By having users gain that social interaction through the web application enables them to enhance their event experiences.

4.0 Further Development or Research

As previously mentioned, there is good potential for future development in this project. Functional Requirements such as Book Ticket and Secure Ticket Transfer has since been removed from the Software Requirements Specification of this report. This is due to the limitations surrounding External APIs. As APIs are constantly being updated and evolved, there is good potential to be able to implement these features in future iterations of this project. Furthermore, as APIs continue to grow there will be more opportunities to broaden the Event Listings further and expand the number of sources the events are taken from.

Outlined below is the Excel spreadsheet I used throughout the development of this project when researching the different APIs that could be implemented into the web application.

| | A | B | C | D | E | F |
|----|------------------|-------|--------------|--|---|---|
| 1 | Site | ? | Type | Comment | Link | |
| 2 | Ticketmaster | Yes | API | Ideal API, currently implemented | https://developer | |
| 3 | Eventbrite | No | API | Deprecated event listings endpoint in 2019 | https://www.event | |
| 4 | Seatgeek | No | Website/API | USA only | https://platform.s | |
| 5 | Eventful | No | API | Outdated, no longer exists | https://rapidapi.c | |
| 6 | AllEvents | No | Website/API | Ideal Concept, Requested API access (no reply), plugin (cannot be edited or changed) | https://allevents.i | |
| 7 | TicketBud | No | API | USA only, No event listings feature | https://api.ticketb | |
| 8 | TickPick | No | Website/API | USA only, API? Affiliate program | https://www.tickp | |
| 9 | StubHub | No | API | No longer exists | https://developer | |
| 10 | LiveNation | No | Promoter/API | Deprecated | https://rapidapi.c | |
| 11 | DataThistle | Yes | API | Ideal API, currently implemented, UK only | https://api.datath | |
| 12 | Songkick | No | Website/API | Not approving new requests | https://www.song | |
| 13 | SelectiveMemory | No | Promoter | | https://selectiver | |
| 14 | Gigson.ie | No | Website | Ideal concept, Source of events? | https://gigson.ie/ | |
| 15 | FoggyNotions | No | Promoter | | https://www.fogg | |
| 16 | WhelansLive | No | Venue | | https://www.whel | |
| 17 | Entertainment.ie | No | Website | Ideal concept, Source of events? | https://entertainm | |
| 18 | Last.fm | No | Website/API | Ideal concept, Deprecated events feature | https://www.last | |
| 19 | Tix | No | API | Can't create an account to get access to API | https://posapi.tix | |
| 20 | Ticket Leap | No | Website/API | Create events to sell tickets on website, API Key? | https://dev.ticketl | |
| 21 | Skiddle | Maybe | API | Ideal API, currently implemented | https://www.skidd | |
| 22 | | | | | | |

Figure 47 API research spreadsheet

Additionally, the SSL certificate limitation previously mentions can be resolved in future versions of this web application. After researching further, I found that obtaining an SSL Certificate from a Trusted Certificate Authority (CA) is the solution for future versions of this project (SiteGround, n.d.).

The motivation behind the choice of technologies also plays a key factor in the future development potential of this project. Each technology selected and implemented in the project is easily scalable and can adapt to handle the web application as it continues to grow

and evolve. Platforms such as AWS EC2 and MongoDB Atlas have scalable features to adapt to the project.

Lastly, future development of this project can expand on the profiles of the users who have created an account, allowing more customization such as dark/light website themes, profile pictures, custom list pictures, and even a real time chat feature.

5.0 References

References

SiteGround, n.d. *How to Fix the "NET::ERR_CERT_AUTHORITY_INVALID" Error?*. [Online]
Available at: https://eu.siteground.com/kb/fix-net-err-cert-authority-invalid/#NET_ERR_CERT_AUTHORITY_INVALID_in_Edge
[Accessed 10 May 2024].

6.0 Appendices

6.1. Project Proposal

National College of Ireland

Project Proposal

Event Point

22/10/2023

Bachelor of Science (Honours) in Computing

Software Development

2023/2024

Jade Fogarty

X20395471

6.1.1. Objectives

The main objective of this project is to enhance users' experiences when browsing, planning, and going to events. Allowing for event discovery, social connections, and secure ticket exchanges.

The goal is to create a web application to list all upcoming events from several different ticket vendors. These include the likes of Ticketmaster, Eventbrite, etc. It looks provide users with an aggregated list of upcoming events to browse and also display information about tickets, venues, and more.

This web app will enable smaller venues, and unticketed events to be shown in the events listings by allowing organizers to post their events directly to the website to be discovered by a wider audience.

Users will have the option to create an account and make various lists for events they are interested in and/or going to. They will also have the ability to connect with friends through their account who will be able to view these lists and create collaborative lists.

To enhance user experience, users will be able to make bookings for selected events directly through the website, or where this is not possible, users will be provided with a link to the official booking website.

Lastly this application looks to implement a feature to allow the secure exchange of tickets and payment between users who are looking to sell or buy tickets for various events. It will essentially act as a mediator in this transaction making it secure for both parties involved.

6.1.2. Background

I chose to undertake this project to solve common problems users face when browsing, planning, or attending events.

By providing a web application displaying events from multiple different sources (including smaller venues and unticketed events), it eliminates the need to search multiple websites to discover new events and the possibility of an event being unnoticed.

Enabling users to create an account, connect with friends, and make (collaborative) lists adds more personalised experience to the website and also assisting users to plan their upcoming events.

The project aims to address the ongoing issues surrounding ticket scams when attempting to resell a ticket. Due to a lack of a secure ticket resale marketplace, the web application looks to act as a trustworthy mediator in the exchange of a ticket and payment between users and reduce this risk for both parties.

These previously mentioned objectives will be achieved by developing a responsive, user-friendly interface which will implement the aforementioned functionalities.

6.1.3. State of the Art

Various similar platforms exist such as Ticketmaster, Eventbrite and MCD, which offer events listings, ticket sales and the ability to create accounts. This project looks to give users the convenience and usability of not having to search each individual platform already in the market. This will be done by providing users with a broad aggregated list of upcoming events and allow users to search for their events based on the category, venue, ticket availability, etc. This project also has the added

inclusivity of smaller venues and unticketed events being included in the listings which differ from existing platforms, who tend to focus more on major ticketed events.

The social aspect of this project is another feature in this project which differs from traditional ticketing platforms. It looks to allow users to connect with friends and create collaborative lists for event planning, making it usable for both individuals and groups who enjoy going to upcoming events.

The secure ticket exchange feature aims to handle the common issue users often face when trying to obtain or resell a ticket. When using Ticketmaster to transfer a ticket to another account only the ticket is handled in this exchange. There is no payment involved in the transaction therefore leaving users' vulnerable to scammers refusing to provide payment.

6.1.4. Technical Approach

The development approach I will use when working on this project is an agile methodology. I considered using a Waterfall model, but this will make it more difficult to adjust my project plan if unexpected issues occur and to carry out the appropriate amount of testing of the final implementation. Using the agile methodology, I plan to work in sprints and adjust the tasks being carried out according to factors such as supervisor feedback, development progress, and testing. This flexibility will allow me to adapt to the needs of the project and manage my time accordingly.

As I develop the project, I plan on using GitLab to store my code and maintain version control using a main branch and a dev branch. This will help me ensure that code being merged to the main branch is functional and minimise errors. I have a basic understanding of this platform from my internship, but I plan on doing further research to see other features I can utilise.

To identify requirements, I will be looking at the project from both a user's perspective and a developer's perspective to gain a full understanding of what is required of this project. For a user, factors such as user interface, functionalities, responsiveness, and security will be taken into consideration to provide a usable web application for all types of users. In terms of development, I need to identify the technologies, frameworks, databases, and APIs needed to carry out this project while also maintaining security, adhering to the ethics requirements, and performing sufficient testing.

Once each milestone is identified, it will be set with a deadline to ensure I manage my time efficiently. They will be further broken down into smaller tasks which will be carried out during each sprint. These milestones will be identified using a detailed requirements specification and visualisations such as flowcharts and use case diagrams to get an overview of what is required to progress while also assigning prioritization to appropriate tasks/milestones.

An example of a milestone would be the Log in / Create an account feature. I would consider this as a milestone as it will be one of the first aspects of the web application that a user will interact with. Tasks associated with this milestone would include creating a database to store user information, connecting this database to the backend server, creating a user interface, and linking the frontend code to interact with the database.

6.1.5. Technical Details

Frontend: HTML, CSS, JavaScript, React

Backend: JavaScript, Node.js, Express

To develop this project, I plan to create a user-friendly, responsive website to carry out the objectives outlined above. From previous modules I have experience in JavaScript using the Node.js environment along with Express framework to develop web applications communicating with an API, which is a similar concept I will be needing to use when displaying upcoming events to the website. However, during my internship I used the Swagger API platform, the ASP.net framework and coded in C# to carry out my tasks on a day to day basis.

Having knowledge of multiple frameworks is useful but it makes this decision more difficult as my experiences with both have been positive. After researching both pathways, I have decided to use the Express framework for the backend part of this project because I think being able to easily install libraries using Node.js will best help me develop this project. For the frontend portion of the project, I will use the React library as the components associated with it allows reusability and will help me maintain best practice. This will be combined with the use of HTML elements and CSS.

Database: MongoDB

To store user's account information, I will need to create a secure database. Although I have experience using MySQL in previous modules for college, I plan on using MongoDB. I briefly used this during my work placement and found it flexible and usable. Rather than create another project using MySQL, I'm hoping to gain more experience using a non-relational database.

Other considerations

APIs will be a key component when gathering the list of upcoming events to be displayed to users. I will need to carry out further research of API documentation to be able to aggregate these events. Vendors such as Ticketmaster and Eventbrite have APIs available which will be utilised in this project. An algorithm to display events from multiple APIs will be implemented to give users a comprehensible and ordered list and allow them to sort and filter this list as needed.

6.1.6. Special Resources Required

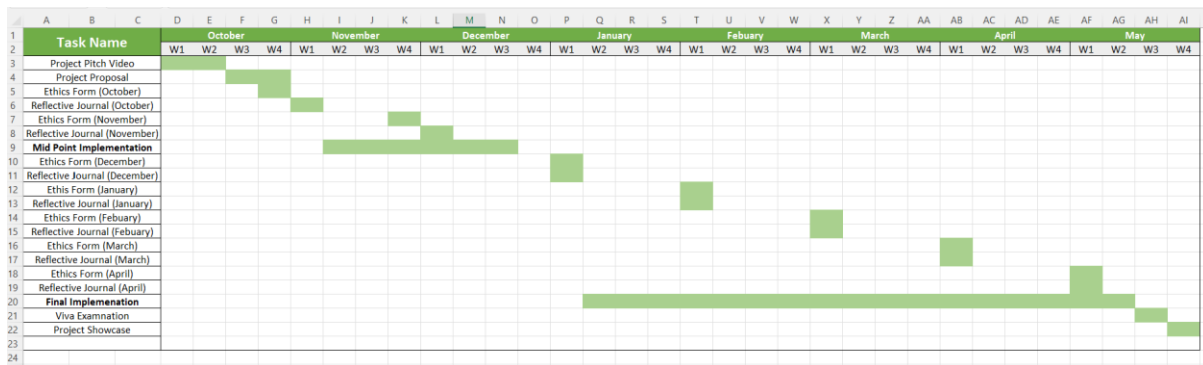
As mentioned in the previous section, in order to create the aggregated list of events access to (Ticketmaster and Eventbrite) APIs will be necessary. This will involve obtaining API keys, and complying to request rates while ensuring the events listed are regularly updates.

As of writhing this proposal document, in the early stages of planning there is no other special resources required, however this may change as the project enters development.

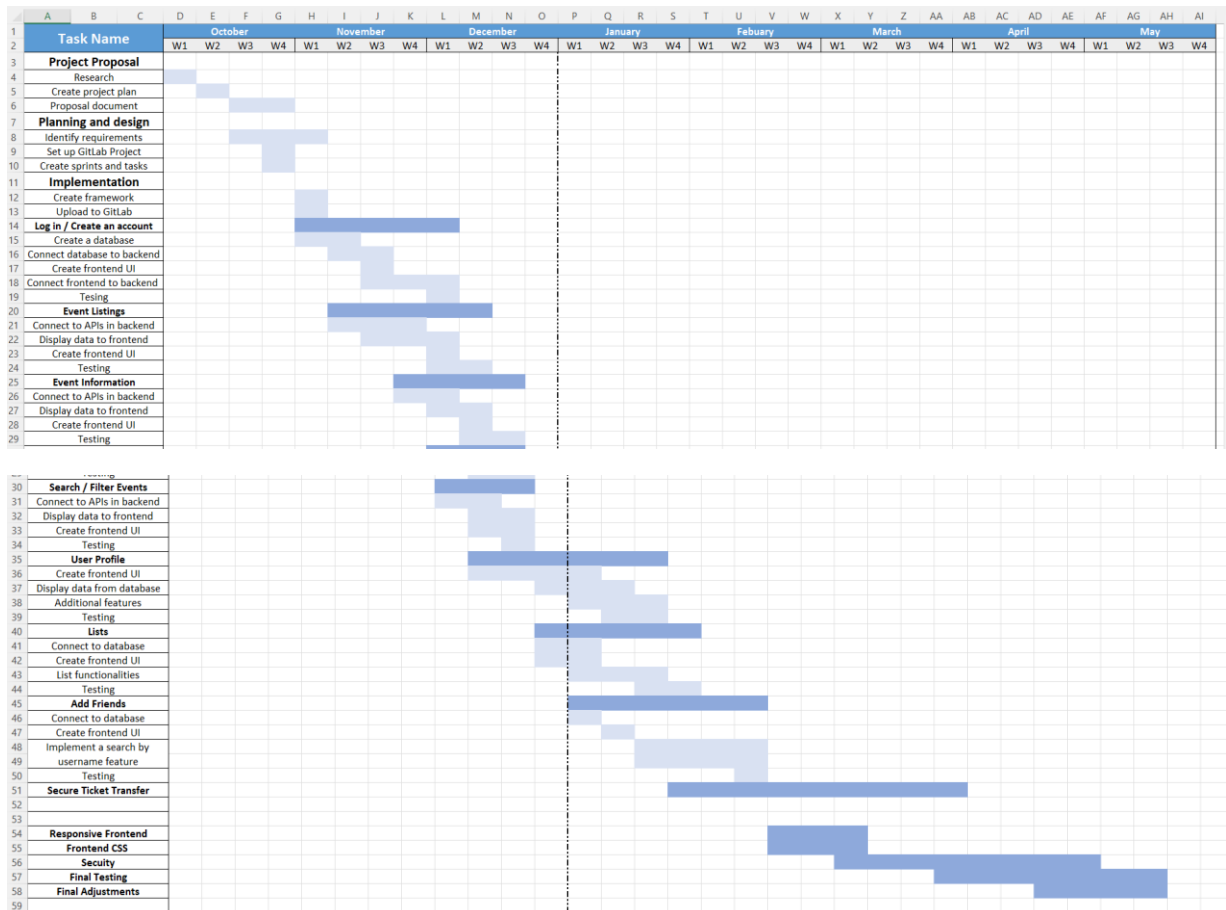
6.1.7. Project Plan

Using Microsoft Excel, I have created two Gantt charts which I will be updating regularly throughout the development of this project. This will track the timeline of upcoming deadlines and help me stay up to date with the progress of the project.

The first Gantt chart below represents the Modules deliverables for semester 1 and 2. As of writing this report specific semester 2 dates for the Reflective Journal entries and Ethics forms have not been released, once these are available the chart will be updated.



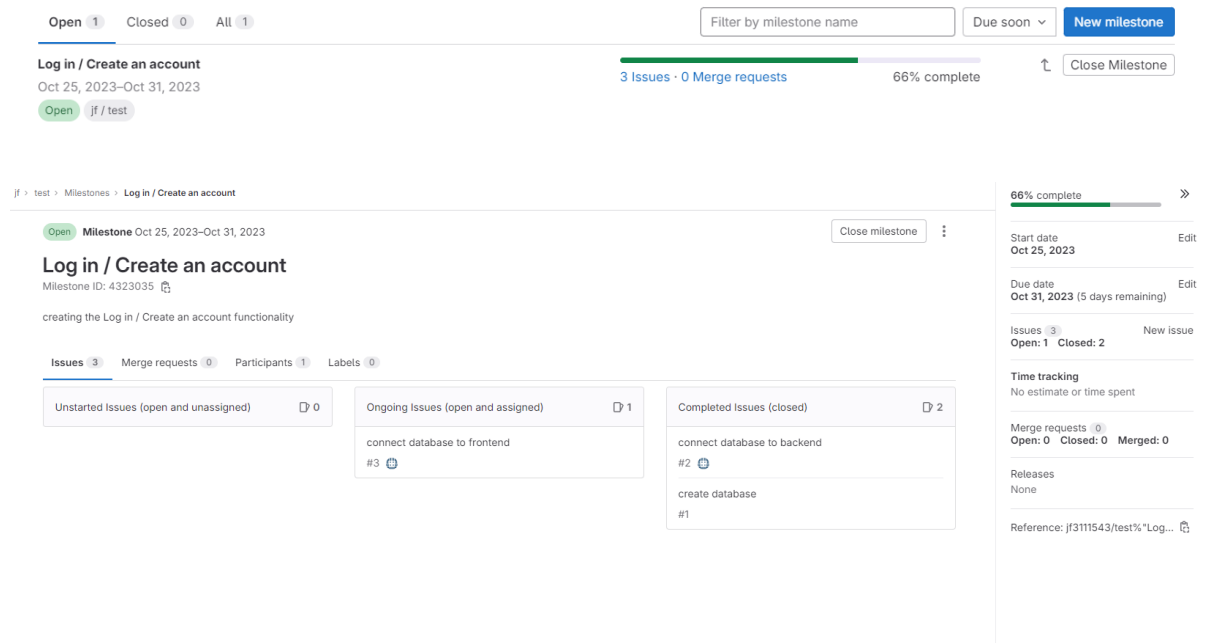
The second Gantt chart outlines a timeline for the specific milestones and tasks being carried out in the project. This gives a brief overview of the overall project and will be updated throughout development (e.g. adding milestones such as ‘ ‘Create Event” for small venues or unticketed events’). More of the specific details regarding tasks and milestones will be given in the Gitlab Project. The darker blue tasks represent milestones, and the lighter blue indicates tasks.



The chart is split into semester 1 (left hand side) and semester 2 (right hand side). Because a large part of the work in semester 1 was dedicated to planning and researching, the initial start to this project will be quite task heavy in order to get a working implementation ready for the midpoint. Because the Secure ticket transfer feature is the most complex, I have allocated a large amount of time to work on this. I have left a long duration towards the end of semester 2 to allow myself enough time to ensure the project is secure, well tested, and implements the required functionalities.

From working on previous projects, I found that getting the backend portion of a web application functional before developing a fully styled frontend is the best approach for me. This is one of the factors I took into consideration when planning the above mentioned timeline.

In order to manage progress of these milestones and tasks I will be using a feature within Gitlab to track progression of the project. Once this project has been approved and can be moved to development, I will create the milestones and tasks necessary. As an example of what is planned, I have included what this will look like in the test project displayed below:



6.1.8. Testing

This project will undergo continuous testing throughout the development to ensure that each functionality is performing correctly and being integrated into the web application successfully. The time this will take has been considered and outlined in the second Gantt chart above. In terms of system tests, unit tests will be performed on each component before they are implemented to ensure they are functioning correctly. Next integration testing will be carried out on the implemented feature to ensure it works in the system alongside the other features.

Towards the end of semester 2, functional tests will be carried out on the final implementation to test all the features and ensure the final web application is able to complete the objectives. To provide enhance user experience, performance testing will be carried out to ensure that the system is responsive in a timely manner with minimal delays. Security tests will also be performed to ensure users data is secure and safe from attacks such as SQL Injection and Cross-Site Scripting (XSS).

To perform the different tests mentioned above, I will use frameworks such as Mocha and Jest which provide a wide range of JavaScript testing. Mocha will be used to test the backend portion of the application and Jest for the frontend. GitLab CI/CD will also be used alongside these frameworks to perform testing after each push to the repository. After using a similar method during my internship, I may create 2 branches to work with. A main branch (which this continuous testing will be performed on) and a development branch (which will contain the current code being implemented). This will allow me to manage the testing more effectively by separating components.

In additionally, in the later stages of development, real users may be asked to use this web application and provide feedback on issues, improvements, and their general experience. Before

carrying out this process, I will ensure that the Ethics Declaration and Ethics Application have been approved, and that the web application is secure and GDPR regulations will be taken into consideration. This is a step that will be considered in the later stages of development.

6.2. Ethics Approval Application (only if required)

N/A

6.3. Reflective Journals

October

Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month:

What?

Throughout this month I spent time working on the Pitch Video and writing the documentation for my Project Proposal. During the first week, I created a list of possible ideas for my pitch video and narrowed it down to the one I feel I could best implement. After submitting the pitch video I began research for the proposal. I created a rough timeline for the project throughout semester 1 and 2 and identified the objectives I hope to complete at the end. I also decided the technical approach I plan on using (agile methodology) and brainstormed the various languages, frameworks, and libraries which will be used to develop the web application.

So What?

From brainstorming my ideas, I was able to finalise the idea I would pitch and base my proposal on. As I await approval, I completed the necessary Ethics Declaration form and was able to perform in depth planning for the project. This was a success as I now have a clear timeline and a good understanding of the deliverables I hope to complete by the end of semester 2. This allowed me to plan further into the functionalities which will be implemented in this project and the smaller tasks in which these can be broken down into.

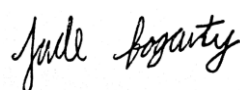
I was also able to further research the technologies I will be using and get a better understanding of how the web application will be structured. This gives me a good head start on the next stage, which will be planning and designing the systems architecture, before moving into development shortly after. Another success I faced was researching further into GitLab and its features. Initially I was going to use a GitHub repository as version control for the code but I decided to use GitLab as I used it throughout my internship. After creating a project to test the features of the platform I discovered that there is a feature to plan milestones and tasks. This will be very beneficial when tracking the progress of my project and keeping on track with the timeline.

A challenge I am facing surrounds the secure ticket transfer objective I have set. As of now I am unsure on how this will be implanted because many factors need to be taken into consideration. These include security, ticket and payment handling, and API limitations. Another challenge I have encountered is the lack of APIs available. Initially I thought MCD productions would have an API available, however after researching I was unable to find one.

Now What?

I plan to do extensive research on implementing the Secure Ticket Transfer feature and ways to overcome the challenges surrounding this task. In order to broaden the events my web application can display I will also be researching more ticket vendors other than Ticketmaster and Eventbrite with available API documentation and looking into other ways to fetch event listings. In the next week I will arrange a meeting with my supervisor and discuss further plans and adjustments that need to be made.

Student Signature



November

Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: November

What?

Throughout this month I began communication with my supervisor to discuss the next steps of the project. After looking over the proposal and discussing the idea for the project in more detail, I received some helpful guidance for planning and writing Use Cases (before beginning to code). I began working on the Technical Report for the midpoint implantation to identify the functional requirements. Once these requirements were established, I worked on writing the Use Cases for each, outlining their processes, and documenting their priority within the project. I also began brainstorming the outline for my database which will store User Information, Event Information, etc.

So What?

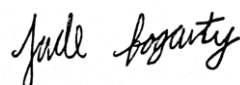
As discussed in the meetings with my supervisor, developing the Use Cases before writing the code allows me to thoroughly plan and identify what will need to be implemented into the project before writing the code. This will help keep my project on the right track when code development begins and if I encounter any issues, the Use Cases will be a good point of reference to help identify a solution. As I was writing the flow for the Use Case Requirements I discovered more possible outcomes, which has not originally been considered when brainstorming the requirements.

After successfully brainstorming an outline for the Mongo DB database I plan to create and establishing a good guideline of what will need to be implemented, the challenge I am facing surrounds writing the documentation in my report. From my experience in past modules and working with MySQL in the previously, I am unfamiliar with how to properly write documentation for Mongo DB.

Now What?

I plan to do research on writing documentation for Mongo DB by looking at resources online to overcome this challenge. Over the next month I also plan on finalising the Use Case requirements and begin writing the code to successfully send API calls. This will allow me to start working on a prototype for the midpoint implantation coming up.

Student Signature



Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: December

What?

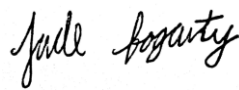
During this month focused on my submission for the Midpoint of the project. Once the Use Cases were completed for the midpoint, I began developing the code in order to get a working implementation for the Midpoint. The functional requirements I focused on were View Events and the Log in / Create an account. I also worked on the technical report and filled out the sections required for the midpoint.

So What?

With the midpoint submission completed, this gives the project a good milestone for development. With requirements such as View events and log in / create an account implemented, I can use these as a guide when continuing to develop the other requirements. Looking at the architecture of the backend and frontend alongside these implemented features helps establish the flow of events going forward. With the MongoDB user schema clearly defined now, this allows me to begin developing the requirements surrounding User accounts and profiles.

Now What?

I plan to take a step back and analyse the progress of my project and take into consideration what the next steps for development are for the upcoming semester. This will help me clearly define my goals and reflect on the timeline so far. I will continue meeting regularly with my supervisor once semester 2 begins.

| | |
|--------------------------|--|
| Student Signature |  |
|--------------------------|--|

January

Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: January

What?

Throughout this month I spent time analysing the progress of my project to date, following the midpoint submission and the break between semesters. I also attended the first lecture of the semester to gain more information about what is expected during semester 2 and how it will affect the project.

So What?

Attending the first lecture was very beneficial as I was able to get a good idea of the deliverables of the module and can now begin planning how I am going to achieve these. Knowing the various deadlines for the final submissions and specific seminars which may be run during the semester helps me manage my time and workload when developing the project alongside the other modules.

Analysing the progress of the project allowed me to identify the upcoming requirements and enables me to update the project's timeline. This will help me manage my time efficiently as the semester continues and hopefully meet the deliverables for the module. From looking at the project's status to date, I was able to identify the features of the functional requirements which need to be worked on in the following months. I

have also identified some of the challenges which will be faced as these functional requirements are being developed.

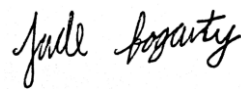
The main challenge I have identified is the limitations surrounding the current APIs available for events and ticketing in Ireland. After researching further into them, many of the ones I originally intended to implement into the project have been deprecated, have limited features, or do not offer a good variety of features for events in Ireland.

Now What?

What can you do to address outstanding challenges?

To address these challenges, I plan to continue researching suitable APIs to implement into the project in order to develop the functional requirements. I also plan on expanding the location of the events being listed to outside of Ireland in order to highlight the features of the project and complete the requirements.

Student Signature



February

Supervision & Reflection Template

| | |
|----------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: February

What?

During this month I began implementing software testing, using Jest, on the frontend React components currently in the project. I also continued researching APIs to expand the list of events on display in the View Events requirement.

Lastly, I received the midpoint grade and discussed with my supervisor the feedback and progress of the project to date.

So What?

Implementing the Jest library into the frontend React project was initially challenging due to the ESM syntax of the project. After some time spent troubleshooting this issue, I was successful in installing the library and running successful tests. From here, I can now easily expand and adapt these test cases as the project continues developing to ensure the components are behaving as expected.

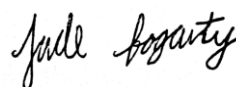
Researching APIs is still a challenge this project needs to overcome. Many of the APIs available offer limited functions which are not suitable to implement into the project or are no longer in service and available to use. Those that do have the ideal features available in their documentation are not approving new requests to access the API or are only offering data on events based in the USA.

After receiving the grade and feedback from my supervisor on the midpoint submission, I was able to reflect on the projects progress and areas which need to be adjusted and improved during this semester.

Now What?

I will continue researching possible alternatives to the APIs for the event listings in the View Events requirement. As mentioned in last months reflection, I had intended to focus the event listings to Irish events, but this is being adjusted and updated to offer event listings from the UK and Ireland to account for the limited services available.

As more components are being added to the project the testing will become more challenging and complex. I plan to continuously update the test cases as the project develops in order keep up with the changes and ensure the project is sufficiently tested.

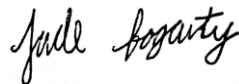
Student Signature

March

Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: March

| | |
|--|--|
| <p>What?</p> <p>Throughout this month I expanded my API research to include UK event listings and have successfully found multiple APIs which can now be implemented into the project to expand the Event listings. These APIs include DataThistle which focus primarily on UK events, and Skiddle which has endpoints for both the UK and Ireland.</p> <p>I have also worked on the development of the functional requirements surrounding User profiles, and Custom user lists. Progress of the projects development has slightly slowed down due to other deadlines however I am still working on the technical report and brainstorming other requirements implementations.</p> | |
| <p>So What?</p> <p>After researching and finding additional APIs which can be implemented into the project, this expands the View Events requirement and allows a broader range of events to be presented to the user. The APIs can all be called simultaneously in the server.js file and their responses can be combined and stored in the database. A challenge faced when working on this task was the fact that each API response is structured differently. To address this issue, I implemented a function to process the results and save the necessary fields from each in an object variable matching the database schema. From here the object variables can be combined and pushed into the database.</p> <p>By beginning development of the user profiles features, this allows further requirements to begin development. Custom user lists creation and adding events to this list allows the data of the list and event to be saved to the users account and viewed in the user profile component. This makes good progress on coding and implementing the functional requirements mentioned in the report.</p> | |
| <p>Now What?</p> <p>I will continue working on developing the user requirements in order of their priority, while also going back and adding the smaller functionalities to those that are already implemented. Since the majority of assignments for other modules have been submitted I can now put more focus into working towards the final implementation of this project.</p> | |
| <p>Student Signature</p> |  |

April

Supervision & Reflection Template

| | |
|-----------------------|----------------------------|
| Student Name | Jade Fogarty |
| Student Number | X20395471 |
| Course | BSc (Honours) in Computing |
| Supervisor | Emer Thornbury |

Month: April

What?

With all other assignments submitted, I was able to dedicate more time to developing this project. I was able to implement the main functional requirements of this project while also moving two of the requirements to the future development section of the report. The removed requirements include the Book Ticket and Secure Ticket Transfer.

I have also worked on adding better error handling throughout the project and implementing a Notifications feature for handling sending friend requests.

So What?

With the majority of the requirements now implemented, this allows me to focus on adding the smaller details to each functionality. The functionalities worked on include: edit and delete custom user lists, display and search user profiles and display their profile information, socket.io for real time notifications, send and receive friend requests, offline notification management, remove event from list, add collaborators to list, remove friend, and strong password requirements. The addition of the previously mentioned functionalities means the project is in the late stages of development and on track to be completed by the final deadline.

I removed the Book Ticket and Secure Ticket Transfer functional requirements from the main software requirement specification section due to the API limitations. They can now be found in the future development section.

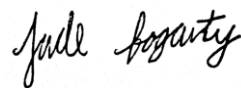
I also changed the structure of the login and create an account functionality in both the frontend and backend. This is to allow a more secure approach while implementing better use of jwt. Session management in the

frontend stores access tokens in cookies. Protected routes that require the user to be logged in take the access token sent in a header to verify the users identity.

Now What?

I will continue working on the technical report and code to have them completed for the final implementation. I also plan to implement a CI pipeline for building and deploying both the frontend and backend projects. This will enhance the overall project and allow an accurate demonstration if a real world web application to be developed.

Student Signature



6.4. Invention Disclosure Form

1. Title of Invention

Event Point

2. Inventors

| Name | School/Research Institute | Affiliation with Institute (i.e. department, student, staff, visitor) | Address, contact phone no., e-mail | % Contribution to the Invention |
|--------------|-----------------------------|---|------------------------------------|---------------------------------|
| Jade Fogarty | National College of Ireland | School of Computing | X20395471@ncirl.ie | 100% |

3. Contribution to the Invention

Each contributor/potential inventor should write a paragraph relating to his/her contribution and include a signature and date at the end of the paragraph.

My contribution to the project involves planning, designing, developing, and testing throughout all stages.

Signature: 

Date: 11/05/2024

4. Description of Invention

The innovative aspects of this project include the View Events [EP001] functionality by providing an aggregated list of events from different sources.

Enabling users to create custom lists of events [EP006] and providing the option to make these lists collaborative [EP008] with friends.

5. Why is this invention more advantageous than present technology?

This is more advantageous in comparison to other applications in the market by providing an aggregated list of events from multiple different ticketing vendors. This provides users with a convenient location to browse events and eliminate the need to search multiple sites.

This project also looks to engage its users in the event planning, browsing, and attending by implementing a social aspect. Allowing users to add friends, create custom event lists, and collaborate with friends in this list creation enhances the event going experience.

6. What is the current stage of development / testing of the invention?

The prototype of the project is in the final/finished stages of development.

7. List the names of companies which you think would be interested in using, developing, or marketing this invention

Consumers

8. Funding Partner(s)

| | |
|--------------------------------|--|
| Government Agency & Department | |
| % Support | |
| Contract/Grant No. | |

| | |
|--------------|--|
| Contact Name | |
| Phone No. | |
| Address | |

| | |
|---------------------------|--|
| Industry or other Sponsor | |
| % Support | |
| Contract/Grant No. | |
| Contact Name | |
| Phone No. | |
| Address | |

9. Where was the research carried out?

National College of Ireland

10. What is the potential commercial application of this invention?

This application could be commercialized to allow advertisers to post their ads throughout the site.

11. Was there transfer of any materials/information to or from other institutions regarding this invention?

If so, please give details and provide signed agreements where relevant.

No

12. Have any third parties any rights to this invention?

If yes, give names and addresses and a brief explanation of involvement.

| |
|--|
| |
|--|

13. Are there any existing or planned disclosures regarding this invention?

Please give details.

14. Has any patent application been made? Yes/No

If yes, give date: _____ Application No.: _____

Name of patent agent: _____

Please supply copy of specification.

15. Is a model or prototype available? Has the invention been demonstrated practically?

Yes, a prototype is currently implemented and is in the final stages of development. The application is deployed to the cloud and a demonstration has been carried out as a part of the final submission.

I/we acknowledge that I/we have read, understood and agree with this form and the Institute's *Intellectual Property and Procedures* and that all the information provided in this disclosure is complete and correct.

I/we shall take all reasonable precautions to protect the integrity and confidentiality of the IP in question.

Inventor: Jade Fogarty

Date: 11/05/2024

Signature: 

6.5. Additional Functional Requirements

6.5.1. Book Ticket

Description & Priority

This feature allows a user to book a ticket to a selected event directly through the web application, or (where not applicable) will be provided a link to the official ticketing vendor's website. This will minimize the amount of browser searching users will have to do and provide users with the quickest and easiest way to book a ticket.

Priority: Medium

Use Case

Book ticket

[EP008]

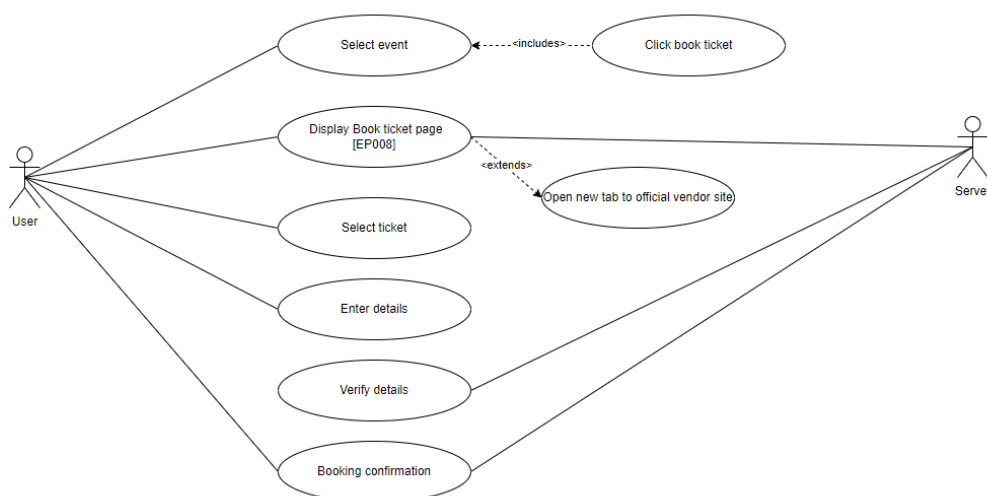
Scope

The scope of this use case is to allow users to book a ticket for an event.

Description

This use case describes the functionality enabling users to book a ticket for a selected event utilizing the web application.

Use Case Diagram



Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user selects an event.

Main flow

8. The system displays information about selected event.
9. The user clicks 'book ticket'.
10. The system checks if there are tickets available (E1).
11. The system displays the 'available tickets' page (A1).
12. The user selects their ticket preference.
13. The system presents the 'checkout' page.
14. The user enters their details to complete the transaction.
15. The system verifies the details entered are correct (E2).
16. The system displays a confirmation of the booking and details.

Alternate flow

A1 : Unable to directly book ticket

6. The system is unable to directly book tickets for specific event.
7. The system presents a message to the user asking for permission to open a new tab with the official ticketing vendor website.
8. The user clicks 'yes' button (A2).
9. The system opens a new tab to the official ticketing vendor's website.
10. The system redirects Event Point tab to the selected event information page.

A2 : No

1. The user clicks 'no' button.
2. The system returns to the selected event information page.

Exceptional flow

E1 : No tickets available

11. The system presents an error message stating that there are no tickets available.
12. The system redirects to the selected event information page.

E1 : Incorrect details

8. The system presents an error message stating that the details entered are incorrect.
9. Return to step 7 of the main flow.

Termination

The system presents the next web page.

Post condition

The system goes into a wait state.

6.5.2. Secure Ticket Transfer

Description & Priority

This requirement enables users to sell or purchase a resale ticket directly through the web application. Once a user selects an event, they will have the option to resell a ticket for (no more than) face value. From here other users can see if there is a resale ticket available for the selected event and can purchase it.

Priority: Medium

Use Case

Secure ticket transfer

[EP009]

Scope

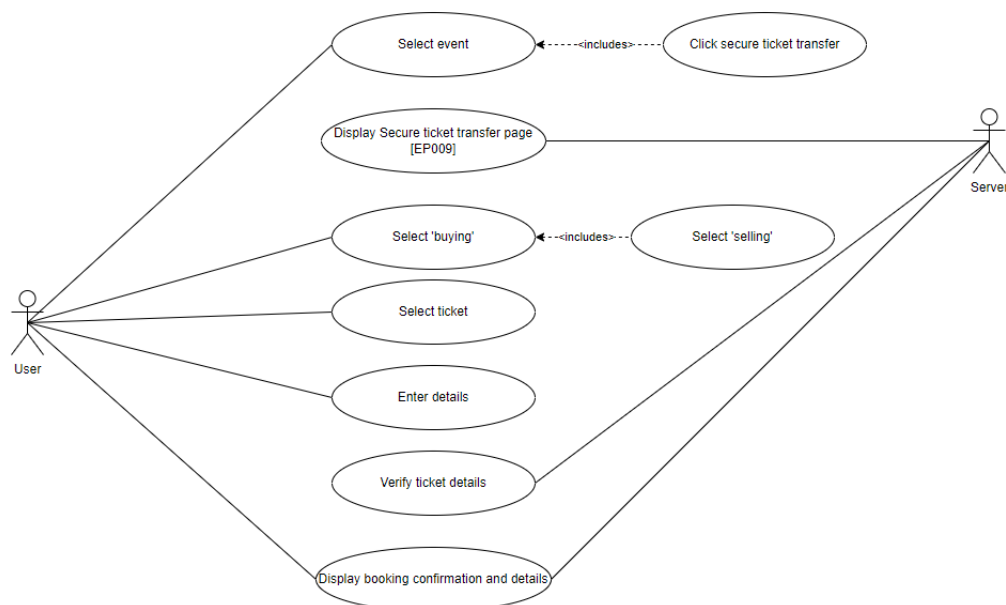
The scope of this use case is to allow users to buy or list a resale ticket for a selected event.

Description

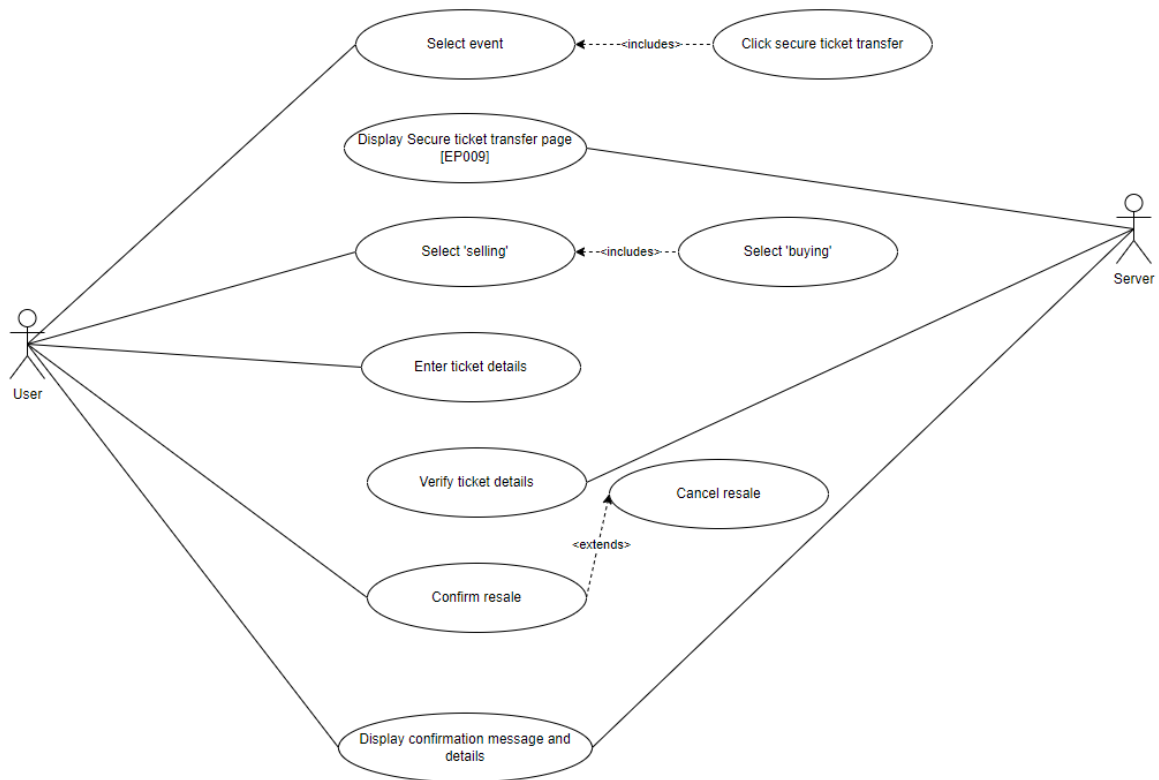
This use case describes the transaction between two users and the way in which the web application acts as a secure mediator for both parties.

Use Case Diagram

Case: buying a resale ticket



Case: reselling a ticket



Flow Description

Precondition

The server is running and accessible to users.

The user is logged in to their account.

Activation

This use case starts when a user selects an event.

Main flow

1. The system displays information about selected event.
2. The user clicks 'secure ticket transfer'.
3. The system asks the user if they are buying or selling a ticket.
4. The user selects 'buying' (A1).
5. The system checks if there are any resale tickets listed (E1).
6. The system displays the 'available resale tickets' page (A3).
7. The user selects a ticket.
8. The system presents the 'checkout' page.
9. The user enters their details to complete the transaction.
10. The system verifies the details entered are correct (E2).
11. The system displays a confirmation of the booking and details.

Alternate flow

A1 : Selling ticket

4. The user selects 'selling'.
5. The system presents the 'resale ticket' page.
6. The user enters the ticket details.
7. The system verifies if the details entered are valid (E3).
8. The system stores the ticket and transaction details.
9. The system presents a message stating the ticket will be listed for users to buy.
10. The user clicks 'confirm' (A2).
11. The system displays a confirmation message and details.
12. The system redirects back to the selected event's information page.

A2 : Cancel resale

10. The user clicks 'cancel'.
11. The system redirects back to the selected event's information page.

A3 : No resale tickets available

6. The system displays an error message stating that there are currently no resale tickets available.
7. The system redirects back to the selected event's information page.

Exceptional flow

E1 : No resale tickets

2. The system presents an error message stating that there are no resale tickets available.
3. The system redirects back to the selected event's information page.

E2 : Incorrect details

5. The system presents an error message stating that the details entered are incorrect.
6. Return to step 7 of the main flow.

E3 : Invalid ticket details

7. The system presents an error message stating that the details entered are invalid.
8. Return to step 6 of A1.

Termination

The system presents the next web page.

Post condition

The system goes into a wait state.