UCL Apps Portal

Team 9

Author:	*Author Email:*
Arjun Bahra	arjun.bahra.22@ucl.ac.uk
Isobel Barkley	isobel.barkley.22@ucl.ac.uk
Tania Turdean	tania.turdean..22@ucl.ac.uk

COMP0067: App Engineering

April 21, 2023

Department of Computer Science

University College London

Abstract

Every year, UCL students and staff develop many cutting-edge software projects. However, most of these projects are abandoned before they are able to achieve their full potential, as developers lack the knowledge, resources and funding to commercialise them. Currently, UCL does not offer centralised resources to aid with software publishing. Our client, Professor Dean Mohamedally of the UCL Computer Science department has tasked us with solving this problem by building the UCL Apps-Portal Website.

Our solution is a fully responsive web application tailored to UCL staff and students to guide them through the software publishing process. The application aims to educate users by providing informative resources, as well as connecting UCL software projects with potential investors, and other students. The solution was created using a React front-end, a Node.js and express back-end and a MySQL database hosted on Azure. The application was also integrated with the UCL API to authenticate UCL members and restrict functionality. After development and testing, the web app was deployed to a virtual machine, also hosted on Azure.

Overall, the project was a success, as the team were able to complete 100% of the "must-have" and "should-have" requirements, leaving only one "could-have" requirement unfulfilled due to time constraints. Through thorough testing, the team were able to eliminate many bugs, as well as raise some possible new features for further work, such as allowing UCL members to edit their listings once posted.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project Introduction

1.1.1 Client Introduction

Prof. Dean Mohamedally is a professor in Computer Science specialising in Software Engineering and Industry Projects.

1.1.2 Project Backend and Problem Statement

Every year, UCL students and staff develop many cutting-edge software projects. Most of these projects do not progress beyond the early stages, as developers lack the knowledge, resources and funding to commercialise them. Currently, UCL does not provide a complete and centralised resource to guide its students and staff through the software publishing process.

Therefore, this project is a fully responsive web application tailored to UCL staff and students to guide them through the software publishing process. The application will be compatible with both desktop and mobile devices.

1.1.3 Project Goals

The main goal of the project follows from the problem statement: increase the proportion of UCL software projects that develop into successful spin-outs. To achieve this, we aim to:

- Provide information about the software publishing process
- Advertise UCL software projects to potential investors
- Connect UCL students and staff members looking to collaborate on software projects

1.2 Project Management

1.2.1 Development Team

Isobel Barkley

Isobel graduated with a BEng in Mechanical Engineering from the University of Nottingham. There, she gained experience coding in MATLAB for fluid mechanics analysis, C for programming micro-controllers, and Python for a machine-learning related dissertation. She

also became familiarised with some basic HTML and CSS for web development during her summer internship at NatWest Group.

Arjun Bahra

Arjun earned a BEng in Aerospace Engineering from the University of Surrey. Throughout his academic journey, he acquired valuable programming experience in MATLAB for analysing control systems, and in Python for automating the structural analysis of aerostructures. Furthermore, Arjun expanded his programming expertise by learning C for his dissertation, which delved into the intricate study of aerosol dispersion within complex indoor environments.

Tania Turdean

Tania obtained a BSc Management Science degree at University College London with a minor in Machine Learning and Robotics. During this time, she gained programming experience through university projects such as applying algorithmics to solve HR problems, designing Python applications, and R projects with NLP, clustering or classification. Outside of class she taught Python courses for UCL Data Science Society and participated in Hackathons such as Google Kickstart to gain experience in solving algoritmic challanges in C++. She gained databases experience through a summer insternship at Bombinate where she was handling the data operations of the company and working in SQL to query their product data. Moreover, she has some frontend experience in HTML.

1.2.2 Scheduling

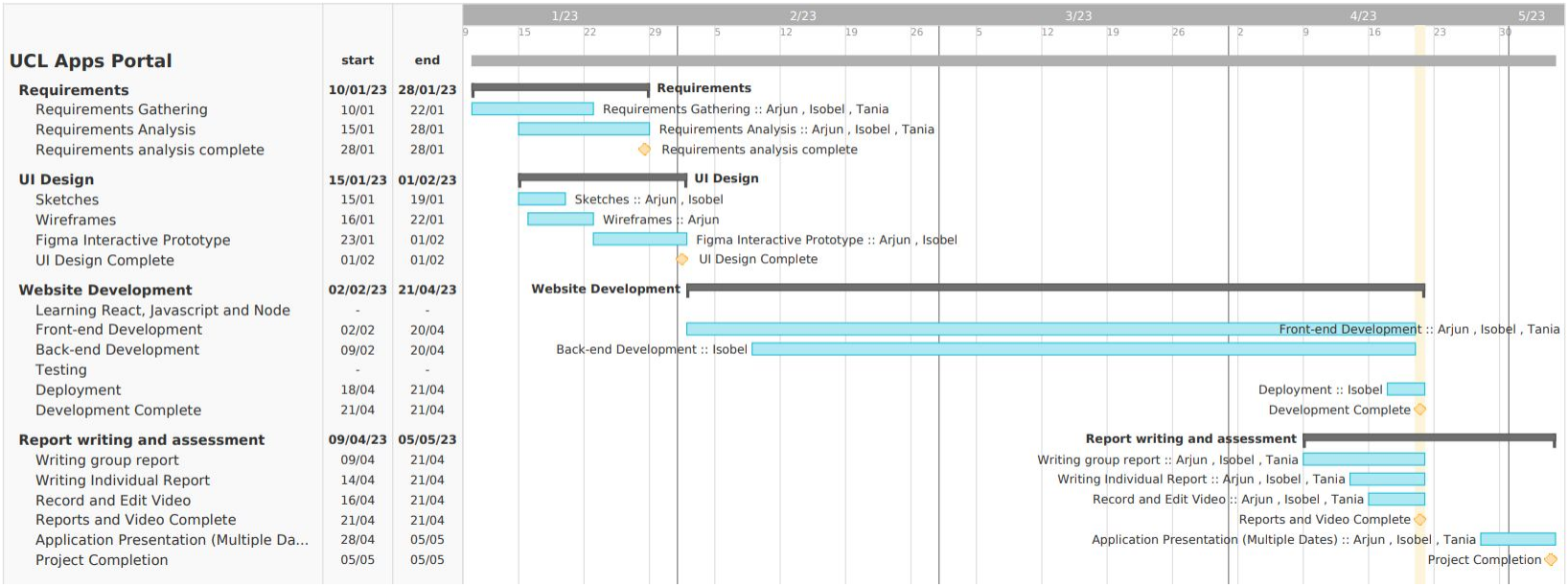

Figure 1: Gantt Chart

2 Requirement

2.1 Requirement Elicitation

To elicit the requirements, 5 different methods were used:

- **6 student teams interviews:** During the UCL Motion Input Computer Science undergraduate student presentation, the second-year undergraduate students were asked to present their projects in front of a board of investors. We managed to take 6 student teams structured interviews based on an open-ended question approach. The interviews allowed us to ask a lot of follow up questions. We worked in teams of two as while one was asking questions, the other was recording every response in a Google Form. We prepared a total of 8 questions related to previous experience in publishing, what would interest students in investors and job adverts.
- **Investor's observations** based on writing down the questions investors asked the teams that were presenting to understand what were the indicators that they were paying attention to.
- **3 postgraduate questionnaires:** We used the same google form questionnaire and sent it to 3 postgraduate students.
- **2 investors interviews:** We interviewed 2 potential investors: a past venture capital investor and a major stakeholder for a cosmetics company. We prepared 3 questions but did not hesitate to follow up for clarifications. Questions were related to what are investors looking for in projects and what is the proffered communication.
- **Previous design review:** We observed 3 other student websites from 3 leading universities to identify features and potential areas of improvement.
- **Professors Interviews:** We have interviewed 2 professors and asked them to imagine they are a website admin that needs to approve the posts.
- **Client meetings:** The meetings with Prof. Dean Mohamedally helped us gain an overall perspective into the project requirements. We have taken notes during each of the meetings.

2.2 Personas and Scenarios

2.2.1 Personas

Personas are fictional characters, user profiles that represent a particular type of user. Personas help in understanding users and their main characteristics. As a result, team members share a consistent understanding of the user groups. The most important aspect of the personas are that they provide directions when making design decisions.

To create the student personas, we have used the results from the student team interviews, student questionnaires, and the client meetings.

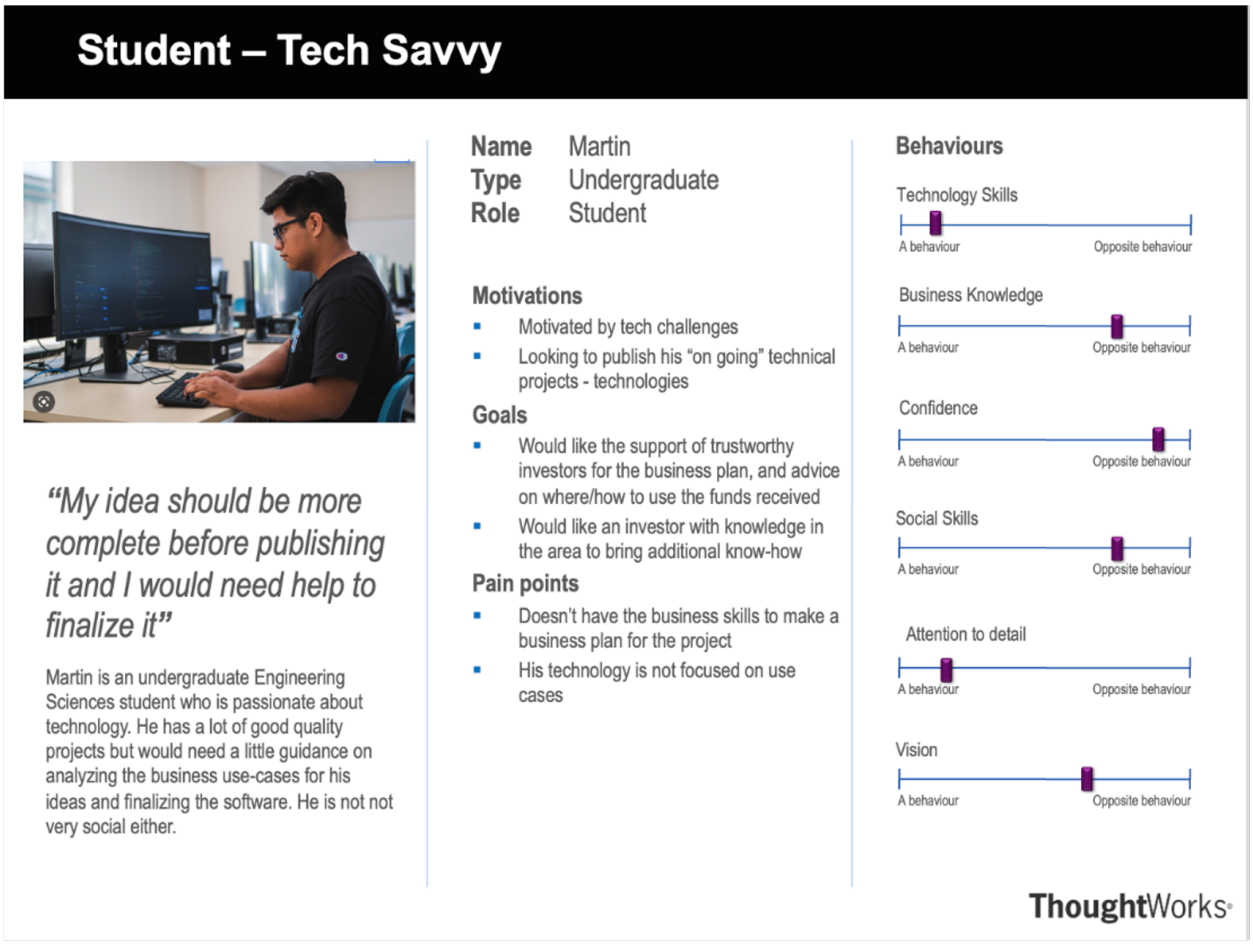

Figure 2.2: Student Persona 1

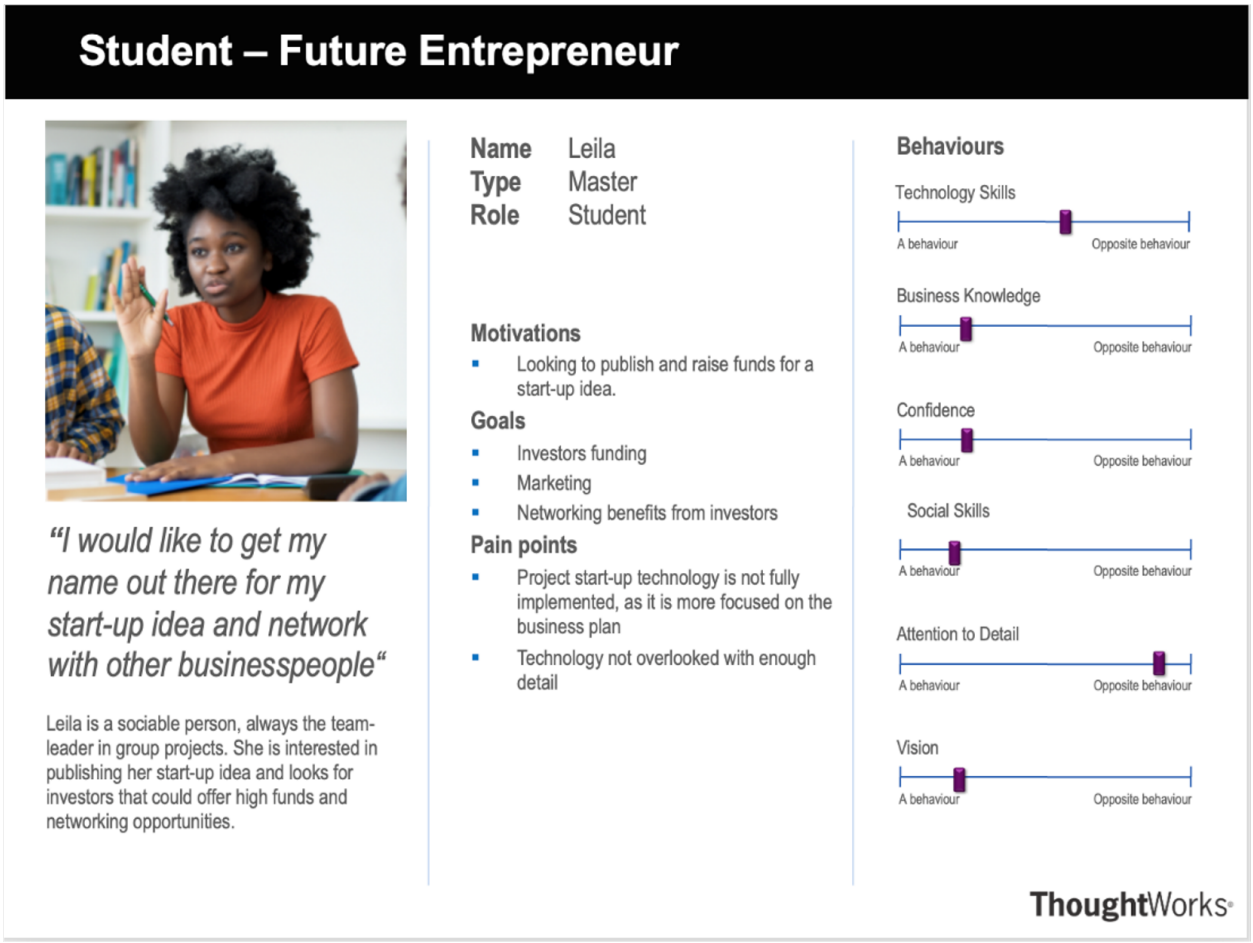

Figure 2.3: Student Persona 2

To create the investors personas, we have used the results from the investors' observations and investors' interviews.

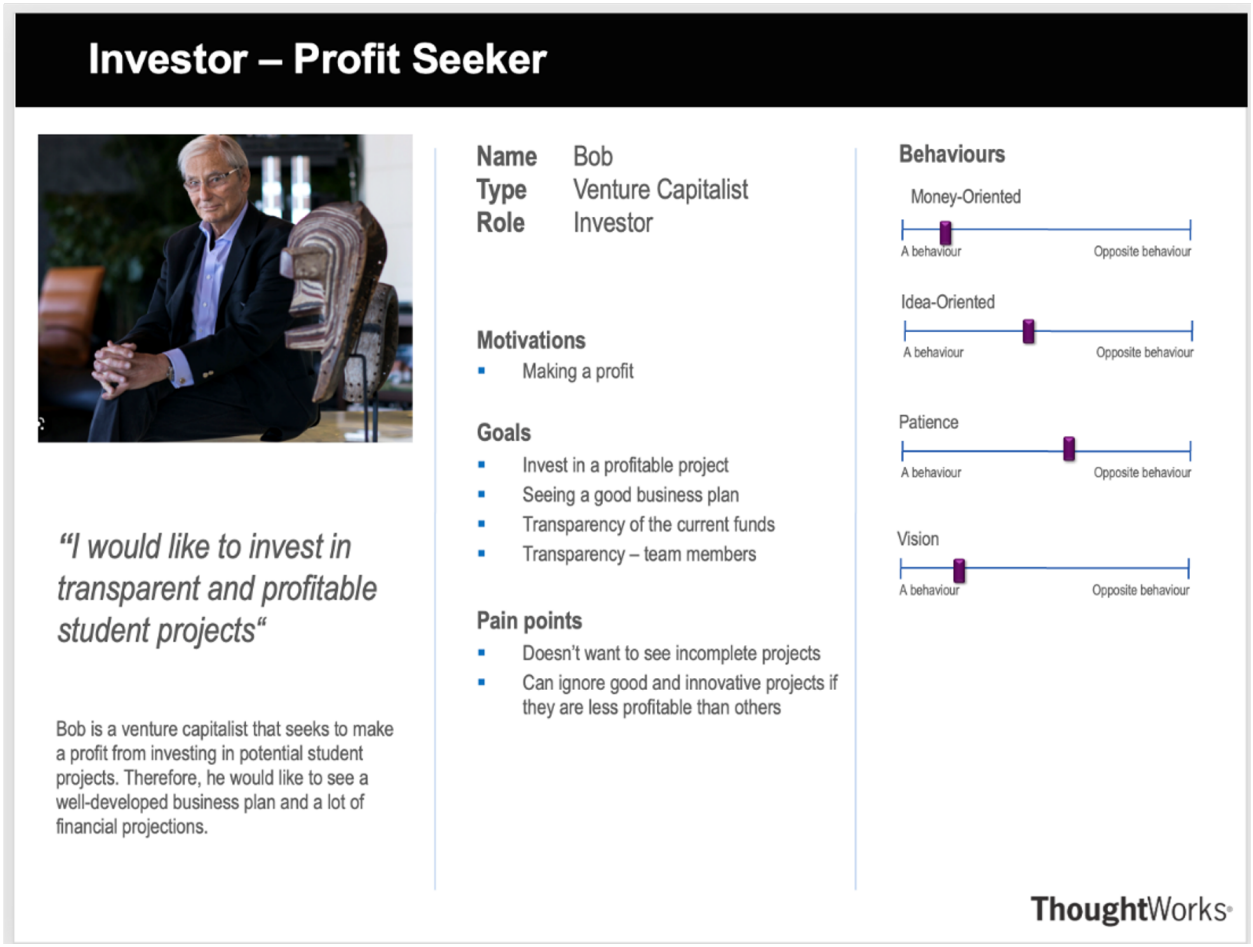

Figure 2.4: Investor Persona 3

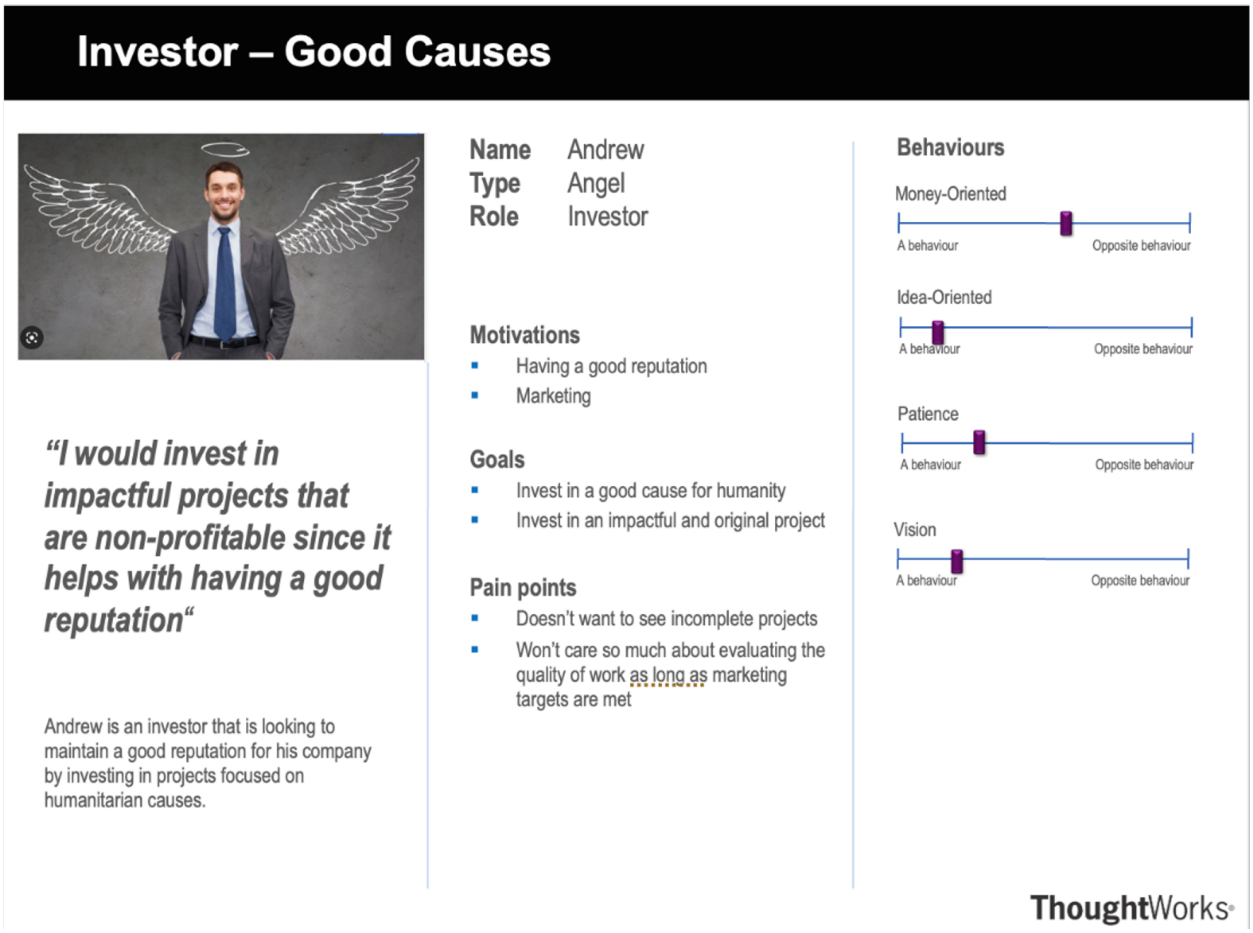

Figure 2.5: Investor Persona 4

To create the admin persona, we have used the results from the professors interviews and the client meetings.

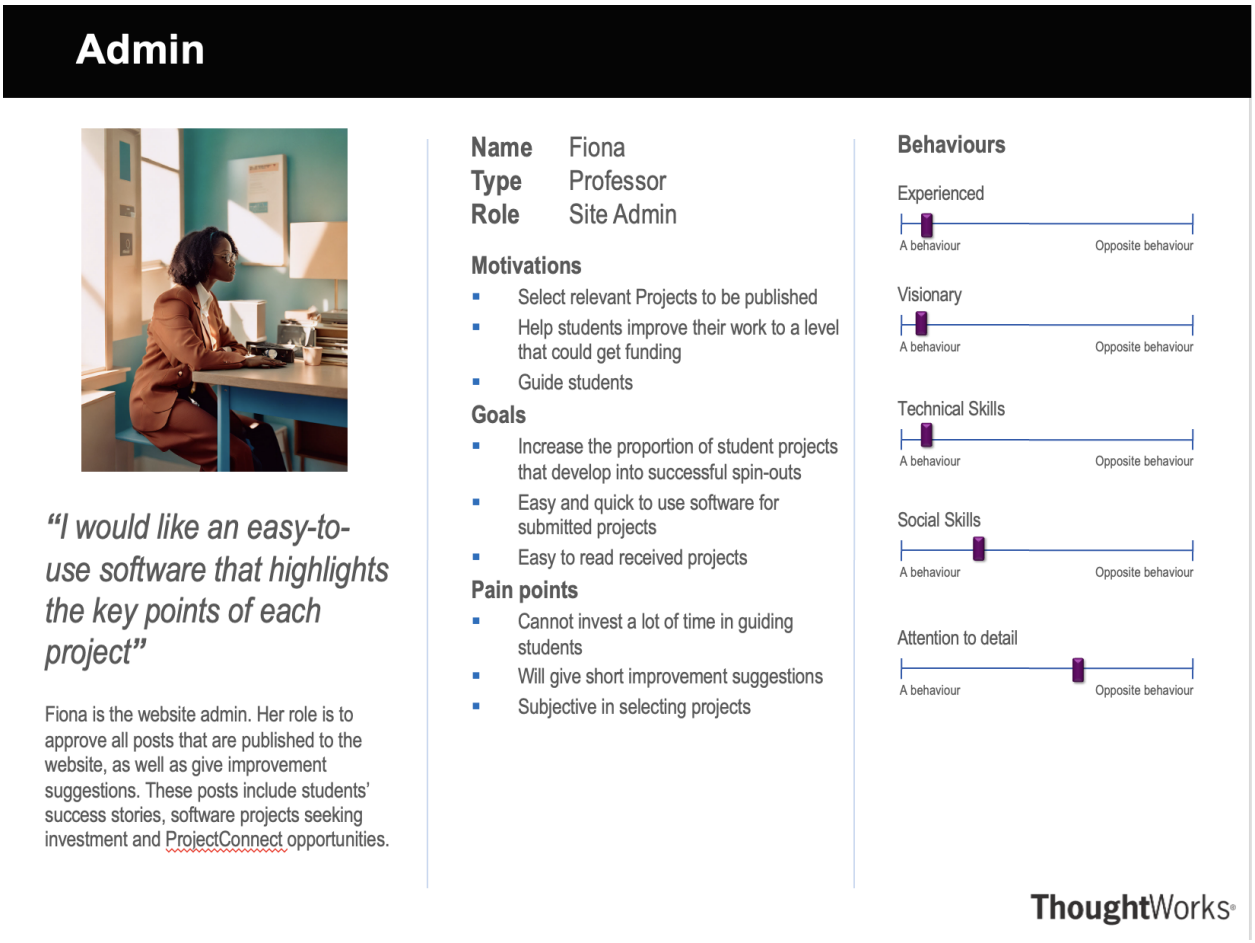

Figure 2.6: Admin Persona 5

2.2.2 Scenarios

Scenarios are stories of how the product might be used to achieve a goal. They help the engineers gain a shared understanding of the context and help discussing the needs and requirements.

How would the personas, Martin, Leila, Bob, Andrew and Fiona interact with the website?

Martin posts a ProjectConnect opportunity which is approved by the site admin. Leila sees his project on the ProjectConnect page and applies to help. Martin is happy that Leila uses her business skills to commercialize his technology. Leila researches on the past projects page for more inspiration and finishes the business plan. She posts the project on the apps portal. The project is approved by the admin and posted on current projects page.

Bob searches current projects that could turn into profit. Bob doesn't have a lot of patience but he has a good experience because the website is easy-to-use. Bob is motivated to invest in Leila's project because he sees the sum is already 50% raised by other important names. In the meantime, Andrew researches current projects and filters by humanitarian cause. He finds Leila and Martin's educational project and is motivated to invest in that cause. He is convinced by their video pitched idea.

2.3 MoSCoW requirement list

If your project include a mobile app and a web application, please produce a group of functional requirements and Non-functional requirements tables for each platform.

ID	Functional Requirements	Priority	Actors
FR1	The Apps Portal shall display **information about the software publishing process** including licensing and legal, marketing and communications, deployment and maintenance, and financials.	Must	UCL staff and students
FR2	The Apps Portal shall allow UCL staff/students to **create profiles on previously published software projects**, including details about product evolution, publishing and investment journey etc.	Must	UCL staff and students
FR3	The Apps Portal shall allow UCL staff/students to **create profiles on software projects seeking investment**, including details on business use cases and plans, financials, required funds, raised funds etc.	Must	UCL staff and students
FR4	The Apps Portal shall allow users to **browse profiles of past UCL staff/student software projects** as "success stories"	Must	All
FR5	The Apps Portal shall allow users to **browse profiles of current UCL staff/student software projects seeking investment**.	Must	All
FR6	The Apps Portal shall display information about **opportunities for UCL students** related to software development and testing, including **links to the relevant UCL NXI and UCL Finshing School webpages**.	Must	UCL staff and students
FR7	The Apps Portal shall allow UCL staff/students to post advertisements of opportunities related to their software project, available to other UCL staff/students (referred to as ProjectConnect opportunities).	Must	UCL staff and students

Continued on next page

FR8	The Apps Portal shall allow UCL staff/students to **browse advertisements related to UCL software projects**, posted by other UCL students/staff (ProjectConnect opportunities).	Must	UCL staff and students
FR9	The Apps Portal shall provide **links to the relavant UCL NXI and UCL Finishing School webpages to allow external investors to submit internship opportunities.**	Must	Non-UCL members
FR10	The Apps Portal shall restrict functionality only available to UCL members through **Single Sign On (SSO**	Must	UCL staff and students
FR11	TThe Apps Portal shall have a **single site admin.**	Must	Admin
FR12	The Apps Portal shall require the **site admin to approve all posts** (including software project profiles and opportunities) before they are published to the website	Must	Admin
FR13	The Apps Portal shall be able to **embed and play YouTube videos** from within project profiles.	Should	All
FR14	The Apps Portal shall have an **"About" section** including information about the UCL Apps Portal team, their contact details and site FAQs.	Should	All
FR15	The Apps Portal shall implement **unbiased and inclusive approaches to recruitment** wherever possible.	Could	All
FR16	The Apps Portal shall allow **documents** (including Word and PowerPoint documents) **to be embedded** in project profiles	Could	All
FR17	The Apps Portal shall allow users to **filter software projects** by categories including date posted, area of research etc.	Could	All
FR18	The Apps Portal shall allow the site admin to reset their password through **"Forgot my password".**	Won't	Admin
FR19	The Apps Portal shall allow external (non-UCL) members to post to the site.	Won't	Non-UCL members

Continued on next page

FR20	The Apps Portal shall allow UCL staff/students to **post an advertisement seeking an opportunity** in a particular sector.	Won't	UCL staff and students
FR21	The Apps Portal shall display standalone webpage **profiles/bios about UCL staff/students** detailing their background and any relevant past projects.	Won't	All
FR22	The Apps Portal shall **host applications** for student-opportunities.	Won't	UCL staff and students
FR23	The Apps Portal shall **store UCL staff/student data** in the database.	Won't	UCL staff and students

Table 2.1: Functional Requirements

ID	Non-Functional Requirements	Priority	Actors
NFR1	The Apps Portal shall be **scalable**, allowing it to easily incorporate new webpages and/or features	Must	N/A
NFR2	The Apps Portal shall use a **MySQL database**	Must	N/A
NFR3	The Apps Portal shall have a minimalist design so it is **easy to navigate and accesible** to as many users as possible, including those with low technical proficiency and/or visual impairments	Must	N/A
NFR4	The Apps Portal shall be **easy to maintain**, such that any system faults can be quickly identified and fixed	Must	N/A
NFR5	The Apps Portal shall be as **efficient** as possible, with minimal reponse times for users	Must	N/A
NFR6	The Apps Portal shall be reliable and available for the duration of UCL's official operating hours (8am - 7pm Monday - Friday) as a minimum, once made public	Must	N/A
NFR7	The Apps Portal shall be a fully responsive web app, compatible with a variety of browsers on **both desktop and mobile devices**	Must	N/A

Table 2.2: Non-functional Requirements

2.4 Use Cases

From the previous requirements analysis, a use case diagram and a use cases list were created to capture and display a summary of the interactions between the actors and the system. (Bittner and Spence, 2008).

2.4.1 Use Case Diagram

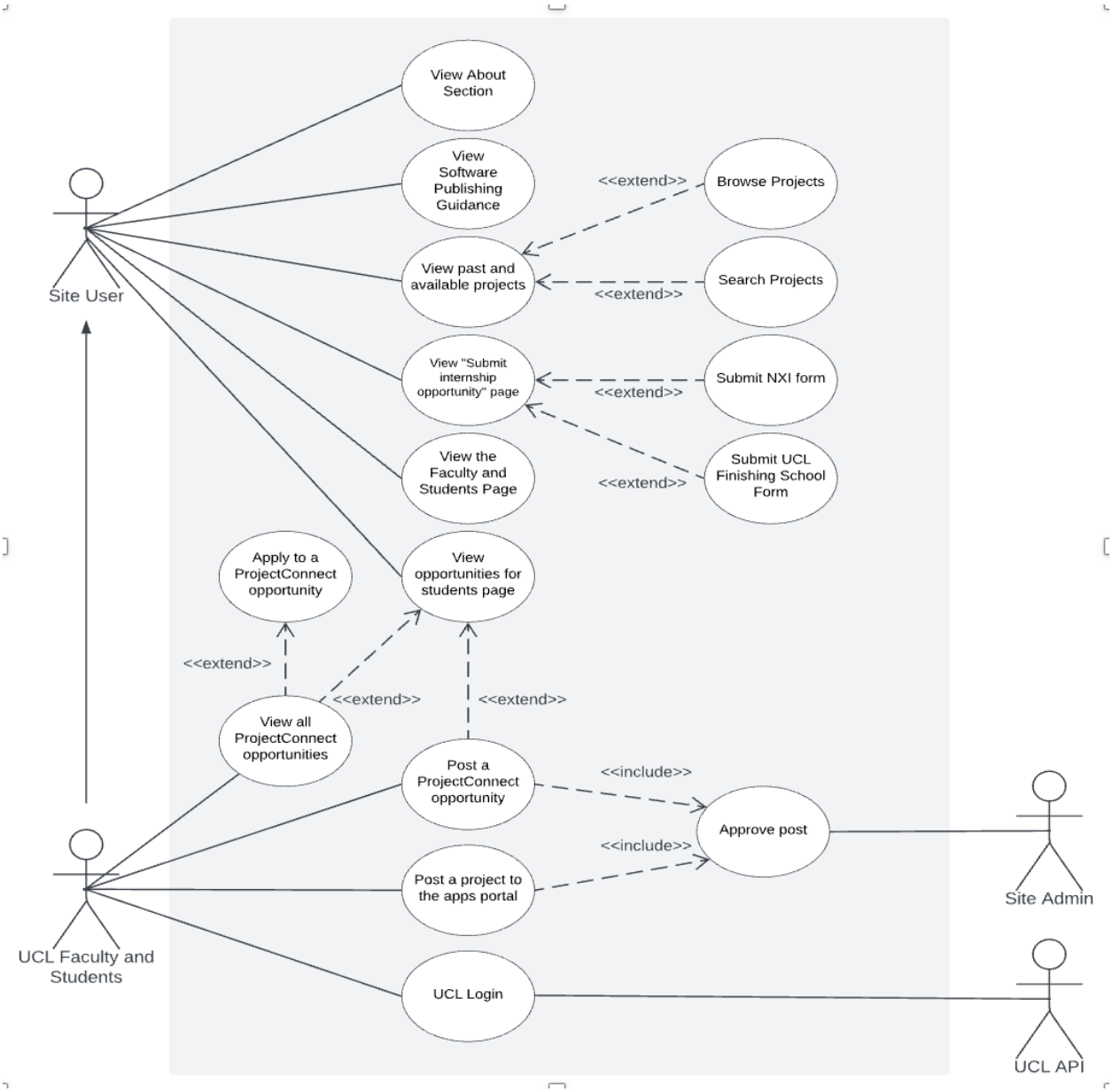

Figure 2.7: Use Case Diagram

ID	Use Case Name	Actor
UC1	Login	UCL staff and students
UC2	ViewPublishingProcess	All users
UC3	SubmitPastProject	UCL staff and students
UC4	SubmitCurrentProject	UCL staff and students
UC5	SubmitConnectOpportunity	UCL staff and students
UC6	BrowsePastProjects	All users
UC7	ViewPastProject	All users
UC8	BrowseCurrentProjects	All users
UC9	ViewCurrentProject	All users
UC10	BrowseProjectConnect	UCL staff and students
UC11	ViewNXIOpportunities	UCL staff and students
UC12	ViewPendingListings	Admin
UC13	ViewListing	Admin
UC14	ApproveListings	Admin
UC15	ViewAboutSection	All users
UC16	FilterCurrentProjects	All users
UC17	FilterPastProjects	All users

Table 2.3: Use Case List

2.4.2 Use Case List

2.4.3 Use Case Specification

Based on the user cast list, a series of user case specifications are formulated . This helps to better understand how users interact with the system, the scenarios they encountered and how the system should response.

Name:	ViewPublishingProcess
ID:	UC2
Brief Description	The user can view information about the software publishing process
Primary Actor(s):	All users
Secondary Actor(s):	System
Preconditions:	None
Main Flow:	1. The use case starts when the user hovers on the "Software Publishing Guidance" on the header 2. The system, displays 5 menu items: Overview and each step of the publishing 3. If the user clicks "Overview", then 3.1. The system displays an overview and a list of steps 4. If the user clicks "Licensing and Legal", then 4.1. The system displays "Licensing and Legal" guidance 5. If the user clicks "Financials", then 5.1. The system displays "Financials" guidance 6. If the user clicks "Marketing and Communications", then 6.1. The system displays "Marketing and Communications" guidance 7. If the user clicks "Deployment and Maintainace", then 7.1. The system displays "Deployment and Maintainance" guidance
Postconditions:	None
Alternative Flows:	None

Table 2.4: UC2

Name:	SubmitCurrentProject
ID:	UC4
Brief Description	UCL staff and students can create profiles on softwarre projects seeking investment
Primary Actor(s):	UCL staff and students
Secondary Actor(s):	System
Preconditions:	User is logged in through UCL SSO
Main Flow:	1. The use care starts when the user clicks "Submit your Technology" 2. The system asks the user to specify the title, investment goal, YouTube Link, GitHub Link, Pitch Deck Link 3. The system uses DB functions to retrieve the values for the select fields: sector, product type, investment offering 4. The system asks the user for the following (with 1000 characters limit) : product description, story, investment details 5. The system asks the user for one team member details 6. The system allows the user to add up to 3 team members 7. The user clicks "Submit"
Postconditions:	The data is added to the DB
Alternative Flows:	None

Name:	BrowseCurrentProjects
ID:	UC8
Brief Description	All users can watch a list of software projects seeking investments
Primary Actor(s):	All users
Secondary Actor(s):	System
Preconditions:	None
Main Flow:	1. Use case starts when the user clicks "Browse Available Technologies" 2. The system displays a list with the current projects by retrieving the data from the DB and shows information such as name, image, type, sector, investment goal, description.
Postconditions:	None
Alternative Flows:	None

Name:	ViewCurrentProject
ID:	UC9
Brief Description	All users can view the detailed profile of a current project.
Primary Actor(s):	All users
Secondary Actor(s):	The system
Preconditions:	User was browsing current projects and selected one.
Main Flow:	1. Use case starts when the user selects a project from the "browse" page. 2. The system displays the information about the project, including story, product description, YouTube embedded video, GitHub Link, Investment details, Pitch Deck Link, information about the team. 3. If the user clicks on one of the links then 3.1. The system directs the user to the link 4. If the user clicks on the video then 4.1. The system starts playing the embedded video
Postconditions:	None
Alternative Flows:	None

Name:	ApproveListings
ID:	UC14
Brief Description	The site admin can approve all posts before they are published to the website
Primary Actor(s):	Admin
Secondary Actor(s):	System
Preconditions:	The admin is logged in
Main Flow:	1. The use case starts when the admin logs in. 2. The admin sees the list of pending listings with the approve button. 3. If the admin toggles the approve for a listing then 3.1. The listing disappears from the pending page and is added to the Browse page
Postconditions:	The system marks the listing as "approved" in the D
Alternative Flows:	None

3 Research

3.1 Related Projects

Features+ Evaluation	Takeaways
Harvard Office of Technology Development (Harvard, no date)	
Easy to use – not a lot of menu tabs and clearly structured – move mouse to see subcategories without clicking	Have a menu with clear and not a lot of categories + scroll not click
Start-ups have description and person of contact (professor)	To add
Can browse start-up by category	Add filter
Impact section – past start-ups	Have current and past start-ups
Opportunity to license technology or create start-up	To add
Standford Office of Technology Licensing (Standford, no date)	
Irrelevant news page	Don't add
Careers – job search page – university jobs	Instead of careers page – can have start-up ideas opportunities to join or post ideas
Clear explanation of licensing process by steps	Explain licensing process by steps
Submit invention page	Add
Start-up guide	Explain start-up formation by steps
Explore technologies page - name and contributors	Add
MIT Technology Licensing office (MIT, no date)	
Difficult to browse – bad colors and can't see subcategories for main pages only with click	Have a clear menu, with appropriate color palette
Too many subcategories confuses users	Add few subcategories
Technologies have investors list but not sum	Have list of investors and sum invested and needed for each start-up
Cannot see who worked on each technology	Add contributors
Can browse by technology category (AI or Robotics)	Browse by technology category and add other filters such as by sum needed or browse by cause/topic (education, health)

Table 3.1: Related Projects

3.2 Related Technologies

Name	Advantages	Disadvantages
	Front End	
React	Large community with lots of resources and support	Can take some time and effort for beginners – uses a component-based architecture which can take some time for developers who are working with traditional HTML
	Virtual DOM provided fast rendering and improved performance	Requires "boilerplate code" – but there are tools that can simplify the process – Create React App
	Reusable components make it easy to maintain and develop large-scale applications	
Angular	Complete framework with built-in features and tools	Complex architecture can be overwhelming for beginners
	Two-way data binding simplifies development	Requires a lot of memory processing power
	Angular CTI streamlines project creation and management	Version upgrades can be disruptive
Vue	Lightweight and easy to learn	Limited community compared to React and Angular
	Simple syntax and template-based rendering makes it beginner friendly	Limited built-in features and tools
		Smaller ecosystem may require more manual configuration
	Back End	
Node.js	Built on JavaScript, making it easy for frontend developers to transition to backend development	Large ecosystem can be overwhelming for beginners
	Large ecosystem of libraries and modules	Not ideal for CPU intensive tasks
	Popular in the development of real-time applications and APIs	Callback coding style can be difficult to read and maintain

Continued on next page

PHP	Easy to learn Can run on any operating system Large community with many resources and support Works well with databases such as MySQL and PostgreSQL	Not well-suited for real time applications Limited OOP programming
Django	Robust framework with built-in features and tools Follows the MVC architecture for clean separation of concerns Admin panel comes built in	Requires more setup and configuration than other frameworks Relies heavily on Django ORM, which can slow down performance for large databases
Flask	Lightweight and modular framework Easy to learn and use for small projects and prototypes Good for building RESTful APIs	Not ideal for large and complex applications Limited built-in features and tools Flask extensions can be difficult to manage and update
Java	Mature and reliable technology Robust and secure with built-in features and tools Good for building large and complex applications Strong OOP capabilities	Large runtime and memory requirements Significant setup and configuration Can be slower than other languages for small-scale applications
Database Management System		
MySQL	MySQL is a relational database, so data is well organised in tables Free download available Compatible with most operating systems	Not efficient for handling large databases Hard to scale Transactions are not ACID compliant

Continued on next page

	The whole team has prior experience developing a database in MySQL from a previous module	
PostgreSQL	Object-relational database management system is good at handling complex data types	Not easy to install for beginners
	Fully ACID and SQL compliant	Not available on all hosts by default
	Requires low maintenance and admin services	Slower (than MySQL) in terms of performance
	Requires low maintenance and admin services	Team has no prior experience
MongoDB	High performance level (can execute queries fast since it stores most of the data in the RAM)	Team has no prior experience so would have to learn MongoDB query language from scratch.
	MongoDB query syntax is simpler than SQL	Joins in MongoDB can be difficult. Lack of joins functionality also leads to high memory usage.
	Easily scalable	Non-relational database is more likely to lead to duplication of data and corruption of the database.

Table 3.2: Related Technologies

3.3 Summary of Decisions

React was selected for the front-end because it is a very fast and modular framework, meaning high response times and high reusability of components throughout the website. Node.js was chosen for the back end as it comes with many built-in libraries and modules, reducing the number of utility functions that must be written from scratch. Finally, MySQL was chosen because the entire team has prior experience with it and it is a relational database, so data is clearly organised into tabular form. Whilst MySQL can be inefficient for handling large database, this is not an issue for this project, since the database will be relatively small.

The team only chose mature and popular technologies for the project to maximise the number of resources available. Furthermore, choosing React, Express, and Node allows us to code the whole project in JavaScript, therefore only requiring the team to learn a single language.

4 User Interface Design

4.1 Design Principals

Prior to formulating any design concepts, we carefully deliberated on the design principles as outlined by Roger, Sharp, and Preece (2011): visibility, feedback, consistency, constraints, affordance, and mapping.

For our website, the most pertinent principles encompass visibility, consistency, constraints, and mapping. We have prioritized the prominence of page links, ensuring seamless navigation and discoverability of all pathways within the site. Additionally, elements such as buttons, accordions, and fonts maintain uniformity throughout the pages. To optimize user experience, we have streamlined functionality by limiting access to the "Faculty and Students" sections for non-UCL members. Lastly, our website provides clear indications of system status through active/inactive menus and breadcrumb navigation.

4.2 Design Cycle

4.2.1 Hand-drawn Sketches

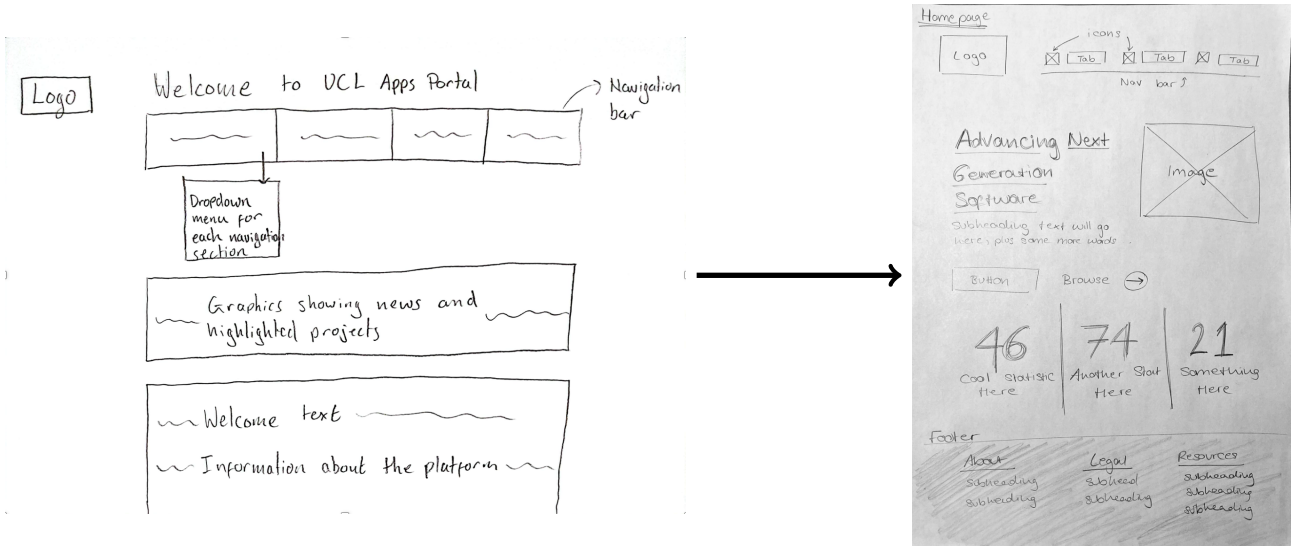

Figure 4.1: The first (left) and final iterations of the home page.

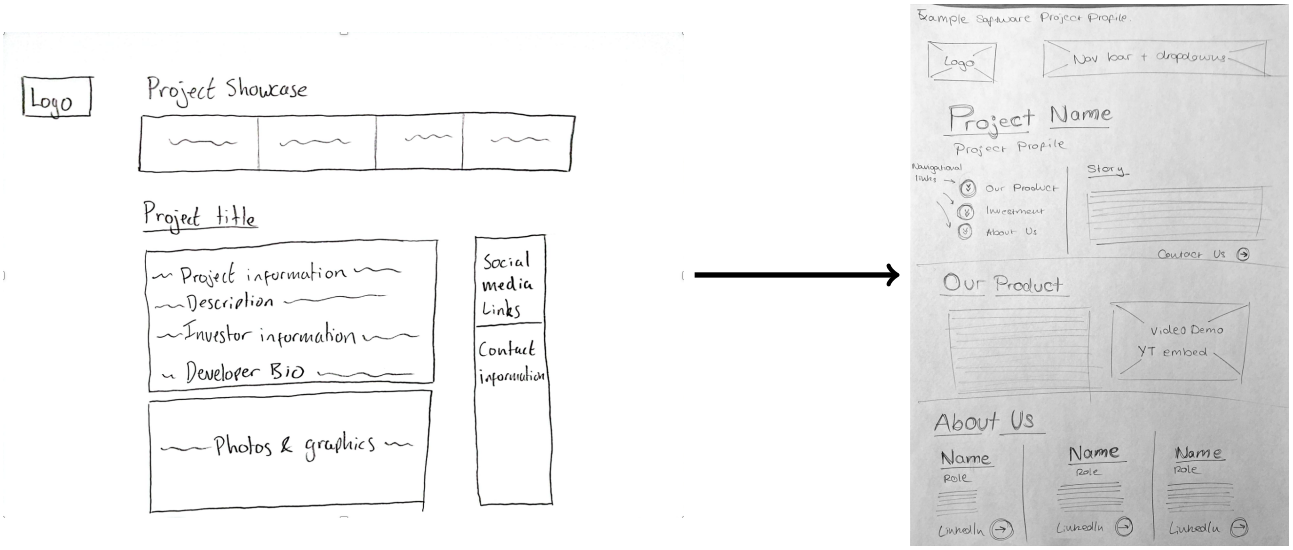

Figure 4.2: The first (left) and final iterations of the project profile page.

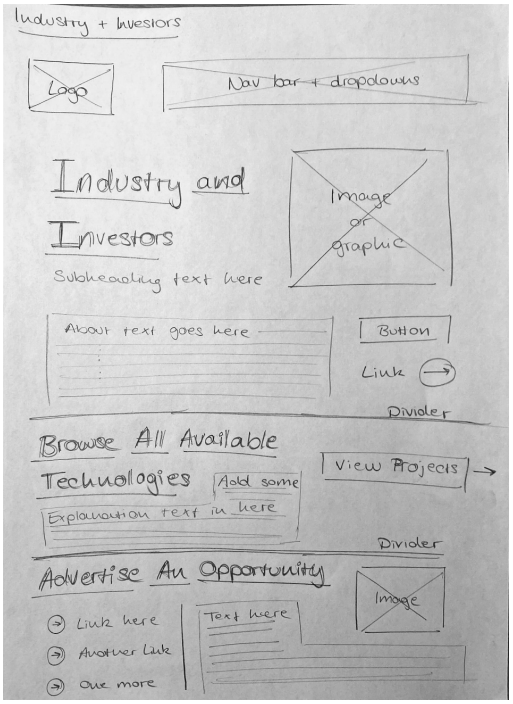

Figure 4.3: Final iteration sketch for the industry and investors dashboard.

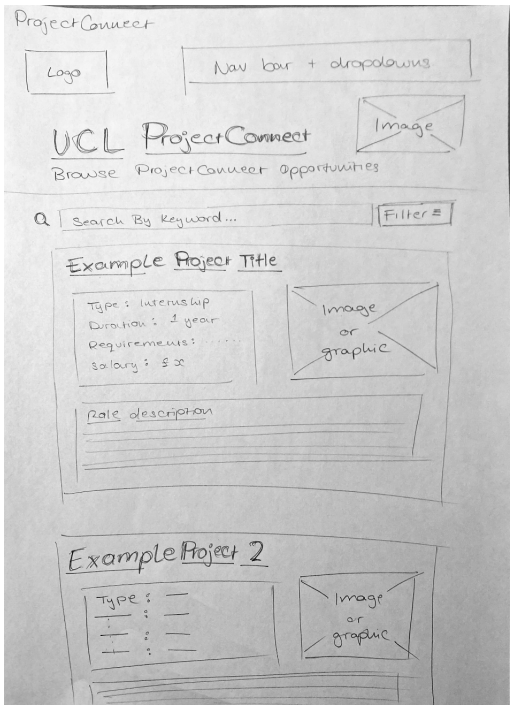

Figure 4.4: Final iteration sketch for project connect browse page.

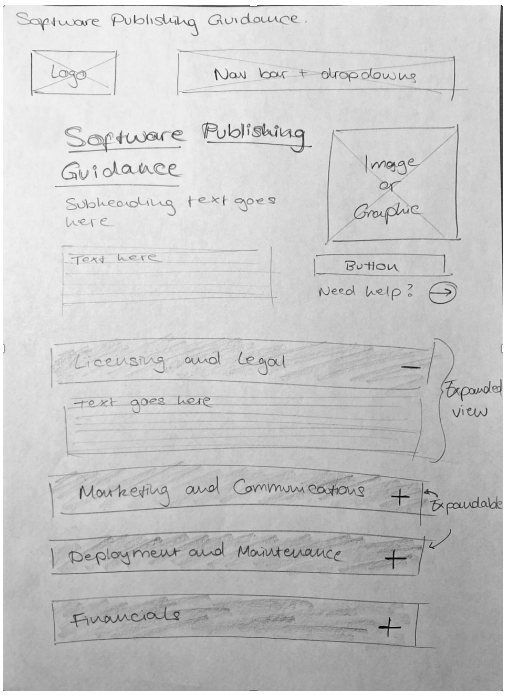

Figure 4.5: Final iteration sketch for the software publishing guidance section.

4.2.2 Wireframe

Continuing from the sketches, several wireframes were created, using Balsamiq, based off of the second iteration of the sketches but contained much more detail in order to create a skeleton design for the interactive prototype. Some of the key pages are shown below:

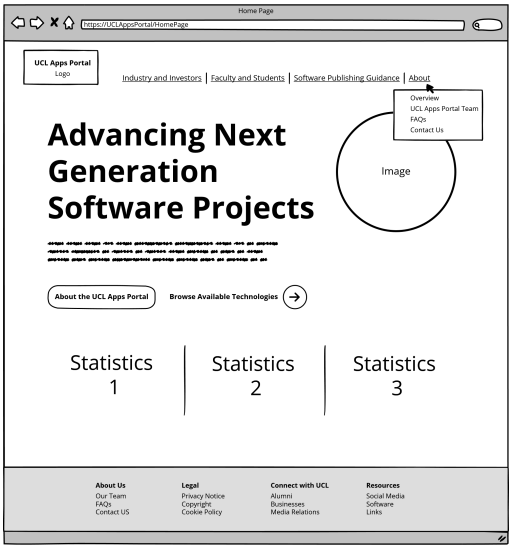

Figure 4.6: Wireframe design of the home page

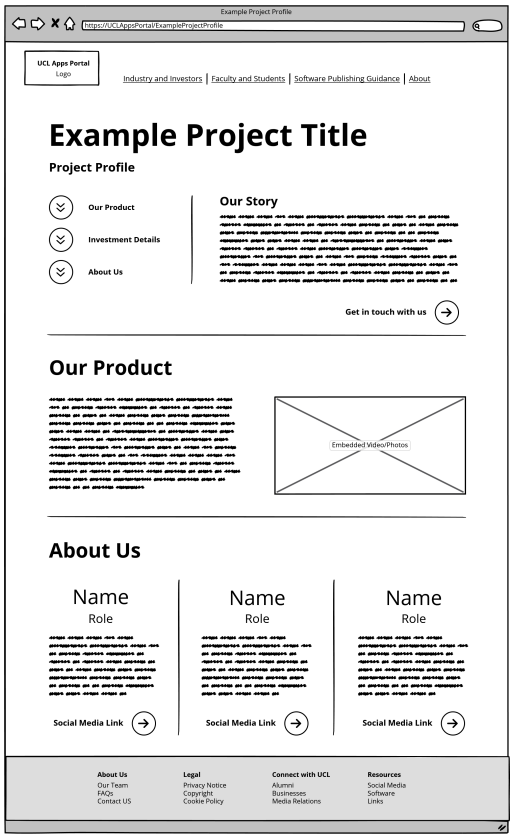

Figure 4.7: Wireframe design of a project profile

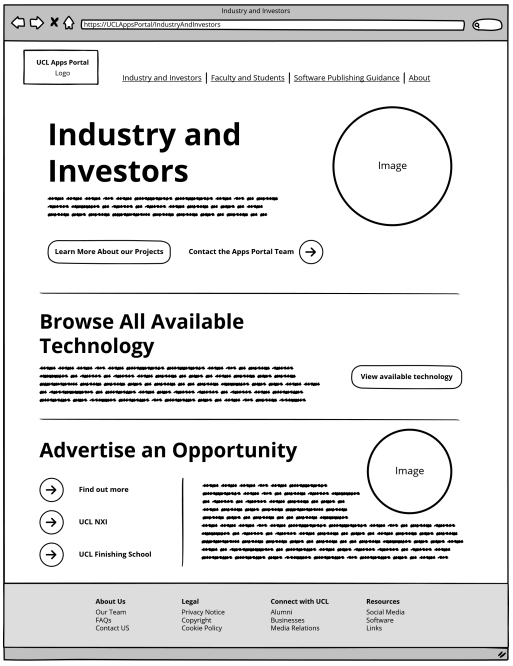

Figure 4.8: Wireframe design of the industry and investors dashboard

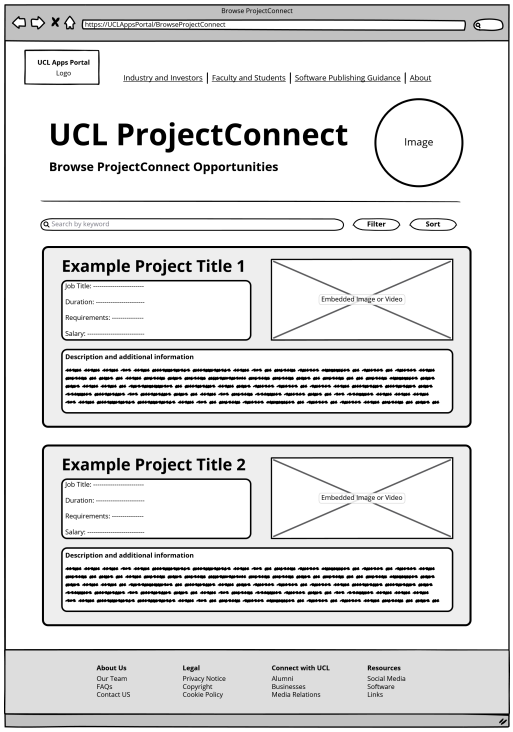

Figure 4.9: Wireframe design of the project connect listings page

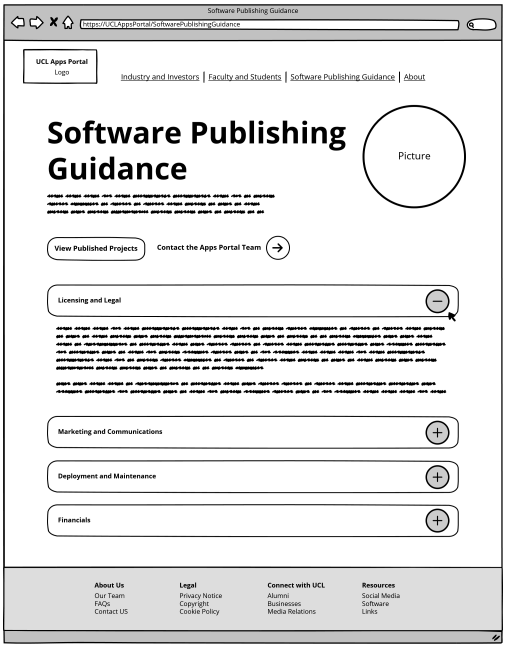

Figure 4.10: Wireframe design of the software publishing guidance section

4.2.3 Interactive Prototype

Following on from the wireframes, a high fidelity interactive prototype was created using Figma that included our final design decisions. The prototype can be accessed from this link: Figma Interactive Prototype.

Some screenshots of the key pages are shown below:

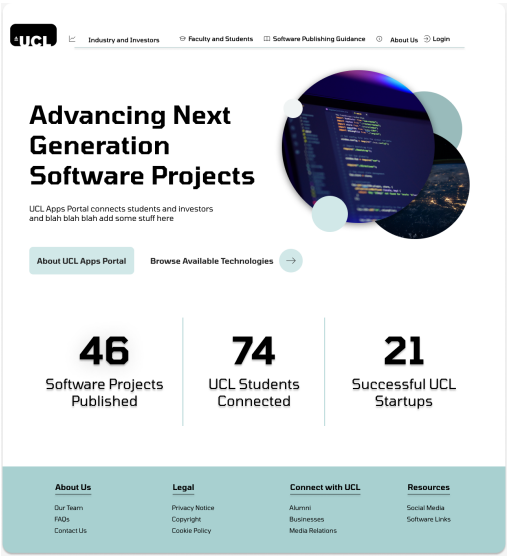

Figure 4.11: Figma design of the home page

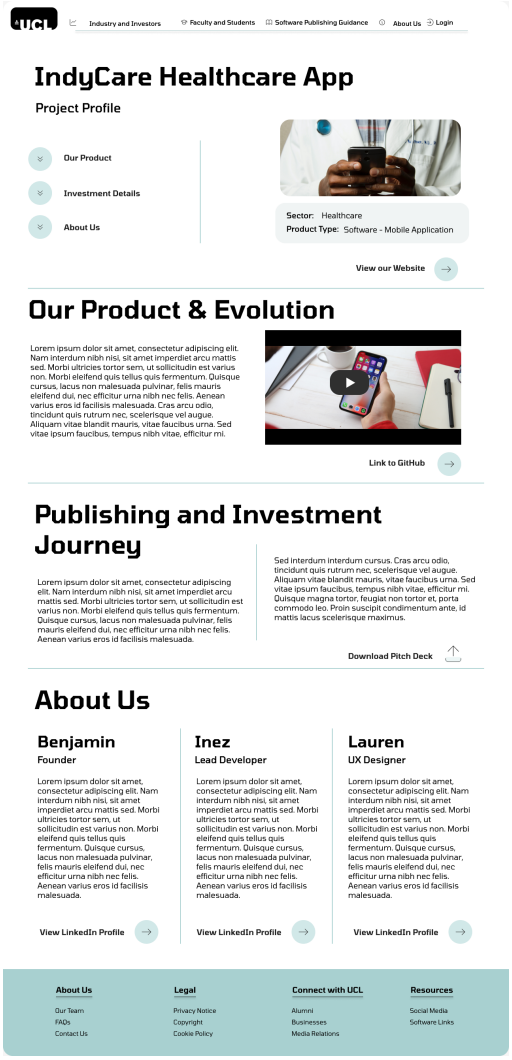

Figure 4.12: Figma design of a project profile

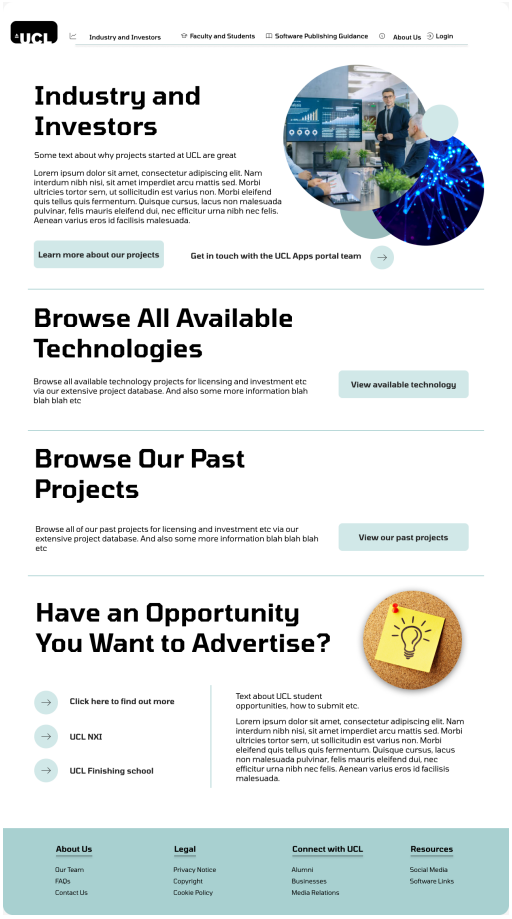

Figure 4.13: Figma design of the industry and investors homepage

Figure 4.14: Figma design of the project connect listings page

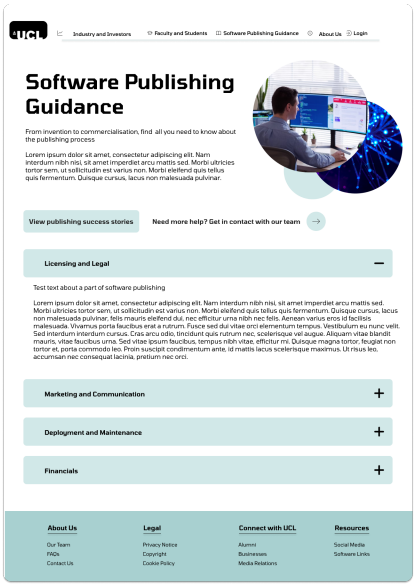

Figure 4.15: Figma design of the software publishing guidance section

5 System Design

5.1 System architecture

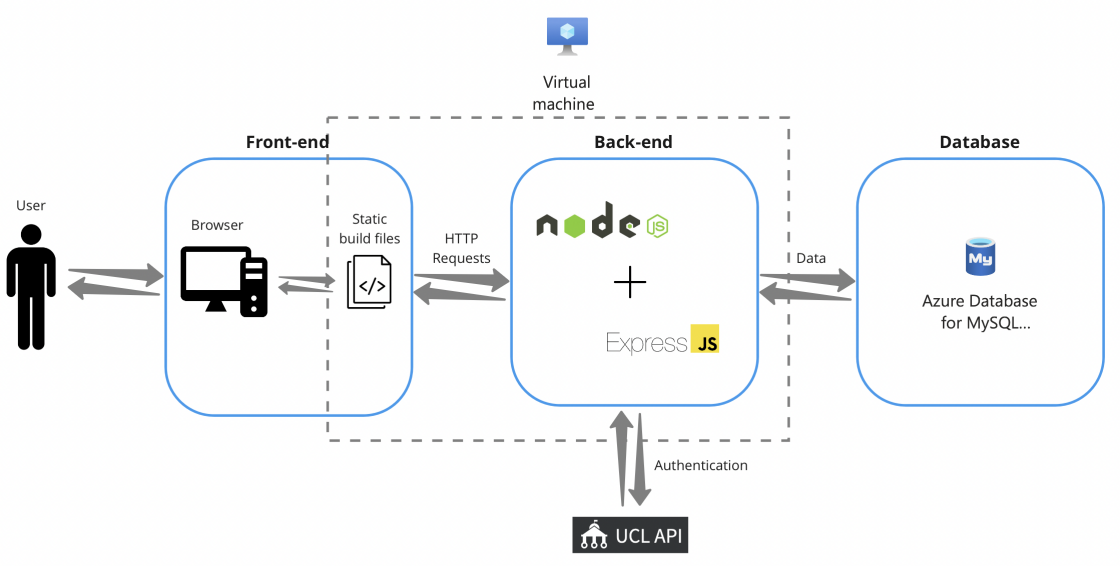

Figure 5.1: System architecture diagram.

- Back end: A `Node.js` app implementing API routes using `express`. The back-end server handles the application logic for the website. It also listens for incoming HTTP requests, as well as sending HTTP requests to the UCL API for authorisation purposes, and the Azure Database for data transmission. Additionally, the back-end serves the client with the static build files from the front-end React app.
- Database: A MySQL database hosted remotely on an Azure server. Contains data concerning the website's listings.
- UCL API: A third party API used to verify the identity of UCL members.
- Front-end: A React app complied into static files, which are then sent to the client and rendered in the browser.
- Virtual machine: A remote Linux environment running the back-end server.

5.2 Site Maps

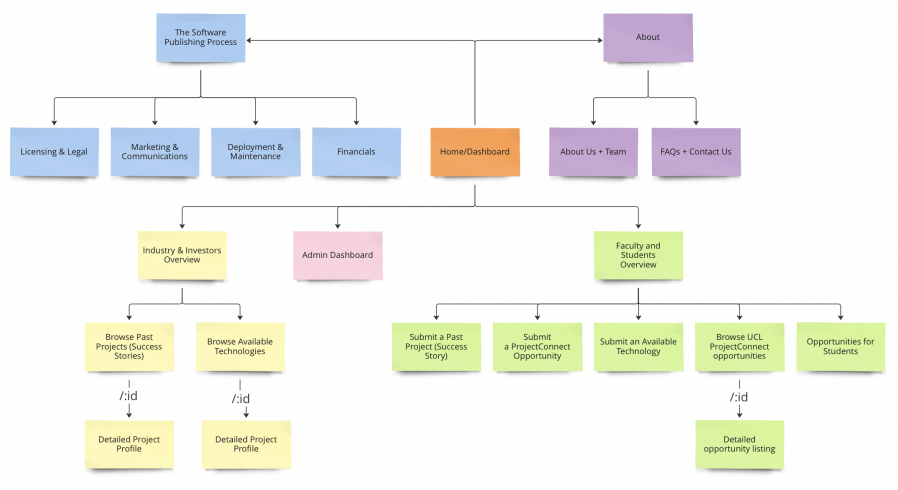

Figure 5.2: Site map for the Admin user

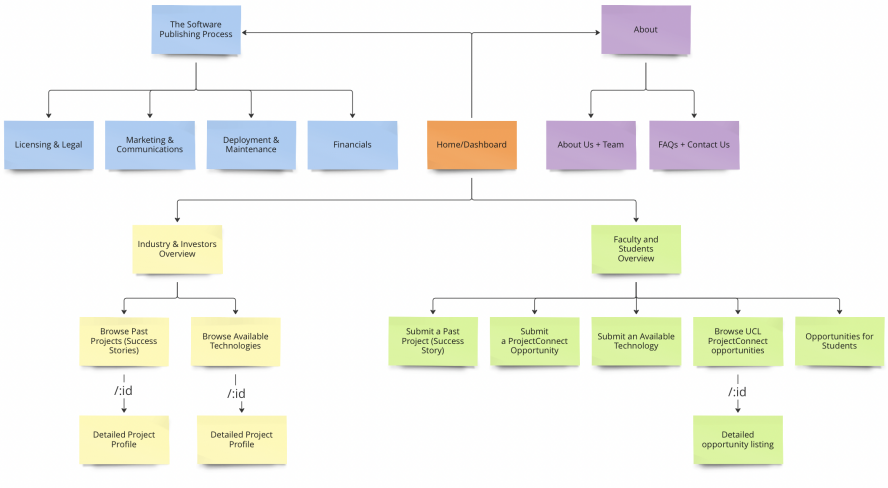

Figure 5.3: Site map for an authenticated UCL member.

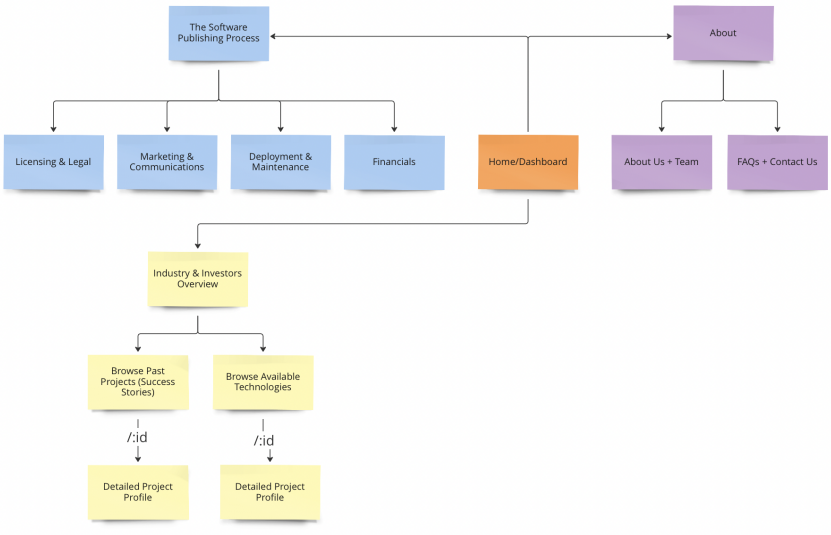

Figure 5.4: Public site map (i.e not logged in)

5.3 Database Design

The database was set up with a number of constraints. Each table has its own primary key, with many also linking to other tables using the foreign key constraint. Additionally, foreign keys referencing the listings table as a parent were set-up with the CASCADE ON DELETE constraint. This means that when a listing is deleted from the database, it can be done with a single query. Any orphaned child references are automatically deleted, instead of having to submit several queries to remove records from relevant tables one-by-one.

Wherever possible, the database was normalised. For example, instead of putting team members as a list in the pastprojects and availabletechnology tables, it was separated out into its own table, as a single field should only contain a single value. Furthermore, optional fields that could potentially be empty, such as a team member's linkedin URL, were separated into their own tables and linked by a foreign key constraint in an attempt to minimise the number of empty fields in the tables.

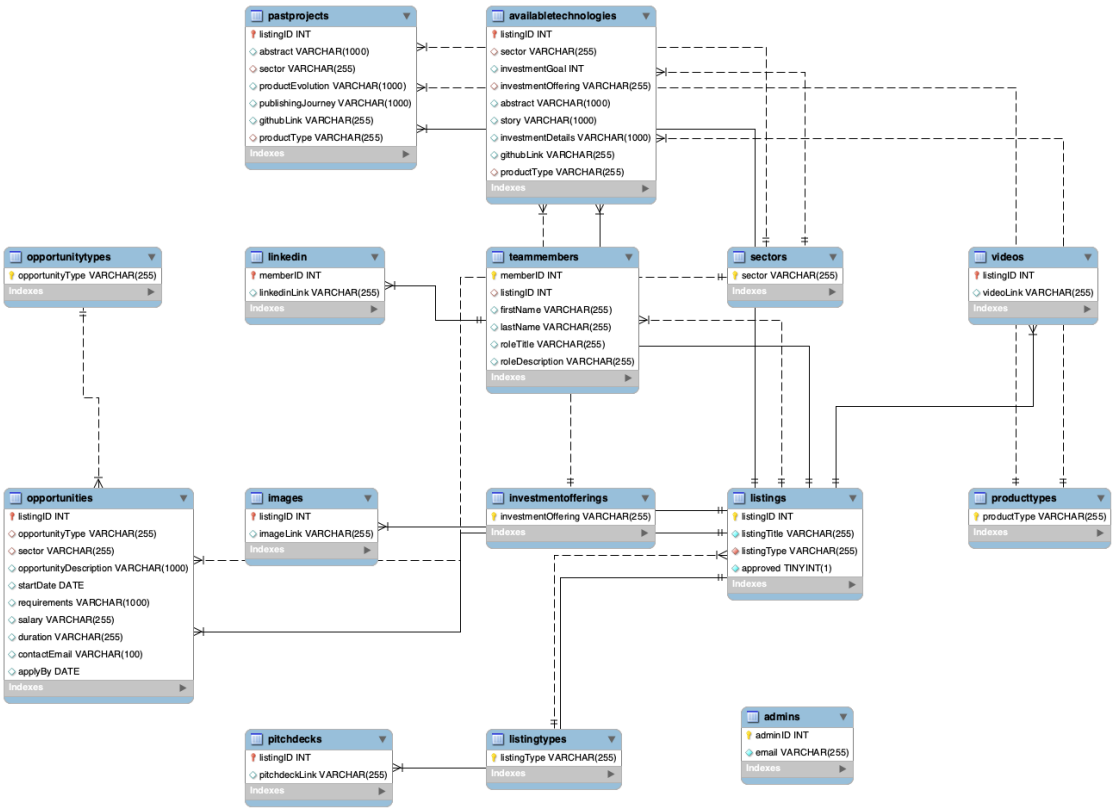

Figure 5.5: An ER Diagram for our Azure Database.

6 Implementation

Please describe how you implement the key features

6.1 Dynamic Front-end rendering

In this web app, one of the key implementation features of the front-end is the dynamic listing page for past projects. This feature provides an interactive and user-friendly way for users to browse and navigate through a list of past projects fetched from a MySQL database. Using the PastProjects listings page as an example, we will explore the libraries, frameworks, and techniques employed to create a seamless experience that includes authentication, authorization, data fetching, pagination, and navigation. The dynamic listings component fetches and displays a list of projects, with pagination, from the MySQL database so therefore this feature provides the best summary of the technologies used on the front-end of the website.

React: A JavaScript library used for building user interfaces. It is the foundation of the component-based architecture in this web app.

Material-UI: A popular React UI framework that provides a collection of pre-built components, such as Pagination and Box.

Axios: A library for making HTTP requests, utilized for fetching data from the server. React Router: A routing library for React applications that allows for easy navigation between different pages and components.

The authentication and authorization for the PastProjects component is handled by the server-side API. When the component fetches data from the API using Axios, the server checks if the user is authenticated. If not, the server sends a response indicating that the authentication has failed, and the component redirects the user to an "error" page.

The PastProjects component fetches the data from a server-side API using Axios. The server-side API is responsible for handling the database connection and querying the required data for past projects. The fetched data is then returned as a JSON object to the client-side, which is subsequently used to update the component's state and render the project listings.

The fetched project listings are displayed in a paginated way. The Pagination component from Material-UI is used to handle this feature. The logic for slicing the fetched data into pages is implemented using indices calculated based on the current page and the number of listings per page. The handlePageChange function updates the currentPage state when the user interacts with the Pagination component, ensuring that the correct listings are displayed as the user navigates through the pages.

The PastProjects component uses the fetched project listings to render the individual projects as a collection of Box components from Material-UI. Each project listing is wrapped in a Link element from React Router, enabling navigation to a detailed view of the project when clicked. Error handling and navigation are managed using the useNavigate hook from React Router, redirecting users to error or expired pages when necessary.

```
//Fetch listings
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    const response = await axios
      .get("http://localhost:5500/past-projects/browse")
      .then((response) => {
        if (response.data.auth === false) {
          navigate('/expired');
        } else {
          setListings(response.data);
        }
      })
      .catch((err) => {
        console.log(err);
        navigate('/error');
      });

    setLoading(false);
  };

  fetchData();
}, []);
```

Figure 6.1: A code snippet from the function to fetch the listing data from the database

```
<div>
  <Pagination
    count={totalPages}
    page={currentPage}
    onChange={handlePageChange}
  />
</div>
```

Figure 6.2: A code snippet from the pagination function

6.2 Responsive Design

We have implemented responsive design for our website to ensure a seamless user experience across various screen sizes, including phones, different types of laptops, large monitors, and landscape tablets. The following sections provide an in-depth explanation of the tools, frameworks, and methods used for the implementation.

Tools and Frameworks:

To achieve the responsive design, we have used the following tools and frameworks: CSS Media Queries: Media Queries are a CSS technique that allows us to apply different styles based on the device's screen size and resolution. We have used media queries to create breakpoints and adjust the layout, typography, and other design elements based on the target screen size.

Implementation Details: The implementation process involved the following steps:

Figure 6.3: A code snippet from the code to render the listings

a. Breakpoints: We have defined breakpoints for various screen sizes to accommodate different devices, including the ones in Figure 6.4.

b. Mobile Header: For better usability on mobile devices, we created a new header with fewer menu components to improve clarity and ease of navigation. The mobile header is implemented using CSS media queries and JavaScript

c. Responsive Pages: Currently, one-third of our website's pages are fully responsive, including Home, Industry and Investors, About Us, and FAQs. We have used CSS media queries, flexible grid layouts, and adaptive images to ensure that these pages adjust to the screen size and provide an optimal user experience.

While one-third of the pages are already responsive, the remaining pages still need to be optimized for mobile devices.

```
const isPhone = useMediaQuery({ minWidth: 0, maxWidth: 450 });
const isLaptop = useMediaQuery({ minWidth: 451, maxWidth: 1823 });
const isLargeMonitor = useMediaQuery({ minWidth: 1824, maxWidth: 2359 });
const isLandscapeTablet = useMediaQuery({ minWidth: 2360 });

const getClassName = (baseClassName) => {
  if (isLargeMonitor) return `${baseClassName}--large-monitor`;
  if (isLaptop) return `${baseClassName}--laptop`;
  if (isLandscapeTablet) return `${baseClassName}--landscape-tablet`;
  if (isPhone) return `${baseClassName}--phone`;
  return baseClassName;
};
```

Figure 6.4: A code snippet from media queries

6.3 Database Connection

To connect to the Azure database, the Node.js app uses the `mysql2` node module. A pool of database connections is created, as shown in Figure 6.5, to allow the connections to be

reused. The pool is also "promisified", allowing the result of DB operations to be returned as a promise. Database credentials are accessed using the `.env` file, shown in the Deployment Manual (see Appendix).

```
export const pool = mysql.createPool({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    port: process.env.DB_PORT,
    ssl:{ca:fs.readFileSync(process.env.DB_SSL_CERT_PATH)}
    }).promise();
```

Figure 6.5: A code snippet from DatabaseFunctions.js showing the pool connection object.

Figure 6.6 shows an example of a function querying the database. This particular function gets the sectors from the sectors table to display in dropdown menus in the listing submission forms. It is an asynchronous function, using the "async-await" syntax, meaning it will wait for the promise to resolve before returning the result.

```
export async function getSectors () {
    const [result] = await pool.query('SELECT * FROM sectors;');
    return result
}
```

Figure 6.6: A code snippet from DatabaseFunctions.js showing an example function querying the database.

6.4 API Routes

The API routes were created using the `express` module for Node.js. A variety of GET, POST, PATCH, and DELETE routes are defined. The routes were also organised into routers to aid code readability and maintainability. Routers are instantiated in their own files, and then imported and mounted in `server.js`, as seen in figure 6.7.

```
// import routers
import { availableTechRouter } from './routes/availableTech.js';
import { pastProjectsRouter } from './routes/pastProjects.js';
import { projectConnectRouter } from './routes/projectConnect.js';
import { formOptionsRouter } from './routes/formDropdowns.js';
import { oauth } from './routes/auth.js';
import { adminRouter } from './routes/admin.js';

// mount routers
app.use('/available-tech', availableTechRouter);
app.use('/past-projects', pastProjectsRouter);
app.use('/project-connect', projectConnectRouter);
app.use('/form-options', formOptionsRouter);
app.use('/oauth', oauth)
app.use('/admin', adminRouter);
```

Figure 6.7: A code snippet from server.js showing the routers being imported from their relevant files and mounted to their desired URLS.

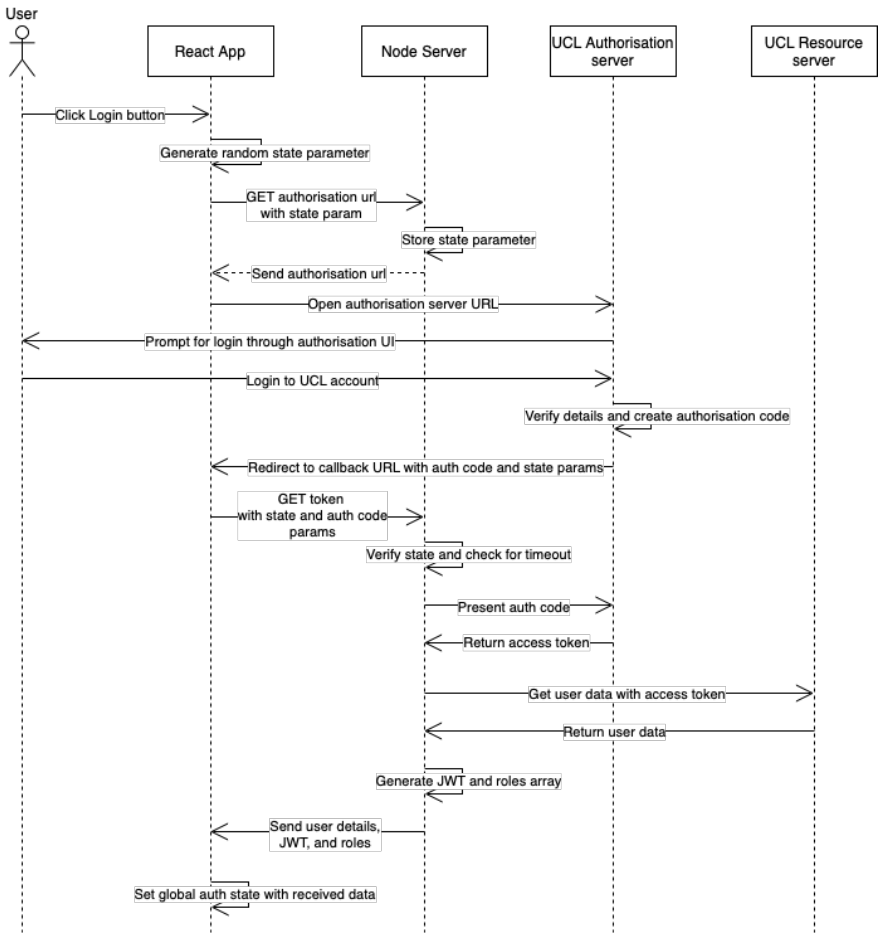

Figure 6.8: A sequence diagram representing a successful OAuth login flow for UCL staff and students.

6.5 Authentication

Figure 6.8 shows the OAuth flow for a successful login functionality to the web app. The flow is initiated on the client side when the user clicks the *Login* button. The react app will generate a random string for the *state* parameter, required by the UCL API to help prevent against Cross-Site Request Forgery (CSRF) attacks. The state is then sent as a query parameter in a GET request to the back-end API, where it is stored on the server side. The server generates an authorisation URL containing both the state and the client ID of the app (assigned to the web app in the UCL API developer dashboard).

The front-end then redirects the browser to the authorisation URL, at which point the user is prompted to enter their UCL login details. Once these details are verified by the UCL authorisation server, an authorisation code is generated and sent along with the original state parameter, in the response to the front-end request. The code and state are then extracted by the React app and sent as query parameters in another GET request to a different API end point on the node server.

The server then verifies that the state sent in this request is the same as the state that it stored during the initial request for the authorisation URL. This ensures that requests

have not been intercepted by other sites. The server will also check to see how long it has been since the initial state was generated; If more than 5 minutes have passed, the authorisation will have timed-out and failed. The server then sends a GET request to the UCL authorisation server presenting the authorisation code, and receives an access token in response. This access token is then used to access user data (a protected resource) through another request.

Upon receipt of the user data, the server will generate a JSON web token, storing the user's email in the payload. The user is also assigned roles in the form of an array containing codes to represent the different roles. After successful login, all users are assigned a roles array of [101], reflecting that they have the standard user permissions. If a users email matches the admin email stored in the .env file, the user will be assigned a roles array of [101, 102] to reflect that they have both standard and admin permissions applied.

Finally, the user data, JWT, and roles array are sent in the response to the front-end. For security reasons, the JWT is sent in a HTTP only cookie. The global auth state (created using the Context API) is then set to an object containing received data.

6.6 Authorisation

6.6.1 Global Auth State

As mentioned above, an authorisation context is created in the `/client/src/components/context/AuthProvider.js` file, using the context API. A custom hook named `useAuth` is then created in the `/client/src/components/hooks/useAuth.js` file applying the `AuthContext` to our instance of react. The custom hook can then be imported directly into files, instead of having to apply the `AuthContext` every time. When called, useAuth can be destructured into a `auth` variable and a `setAuth` function, similar to `useState` hook.

6.6.2 Role-based Authentication

Another component, `RequireAuth` is created in `/client/src/components/RequireAuth.js` to allow for role-based authentication of routes using React router v6. The return value of the component can be seen in figure 6.9. `Outlet` here represents any child component of `RequireAuth`, which in this case will be the protected routes around which the component is wrapped. The component will search through the roles array in the global auth state to see if they match the allowed roles (passed as a prop). If the user's role is found in the list of allowed roles then the outlet component will be returned else, if the user is logged in they will be redirect to `/Forbidden`, since they don't have the required permissions. If the user is not logged in they will be redirected to `/Unauthorised`, and prompted to login.

The `RequireAuth` component is then wrapped around the required protected routes in `App.js` and passed the allowed roles array as a prop, demonstrated in Figure 6.10.

```
auth?.roles?.find((role) => allowedRoles.includes(role)) ? (
  <Outlet />
) : auth?.user ? (
  <Navigate to="/Forbidden" state={{ from: location }} replace />
) : (
  <Navigate to="/Unauthorised" state={{ from: location }} replace />
));
```

Figure 6.9: A code snippet from RequireAuth.js showing the return value of the component.

```
{/* protected Routes - ucl staff/student*/}
<Route element={<RequireAuth allowedRoles={[101]} />}>
  <Route path="/FacultyAndStudents/Overview" element={<FacultyAndStudents />}/>
  <Route path="/FacultyAndStudents/ProjectConnect" element={<ProjectConnect />} />
  <Route path="/FacultyAndStudents/OpportunitiesForStudents" element={<OpportunitiesForStudents />}/>

  <Route path="/FacultyAndStudents/SubmitAvailableTechnology" element={<SubmitAvailableTechnology />}/>
  <Route path="/FacultyAndStudents/SubmitPastProject" element={<SubmitPastProject />} />
  <Route path="/FacultyAndStudents/SubmitOpportunity" element={<SubmitOpportunity />} />
  <Route path="/FormSubmitted" element={<FormSubmitted/>} />
</Route>

{/* protected Routes - admin only*/}
<Route element={<RequireAuth allowedRoles={[102]} />}>
  <Route path="/Admin" element={<AdminDashboard />} />
</Route>
```

Figure 6.10: A code snippet from App.js showing the use of the RequireAuth component for role-based authentication of routes.

6.6.3 Protected API routes

Protected API routes, such as the admin routes that approve or delete listings, have a custom middleware function applied. For admin routes, this will be the verifyJWT function shown in figure 7.2. The middleware ensures that the user can only access the route if their JWT (sent as a HTTP only cookie in the request header) is verified, and the email in the payload matches the admin email. This is to protect the server, in case its routes are accessed directly through the URL.

7 Testing

7.1 Testing Strategy

The testing for this project was done continuously during development and utilised tools such as Chrome developer tools for the front-end, and Postman for back-end API testing.

After the majority of the features were complete, unit tests were written for small, relatively self-contained functions such as the database functions and the JSON web token (JWT) verification functions. Tests aimed to cover a range of scenarios and outcomes for each function. Integration tests were also written to test the responses to the API routes.

Responsive design testing was also undertaken to assess the app's adapatbility to different screen sizes, as well as compatibility testing for different browsers, and stress testing. Finally potential users were surveyed for their feedback with regards to user acceptance testing.

7.2 Heuristic Testing

During the HCI report, we conducted an Analytic Heuristic Evaluation with 3 experienced developers and found 13 problems that could be improved within our Figma prototype. We have chosen to implement the most severe ones: Add link to Github for projects to better match the requirements and breadcrumb to show visibility of system status.

7.3 Unit Testing

The unit tests for this project were written using `Mocha`, a popular JavaScript testing framework for `Node.js`. The unit tests also made use of both the `assert.js` and `should.js` modules to compare the expected and actual outcomes of the selected functions.

Unit tests were written for a selection of database functions, aiming to cover the majority of the database's functionality (i.e SELECT, INSERT, UPDATE, DELETE), as well as the functions verifying the JSON web tokens for both UCL users and the site admin. A full list of unit tests can be seen in Figure **??**.

Figure **??** shows that of the 13 unit tests, 12 passed and 1 failed. The unit test that failed can be seen in figure 7.1, which shows the function verifying the JWT of a standard user, in the case that no JWT cookie is sent in the request header. The expected response is that the authorisation will fail, returning an object with a value of `auth = false` and a `message = "No token found in header"`. In this instance, the function was attempting to trying to search for a key in an object that was undefined, throwing an error. To remedy this, conditional chaining was implemented and, upon further testing, the unit test passed. The amendment can be seen in figure 7.2.

```
describe("Verify JWT functions", function () {
  describe("Verify JWT for standard user", function () {
    it("No token, auth should equal false", function () {
      // create mock req, res and next objects
      const req = {};
      const res = {
        json: function (obj) {
          res.data = obj;
        },
      };
      const next = function () {};

      verifyJWT(req, res, next);

      assert.deepStrictEqual(res.data.auth, false);
      assert.deepStrictEqual(res.data.msg, "No token found in header");
    });
```

Figure 7.1: An example unit test.

```
export const verifyJWT = (req, res, next) => {

  // before
  // const jwt = req.cookies.jwt;

  // after
  const jwt = req?.cookies?.jwt;

  if (!jwt) {
    res.json({ auth: false, msg: "No token found in header" });
  } else {
    verify(jwt, process.env.JWT_SECRET, (err, decoded) => {
      if (err) {
        // console.log(err);
        res.json({ auth: false, msg: "Failed to authenticate" });
      } else {
        next();
      }
    });
  }
};
```

Figure 7.2: The correction made to the function failing its unit test.

7.4 Integration Testing

The integration tests were conducted in a similar fashion to the unit testing, accept that they tested entire routes rather than individual units. `Mocha` was again used as the testing framework, but this time with the `chai.js` and `chai-http.js` modules. Ideally, the integration testing would have covered one of each type of route used: GET, POST, DELETE, PATCH, as well as routes with variable parameters such as GET /browse/:id. However, due to time constraints only one GET and one POST route was tested.

All integration tests passed, the results of which can be seen in figure 7.3, as well as an example integration test in figure 7.4 for the GET `/project-connect/browse` route.

```
Route testing
  /GET route
    ✔ should return all approved project opportunities (110ms)
  /POST a past project
    ✔ should submit a new project form (99ms)
    ✔ should return 400 error if required fields are missing (81ms)


3 passing (294ms)
```

Figure 7.3: Integration testing results.

```
describe("/GET route", () => {
  const adminEmail = "testemail@ucl.ac.uk";
  const jwt = sign({ email: adminEmail }, process.env.JWT_SECRET, {
    expiresIn: 60 * 60, // 60 mins
  });

  it("should return all approved project opportunities", (done) => {
    chai
      .request(app)
      .get("/project-connect/browse")
      .set("Cookie", `jwt=${jwt}`)
      .end((err, res) => {
        expect(res).to.have.status(200);
        expect(res).to.be.json;
        expect(res.body).to.be.an("array");
        expect(res.body.length).to.be.above(0);
        // check if each project opportunity has required properties
        res.body.forEach((project) => {
          expect(project).to.have.property("listingID");
          expect(project).to.have.property("listingTitle");
          expect(project).to.have.property("opportunityType");
          expect(project).to.have.property("sector");
          expect(project).to.have.property("opportunityDescription");
          expect(project).to.have.property("startDate");
          expect(project).to.have.property("salary");
          expect(project).to.have.property("duration");
          expect(project).to.have.property("contactEmail");
        });
        done();
      });
```

Figure 7.4: Example integration test.

7.5 Responsive Design Testing

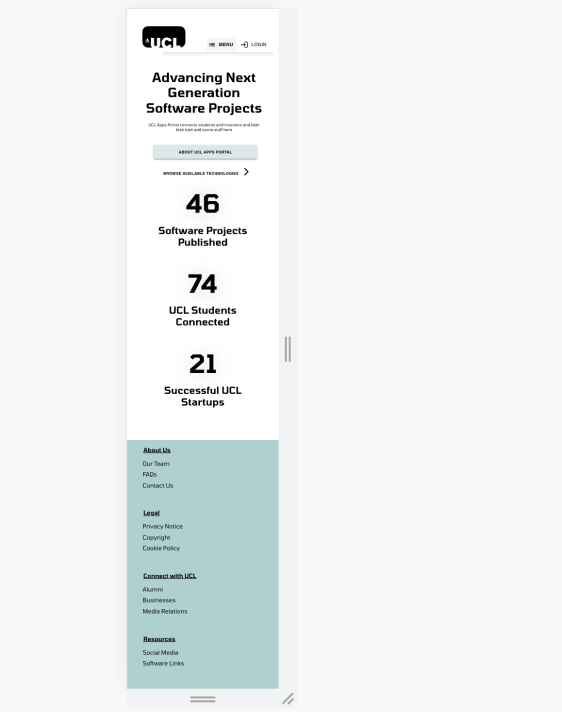

Figure 7.5: Iphone 12 Pro View

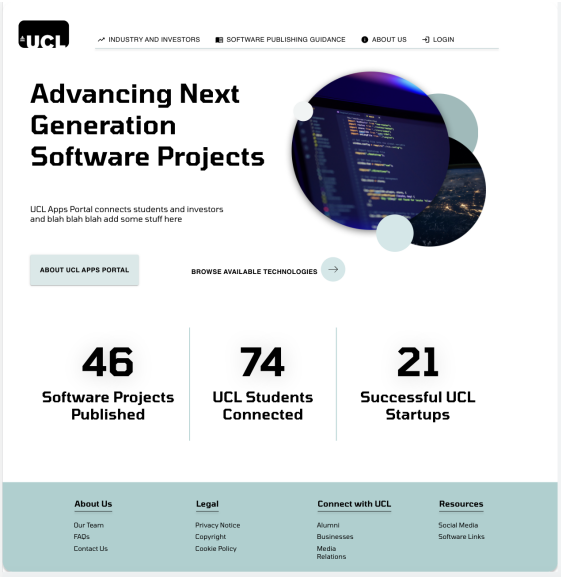

Figure 7.6: Laptop Width 1366 View

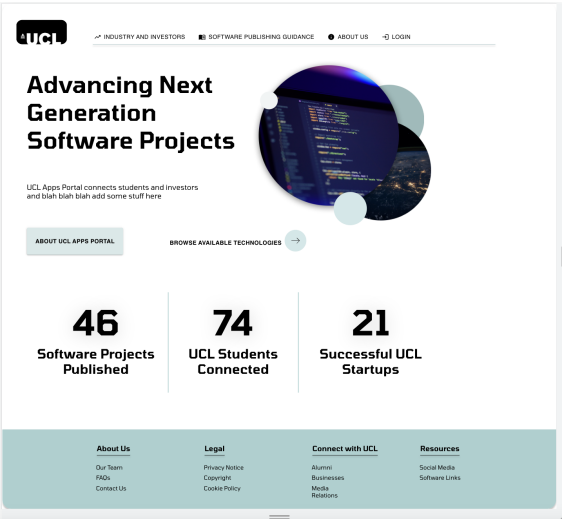

Figure 7.7: Laptop Width 1536 View

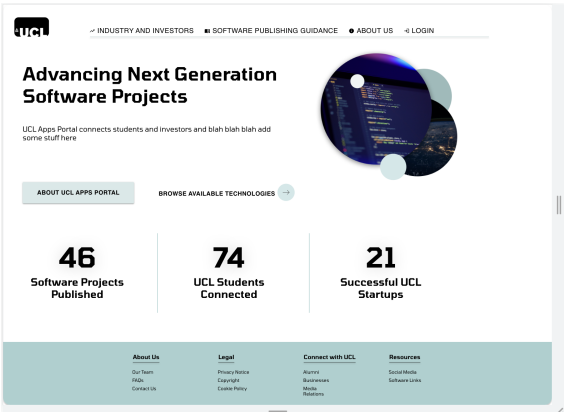

Figure 7.8: Large Monitor Width 1920 View

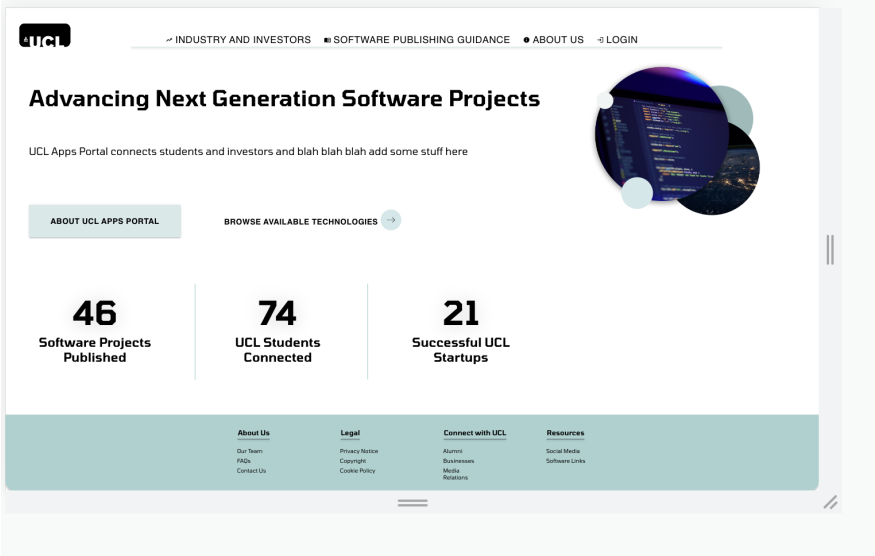

Figure 7.9: Landscape Tablet Width 2360 View

The purpose of responsive design testing is to ensure that our website is compatible with a wide range of devices, including mobile phones. This compatibility is a crucial requirement for our project. We have conducted tests using Google Chrome's DevTools to emulate various screen sizes and resolutions for the key pages. The following sections detail the results of our testing, including screenshots and future work.

Testing Methodology: We used Google Chrome's DevTools to emulate different screen sizes and resolutions for the key pages of our website. This allowed us to evaluate the responsiveness and compatibility of each page with various devices.

Test Results: The test results, presented as screenshots below, show the website's performance on different devices. Each device is captioned to indicate the specific screen size and resolution tested.

a. Home Page: The home page has been confirmed to be fully compatible with all tested devices and screen sizes. The responsive design adapts smoothly to the screen size, providing an optimal user experience.

b. Remaining Pages: While a majority of the pages display adequate responsiveness, a few pages still lack mobile functionality. These pages have been documented and will be addressed in our future work.

7.6 Compatibility Testing

Compatibility testing is an essential aspect of web development, it ensures that a web application works correctly and provides a consistent user experience across different browsers, devices, and operating systems. The primary goal of compatibility testing is to identify

and fix potential issues that might arise due to varying browser implementations, rendering engines, and platform-specific quirks. This process helps us maintain a high level of user satisfaction.

We have thoroughly tested our web app and confirmed its compatibility with three major browsers on a desktop environment: Google Chrome, Mozilla Firefox, and Apple Safari to ensure that the MoSCoW requirements are met during use of all of these browsers. These browsers represent a significant portion of the global browser market, which means that our web app will be accessible and functional for the vast majority of users.

When using mobile browsers, several pages of our web app are responsive and adapt well to the smaller screen sizes. However it must be noted that some pages remain unresponsive, which might affect the user experience on mobile devices. In such cases, it is necessary to use the 'desktop mode' on the mobile browser to view the full pages where responsive design has not been implemented.

7.7 Stress Testing

Had there been more time, we would have performed stress testing on the website to evaluate its performance, stability, and reliability under heavy load conditions. This would have started by outlining the testing environment, scenarios, tools, and metrics to be recorded.

Appropriate testing tools, such as Gatling, would have been selected for simulating load scenarios and recording performance metrics. We would have prepared a testing environment resembling the production setup and designed test scenarios to simulate real-world user behaviour.

The stress test would have been executed by gradually increasing the load while monitoring the website's performance, error rates, and resource usage. Subsequently, we would have analysed the test results to identify bottlenecks and performance issues in the React components, Node.js server, or MySQL database.

After identifying the issues, we would have optimised the website through code improvements, infrastructure enhancements, or configuration adjustments. The website would then have been retested to validate the effectiveness of these optimisations.

7.8 User Acceptance Testing

In the final phase of our testing process, we conducted user acceptance tests. These tests enabled us to identify additional bugs and website functionalities that might be challenging for new users, given that the test participants were not familiar with the website.

We began by defining a list of tasks, as displayed in the first table. These tasks represent the essential functional requirements we had previously determined. We then created a table

in Excel to be filled out by each tester. They were asked to indicate "Pass" or "Fail" for each task and provide relevant comments. We selected UCL students as our target audience since they represent a typical user group for the website. As the website was not yet live, sharing a link with the testers was not feasible. Instead, we approached random students at the UCL Student Centre and asked them to interact with the website on our laptop and complete the Excel file. We observed three students in total.

The first table presents the completion rate for each task. The second table compiles the comments provided by the testers, which gave us insights into why some tests failed, and consequently, offered us valuable information for future improvements. Our observations revealed that users prefer buttons in a menu section to lead exclusively to that section, users have difficulty locating the forms, and clicking on a blank area sometimes redirects users to pages unexpectedly, causing confusion.

Task	Pass/Fail
Go to About Section	3/3
View Software Publishing Proccess	2/3
Browse Profiles of past UCL projects as Success Stories	3/3
View Detailed Project	2/3
Browse Profiles of current UCL projects seeking investment (as available tech)	3/3
View Detailed Project	3/3
Login	3/3
Browse Project Connect Opportunities for students	2/3
Advertise/Submit your past project - to be published on the website	2/3
Create profile on your projects/technologies seeking investment	3/3
Post Opportunities for others to join your project	3/3

Table 7.1: Task evaluation table

Comments
When you click the about button in header you should automatically be redirected to the "about" page
"view publishing success stories" button in Publishing guidance is not clear (takes you to a different section of the menu)
Weird that the font size is different for header when login
Apply by button should be above the start date (the apply by button can be confused with end date) in connect form
Success stories page - if you click outside the projects - it redirects you as in when you click on the project itself
After I submit I want to be able to see and edit my submission
Very easy to find about
Can't find publishing guidance
Don't realise I can click on project for detailed view
Difficult to submit past project
Apply by date needs to be put before start date
Cant find browse project connect
Cant find submit past project
Very well done. Easy and accessible website. Very enjoyable user experience.

Table 7.2: User comments table

We are aware that partner feedback is a requirement for user acceptance testing. Unfortunately, our client is now on annual leave until 24th April, meaning we won't be able to get feedback from him until after the report is due.

Chapter 7 Testing

- Testing Strategy
- Unit and Integration testing
- Responsive design testing
  - Test for at least 3 screen sizes
  - Typical screen sizes
    - Large monitor: width 2560px
    - Laptop: width 1920px or 1280px
    - Landscape tablet: width 2360px
    - Portrait tablet: width 1640px
    - Portrait 6.1-inch phone (e.g. iPhone 14): width 2556px
    - Portrait 6.7-inch phone (e.g. iPhone 14 plus): width 2796px
  - Ask your partner the screen sizes they use most
- Compatibility testing
  - Test on at least 3 Internet browsers (e.g. Chrome, Safari, Firefox) for web applications
  - Test on at least 2 iOS devices and 2 Android devices for cross-platform mobile applications
- Stress testing
- User acceptance testing
  - Define a list of test tasks
  - At least 3 testers. Testers can be students from other teams.
  - Feedback from testers
  - Feedback from partners
- For each test, you should explain why you do this test, which test tool you use, how you conduct the test, what results you get, and your analysis or conclusion to the results.

8 Conclusion and Future Work

Add a user portal so that users are able to edit their forms/submissions Add a button so that admins can provide comments to a pending listing (rather than just approve or reject)

Chapter 8 Conclusion and Future Work

- Summary of achievements
  - A table to list the MoSCoW requirements, the completed states, and contributors
  - A list of known bugs
  - Individual contribution table
- A critical evaluation of the project
  - User interface design and user experience
  - Functionality
  - Stability
  - Efficiency
  - Compatibility
  - Maintainability
  - Project management
- Future work
  - How could the project be extended if you had another three months

8.1 Summary of Achievements

ID	Requirements	Priority	State	Contributors
FR1	The Apps Portal shall display **information about the software publishing process** including licensing and legal, marketing and communications, deployment and maintenance, and financials.	Must	Done	Arjun

Continued on next page

FR2	The Apps Portal shall allow UCL staff/students to **create profiles on previously published software projects**, including details about product evolution, publishing and investment journey etc.	Must	Done	Arjun, Isobel
FR3	The Apps Portal shall allow UCL staff/students to **create profiles on software projects seeking investment**, including details on business use cases and plans, financials, required funds, raised funds etc.	Must	Done	Arjun, Isobel
FR4	The Apps Portal shall allow users to **browse profiles of past UCL staff/student software projects** as "success stories"	Must	Done	Arjun, Isobel
FR5	The Apps Portal shall allow users to **browse profiles of current UCL staff/student software projects seeking investment**.	Must	Done	Arjun, Isobel
FR6	The Apps Portal shall display information about **opportunities for UCL students** related to software development and testing, including **links to the relevant UCL NXI and UCL Finshing School webpages**.	Must	Done	Tania, Isobel
FR7	The Apps Portal shall allow UCL staff/students to post advertisements of opportunities related to their software project, available to other UCL staff/students (referred to as ProjectConnect opportunities).	Must	Done	Arjun, Isobel
FR8	The Apps Portal shall allow UCL staff/students to **browse advertisements related to UCL software projects**, posted by other UCL students/staff (ProjectConnect opportunities).	Must	Done	Arjun, Isobel

Continued on next page

FR9	The Apps Portal shall provide **links to the relavant UCL NXI and UCL Finishing School webpages to allow external investors to submit internship opportunities.**	Must	TBD	
FR10	The Apps Portal shall restrict functionality only available to UCL members through **Single Sign On (SSO**	Must	Done	Isobel, Tania
FR11	TThe Apps Portal shall have a **single site admin.**	Must	Done	Isobel, Arjun
FR12	The Apps Portal shall require the **site admin to approve all posts** (including software project profiles and opportunities) before they are published to the website	Must	Done	Arjun, Isobel
FR13	The Apps Portal shall be able to **embed and play YouTube videos** from within project profiles.	Should	Done	Arjun, Tania
FR14	The Apps Portal shall have an **"About" section** including information about the UCL Apps Portal team, their contact details and site FAQs.	Should	Done	Tania
FR15	The Apps Portal shall implement **unbiased and inclusive approaches to recruitment** wherever possible.	Could	Done	Isobel
FR16	The Apps Portal shall allow **documents** (including Word and PowerPoint documents) **to be embedded** in project profiles	Could	Done	Tania, Isobel
FR17	The Apps Portal shall allow users to **filter software projects** by categories including date posted, area of research etc.	Could	TBD	
NFR1	The Apps Portal shall be **scalable**, allowing it to easily incorporate new webpages and/or features	Must	Done	All

Continued on next page

NFR2	The Apps Portal shall use a **MySQL database**	Must	Done	Isobel
NFR3	The Apps Portal shall have a minimalist design so it is **easy to navigate and accesible** to as many users as possible, including those with low technical proficiency and/or visual impairments	Must	Done	All
NFR4	The Apps Portal shall be **easy to maintain**, such that any system faults can be quickly identified and fixed	Must	Done	All
NFR5	The Apps Portal shall be as **efficient** as possible, with minimal reponse times for users	Must	Done	All
NFR6	The Apps Portal shall be reliable and available for the duration of UCL's official operating hours (8am - 7pm Monday - Friday) as a minimum, once made public	Must	TBD	
NFR7	The Apps Portal shall be a fully responsive web app, compatible with a variety of browsers on **both desktop and mobile devices**	Must	Partially Done	Tania
Key Functionalities (must have and should have)		90% completed		
Optional Functionalities (could have)		75% completed		

Table 8.1: Summary Achievements

8.2 List of known bugs

Bug	Description	Priority
Footer for dynamic pages	Footer renders in the middle of the screen	High
Software publishing stuck accordion	When a software publishing guidance accordion is opened via the NavBar it cannot be closed when clicked on	High
Admin dashboard issue	When a listing is unapproved, the admin cannot view the detailed listing page when a listing is clicked	High
UCL API login bug	The user is only prompted for their login the first time. Subsequent times, the UI prompt is not displayed to users, meaning you couldn't logout and login to another account. You can bypass this issue by clearing the browser history for the last hour and would will be prompted to authorise the sign in when you login. We believe this to be an issue with the UCL API, rather than our web app, since other apps displayed on the UCL API examples page exhibit the same issue.	Low
UCL API SSL	Since our web app is not secure (it operates on HTTP only), once it has been deployed, the UCL API developer dashboard will not accept the IP address of the app as a valid callback. This means that the login functionality only works locally and not on the deployed website.	Medium
Browse Projects	Success stories page - if you click outside the projects - it redirects you as in when you click on the project itself	Low

Table 8.2: Bug report with descriptions and priorities

8.3 Individual Contribution Table

Work Packages	Isobel	Arjun	Tania
Client Liaison	100%	0%	0%
Requirements Analysis	25%	25%	50%
Research	25%	0%	75%
UI Design	45%	45%	10%
Coding - Front end	10%	55%	35%
Coding - Back end	100%	0%	0%
Testing	30%	20%	50%
Bi-weekly Report	52%	10%	38%
Project Report	33%	33%	33%
Video Contents	33%	33%	33%
Video Editing	0%	100%	0%
Overall Contribution	33%	33%	33%
Roles	Client liaison, UI Designer, Programmer, Report Editor	UI Designer, Programmer, Report Editor, Video Editor	Researcher, Programmer, Tester, Report Editor

Table 8.3: Individual Contributions

8.4 Critical Evaluation

8.4.1 User interface and user experience design

Clarity and Simplicity: Although our goal was to create a clear and simple application, feedback from user acceptance testing revealed that some features are not easily discoverable for new users. This indicates a need for further refinement to enhance the user experience..

Consistency and Predictability: Our website exhibits consistent design elements, including fonts, colors, icons, and button styles, throughout the entire site. This consistency facilitates users' ability to predict the interface's behavior and contributes to improved usability.

Visual Hierarchy and Structure: While our website does employ various structural elements, such as active menus and breadcrumbs, to guide users, feedback has shown that

the customer journey is not entirely clear. Some users reported confusion regarding buttons leading to different sections of the menu. To address this, we may need to reassess and modify the visual hierarchy to create a more logical user flow.

Accessibility: As an inclusive institution, UCL values accessibility and strives to ensure that design elements like color contrast, font sizes, and alternative text for images meet accessibility standards such as WCAG 2.1. Our heuristic evaluation highlighted this aspect, and we plan to address it in our future work.

Feedback and Error Handling: It is essential for users to understand and correct errors with minimal difficulty. Our forms currently handle incorrect inputs by displaying error messages and instructions, such as "Investment needs to be a number." This approach helps users identify and rectify errors efficiently.

8.4.2 Functionality

Overall, the application covers the vast majority of the desired functionality, when deployed locally. It fulfilled 100% of the "must have" and "should have" MoSCoW requirements, only not fulfilling one "should have" requirement (of filtering and sorting the listings) due to time constraints. This is reflected by our unit and integration tests, which showed that when run locally, 100% of tests passed.

However, issues during deployment have meant that not all of the app's functions work as expected when it is deployed to the virtual machine. For instance, since the app does not operate on a secure protocol (it is HTTP only), the UCL API will not accept the virtual machine's IP address as a valid callback URL. This means that the login functionality for UCL staff and students does not work on the deployed version of the app.

Future work for this application will include securing the protocol to re-enable the login functionality. Whilst the team covered almost all of the MoSCoW requirements, user acceptance testing showed that some of the more basic functionality had been overlooked. For example, some users reported that they would like to be able to edit their listings, and this functionality should be added in any further work.

8.4.3 Stability

To help improve the stability of the web app, error handling was implemented throughout the code. For example, ensure the status code of HTTP request is checked before trying to access any data in the response (that may be undefined). The unit and integration testing revealed minimal bugs, showing that the error handling was effective. Ensuring there are no bugs in the code is crucial for maintaining stability, as undetected bugs may crash the server.

The results of the compatibility and responsiveness tests showed that the web app is sufficiently stable when viewed in a desktop environment. However, mobile environments had only partial responsiveness, showing that further work is needed to ensure stability on mobile devices.

8.4.4 Efficiency

Throughout the project, we have developed the website with a focus on efficiency. We used React for the front-end, a popular and powerful JavaScript library that excels in creating fast, responsive user interfaces. React's component-based architecture and virtual DOM allow for efficient rendering and updating of elements on the web page, contributing to an improved overall performance.

For styling and design consistency, we used Material UI, a well-known library that provides pre-built components following Google's Material Design guidelines. Material UI helped us streamline the development process while ensuring a polished and modern look. On the back-end, we chose Node.js for its high-performance capabilities and efficient resource management. Node.js is built on the V8 JavaScript engine and uses an event-driven model, allowing it to handle multiple connections and tasks simultaneously without a significant performance loss.

Our data was stored in a MySQL database, hosted on Microsoft's Azure cloud platform. This combination provided us with a reliable and scalable data storage solution, which contributed to the efficient use of resources and the capacity to handle increased workloads.

Scalability was an important aspect of our project. React's modular structure and Node.js's event-driven architecture would make it easy for our application to adapt to increased user traffic and data volume. Azure's cloud-based infrastructure would allow us to scale our MySQL database as needed, ensuring optimal performance under various load conditions.

Throughout the development process, we prioritised efficient design patterns to minimise resource consumption. React's built-in performance optimisations, such as the virtual DOM and lazy-loading, played a significant role in enhancing our application's efficiency. On the back-end, the use of Node.js, combined with our careful database schema design, allowed us to have good query performance and data retrieval speed.

8.4.5 Compatibility

The project's main objective is to implement responsive design for the website to ensure seamless user experience across various screen sizes and devices. This section provides a critical evaluation of the project's compatibility based on the tools, frameworks, implementation, and testing conducted.

Tools and Frameworks: The use of CSS Media Queries and flexible grid layouts to create breakpoints and adjust layouts, typography, and other design elements is an effective approach. However, the reliance on JavaScript for the mobile header could limit compatibility with certain devices or browsers that have JavaScript disabled.

Implementation: While one-third of the website pages are fully responsive, two-thirds of the pages still need optimization for mobile devices. This indicates that the project is not yet

complete, and a significant portion of the website may not provide a seamless user experience across all devices.

Testing Methodology: Using Google Chrome's DevTools to emulate different screen sizes and resolutions is a reliable method for testing responsiveness.

Test Results: The home page exhibits excellent responsiveness and compatibility, while some of the remaining pages lack mobile functionality. These shortcomings need to be addressed to provide a seamless user experience across all devices.

8.4.6 Maintainability

The long-term maintainability of the application remains to be determined. Nevertheless, during the development process, the team of developers made a conscious effort to maintain consistency in naming variables and functions, making it more accessible for each member to understand one another's code. However, it should be noted that some parts of the code may lack sufficient comments, which could pose challenges for future reviewers.

To enhance maintainability, the team opted to use popular and widely-adopted technologies, such as the React framework and the Material-UI library. Additionally, NPM was utilised to keep track of all the packages used in the project. It is important to mention that some packages experienced conflicts, which, while manageable at present, might pose challenges for future maintainers.

Despite these challenges, the use of established technologies ensures that a broader pool of developers can easily understand and modify the codebase if required, thus contributing to the overall maintainability of the application.

8.4.7 Project Management

During this project, the team implemented the Agile Scrum framework, with our sprints having a two-week duration to align with the deadlines for the bi-weekly reports. The team met regularly, having stand-up meetings twice weekly, as well as a bi-weekly sprint planning session. In advance of the sprint-planning session, the team leader would draft a list of proposed tasks for the sprint, and the team would work together to assign them based on team-member experience and capacity.

Throughout the project, team communication and organisation was strong, with all resources being pooled on a shared Teams channel and regularly pushing code to the shared repository.

Overall, the project ran somewhat behind the original schedule, particularly once coding commenced. Upon reflection, the team spent too long learning about our chosen web technologies when we should have begun implementing them sooner and learning on-the-go. Additionally, since the entire team was new to app development, effort estimation proved challenging. Tasks often took significantly longer than estimated, contributing to the late running of the project. In hindsight, the team should have spend less time implementing the could-have and should-have features, and more time testing and deploying the application in an attempt to combat the late-running of the project.

8.5 Future Work

Table 8.4: Feature improvements and descriptions

Feature	Description
Search	For listings pages and for admin dashboard
Filter	For listings pages and for admin dashboard
Sort	For listings pages and for admin dashboard
Responsiveness Device	Add responsiveness for all pages for mobile phone, and also add responsiveness for more types of tablets
Secure protocol (HTTPS)	Purchase and deploy our app on a domain, as well as obtaining an SSL certificate for that domain. This would allow the login functionality to work on the deployed server.
Undo button for admin dashboard	Currently, if an admin deletes a listing it cannot be undone
Footer missing links	
Add persistence to login functionality	Currently, if the user navigates away from the page, they will have to log back in when they return. Login only persists as long as the user remains on the site.
Allow users to view and edit their listings	
Implement JWT refresh tokens	Currently, only a single JWT is issued with an expiry of 1 hour. It would be better to implement refresh tokens as well for security purposes.
Populate database with additional listings	
Software publishing stuck accordion	Fix the accordion
Browse Projects	Should only redirect you to listings if clicked in the specific area
Consistency	Buttons in a menu section to only direct in that section
Connect Form	Connect Form should have a start date after the apply date
Detailed Projects	Make it more visible that it's possible to click

Continued on next page

Table 8.4 – *Continued from previous page*

Feature	**Description**
Forms Visibility	Make forms more easy to find
Accessibility	Ensure that design elements like color contrast, font sizes, and alternative text for images meet accessibility standards such as WCAG 2.1

Table 8.4: Feature improvements and descriptions

References

[1] Bittner, K. and Spence, I. (2008). Use case modeling. Boston, Ma: Addison Wesley

[2] (no date) Harvard Office of Technology Development. Harvard. Available at: https://otd.harvard.edu/ (Accessed: January 23, 2023).

[3] (no date) Office of Technology Licensing. Stanford. Available at: https://otl.stanford.edu/ (Accessed: January 23, 2023).

[4] Massachusetts Institute of Technology (no date) MIT Technology Licensing Office. MIT. Available at: https://tlo.mit.edu/ (Accessed: January 23, 2023).

[5] Rogers, Y., Sharp, H., Preece, J. (2011). Interaction Design: Beyond human-computer interaction. Retrieved from https://arl.human.cornell.edu/879Readings/Interaction

Appendix

8.6 User Manual

Our website has 3 main users: Investors, UCL faculty and students, and admin. This section provides a user manual for each type of user.

8.6.1 Investors

The first page the investors see is the home page with information about Apps-Portal.

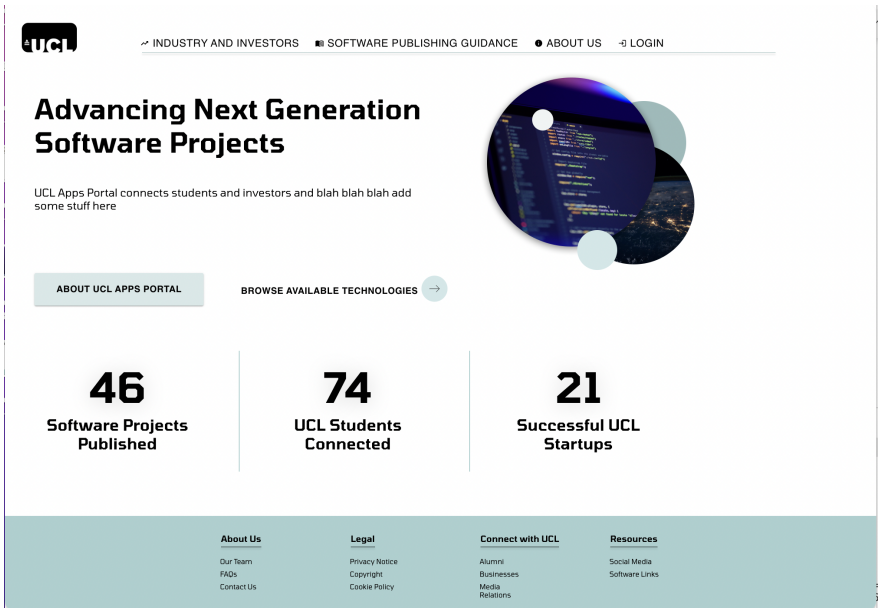

Figure 8.1: Home.Page

They can interact with the website using the menu and go to their dedicated section.

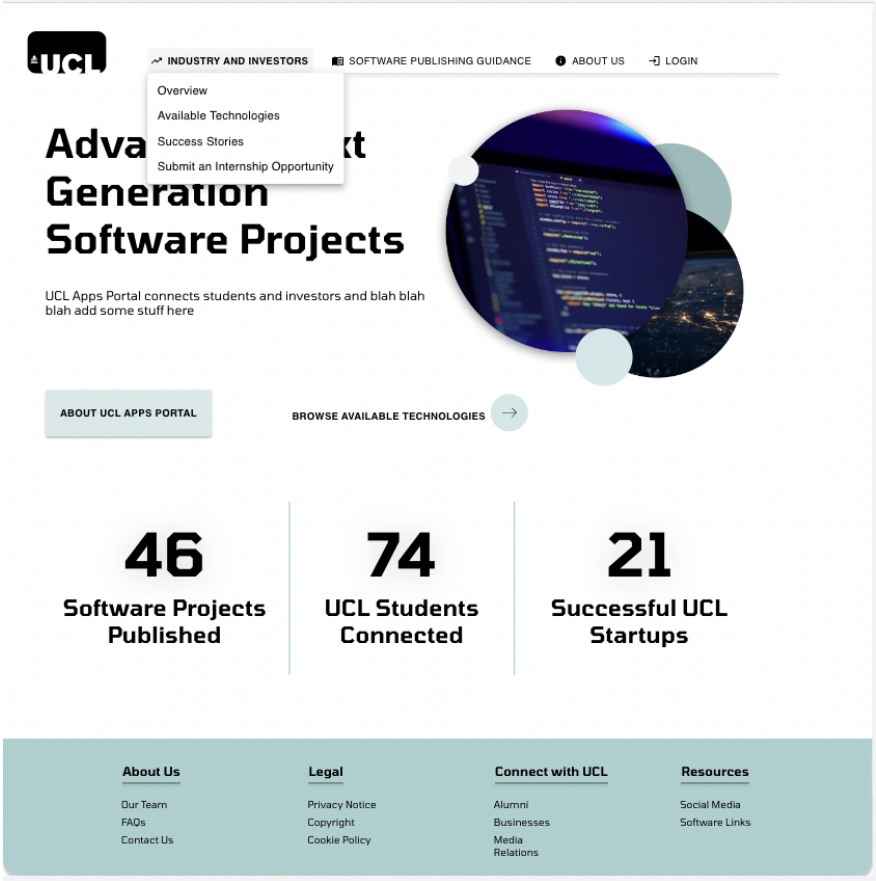

Figure 8.2: Header

Once they clicked available technology, the investors are redirected to the browse technologies page. The menu highlights the active component for better clarity.

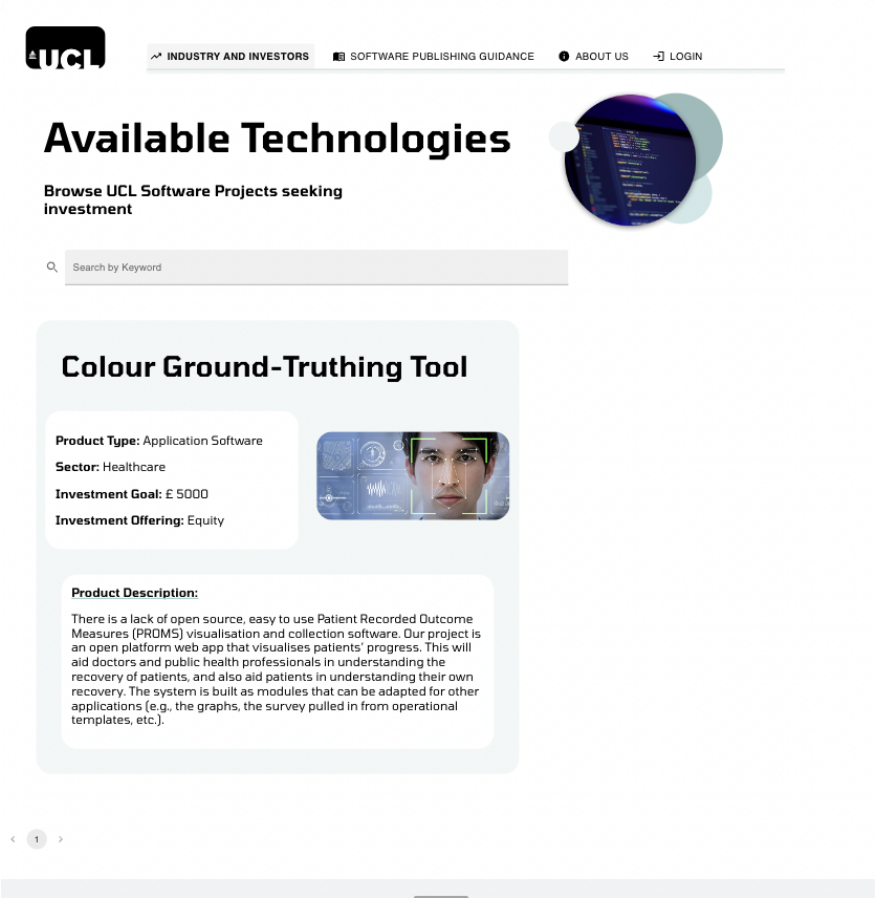

Figure 8.3: Available Technology

Investors can view a detailed listing, including embedded videos and pitch deck. Investors are aware of where they are in the website because of the breadcrumb.

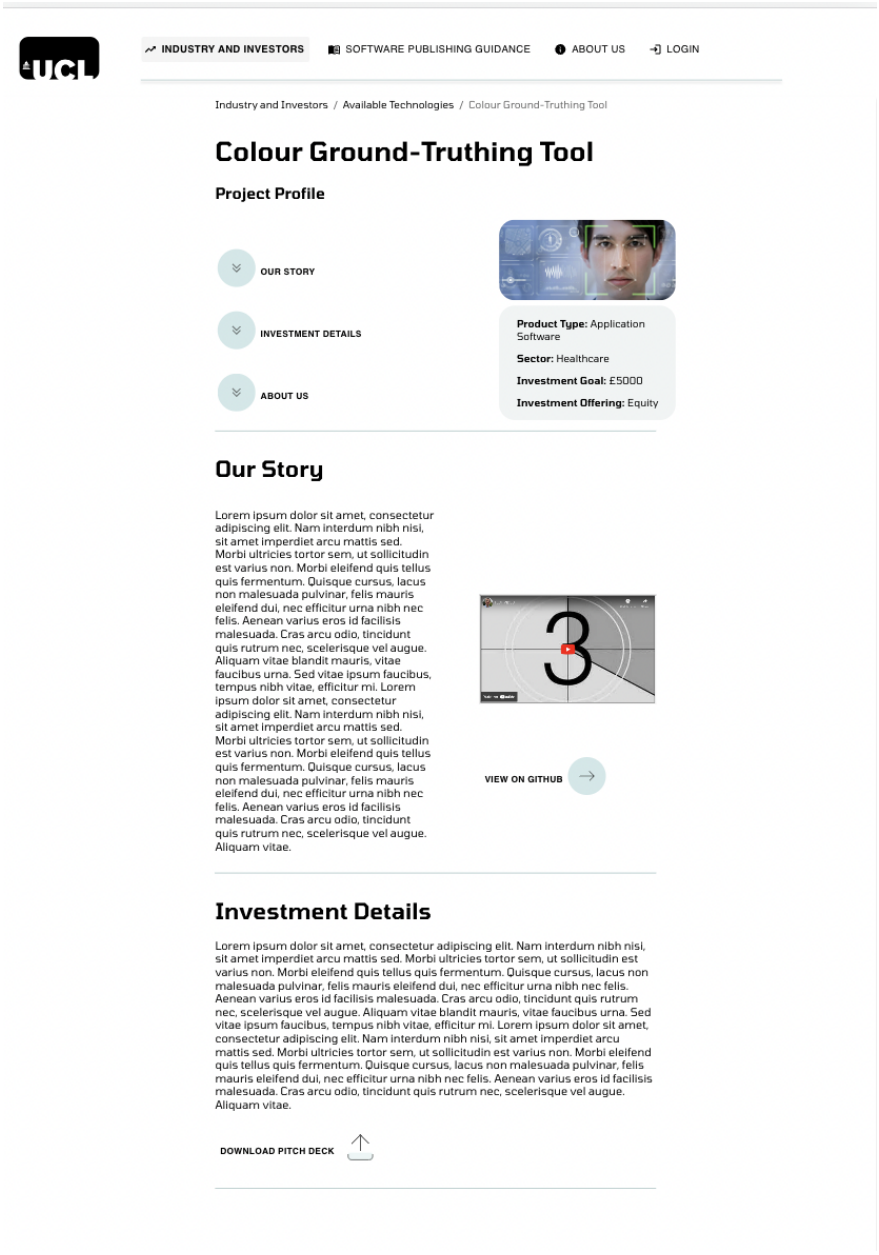

Figure 8.4: Detailed Listing

8.6.2 Faculty and Students

UCL members can click the login button and a new menu component will be displayed for them.

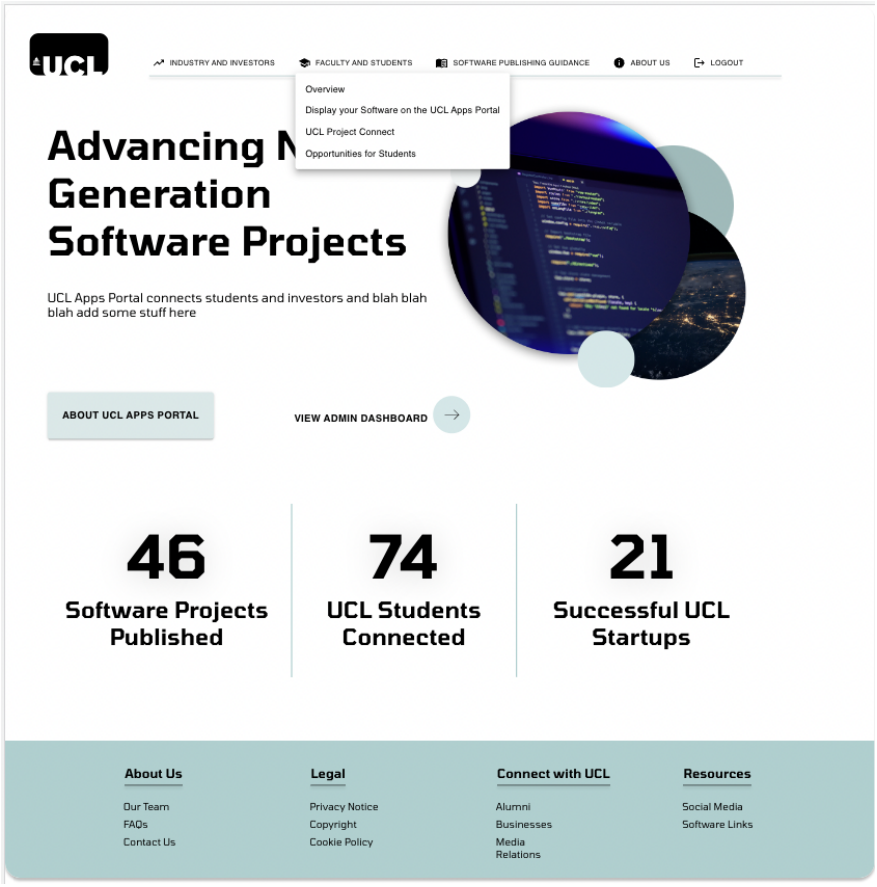

Figure 8.5: Faculty Login

UCL students can browse Project Connect opportunities.

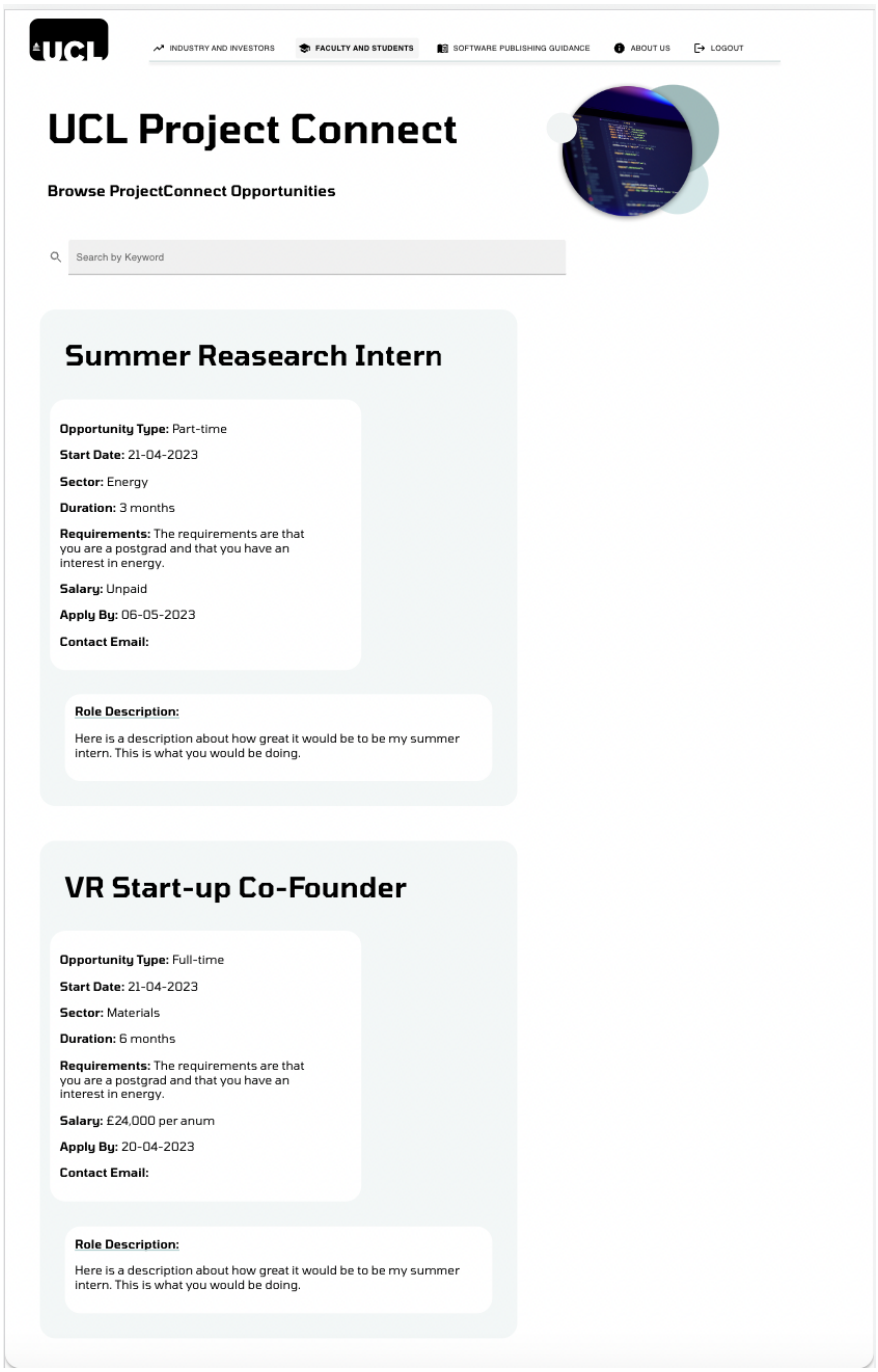

Figure 8.6: Project Connect

Figure 8.7 shows the software publishing guidance page, displaying information about various aspects of the software publishing process, organised into drop-downs. It is accessible publicly.

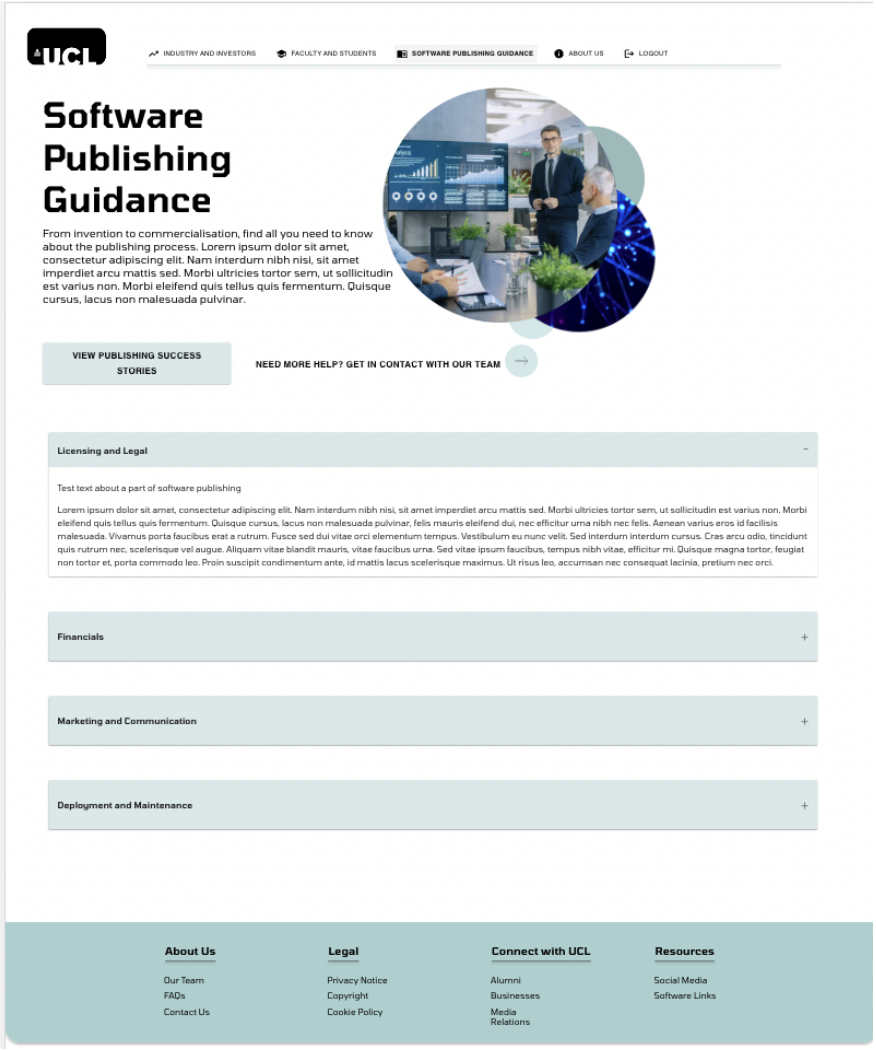

Figure 8.7: Publishing Guidance

Figure 8.8 shows the form that authenticated UCL members can fill out in order to display their available technologies publicly on the apps portal to investors.

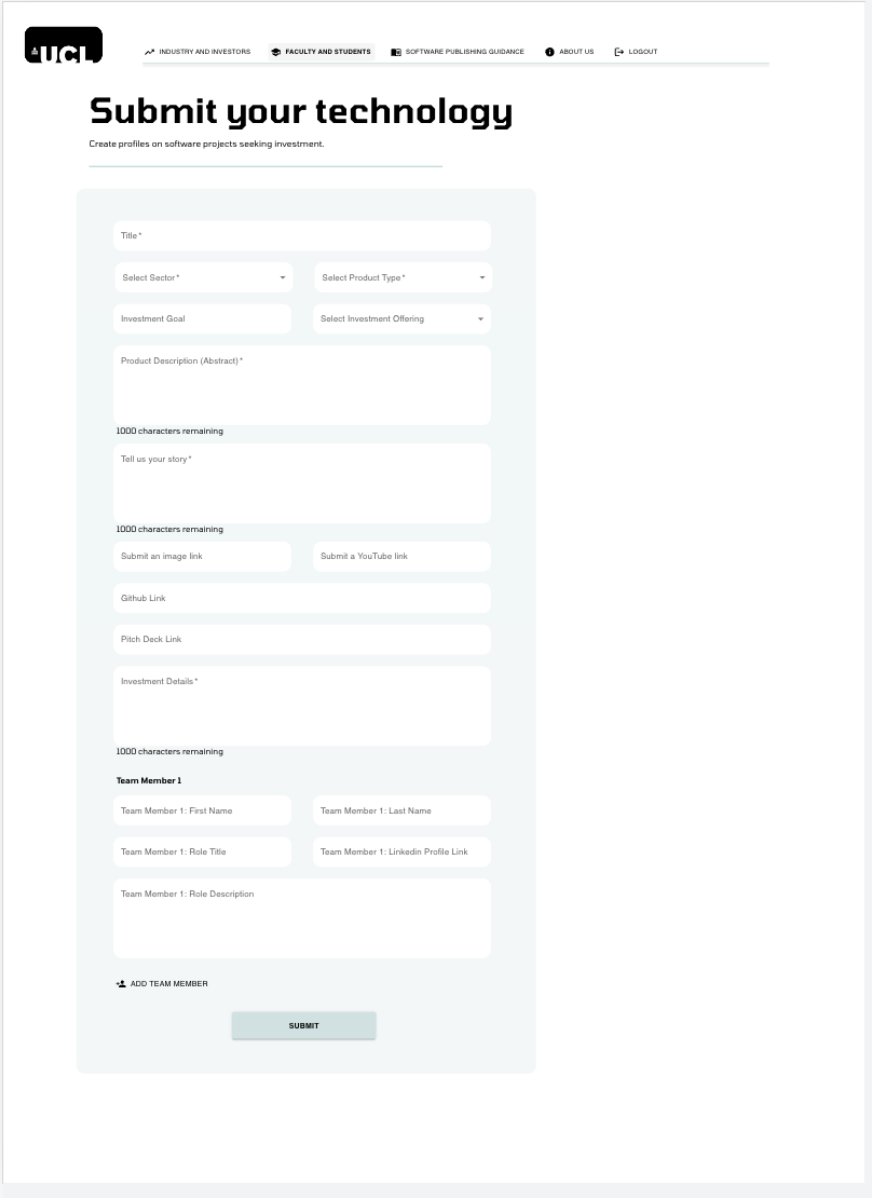

Figure 8.8: Submit Form - Available Technology

8.6.3 Admin

Once the admin logs in, they can navigate to the admin dashboard shown in figure 8.9 from the homepage. There they will see all of the listing in the database organised into available technologies, past projects, and opportunities. The admin will then have the option to approve or un-approve a listing using the toggle button. Any approved listings will be shown in the browse pages. Un-approved listings are only available to the admin. Finally, the admin can press the delete button under a listing to permanently remove it from the database.

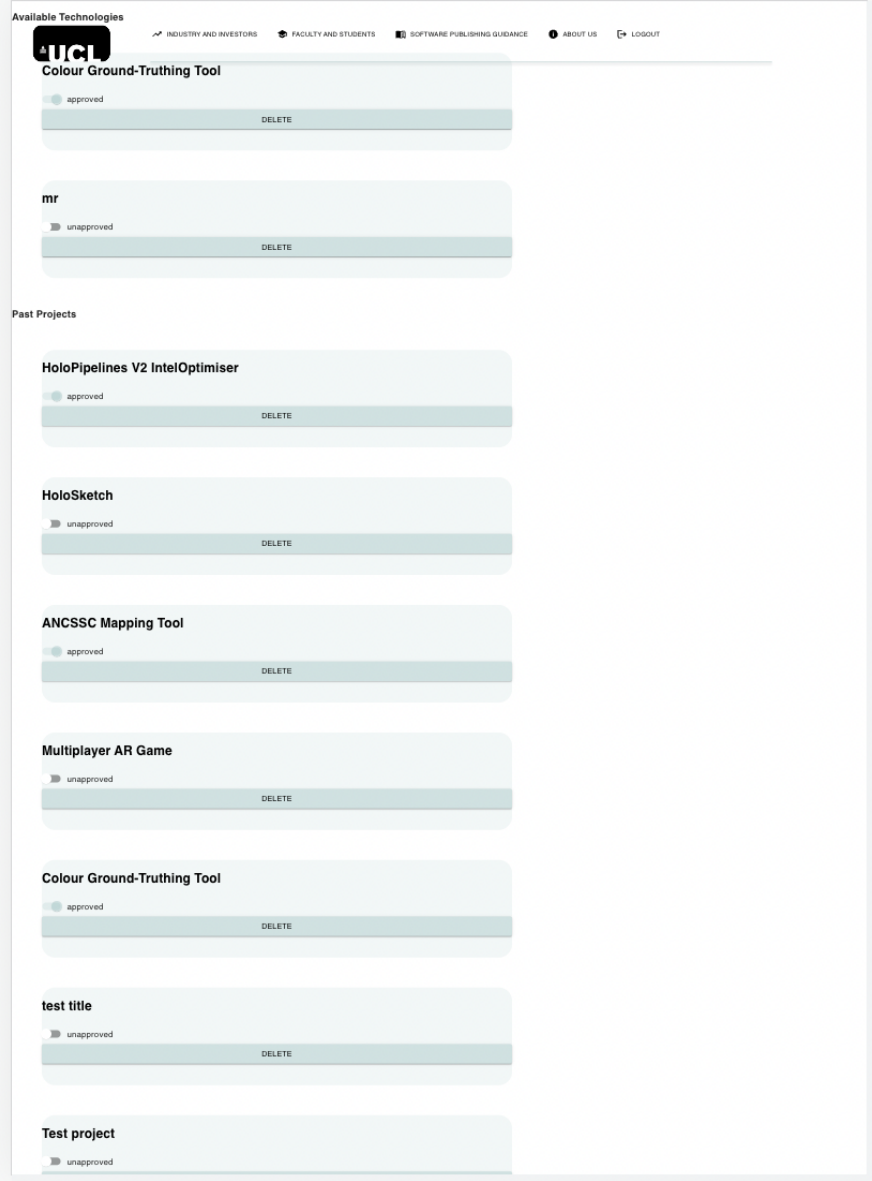

Figure 8.9: Admin Dashboard

8.7 Deployment Manual

8.7.1 Estimated Monthly Cost

- Virtual Machine: $34.46 /month
- Database: $8.50 /month
- **Total Cost**: $42.96 /month

8.7.2 Azure Database

The database for this project was developed as a cloud-native database - i.e, it was created on Azure and then data was entered directly, rather than developing it locally and then deploying it.

1. Create an Azure Database for MySQL flexible server

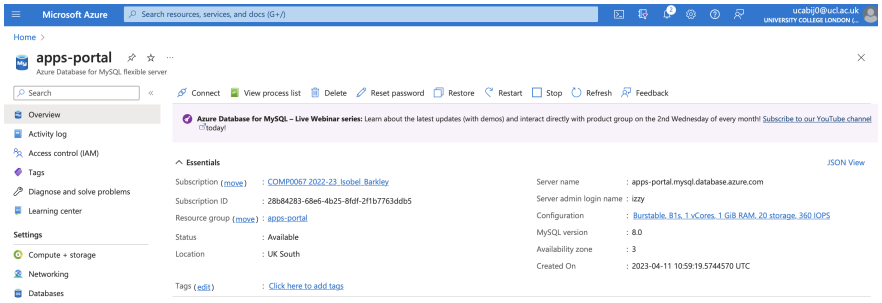

Figure 8.10: Azure Database configuration.

2. Add a firewall rule to allow any IP address

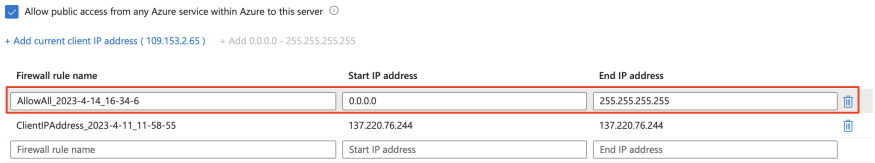

Figure 8.11: Firewall rule to be added.

(a)

3. Download the SSL certificate

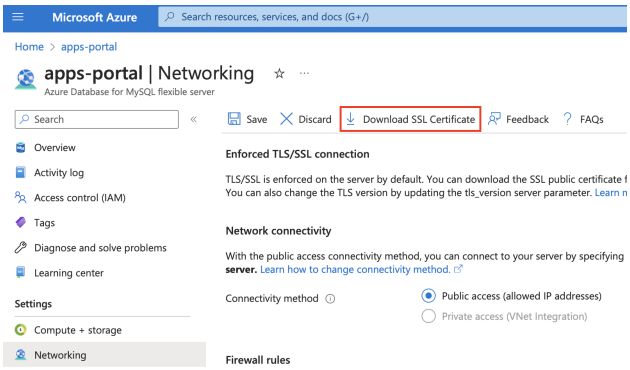

Figure 8.12: Download the SSL certificate for the Azure DB.

(a) Save the certificate with the following path:
`/home/team9/COMP0067_2023_Team09/ucl-apps-portal /server/routes/DigiCertGlobalR`

4. Connect to the Azure database through MySQL Workbench to run SQL queries directly

   (a) Download and install MySQL workbench

   (b) Add a new connection

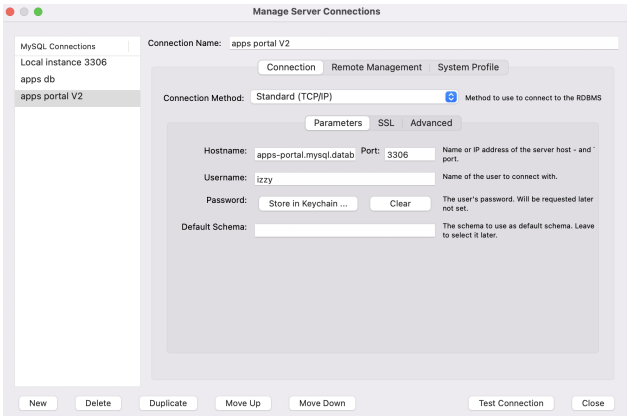

Figure 8.13: MySQL workbench connection configuration.

8.7.3 Azure Virtual Machine

The Azure VM instance used to host the web app uses Ubuntu 20.04. To deploy the web app to the virtual machine, the following steps were taken:

1. Create a build version of the source code

   (a) Delete the `/client/build` folder

   (b) Delete the contents of the `/server/views` folder

   (c) Navigate to the client directory in the terminal

   (d) `> npm run build`

   (e) Copy the contents of the `build` folder and paste it into the `views` folder

   (f) Save and push your code to repository

2. Create an Azure Virtual Machine

   (a) Create an Azure Virtual Machine Resource using Ubuntu 20.04. This project used the `Standard B2s (2 vcpus, 4 GiB memory)`, as anything less struggled to install the node modules and compile the code.

3. Connect to the VM via SSH

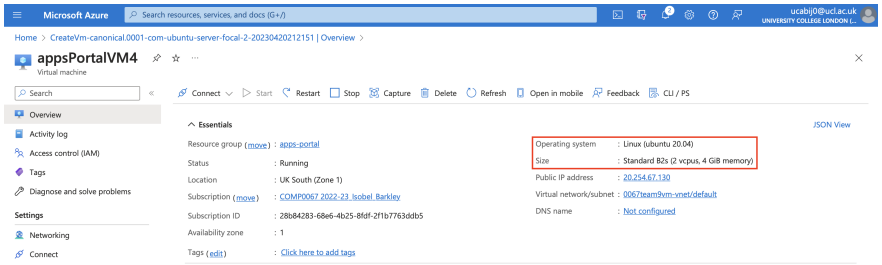

Figure 8.14: Azure Virtual Machine configuration.

(a) Open a terminal.

(b) `> ssh team9@20.254.67.130`

(c) Enter password: `UCLappsportal2023`

4. Install dependencies

(a) `> sudo apt update`

(b) `> sudo apt install nodejs npm`

(c) Update the version of node:
```
> npm install -g n
> n 18.13.0
```

(d) `> sudo apt update`

(e) `> sudo apt upgrade`

(f) `> sudo apt install nginx`

5. Configure nginx to redirect traffic to server port

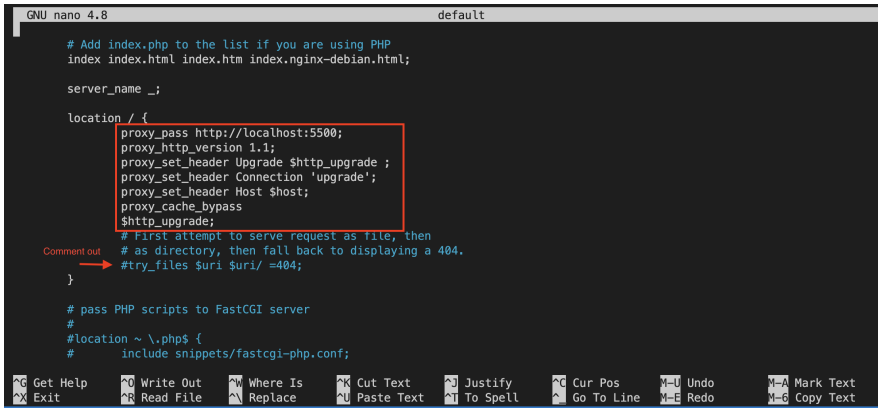

Figure 8.15: Make these amendments to the default file to redirect traffic.

(a) `> cd /etc/nginx/sites-available`

(b) `> sudo nano default`

(c) Add the following lines under "location":

```
> proxy_pass http://localhost:5500;
> proxy_http_version 1.1;
> proxy_set_header Upgrade $http_upgrade ;
> proxy_set_header Connection 'upgrade';
> proxy_set_header Host $host; ;
> proxy_cache_bypass $http_upgrade;
```

(d) Comment out the line `"try_files $uri $uri/ =404"` by adding "#"

(e) Save and exit text editor

6. Install and configure pm2

   (a) Add here

7. Clone build version to VM from GitHub

   (a) Generate SSH key:
   `> ssh-keygen`

   (b) View key:
   `> cd /.ssh`
   `> more id_rsa.pub`

   (c) Copy SSH key and add it to GitHub under `settings > SSH keys`

   (d) Navigate to correct directory:
   `> cd /home/team9`

   (e) Clone (optionally specifying a branch): `> git clone git@github.com:UCLComputerScience/C`
   `-b dev`

   (f) Install server dependencies:
   `> cd /COMP0067_2023_Team09/ucl-apps-portal/server`
   `> npm install`

   (g) Install client dependencies:
   `> cd /COMP0067_2023_Team09/ucl-apps-portal/client`
   `> npm install`

8. Set-up .env file

   (a) `> cd /COMP0067_2023_Team09/ucl-apps-portal/server`
   `> sudo touch .env`
   `> sudo nano .env`

   (b) Add relevant data to .env file and save:

```
# ports
CLIENT_PORT = 3000
```

```
SERVER_PORT = 5500

# DB config
DB_HOST = apps-portal.mysql.database.azure.com
DB_USER = izzy
DB_PASSWORD = UCLappsportal2023
DB_NAME = app
DB_PORT = 3306
DB_SSL_CERT_PATH = /home/team9/COMP0067_2023_Team09/ucl-apps-portal/server/rout

# oauth config
CLIENT_ID = 7930762428231008.5471166532286282
CLIENT_SECRET = 912d71221b8a52b38ddabb25a32f44cf80081b3607b5968e4719574acb027b0

# session
SESSION_SECRET = 80d28c70-e611-45e1-8e34-f3ad945b630e

# jwt
JWT_SECRET = eb303ded-e095-441e-9c93-c2866abcb342

# Admin permissions
ADMIN_EMAIL = <admin email in ucabij0@ucl.ac.uk format>
```

(c) Change the base URL for the React axios instance

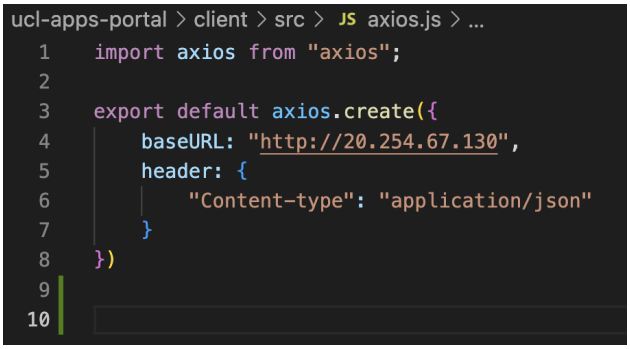

Figure 8.16: Axios config.

i. Navigate to: `/Users/isobelbarkley/Documents/appeng/COMP0067_2023_Team09/ucl-apps-portal/client/src/axios.js`

ii. Set `baseURL: "http://20.254.67.130"`

8.8 Code Citation

No.	Function Name	File Name	Source
1	app.get('/token')	/server/routes/auth.js	https://glitch.com /edit/#!/ uclapi-oauth-demo? path=README.md %3A1%3A0

Table 8.5: Individual Contributions