



Final Year Project Expo App

Technical Guide

Student 1 Name	Matt Simon Enriquez
Student 1 ID	20369441
Student 2 Name	Bien Dominic Managbanag
Student 2 ID	20415296
Supervisor	Dr. Jennifer Foster
Date Completed	21/04/2024

Abstract. This technical guide provides a detailed exploration through the motivation behind development of this app. It delves into the research that was conducted behind the scenes, illuminating the various factors and considerations that influenced decision making throughout the project. The guide documents the design process of the application, looking at the thoughts that crafted the interface and interactive features aimed at optimising the user experience. It gives a thorough examination of the implementation of all the work that was done, giving insights into the practical execution of the project, from setting up the development environment to testing and refinement. Furthermore, it explores the final results of the Final Year Project Expo Application, including an examination of samples of the code used . The guide also discusses any possible future ideas or work that can be done on the application for future expos and the potential adaptation of the application for other events.

Table of Contents

1. Motivation	5
1.1 Values and Beliefs	5
1.2 Personal Growth and Development	5
1.3 Long-term Vision	5
1.4 Creative Expression	5
2. Research	5
2.1 Frameworks and Languages	5
2.1.1 Researching Frameworks	5
2.1.2 Performance and Scalability	6
2.1.3 Database	7
2.1.4 Interactive Features	7
2.2 Gathering Data for the Application	7
2.3 User Interface	7
2.4 App Design	8
3. Design	8
3.1 Planning the Design	8
3.1.1 Defining the Design Goals	8
3.1.2 Creating Wireframes and Prototypes	9
3.1.3 Collaboration and Feedback	10
3.2 Interactive Map	10
3.3 Filtering	10
3.4 Database	10
3.5 Final UI Design	10
4. Implementation	11
4.1 Application	11
4.1.1 Project List	11
4.1.2 Filtering	11
4.1.3 Interactive Map	12
4.2 Database	12
4.3 Testing Methods (not sure if it goes here or in results)	13
4.3.1 Backend Unit Testing	13
4.3.2 User Acceptance Testing	13
5. Sample Code	14
5.1 Models	14
5.1.1 Student Programme	14
5.1.2 Students	15
5.1.3 Project Technology	15
5.1.4 Project Area	16

5.1.5 Project Location	17
5.2 Project List	17
5.3 Project Details Page	18
5.5 Filtering	18
5.4 Interactive Map	19
5.6 Unit Tests	21
6. Results	23
6.1 Application	23
6.2 Backend Unit Testing	26
7. Future Work	26

1. Motivation

1.1 Values and Beliefs

The application has an economic value as it can reduce paper waste since many project expo booklets are printed out. By digitising everything, users can access everything on their phone, contributing to environmental sustainability.

1.2 Personal Growth and Development

We have never made an app before and we wanted to challenge ourselves. We aimed to learn new languages, try different frameworks and experiment with user interfaces. We also wanted to go past our comfort zones and expand our skill sets.

1.3 Long-term Vision

We hope that the app will be developed further and will eventually be used by DCU for their future project expos. Additionally the app has the potential to be altered for other expos or conventions, ensuring ongoing progress and versatility.

1.4 Creative Expression

With making an application it is important that the application and interface itself is visually appealing whilst also being user-friendly and easy to navigate. Because of this our goal was to demonstrate that we are able to achieve a good balance between aesthetics and usability.

2. Research

2.1 Frameworks and Languages

When selecting the languages to use for developing the application, we considered multiple factors such as performance requirements, scalability and platform capabilities.

2.1.1 Researching Frameworks

When developing the app, our goal was to ensure compatibility with both Android and IOS platforms. Due to this requirement, we needed to choose a framework and programming language that allowed us to ensure that the application will have cross platform functionality to allow the app to be used by a wider range of users. During our research we found that the most suitable frameworks to use were React Native and Ionic.

React Native enables us to build the app using JavaScript and React with a wide range of libraries and tools to use. React Native is often referred to as “Learn Once, Write Everywhere”. This is because it allows developers to use a single codebase to build applications that can run on multiple platforms. This means that developers can learn the React Native framework and write code once, then deploy the same code to both iOS and Android platforms without significant modifications.

Ionic is an open source UI toolkit for developing high quality cross platform mobile and web applications using tech like HTML, CSS and JavaScript. The major objective of Ionic is “write once and use everywhere” meaning developers have to create one app for various platforms and devices meaning developers can easily build hybrid apps using Angular, Vue or JavaScript. It has easy documentation, Ui components and an easier learning curve.

Although both frameworks excel in app development, they both have different learning curves. As app development and both frameworks are generally new to us we had to take a look at the learning curve of both frameworks. With React Native the learning curve was quite steep as through our research we discovered that it was harder to learn and understand. However with Ionic, as it uses HTML, CSS and JavaScript/TypeScript we are essentially creating a web app and converting it into a mobile app, which is easier to learn meaning it takes less time to learn how to use the framework.

Ultimately, we opted to utilise Ionic with Angular as our technology stack due it being more flexible compared to React Native as we can use HTML, CSS and JavaScript to develop the app. This means it would take us less time to learn the framework and would enable us to focus more on developing the application more efficiently and focus on refining the application.

2.1.2 Performance and Scalability

Our decision making process involved us taking a close look into the performance and scalability requirements of our application. We considered factors such as execution speed, memory usage, cross platform support and maintainability. By selecting languages that are most optimised for these factors, we aimed to deliver a smoother and more efficient user experience whilst also ensuring the ability that the application could accommodate a growing workload and large amounts of data.

By prioritising performance, we aimed to ensure swift execution speeds across all sections of the application, thereby providing users with a seamless and responsive experience. Additionally, cross platform was an important factor, as it enables iOS and Android users to use the application.

Scalability was another concern we had to examine. We anticipated that there would be a need for the application to handle an increasing workload and accommodate vast amounts of data. Therefore, we chose languages and frameworks that offered scalability features to sustain our applications growth. This approach ensures that the app can handle growing demands over time.

2.1.3 Database

For our database, we wanted something that was easy to use and not difficult to convert into an online database. In this case, we looked at Microsoft SQL Server Management Studio. We took this software application into consideration since it works very well with Visual Studio. We could create the backend on Visual Studio and use SSMS to view the database. We could then use the features of SSMS to add data to the database.

2.1.4 Interactive Features

We looked into what kind of interactive features our application required. The list of these features can be summed down into the following points:

- An Interactive Map
 - The Interactive Map should prioritise being user friendly and easily accessible. Given that the expo is being held in multiple locations, we were given a map for each room. A way we can display this in the application is by having the user pick which room they would like to view the map of allowing them to navigate through the expo's various rooms.
- Filtering
 - When filtering users should be able to pick filters to specify which projects they want to look at. These filters would include:
 - Project Area
 - Technology
 - Programme (i.e Computer Applications, Biomed Engineering)
 - Area
- Project Description
 - The project description should contain all the project information such as a description of the project, the students who created them and their emails, the supervisors, project number, the project areas (i.e AI, Automation, Blockchain, ect.), the technology being used, and the location.

2.2 Gathering Data for the Application

In order to procure the data that was essential for the development of the application, we contacted the organisers of the DCU Project Expo to inquire about a meeting to discuss the required data for the application, to showcase the preliminary designs that we had created and obtain any feedback and to discuss about any added features they would like to add to the application.

2.3 User Interface

When determining how to approach the design of the User Interface (UI), we researched what the specific requirements and needs of the application were. These requirements can be summed into the following points:

- User-Friendly Interface

- The app should feature an intuitive and visually appealing interface that allows the users to navigate through the app efficiently, search for specific projects, and access relevant information without difficulty.
- Interactive Features
 - The app should include interactive features such as filtering options and the interactive map.
- Short Loading times
 - The app should be able to seamlessly transition from one section to another.

2.4 App Design

When researching application designs, we found that it was critical to ensure that the usability of the User Interface (UI) of the application was user-friendly to deliver a seamless user experience. We decided to focus not only on the visual and interactive aspects of the application, but also on the overall user experience, functionality, content, information architecture, and the flow of the application. We acknowledged the importance of creating a cohesive and seamless experience for the users the moment they open the app. To do this we needed to ensure that the app's functionality was intuitive and easy to navigate.

3. Design

3.1 Planning the Design

3.1.1 Defining the Design Goals

We planned the design of the app. We identified several design goals that were crucial for developing our app, ensuring that the creation of the app was as efficient as possible. These goals were:

- Usability
 - The app should be intuitive and easy to navigate for users of all technical proficiency.
- Intuitiveness
 - The app should be easy to understand and navigate, even for first time users. An intuitive and user-friendly user interface design ensures that the users can navigate through the app with ease.
- Scalability
 - The app should be able to scale to accommodate an increasing number of users and data over time.
- Performance
 - The app should be fast, responsive, and reliable. Optimising this would ensure smooth interactions and minimise the loading times, enhancing the user experience.
- Consistency

- Consistent design elements such as layout and colour schemes allow the app to be more efficient, create familiarity, and to ensure the app has a smoother flow.

3.1.2 Creating Wireframes and Prototypes

We created multiple wireframes to provide a visual representation of the app's layout, structure, and flow. Wireframing allowed us to modify and refine the app in response to feedback from both the Project Expo Organisers and our project supervisor. Moreover, wireframing serves as a tool to enable us to define the functionality of the app by allowing us to map out the user interface and interactions helping us prioritise usability and intuitive navigation ensuring the app is user friendly and easy to navigate.

Wireframing also played a role in identifying and addressing any usability issues early in the development process. This allowed us to optimise our workflow and avoid delays that could have occurred in later development stages .

Below is the original designs of the application:

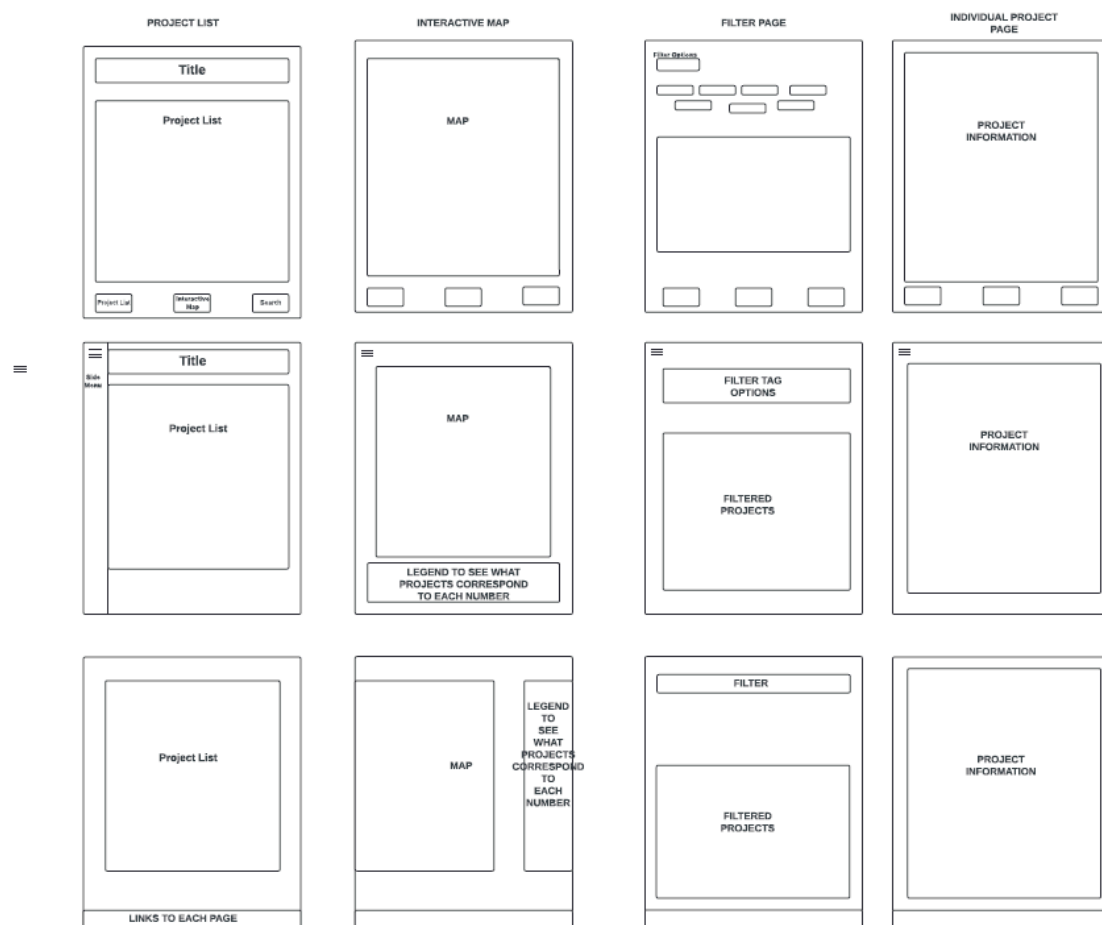


Figure 1. UI Design Wireframes

3.1.3 Collaboration and Feedback

We organised a meeting with the Project Expo Organisers to get their feedback on the initial designs of the application. The objective of this meeting was to gather their perspectives, opinions and specific requirements for the application. We integrated their feedback into the design process, with the objective of refining the final design to closely align with their expectations and needs. We also took into consideration any improvements or added functions that our supervisor suggested through weekly meetings, helping us further refine the application.

3.2 Interactive Map

Following the meeting, we obtained the maps from the Project Expo Organisers. When thinking about how to incorporate an interactive design to the maps, we came up with the idea of having the users choose which room they would like to see the map of. Once the map is chosen

3.3 Filtering

We designed the filtering system so that the users can filter by Project Area (i.e AI, Automation, and Machine Learning), Technology (Angular, Django, Machine Learning, ect), Programme (i.e Computer Applications, Engineering), and the Project Location. We decided to utilise a drop down menu system for the filters to make it easier for users to pick the specific filters they required. This design choice allows users to conveniently select their filters with ease, allowing for a more efficient exploration of the projects tailored to their specific interests.

3.4 Database

We designed our database based on the Excel sheet that was given to us by the marketing team in DCU. Each row contained a project with its details and information separated into columns. For example, project title, summary, area, technology, student, supervisor etc. The location of each project originally was not included in the sheet that was given to us so we had to find the location of each project and put it into a new column. This was necessary for the interactive map so the project could be mapped to its room. For columns such as Project Area, Project Technology and Project Students, we decided to put them into arrays since they can contain a different amount of parameters. Designing them as arrays made it easier to filter since it can return a certain parameter.

3.5 Final UI Design

After altering and refining the design of the application to fit the Project Expo's requirements and needs we finally came up with the design below:

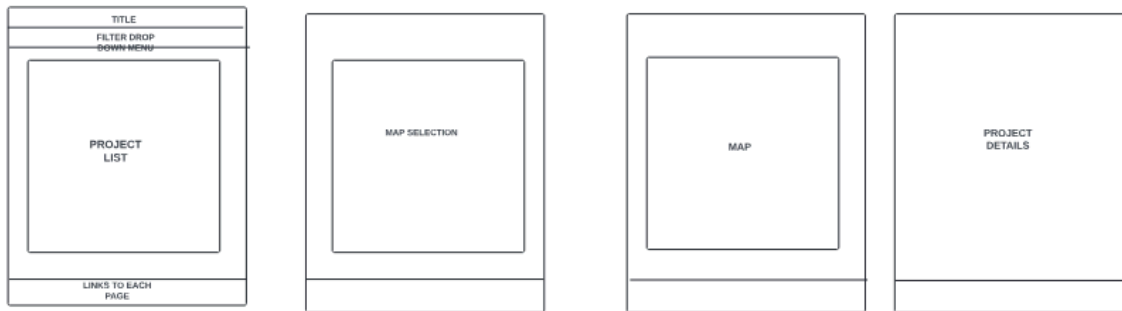


Figure 2. Final Wireframe Design of the Application

In this design, the main screen displays the project list, providing access to the project details. Beneath the title, users will find the filtering option, utilising a dropdown menu for efficient search refinement. Users will be able to filter by project area, technology used, programme and location. At the bottom of the screen users will be able to find links to access the maps and project details of the previously selected project.

When a user wishes to look at the maps they will be greeted with a list of the rooms the project expo is being held in. By selecting a room, users will be directed to the corresponding map, simplifying navigation. Once on the map, users can access the project details by selecting a project number on the map. Alternatively, they can navigate through the project list to access the same information, ensuring flexibility in access to the project details.

If a user navigates away from a project, either by returning to the map or the project list, and wishes to revisit the last project they viewed, they can easily do so by selecting the project details icon located on the bottom of the screen. This feature allows users to seamlessly access previously viewed project details without any hassle.

4. Implementation

4.1 Application

4.1.1 Project List

We used '`<ion-list>`' to structure the displayed project items. Each of the project items is generated using Angular's '`*ngFor`' directive, iterating over the '`data`' array to fill the list. Clicking onto a project item will bring you to the project description tab.

4.1.2 Filtering

We utilized Ionic's accordion component to create a collapsible "Filters" section. Within this section, we provided multiple dropdown select components for users to filter the list of projects. Each dropdown corresponds to a specific filter criteria such as area, technology, programmer, or location. Upon selecting an option from any of the dropdown menus, a corresponding event is triggered causing '`geProjectByArea`', '`getProjectByTechnology`',

'getProjectByProgramme', or 'getProjectByLocation' to filter the project list based on the selected criteria.

4.1.3 Interactive Map

When developing the maps tab, we decided to go for a dual state approach. The initial state being the "Select Map mode" where users will be able to pick the map they would like to access. The second state is the "Map mode" which provides the users with their desired map, showcasing all the projects that are available in that location with highlighted markers.

We decided to incorporate variables that were defined for each room, where each variable corresponds to a room and contains a list of project, along with their respective coordinates.

4.2 Database

Initially, the database was accessed locally using Swagger API. This involved running Visual Studio and clicking the https run button. However, as we developed the application much more, we decided it would be best to implement the database online. This was done by using Amazon Web Services. Through the AWS Plugin on Visual Studio, we were able to set up an Amazon Relational Database Service (RDS) through Elastic Beanstalk. The database is then hosted on an EC2 instance which is a virtual computer on the cloud. The database is now online. We then changed the Get API links on the frontend to the endpoint of the online database.

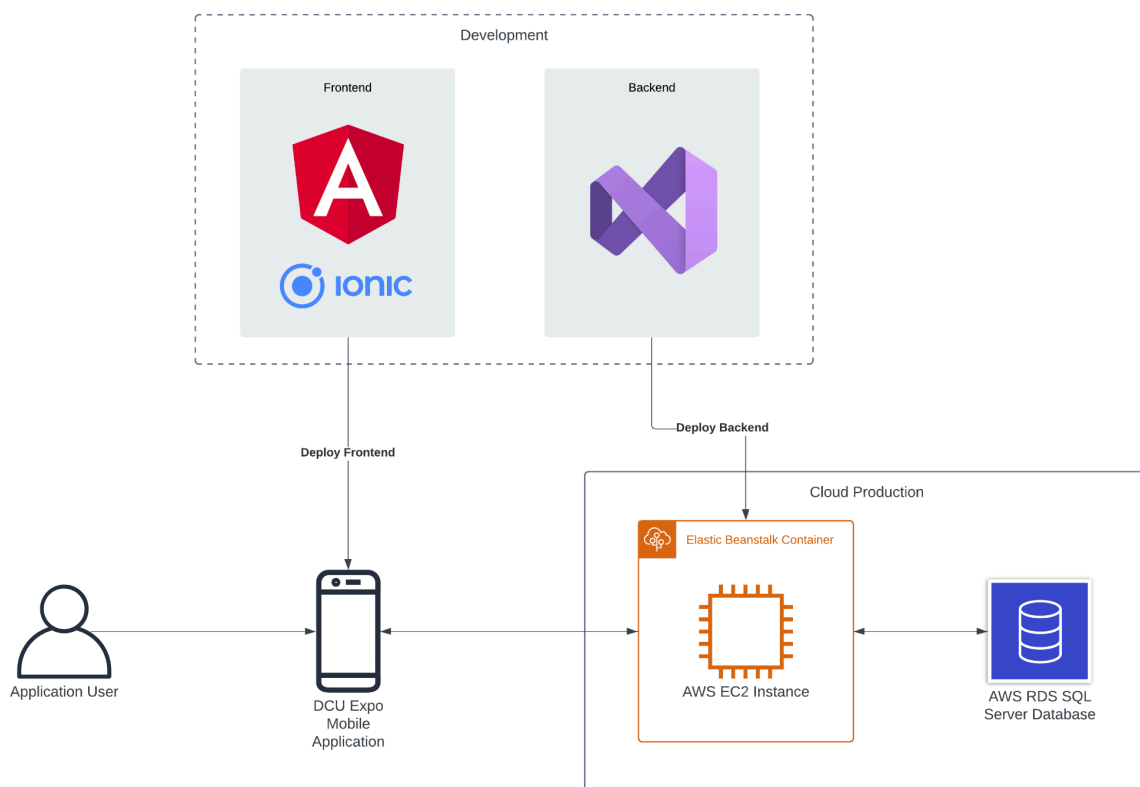


Figure 3. Architecture Diagram

4.3 Testing Methods

4.3.1 Backend Unit Testing

When it came to testing, we decided to focus on unit testing in the backend as we had to deal with a large amount of data. We found it more important to make sure that the data was correct and that it was returning and displaying correctly on the frontend. Unit testing was done through Visual Studio in the backend which is written in C#. We made six types of tests involving the Crud service functions, these include; `GetAllProjects()`, `GetProject()`, `GetProjectByProjectArea()`, `GetProjectByProjectTechnology()`, `GetProjectByStudentProgramme()`, and `GetProjectByProjectLocation()`.

We created a mock database with 10 projects called `mockExpoProjectsDbSet`. We also created another mock function of the Crud service. For this to work, we had to also create a mock context of `ExpoContext` which essentially means it will use the data of the projects defined and put it into a fake or pretend database.

Each original Crud service function was tested. Each function had 3 tests except for `GetAllProjects()`:

- `GetProjectCountTest()` - This function uses `GetAllProjects()` and asserts that there are 10 projects in the mock database.
- `GetProjectByIdTest()` - This function uses `GetProject()` and asserts the `ProjectId` is equal to the id that was passed in.
- `GetProjectByAreaTest()` - This function uses `GetProjectByProjectArea()` and asserts the `ProjectAreaName` string is equal to the string that was passed in.
- `GetProjectByTechnology()` - This function uses `GetProjectByProjectTechnology()` and asserts the `ProjectTechnologyName` string is equal to the string that was passed in.
- `GetProjectByProgramme()` - This function uses `GetProjectByStudentProgramme()` and asserts the `StudentProgrammeName` string is equal to the string that was passed in.
- `GetProjectByLocation()` - This function uses `GetProjectByProjectLocation()` and asserts the `ProjectLocation` string is equal to the string that was passed in.

4.3.2 User Acceptance Testing

This method of testing was conducted by ourselves and it involves testing the user interface and functionality of the application. This was a very simple form of testing as all we had to

do was put ourselves in the perspective of the user. This meant that we had to follow what the user wanted to do.

For example, the goal for the user could be viewing the details of a certain project. The user would have to select a project and they would be taken to the details tab. If they reached the details tab and the information is, we deemed this test successful. This was done locally and on the Android emulator. Other tests we conducted were reaching the map tab with the highlighted project by pressing the map button on the details tab and selecting a project from the map and reaching the details tab.

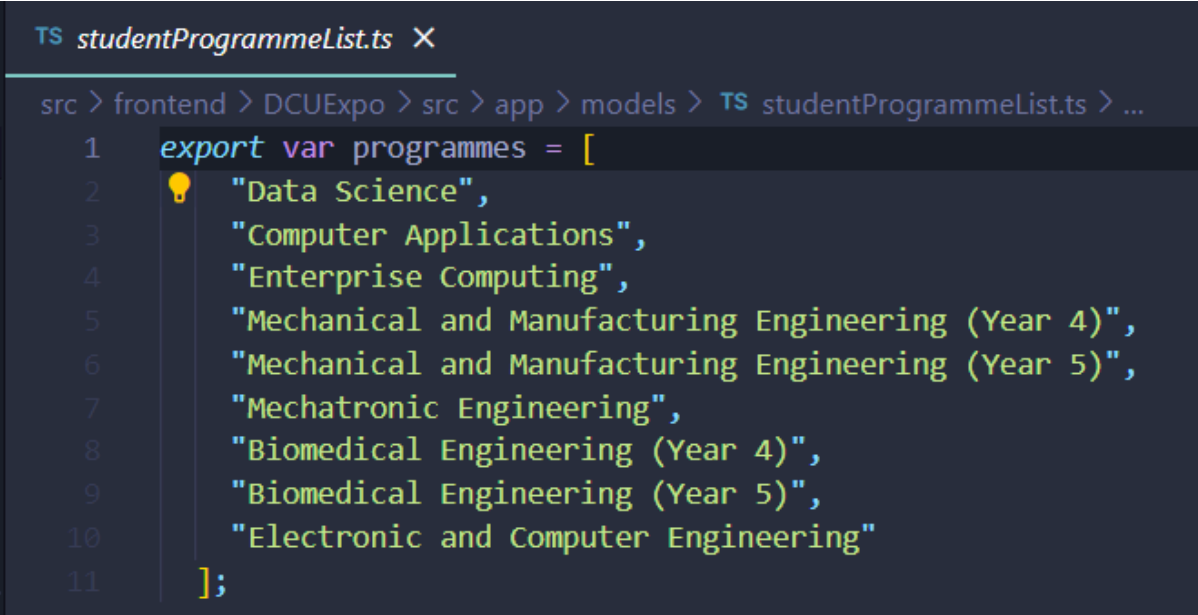
4.3.3 Ad hoc Testing

Ionic's live reload feature allowed us to conduct ad hoc testing. As we developed our app, we were able to see changes made to it in real time thanks to the live reload feature. For example, live reload supported us with styling as we were able to change the code and see changes very quickly without having to wait a while.

5. Sample Code

5.1 Models

5.1.1 Student Programme

A screenshot of a code editor window titled "TS studentProgrammeList.ts". The breadcrumb navigation shows the path: "src > frontend > DCUExpo > src > app > models > TS studentProgrammeList.ts > ...". The code is as follows:

```
1 export var programmes = [  
2   "Data Science",  
3   "Computer Applications",  
4   "Enterprise Computing",  
5   "Mechanical and Manufacturing Engineering (Year 4)",  
6   "Mechanical and Manufacturing Engineering (Year 5)",  
7   "Mechatronic Engineering",  
8   "Biomedical Engineering (Year 4)",  
9   "Biomedical Engineering (Year 5)",  
10  "Electronic and Computer Engineering"  
11 ];
```

Figure 4. Sample code for student programme List

```

TS studentprogramme.ts X
src > frontend > DCUExpo > src > app > models > TS studentprogramme.ts > StudentProgramme
1  export interface StudentProgramme {
2      studentProgrammeId : number
3      studentProgrammeName : string
4  }

```

Figure 5. Student Programme Sample Code

5.1.2 Students

```

TS projectstudents.ts X
src > frontend > DCUExpo > src > app > models > TS projectstudents.ts > ProjectStudents
1  export interface ProjectStudents {
2      id : number
3      studentName : string
4      studentId : number
5      studentEmail : string
6  }

```

Figure 6. List of Students

5.1.3 Project Technology

```

TS projectTechnologyList.ts X
src > frontend > DCUExpo > src > app > models > TS projectTechnologyList.ts > technologies
1  export var technologies = [
2      ".NET",
3      "3D slicer software",
4      "Amplify",
5      "Analog Electronics",
6      "Android Studio",
7      "Angular",
8      "AngularJS",
9      "ANSYS Fluent",
10     "ANSYS Workbench",
11     "API Gateway",
12     "AppSync",

```

Figure 7. Project Technology Sample Code

```
TS projecttechnology.ts X
src > frontend > DCUExpo > src > app > models > TS projecttechnology.ts > ProjectTechnology
1 export interface ProjectTechnology {
2     projectTechnologyId : number
3     projectTechnologyName : string
4 }
```

Figure 8. Project Technology Sample Code

5.1.4 Project Area

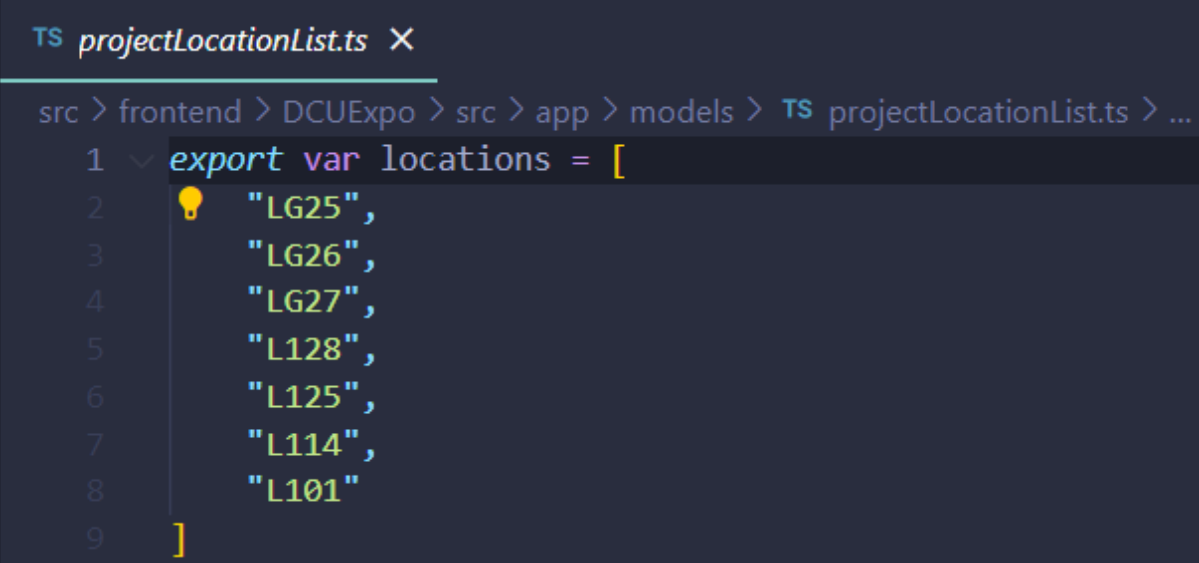
```
TS projectAreaList.ts X
src > frontend > DCUExpo > src > app > models > TS projectAreaList.ts > ...
1 export var areas = [
2     "3D Bioprinting",
3     "3D modelling",
4     "3-D Modelling",
5     "Additive Manufacturing",
6     "Advanced Material Engineering",
7     "Android",
8     "Arduino",
9     "Artificial Intelligence",
10    "Audio Processing",
11 ]
```

Figure 9. Sample Code of the Project Area

```
TS projectarea.ts X
src > frontend > DCUExpo > src > app > models > TS projectarea.ts > ProjectArea
1 export interface ProjectArea {
2     projectAreaId : number
3     projectAreaName : string
4 }
```

Figure 10. Project Area Sample Code

5.1.5 Project Location



```
TS projectLocationList.ts X
src > frontend > DCUExpo > src > app > models > TS projectLocationList.ts > ...
1 export var locations = [
2     "LG25",
3     "LG26",
4     "LG27",
5     "L128",
6     "L125",
7     "L114",
8     "L101"
9 ]
```

Figure 11. Project Location

5.2 Project List



```
58 <ion-list>
59   <ion-item *ngFor="let item of data" class="projectitem" (click)="selectProject(item)">
60     <ion-label>
61       <h2>{{ item.projectTitle }}</h2>
62     </ion-label>
63   </ion-item>
64 </ion-list>
65 <!-- <ion-spinner *ngIf="!dataLoaded"></ion-spinner> -->
66 <ngx-spinner bdColor="rgba(51,51,51,0.8)" size="medium" color="#fff" type="ball-scale-pulse">
67   <p style="font-size: 20px; color: white">Loading...</p>
68 </ngx-spinner>
```

Figure 12. Sample code of the Project List Page

5.3 Project Details Page

```
13 |     <ion-title size="large">Details</ion-title>
14 |   </ion-toolbar>
15 | </ion-header>
16 |
17 |   <ion-content class="ion-padding" *ngIf="selectedId > 1">
18 |     <h2 class="headings">{{ data?.projectTitle }}</h2>
19 |     <p>{{ data?.projectSummary }}</p>
20 |
21 |     <h3 class="headings">Class</h3>
22 |     <p> {{ data?.studentProgramme?.studentProgrammeName }} </p>
23 |
24 |     <h3 class="headings">Project Area</h3>
25 |     <ion-item *ngFor="let item of data?.projectAreas">
26 |       <ion-label>
27 |         <p>{{ item.projectAreaName }}</p>
28 |       </ion-label>
29 |     </ion-item>
30 |
31 |     <h3 class="headings">Project Technology</h3>
32 |     <ion-item *ngFor="let item of data?.projectTechnologies">
33 |       <ion-label>
34 |         <p>{{ item.projectTechnologyName }}</p>
35 |       </ion-label>
36 |     </ion-item>
```

Figure 13. Sample Code of the Project Details Page

5.5 Filtering

```
12 |     <ion-title size="large">Project List</ion-title>
13 |   </ion-toolbar>
14 | </ion-header>
15 |
16 |   <ion-accordion-group #accordionGroup>
17 |     <ion-accordion value="filters">
18 |       <ion-item slot="header" color="light">
19 |         <ion-label>Filters</ion-label>
20 |         <ion-label class="header-center">{{searchString}}</ion-label>
21 |       </ion-item>
22 |       <div slot="content">
23 |         <ion-item>
24 |           <ion-select #area label="Filter by Area" placeholder="Area" (ionChange)="getProjectByArea($event.target.value)">
25 |             <ion-select-option *ngFor="let projectArea of areasList" [value]=projectArea>{{ projectArea
26 |               }}</ion-select-option>
27 |           </ion-select>
28 |         </ion-item>
```

Figure 14. Sample code of the Filtering Section on Project List Page

5.4 Interactive Map

```
6  export class MapService {
7
8      selectedMap : string = "None";
9      mode : string = "Map Select"
10     coordinates: any = null;
11
12     constructor() { }
13
14     setSelectedMap(newMap: string)
15     {
16         |   this.selectedMap = newMap;
17     }
18
19     setMode(newMode: string)
20     {
21         |   this.mode = newMode;
22     }
23
24     getSelectedMap(){
25         |   return this.selectedMap;
26     }
27
28     getMode(){
29         |   return this.mode;
30     }
31 }
```

Figure 15. Map States

```

12 export class ImageMapComponent implements OnInit {
13
14   @Input()
15   src: string = ""
16
17   @Input()
18   coordinates: ImageMapCoordinate[] | undefined;
19
20   @Input()
21   canEdit: boolean = false;
22
23   @Output('onClick')
24   onClick: EventEmitter<ImageMapCoordinate> = new EventEmitter();
25
26   constructor() { }
27
28   ngOnInit() { this.onAreaCreate(0,0) }
29
30   getCoordinateStyle(coordinate: ImageMapCoordinate): object {
31     return {
32       top: `${coordinate.y}%`,
33       left: `${coordinate.x}%`,
34       height: `${coordinate.height}%`,
35       width: `${coordinate.width}%`
36     };
37   }
38 }

```

Figure 16. Sample Code of the Image Map Components

TS imagedata.ts X

```

src > frontend > DCUExpo > src > app > data > TS imagedata.ts > coordinatesL101
1  import { ImageMapCoordinate } from "../image-map/image-map.component"
2
3  export var coordinatesL101: ImageMapCoordinate[] = [
4    {
5      name: '173',
6      x: 24,
7      y: 2,
8      width: 11,
9      height: 8
10   },
11   {
12     name: '174',
13     x: 50,
14     y: 2,
15     width: 11,
16     height: 8
17   },

```

Figure 17. Image Data Sample Code

5.6 Unit Tests

```
var data = expoProject.ToList();

var mockExpoProjectsDbSet = new Mock<DbSet<ExpoProject>>();
mockExpoProjectsDbSet.As<IQueryable<ExpoProject>>().Setup(m => m.Provider).Returns(data.AsQueryable().Provider);
mockExpoProjectsDbSet.As<IQueryable<ExpoProject>>().Setup(m => m.Expression).Returns(data.AsQueryable().Expression);
mockExpoProjectsDbSet.As<IQueryable<ExpoProject>>().Setup(m => m.ElementType).Returns(data.AsQueryable().ElementType);
mockExpoProjectsDbSet.As<IQueryable<ExpoProject>>().Setup(m => m.GetEnumerator()).Returns(data.GetEnumerator());
mockExpoProjectsDbSet.Setup(m => m.Add(It.IsAny<ExpoProject>())).Callback<ExpoProject>(data.Add);
return mockExpoProjectsDbSet;
```

Figure 18. Defining Mock ExpoProjectDbSet

```
public CrudService GetMockCrudService()
{
    var mockExpoDbSet = GetMockExpoProject();

    //Creating a mock ExpoContext (DbContext)
    var mockContext = new Mock<ExpoContext>();

    //Setup the mock context to use data from above, this makes a fake database
    mockContext.Setup(m => m.Projects).Returns(mockExpoDbSet.Object);

    //create a mockCrudService
    var crudService = new CrudService(mockContext.Object);

    return crudService;
}
```

Figure 19. Defining Mock ExpoContext and crudService

```
[Test]
0 references
public void GetProjectCountTest()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetAllProjects();

    Assert.AreEqual(10, results.Count);
}
```

Figure 20. GetAllProjects() Test. Note that the error line is because Assert is used in classic form

```

[Test]
✓ | 0 references
public void GetProjectByIdTest1()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetProject(88);

    Assert.AreEqual(88, results.ProjectId);
    Assert.AreEqual("CrypTic", results.ProjectTitle);
    Assert.AreEqual("L128", results.ProjectLocation);
    Assert.AreEqual("Dr Geoff Hamilton", results.SupervisorsName);
}

```

Figure 21. GetProject() Test

```

[Test]
✓ | 0 references
public void GetProjectByAreaTest1()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetProjectByProjectArea("Artificial Intelligence");

    Assert.True(results.Any(project => project.ProjectId == 5));
    Assert.True(results.Any(project => project.ProjectTitle == "Size and Fit Prediction in Fashion E-Commerce"));
    Assert.True(results.Any(project => project.ProjectLocation == "LG25"));
    Assert.True(results.Any(project => project.SupervisorsName == "Prof Alan Smeaton"));
}

```

Figure 22. GetProjectByProjectArea() Test

```

[Test]
✓ | 0 references
public void GetProjectByTechnology1()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetProjectByProjectTechnology("Python");

    Assert.True(results.Any(project => project.ProjectId == 60));
    Assert.True(results.Any(project => project.ProjectTitle == "Smart Commute"));
    Assert.True(results.Any(project => project.ProjectLocation == "LG27"));
    Assert.True(results.Any(project => project.SupervisorsName == "Dr Stephen Blott"));
}

```

Figure 23. GetProjectByProjectTechnology() Test

```

[Test]
0 | 0 references
public void GetProjectByProgramme1()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetProjectByStudentProgramme("Computer Applications");

    Assert.True(results.Any(project => project.ProjectId == 88));
    Assert.True(results.Any(project => project.ProjectTitle == "CrypTic"));
    Assert.True(results.Any(project => project.ProjectLocation == "L128"));
    Assert.True(results.Any(project => project.SupervisorsName == "Dr Geoff Hamilton"));
}

```

Figure 24. GetProjectByStudentProgramme() Test

```

[Test]
0 | 0 references
public void GetProjectByLocationTest1()
{
    var crudService = GetMockCrudService();

    var results = crudService.GetProjectByProjectLocation("LG25");

    Assert.True(results.Any(project => project.ProjectId == 5));
    Assert.True(results.Any(project => project.ProjectTitle == "Size and Fit Prediction in Fashion E-Commerce"));
    Assert.True(results.Any(project => project.ProjectLocation == "LG25"));
    Assert.True(results.Any(project => project.SupervisorsName == "Prof Alan Smeaton"));
}

```

Figure 25. GetProjectByProjectLocation() Test

6. Results

6.1 Application

At the end of our development of the Final Year Project Expo App, we were able to develop a fully functional application that can be used for Android devices. However, due to time constraints, we had to cut out some of the initial features we were hoping to add and decided to focus on the functionality of the app with limited features. The app is most suitable for an expo goer with functions such as filtering and searching a project, navigating room maps and locating a project by using the map button. See screenshots below:

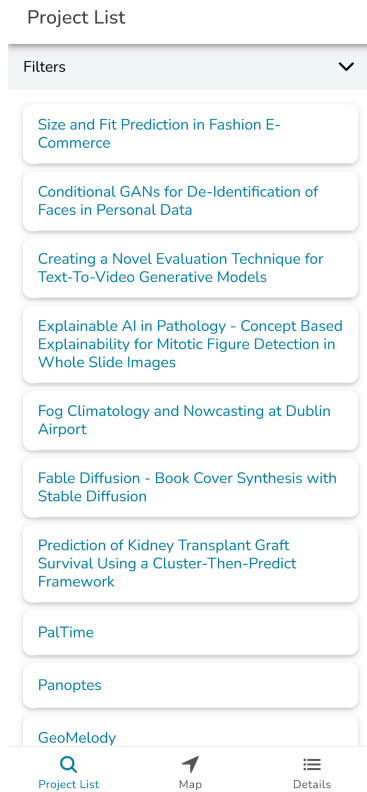


Figure 26. Project List tab

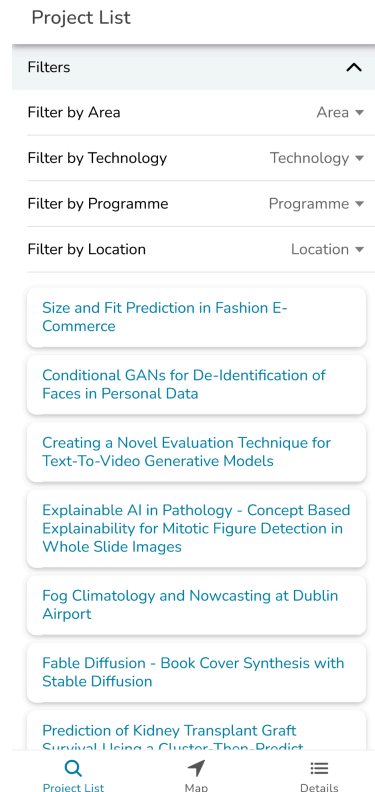


Figure 27. Filter option

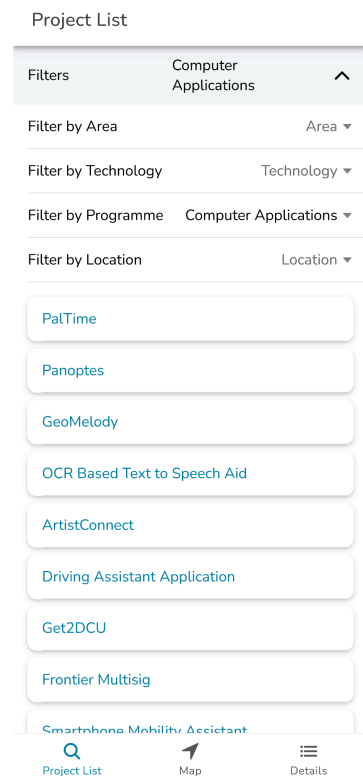


Figure 28. Filter selected

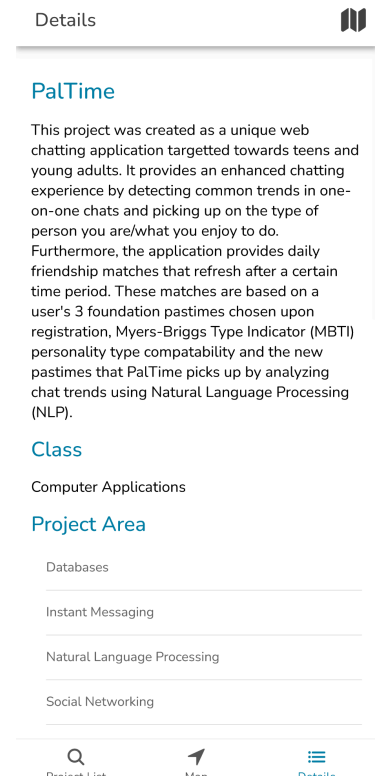


Figure 29. Project Details tab

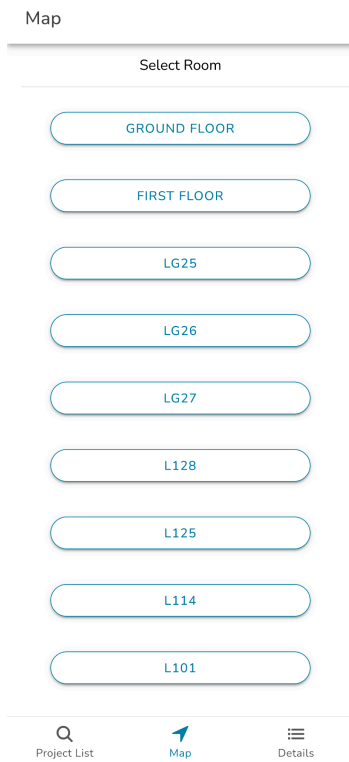


Figure 30. Map tab

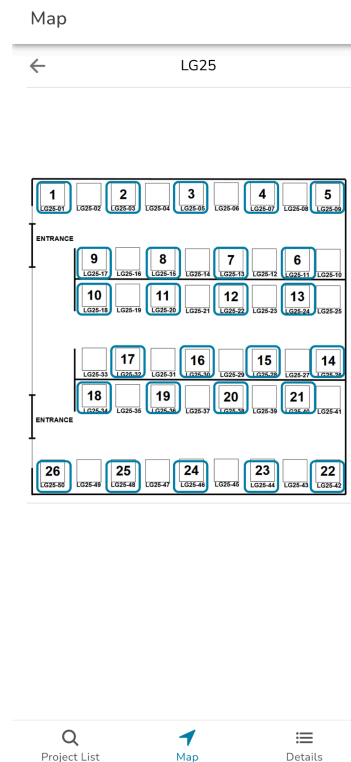


Figure 31. Room selected

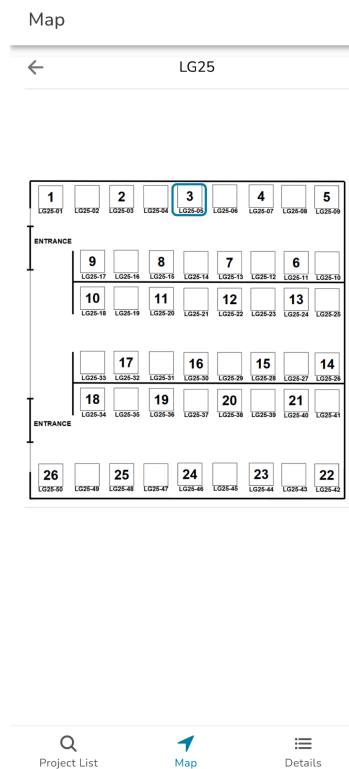


Figure 32. Highlighted project selection on map

6.2 Backend Unit Testing

We conducted 16 unit tests in the backend on Visual Studio. These tests were based on a mock database with 10 projects. Six crudService functions were tested and each test passed. See below for image of the tests:

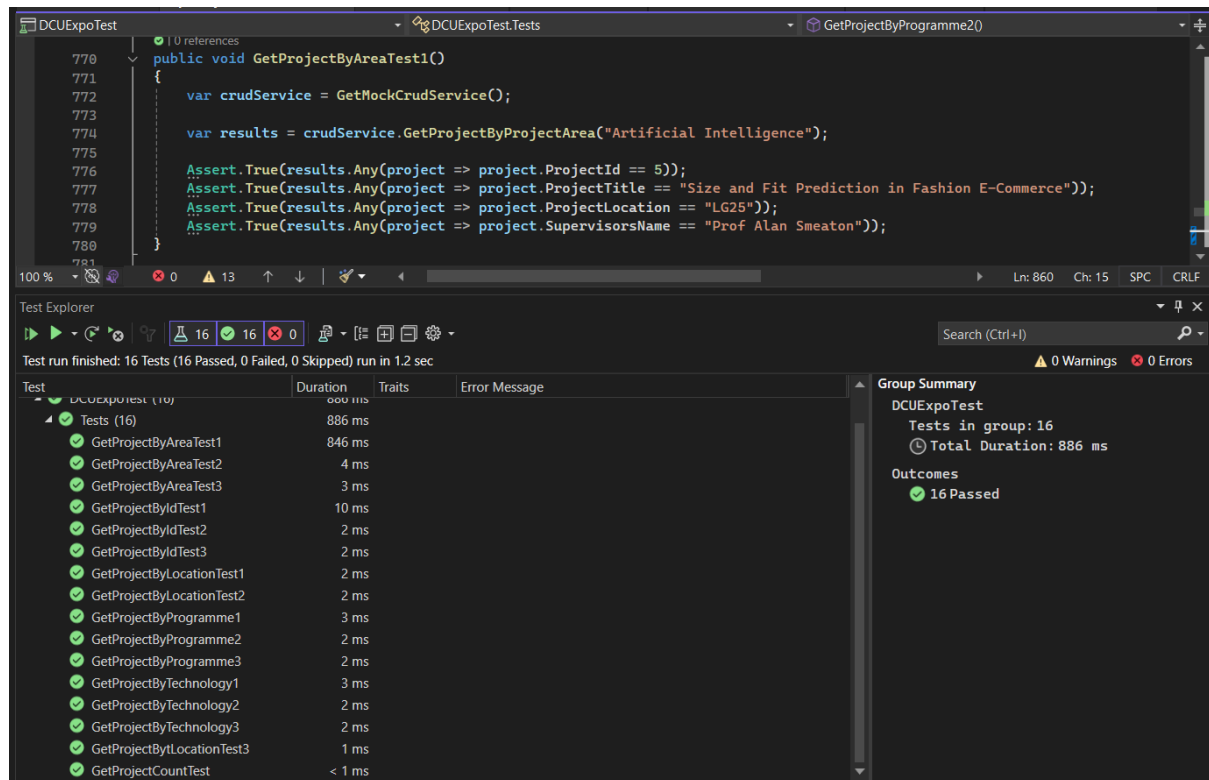


Figure 33. Unit Tests in Test Explorer

7. Future Work

As stated before, we intend for the app to be developed further to be used in future DCU Project Expos. These ideas are features that we thought of including but were not able to due to time constraints.

- Social Integration
 - Introduce more social features that will allow users to connect with each other, e.g share their favourite project, project discussions, social media platforms such as LinkedIn
 - Integration with social media can allow the users to create a form of community among users with similar interests
- Interactive Content
 - We hope to be able to enhance the interactive features by adding the ability to view video showcases of the projects
- Feedback Mechanism
 - Incorporating a feedback mechanism can allow users to review projects and contact the researchers. This can provide valuable insights to the researchers

and organisers and improve the quality of future expos and enhance the user experience of the application

- Multilingual Support
 - Allowing multilingual support can allow users from different countries to allow better accessibility.
 - Additionally it allows the possibility to expand the use of the expo application to universities in other countries.
- Analytics
 - We can implement an analytics tool or API which can allow the organisers to track app performance and user behaviour (e.g what are the most popular searches). This can allow the app to generate personalised content based on the user preferences or trending topics.
- Accessibility Features
 - For users with disabilities we can incorporate features such as reader support or including a high contrast mode to adjust the appearance of text and icons.