Title: ***Graduation Gathering***

By
Adam Murray

Course: Computer Science BSc

Project code: ***PJE40***

Supervisor: ***Dr Gail Ollis***

April 2024

Word count: 10998

Please tick

☒	I give permission for my project to be published in the University library and/or be made available to other students as examples of previous work. (optional).
☒	I confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct.
☒	I declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced.

Date: 03/05/2024

Contents

Acknowledgements

I would like to thank my supervisor Dr Gail Ollis for all of their help and guidance through this project. Their words have always helped to calm and focus me when this project has gotten stressful.

Abstract

Graduation day is a very busy and chaotic day, yet it is also the last chance for many students and staff to see each other. There exists no way for students and staff to efficiently meet up to give congratulations and say goodbye. This project details the creation of a location sharing application that would allow students and staff to decide who they want to be able to find them ahead of time and so on graduation day they can efficiently meet up. This application could be implemented by in an official capacity by the university.

Introduction

The problem this project aims to solve is that on graduation day, among the chaos and the crowds, it can be difficult to find other students and staff who you would like to talk to. This day for many could be the last opportunity to talk to those who you have connected with over the previous years. The solution Graduation Gathering hopes to provide is a real-time location sharing platform from which students and staff will be able to share their location and find those who they have connected with.

In order to connect as many students and staff as possible, Graduation Gathering must be available on both iOS and Android whilst providing safeguards so that the system cannot be abused. This will be achieved by restricting the use of the app to exclusively students and staff of the University of Portsmouth and only enabling location sharing on graduation day itself and in areas that are related to graduation. Users will also be able to clearly manage who has permission to see their location so that they can be sure that their location is only shared with those who they intend to share it with. Due to the nature of sharing your real-time location, the safety of the users is my first priority when making this application.

The idea of an app that has the ability to help people find one another is obviously not new, any messaging app can be used by individuals to arrange to meet up. However, most existing software focuses on arranging meeting up ahead of time at a specific time and place which is unsuitable for the dynamic nature of graduation day. Graduation Gathering would solve this problem by allowing users to select who they want to see their location ahead of time, and then on the day they would just use the app to quickly find the other users.

For Graduation Gathering to be successful in its use, it will need to be both accurate in its location sharing and be user friendly. Graduation Gathering is intended to only be used for a short time and so the UI must be intuitive or the user will simply not use the app.

This will require a system that can display users locations on a map of Portsmouth, as well being able to handle location sharing for a large number of simultaneous users. This location sharing creates a problem as for the tracking to be accurate the locations of users will need to be updated very frequently and so the server will be called frequently by the users phones. At the same time, due to the nature of all the users having the potential to be sharing their locations with each other, the server must store the location data in a single place creating a single point of failure.

Aims and Objectives

The aim of this project is to create an application that will allow for its users to securely share their location with other users. This app should be available on the latest version of iOS and the latest version of Android. It should be able to support a realistic number of simultaneous users.

This artefact should also be rigorously tested to ensure that it is bug free. Due to the nature of location-sharing, the security should also be tested to ensure that the app cannot be abused.

Constraints

The main constraints for this project are that the application needs to be deployed by graduation day, estimated 12 July to 28 July 2024, and the project deadlines which are as follows:

- Submit PID, 20 October
- Pass Ethics Review, 8 December
- Submit Final Report, 3 May

Since the app needs to be deployed before the final report is written the deadline of graduation day is redundant.

Risks

No	Description	Likelihood (high, medium, low)	Impact	Mitigation/Avoidance
1	Unable to source sufficient access to an Apple Computer	High	Could cause the IOS app the be released with known bugs.	Develop for both IOS and Android so there will still be an Android app produced irrespective of issues with IOS
2	Computer breaking	Low	Delays to the project and data loss	Frequently save work to Github to ensure as little data is lost as possible
3	Personal Life Issue	Low	Delays to project	Communicate effectively with supervisor when issues arise
4	AWS becomes unsuitable to host the server.	Medium	Delays to project as new server solution would have to be found	Look into potential alternatives to AWS

Literature Review

There is a large amount of literature on the subject such as a study done on the crowdsensing framework used at the Hajj pilgrimage site (M. A. Rahman and M. S. Hossain 2017).

There are also existing applications such as WhatsApp which allows for real-time location sharing between individuals and groups, however you do require the other people's phone numbers. Google Maps also contains a location sharing feature that allows you to share with family and friends. You can select a time limit on how long you want to share your location for, however this requires having the other users google account.

Methodology

Software Development Methodology

The methodology I will use needs to be chosen, and altered, according to the specifics of this project. So to summarise, this project will be developed in its entirety by 1 developer, this project will require a front-end and a back-end, this separation allows different parts of the project to be completed separately, this project will also require a database. Also as the solo developer, I do not have a complete understanding of all of the technology that might be required. Finally, the stakeholders of the app are its potential users, student and staff of the University of Portsmouth, as well as the graduation team for the University. Importantly, since this app is designed solely for graduation day, it is hard to get accurate feedback from the stakeholders as no one knows how it will actually be used on the day, until graduation day happens.

The work will be completed more efficiently if tasks related by how they are implemented are completed sequentially. For example, the database queries will be written more efficiently if written closer together in time due to the knowledge of the query language being fresh in my mind. Since there is only 1 developer, this means that we should try to use a methodology that allows us to focus on certain groups of tasks instead of implementing all of the project at once.

Since feedback is difficult to attain, but not impossible, for instance if a meeting with the graduation team is able to happen, we still need to allow flexibility within our methodology so that such feedback gained can still be acted upon.

So we need a methodology that allows us to update our requirements and design as we go along, whether from receiving feedback or from an improved understanding of the technology that is being used. The methodology must also allow us to work on related components of multiple features sequentially instead of demanding that a feature be completed before work can be started on another. For example, implementing the back-end of both the location sharing feature and the profile settings feature before going and implementing the front-end of either.

I considered Kanban, however with Kanban I would be required to write down all related tasks that would need implementing before I start the first task in a group. This means that as I complete the tasks, and so my knowledge of the areas grows, if I decide that I need to redesign a part of the system, I would then also need to redo the Kanban board as with even just a small redesign, the tasks would likely change. This will waste time, I believe it to be a better approach to keep future tasks in a group vague so that I can adapt easily to changes. I admit that in a group environment, this thinking ahead is required to keep all team members working in the same direction, and a Kanban board can then be an excellent way to do this. However, since I am working by myself, I would not gain this teamwork benefit and so Kanban is not suitable for this project.

I believe the most effective way to do this would be to use a modification of the Scrum software methodology. Scrum has teams complete work on related features in time-bounded sprints. Within a sprint I will choose part of the project to work on and I will aim to complete all tasks that fall under that part of the project. Examples of different sprint topics are as follows, the back-end of the project, the UI, the map interaction. I will modify Scrum by implementing sprints of varying time. Since not all of my topics will take a similar amount of time to complete, I will vary the length of my sprints accordingly. This approach allows me to iteratively build my

project efficiently and alter the requirements and design at any point without causing major delays.

Requirements Gathering

For this project, as a student I have a good understanding of what I would want from an app marketed to graduating students. However, it is both good to get the opinion of other students as well as staff as this app is marketed towards both students and staff. I also thought it was important to contact the graduation team as they are experts on graduation day. In order to get the opinions of students and staff I created a custom questionnaire that gave me the freedom to ask whatever questions I wanted to without being restrained by a tool such as Google Forms. As for my contact with the graduation team, I arranged a 1 hour meeting with a member of the team during which I was able to get feedback on my ideas as well as talk about their ideas for my app. I have now split my requirements into functional and non-functional requirements and then ordered them using MoSCoW prioritisation.

Graduation Gathering Questionnaire

This questionnaire targets students and staff, primarily asking them to rank a list of features from most important to least important. It also asks about whether the person is a student or member of staff as well as which areas of Portsmouth they would like to become graduation zones, and finally asks for suggestions.

Question 1 asked whether the participant was a student or member of staff of the University of Portsmouth. This allows me to gauge the difference in opinions between staff and students.

For question 2, I asked the participants to rank a number features from which are most important to them to least important to them. This allows me to prioritise which features should be developed first, and so if time runs out, it will lead to only the most desired features having been added. These features are:

- Users of the app are verified University of Portsmouth staff or students.
- You can be selective about which graduation zones you appear in.
- You can be selective about which graduation zones you can see others in.
- You can filter which users you can see by their course.
- You can filter which users you can see by their name.
- You can filter which users you can see by whether you've already met.
- A messaging service that allows you to communicate with other users.

For question 3, which posed 3 possible graduation zones (Guildhall Square, Ravelin Park, and the area between Guildhall Square and Ravelin Park) and asked for each of them whether users would like these areas to be zones or not. This allows me to only add desired zones to the map.

For question 4, I asked as an optional question are there any other areas that the questionnaire taking would like to be designated as a graduation zone, I could then add these as additional zones.

For question 5, I asked as an optional question for any other ideas. I could then consider adding these features as requirements.

Graduation Team Meeting

I had meeting with a member of the graduation team from the University of Portsmouth, I got a consent form signed by them that allowed me to use their ideas and name in this project. During this meeting I showed them the app, as it was at the time, in order to get feedback, they also gave me ideas for future features.

Requirements

Graduation Gathering Questionnaire

I received 17 responses to my questionnaire, 14 of which from students, 3 from staff.

For question 2, the messaging service was ranked lowest on 88% of the responses. All the other features were pretty randomly mixed together throughout the responses. Due to this I will not add the messaging service to the app.

For question 3, every single response said that all 3 of the zones should be zones. This tells us that these zones were selected correctly during initial analysis.

For question 4, I got 2 responses, one from a student asking the ability to set a larger area of Portsmouth as a zone for example 1 mile outside of the current zones. The other response from a member of staff asking to be able to draw their own zones. Both of these are asking for a larger amount of control over the zones, due to the added complexity of implementing this, it will be considered for future work but not added to the requirements of this project.

For question 5, I asked as an optional question for any other ideas, for which I got no responses.

Graduation Team Meeting

I met with Lisa Scott, a member of the graduation team for the University of Portsmouth. The ideas Lisa had include but are not limited to:

- Showing key locations on the map such as photography at Ravelin Sports Centre and the ceremony at the Guildhall.
- A reminder sent to students before their ceremony reminding them to be at least 10 minutes early and that they cannot be late.
- A checklist of what students need to do on graduation day such as picking up their gown, attending their ceremony, going to the reception and collecting their certificate.
- Adding a routing feature to help students both navigate to the above mentioned locations but also the other app users that they want to find.

Lisa was also able to give me estimates on the number of students on site on a graduation day and so this can be interpreted as the maximum number of simultaneous users of the app, which would be 2000.

Most of these ideas are unrelated to the core functionality of the app. However, these are ideas that will assist students on graduation and would benefit from the map that will already be integrated into Graduation Gathering. For this reason these features will be added to the lowest priority of work and future work.

Functional

Must Have

ID	Description
1	A map that is rendered on screen showing Portsmouth
2	The user's current location displayed on the map when the user has location enabled
3	Other users current locations to be displayed on the map to the user
4	Manage who has permission to see your current location
5	Manage who's location you have permission to see
6	Users login with their university emails (@myport.ac.uk and @port.ac.uk) by having a code sent to their email that they will then enter into the app
7	All requests sent to the server encrypted using HTTPS
8	All requests sent to the server must be authenticated
9	Designated graduation zones to appear on the map, only in which will users locations be shared
10	Users locations only being shared when they have the app open and are logged in
11	Users locations only being shared on graduation days(8am – 1am the next day)

Should Have

ID	Description
12	Users can add their name to their account
13	Users can add their faculty to their account
14	Users can add their school to their account
15	Users can add their course to their account
16	The account type (Student/Staff) saved to an account based on the email address that they used to login
17	Users can select which graduation zones they must be in for their location to be shared
18	To search for other users by their email address, when deciding who has permission to see you and who you can to see
19	To search for other users by their name, when deciding who has permission to see you and who you can to see
20	To search for other users by their faculty, when deciding who has permission to see you and who you can to see
21	To search for other users by their school, when deciding who has permission to see you and who you can to see
22	To search for other users by their course, when deciding who has permission to see you and who you can to see
23	To search for other users by their account type, when deciding who has permission to see you and who you can to see

Could Have

ID	Description
24	The ability to tap users you can see on your map to display the information they have added to their profile
25	The ability to tap users you can see on your map and tick a box to say you have already seen this person and so to stop that user from continuing to display on your map
26	To filter the users that appear on your map by faculty
27	To filter the users that appear on your map by school
28	To filter the users that appear on your map by course
29	To filter the users that appear on your map by email
30	To filter the users that appear on your map by name
31	To filter the users that appear on your map by graduation zones
32	To filter the users that appear on your map by account type
33	A checklist of what students need to do on graduation day such as picking up their gown, attending their ceremony, going to the reception and collecting their certificate
34	Markers on the map showing the key locations for graduation (Guildhall, Ravelin Sports Centre, Gun House Green)
35	The above mentioned markers clickable to display information about what is happening at these locations

Will Not Have Right Now

ID	Description
36	A routing feature to help users navigate to the above mentioned key locations
37	A routing feature to help users navigate to other users
38	Draw your own graduation zones that you can appear in to others
39	Message other users
40	Send a push notification to another user to let them know that you're trying to find them

Non-Functional

Must Have

ID	Description
41	Run on the latest version of Android(Android 14)
42	Handle 50 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)
43	A users location update on other users devices at most 30 seconds after the users client get an update of its own location (assuming all users have a good connection to the internet)
44	The Authentication tokens allow users to login for no more than 24 hours
45	The login codes sent to users emails should be valid for no more than 5 minutes

Should Have

ID	Description

46	Run on the latest version of iOS(iOS 17)
47	Handle 500 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)

Could Have

ID	Description
48	Handle 2000 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)

Will Not Have Right Now

ID	Description
49	Run on any Android operating systems before Android 14
50	Run on any iOS operating systems before iOS 17

Design

For this project I have 4 pieces that need to come together, I need a map on which to display Portsmouth and all the users, the rest of the front-end of the app, the database, and finally the back-end server to link the database to the front-end.

For the map I will use the OpenLayers JavaScript library as I have a lot of experience with it using it for 2 previous projects. It is also free and allows me to both display a map, which will be an OpenStreetMap map, as well as overlay markers and shapes onto said map. Also since this is a JavaScript library it is compatible with any framework through the use of a WebView.

For the rest of the front-end of the app, I will use Flutter as it allows me to develop both for iOS and Android whilst writing just one set of code. Flutter also has a very large set of libraries developed for it that will allow me to efficiently extend the functionality of the app. Finally, I have experience using Flutter to develop an App as I have previously used it to create my app University Bus Portsmouth.

The server will be hosted on AWS as I have experience using this tool and it allows me to efficiently create a database using Amazon RDS and create AWS lambda methods to add additional features. AWS also has an email service that I can utilise to send the login codes.

The database will be a MySQL database as this is the type that I am most familiar with and it has the capabilities that I require.

I have split the rest of the design of this project into 3 distinct sections: the UI and visual design; the systems architectural design; and finally the database design. Whilst these are presented as separate, the design was not done separately, instead an iterative approach was taken to the design when adding each feature. A part of the UI would be designed to display the feature to the user, the architectural design was then updated to support this feature, and finally the database was updated as necessary to complete the design of the feature from front end to back.

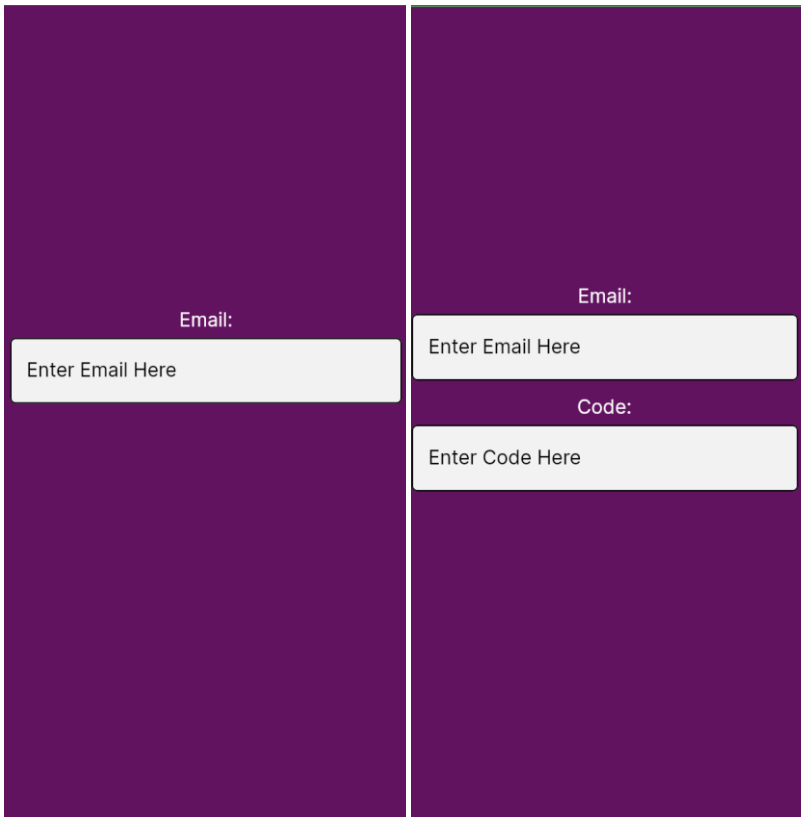
UI and Visual Design

Since this is an app for the University of Portsmouth, the aim is to use the colours of the university, which are primarily a dark purple and blue.

To create initial designs for the UI I will use Flutter Viz as it is a graphical tool meant for building Flutter UIs. Due to this the designs used in the project itself should be similar, but will have to be recreated out of Flutter code.

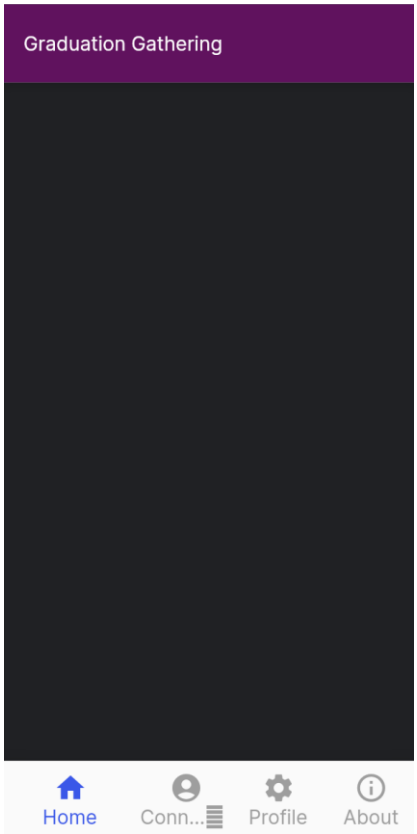
Login Screen

The login screen needs to allow users to enter their email and then enter the code sent to their email. So first I will display a text box and text field allowing users to enter their email. Then only once that is done I will display a box for the code to be entered. The background will be coloured in purple.



Main Screen

The main screen will be divided into three sections, the app bar at the top coloured in purple, the navigation bar at the bottom, and the content of the current screen in the middle.

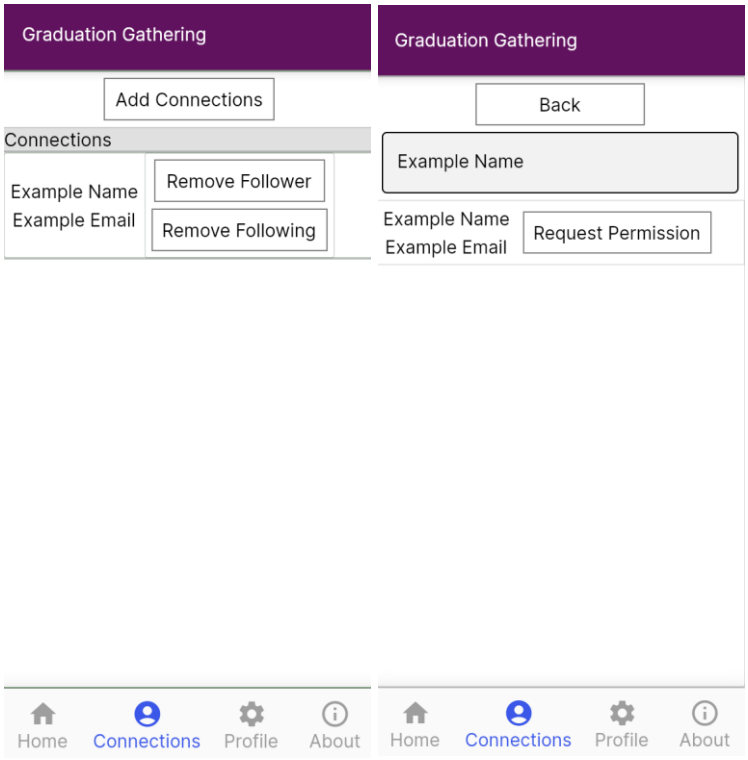


In this case since the current screen is the home screen which contains a Map at the centre which cannot be replicated in Flutter Viz so this Map has been left blank. The map displayed will be an Open Street Map with markers and shapes overlaid.

Connections

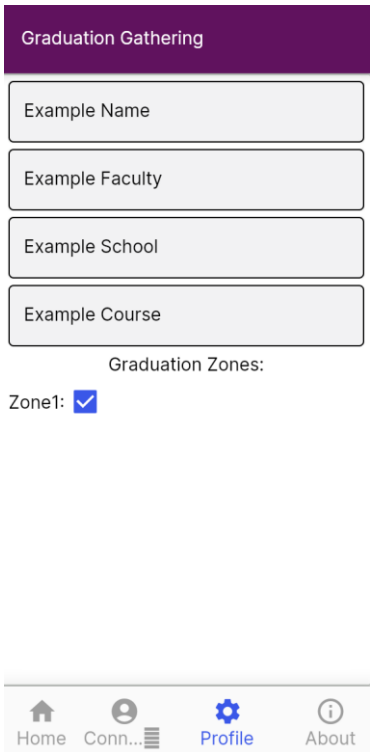
The connections screen should allow users to manage who has permission to see them, and who they have permission to see. The connections screen will first display a list of all the users that this user either has permission to see, or that has permission to see this user. I will also supply buttons next to each user in the list so the given user can manage whether they should have permission or not.

I will also have a button at the top that takes the user to a screen that will let them search for other users and request permission to locate those users.



Profile

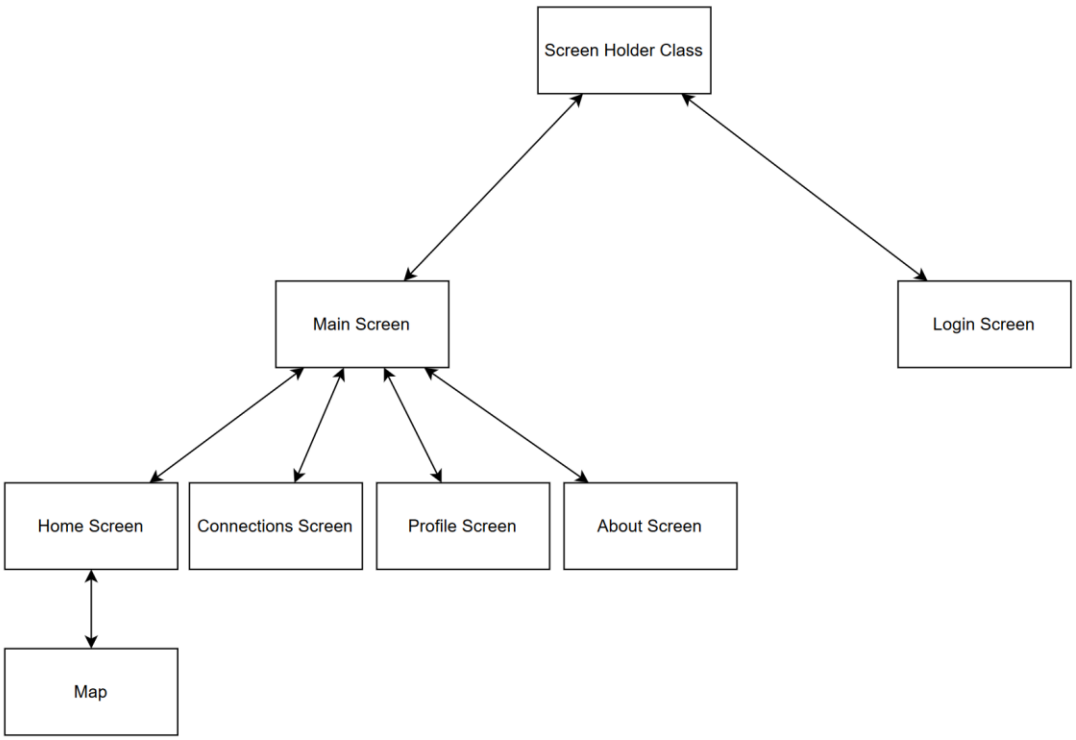
The profile screen should allow users to enter their name, faculty, school, and if applicable, course. It should also allow users to select which graduation zones they want to be seen in.



Architectural Design

The architecture can be split into 2 parts, the front-end and back-end. The back-end will be created out of a set of AWS lambda methods that will retrieve and edit data from the database. All of these lambda methods, except the login methods, will be authenticated with a JWT(json web token) that will be generated by the server when the user logs in. The use of JWTs will allow all requests sent to the server to be associated with the user that sent the request.

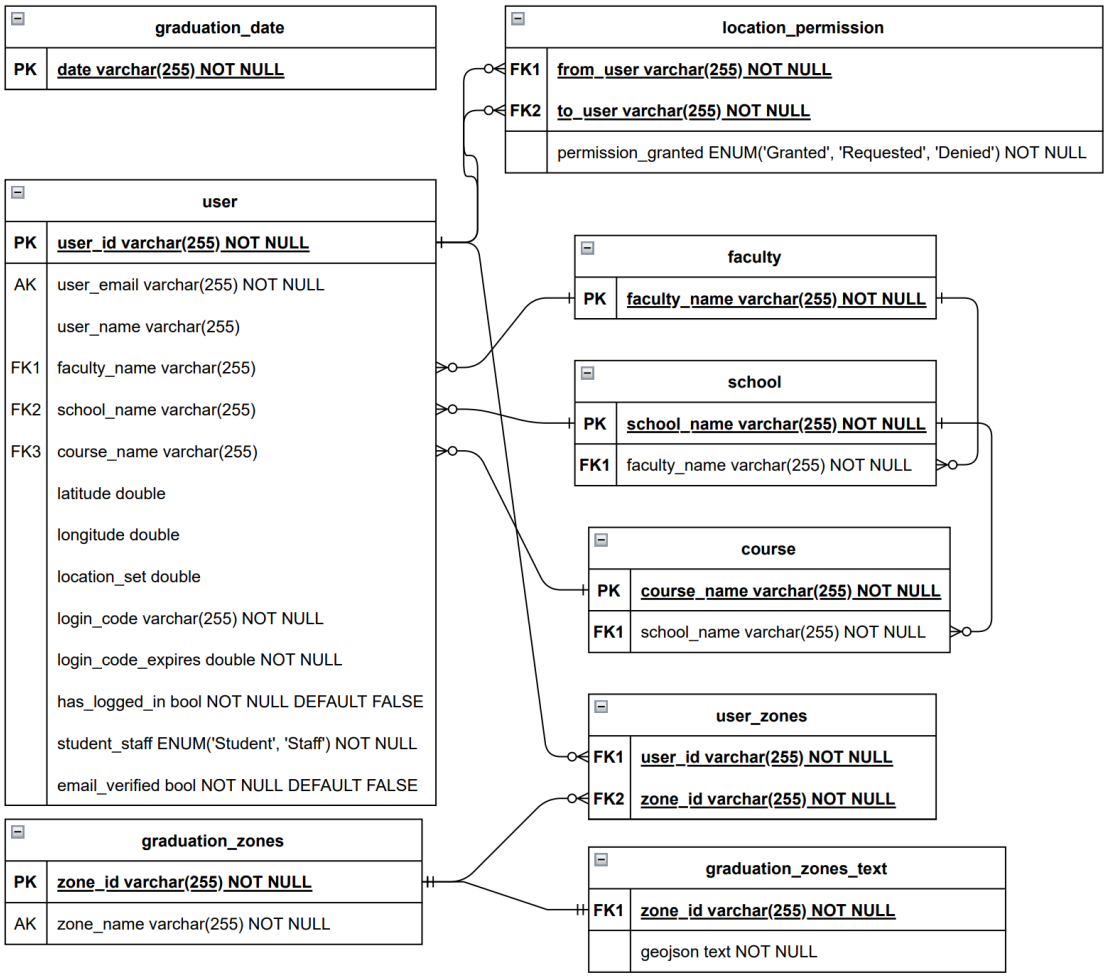
The front-end design will be based around the different screens as there will not be a significant part of the front end that is not directly working with the UI. The UI will be built by embedding screens(widgets) within other screens(widgets), this is standard for Flutter. However, for this project, all functionality will be attached to its relevant screen, and if its needed by multiple screens, then it is attached to the closest parent screen to both of the child screens.



Database

The database is a MySQL database hosted on AWS using the RDS. AWS was chosen for this project as they have a very reliable database service that I have used before whilst also allowing for back-end processing to be written and deployed efficiently using AWS lambda.

I used https://app.diagrams.net/ to create an entity relationship diagram for my database. There are five parts of this schema, 4 of which are linked by the user table, the location permission part, the course part, the graduation zones part, and finally the user table itself. The one other part is the Graduation Dates table which is on its own an just holds the days on which graduation ceremonies are held.



User Table

This table stores everything directly related to a user, which includes data such as their name and email address, but also their course, their last known latitude and longitude along with the time when that location was set, as well as their login code. When each user first enters their email address a unique user id is generated for them and that is set as the primary key in the database so that in the future if support for changing your email address is desired then it will be more feasible to add.

The has_logged_in row and the email_verified are separate as has_logged_in is used to direct the app to take the user to their profile if they have never edited it before and so remains false until the profile has been edited, whilst email_verified is false until the user correctly enters the code which will log them in for the first time.

Location Permissions

The location_permission table is used to give permission to see other users. When a user, let's say User A, first asks for permission from another user, User B, a new entry gets added to the location_permission table where from_user is the user id of User A, to_user is the user id of user B, and permission_granted is set to Requested. Then when user B accepts the request, the permission_granted row is changed to Granted. This now allows for User A to see the location of User B, but not the other way around. If User B wants to see User A then they must request

permission separately, at which point a new entry will be created in the location_permission table where from_user is the user id of User B, to_user is the user id of user A, and permission_granted is set to Requested.

This approach of creating separate entries in the location_permission table for when User A requests permission from User B and from the other way around allows for location permissions to be managed separately, as in it allows User A to see User B but not the other way around, whilst still keeping an efficient and easy to understand database design.

Course

The faculty, school and course tables store the academic structure of the university, all the courses on offer, linked to which school they are in, and linking each school to which faculty they are in.

The advantage of this is that when a course, school or even faculty is removed from the University of Portsmouth, its entry into its table can simply be deleted and its presence in the entries of the users who have added it to their profile and easily be deleted as well by cascade.

Graduation Zones

The actual geojsons for the graduation zones are stored in the graduation_zones_text table whilst the label for the zone is stored in the graduation_zones table. These are separate due the fact that the geojsons are too large to be indexed in a MySQL database, and so the solution to this problem is to store the zone id and label in a separate table that can then be indexed for fast retrieval. These zones are then linked to users using the user_zones table.

When a user saves to their profile that they want to appear in a given zone, then an entry in the user_zones table is created with both the users id and the zones id. When a user then saves that they no longer want to appear in a zone, that entry is then deleted. This allows for quick retrieval of which zones a user wants to appear in, not getting slowed down by the lack of indexing in the graduation_zones_text table.

Development

This section outlines the development process, technologies and implementation. This explains both the big picture decisions of how to structure the code and the implementation of individual methods. All screen shots are of the code and UI as it is at the end of development.

Overview of Technologies

The back-end is hosted on AWS utilising AWS lambda to provide server-side logic and AWS RDS to provide a database. AWS RDS was chosen as it is a fast reliable service that can be easily scaled to meet the needs of the application. AWS also allows for one free RDS instance and so I am able to use it for this project free of charge. The database is a MySQL database as it is reliable and easy to use, I also have previous experience writing SQL queries for MySQL databases. AWS Lambda was chosen as it is a serverless approach that allows for individual lambda methods to be created very quickly and efficiently. I also have experience creating lambda methods and I am able to use AWS Lambda free of charge up to 100,000 requests per month. Since 100,000 is far more requests than an individual would ever make, this allows for

the development of Graduation Gathering to not occur any charge. The lambda methods have been written in Python as my previous experience with AWS Lambda was using Python.

The front-end of the application was written in Dart using the Flutter framework. Flutter allows for development for both iOS and Android and so only one front-end had to be written instead of two. I have previous experience with Flutter development for iOS and Android.

To render and interact with the map the OpenLayers JavaScript package has been used. This retrieves and displays an OpenStreetMap map. I have already built a Flutter application that uses OpenLayers and so I will be reusing that code for this project. The reuse of this code saves time in both the implementation of the code and the testing.

Git has been used throughout the development of this project to back up the progress made and to allow for version control. In order to properly allow for version control the branch features of Git have been used. GitHub and GitHub Desktop have been used to make using Git easier and so more efficient.

To manually test and debug the front-end software Android emulators were used provided by Android Studio.

Sprint 1 – Project Start Up

The first sprint I undertook focused on creating an initial framework from which all other sprints could be based upon. By this I mean making a start in each area of the project and integrating each of the main technologies, packages and tools.

Task 1 – Flutter Project Creation

The first task was to create the Flutter project and set up the initial UI. The initial UI is the base for the main screen of the application, the screen that you see once logged in. It has a bar at the top displaying the words “Graduation Gathering”, a navigation bar at the bottom that allows you to select which screen you would like to look at, and finally a space in the middle of the screen to display the content of the screen selected by the navigation bar.

Task 2 – Navigation Bar

The next task was to add the logic for the navigation bar so that screens can be added efficiently in the future. This is handled in the file navigation_bar_items.dart through the class NavigationBarItems and the enum NavigationBarItemEnum. To add a new screen to the navigation bar first you add an enum for the screen that will hold the icon and name of the screen to be displayed on the navigation bar, as well as an integer that will hold the position in the navigation bar that the screen should be.

```
enum NavigationBarItemEnum {
  mapScreen(0, Icon(Icons.home), "Home"),
  manageUserPermissionsScreen(1, Icon(Icons.person), "Connections"),
  profileScreen(2, Icon(Icons.settings), "Profile"),
  aboutScreen(3, Icon(Icons.info_outline), "About");

  const NavigationBarItemEnum(this.position, this.icon, this.label);

  final int position; // The position in the nav bar, starting from 0.
  final Icon icon;
  final String label;
}
```

After adding the screen as an enum you then instantiate the screen within the constructor of NavigationBarItems and add the screen to the list of screens as shown below. The main screen then gets all the child screens from the NavigationBarItems class and automatically adds them to the navigation bar and makes it so that the screen appears when the user clicks on the icon for the screen from the navigation bar. This setup makes it very efficient to add new screens and allows the screens to be managed easily.

```
NavigationBarItems(
    AuthToken authToken,
    ProfileSettings profile,
    AcademicStructure structure,
    GradZones zones,
    Function() logoutFunction,
    Connections connections,
    OtherUserProfiles otherUserProfiles,
    GraduationDates graduationDates,
    {MainMapWidget? mainMapWidget}) {
  for (GradZone zone in profile.getUserGradZones()) {
    zones
        .getZoneFromId(zone.getId())
        ?.setColour(ZoneColours.blue.getColourRGB());
  }
  mapScreen = Tuple2(
      NavigationBarItemEnum.mapScreen,
      MapScreen(
          authToken: authToken,
          allGradZones: zones,
          usersGradZones: profile.getUserGradZones(),
          mainMapWidgetStateKey: mainMapWidgetStateKey,
          mainMapWidget: mainMapWidget,
          graduationDates: graduationDates));  // MapScreen, Tuple2
  manageUserPermissionsScreen = Tuple2(
      NavigationBarItemEnum.manageUserPermissionsScreen,
      ConnectionsScreen(
          authToken: authToken,
          connections: connections,
          otherUserProfiles: otherUserProfiles,
          academicStructure: structure,
          userProfile: profile));  // ConnectionsScreen, Tuple2
  profileScreen = Tuple2(
      NavigationBarItemEnum.profileScreen,
      ProfileScreen(
          authToken: authToken,
          profile: profile,
          academicStructure: structure,
          allGradZones: zones,
          logoutFunction: logoutFunction,
          mainMapWidgetStateKey: mainMapWidgetStateKey));  // ProfileScreen, Tuple2
  aboutScreen =
      const Tuple2(NavigationBarItemEnum.aboutScreen, AboutScreen());
  _itemsInOrder.add(mapScreen);
  _itemsInOrder.add(manageUserPermissionsScreen);
  _itemsInOrder.add(profileScreen);
  _itemsInOrder.add(aboutScreen);
}
```

Task 3 – Map Screen Creation

After this, the next task was to add the map screen to the application. I reused code I wrote for an earlier project here and so this was done quickly and efficiently. The MapScreen class creates the map widget and allows for other widgets to appear in the map screen apart from just the map.

```
  @override
  initState() {
    if (widget.mainMapWidget == null) {
      _mapWidget = MainMapWidget(
        key: widget.mainMapWidgetStateKey,
        authToken: widget.authToken,
        allGradZones: widget.allGradZones,
        usersGradZones: widget.usersGradZones,
        markerClickedFunction: (String markerId) {
          setState(() {});
        },
        graduationDates: widget.graduationDates
      );
    }
    else
    {
      _mapWidget = widget.mainMapWidget!;
    }
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: false,
      body: Container(
          margin: const EdgeInsets.all(0),
          padding: const EdgeInsets.all(0),
          width: MediaQuery.of(context).size.width,
          decoration: BoxDecoration(
            color: const Color(0x1f000000),
            shape: BoxShape.rectangle,
            borderRadius: BorderRadius.zero,
            border: Border.all(color: const Color(0x4d9e9e9e), width: 1),
          ),  // BoxDecoration
          child: _mapWidget),  // Container
    );  // Scaffold
  }
}
```

The MapWidget abstract class is used to interact with the OpenLayers JavaScript library. MapWidget loads the html file map.html which contains the JavaScript to interact with the OpenLayers API.

```
<html lang="en">
<head>

    <meta charset="UTF-8">
    <title>Map</title>

    <!-- OpenLayers v8.2.0 (https://openlayers.org/) -->
    <script src="ol/dist/ol.js"></script>

    <link rel="stylesheet" href="ol/ol.css">
</head>

<body>
<div id="map" class="map"></div>
<script>
    let map;
    let tileLayer;
    let IDLayersMap = new Map();
    // Enum to identify the type of layer
    let layerEnum = {
        markerLayer: 'markerLayer',
        geoJsonLayer: 'geoJsonLayer'
    };

    /*
    Map
    */

    map = new ol.Map({
        view: new ol.View({
            center: new ol.proj.fromLonLat([1, 1]),
            zoom: 1,
        }),
        target: 'map',
    });

    // Detects if a marker has been clicked
    map.on("click", function (e) {
        let markerFound = false;
        map.forEachFeatureAtPixel(e.pixel, function (feature, layer) {
            if (layer.isClickableMarkerLayer && markerFound == false) {
                markerClicked(feature);
                markerFound = true;
            }
        })
    });
```

```
// adds a tile layer with the given tile server.
function addOSMTileServer(server)
{
    tileLayer = new ol.layer.Tile({
        source: new ol.source.OSM({
            attributions: [
          server.attribution,
          ol.source.OSM.ATTRIBUTION,
        ],
        url:
          server.url,
      }),
    });
    map.addLayer(tileLayer);
}

// Sets the centre of the map and the zoom
function setCentreZoom(value) {
    map.getView().setZoom(value.zoom);
    map.getView().setCenter(ol.proj.fromLonLat([value.long, value.lat]));
}



// Creates a marker layer
function createMakerLayer(layer)
{
    var markerLayer = new ol.layer.Vector({
        source: new ol.source.Vector(),
        style: new ol.style.Style({
            image: new ol.style.Icon({
                anchor: [layer.anchorX, layer.anchorY],
                anchorXUnits: 'fraction',
                anchorYUnits: 'fraction',
                scale: [layer.markerSize, layer.markerSize],
                src: layer.image,
            }),
        }),
    });
    markerLayer.isClickableMarkerLayer = layer.markersClickable;
    map.addLayer(markerLayer);
    if (IDLayersMap.get(layer.layerId) == null)
    {
        createLayerGroup(layer.layerId);
    }
    var layerGroup = IDLayersMap.get(layer.layerId);
    layerGroup.set(layerEnum.markerLayer, markerLayer);
}
```

```
// Creates a geo json layer
function createGeoJsonLayer(layer)
{
    var geoJsonLayer = new ol.layer.VectorImage({
        source: new ol.source.Vector({
            format: new ol.format.GeoJSON(),
        }),
        style: new ol.style.Style({
            stroke: new ol.style.Stroke({
                color: `rgb(${layer.colour.r}, ${layer.colour.g}, ${layer.colour.b})`,
                width: 8,
            }),
            fill: new ol.style.Fill({
                color: `rgba(${layer.colour.r}, ${layer.colour.g}, ${layer.colour.b}, 0.1)`,
            }),
        }),
    });
    map.addLayer(geoJsonLayer);
    if (IDLayersMap.get(layer.layerId) == null)
    {
        createLayerGroup(layer.layerId);
    }
    var layerGroup = IDLayersMap.get(layer.layerId);
    layerGroup.set(layerEnum.geoJsonLayer, geoJsonLayer);
}

// Creates the layer group
function createLayerGroup(id)
{
    IDLayersMap.set(id, new Map());
}



// Adds the markers
function addMarker(markedFeature) {
    const layer = IDLayersMap.get(markedFeature.layerId).get(layerEnum.markerLayer);
    const marker = new ol.Feature(new ol.geom.Point(ol.proj.fromLonLat([markedFeature.longitude, markedFeature.latitude])));
    marker.setId(markedFeature.id);
    layer.getSource().addFeature(marker);
}

// Removes the marker
function removeMarker(markedFeature) {
    const layer = IDLayersMap.get(markedFeature.layerId).get(layerEnum.markerLayer);
    const source = layer.getSource();
    source.removeFeature(source.getFeatureById(markedFeature.id));
}
```

```
// Updates the location of a marker
function updateMarker(markedFeature) {
    removeMarker(markedFeature);
    addMarker(markedFeature);
}

// Clears a marker layer
function clearMarkerLayer(layerId)
{
    const layer = IDLayersMap.get(layerId.layerId).get(layerEnum.markerLayer);
    layer.getSource().clear();
}

// Toggles the visibility of the markers
function toggleShowLayers(visible) {
    const layers = Array.from(IDLayersMap.get(visible.layerId).values());
    layers.forEach(function (layer) {
        layer.setVisible(visible.visible);
    })
}

// Marker interaction
function markerClicked(marker) {
    MarkerClickedDart.postMessage(marker.getId());
}

// Adds a geojson default colour
function addGeoJson(geoJsonFeature) {
    const layer = IDLayersMap.get(geoJsonFeature.layerId).get(layerEnum.geoJsonLayer);
    drawGeoJsonLines(geoJsonFeature, layer);
}

// Adds a geojson with given colour
function addGeoJsonWithColour(geoJsonFeature) {
    const layer = IDLayersMap.get(geoJsonFeature.layerId).get(layerEnum.geoJsonLayer);
    drawGeoJsonLinesWithColour(geoJsonFeature, layer);
}

// Clears a geojson layer
function clearGeoJsonLayer(layerId)
{
    const layer = IDLayersMap.get(layerId.layerId).get(layerEnum.geoJsonLayer);
    layer.getSource().clear();
}
```

```
// Add the given GeoJson to the map
function drawGeoJsonLines(geoJson, layer) {
    layer.getSource().addFeatures((new ol.format.GeoJSON({
        featureProjection: 'EPSG:3857',
        dataProjection: 'EPSG:4326'
    })).readFeatures(geoJson.geoJson));
}

// Add the given GeoJson to the map
function drawGeoJsonLinesWithColour(geoJson, layer) {
    var features = (new ol.format.GeoJSON({
        featureProjection: 'EPSG:3857',
        dataProjection: 'EPSG:4326'
    })).readFeatures(geoJson.geoJson);
    features.forEach(function (feature) {
        feature.setStyle(new ol.style.Style({
            stroke: new ol.style.Stroke({
                color: `rgb(${geoJson.colour.r}, ${geoJson.colour.g}, ${geoJson.colour.b})`,
                width: 8,
            }),
            fill: new ol.style.Fill({
                color: `rgba(${geoJson.colour.r}, ${geoJson.colour.g}, ${geoJson.colour.b}, 0.1)`,
            }),
        }));
    });
    layer.getSource().addFeatures(features);
}

function isPointInsidePolygons(point, geojsons) {
    if (geojsons.length == 0)
    {
        return false;
    }
    for (var geoJsonIndex in geojsons) {
        const geoJson = geojsons[geoJsonIndex];
        const polygons = (new ol.format.GeoJSON({
            featureProjection: 'EPSG:3857',
            dataProjection: 'EPSG:4326'
        })).readFeatures(geoJson.geoJson);
        for (var polygonIndex in polygons) {
            if (!polygons[polygonIndex].getGeometry().intersectsCoordinate(new ol.proj.fromLonLat([point.longitude, point.latitude]))) {
                return false;
            }
        }
    };
    return true;
}

</script>
</body>
```

To render a map on the screen a child class must be created that extends MapWidget. For Graduation Gathering, the child class used to create the map is MainMapWidget. Whilst MapWidget, MainMapWidget and the html file called map.html were all originally used for a different project, they have been edited to better serve their purpose within Graduation Gathering.

```dart
// The tile server from which the map should be retrieved.
final TileServer _tileServer = TileServer("https://tile.openstreetmap.org/{z}/{x}/{y}.png", "")

/// Creates the webview with the map.
@override
void initState() {
  _webViewController = WebViewController()
    ..setJavaScriptMode(JavaScriptMode.unrestricted)
    ..setBackgroundColor(const Color(0x00000000))
    ..setNavigationDelegate(
      NavigationDelegate(
        onProgress: (int progress) {
          // Update loading bar.
        },
        onPageStarted: (String url) {},
        onPageFinished: (String url)
        async {
          await _addTileServer();
          await createLayers();
          onPageFinished?.call(url);
        },
        onWebResourceError: (WebResourceError error) {},
        onNavigationRequest: (NavigationRequest request) {
          return NavigationDecision.navigate;
        },
      ),
    )
    ..addJavaScriptChannel('MarkerClickedDart',
        onMessageReceived: (JavaScriptMessage markerIdMessage) {
          _markerClicked(markerIdMessage.message);
        })
    ..loadFlutterAsset(_mapPath);
  _webView = WebViewWidget(controller: _webViewController);
  super.initState();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: _webView,
  );
}

/// Allows the creation of layers upon the map.
/// Must be called before the map itself is loaded.
@protected
createLayers();

/// Adds the tile server to the map.
_addTileServer()
{
  String jsObject = "{url: '${_tileServer.url}', attribution: '${_tileServer.attribution}'}";
  _webViewController.runJavaScript("addOSMTileServer($jsObject)");
  widget.pingTileServerFunction?.call(_tileServer.getUrlDomains()!);
}
```

```dart
/// Centres and zooms the map around the given lat, long and zoom.
@protected
setMapCentreZoom(double lat, double long, double zoom) {
  String jsObject = "{lat: $lat, long: $long, zoom: $zoom}";
  _webViewController.runJavaScript("setCentreZoom($jsObject)");
}

/// Adds the markers.
@protected
createMakerLayer(String layerId, String image, double size, double anchorX, double anchorY, bool markersClickable)
{
  String jsObject =
      "{layerId: '$layerId', image: '$image', markerSize: '$size', anchorX: $anchorX, anchorY: $anchorY, markersClickable: $markersClickable}";
  _webViewController.runJavaScript("createMakerLayer($jsObject)");
}

/// Adds the markers.
@protected
createGeoJsonLayer(String layerId, Colour colour, int width)
{
  String jsObject =
      "{layerId: '$layerId', colour: {r: ${colour.red}, g: ${colour.green}, b: ${colour.blue}}, width: $width}";
  _webViewController.runJavaScript("createGeoJsonLayer($jsObject)");
}

/// Adds a marker.
@protected
addMarker(String layerId, String id, double long, double lat)
{
  String jsObject =
      "{layerId: '$layerId', id: '$id', longitude: $long, latitude: $lat}";
  _webViewController.runJavaScript("addMarker($jsObject)");
}

/// Adds a marker.
@protected
removeMarker(String layerId, String id)
{
  String jsObject =
      "{layerId: '$layerId', id: '$id'}";
  _webViewController.runJavaScript("removeMarker($jsObject)");
}

/// Updates the position of the marker.
@protected
updateMarker(String layerId, String id, double long, double lat) async {
  String jsObject =
      "{layerId: '$layerId', id: '$id', longitude: $long, latitude: $lat}";
  _webViewController.runJavaScript("updateMarker($jsObject)");
}

/// Toggles the visibility of the U1 bus stop maon the map.
toggleMarkers(String layerId, bool visible) async {
  String jsObject = "{layerId: '$layerId', visible: $visible}";
  _webViewController.runJavaScript("toggleShowLayers($jsObject)");
}
```

```dart
/// Adds the geo json.
@protected
addGeoJson(String layerId, String geoJson)
{
  String jsObject = "{layerId: '$layerId', geoJson: '$geoJson'}";
  _webViewController.runJavaScript("addGeoJson($jsObject)");
}

/// Adds the geo json with given colour.dart.
@protected
addGeoJsonWithColour(String layerId, String geoJson, Colour colour)
{
  String jsObject = "{layerId: '$layerId', geoJson: '$geoJson', colour: {r: ${colour.red}, g: ${colour.green}, b: ${colour.blue}}}";
  _webViewController.runJavaScript("addGeoJsonWithColour($jsObject)");
}

/// Clears the all the markers from a layer.
@protected
clearMarkerLayer(String layerId)
async {
  String jsObject = "{layerId: '$layerId'}";
  await _webViewController.runJavaScript("clearMarkerLayer($jsObject)");
}

/// Clears the geoJson layer.
@protected
clearGeoJsonLayer(String layerId)
{
  String jsObject = "{layerId: '$layerId'}";
  _webViewController.runJavaScript("clearGeoJsonLayer($jsObject)");
}

/// Checks if the given point is inside the given zone.
@protected
Future<bool> isPointInsideGeojson(double long, double lat, GradZones zones) async
{
  if (zones.isEmpty)
  {
    return false;
  }
  List<Map<String, dynamic>> zonesGeojsons = zones.geojsonsAsList();
  List<Map<String, Map<String, dynamic>>> zonesInJsForm = [];
  for (Map<String, dynamic> zoneGeojson in zonesGeojsons) {
    zonesInJsForm.add({"geoJson": zoneGeojson});
  }
  String jsObjectPoint = "{longitude: $long, latitude: $lat}";
  String jsObjectGeojsons = json.encode(zonesInJsForm);
  return await _webViewController.runJavaScriptReturningResult("isPointInsidePolygons($jsObjectPoint, $jsObjectGeojsons)") as bool;
}

// This is called when a marker on the map gets clicked.
_markerClicked(String markerId) {
  if (widget.markerClickedFunction == null) {
    return;
  }
  widget.markerClickedFunction!(markerId);
}
}
```

All communication between Dart and JavaScript is done within the MapWidget class, MainMapWidget only calls methods from MapWidget. This encapsulation simplifies the process of interacting with the map from the perspective of MainMapWidget and so makes it easier to add additional features.

The map is interacted with by adding layers that are rendered on top of the map, markers and shapes can then be added to these layers. You must specify an ID for these layers that can be used later to interact with the layer. The layer ID's on the Flutter side are stored in the MapDataId enum.

```
// Assigns an id to each layer used by this map to be referenced later.
@override
createLayers() {
  createGeoJsonLayer(MapDataId.zones.idPrefix, Colour(0, 0, 255), 8);
  createMakerLayer(MapDataId.otherUsers.idPrefix, "test.png", 0.2, 0.5, 1, true);
  createMakerLayer(MapDataId.userLocation.idPrefix, "UserIcon.png", 0.1, 0.5, 0.5, false);
}
```

```
/// Defines the ids for the map layers.
enum MapDataId
{
  userLocation("UL-"),
  otherUsers("OU-"),
  zones("GZ-");

  const MapDataId(this.idPrefix);
  final String idPrefix;

  /// Returns the MapDataId enum that corresponds to the given id.
  static MapDataId getMapDataIdEnumFromId(String id)
  {
    for (MapDataId mapDataId in MapDataId.values)
    {
      if (id.startsWith(mapDataId.idPrefix))
      {
        return mapDataId;
      }
    }
    throw MapDataIdNotFoundException("Id not found.");
  }
}
```

The user's location is shown on the map through a marker. Whenever LocationHandler gets an update on users location, it then calls the method given to it by MainMapWidget to update the location of the marker on the map. This method also sends a request to the server to update the user's location for the other users of the app.

```
// Adds the users location to the map as a marker and sets for it be updated whenever the user moves.
// Also sends the users location to the server when the user is within one of their chosen zones.
_addUserLocationIcon() {
  LocationHandler handler = LocationHandler.getHandler();
  handler.onLocationChanged((location_data.LocationData currentLocation) async {
    if (!widget.graduationDates.isGraduationDayToday())
    {
      return;
    }
    updateMarker(MapDataId.userLocation.idPrefix, MapDataId.userLocation.idPrefix,
        currentLocation.longitude!, currentLocation.latitude!);
    if (await isPointInsideGeojson(currentLocation.longitude!, currentLocation.latitude!, widget.usersGradZones)) {
      _sendLocationData.send(
          Location(currentLocation.longitude!, currentLocation.latitude!));
    }
  });
}
```

The locations of the other users that the current user has permission to see is updated by setting a repeated timer which fires every 10 seconds. It sends a request to the server asking for the other users locations and then loops through them adding each one to the map as a marker.

```
// creates a repeating function to retrieve the other users locations.
_setUpGetOtherUsersLocations() {
  const dur = Duration(seconds:10);
  Timer.periodic(dur, (Timer t) => _getOtherUsersLocations());
}

// Adds the other users markers to the map.
_getOtherUsersLocations() async {
  if (!widget.graduationDates.isGraduationDayToday())
  {
    return;
  }
  _addOtherUsersMarkers(await _getOtherUsersLocation.send());
}

// Updates the other users markers by removing all markers that are not in the
// list given, and checking that all the given markers are in the graduation zones.
_addOtherUsersMarkers(List<dynamic> users) async {
  await clearMarkerLayer(MapDataId.otherUsers.idPrefix);
  for (Map<String, dynamic> user in users)
  {
    if (await isPointInsideGeojson(user["longitude"], user["latitude"], widget.allGradZones)) {
      updateMarker(MapDataId.otherUsers.idPrefix, user["email"],
          user["longitude"], user["latitude"]);
    }
  }
}
```

Task 4 – Create Server and Framework For Calling the Servers Endpoints

Here I created the AWS RDS instance and the initial database schema. I also created initial AWS lambda methods to test the interaction with the database. These Lambda Methods are then exposed using AWS API Gateway.

The lambda method CreateAndAddToDatabase was created to manage the database schema as well as insert any data I needed to directly into the database. This method is not exposed by the API Gateway and can only be accessed using the AWS management console.

A blueprint for adding tables was created, for example see the SQL for the faculty table below.

```
# Gets the faculty table sql
def get_faculty_table_sql():
    return "create table if NOT EXISTS faculty ( faculty_name varchar(255) NOT NULL, PRIMARY KEY (faculty_name))"
```

Data could also be added directly to the database as shown below for the faculty table.

```
faculty_sql_string = 'insert into faculty (faculty_name) values("{FacultyName}")'
```

```
for faculty in FacultyData: # Adds the faculty data to the database.
    try:
        cur.execute(faculty_sql_string.format(FacultyName = escape_sql_string(faculty['faculty_name'])))
        item_count += 1
    except pymysql.MySQLError as e:
        logger.error(e)
    conn.commit()
```

All lambda methods that access the database and interact with the database by giving it strings generated by the user contain a method to clean the string so that is cannot be exploited by these users through SQL injection attacks.

```
# Escapes a string to be given to the database to protect the database.
def escape_sql_string(sql_string):
    translate_table = str.maketrans({"]": r"\]", "\\": r"\\",
                                     "^": r"\^", "$": r"\$", "*": r"\*", "'": r"\'", '"': r'\"'})
    if (sql_string is None):
        return sql_string
    return sql_string.translate(translate_table)
```

The main part of this task was to create a framework in Flutter for accessing the lambda methods in the server.

The design that I settled on is the SendRequest class which stores the server URL and has methods to send both GET and POST requests to the server. The way the system works is that, if you have an endpoint you want the Flutter app to reach, you create a child class of Send Request and in the getRoute() method that you must create, you return the route with the server to the endpoint that you want to hit. Then you create a method within this class that calls either post or get within SendRequest to actually send the request to the endpoint at the given route with the server. See below both SendRequest and SendLocationData as an example of how to extend SendRequest to hit a specific endpoint.

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

/// Sends a request to the server.
abstract class SendRequest {

  // Server URL as String object.
  final String _serverURL = "https://058sjjvnef.execute-api.eu-west-2.amazonaws.com/";

  // Combines the server URL and endpoint route into a URI object.
  Uri _getUri() {
    Uri uri = Uri.parse(_serverURL + getRoute());
    return uri;
  }

  /// Returns the route within the server to the endpoint.
  @protected
  getRoute();

  /// Sends a post request to the server.
  @protected
  Future<String> post(String body, {Map<String, String>? headers}) async {
    Uri uri = _getUri();
    http.Response response = await http.post(uri, headers: headers, body: body);
    return response.body;
  }

  /// Sends a get request to the server.
  @protected
  Future<String> get(Map<String, String>? headers) async {
    Uri uri = _getUri();
    http.Response response = await http.get(uri, headers: headers);
    return response.body;
  }
}
```

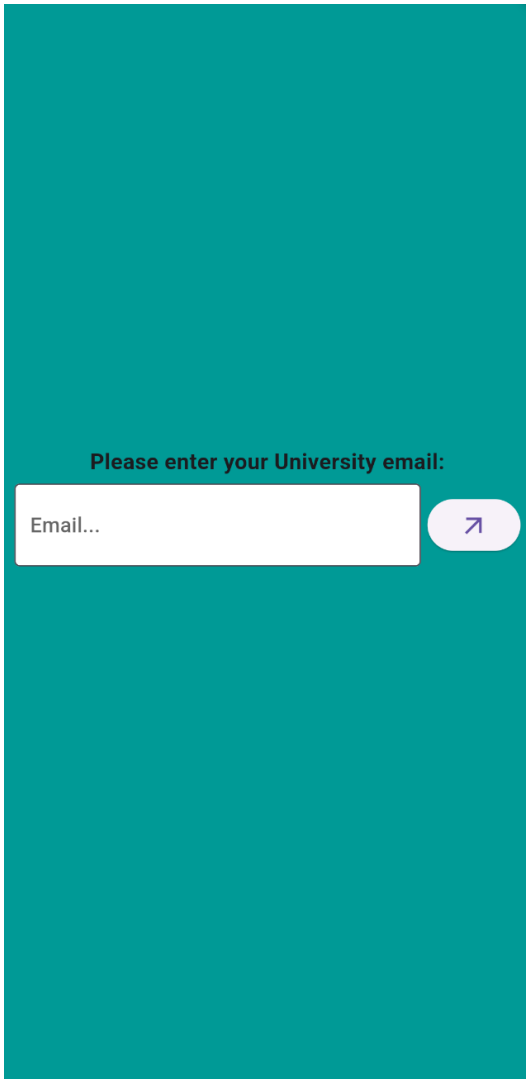
```dart
/// Sends requests to the server to set your location.
class SendLocationData extends SendRequest {

  // The auth token to authenticate the request.
  final AuthToken _token;

  /// Creates the object with the token to authenticate the request.
  SendLocationData(this._token);

  /// Sends the request.
  send(Location location) {
    Map<String, String> headers = {"Authorization": _token.getToken()};
    Map<String, Map<String, double>> bodyJson =
      {"location": {"lat": location.getLatitude(), "long": location.getLongitude()}};
    String body = json.encode(bodyJson);
    post(body, headers: headers);
  }

  /// Returns the route within the server to the setLocation endpoint.
  @override
  getRoute() {
    return "setLocation";
  }
}
```

Task 5 – Login and Authentication

To secure the server from unauthorised access, JWTs (Json Web Tokens) are used to authenticate every request to the server that is not used to login. AWS API Gateway makes it very easy to attach a lambda method that authenticates requests before the exposed endpoint is called. Using this feature I created the Auth lambda method which validates a JWT.

```
key = os.environ['key']

# Allows for AWS Logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context): # Entry point for AWS.
    response = {
        "isAuthorized": False,
        "context": {
            "stuff": None,
        }
    }
    try:
        token = event["headers"]["authorization"] # Retrieves the auth token (String) passed with the request.
    except:
        return response

    decodedToken = decodeToken(token) # Turns the token String into either a Map<String, dynamic> (The token in json form) or returns invalid token.
    validToken = validateToken(decodedToken)

    if validToken:
        response["isAuthorized"] = True
        response["context"]["email"] = getEmail(decodedToken)
        response["context"]["userID"] = getUserID(decodedToken)
    return response

# Turns the token String into either a Map<String, dynamic> (The token in json form) if it was made by the server, or returns invalid token.
def decodeToken(token):
    secret = get_secret()
    try:
        decodedToken = jwt.decode(token, secret, algorithms="HS256")
    except:
        decodedToken = "Invalid Token"
    return decodedToken

# Checks if a token is valid by checking if it has expired.
def validateToken(token):
    if token == "Invalid Token":
        return False
    return getExpires(token) >= time.time()

# Returns the secret for signing and validating the tokens.
def get_secret():
    return key

# Returns the email contained in a token.
def getEmail(token):
    return token["email"]

# Returns the user id contained in a token.
def getUserID(token):
    return token["id"]

# Returns the expire time of the token.
def getExpires(token):
    return token["expires"]
```

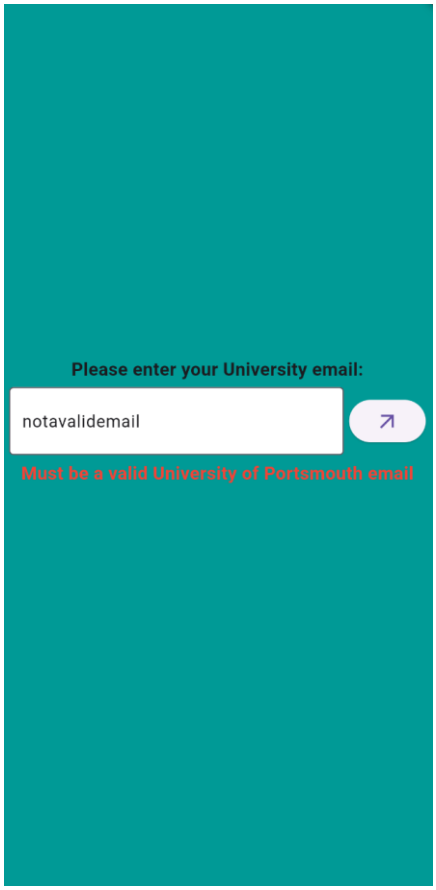
Users must also be able to obtain these tokens when they login. So I created a login screen in Flutter that allows users to enter their University of Portsmouth email. This screen is coloured in the light blue colour of the University of Portsmouth.



This email is parsed to check if it is a valid University of Portsmouth email. This is done in the ParseEmail class which uses a regular expression.

```dart
/// Checks if a given email is a valid student or staff email for the University
/// of Portsmouth.
class ParseEmail
{
  // Regex for the start of an email (every thing before the @) in String form.
  final String _startOfEmail =
      '(?:[a-z0-9!#\$%&\'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#\$%&\'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")';
  // Regex object for a student email.
  late final RegExp _studentEmail;
  // Regex object for a staff email.
  late final RegExp _staffEmail;

  ParseEmail()
  {
    _studentEmail = RegExp('$_startOfEmail@myport.ac.uk');
    _staffEmail = RegExp('$_startOfEmail@port.ac.uk');
  }

  /// Validates a given email by whether it's a valid student or staff email for
  /// the University of Portsmouth
  bool validate(String email)
  {
    email = email.trim();
    return _studentEmail.hasMatch(email) || _staffEmail.hasMatch(email);
  }
}
```

If the email fails the parse then the screen asks again for an email from the University of Portsmouth.



If the email passes the parse, then this email is then sent to the SendEmailCode lambda method. This lambda method checks if the email is a valid University of Portsmouth email, then checks if an account for this email exists in the database, if not then it creates an entry in the database for this email, then creates a login code for the account and saves that to the database, then sends a request to AWS SES (Simple Email Service) to send the login code in an email to the email address given.

This feature does not fully work. Emails are not sent most of the time. AWS SES has put my account in sandbox mode, meaning that I can only send emails to email addresses that are pre-verified. To get out of sandbox mode I would need to prove I am a organisation with more resources than I currently have. I looked into alternative mail providers such as MailGun and Postmark, however all alternatives were either not free or had the same requirements as AWS.

To check if the email address is valid, it uses the same regular expression as the Flutter App. This check is done on the client side to stop unnecessary requests being sent to the server, and its done on the server side for security so people cannot call the endpoint directly outside of the Flutter App and enter any email address they want.

```
# Checks whether the given email is a valid student or staff email for the University of Portsmouth.
def validEmail(email):
    startOfRegEmail = '(?:[a-z0-9!#\$%&\'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#\$%&\'*+/=?^_`{|}~-]+)*|"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])*")'
    studentReg = startOfRegEmail  + '@myport.ac.uk'
    staffReg = startOfRegEmail + '@port.ac.uk'
    return re.search(studentReg, email) or re.search(staffReg, email)
```

Accounts are checked if they exist by trying to find the userID from the entry with the given email. This ID is the primary key in the database instead of the users email so that in the future it is possible to add a feature that allows users to change their email address.

```
# Returns the user Id from the database for the given email.
def getUserID(email):
    with conn.cursor() as cur:
        cur.execute(f"select user_id from user where user_email = '{email}'")
        userID = None
        for row in cur:
            userID = row[0]
        cur.close()
    conn.commit()
    return userID
```

If no user ID is found then a new unique user ID is generated for this email. It is guaranteed to be unique by then checking if the generated user ID already exists in the database, if it does then a new user ID is generated.

```
# Creates a potential user ID.
def generateUserID():
    codeList = [str(random.randint(0,9)) for _ in range(10)]
    return "US-" + "".join(codeList)
```

```
# Creates a new unique user ID.
def createUserID():
    with conn.cursor() as cur:
        uniqueCodeCreated = False
        while uniqueCodeCreated == False:
            userID = generateUserID()
            cur.execute(f"select user_id from user where user_id = '{userID}'")
            id = None
            for row in cur:
                id = row[0]
            if id == None:
                uniqueCodeCreated = True
        cur.close()
    conn.commit()
    return userID
```

Next a login code is generated for the user.

```
# Creates a 5 digit login code (Number from 0 - 99999 but always with padded zeros to make it 5 digits)
def generateCode():
    codeList = [str(random.randint(0,9)) for _ in range(5)]
    return "".join(codeList)
```

The code is also given an expiration time as 5 minutes from the current time.

```
# Gets when the code should expire (24 hours from now) as seconds since unix epoch
def getCodeExpireTime():
    return time.time() + (5 * 60)
```

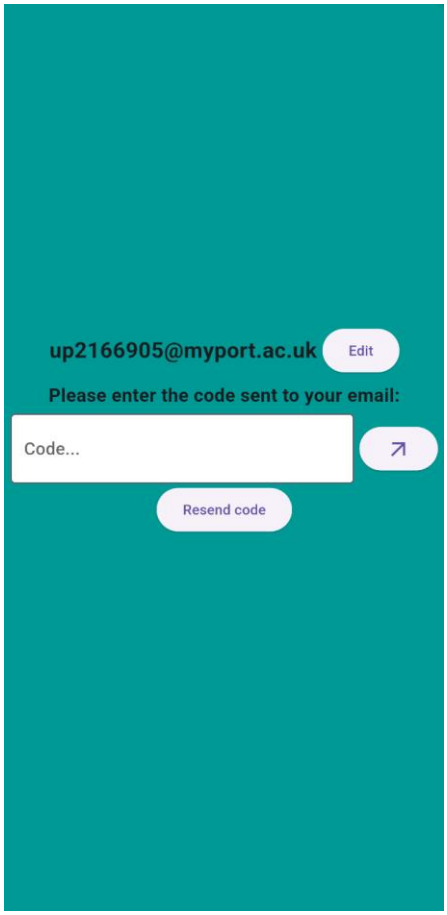
Then the code and expiration time is written to the database, if an account for this email does not exist then the user ID generated and the email are also written to the database.

```
# Creates a new entry in the database for this user and adds the login code.
def writeCodeToRDSWIthoutUserID(email, code):
    userID = createUserID()
    expires = getCodeExpireTime()
    studentStaff = getStudentStaff(email)
    with conn.cursor() as cur:
        try:
            sql_string = 'insert into user (user_id, user_email, login_code, login_code_expires, student_staff) values("{userID}", "{userEmail}", "{loginCode}", {expires}, "{studentStaff}")'
            cur.execute(sql_string.format(userID = escape_sql_string(userID), loginCode = escape_sql_string(code), expires = expires, userEmail = escape_sql_string(email), studentStaff = escape_sql_string(studentStaff)))
        except pymysql.MySQLError as e:
            logger.error(e)
        conn.commit()

# Adds the code to the database for the existing entry for the user.
def writeCodeToRDSWIthUserID(userID, code):
    expires = getCodeExpireTime()
    with conn.cursor() as cur:
        try:
            sql_string = 'update user set login_code = "{loginCode}", login_code_expires = {expires} where user_id = "{userID}"'
            cur.execute(sql_string.format(loginCode = escape_sql_string(code), expires = expires, userID = escape_sql_string(userID)))
        except pymysql.MySQLError as e:
            logger.error(e)
        conn.commit()
```

Finally an email is sent to the given email address with the code generated.

```
client = boto3.client('ses', region_name='eu-west-2')

# Sends a request to AWS SES to send an email containing the login code.
def sendEmail(receiver_email, code):

    response = client.send_email(
    Destination={
        'ToAddresses': [receiver_email]
    },
    Message={
        'Body': {
            'Text': {
                'Charset': 'UTF-8',
                'Data': "Your code is " + code + ". This will expire in 5 minutes.",
            }
        },
        'Subject': {
            'Charset': 'UTF-8',
            'Data': 'Login Code',
        },
    },
    Source='graduation.gathering.login@gmail.com'
    )

    return {
    'statusCode': 200,
    'body': json.dumps("Email Sent Successfully. MessageId is: " + response['MessageId'])
    }
```

Now that an email has been entered, a code generated, and an email containing the code has been sent to the entered email, the user is asked to enter the code.



The code entered is then sent along with the email to the ValidateCode lambda method. This then verifies if the code has been entered by retrieving the login code and expiration time for the given email from the database.

```
# Verifies whether the given code is correct and with the time constraint for the given email.
def verifyCode(email, code):
    with conn.cursor() as cur:
        cur.execute(f"select login_code, login_code_expires from user where user_email = '{escape_sql_string(email)}'")
        loginCode = None
        for row in cur:
            loginCode = row[0]
            expires = row[1]
        cur.close()
    conn.commit()
    if loginCode == None:
        return False

    if loginCode == code:
        return float(expires) >= time.time()
    else:
        return False
```

If the codes match and the time hasn't expired then a JWT is generated for the user to authenticate future requests. This JWT is also given an expiration date of 24 hours from its creation.

```python
# Generates a JWT token for the user to use to authenticate their request.
def generateToken(email, userID):

    payload = {
        'id': userID,
        'email': email,
        'created': time.time(),
        'expires': time.time() + (60*60*24) # Valid for 24 hours - stored in seconds from unix epoch
    }

    secret = get_secret()

    encoded_jwt = jwt.encode(payload, secret, algorithm="HS256")

    token = encoded_jwt
    return token

# Gets the key to sign the token.
def get_secret():
    return key
```

The last stage is for this JWT to be stored by the Flutter app in secure storage so that it can be used to automatically login the user subsequent times the user logs in until the token expires. The JWT is stored securely in the TokenStorage class using the FlutterSecureStorage package.

```dart
/// Securely stores and retrieves authorisation tokens.
class TokenStorage
{
  // The secure storage used.
  final _storage = const FlutterSecureStorage(aOptions: AndroidOptions(
    encryptedSharedPreferences: true,
  ));  // AndroidOptions, FlutterSecureStorage

  /// Retrieves a token that has been stored on the device.
  Future<AuthToken?> getToken() async {
    String? value = await _storage.read(key: "auth token");
    if (value == null)
    {
      return null;
    }
    return AuthToken(value);
  }

  /// Securely stores the given token in storage on the device.
  writeToken(AuthToken authToken) async {
    await _storage.write(key: "auth token", value: authToken.getToken());
  }

  clearToken() async
  {
    await _storage.write(key: "auth token", value: null);
  }
}
```

Sprint 1 Summary

At the end of the first sprint, a Flutter app has been created with a login screen, a main screen with a working navigation bar, a map screen that displayed both a map and the user's location. The server was also set up with a framework to easily add and call additional lambda methods and increase the capabilities of the database. This is a great jumping off point for the rest of the project and allows for the rest of work to be completed efficiently.

Sprint 2 – Server-side API, Lambda Methods and Database Development

The goal of the second sprint was to develop the lambda methods and database to support the front-end that would be implemented later. The idea behind implementing all the lambda methods together instead of implementing a single method and then going and implementing the front-end for that feature, is that focusing on the lambda methods will keep them fresh in my mind and so allow me to work efficiently and effectively instead of having to refresh myself on how they work every time I want to implement a new method.

Task 1 – Location Sharing

The first task is to implement the key feature of this project, location-sharing. This is done by having one lambda method which a user sends its location to, and a second lambda method which the user will get the other users locations from. These requests are done separately because the users location on the server should be updated every time the app receives an update of the users location, however, if the users location on the app isn't getting frequent updates then if these requests were done together then the user wouldn't get frequent updates on other users locations. This approach unties the user getting location updates of other users from their own location being updated.

The user's location is updated on the server using the SetLocation lambda method. First the date is checked to see if it is a graduation day (8 am – 1 am next day), as location sharing is only enabled on graduation days. In reality this aspect was implemented at the end of development to make testing easier.

```python
# Checks if it is currently during graduation time
def isGraduationDay():
    datesStrings = getDates()
    dates = []
    for dateString in datesStrings:
        dates.append(datetime.strptime(dateString, '%y/%m/%d').date())

    today = datetime.today().date()

    for date in dates:
        if today == date:
            if today.time() >= datetime.strptime("08", "%H"):
                return True
        else:
            tomorrowDate = datetime.strftime(date - datetime.timedelta(1), '%Y-%m-%d')
            if today == tomorrowDate:
                if today.time() < datetime.strptime("01", "%H"):
                    return True

    return False

# Gets the graduation dates from the database
def getDates():
    dates = []
    with conn.cursor() as cur:
        # Gets the graduation dates
        sql = "SELECT date FROM graduation_date"
        cur.execute(sql)
        for row in cur:
            date = row[0]
            dates.append(date)
    conn.commit()

    return dates
```

Then the location is simply inserted into the user's entry using the userID taken from the JWT used to authenticate this request. The time that the location is set is also recorded so that once it is outdated information it isn't sent out to users anymore.

```python
sql_string = 'UPDATE user SET latitude = {latitude}, longitude = {longitude}, location_set = {locationSet} WHERE user_id = "{userID}"'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(latitude = location["lat"], longitude = location["long"], locationSet = time.time(), userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)
    cur.close()
conn.commit()
```

The locations of other users are retrieved using the GetOthersLocations lambda method. It returns all the profile information and location of every user that this user has permission to see.

```python
otherUsers = []
with conn.cursor() as cur:
    sql = "SELECT user_email, latitude, longitude, user_id, user_name, faculty_name, school_name, course_name, student_staff FROM user WHERE user_id != '{userID}' AND location_set > {time} AND '{userID}' in (SELECT from_user FROM location_permission WHERE '{userID}' = from_user AND user_id = to_user AND permission_granted = 'Granted')"
    locationValidTime = time.time() - 60
    cur.execute(sql.format(userID = userID, time = locationValidTime))
    for row in cur:
        otherUserEmail = row[0]
        otherUserLatitude = row[1]
        otherUserLongitude = row[2]
        otherUserID = row[3]
        otherUserName = row[4]
        otherUserFaculty = row[5]
        otherUserSchool = row[6]
        otherUserCourse = row[7]
        otherUserStudentStaff = row[8]
        otherUser = {
            'email': otherUserEmail,
            'latitude': otherUserLatitude,
            'longitude': otherUserLongitude,
            'id': otherUserID,
            'name': otherUserName,
            'faculty': otherUserFaculty,
            'school': otherUserSchool,
            'course': otherUserCourse,
            'student/staff': otherUserStudentStaff
        }
        otherUsers.append(otherUser)
    cur.close()
conn.commit()

return otherUsers
```

Task 2 – Profiles

The second task was to store user profile information and allow it to be retrieved from other users. Users edit their own profile using the SetUserProfile and GetUserProfile lambda methods.

GetUserProfile is very simple, it just returns all the information on the user with the userID given by the JWT, this means that this information is only accessible to the user that the information is about.

```
with conn.cursor() as cur:
    profile_sql = "SELECT user_email, user_id, user_name, faculty_name, school_name, course_name, student_staff, has_logged_in FROM user WHERE user_id = '{userID}'"
    cur.execute(profile_sql.format(userID = userID))
    for row in cur:
        userEmail = row[0]
        userID = row[1]
        userName = row[2]
        userFaculty = row[3]
        userSchool = row[4]
        userCourse = row[5]
        userStudentStaff = row[6]
        userHasLoggedIn = row[7]
    zones = []
    zones_sql = "SELECT zone_id FROM user_zones WHERE user_id = '{userID}'"
    cur.execute(zones_sql.format(userID = userID))
    for row in cur:
        zones.append(row[0])
    user = {
            'email': userEmail,
            'id': userID,
            'name': userName,
            'faculty': userFaculty,
            'school': userSchool,
            'course': userCourse,
            'accountType': userStudentStaff,
            'hasLoggedInBefore': userHasLoggedIn,
            'userGradZoneIds': zones
        }
    cur.close()
conn.commit()

return user
```

SetUserProfile allows a user to edit their own profile.

```
if name == None:
    name = "Null"
else:
    name = '"' + escape_sql_string(name) + '"'
if faculty == None:
    faculty = "Null"
else:
    faculty = '"' + escape_sql_string(faculty) + '"'
if school == None:
    school = "Null"
else:
    school = '"' + escape_sql_string(school) + '"'
if course == None:
    course = "Null"
else:
    course = '"' + escape_sql_string(course) + '"'

sql_string = 'UPDATE user SET has_logged_in = {hasLoggedInBefore}, user_name = {name}, faculty_name = {faculty}, school_name = {school}, course_name = {course} WHERE user_id = "{userID}"'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(hasLoggedInBefore = hasLoggedInBefore, name = name, faculty = faculty, school = school, course = course, userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)

    delete_sql = 'DELETE FROM user_zones WHERE user_id = "{userID}"'
    try:
        cur.execute(delete_sql.format(userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)

    insert_sql = 'INSERT INTO user_zones (user_id, zone_id) VALUES ("{userID}", "{zone}")'
    for zone in zones:
        try:
            cur.execute(insert_sql.format(userID = escape_sql_string(userID), zone = zone))
        except pymysql.MySQLError as e:
            logger.error(e)
    cur.close()
conn.commit()
```

The methods allow the user to edit their faculty, school, course and graduation zones. However, at the moment there is no way for the user to retrieve the possible faculties, schools, courses and graduation zones.

The lambda method AcademicStructure gets the faculties, schools and courses.

```
structure = {}
with conn.cursor() as cur:
    # Gets the faculties
    faculty_sql = "SELECT faculty_name FROM faculty"
    cur.execute(faculty_sql)
    for row in cur:
        faculty = row[0]
        structure[faculty] = {}

    # Gets the schools
    school_sql = "SELECT school_name, faculty_name FROM school"
    cur.execute(school_sql)
    for row in cur:
        school = row[0]
        faculty = row[1]
        structure[faculty][school] = []

    # Gets the courses
    course_sql = "SELECT course_name, school_name FROM course"
    cur.execute(course_sql)
    for row in cur:
        course = row[0]
        school = row[1]
        structure[faculty][school].append(course)
    cur.close()
conn.commit()
```

The lambda method GetGradZones returns the graduation zones. It gets the zones IDs, names and geojsons.

```
zones = []
with conn.cursor() as cur:
    sql = "SELECT graduation_zones.zone_id, graduation_zones.zone_name, graduation_zones_text.geojson FROM graduation_zones INNER JOIN graduation_zones_text ON graduation_zones.zone_id = graduation_zones_text.zone_id"
    cur.execute(sql)
    for row in cur:
        id = row[0]
        name = row[1]
        geojsonString = row[2]
        geojson = json.loads(geojsonString)
        zones.append({"id": id,"name": name, "geojson": geojson})
conn.commit()

return zones
```

Finally, a user needs to be able to see the profiles of other users. This is done with the user of the lambda method GetAllOtherUsersProfiles which gets the user ID, email, name, faculty, school, course, and account type of every single other user that has verified their email address.

```
users = []
with conn.cursor() as cur:
    faculty_sql = "SELECT user_id, user_email, user_name, faculty_name, school_name, course_name, student_staff FROM user WHERE user_id != '{userID}' AND email_verified = true"
    cur.execute(faculty_sql.format(userID = userID))
    for row in cur:
        otherUserID = row[0]
        userEmail = row[1]
        userName = row[2]
        faculty = row[3]
        school = row[4]
        course = row[5]
        accountType = row[6]
        users.append({"userID": otherUserID, "userEmail": userEmail, "userName": userName, "faculty": faculty, "school": school, "course": course, "accountType": accountType})
conn.commit()

return users
```

Task 3 - Connections

The only part of the server now missing is the ability to request and grant permission to see other people's locations. First, GetConnections returns all the connections that a given user

has, this is both granted and requested permission, and both for this user to see other users and for other users to see this user.

```
connections = []
with conn.cursor() as cur:
    faculty_sql = "SELECT from_user, to_user, permission_granted FROM location_permission WHERE (from_user = '{userID}' OR to_user = '{userID}') AND permission_granted != 'Denied'"
    cur.execute(faculty_sql.format(userID = userID))
    for row in cur:
        fromUser = row[0]
        toUser = row[1]
        permission = row[2]
        connections.append({"fromUser": fromUser, "toUser": toUser, "permission": permission})
conn.commit()

return connections
```

Finally we have the four lambda methods for managing connections:

SendConnectionRequest for requesting permission from another user to see them.

```
sql_string = 'INSERT INTO location_permission (from_user, to_user, permission_granted) VALUES ("{userID}", "{otherUserID}", "Requested")'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(otherUserID = escape_sql_string(otherUserID), userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)
    cur.close()
conn.commit()
```

GrantFollowerRequest for granting permission to another user to see you.

```
sql_string = 'UPDATE location_permission SET permission_granted = "Granted" WHERE from_user = "{otherUserID}" AND to_user = "{userID}"'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(otherUserID = escape_sql_string(otherUserID), userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)
    cur.close()
conn.commit()
```

RemoveFollowingRequest for deleting your request for permission to see another user.

```
sql_string = 'DELETE FROM location_permission WHERE from_user = "{userID}" AND to_user = "{otherUserID}"'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(otherUserID = escape_sql_string(otherUserID), userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)
    cur.close()
conn.commit()
```

RemoveFollowerRequest for denying another user permission to see you.

```
sql_string = 'UPDATE location_permission SET permission_granted = "Denied" WHERE from_user = "{otherUserID}" AND to_user = "{userID}"'

with conn.cursor() as cur:
    try:
        cur.execute(sql_string.format(otherUserID = escape_sql_string(otherUserID), userID = escape_sql_string(userID)))
    except pymysql.MySQLError as e:
        logger.error(e)
    cur.close()
conn.commit()
```

Sprint 2 Summary

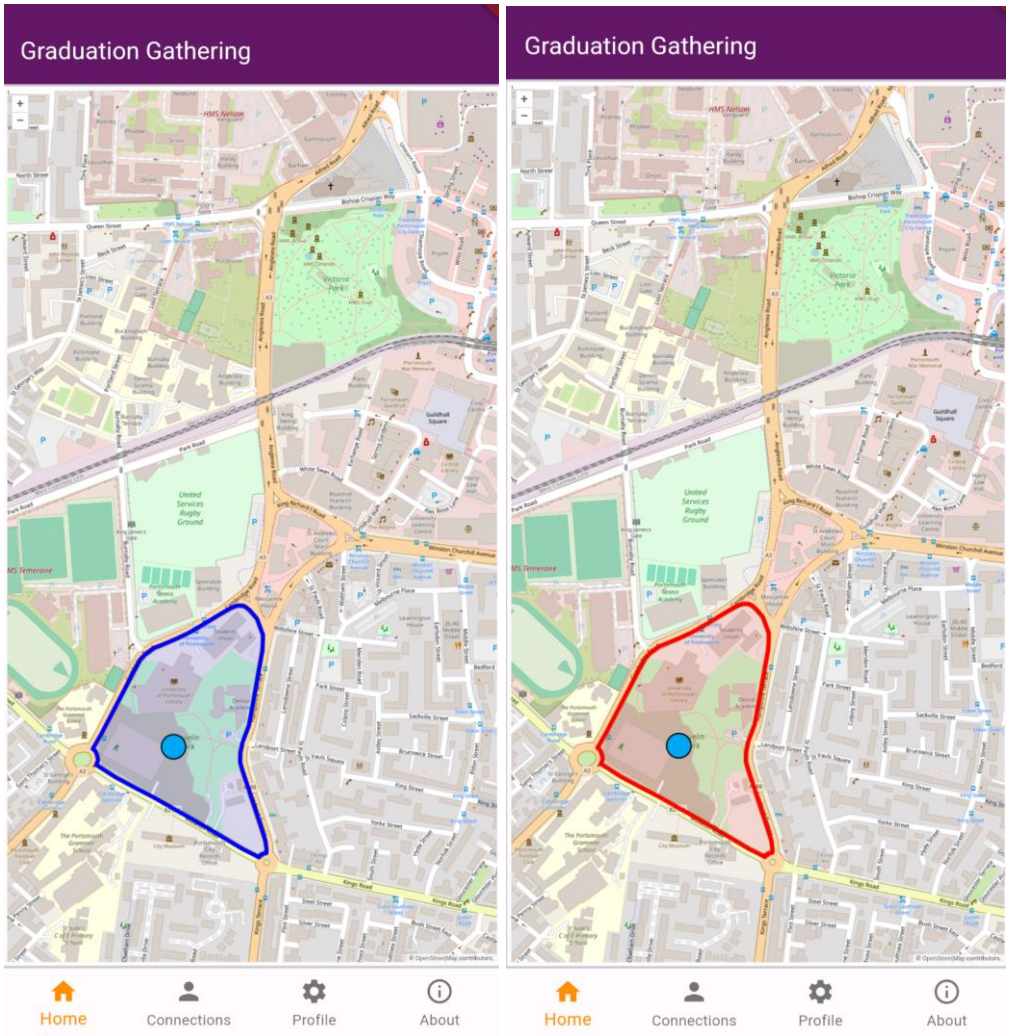
The server-side of the application is now in a near complete state, further editing would be done in later sprints as the design changes, but apart from these minor changes the server is done. Now the app itself can be built and focused on.

Sprint 3 – UI and Front-end Logic

Sprint 3 is the final sprint, during which the UI and the logic to support it was built. This sprint provided a chance to evaluate the whole working product.

Task 1 – Graduation Zones

The graduation zones are the areas of Portsmouth where a user's location can be shared. A zone is coloured blue on a user's map if they will appear within it, and red if they will not appear within it.



The zones are stored in the database as geojsons. These geojsons can then be directly given to OpenLayers to display on the map.

In Flutter these zones are stored in their own data structure, GradZones, which provides specific methods to decouple the zones from the classes they interact with.

```dart
/// Zone colours enum.
enum ZoneColours
{
  /// Pure blue colour
  blue(0,0,255),
  /// Pure Red colour
  red(255,0,0);

  const ZoneColours(this._r,this._g,this._b);

  final int _r;
  final int _g;
  final int _b;

  /// Returns the RGB Colour Object.
  getColourRGB()
  {
    return Colour(_r,_g,_b);
  }
}
```

GradZones provides a method for the zones to be given as a sorted list by alphabetical order of their ID's. Sorting by alphabetical order of the ID's will keep the order they are displayed consistent without actually ordering them by any meaningful characteristic.

```dart
/// Returns the zones as a list sorted in alphabetical order of their ids.
List<GradZone> getZonesInOrder()
{
  HeapSort<String> sort = HeapSort<String>(Comparator.alphabeticalComparator());
  List<String> sortedZoneIds = sort.sort(getIds());
  List<GradZone> zones = [];
  for (String zoneId in sortedZoneIds)
  {
    zones.add(getZoneFromId(zoneId)!);
  }
  return zones;
}
```

The sorting algorithm used is the heap sort as it is efficient for all input sizes and so it can be reused to sort larger datasets if needed.

```dart
/// Performs a heap sort on the given list using the comparator given in the
/// constructor.
List<E> sort(Iterable<E> iterable)
{
  List<E> sortedList = [];
  for (E element in iterable)
  {
    _insert(element);
  }
  for (E element in iterable)
  {
    sortedList.add(_removeMin());
  }
  return sortedList;
}
```

```dart
// Insets a new element into the heap.
_insert(E element)
{
  _heap.insert(_heapSize, element);
  if (_heapSize == 0)
  {
    _heapSize++;
    return;
  }
  int currentIndex = _heapSize;
  int i = _heapSize;
  do
  {
    i = ((i - 1)/ 2).floor();
    if (_compare(element, _heap[i]) == Comparator.before)
    {
      _swap(currentIndex, i);
    }
    currentIndex = i;
  } while (i > 0);
  _heapSize++;
}
```

```
// Returns and removes the first element and adjusts the heap accordingly.
E _removeMin()
{
  _heapSize--;
  E min = _heap[0];
  _swap(0, _heapSize);
  _heap.removeAt(_heapSize);
  int currentIndex = 0;
  if (_heapSize == 0)
  {
    return min;
  }

  while (((2 * currentIndex) + 1 <= _heapSize - 1 &&
      _compare(_heap[currentIndex], _heap[(2 * currentIndex) + 1]) == Comparator.after) ||

      ((2 * currentIndex) + 2 <= _heapSize - 1 &&
          _compare(_heap[currentIndex], _heap[(2 * currentIndex) + 2]) == Comparator.after))
  {
    if ((2 * currentIndex) + 2 <= _heapSize - 1)
    {
      if ((2 * currentIndex) + 1 <= _heapSize - 1 && _compare(_heap[(2 * currentIndex) + 1], _heap[(2 * currentIndex) + 2]) == Comparator.before)
      {
        _swap(currentIndex, (2 * currentIndex) + 1);
        currentIndex = (2 * currentIndex) + 1;
      }
      else
      {
        _swap(currentIndex, (2 * currentIndex) + 2);
        currentIndex = (2 * currentIndex) + 2;
      }
    }
    else
    {
      _swap(currentIndex, (2 * currentIndex) + 1);
      currentIndex = (2 * currentIndex) + 1;
    }
  }
  return min;
}
```

```
// Swaps the elements at the given index's
_swap(int index1, int index2)
{
  E temp = _heap[index1];
  _heap[index1] = _heap[index2];
  _heap[index2] = temp;
}
```

This heap sort also takes a comparator to sort the elements making it highly reusable.

```
/// The enums for the return value of a comparator.
enum Comparator {
  before,
  same,
  after;
```

The comparator used to sort the zones is as follows:

```
/// Alphabetical comparator.
static Comparator Function(String string1, String string2) alphabeticalComparator() {
  return ((String string1, String string2) {
    string1 = string1.toLowerCase();
    string2 = string2.toLowerCase();
    if (string1 == string2) {
      return Comparator.same;
    }
    for (int i = 0; i <= string1.length; i++) {
      if (string1.length <= i) {
        return Comparator.before;
      } else if (string2.length <= i) {
        return Comparator.after;
      } else if (string1.codeUnitAt(i) < string2.codeUnitAt(i)) {
        return Comparator.before;
      } else if (string1.codeUnitAt(i) > string2.codeUnitAt(i)) {
        return Comparator.after;
      }
    }
    return Comparator.before;
  });
}
```

The method for testing if a user if within a given zone is performed by OpenLayers. The JavaScript below displays how a point of latitude and longitude is tested against a geojson string.

```
function isPointInsidePolygons(point, geojsons) {
    if (geojsons.length == 0)
    {
        return false;
    }
    for (var geoJsonIndex in geojsons) {
        const geoJson = geojsons[geoJsonIndex];
        const polygons = (new ol.format.GeoJSON({
            featureProjection: 'EPSG:3857',
            dataProjection: 'EPSG:4326'
        })).readFeatures(geoJson.geoJson);
        for (var polygonIndex in polygons) {
            if (!polygons[polygonIndex].getGeometry().intersectsCoordinate(new ol.proj.fromLonLat([point.longitude, point.latitude]))) {
                return false;
            }
        }
    };
    return true;
}
```

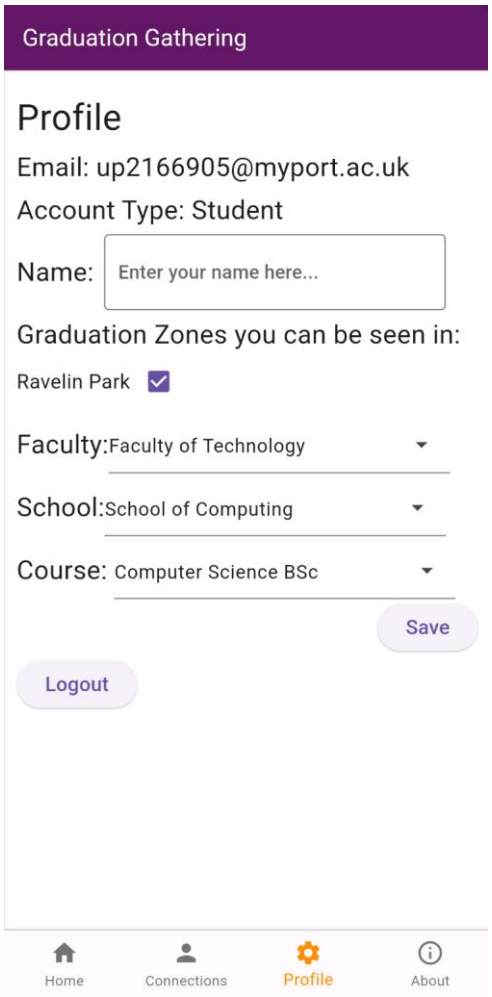
In order to test a point against all zones, the zones are looped through as shown below.

```
/// Checks if the given point is inside the given zone.
@protected
Future<bool> isPointInsideGeojson(double long, double lat, GradZones zones) async
{
  if (zones.isEmpty)
  {
    return false;
  }
  List<Map<String, dynamic>> zonesGeojsons = zones.geojsonsAsList();
  List<Map<String, Map<String, dynamic>>> zonesInJsForm = [];
  for (Map<String, dynamic> zoneGeojson in zonesGeojsons) {
    zonesInJsForm.add({"geoJson": zoneGeojson});
  }
  String jsObjectPoint = "{longitude: $long, latitude: $lat}";
  String jsObjectGeojsons = json.encode(zonesInJsForm);
  return await _webViewController.runJavaScriptReturningResult("isPointInsidePolygons($jsObjectPoint, $jsObjectGeojsons)") as bool;
}
```

The original plan was to test if a user is with their allowed zones on both their phone and the server. However, when I attempted to use the python package Shapely for the test on server, AWS Lambda rejected the package. This was caused by AWS Lambda being run on a version of Linux whilst my host machine which was used to download the Shapely package and implement the Python method is run on Windows 11. I first attempted to fix this issue by installing Docker, then later attempted to fix this using a virtual machine running Linux, however I was never able to fix the issue and so abandoned the idea of checking if the users location is within one of its allowed zones on the server. The check is still completed on the users phone and so this isn't a security issue.

Task 2 – Profile Screen

The profile screen allows a user to set their name, faculty, school, course, and finally which graduation zones they would like to appear in. The final design is as shown below.



The faculties, schools and courses as seen from the profile screen are stored in the AcademicStructure class. This data structure stores the values in a Map in the form Map<Faculty, Map<School, Set<Course>>>. This allows for it to return all the faculties, all the schools, all the courses, all the schools in a faculty and all the courses in a school very efficiently and, for a data structure this small far more importantly, the code to do so is very readable and editable.

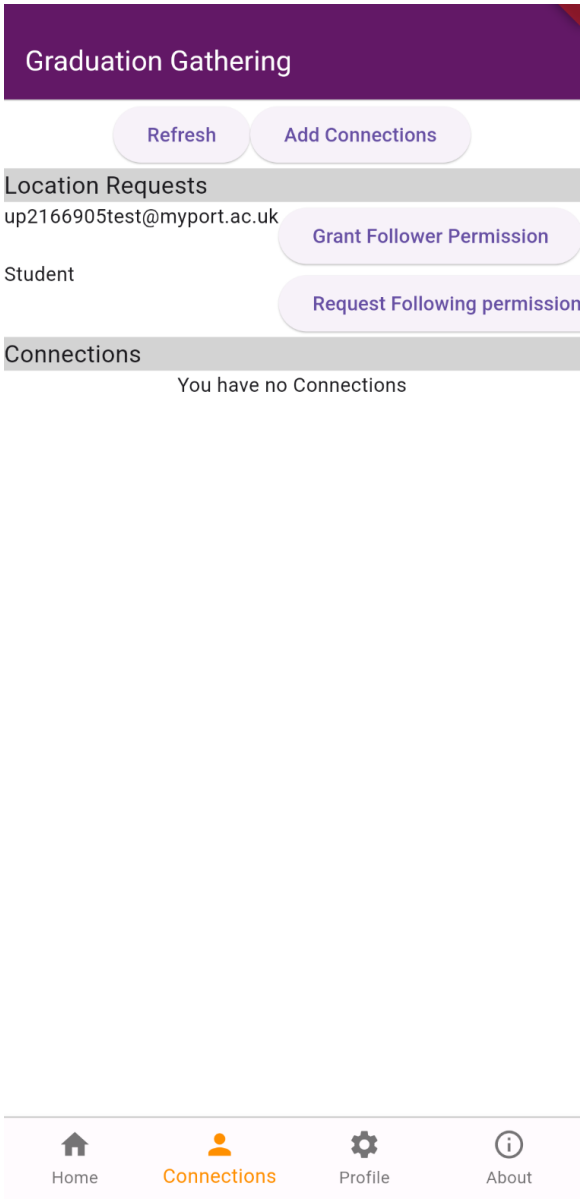
```dart
/// Holds the academic structure of the University of Portsmouth.
class AcademicStructure
{
  final Map<String, Map<String, Set<String>>> _structure = <String, Map<String, Set<String>>>{};

  AcademicStructure(Map<String, dynamic> structure)
  {
    structure.forEach((key, value) {_structure[key] = {};
    for (String valueKey in (value as Map<String, dynamic>).keys)
    {
      _structure[key]![valueKey] = <String>{};
      for (String valueValue in (value[valueKey]))
      {
        _structure[key]![valueKey]!.add(valueValue);
      }
    }
    });
  }

  /// Returns the collection of the Faculties.
  Iterable<String> getFaculties()
  {
    return _structure.keys;
  }

  /// Returns the collection of the schools from a given faculty.
  Iterable<String>? getSchoolsFromFaculty(String faculty)
  {
    return _structure[faculty]?.keys;
  }

  /// Returns the collection of the courses from a given school and faculty.
  Iterable<String>? getCoursesFromSchoolAndFaculty(String school, String faculty)
  {
    return _structure[faculty]?[school];
  }
}
```

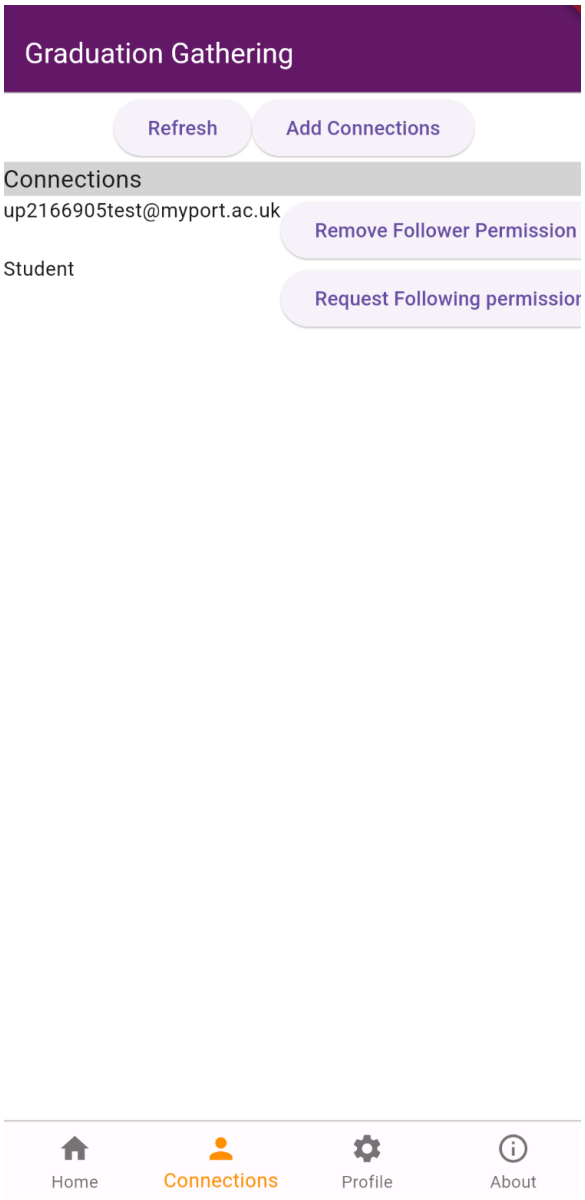
The account type of the user is also stored as an enum to simplify the code for when not to display courses to staff users.

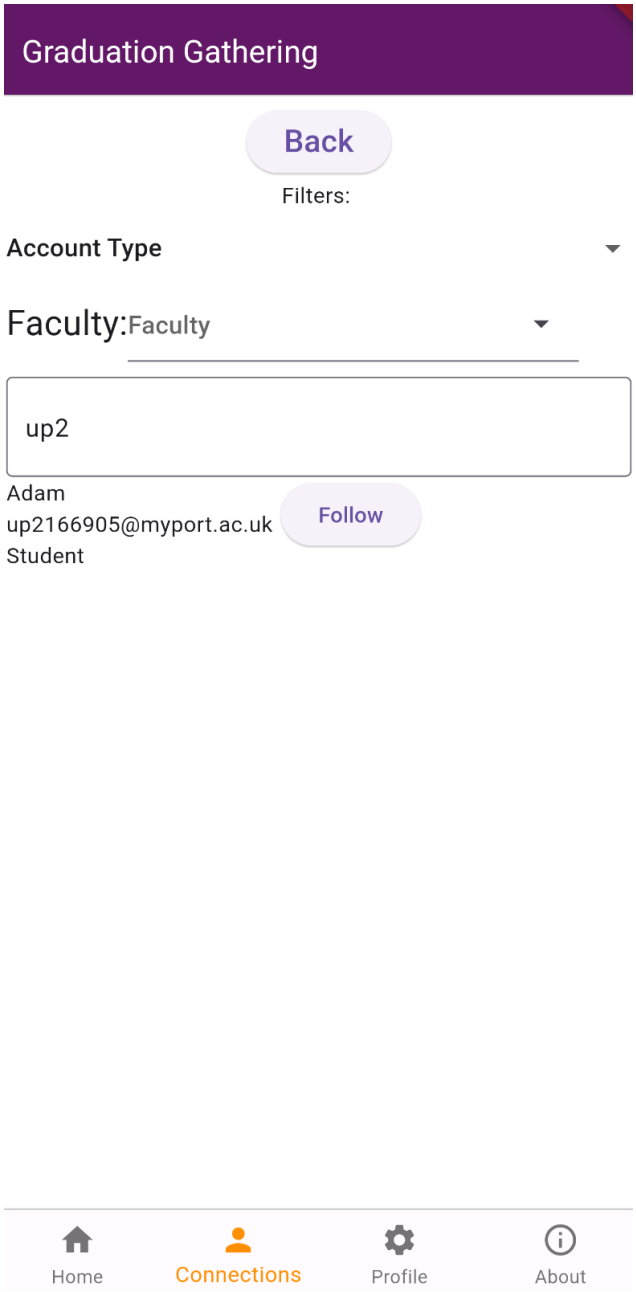
```dart
/// Enum for the types of accounts users can have.
enum AccountType
{
  student("Student"),
  staff("Staff");

  const AccountType(this.accountTypeAsString);

  final String accountTypeAsString;

  /// Returns the account type enum that a string represents.
  static AccountType? getAccountTypeFromString(String accountTypeString)
  {
    if (accountTypeString.toLowerCase() == "student")
    {
      return AccountType.student;
    }
    else if (accountTypeString.toLowerCase() == "staff")
    {
      return AccountType.staff;
    }
    return null;
  }
}
```

Task 3 – Connections Screen

The connections screen has two parts, the first displays the connections you currently have, and the second allows you to request permission to see other users to connect with them.


Graduation Gathering
Refresh
Add Connections
Location Requests
up2166905test@myport.ac.uk
Grant Follower Permission
Student
Request Following permission
Connections
You have no Connections
Home
Connections
Profile
About


Graduation Gathering
Refresh
Add Connections
Connections
up2166905test@myport.ac.uk
Remove Follower Permission
Student
Request Following permissior
Home
Connections
Profile
About



The connections are stored in the Connections data structure.

```
Connections(List<dynamic> connections, OtherUserProfiles allOtherUserProfiles, ProfileSettings userProfile)
{
  Map<String, Map<String, String>> transformedConnections = {};
  // Transforms the connections from the format User A, User B, permission (I.e. granted or requested)
  // into the format, User that is not this client user, permission from this
  // user to that user, permission to this user from that user.
  for (Map<String, dynamic> connection in connections)
  {
    if (connection["fromUser"] == userProfile.getId())
    {
      String otherUser = connection["toUser"];
      if (transformedConnections[otherUser] == null)
      {
        transformedConnections[otherUser] = {"toUser": connection["permission"]};
      }
      else
      {
        transformedConnections[otherUser]!["toUser"] = connection["permission"];
      }
    }
    else if (connection["toUser"] == userProfile.getId())
    {
      String otherUser = connection["fromUser"];
      if (transformedConnections[otherUser] == null)
      {
        transformedConnections[otherUser] = {"fromUser": connection["permission"]};
      }
      else
      {
        transformedConnections[otherUser]!["fromUser"] = connection["permission"];
      }
    }
  }
```

Sprint 3 Summary

Graduation Gathering, from front-end to back, is now complete. Not all requirements have been met, but most have and the most important features have been added. This sprint has seen the UI being built and become responsive as well as the data needed to power the app being stored effectively in various data structures.

Verification and Validation

Verification is checking that the software does what we expect it to do without having any bugs. Validation is checking that the software does what the requirements ask it to do.

For this project, at each stage of the implementation, I will first consider validation as it is pointless to test and debug only to realise afterwards that you built the wrong thing. I will think through the feature that I want to add, both before and after adding it, to ensure that it is what is required. Remember, the key goal for this project is to make it easier for students and staff to meetup on graduation day.

In order to verify a feature it will undergo 2 sets of testing, manual debugging and automated unit testing. The server will also undergo load testing in order to verify it can handle the expected number of concurrent users.

Manual Debugging

As I implemented every feature, I tested them extensively with a variety of inputs to verify that the software is acting as expected. This level of testing focused on the larger scale of the feature and so did not send specific inputs to the individual methods. However, this level of testing does include interacting with the UI and making sure that it reacts to user input appropriately, it also includes testing the servers endpoints. This level of testing uses the live server and not a mock of the methods and so could be unpredictable in exactly the results presented, however it is reflective of how the app would be used and so gets a good perspective on the app. To be clear, this perspective is not as good as the feedback that would be given by actual users.

Automated Unit Testing

For each public method implemented in the client side of the app, barring a few exceptions, unit tests have been written to automatically give the method a variety of inputs to check it returns the expected result. For this level of testing, mocks have been used to return data from the database and from local storage, this is to ensure consistency in the inputs to the test. Due to this reason any method that's sole purpose is to retrieve data from the server unedited, has not had tests written for it as these tests would not accomplish anything. Unit testing is rigid and predictable and doesn't reflect how actual users would interact with the program, however it does help you check you haven't inadvertently broken an existing part of the program. The act of writing unit tests also makes you think about what the software is doing and how it accomplishes its task, this both helps with validation and verification.

Load Testing

In order to test if the server can support the desired number of concurrent users, I have designed a load testing experiment that can be run efficiently and repeated as the server is updated. I have written a poster on this experiment and the results that I got when I conducted the experiment (A. Murray, Distributed Systems and Security Coursework, April 18, 2024). This experiment found that the server works seamlessly up to 70 concurrent users, reasonably well up to 300 users, and fails above this point. This means that it falls far short of the desired 2000 users.

Evaluation

The aim of this project was to create a location-sharing application to help students and staff of the University of Portsmouth connect on graduation day. In order to do this I have developed an app for both iOS and Android using Flutter with a server hosted on AWS. I have thoroughly tested each feature implemented to ensure it is bug free.

In the tables of requirements below I have coloured in the rows of the completed requirements green, of the non-completed requirements red, any requirements where it is unknown whether

it has been met shall be coloured orange, and finally any requirements where it has been partially met will be coloured blue and discussed further below.

Functional

Must Have

ID	Description
1	A map that is rendered on screen showing Portsmouth
2	The user's current location displayed on the map when the user has location enabled
3	Other users current locations to be displayed on the map to the user
4	Manage who has permission to see your current location
5	Manage who's location you have permission to see
6	Users login with their university emails (@myport.ac.uk and @port.ac.uk) by having a code sent to their email that they will then enter into the app
7	All requests sent to the server encrypted using HTTPS
8	All requests sent to the server must be authenticated
9	Designated graduation zones to appear on the map, only in which will users locations be shared
10	Users locations only being shared when they have the app open and are logged in
11	Users locations only being shared on graduation days(8am – 1am the next day)

Should Have

ID	Description
12	Users can add their name to their account
13	Users can add their faculty to their account
14	Users can add their school to their account
15	Users can add their course to their account
16	The account type (Student/Staff) saved to an account based on the email address that they used to login
17	Users can select which graduation zones they must be in for their location to be shared
18	To search for other users by their email address, when deciding who has permission to see you and who you can to see
19	To search for other users by their name, when deciding who has permission to see you and who you can to see
20	To search for other users by their faculty, when deciding who has permission to see you and who you can to see
21	To search for other users by their school, when deciding who has permission to see you and who you can to see
22	To search for other users by their course, when deciding who has permission to see you and who you can to see

23	To search for other users by their account type, when deciding who has permission to see you and who you can to see

Could Have

ID	Description
24	The ability to tap users you can see on your map to display the information they have added to their profile
25	The ability to tap users you can see on your map and tick a box to say you have already seen this person and so to stop that user from continuing to display on your map
26	To filter the users that appear on your map by faculty
27	To filter the users that appear on your map by school
28	To filter the users that appear on your map by course
29	To filter the users that appear on your map by email
30	To filter the users that appear on your map by name
31	To filter the users that appear on your map by graduation zones
32	To filter the users that appear on your map by account type
33	A checklist of what students need to do on graduation day such as picking up their gown, attending their ceremony, going to the reception and collecting their certificate
34	Markers on the map showing the key locations for graduation (Guildhall, Ravelin Sports Centre, Gun House Green)
35	The above mentioned markers clickable to display information about what is happening at these locations

Non-Functional

Must Have

ID	Description
41	Run on the latest version of Android(Android 14)
42	The ability to handle 50 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)
43	A users location update on other users devices at most 30 seconds after the users client get an update of its own location (assuming all users have a good connection to the internet)
44	The Authentication tokens allow users to login for no more than 24 hours
45	The login codes sent to users emails should be valid for no more than 5 minutes

Should Have

ID	Description
46	Run on the latest version of iOS(iOS 17)
47	The ability to handle 500 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)

Could Have

ID	Description
48	The ability to handle 2000 concurrent users without denying any of them service(Returning Service Unavailable when a user's client calls an endpoint on the server)

All but one of the requirements I marked as a must have requirement, have been successfully implemented. Further, all of the functional should have requirements have also been implemented. However, only one of the functional could have requirements has been met. For the non-functional requirements in the should have and could have categories, the server has failed to meet expectations and cannot handle the maximum expected number of concurrent users.

However, note that for requirement 46, this requirement has been marked as unknown as although the app has been built using Flutter and only contains packages that can be used on iOS, in order to actually compile and so test the software on iOS, you require an apple computer which I have been unable to acquire. Due to this the software most likely works on iOS but is completely untested and so this requirement cannot be verified.

For requirement 6, this has been partially met however does not fully work as intended. The system correctly only allows for accounts to be made using university emails. However, the server uses AWS Simple Email Service to send emails, and whilst the code I have written works currently to send the email, my AWS account is restricted to only being able to send emails to reverified email accounts.

Conclusion

Graduation Gathering does successfully allow for seamless location sharing between up to 70 concurrent users on Android devices. It also does this securely encrypting and authenticating all requests made to the server. Users are also able to efficiently manage who they want to have permission to see them and who they have permission to see. Finally users are also able to customise their profile if they wish. This is a major success and should not be understated as this means that Graduation Gathering has achieved the goal it set out to achieve.

However, it lacks some quality of life features that would make it easier to use, such as being able to filter who displays on your map at a given time.

Graduation Gathering's UI is also not the prettiest. Whilst it does stick to a nice colour scheme based off of the University of Portsmouth's colour, the look of the buttons and text is not the nicest in places.

Finally, the server does not meet expectations. It does allow for the must have required 50 concurrent users which does make the app usable under its current state. However, if Graduation Gathering were ever to be rolled out to actually be used and endorsed by the graduation team at the University of Portsmouth, it would need to be far more capable than it currently is.

Overall, this project is a success, however too many requirements were taken on for the time frame given. The main problem was unforeseen events affecting production far more than I anticipated.

Conclusions and Future Work

Graduation Gathering was created to solve the problem of people being unable to find each other on graduation day. I wish it could be a more complete product with more of the requirements having been met, however I am happy to say that it does successfully solve this problem.

Risks Faced

I have faced two main problems during this project. The first was an issue in my personal life that had a very large impact on me and delayed the start of the project resulting in fewer requirements being completed than initially planned. The second is that I was unable to source an Apple computer to use to build and test the app on iOS.

Future Work

The future work for this project is easy to point to as not all the requirements were met. In order for Graduation Gathering to be deployed, the servers scaling issue will need to be fixed and it will need to be properly tested on iOS. The ability to filter the users shown on the map should also be a priority to add.

The features originally put into the will not have right now category during the requirements faze would also be targets to add. The routing feature mentioned could be implemented with the help of OSRM (Open Source Routing Machine) which is made to work with OpenStreetMap data, which is what this app currently uses. The ability for users to draw their own zones is also a nice idea, it can be implemented with the help of OpenLayers, however I remain unsure of how to allow users to manage these new zones. The ability for users to send push notifications to each other would also be a must have feature, as currently for safety reasons, the app only will only share your location when you have the app open on your screen, due to this the app has the flaw of that if a user forgets about, then no one will be able to use the app to find them at all.

The future work for the server scaling issue has a variety of potential solutions. If Graduation Gathering were to be adopted by the University in an official capacity it is possible that the server would be moved from AWS to a University of Portsmouth server. The increase in computing power from this move might fix the problem entirely. Another solution to look at might be the configuration of my AWS account. My AWS account is set to not charge me any money, as part of this it might have automatically disabled load scaling which would have caused this issue. Another solution would be to move from basing the server upon AWS Lambda and instead use AWS EC2 (elastic computing) which provides a server environment with far more control given to the developer.

Finaly, I would also like to get the email system working. Once again, if officially endorsed by the University of Portsmouth, this issue could be fixed by using their system.

Final Thoughts

I have learnt a great deal working on this project, especially about how to use AWS. I have furthered many of my existing skills and gained even more experience developing apps for

Android. I am very proud of what I have achieved here, an application that can support 70 concurrent users that will bring people together, literally and hopefully figuratively as well.

References

- M. A. Rahman and M. S. Hossain, "A Location-Based Mobile Crowdsensing Framework Supporting a Massive Ad Hoc Social Network Environment," in IEEE Communications Magazine, vol. 55, no. 3, pp. 76-85, March 2017, doi: 10.1109/MCOM.2017.1600725CM.

Appendix A: Project Initiation Document

School of Computing

Final Year Engineering Project

Project Initiation Document

Adam Murray

Computer Science

Graduation Gathering

1. Basic details

Student name:	Adam Murray
Draft project title:	Graduation Gathering
Course and year:	Computer Science 3rd Year
Project supervisor:	Dr Gail Ollis

1. Degree suitability

I am a computer science student and this project is relevant to what I have been taught as I intend to create a mobile mapping application which will help connect people on graduation day. Last year through my Programming Applications and SETAP modules I learnt how to use Flutter to create mobile applications for both IOS and Android, further I learnt how to use OpenLayers to render a map in my app to allow for navigation for my SETAP coursework. This has prepared me perfectly to undertake this project.

2. The project environment and problem to be solved

The problem that I'm trying to solve is that with the chaos of graduation day, staff can find it difficult to find students in order to congratulate them. I intend to create a mobile app that will allow students and staff to share their location with each other and thus allow an easy way for staff and students to quickly find each other despite the crowds of the day.

3. Project aim and objectives

Aim: Allow students and staff to easily find each other on graduation day.

Objectives:

- Create a system to allow students and staff to share their location with each other.
- Create a system to allow students and staff to send messages to each other.
- Allow for only students and staff off the University Of Portsmouth to be able to sign into my app.

4. Project constraints

The main constraints for this project are that the application needs to be deployed by graduation day, estimated 12 July to 28 July 2024, and the project deadlines which are as follows:

- Submit PID, 20 October
- Pass Ethics Review, 8 December

- Submit Final Report, 3 May

Since the app needs to be deployed before the final report is written the deadline of graduation day is redundant.

5. Facilities and resources

I will require the following equipment:

- Computer that can run IntelliJ for general development.
- Computer that can run Android Studio to create the deployment package for android for both testing and release purposes.
- Computer that can run the Android Virtual Device manager to test the application on a variety of Android devices.
- Apple computer that can run Xcode to create the deployment package for IOS for both testing and release purposes.
- Apple Computer that can run the IOS Device manager to test the application on a variety of IOS devices.
- Access to Github.

I will require the following resources:

- License to publish apps to the Android Play Store.
- License to publish apps to the IOS App Store.
- A server run on AWS.

The only constraint I have to accessing any of these is my ability to access and apple computer for IOS testing and release is limited. This is due to me not personally owning an Apple Computer. However, I can currently borrow an apple computer for very limited testing and to release my app to the IOS App Store. This does pose an issue as if a major bug if found on IOS that is not present on android I may not have enough access to the Apple Computer to be able to fix it.

6. Log of risks

No	**Description**	**Likelihood** **(high, medium, low)**	**Impact**	**Mitigation/Avoidance**
1	Unable to source sufficient access to an Apple Computer	High	Could cause the IOS app the be released with known bugs.	Develop for both IOS and Android so there will still be an Android app produced irrespective of issues

				with IOS
2	Computer breaking	Low	Delays to the project and data loss	Frequently save work to Github to ensure as little data is lost as possible
3	Personal Life Issue	Low	Delays to project	Communicate effectively with supervisor when issues arise
4	AWS becomes unsuitable to host the server.	Medium	Delays to project as new server solution would have to be found	Look into potential alternatives to AWS

7. Project deliverables

This project will produce an application on both IOS and Android that allows for staff and students of the University Of Portsmouth to find each other on graduation day. Along with the application, a set of diagrams will be produced:

- An EDR for the database used to store messages sent between users.
- An architectural model diagram will be produced.
- Use case diagrams will be made to display how staff and students might use the app.
- A test suite will be made to automatically run unit tests.

Finally, a project report and PID will also be produced as part of this project.

8. Project approach

I plan to use Google and Google Scholar for my research to look into how similar problems have been approached in the past. Once the research is complete I will begin requirements gathering through interviewing staff and students to gain their incite on what they desire from my app. After this point I intend to use a kanban board and enter into 2 week sprints to complete the design, development and testing of my app. These 2 week sprints will have a number of related tasks brought in at the start of the sprint with the intention of completing all tasks allocated in the 2 weeks, these tasks will be added to my kanban board and given a priority in order to complete the most important tasks first. This allows for me to be able change my plans for further development and react quickly if opportunities arise such as an Apple Computer becoming available for a short while. I will prioritise my tasks according to the Mosco system.

9. Project tasks and timescales

No	Stage	Dates	Main Tasks
1	Project start up	18/09/23 – 06/10/23	Find supervisor and topic
2	Project initiation	09/10/23 – 20/10/23	Write PID
3	Literature review	23/10/23 - 31/10/23	Find relevant topics
4	Requirements Gathering	01/11/23 – 27/11/23	Interview staff and students to create the requirements
5	Software Design	28/11/23 – 15/12/23	Create the architectural model and other necessary documents
6	Build Software	18/12/23 – 04/03/24	Build the app
7	Testing	18/12/2 – 13/03/24	Test the entire application
8	Release Application	14/03/24 – 25/03/24	Create listings on the IOS App store and Android Play store and release the app to both
9	Project Report	18/09/23 – 02/05/24	Write the report
10	Close Project	03/05/24 – 17/05/24	Prepare and perform the presentation

10. Supervisor meetings

Meetings are face-to-face every Thursday at 12:30pm during term time.

11. Legal, ethical, professional, social issues

Legal:

- My project must comply with GDPR by properly handling user data. I will do this by never permanently storing user location information; giving the user the option to delete all information my app has about them, this would be their email address and messages with other users; and finally by only sharing their location with users that they have given permission too.

Ethical:

- My project will ethically handle the users location data by giving the user control over who can see their location. Setting the default to be no one can see their location and then allowing them to give other users permission to view their location and allowing them to rescind that permission at any time.

Professional:

- My project will stay professional by ensuring that everything is done legally and ethically and all personal data is handled responsibly.

Social (if any):

- My project will comply with social issues by remaining ensuring that all location data is handled ethically and only shown to other people with the users permission. Social issues will also be mitigated by ensuring that location sharing is only available during graduation times in graduation locations.

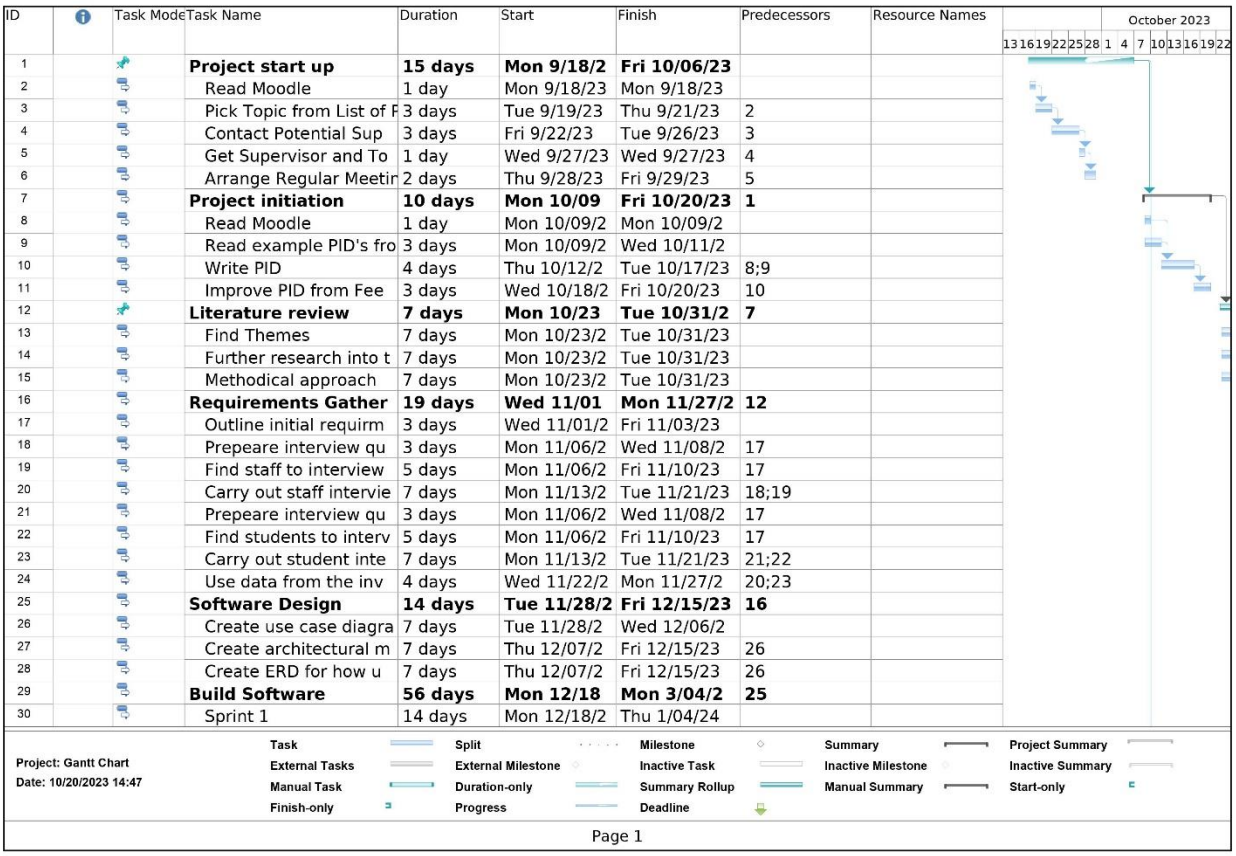
12. Permission

Please tick

☒ I give permission for my PID to be made available to other students as examples of previous work.

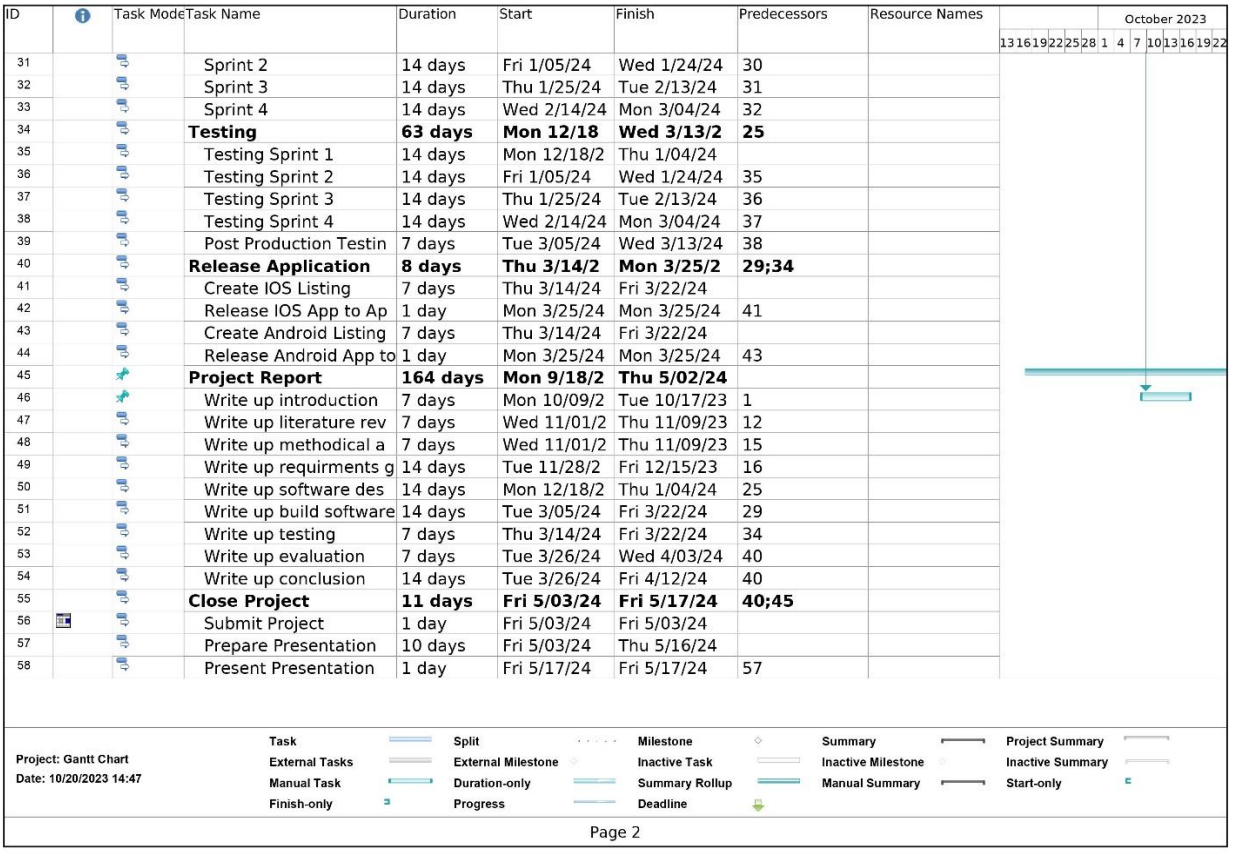
Date: 18/10/2023

Appendix B: Gantt chart



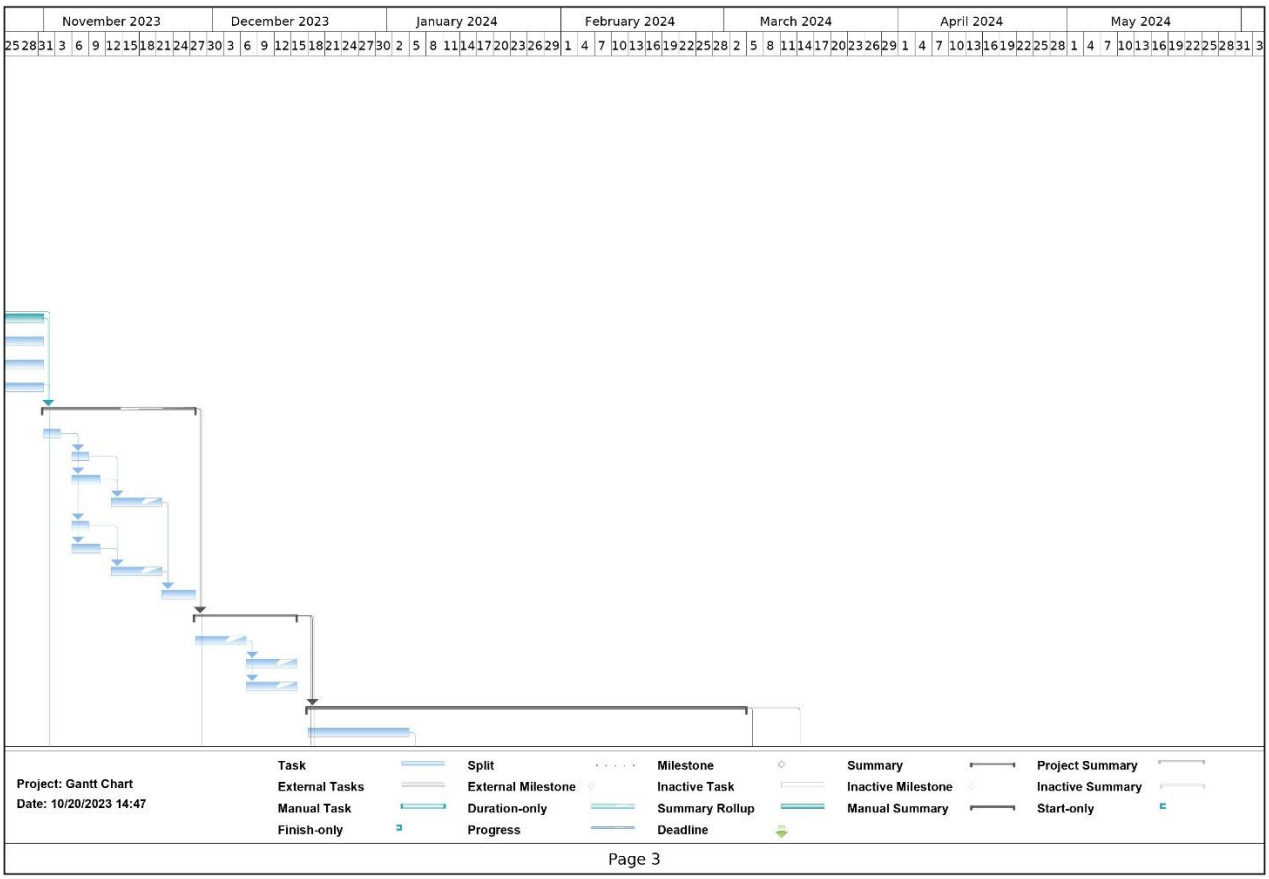
ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		**Project start up**	**15 days**	**Mon 9/18/2**	**Fri 10/06/23**		
2		Read Moodle	1 day	Mon 9/18/23	Mon 9/18/23		
3		Pick Topic from List of F	3 days	Tue 9/19/23	Thu 9/21/23	2	
4		Contact Potential Sup	3 days	Fri 9/22/23	Tue 9/26/23	3	
5		Get Supervisor and To	1 day	Wed 9/27/23	Wed 9/27/23	4	
6		Arrange Regular Meetin	2 days	Thu 9/28/23	Fri 9/29/23	5	
7		**Project initiation**	**10 days**	**Mon 10/09**	**Fri 10/20/23**	**1**	
8		Read Moodle	1 day	Mon 10/09/2	Mon 10/09/2		
9		Read example PID's fro	3 days	Mon 10/09/2	Wed 10/11/2		
10		Write PID	4 days	Thu 10/12/2	Tue 10/17/23	8;9	
11		Improve PID from Fee	3 days	Wed 10/18/2	Fri 10/20/23	10	
12		**Literature review**	**7 days**	**Mon 10/23**	**Tue 10/31/2**	**7**	
13		Find Themes	7 days	Mon 10/23/2	Tue 10/31/23		
14		Further research into t	7 days	Mon 10/23/2	Tue 10/31/23		
15		Methodical approach	7 days	Mon 10/23/2	Tue 10/31/23		
16		**Requirements Gather**	**19 days**	**Wed 11/01**	**Mon 11/27/2**	**12**	
17		Outline initial requirm	3 days	Wed 11/01/2	Fri 11/03/23		
18		Prepeare interview qu	3 days	Mon 11/06/2	Wed 11/08/2	17	
19		Find staff to interview	5 days	Mon 11/06/2	Fri 11/10/23	17	
20		Carry out staff intervie	7 days	Mon 11/13/2	Tue 11/21/23	18;19	
21		Prepeare interview qu	3 days	Mon 11/06/2	Wed 11/08/2	17	
22		Find students to interv	5 days	Mon 11/06/2	Fri 11/10/23	17	
23		Carry out student inte	7 days	Mon 11/13/2	Tue 11/21/23	21;22	
24		Use data from the inv	4 days	Wed 11/22/2	Mon 11/27/2	20;23	
25		**Software Design**	**14 days**	**Tue 11/28/2**	**Fri 12/15/23**	**16**	
26		Create use case diagra	7 days	Tue 11/28/2	Wed 12/06/2		
27		Create architectural m	7 days	Thu 12/07/2	Fri 12/15/23	26	
28		Create ERD for how u	7 days	Thu 12/07/2	Fri 12/15/23	26	
29		**Build Software**	**56 days**	**Mon 12/18**	**Mon 3/04/2**	**25**	
30		Sprint 1	14 days	Mon 12/18/2	Thu 1/04/24		

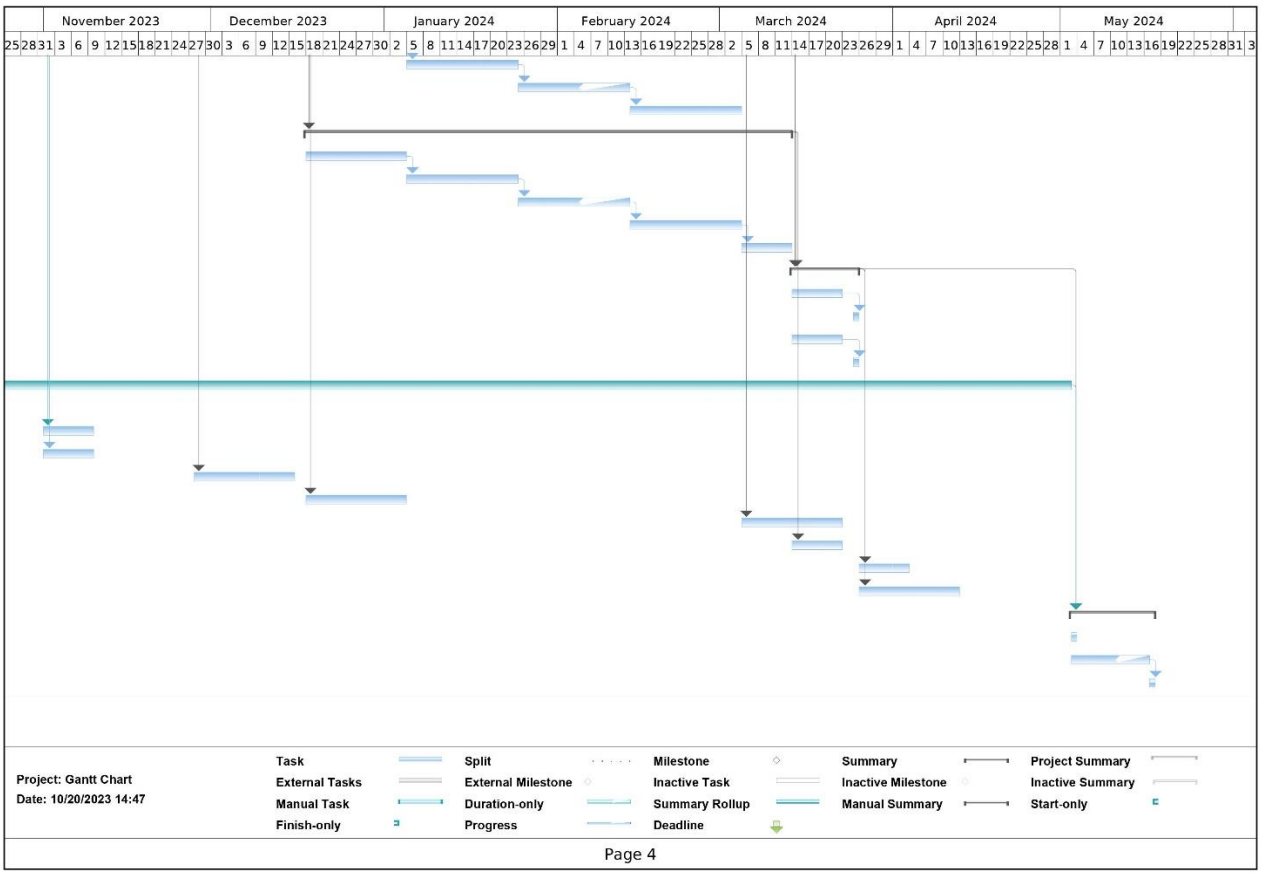
Page 1



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
31		Sprint 2	14 days	Fri 1/05/24	Wed 1/24/24	30	
32		Sprint 3	14 days	Thu 1/25/24	Tue 2/13/24	31	
33		Sprint 4	14 days	Wed 2/14/24	Mon 3/04/24	32	
34		**Testing**	**63 days**	**Mon 12/18**	**Wed 3/13/2**	**25**	
35		Testing Sprint 1	14 days	Mon 12/18/2	Thu 1/04/24		
36		Testing Sprint 2	14 days	Fri 1/05/24	Wed 1/24/24	35	
37		Testing Sprint 3	14 days	Thu 1/25/24	Tue 2/13/24	36	
38		Testing Sprint 4	14 days	Wed 2/14/24	Mon 3/04/24	37	
39		Post Production Testin	7 days	Tue 3/05/24	Wed 3/13/24	38	
40		**Release Application**	**8 days**	**Thu 3/14/2**	**Mon 3/25/2**	**29;34**	
41		Create IOS Listing	7 days	Thu 3/14/24	Fri 3/22/24		
42		Release IOS App to Ap	1 day	Mon 3/25/24	Mon 3/25/24	41	
43		Create Android Listing	7 days	Thu 3/14/24	Fri 3/22/24		
44		Release Android App to	1 day	Mon 3/25/24	Mon 3/25/24	43	
45		**Project Report**	**164 days**	**Mon 9/18/2**	**Thu 5/02/24**		
46		Write up introduction	7 days	Mon 10/09/2	Tue 10/17/23	1	
47		Write up literature rev	7 days	Wed 11/01/2	Thu 11/09/23	12	
48		Write up methodical a	7 days	Wed 11/01/2	Thu 11/09/23	15	
49		Write up requirments g	14 days	Tue 11/28/2	Fri 12/15/23	16	
50		Write up software des	14 days	Mon 12/18/2	Thu 1/04/24	25	
51		Write up build software	14 days	Tue 3/05/24	Fri 3/22/24	29	
52		Write up testing	7 days	Thu 3/14/24	Fri 3/22/24	34	
53		Write up evaluation	7 days	Tue 3/26/24	Wed 4/03/24	40	
54		Write up conclusion	14 days	Tue 3/26/24	Fri 4/12/24	40	
55		**Close Project**	**11 days**	**Fri 5/03/24**	**Fri 5/17/24**	**40;45**	
56		Submit Project	1 day	Fri 5/03/24	Fri 5/03/24		
57		Prepare Presentation	10 days	Fri 5/03/24	Thu 5/16/24		
58		Present Presentation	1 day	Fri 5/17/24	Fri 5/17/24	57	

Page 2


November 2023
December 2023
January 2024
February 2024
March 2024
April 2024
May 2024
Project: Gantt Chart
Date: 10/20/2023 14:47
Task
External Tasks
Manual Task
Finish-only
Split
External Milestone
Duration-only
Progress
Milestone
Inactive Task
Summary Rollup
Deadline
Summary
Inactive Milestone
Manual Summary
Project Summary
Inactive Summary
Start-only
Page 3


November 2023
December 2023
January 2024
February 2024
March 2024
April 2024
May 2024
Project: Gantt Chart
Date: 10/20/2023 14:47
Task
External Tasks
Manual Task
Finish-only
Split
External Milestone
Duration-only
Progress
Milestone
Inactive Task
Summary Rollup
Deadline
Summary
Inactive Milestone
Manual Summary
Project Summary
Inactive Summary
Start-only
Page 4

Appendix C: Ethics Certificate

Certificate of Ethics Review

Project title: Graduation Gathering

Name:	Adam Murray	**User ID**:	2166905	**Application date**:	02/11/2023 13:02:23	**ER Number**:	TETHIC-2023-106661

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Elisavet Andrikopoulou**, **Kirsten Smith**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**

What is your primary role at the University?: **Undergraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Dr Gail Ollis**

Is the study likely to involve human subjects (observation) or participants?: Yes

Will you gather data about people (e.g. socio-economic, clinical, psychological, biological)?: No

Will your data collection be strictly limited to gathering anonymous insights about a particular artefact or research question (e.g. opinions, feedback)?: Yes
Confirm whether and explain how you will use participant information sheets and apply informed consent.: Yes, for the use of surveys I will include an information sheet and ensure that to start the survey the participant ticks a box to give consent. For interviews I will explain the information, give them a chance to ask questions and have them complete a consent form.
Confirm whether and explain how you will maintain participant anonymity and confidentiality of data collected: No personal data such as the participants names will be stored. The data will be stored on a password protected university of Portsmouth google drive with MFA.
Will the study involve National Health Service patients or staff?: No

Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No
Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No
Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g.

children, people with learning disabilities, unconscious patients, Portsmouth University students)?: No Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No
Will blood or tissue samples be obtained from participants?: No

Is pain or more than mild discomfort likely to result from the study?: No

Could the study induce psychological stress or anxiety in participants or third parties?: No

Will the study involve prolonged or repetitive testing?: No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No

Could the research outputs potentially be harmful to third parties?: No

Could your research/artefact be adapted and be misused?: No

Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No
Please read and confirm that you agree with the following statements: I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc.), I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs, I confirm that I will act ethically and honestly throughout this project

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor comments: **Reviewed and approved. Minor wording changes suggested for ease of use by participants, with no ethical implications.**

Supervisor's Digital Signature**:** **gail.ollis@port.ac.uk** Date**: 29/01/2024**

Faculty Ethics Committee Review

Ethics Rep comments:

Faculty Ethics Committee Member's Digital Signature(s)**:**

kirsten.smith@port.ac.uk Date**: 28/02/2024**

Appendix D: Questionnaire

Participant Information Sheet

PARTICIPANT INFORMATION SHEET

Name and Contact Details of Researcher: Adam Murray, up2166905@myyport.ac.uk

Name and Contact Details of Supervisor: Dr Gail Ollis, gail.ollis@port.ac.uk

Ethics Committee Reference Number: TETHIC-2023-106661

1. Invitation

I am a final year student studying Computer Science BSc at the University Of Portsmouth. I would like to invite you to take part in my research study. Joining the study is entirely up to you, before you decide I would like you to understand why the research is being done and what it would involve for you. Please feel free to talk to others about the study if you wish. Taking part will take about 5 minutes. Please ask/contact us if you have any questions.

2. Study Summary

The purpose of this study is to understand what features both members of staff and students of the University Of Portsmouth desire from an app designed to connect them on graduation day.

3. What data will be collected and / or measurements taken?

The only data that will be collected will be your answer to whether you are a member of staff or a student, as well as a list of ID numbers relating to the features you have ordered on the website, IDs for which graduation zones you want, and finally the text that you input for the question asking about any ideas from you, about where you would like graduation zones to be designated. No personally identifiable data will be collected. Data will be stored securely on a google drive that can only be accessed by members of the research team. It will be deleted after the end of the project.

4. Do I have to take part?

No, taking part in this research is entirely voluntary. It is up to you to decide if you want to volunteer for the study. You can still withdraw from the study at any time for any reason until you click submit on the questions page.

5. Expenses and payments

There is no payment for taking part.

6. What if there is a problem?

If you have a query, concern or complaint about any aspect of this study, in the first instance you should contact the researcher if appropriate. If there is a complaint please contact the Supervisor with details of the complaint. The contact details for both the researcher and any supervisor are detailed above.

Thank you

Thank you for taking time to read this information sheet and for considering volunteering for this research.

Continue

Consent Form

Back To Participant Information Sheet

CONSENT FORM

Title of Project: Graduation Gathering

Name and Contact Details of Researcher: Adam Murray, up2166905@myyport.ac.uk

Name and Contact Details of Supervisor: Dr Gail Ollis, gail.ollis@port.ac.uk

University Data Protection Officer: Samantha Hill, 023 9284 3642 or information-matters@port.ac.uk

Ethics Committee Reference Number: TETHIC-2023-106661

1. I confirm that I have read and understood the information sheet for this study.
2. I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason until I click submit on the questions page.
3. I understand that data collected during this study will be processed in accordance with data protection law as explained in the Participant Information Sheet.

I agree to take part in the above study

Questions

Back To Consent Form

Q1. Please select the following that applies to you:

- Student ☐
- Staff ☐
- Prefer Not to Say ☐

A Graduation zone in this context means an area of Portsmouth designated on the app to show people's locations.

Q2. Please arrange the following list of features from most important to you at the top, to least important to you at the bottom (if on a computer use the handles on the right, if on mobile hold and drag):

- Users of the app are verified as University of Portsmouth staff or graduates
- You can be selective about which graduation zones you can be seen in by others
- You can be selective about which graduation zones you can see others in
- You can filter which users you see by course
- You can filter which users you see by name
- You can filter which users you see by whether you've already met
- A messaging service to allow you to communicate with others

Q3. For each of the following potential graduation zones, please selected whether you want it as a zone, don't mind, or don't want it as a zone:

Guildhall square: I want this as a zone

Ravelin park: I want this as a zone

The area between Guildhall and Ravelin park: I want this as a zone

Q4(Optional). Please enter any other parts of Portsmouth you would like to be designated as graduation zones:

Enter text here...

Q5(Optional). Please enter any other features you would like to see in this app:

Enter text here...

Submit