

University of Sheffield

COM Opportunities Hub: An efficiency based approach for managing student/staff opportunities



Mohammad Tahir Khan

Supervisor: Mr. Andrew Stratton

Module Code: COM3610

A report submitted in fulfillment of the requirements
for the degree of BEng in Software Engineering

in the

Department of Computer Science

May 15, 2024

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name:

Mohammad Tahir Khan

Signature:

MTK

Date:

May 15, 2024

Abstract

Looking for opportunities as a student or staff in university can be a daunting task. The main reason being, an abundance of opportunities and a lack of a centralised system for managing and disseminating them. The Department of Computer Science at the University of Sheffield currently relies heavily on manual processes for these tasks. Although there is a system in place, it is neither efficient nor straightforward, resulting in missed opportunities for interested individuals.

Our project, the COM Opportunities Hub aims to address the aforementioned challenges by developing an efficient web application that streamlines the process of managing and announcing opportunities within the department. For department staff or external individuals responsible for disseminating opportunities, the web app facilitates simple creation and management of opportunities while also automating the email process. For students/staff seeking opportunities, the web app enables efficient browsing, searching, tracking, and setting up preferences for personalised notifications.

The system prioritises user-friendliness, accessibility, and efficiency, with a focus on simplicity and usability. Research was conducted to understand the user needs. Gaps were observed between what the current system offers and what users are looking for. This paper explains an efficiency-based solution that bridges these gaps by leveraging cutting-edge technologies, design principles, implementation strategies, and testing methodologies to develop a robust and user-friendly system.

Acknowledgements

First of all, I would like to thank God, the most gracious and the most merciful, for giving me the strength and patience to complete this project. I would like to express my deepest gratitude to my supervisor, Mr. Andrew Stratton, for his guidance throughout the project. His valuable feedback and suggestions have been instrumental in shaping the project and improving its quality. I would also like to thank my parents, my sisters, and my friends for their unwavering support and encouragement. Their love and belief in me was a constant source of motivation. Last but not least, I would like to thank the Department of Computer Science at the University of Sheffield for providing me with the opportunity to undertake this project and for their support throughout this journey.

Contents

1	Introduction	1
1.1	Project Background	1
1.2	Aims and Objectives	2
1.3	Overview of the Report	2
2	Literature Survey	4
2.1	Existing Research	4
2.1.1	University Emails	4
2.1.2	Notification Systems	5
2.1.3	Web Portals	6
2.1.4	Summary	7
2.2	Emerging Trends	7
2.2.1	Chatbots	7
2.2.2	Web Scraping	8
2.2.3	Integrated Approach: Leveraging AI Chatbots and Web Scraping . . .	8
2.3	UI/UX Design	9
2.3.1	Usability	9
2.3.2	Visual Design	10
2.3.3	Accessibility	11
2.4	Software Architectural Patterns	12
2.4.1	Model-View-Controller (MVC)	12
2.4.2	Model-View-Presenter (MVP)	12
2.4.3	Model-View-ViewModel (MVVM)	13
2.4.4	Summary	13
2.5	Testing and Evaluation: Assessing the System's Effectiveness	13
2.5.1	Testing techniques	14
2.5.2	Evaluation techniques	14
3	Requirements and Analysis	16
3.1	Project Requirements	16
3.1.1	Functional Requirements	16
3.1.2	Non-Functional Requirements	20

3.2	Risk Analysis	20
3.3	Testing and Evaluation	21
3.3.1	Unit Testing	21
3.3.2	Integration Testing	21
3.3.3	System Testing	21
3.3.4	Usability Testing	21
3.3.5	User Acceptance Testing	21
3.4	Ethical, Professional and Legal Issues	22
4	Design	23
4.1	Technology Stack Choices	23
4.2	Overview of the Design	24
4.2.1	User Registration, Authentication and Editing	24
4.2.2	Post Creation and Management	25
4.2.3	Browsing and Tracking Opportunities	26
4.2.4	Personalised Notifications	26
4.2.5	Administration	26
4.3	Database	27
4.4	User Flow Diagram	28
5	Implementation	31
5.1	User Registration, Authentication and Editing	31
5.2	Post Creation and Management	35
5.3	Browsing and Tracking Opportunities	38
5.4	Personalised Notifications	39
5.5	Administration	41
6	Testing	45
6.1	Automated Testing	45
6.1.1	Unit Testing	45
6.1.2	Integration Testing	46
6.1.3	Browser Testing	46
6.2	Manual Testing	47
6.2.1	System Testing	47
6.2.2	Usability Testing	47
7	Results and Discussion	48
7.1	Testing and Evaluation Results	48
7.2	Requirement Evaluation	50
7.3	Future Work	51
8	Conclusion	53

<i>CONTENTS</i>	vi
Appendices	59
A Summary of RSpec Tests	60

List of Figures

2.1	A demonstration of how font size and font style can affect the readability. . .	10
2.2	A demonstration of how colour can be used to draw attention to certain parts of the screen.	10
2.3	A demonstration of how different colours can be used to evoke different feelings and incompatibility of colours.	11
4.1	Ruby on Rails Architecture with MVC Pattern	24
4.2	Database Diagram	27
4.3	User Flow Diagram: User (Browser)	28
4.4	User Flow Diagram: Poster	29
4.5	User Flow Diagram: Admin	30
5.1	Comparison of Landing Page on Desktop and Mobile	32
5.2	Comparison of Sign In Page on Desktop and Mobile	32
5.3	Sign Up and Forgot Password forms	33
5.4	Profile Page on Desktop and Mobile (Poster view)	34
5.5	Edit Profile Page on Desktop and Mobile (Poster view)	35
5.6	New Post Form	36
5.7	My History Page	37
5.8	Post Details Page	37
5.9	Upcoming Page on Desktop and Mobile	38
5.10	Saved Page on Desktop and Mobile	39
5.11	Search Results Page: Mobile	39
5.12	Selecting Tags on Profile Page: Mobile	40
5.13	Notifications on Mobile	41
5.14	Admin Dropdown on Navigation bar	42
5.15	Pending Page	42
5.16	Email Page	43
5.17	Email Draft	43
5.18	Users Page	44
5.19	Edit User Role Form	44

6.1	Unit Test for validating the length of the password	45
6.2	Integration Test for the approval functionality of the Admin	46
7.1	Desktop Lighthouse Score	49
7.2	Mobile Lighthouse Score	49
A.1	Summary of RSpec Tests (1)	60
A.2	Summary of RSpec Tests (2)	61
A.3	Summary of RSpec Tests (3)	62
A.4	Summary of RSpec Tests (4)	63
A.5	Summary of RSpec Tests (5)	64

List of Tables

3.1	Must Have User Stories	18
3.2	Should Have User Stories	19
3.3	Could Have User Stories	19
3.4	User Stories Summary	19
3.5	Non-Functional Requirements	20
3.6	Risks and Mitigation Strategies	20
7.1	Requirement Evaluation	50

Chapter 1

Introduction

This project proposes a web solution to streamline the process of managing and announcing opportunities in the University of Sheffield's Department of Computer Science. In this chapter, we delve into the background of the project, exploring the motivation behind it, and outline the project's aims and objectives. An examination of the current system is presented, highlighting its limitations, and how the proposed solution will enhance it. Finally, we provide an overview of the report, outlining the contents of each chapter.

1.1 Project Background

The Department of Computer Science at the University of Sheffield regularly offers a variety of opportunities for students and staff, ranging from internships and placements to events, and workshops. Currently, these opportunities are announced through various channels, including emails and a dedicated department website (COM-Opportunities 2023). While seemingly functional, the existing system has its drawbacks. It heavily relies on manual processes, resulting in a time-consuming and cumbersome experience.

Receiving a large number of emails, some of which may not be relevant to the recipient is a less than ideal method of announcing opportunities. Being a student myself, I have experienced this first-hand. Due to the large number of emails I receive, the University email inbox is often cluttered with emails, most of which are either not read or are irrelevant to me. This is a common problem faced by many students in the department, often causing them to miss out on opportunities that may be of interest to them. Moreover, given that certain opportunities hold significant importance for both students and staff, missing out on them can be detrimental to their career. Hence, the current scenario, where crucial opportunities are lost in barrage of emails, is far from optimal.

The other channel for announcing opportunities, the department website, is not straightforward either. Posting new opportunities on the website is done manually which can be quite time-intensive and navigating through the right opportunities is not intuitive either. Although

the website includes upcoming, ongoing, and archived opportunities, some of which recur periodically, its format makes it challenging to navigate and identify relevant ones as they are all listed on a single published web page. Moreover, the website lacks any functionality to track or check on recurring opportunities or events. All these shortcomings raise the need for a new system that can address these issues and enhance the current process making it more efficient.

1.2 Aims and Objectives

The primary objective of this project is to design and develop an efficient web system for managing and announcing opportunities within the Department of Computer Science. The main aim is to facilitate seamless communication for staff and students regarding opportunities. It seeks to improve the existing system by providing a user-friendly platform, replacing manual processes with automated ones, and incorporating additional features to elevate effectiveness.

The project aims to offer a simple and straightforward browser-based platform for staff or external posters to effortlessly post and manage opportunities while presenting them in an organised manner for interested students or staff. Additionally, it aims to provide, potentially a mobile app to enable browsing, searching, and tracking opportunities along with personalised notifications for users based on their preferences, enhancing the overall browsing experience.

Moreover, the system aims to incorporate features for administrative tasks, ensuring efficient management of users and posts. We aim to develop a system that is easy and intuitive to use for both posters and opportunity seekers, emphasising simplicity, accessibility, and usability. The goal is not only to bridge existing gaps but also to introduce innovative features and functionality, thereby enhancing the overall system.

1.3 Overview of the Report

This report is organised into 8 chapters.

- Chapter 1, Introduction: Provides an overview of the project background, motivation, aims, and objectives.
- Chapter 2, Literature Survey: Explores existing research, studies, and technologies relevant to the project, along with a discussion on UI/UX design, accessibility, usability, software architecture, and testing and evaluation strategies.
- Chapter 3, Requirements and Analysis: Outlines project requirements and conducts an analysis of crucial aspects, including risks, testing, and ethical, professional, and legal issues.

- Chapter 4, Design: Discusses the design of the proposed system, including the technology stack, architecture, user flow, and database design.
- Chapter 5, Implementation: Details the implementation of the system, including the developed components, features, and functionalities.
- Chapter 6, Testing: Discusses the testing of the system, including automated and manual testing.
- Chapter 7, Results and Discussion: Presents the results of the system implementation and testing, also discusses the findings, including any future work.
- Chapter 8, Conclusion: Concludes the project, summarising the achievements, and reflecting on the project as a whole.

Chapter 2

Literature Survey

This chapter discusses the research findings, existing studies, applicable technologies, design techniques, and evaluation strategies relevant to the project. A systematic approach was used to conduct the literature review as shown in (Kofod-Petersen 2014). The following research questions were used to guide the literature review:

- RQ1 - What has been reported regarding digital communication of events and opportunities for students and staff at universities?
- RQ2 - How do emerging trends, such as AI, as reported in the literature, could be employed to aid in the development of a new system?
- RQ3 - How do suitable design, testing, and evaluation strategies as reported in the literature could improve the design and evaluate the effectiveness of the proposed system?

2.1 Existing Research

This section examines existing research relevant to the project, focusing on digital communication of events or opportunities at university. The literature identified three main forms and methods of digital communication: university emails, notification systems, and web portals. We explore each of these methods in detail below.

2.1.1 University Emails

Email is the most prevalent mode of communication within universities, as highlighted by Robson et al. (2016). Despite its perceived effectiveness, the research indicates a lack of commitment to using the university email by students. A 2017 study discovered that only 23.2 percent of the students consistently utilise the university email, unlike the faculty members who show 80 percent commitment (Alwagadani & Alomar 2017). This shows the potential of emails as a communication tool, especially for staff, but also highlights the need for a better system for students. Moreover, students may not check their emails frequently due to the

overwhelming number of emails they receive, leading them to miss important announcements from the university (Waycott et al. 2010). This can be detrimental to their academic and professional growth as they may not be aware of the opportunities available to them.

Chodos & Cohen (2006) addressed these problems by presenting an innovative approach for sharing research opportunities at universities. They looked past the traditional method of sending lengthy emails to a large number of recipients and designed a system that suggests using short emails that link to detailed web pages. They incorporated a searchable opportunities database, a well-maintained recipient list targeted to the right audience, and more. The paper also discusses solutions to manage email overload, as mentioned above, by using a well-maintained recipient list, allowing users to manage their email preferences, and using well-formatted emails with minimal text and links to detailed web pages. This is a good solution to the problems mentioned by Alwagadani & Alomar (2017) and Waycott et al. (2010) to some extent and can be further explored, certain aspects of which like the recipient list for a targeted audience, using short, well-formatted emails, and allowing users to manage their email preferences can be explored for our project to fill up the gaps in the current system.

2.1.2 Notification Systems

Notification systems have become an integral part of our daily lives. People use it all the time to get notified about things like weather, news, and more. It can be said that we as a society are dependent on notification systems in many ways which is evident by their widespread use in many different fields, such as, disaster-related emergency (Sikder et al. 2017), health care (Subbe et al. 2017), and more. This section narrows the focus to the use of notification systems for university announcements which is especially relevant to our mobile app component used by students and staff to receive notifications about opportunities.

Sanmorino & Fajri (2018) and Heryati et al. (2019) targeted android smartphone based notification systems that can be used to send announcements to students. The main focus of Sanmorino & Fajri (2018) was to design a system that can meet the needs of academic announcements broadcast quickly and massively, while also not being burdened by time, storage, and security issues to operate the system. Heryati et al. (2019) on the other hand, focused on designing a system for a targeted audience that can be operated at a low cost. Both designs offer valuable insights for our project, particularly in personalised notifications and security features. Personalised notifications can be used to send notifications to a targeted audience, while security features can be used to make sure that the information is not leaked to the wrong audience.

However, it's essential to acknowledge the potential nuisance and distraction that notifications can cause. A study conducted by Heitmayer & Lahlou (2021) highlighted participants' annoyance with notifications, which they described as disruptive and often led them to

immediately interrupt their current task and check their phone. This poses a challenge for our project as we want the notifications to be non-disruptive and only sent to the right audience at the right time. To address this, our design can incorporate techniques that can suspend notifications for a certain period of time or until the user is done with their current task.

2.1.3 Web Portals

Web portals within the context of this project refers to websites facilitating the posting of opportunities for students and staff by the university. While there are many web portals available for similar purposes, their design and functionality can be improved to enhance the user experience and effectiveness. The University of Sheffield's computer science opportunity hub (COM-Opportunities 2023) is one such example. Although functional, the website could further benefit from incorporating features such as, the ability to check on recurring opportunities, automated post creation, and more.

Jobs are another form of opportunity that students and staff are interested in. There exists popular job search websites such as, Monster (monster.co.uk 2023) and Reed (reed.co.uk 2023) that can be considered web portals for job opportunities. These websites are very popular for job advertisement, and can be taken as an example for our project to some extent. However, the thing we need to avoid is the prioritisation of income over user experience. These websites let the employers push their posts to job seekers by paying extra which contradicts our project's goal of ensuring only relevant opportunities reach users as it can lead to irrelevant posts reaching the users. On the other hand, these websites can be used as an example for our project in terms of their usability and design. They offer a simple and intuitive interface for job seekers to search for jobs, and for employers to post jobs. This is an important aspect of our project as well and can be incorporated into our design.

Improving web portals for university opportunity announcements has been researched before. Choudhary et al. (2016) proposed a placement portal focusing on automating the placement process, aiding the alumni students in finding suitable placements by sharing opportunities and improving the communication between students and university. Their design included managing posts, notifications, student lists to target right students according to the job criteria, and more. Some of these features like automated opportunity sharing using notifications and student lists can be adapted for our project. While their design is quite specific to the placement process, it can be used as a reference for our project to some extent.

Onyesolu et al. (2013) on the other hand suggested a different approach consisting of using an SMS based user interface system. Their design improved information sharing and communication between students and university by using SMS to send timely information such as examination results, examination schedules and more. The design created a synergy between the university portal and the SMS system, allowing students to access information from the portal through

SMS in real-time ultimately improving student satisfaction. This is a good approach to improve communication between students and university, and can help us in automating the process of sending notifications to users from the posting hub.

2.1.4 Summary

The literature review in this section provides a comprehensive understanding of existing research and methods for digital communication of events and opportunities in university settings. It emphasises the effectiveness of emails, notification systems, and web portals for this purpose. The insights revealed the challenges of email underutilisation by students due to overload and the potential distractions from notification systems. Implementing these findings into the project involves adopting concise and well-formatted email strategies, empowering users to manage preferences, and ensuring personalised, non-disruptive notifications through techniques to suspend incoming notifications. The examination of web portals suggests refining the design and functionality of the university's current system, drawing inspiration from successful job search portals for enhanced usability. Integration of these insights aims to create an improved system with automated opportunity sharing, user-friendly interfaces, and effective communication, ultimately addressing identified gaps and streamlining the announcement process for university opportunities.

2.2 Emerging Trends

The field of computer science has seen a rapid growth in the utilisation of artificial intelligence (AI) in recent years. Looking at the fascinating growth and popularity of AI, it becomes important to explore its use in our project. This section discusses some emerging tools and trends that can aid in the development of our project.

2.2.1 Chatbots

Chatbots are one of the most common and widely used application of AI, with examples like, chatGPT (openai.com/chatgpt 2023), Siri (apple.com/siri 2023), Alexa (alexa.amazon.co.uk 2023), and Google Assistant (assistant.google.com 2023) being broadly used by many people on a daily basis for various purposes such as customer support and personal assistance. However, their core purpose is to provide a conversational experience to users. This raises the question, could chatbots be of any use to our project?

A survey conducted by Wolff et al. (2019) explored the potential use of chatbots in university settings, where participants highlighted their expectations from a chatbot. The most mentioned response was the need for chatbots to address questions related to events and serve as a platform for inquiries about open vacancies or possible internships. This indicates that the use of chatbots in university settings is not a new concept and has garnered attention and interest from students and staff.

Carayannopoulos (2017) presented a chatbot designed for the purpose of information acquisition in university. The chatbot could answer inquiries about upcoming events, pending work, and incorporated features like push notifications that the users could opt in and out of. Further testing of the chatbot on a group of students revealed a positive response, highlighting their satisfaction with the chatbot's features. This shows the potential of chatbots in context of digital communication for providing information to students and staff, reinforcing the idea that chatbots can be considered for our project.

2.2.2 Web Scraping

Web scraping or web crawling is a swift and effective method to gather data or information from across thousands, or over millions, of web pages (Khder 2021). Typically, we cannot rely on APIs to access information from websites as they are either not provided or are restricted by what data is available and how frequently it can be accessed (Lawson 2015). Hence, instead of manually copying the information from websites, web scraping offers automated solution for extracting data precisely and quickly than a human (Lawson 2015).

In our project, manually gathering information about opportunities from various sources for posting on the hub could be time-consuming. However, web scraping offers a solution to this challenge by automating the data collection process.

Web scraping enables structured information extraction from websites without the need for complex coding. Tools and libraries such as Octoparse (octoparse.com 2023), Scrapy (scrapy.org 2023), and Import.io (import.io 2023) provide additional options for efficient data extraction. Integrating web scraping into our project can significantly reduce the time and effort required for collecting information from different websites, ultimately enhancing productivity.

2.2.3 Integrated Approach: Leveraging AI Chatbots and Web Scraping

An integrated approach combining AI chatbots and web scraping can be a powerful tool when it comes to automating the process of collecting and sharing information. An illustrative example comes from Thakkar et al. (2018), who developed an AI chatbot for the purpose of answering student queries and reducing search time. To do this, they employed web scraping to gather information from the university website. Their tool of choice was Import.io (import.io 2023), which uses AI and machine learning for automated data extraction.

The research indicates that AI chatbots and web scraping can work synergistically to automate the process of gathering and sharing information, thus reducing the time and effort required compared to manual methods. This integrated approach holds promise for our project, especially in automating the collection of opportunity information from various sources

for posting on the hub. The chatbot can handle user queries and provide details about opportunities, while web scraping can extract relevant information from websites, streamlining the process and saving valuable time and resources.

2.3 UI/UX Design

When user interacts with a computer system, they do so through the user interface (UI) (Stone et al. 2005). User experience (UX), on the other hand, is a broad discipline with an intricate set of components that contributes to the overall experience of a person using a digital product (Ritter & Winterbottom 2017). Although UI and UX are often used interchangeably, they represent distinct concepts. While UI focuses on the interface, UX goes beyond, concentrating on the entire user experience, including aspects like usability and accessibility.

The importance of UI/UX design in software development cannot be overstated, given their huge impact on the success of a product. According to Orlova (2016), humans are most likely to cause errors or system failures, hence design of the system must take susceptibility of the human for errors into account to avoid them. This highlights how crucial UI/UX design is for a robust and successful system. This section delves into key aspects of UI/UX design, exploring how these principles can enhance the design of our project.

2.3.1 Usability

ISO (2018) defines usability as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use”. Simply put, usability is characterised by the system’s simplicity and ease of use. In accordance with this, Wong (2016) proposed four strategies to achieve simplicity in design which are as follows:

- **Maintain Clarity:** Understand and design for your users’ main goals
- **Make Use of Automation:** Design for a minimum amount of conscious and cognitive effort
- **Limit Options:** Design for a strong “information scent”
- **Reduce the “Gulf of Execution”:** Make your users see why they should use your product

The four strategies mentioned above beautifully capture the key elements of simplicity in design and are worth considering for our project. Usability or simplicity might not be the first thing that comes to mind during system development, but it is a critical aspect that significantly influences the success of any digital project. Joe Sparano, an American graphic designer and educator says it best, “Good design is obvious. Great design is transparent”.

2.3.2 Visual Design

An additional vital aspect of UI/UX is visual design, concentrating on the visual communication and aesthetic appeal of a system. It encompasses the visual elements of a system, including typography, colour schemes, and more, with font size and font style emerging as highly important components. Research by Weinschenk (2011) provided insights into the role of fonts in evoking a mood, brand, or association, where different font families were found to invoke a sense of time period (old fashioned versus modern), while others convey seriousness or playfulness.

When it comes to readability, the choice of fonts matters less, as long as they are not overly decorative, making it hard to identify the letters. Font size, however, is rather crucial, as Weinschenk (2011) found out, the font size should be large enough to be read without straining the eyes as people of all ages tend to complain about small font sizes. Figure 2.1 illustrates this, highlighting how a larger font size significantly enhances readability, contrasting with smaller sizes or highly decorative fonts.

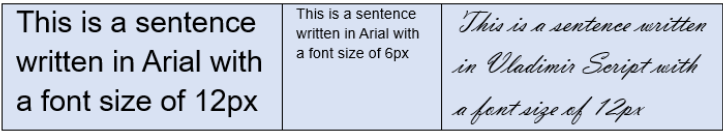


Figure 2.1: A demonstration of how font size and font style can affect the readability.

The colour scheme is another crucial element in visual design. Weinschenk (2011) found that colour can influence what people see by drawing attention to certain parts of the screen. The research further highlighted how combining colours that do not go well together can cause eye strain and make the design look unprofessional. In addition to this, Weinschenk (2011) emphasised the highly personal nature of colour in design, as subtle changes can evoke different emotions. To demonstrate the points mentioned above, Figure 2.2 shows how colour can be strategically used to draw attention to certain parts of the screen. Meanwhile, Figure 2.3 illustrates the potential of different colors to evoke diverse emotional responses.

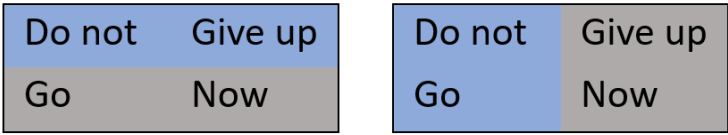


Figure 2.2: A demonstration of how colour can be used to draw attention to certain parts of the screen.



Figure 2.3: A demonstration of how different colours can be used to evoke different feelings and incompatibility of colours.

It is worth noting how the background colours of the two rows in Figure 2.3 evoke contrasting feelings. The first row gives a feeling of gloom and mystery, while the second radiates happiness and joy. Additionally, the compatibility or contrast between font and background colors significantly impacts readability, as evident in the second row's harmonious combination and the first row's poor compatibility, making the text hard to read.

2.3.3 Accessibility

Accessibility holds a significant role in UI/UX design, ensuring that the system is accessible to as many people as possible, but why is it often overlooked? Lazar et al. (2004) conducted a survey revealing that while most webmasters supported the idea of accessibility, they did not implement it because of too many perceived roadblocks, such as time, software tools, confusing guidelines, lack of managerial and client support, and more. Filipe et al. (2023) countered these hesitations, claiming that accessibility is here to stay, becoming increasingly manageable and easier to implement with each version of Web Content Accessibility Guidelines (WCAG). This claim aligns with the W3C's commitment to making the web more accessible for everyone, releasing new comprehensive guidelines, and making it easier to implement accessibility than ever before.

The W3C's latest Web Content Accessibility Guidelines (WCAG) 2.2 covers a wide range of recommendations to ensure web content's accessibility to individuals with disabilities, such as visual, auditory, learning disabilities, poor literacy, and more (W3C 2023). Moreover, it emphasises improving content accessibility and usability for all users in general. W3C also provides a checklist for website owners, which should be considered when designing a website to make it more accessible for less abled users and ensure that the content is perceivable, operable, understandable, and robust for users of all abilities (W3C 2021). The University of Sheffield underscores this commitment on their website, pledging to make their website accessible to all (sheffield.ac.uk/accessibility 2023).

Legislation in the UK, namely the Equality Act 2010, mandates developers to prioritise accessibility when creating websites. According to the act, it is illegal to discriminate against people with disabilities when providing goods, facilities and services (legislation.gov.uk 2010).

While the act does not explicitly mention websites or apps, it deems them as public services, as mentioned by government's Equality and Human Rights Commission (equalityhumanrights.com 2011). Thus, incorporating accessibility in our project not only aligns with legal obligations but also ensures inclusivity, broadening our project's reach to a wider and diverse audience.

2.4 Software Architectural Patterns

Software architectural patterns provide a blueprint for the structure of a software system, guiding the design and development process. They help in organising the system, and offer solutions to recurring design and architectural problems (Lin 2019). Choosing the right architectural pattern is crucial for ensuring that it is scalable, maintainable, and robust. This section explores some of the most popular software architectural patterns that can be used to design our project such as MVC, MVP, and MVVM along with their applications in software development.

2.4.1 Model-View-Controller (MVC)

Model-View-Controller (MVC) is one of the most widely used architectural patterns in software development. It divides the system into three interconnected components: the model, the view, and the controller (Lou 2016). The model represents the data and business logic of the system, the view represents the user interface, and the controller acts as an intermediary between the model and the view, handling user input and updating the model and view accordingly. MVC is known for its simplicity, modularity, and reusability, making it an ideal choice for designing our project. By separating the concerns of the system into three distinct components, MVC helps in improving the maintainability and scalability of the system, making it easier to manage and extend.

MVC is designed to bridge the gap between human mental models and software models by providing developers with the ability to directly manipulate the model and view, and to control the flow of the application (Lin 2019). MVC is widely used in web development, desktop applications, and mobile applications, making it a versatile and flexible architectural pattern that can be adapted to various types of systems. By following the MVC pattern, we can design a system that is well-structured, easy to maintain, and scalable, ensuring the success of our project.

2.4.2 Model-View-Presenter (MVP)

The MVP architectural pattern is an evolution of the MVC pattern. In MVP, the presenter acts as a loose controller, handling UI business logic of the view (Lin 2019). Lou (2016) highlighted the key differences between MVC and MVP, noting that in MVP, the flow begins with the view component, which captures user input events and forwards them to the presenter. The presenter then makes decisions based on simple events or requests data from

the model for complex actions. After processing, the presenter updates the view accordingly. If user input affects the model data, the view does not directly modify it; instead, it notifies the presenter to update the model.

2.4.3 Model-View-ViewModel (MVVM)

MVVM, based on the Presentation Model pattern, combines all the view's state and behavior into a single entity known as the view model, which coordinates with the model and serves as an interface to the view (Lin 2019). MVVM architecture consists of three parts: Model, representing data; View, displaying the application's user interface; and ViewModel, which stands for a model of view, intended to manage the state of the view, pass data and operations to the view, and manage the view's logic and behavior (Lou 2016).

2.4.4 Summary

Software architectural patterns play a crucial role in the design and development of software systems, providing a blueprint for the structure of the system and guiding the development process. The MVC, MVP, and MVVM architectural patterns are widely used in software development, offering solutions to recurring design and architectural problems. By choosing the right architectural pattern, we can ensure that our project is well-structured, maintainable, and scalable, ultimately leading to the success of the project. After reviewing the literature on the three architectural patterns, it can be concluded that the MVC pattern is the most suitable for our project.

As Lin (2019) pointed out, choosing MVC over MVP and MVVM can offer several advantages, particularly in building stable and reliable web applications with a high degree of layering. MVC, as a classic web framework model, is well-known for its distinct layered architecture, which includes the business layer, logic layer, and DAO layer. This layered architecture promotes highly reusable benefits and facilitates the construction of large applications. Moreover, compared to MVP and MVVM, MVC is often considered more robust and flexible. Its clear layering allows for easier maintenance and scalability of applications. Hence, for projects where stability, reliability, and scalability are top priorities, MVC is the preferred choice.

2.5 Testing and Evaluation: Assessing the System's Effectiveness

Testing and Evaluation is a crucial part in the software development process, helping us to determine whether the system is working as intended or not. It not only helps measure the effectiveness but also ensures that the system meets the usability standards. Unfortunately, evaluation is often neglected by many organisations in their development processes, and many developers test systems only after failures or serious complications (Zahran et al. 2014). This negligence poses substantial risks to project success, as it can lead to many problems at the

later stages. This section delves into existing testing and evaluation techniques that can be used to evaluate our project and measure its effectiveness.

2.5.1 Testing techniques

One of the most common testing techniques is manual testing, where testers manually execute test cases without using any automation tools. Although it is a simple and cost-effective method, manual testing can be quite laborious, time-consuming, and error-prone (Jamil et al. 2016). Hence, given the complexity and scale of our project, relying solely on manual testing might not be the best choice. To address the issues with manual testing, the two most popular testing techniques, automated testing and test-driven development (TDD), can be researched and considered for our project.

Automated Testing

Automated testing is a technique that involves using automation tools to execute test cases, compare actual outcomes with expected outcomes, and generate detailed test reports (Jamil et al. 2016). Automated testing is faster, more reliable, and more efficient than manual testing, making it an ideal choice for our project. Automated testing can be used to test various aspects of the system, including functionality, performance, and security. By automating the testing process, we can identify bugs and issues early in the development cycle, reducing the time and effort required for testing and ensuring the quality of the system.

TDD (Test Driven Development)

TDD is a software development technique that involves writing test cases before writing the actual code (Jamil et al. 2016). The process consists of three steps: writing a failing test case, writing the code to pass the test, and refactoring the code to improve its quality. TDD helps ensure that the code is working as intended, reduces the number of bugs, and improves the overall quality of the system. By following the TDD approach, we can develop a robust and reliable system that meets the requirements and expectations of users and save a great amount of time that might get wasted later over the debugging process (Jamil et al. 2016).

2.5.2 Evaluation techniques

Evaluating the effectiveness of a system is crucial to ensure that it meets the requirements and expectations of users. There are several evaluation techniques that can be used to assess the effectiveness of our project, including web metrics, usability evaluation methods, and more.

Web Metrics

Web metrics emerge as an important technique in website evaluation, providing a quantitative measure of the effectiveness of a website. Hong (2007) noted in a survey of independent

organisations in Korea that website metrics stand as a key enabler for measuring website success. These metrics play a crucial role in determining if a site performs in alignment with user and business expectations, while also identifying potential website design problems. Stolz (2005) contributed to this by introducing a web metrics model that can be used to assess the success of the information-driven websites by utilising user feedback, analysing behaviour patterns, and visited content.

Usability Evaluation Methods

Usability evaluation methods present another powerful technique for evaluating our project. Larusdottir (2009) conducted an in-depth investigation into two usability evaluation methods: one involving user participation and the other centered around expert analysis.

In the “Evaluation through User Participation” method, users actively take part in the evaluation process. Users are asked to perform predefined tasks in controlled settings, and their performance is measured based on factors such as task-solving time, error frequency, and navigation patterns. To observe user behaviour, observation methods are available, with the most popular being the think-aloud method, first introduced by Lewis (1982). In a think-aloud session, user is asked to solve predefined tasks and talk while interacting with the system, helping the evaluator understand users’ thoughts about the system as they interact with it.

On the other hand, the “Evaluation through Expert Analysis” method involves experts in the evaluation process. Experts assess the system based on their knowledge and experience, inspecting the interface and predicting the impact on users. The experts could use interface guidelines, user interface standards, the users’ tasks, or their own knowledge to identify potential problems users might encounter while interacting with the system. Heuristic evaluation, first introduced by Nielsen & Molich (1990), is a widely used expert analysis method where a group of evaluators individually examine the system and then collaborate to discuss and compile a list of usability problems based on their findings.

Cognitive walkthrough, another evaluation method where the evaluator simulate the user’s thought process while interacting with the system, during which they identify potential problems users might encounter, can be used to evaluate the system (Paz & Pow-Sang 2016). More methods such as evaluation through user feedback using surveys or questionnaires to assess user satisfaction (Larusdottir 2009), as well as comparing our project with existing solutions can also be considered to evaluate the effectiveness of our project. The method chosen for evaluation must be determined by further analysis of the project requirements and the most suitable evaluation method for our project can be chosen accordingly.

Chapter 3

Requirements and Analysis

This chapter outlines the objectives and requirements of the project. These requirements and implementation decisions are based on the findings of the previous chapters, feedback from a potential user, and the analysis of the existing system. The chapter further discusses risks involved in the project and how they will be mitigated. Finally, this chapter will outline the evaluation and testing strategies that will be used to check the suitability of the final product, along with ethical, professional, and legal considerations.

3.1 Project Requirements

This section outlines the functional and non-functional requirements for the project. Each requirement is assigned weighting using the prioritisation technique called the MoSCoW method (Consortium 2023). The weighting in Tables 3.1, Table 3.2, Table 3.3, and Table 3.5 is as follows:

- **Must Have (M)** - These requirements are vital to the success of the project and must be included in the final product.
- **Should Have (S)** - These requirements are important but not vital to the success of the project.
- **Could Have (C)** - These requirements are desirable to improve the system however, they are optional and will only be implemented if time permits.

3.1.1 Functional Requirements

Functional requirements consist of the actions that the system must be able to perform. These requirements are displayed in form of user stories. User stories are a way of breaking down the requirements into smaller, more manageable tasks. They are written from the perspective of the user and are used to define the functionality of the system. The format of agile user stories according to Agile-Modeling (2023) is as follows:

As a type of user, I can perform an action so that I gain some kind of value.

In this project, users are categorised into three distinct types: Browsers, Posters, and Admins. Browsers comprise the students and staff of the university actively seeking opportunities within the system. Posters, on the other hand, consist of university staff or external individuals who create and manage posts for events or opportunities. Lastly, Administrators represent university staff members responsible for the overall management of the system, including users and posts.

To gain deeper insights into the system requirements, an interview was conducted with a university staff member, who also serves as a user of the existing system. Drawing from their experience with the existing system, the interview aimed to understand their needs, expectations, and desired features for the new system. Based on their responses, user stories were crafted to guide the development process.

The participant's feedback highlighted several critical shortcomings of the current opportunity posting system. They expressed frustration with the manual processes involved in creating and managing posts, advocating for a more supportive and automated system that can "support the user". They underscored the need for automation in the creation process, citing the laborious task of manually transferring and formatting information from websites to the department webpage. They described the current system as "tedious" and emphasised the importance of automation to enhance efficiency and streamline operations.

The participant further emphasised the need for a system that allows them to input information directly, enabling automatic formatting and display instead of manually dragging the information to its appropriate place. They also found the single Google webpage interface cumbersome and the manual arrangement of opportunities to the correct sections time-consuming. The participant raised concerns about the lack of automation in handling passed opportunities, suggesting automatic flipping of passed opportunities from the upcoming/ongoing section to the recent section to ensure timely updates. Automating the archiving of opportunities after one year of their passing was further recommended to streamline the process. The absence of functionality to track recurring opportunities or events was also identified as a significant limitation.

Communication emerged as a key area for improvement, with the participant highlighting the inefficiency of the current emailing process, which requires manual drafting and sending of emails to all students and staff. They expressed a desire for automated email draft generation that could be reviewed and copied to the clipboard for sending, thus saving time and enhancing efficiency.

This feedback and analysis recognised automation of the manual processes outlined by the participant as the main priority. Additional features such as introducing tags for opportunities, allowing users to subscribe to tags, and receive notifications based on their preferences were identified. Introducing a save as favourite feature for users to save and track opportunities of interest was also deemed necessary. Improvements in handling recurring opportunities, streamlining email communication, and browsing of opportunities emerged as key areas for enhancements.

This analysis served as a valuable guide for prioritising features and refining the system to better meet user needs. Based on this, user stories were created for Browsers, Posters, and Admins, as shown in Tables 3.1, 3.2, and 3.3. The user stories are categorised based on the priority assigned to them using the MoSCoW method.

ID	User Stories	MoSCoW
1.1	As a Browser, I can create an account in the system	M
1.2	..., I can login or logout to the system	M
1.3	..., I can request a password reset if forgotten	M
1.4	..., I can view all recent, ongoing, upcoming or recurring posts	M
1.5	..., I can subscribe to tags to receive notifications when a post is created with that tag	M
1.6	As a Poster, I can create a poster account in the system	M
1.7	..., I can login or logout to the posting hub	M
1.8	..., I can request a password reset if forgotten	M
1.9	..., I can create a post for an event or opportunity	M
1.10	..., I can assign tags to my posts	M
1.11	As an Admin, I can login or logout to the admin dashboard	M
1.12	..., I can request a password reset if forgotten	M
1.13	..., I can view all recent, ongoing, upcoming or recurring posts	M
1.14	..., I can approve or reject newly created pending posts	M

Table 3.1: Must Have User Stories

ID	User Stories	MoSCoW
2.1	As a Browser, I can unsubscribe to tags to stop receiving notifications	S
2.2	..., I can search for posts using a search bar	S
2.3	..., I can update my profile including email and password	S
2.4	As a Poster, I can view recent posts	S
2.5	..., I can edit or delete upcoming posts created by me	S
2.6	As an Admin, I can view all users and their limited details	S
2.7	..., I can edit user roles and permissions	S
2.8	..., I can create, edit or delete posts	S

Table 3.2: Should Have User Stories

ID	User Stories	MoSCoW
3.1	As a Browser, I can suspend notifications for a specific period of time	C
3.2	..., I can save or unsave my favourite posts to view and track them later	C
3.3	As a Poster, I can use web scraping techniques to extract content from a website when creating a post	C
3.4	..., I can view the history of posts I created	C
3.5	As an Admin, I can view the history of posts I created	C
3.6	..., I can select the newly approved posts to generate an email draft to be sent to users via email	C

Table 3.3: Could Have User Stories

Note: There are no Would Have stories as they were deemed unnecessary and out of scope for the project.

User Type	No. of Must Have Stories	No. of Should Have Stories	No. of Could Have Stories
Browser	5	3	2
Poster	5	2	2
Admin	4	3	2

Table 3.4: User Stories Summary

3.1.2 Non-Functional Requirements

Non-functional requirements are the constraints that the system must adhere to. It consists of the quality attributes of the system as well as the best practices in software engineering. The requirements are listed in Table 3.5.

ID	Requirements	MoSCoW
1	The system must be able to run on various devices of different screen sizes	M
2	All code for the creation of the system must be well tested and run without errors	M
3	All code for the creation of the system must be well organised and documented	S
4	The simplicity and usability of the system should be considered in the design	S
5	The system should abide by the Data Protection Act	S
6	The accessibility standards of the system should be considered	S
7	The security of the system should be considered	S

Table 3.5: Non-Functional Requirements

3.2 Risk Analysis

This section consists of potential risks involved in the project and how they can be mitigated, as shown in Table 3.6.

Risk	Mitigation
Unable to complete the project on time, specifically the must have requirements	Break down the project into smaller tasks, allocate sufficient time, and utilise Gantt chart. Consult the supervisor regularly to ensure the project is on track
Insufficient knowledge of the technologies used	Research and test the technologies before using them in the project
Insufficient time for testing	Allocate sufficient time specifically for testing
Loss of critical data or code	Use version control system such as Git and backup the data regularly
Security vulnerabilities	Use secure coding practices, stay informed about potential threats
Users not finding the system useful or user-friendly	Get feedback from the users and make changes accordingly

Table 3.6: Risks and Mitigation Strategies

3.3 Testing and Evaluation

This section outlines the testing and evaluation strategies that will be employed to ensure the robustness, reliability, and performance of the developed system. The testing of the system will follow a comprehensive and systematic approach, primarily employing a combination of automated and manual testing. Automated testing involves using software tools to compare the expected outcomes of the software with its actual outcomes. Whereas manual testing involves human testers who evaluate the system's functionality and performance. The testing process will comprise several stages, each serving a distinct purpose. These stages include unit testing, and integration testing as the automated testing techniques, and system testing and usability testing as the manual testing techniques.

3.3.1 Unit Testing

Unit testing is an automated testing technique that involves testing individual units or components of the system. The aim is to verify that each unit operates as intended.

3.3.2 Integration Testing

Integration testing is an automated testing technique that evaluates the integration of individual units or components within the system. The goal is to ensure that these integrated units function as expected when combined. Integration tests (also known as assembly tests or union tests) are a logical extension of unit tests.

3.3.3 System Testing

System testing is a manual testing technique that assesses the entire system as a whole. Its objective is to confirm that the system meets the specified requirements and operates as anticipated. This testing method purely focuses on the system's functionality and performance.

3.3.4 Usability Testing

Usability testing is a manual testing technique that evaluates the system's user interface and user experience. The goal is to ensure that the system is user-friendly, and meets the requirements and expectations of the end-users. This testing method focuses on the system's usability, simplicity, and efficiency. Usability testing will be conducted through cognitive walkthroughs, allowing the developer to examine the system through the eyes of end-users.

3.3.5 User Acceptance Testing

User acceptance testing (UAT) is a manual testing technique that involves end-users testing the system themselves. The goal is to ensure that the system meets the users' needs and expectations, and that they accept the system as fit for purpose. Due to the limited timeframe

of the project, UAT with real users will not be conducted at this stage. However, It will be conducted in the future.

3.4 Ethical, Professional and Legal Issues

The system will be developed in accordance with ethical, professional and legal standards. The developer will adhere to the Data Protection Act (gov.uk 2023), ensuring the system's security and safeguarding user privacy. To address privacy concerns, the system will only collect and store minimal user data necessary for identification and authentication purposes. Secure coding practices and encryption techniques will be employed to fortify the system's security and protect user data. Additionally, the system design will prioritise responsiveness across various devices, enhancing accessibility for all users. The system will be developed following best practices in software engineering and will adhere to ethical, professional, and legal standards.

Chapter 4

Design

This chapter discusses the design of the system. It outlines the technology stack choices for the development of the system, and provides an overview of the design components. Finally, it discusses the database design and user flow diagrams for more detailed understanding of the system's functionality.

4.1 Technology Stack Choices

During the design phase, we recognised that consolidating the system into a single web application, rather than separating it into two distinct applications, a browser-based posting hub for university staff or external posters, and a mobile application for students and staff, would be more efficient. This would allow for a more streamlined development process and a more intuitive user experience. The system would be designed to be responsive, allowing it to be used on both desktop and mobile devices, making it accessible to a wider range of users.

For the development of this web application, Ruby on Rails was chosen as the primary framework due to its robustness, scalability, and rapid development capabilities, which works well with our chosen MVC (Model-View-Controller) architecture. Ruby on Rails follows the convention over configuration principle, which allows for simple and faster development by reducing the need for developers to write boilerplate code. Additionally, It provides a wide range of built-in tools and libraries for common web development tasks, such as database management, routing, and authentication. This would allow for a more efficient development process, as developer would only need to focus on the business logic of the application.

In Figure 4.1 provided by TK (2018), the architecture of Ruby on Rails with MVC pattern is explained in detail.

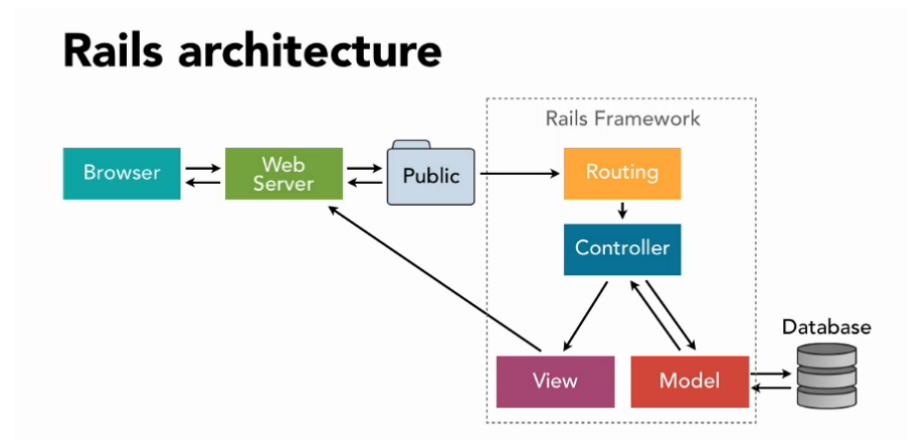


Figure 4.1: Ruby on Rails Architecture with MVC Pattern

PostgreSQL was chosen to manage the database for the system for its reliability, scalability, and support for complex data structures and queries. Bootstrap was chosen as the front-end framework due to its ease of use, responsiveness, and wide range of pre-built components and styles. Bootstrap would allow for a more consistent and visually appealing user interface design. Finally, RSpec testing framework was selected to ensure the reliability and maintainability of the codebase through automated testing, enabling developers to write expressive and readable tests to verify the functionality of the system.

Scrum, a popular agile methodology, was chosen as the project management methodology for the development of the system. Even with a single developer, Scrum allows for iterative and incremental development, which would enable the system to be adaptive to changing requirements and priorities. This would allow for a more efficient and effective development process, as well as a more reliable and maintainable codebase.

4.2 Overview of the Design

The design for the system was divided into five main components, each of which was essential for the functionality of the system. This section discusses these components, derived from the requirements outlined in the previous chapter.

4.2.1 User Registration, Authentication and Editing

There were two main possible solutions for user registration and authentication. The first was to use the built-in authentication system provided by the Devise gem, a popular authentication solution for Rails applications. Devise offers a wide range of comprehensive features for user registration, authentication, and password recovery. On creating an account, user details would be stored securely in the database, and users would be able to log in using their email address and password. Passwords would be encrypted using robust algorithms to ensure

data security. During login, password decryption would occur by comparing the stored hash with the entered password. If the password matches, the user would be authenticated and granted access to the system. Using this solution would provide a more secure and reliable authentication system, and would also allow for a straightforward implementation process.

The second solution was to use the OAuth2 protocol with the Omniauth gem, which would allow users to log in using their Google accounts. This would provide a more seamless and user-friendly experience for users, as they would not need to create a new account. It would also provide a more secure authentication system, as the user's credentials would be stored securely by Google. After careful consideration, It was decided that both solutions would be implemented, allowing users to choose between creating an account or logging in with their Google account. This approach would provide users with flexibility as well as a more user-friendly experience, while ensuring accessibility for all users, including those without Google accounts.

User access rights would be determined by their assigned role stored in the database, including User (Browser), Poster, and Admin roles. Upon registration, users would be assigned the User (Browser) role by default, granting access to basic system features such as browsing and tracking opportunities. Those interested in posting opportunities could opt for a Poster account during registration, granting them posting privileges. Admin accounts would be pre-defined and stored in the database, granting access to system administration features and remaining inaccessible for registration.

Furthermore, users would have the capability to edit their profile details, including name, email address, and password, through a user-friendly interface. This feature would enable users to conveniently update their information, ensuring accuracy and reliability. Changes made by users would be stored in the database, and their profiles would be updated accordingly.

4.2.2 Post Creation and Management

Post creation and management would be a core feature of the system, allowing users to post opportunities for other users to browse and track. Users with Poster accounts would have access to the post creation interface, where they could input details such as the title, location, expected start and end dates, time, description, deadline, URL, and tags for better categorisation. Additionally, if the opportunity recurs, they could specify relevant recurring information such as intervals and more. The interface would be user-friendly and intuitive, enabling quick and efficient post creation, with each post being associated with poster's account for reliability.

Upon creation, posts would undergo admin approval before appearing on the platform. Once approved, they would be visible on the Upcoming page for users to browse. The Poster would be able to view recent posts to get an overview of the opportunities that have been

posted in the past. If needed, they could edit or delete their posts, with edits requiring admin re-approval. This provides posters with control and ensures information accuracy. Posters could also track their post history, facilitating transparency and accountability.

4.2.3 Browsing and Tracking Opportunities

Users (Browsers) would have access to various features for browsing opportunities within the system. They could navigate upcoming/ongoing opportunities on the Upcoming page, recently passed posts on the Recent page, and long-expired posts on the Archives page. Filtering options based on tags and dates would be available, enabling users to find opportunities that match their preferences. Additionally, a search bar would be provided for users to search for specific posts, enhancing the browsing experience. Each post would have a user-friendly and intuitive view, displaying essential information and highlighting whether the post is recurring, available, ongoing, or expired. Posts could also be opened in a more detailed view, providing comprehensive information.

Tracking opportunities would allow users to express interest in posts. Users could track posts by saving them as favorites, enabling them to keep track of posts they are interested in and view them later on the Saved page. Users could also remove posts from their favorites or saved lists, providing flexibility and control over their favorite opportunities.

4.2.4 Personalised Notifications

Personalised notifications would be a key feature of the system, providing Users (Browsers) with real-time updates when new opportunities are posted that match their preferences. Users could set their notification preferences by editing their profile and selecting tags of interest. When a new post is created with matching tags, users would receive a notification, indicated by a highlighted icon on the navigation bar with a notification count. Clicking on the icon would display a list of notifications, allowing users to view new opportunities. This feature ensures users are informed of relevant opportunities in real-time, enhancing their personalised and engaging experience.

4.2.5 Administration

Admin accounts would have access to a range of system administration features, enabling them to manage users and posts effectively. Admins would be able to view all users through the Users page, providing an overview of the user base. They could also edit user roles, granting or revoking posting or admin privileges as needed. Admins would approve or reject posts through the Pending page, ensuring the accuracy and reliability of platform information. They could also view post details to make informed decisions and edit or delete posts, maintaining control over the system's content.

Admins would also have access to most abilities that User (Browser) and Poster have, allowing them to post, browse, and track opportunities like any other user. This would provide admins with a more comprehensive understanding of the system’s functionality to ensure effective management. Lastly, Admin would have access to Email page, where they could select a number of newly posted opportunities and automatically generate an email draft which could be edited or copied to the clipboard for efficient emailing.

4.3 Database

The database design for the system was essential for its functionality and reliability. It stored all user and post data, ensuring seamless operation. The database design consisted of two main tables: Users and Posts, each critical to the system’s functionality. The database diagram in Figure 4.2 provides an overview of the design, including the tables, columns, and their data types.

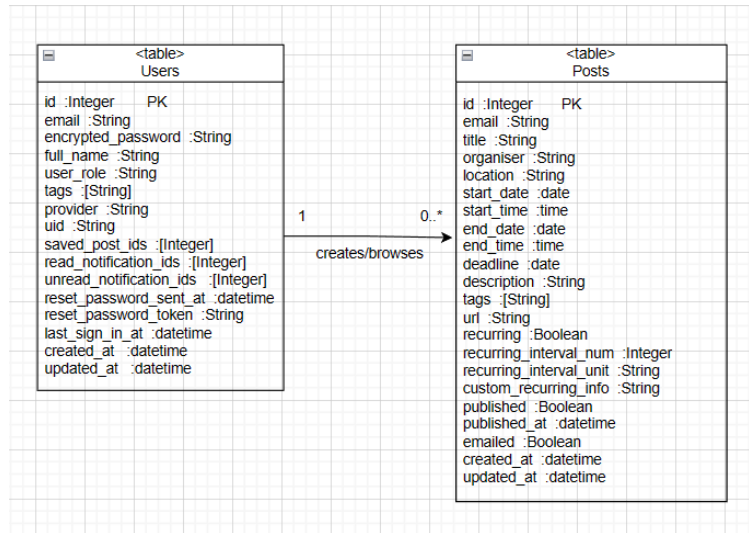


Figure 4.2: Database Diagram

The Users table would store user data required for registration, authentication, and access control. *user_role* column would store the role or type of the user, it would be either ‘0’ for User (Browser), ‘1’ for Poster, or ‘2’ for Admin. The table would include *tags* for personalised notifications, allowing users to select their preferences. The *read_notification_ids* and *unread_notification_ids* columns would store the IDs of notified posts that user has seen and not seen yet, respectively. Lastly, *saved_post_ids* would store the IDs of posts that user has saved as favourites. Other columns present in the table are there for additional information and functionality.

The Posts table would store all the details regarding the created post including the *email* of the poster. The table would also include *tags* for better categorisation and *recurring*

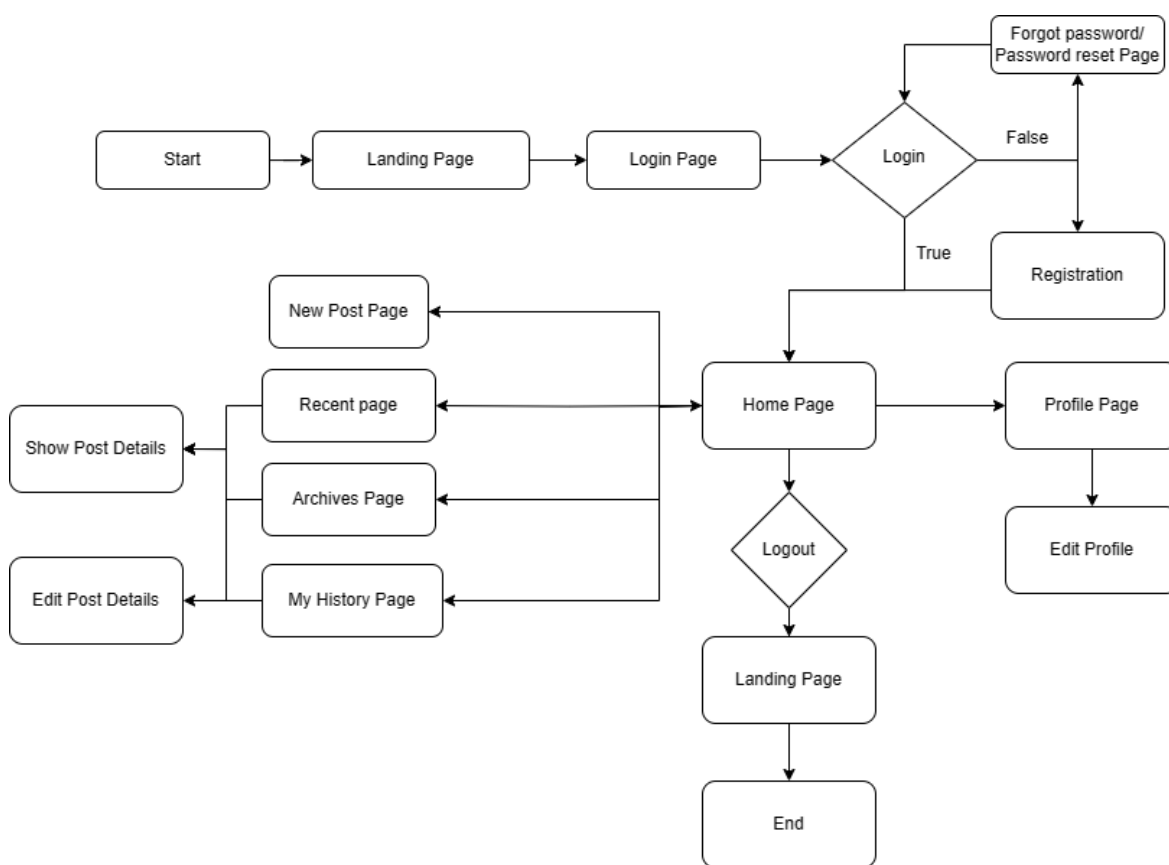


Figure 4.4: User Flow Diagram: Poster

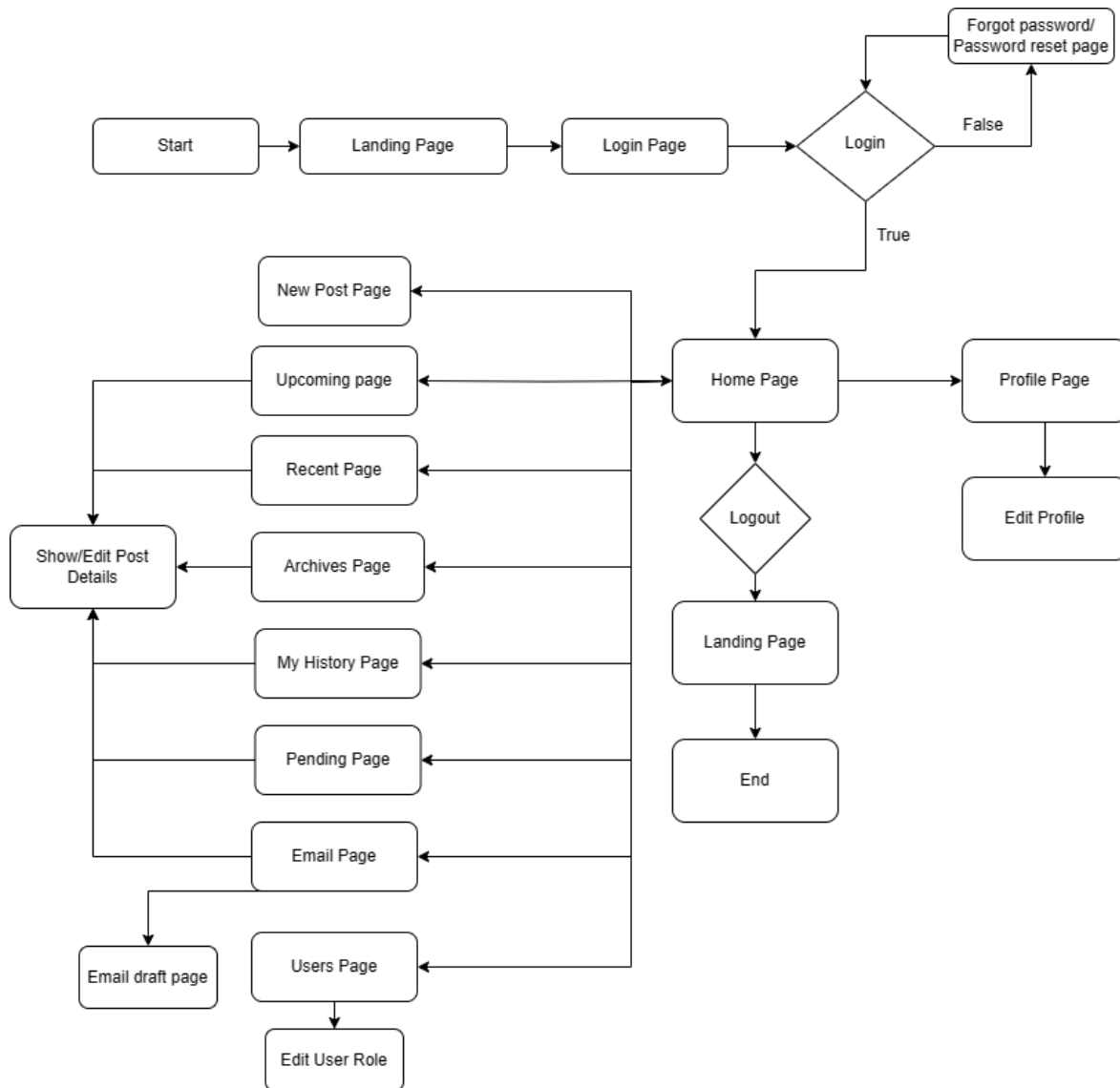


Figure 4.5: User Flow Diagram: Admin

Chapter 5

Implementation

This chapter discusses the implementation of the web application. The structure of the chapter is based on the components discussed in the previous chapter.

The responsive web application was developed using Ruby on Rails with MVC pattern. The back-end was handled by PostgreSQL, while Bootstrap was primarily used for the front-end. Additionally, Agile scrum approach was adopted for the development of the system, though modified for a solo project. ‘Scrum For One’ by Lucidchart (2024) was used as a reference for a tailored development process. Tools, such as kanban board, were used to keep track of the progress. Lastly, risk management plan, and other diagrams as discussed in the previous chapter were used to guide the development.

The implementation of the system was done keeping each user role in mind. Priority was given to the Poster functionalities first, followed by the major functionalities of the Browser and the Admin. The implementation was done in sprint cycles, with each sprint focusing on a particular component of the system. Implementation of the 5 main components of the system, as outlined in the previous chapter, are described below.

5.1 User Registration, Authentication and Editing

This component encompasses the implementation of the Landing page, user registration, authentication, and profile editing. When the web app’s URL was accessed, the Landing page was displayed, providing users with essential information such as the app’s title, basic definition, details about the COM founders club, Slack community, and contact information for support. The Landing page was inspired by the existing COM opportunities hub at the University of Sheffield (COM-Opportunities 2023).

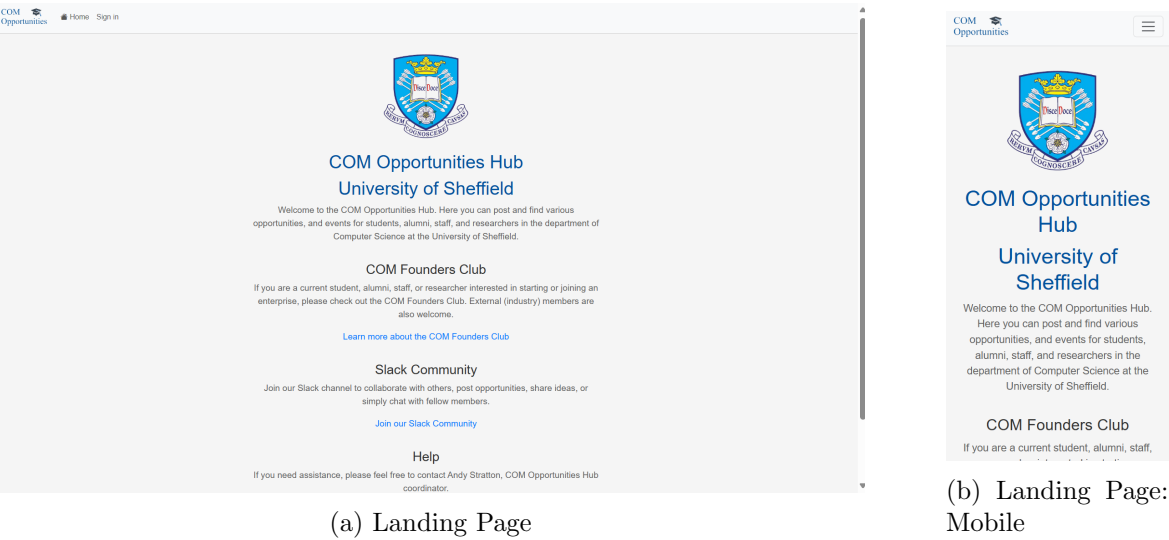


Figure 5.1: Comparison of Landing Page on Desktop and Mobile

Clicking the sign in link on the navigation bar sent request to the server to render the Sign in page. The Sign in page featured a form with fields for email and password. Upon submission, the server checked the database for a user with the provided email. If the user was found, the password was verified. If correct, the user was redirected to the Home page, otherwise, an error message was displayed. Additionally, the Sign-in page includes links for new user registration, password reset, and an option for login using Google account.

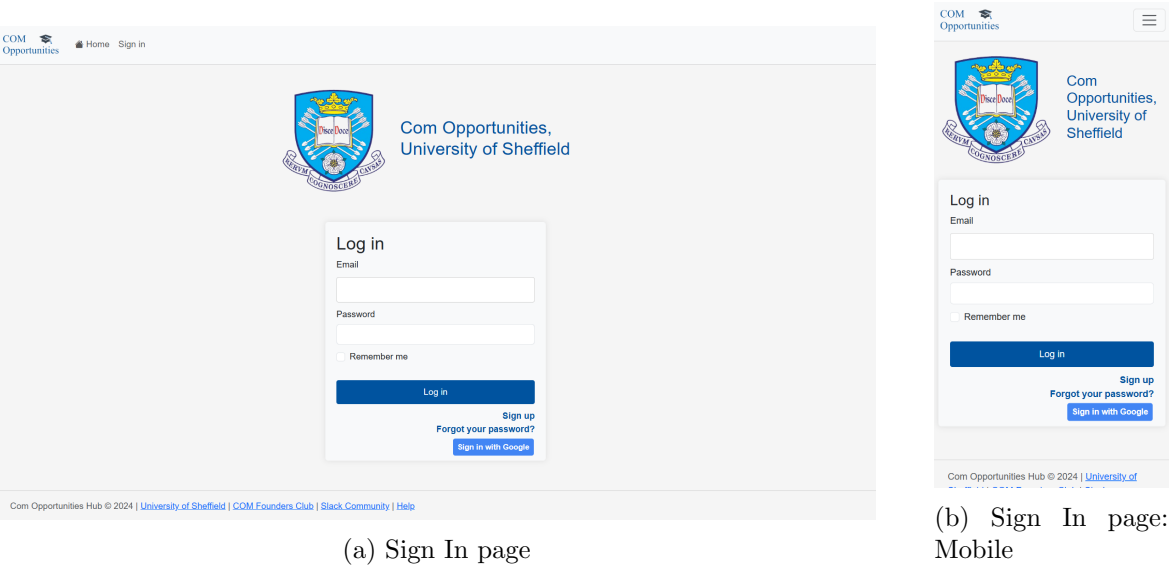


Figure 5.2: Comparison of Sign In Page on Desktop and Mobile

The Sign up page featured a form with fields for email, full name, password, password confirmation, and a checkbox for indicating if the user intended to post opportunities. Upon

submission, the server verified if the email was already in use. If the email was not previously registered and all other fields were valid (e.g., password and password confirmation matched, password met the minimum length requirement, etc.), the user was created, and redirected to the Home page. Type of user was determined based on the checkbox. If checked, Poster account was created, otherwise, User (Browser) account was created. Admin account could not be created through the Sign up page. Additionally, the form included links for sign in page, and login in using Google account.

The Forgot password page comprised a form with email field. After submission, the server checked if the email existed. If registered, an email was sent to the user containing a reset password link. If the email did not exist, error message was displayed.

(a) Sign Up form

(b) Forgot Password form

Figure 5.3: Sign Up and Forgot Password forms

The user registration and authentication process was managed by the Devise gem, utilising bcrypt for password encryption. Although two modes of authentication were planned in the previous chapter, the Google authentication was not fully implemented. omniauth-google-oauth2 gem was employed for Google authentication, but due to errors in the implementation, and time constraints, the implementation remained incomplete. This feature will be implemented in the future.

Once signed in, the user could access their Profile page by clicking the profile link on the navigation bar. The Profile page featured user details, including account creation and update timestamps. Users could edit their profile details using the Edit button, which rendered the Edit Profile form with fields for email, full name, password, and password confirmation. Upon submission, the server checked that the validations were met, and update the user. Afterwards, the user was signed out, and redirected to the Sign in page, where the user could sign in with the new details. The Profile page also included a button to delete the account, which on click, sent a request to the server to delete the user. Upon deletion, the user was removed from the database, and redirected to the Landing page. The user could not sign in unless registered again. Admin account could not be deleted through the Profile page.

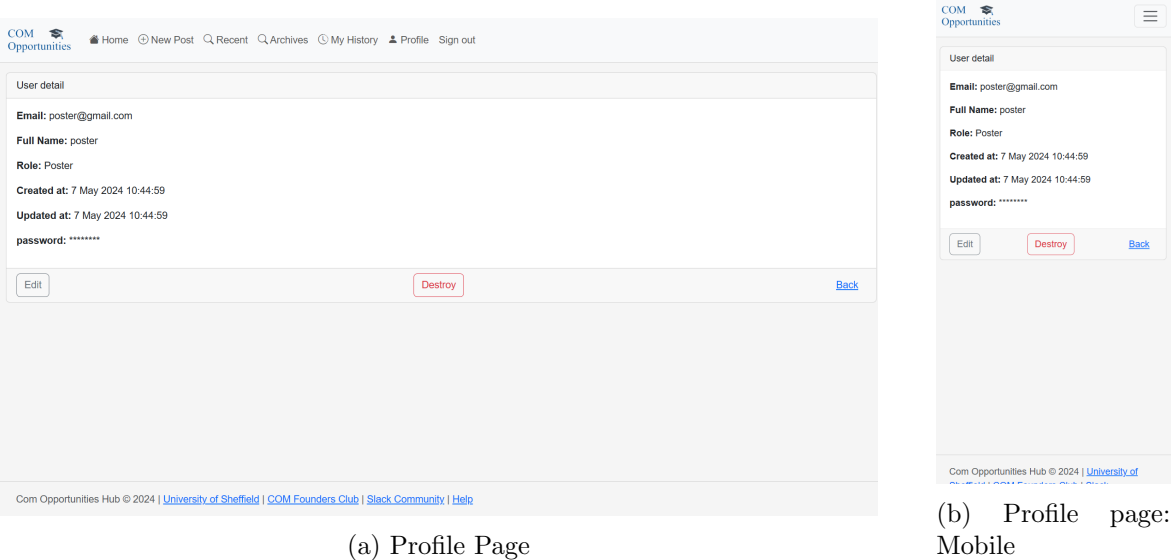


Figure 5.4: Profile Page on Desktop and Mobile (Poster view)

(a) Edit Profile page

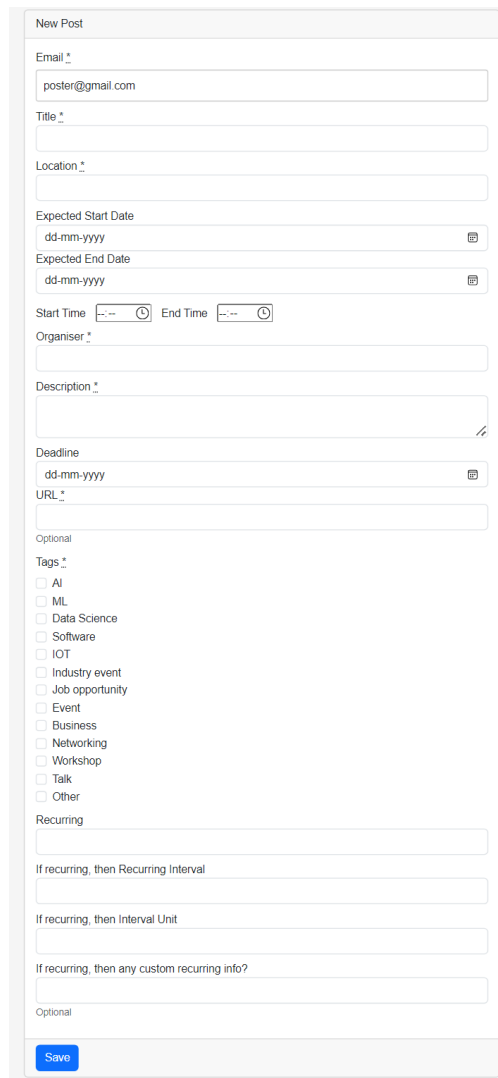
(b) Edit Profile page: Mobile

Figure 5.5: Edit Profile Page on Desktop and Mobile (Poster view)

5.2 Post Creation and Management

This component contains the implementation of the post creation and management functionalities. Users with the role of Poster could create new posts by clicking the New Post link on the navigation bar. The New post page featured a form with various fields for inputting post details, as shown in Figure 5.6. Upon submission, the server validated all fields to ensure they met the required criteria. If any invalid input was detected, appropriate error messages were displayed. Once all fields were valid, the post was created, and the Poster was redirected to the home page.

The post creation and management functionalities were handled by the Post model and the Posts controller. The Post model included the validations for the fields, and other criteria to be checked before creating a post (e.g., the start date should be before the end date, all dates should be in the future, etc.). The Posts controller handled actions related to creating, updating, and deleting posts, as well as displaying posts on different pages (upcoming, recent, archives, saved, my history, searched posts, pending posts, etc.). Additionally, the controller managed actions for approving and rejecting posts, saving and unsaving posts, and handling notifications.



The form is titled "New Post" and contains the following fields and sections:

- Email ***: Text input with "poster@gmail.com" pre-filled.
- Title ***: Text input.
- Location ***: Text input.
- Expected Start Date**: Date picker showing "dd-mm-yyyy".
- Expected End Date**: Date picker showing "dd-mm-yyyy".
- Start Time** and **End Time**: Time pickers showing "--:--".
- Organiser ***: Text input.
- Description ***: Text area with a rich text editor icon.
- Deadline**: Date picker showing "dd-mm-yyyy".
- URL**: Text input.
- Optional** section:
 - Tags ***: A list of checkboxes for categories: AI, ML, Data Science, Software, IOT, Industry event, Job opportunity, Event, Business, Networking, Workshop, Talk, and Other.
 - Recurring**: Text input.
 - If recurring, then Recurring Interval**: Text input.
 - If recurring, then Interval Unit**: Text input.
 - If recurring, then any custom recurring info?**: Text input.
- Optional**: A final text input field.
- Save**: A blue button at the bottom.

Figure 5.6: New Post Form

After creating a post, the poster could view the previously created posts by clicking the My History link on the navigation bar. The My History page displayed all the posts created by the poster, along with the status of the post (awaiting approval, published). Each post card included buttons for editing, deleting, and viewing the post. Clicking the Edit button rendered a form with the post details which the Poster could edit and submit. The post needed re-approval if edited. The Poster could delete the post by clicking the Destroy button, which removed the post from the database. The Show button redirected the Poster to the Post details page, providing comprehensive information about the post.

Figures 5.7 and 5.8 show My History page and the Post details page in the desktop view. Desktop because post creation and management functionalities are more likely to be used on a desktop, however, the system is responsive and works well on mobile as well.

Note: The opportunities shown in the subsequent figures, were real opportunities publicly available online at the time. The opportunities, SIAN (2024), Collective (2024), Tech (2024), and BrainStation (2024), were only used to show the implementation of the system.

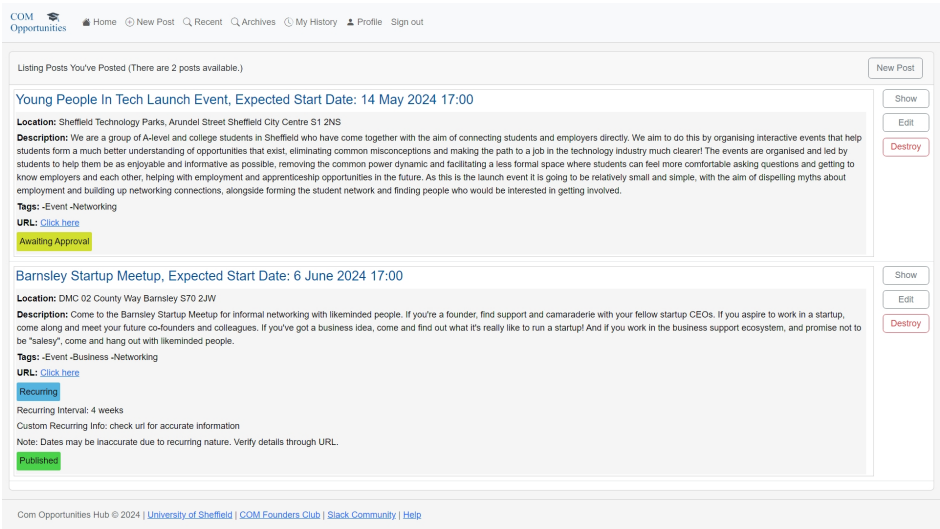


Figure 5.7: My History Page

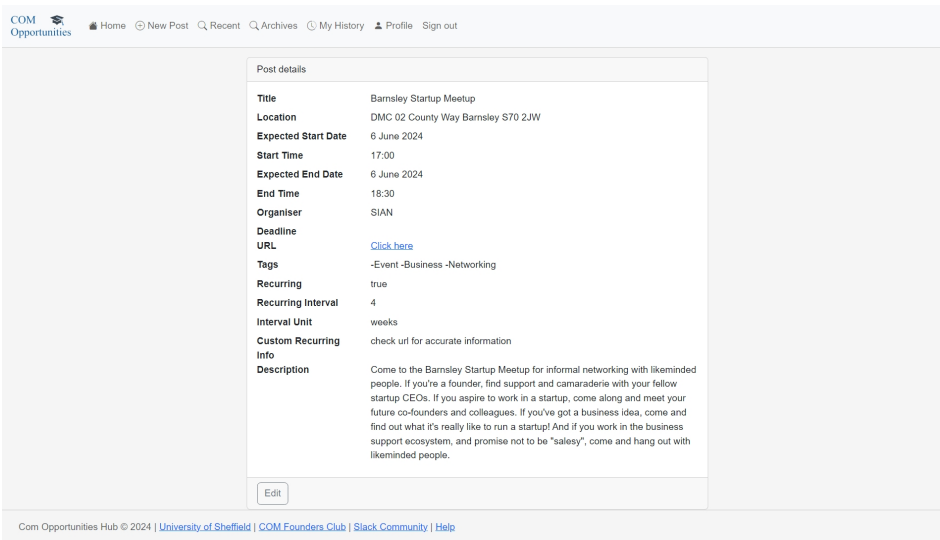


Figure 5.8: Post Details Page

5.3 Browsing and Tracking Opportunities

This component implements the Upcoming, Recent, and Archives pages, along with saving, and searching functionalities. As mentioned in the previous chapter, the User (Browser) could view all upcoming and ongoing opportunities on the Upcoming page, recently expired opportunities on Recent, and long expired opportunities on Archives page. The Upcoming page displayed posts with ongoing first, followed by those closest to start date. The posts were displayed in a card format, with important information and highlighted blocks indicating if the post was recurring, ongoing, or upcoming.

User (Browser) could view details by clicking the Show button on the card. The Post details page displayed all post details, similar to Figure 5.8. The User (Browser) could save posts by clicking the Save button on the card, updating the saved post IDs in the user's database. Saved posts were displayed on the Saved page. The Save button changed to Unsave once clicked for a particular post. User (Browser) could search for posts by title using the search bar. The server would search the database for posts with the title containing the searched query, and display the results on the Search results page.

Figures 5.9, 5.10, and 5.11 show the Upcoming page, Saved page, and Search functionality.

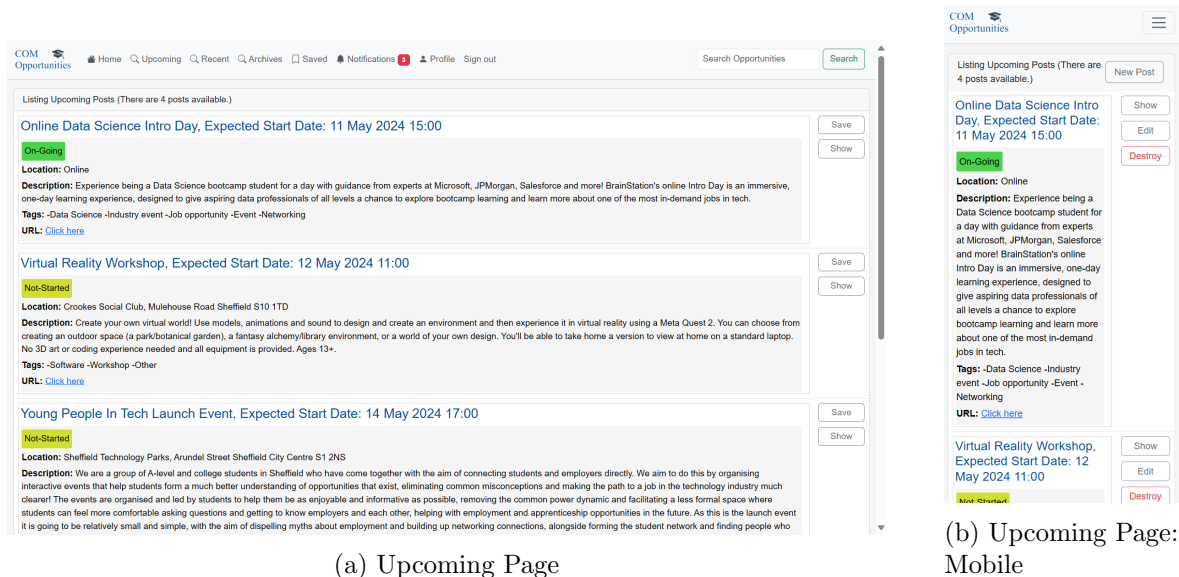
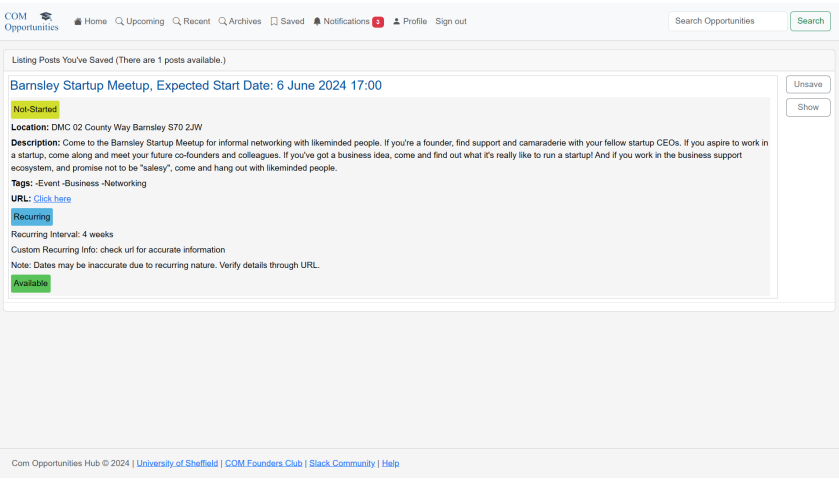
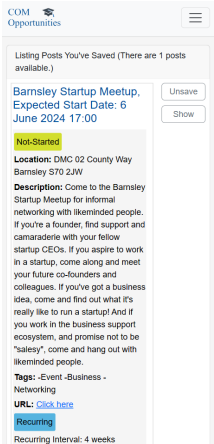


Figure 5.9: Upcoming Page on Desktop and Mobile



(a) Saved Page



(b) Saved Page: Mobile

Figure 5.10: Saved Page on Desktop and Mobile

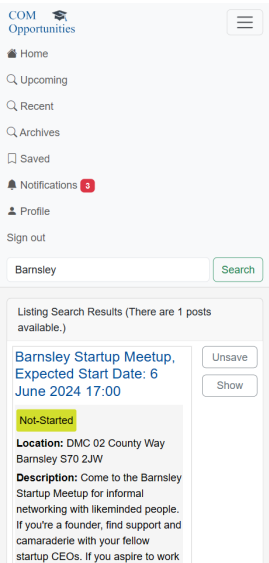


Figure 5.11: Search Results Page: Mobile

5.4 Personalised Notifications

The personalised notifications component implements the notifications feature. After signing up, Users (Browsers) could edit their profile to select tags of interest. Upon form submission, the server updated these tags in the user's database, enabling notifications for posts with matching tags.

When a new post was created, the server compared its tags with users’ tags of interest. On finding a match, the server updated unread and read notification ids in the user’s database. Users could view notifications by clicking the notification icon, which displayed a badge with the number of unread notifications. Notifications were listed with the most recent at the top and included the title and expected start date. Highlighted blocks indicated if the post was Available, Not-Available, or Ongoing. Once read, the server updated the unread notification IDs to empty while storing read notification IDs for future reference.

Figures 5.12, 5.13a, and 5.13b illustrate Edit Profile functionalities, the notification navigation tab, and the Notifications page in mobile view, though the pages are responsive for desktop use as well.

Editing Profile

Email
user@gmail.com

Full name
user

Password

Type a new password if you want to change it, otherwise type the current password.

Password confirmation

Type the password again to confirm.

Tags

- ☐ AI
- ☐ ML
- ☐ Data Science
- ☒ Software
- ☒ IOT
- ☒ Industry event
- ☒ Job opportunity
- ☒ Event
- ☒ Business
- ☐ Networking
- ☐ Workshop
- ☐ Talk
- ☐ Other

Update Back

(a) Edit Profile Form for Editing Tags

COM Opportunities

User detail

Email: user@gmail.com

Full Name: user

Role: User

Created at: 6 May 2024 12:42:44

Updated at: 7 May 2024 15:23:48

password: *****

tags: Software -IOT -Industry event -Job opportunity -Event -Business

Edit Destroy Back

Com Opportunities Hub © 2024 | [University of Sheffield](#) | [COM Founders Club](#) | [Slack Community](#) | [Help](#)

(b) Profile Page with Selected Tags

Figure 5.12: Selecting Tags on Profile Page: Mobile

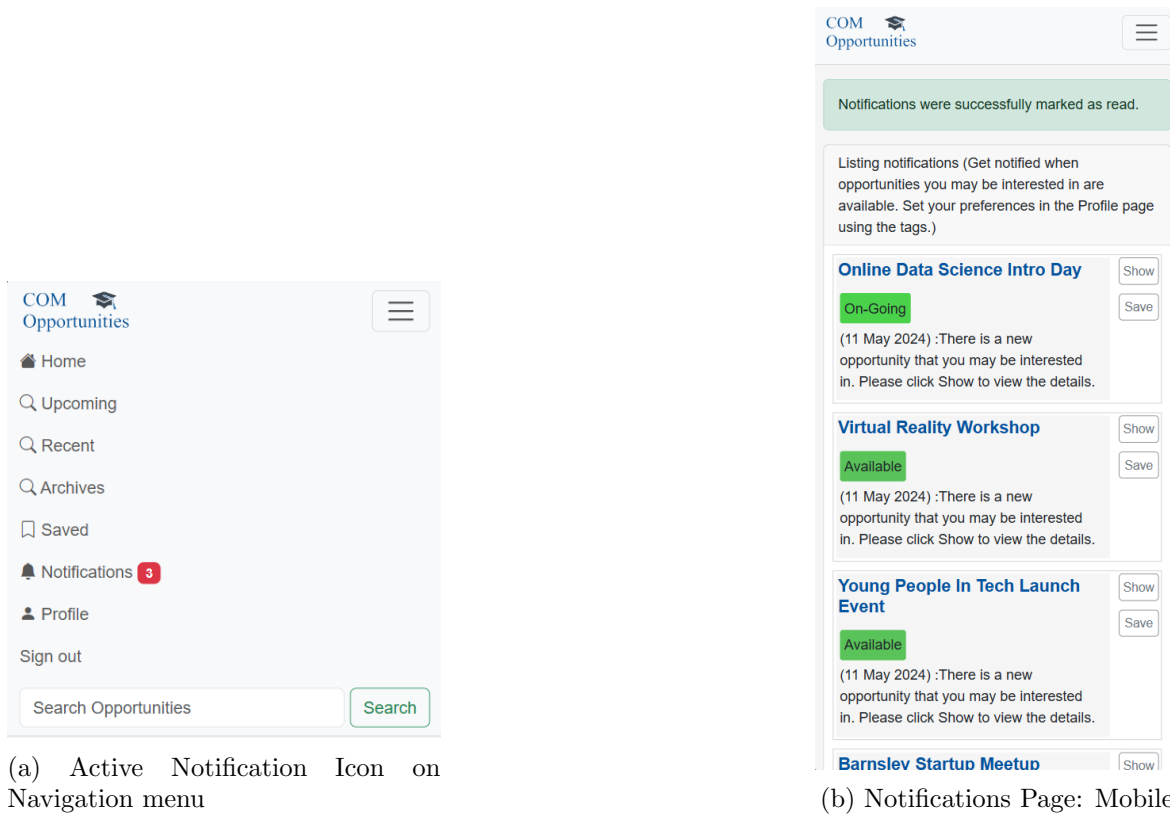


Figure 5.13: Notifications on Mobile

5.5 Administration

The administration component implements the Admin functionalities. As detailed in the previous chapter and shown in the user flow diagram, the Admin has all the capabilities of a Poster and some additional permissions. The Admin could approve and reject posts, view and search all posts, manage users, select posts for emailing, and generate email drafts for students/staff.

After logging in, the Admin could access the Pending page via the Admin dropdown on the navigation bar to view posts awaiting approval. Each post card included Show, Edit, Destroy, and Approve buttons. On clicking Approve, the server updated the post status to 'published' in the database, displaying it on the Upcoming page. This functionality was handled by the Posts controller.

The Admin could also generate an email draft by clicking the Email link via the Admin dropdown on the navigation bar. The Email page displayed all the posts that were recently published, along with a Select button which on click selected the post for emailing. After selection, the Admin could click the Email button. This generated a draft with the selected

posts' details, which the Admin can edit or copy to send. The server updated the emailed status of the selected posts to true. The emailing functionalities explained were handled by Email drafts controller.

Lastly, The Admin could view all users by clicking the Users link via the Admin dropdown on the navigation bar. The Users page displayed the user emails and roles, with an Edit button for each user. The Edit button rendered the Edit user role form containing a dropdown for changing the user role. Upon submission, the server updated the user's role in the database. Admin functionalities are managed by the Admin controller and Users model.

Figures 5.14, 5.15, 5.16, 5.17, 5.18, and 5.19 illustrate the Admin dropdown, Pending page, Email pages, Users page, and Edit user role form. These figures are in desktop view as Admin tasks are more likely performed on a desktop, though the pages are responsive for mobile use.

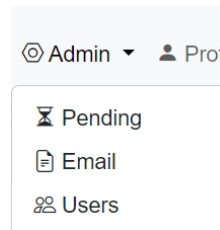


Figure 5.14: Admin Dropdown on Navigation bar

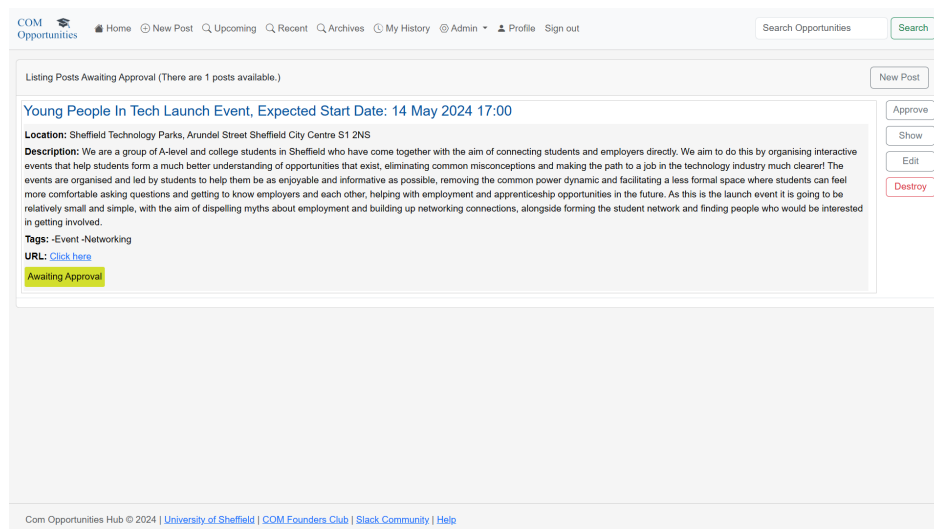


Figure 5.15: Pending Page

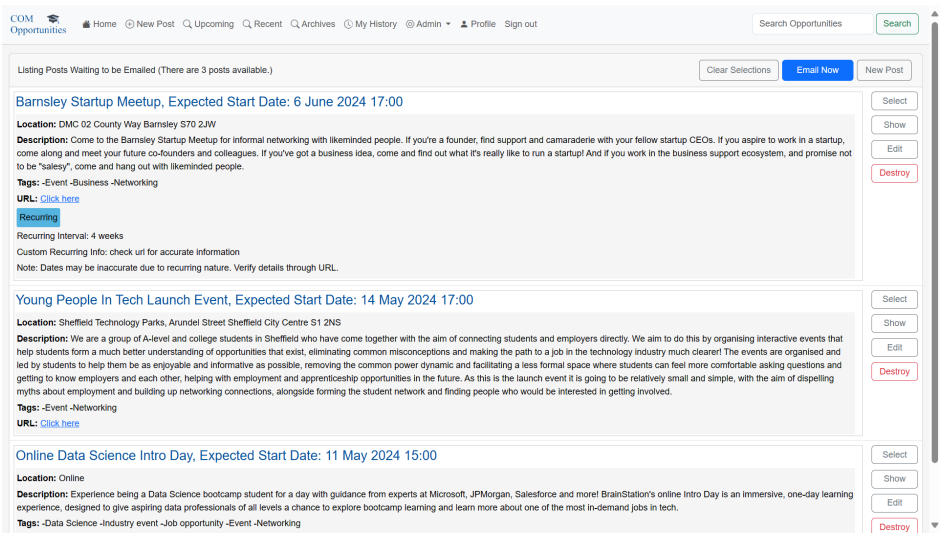


Figure 5.16: Email Page

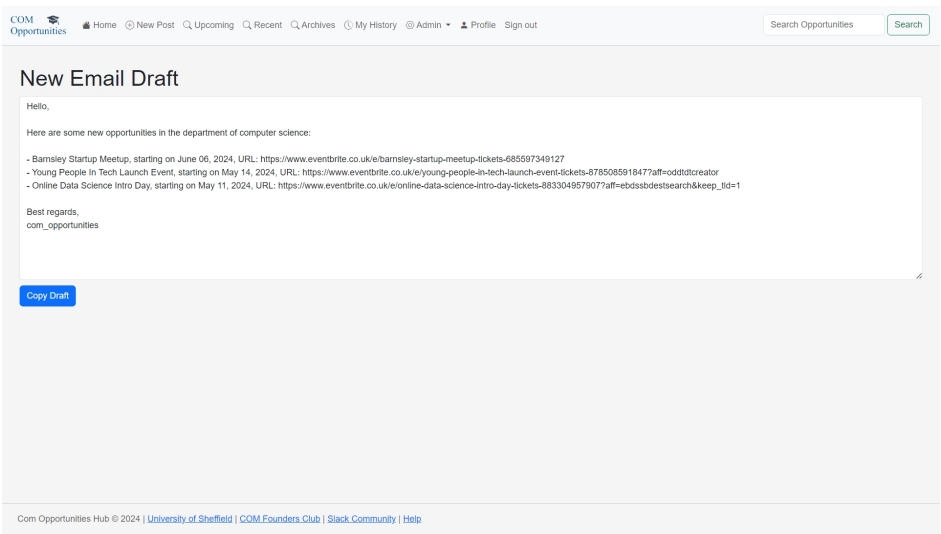


Figure 5.17: Email Draft

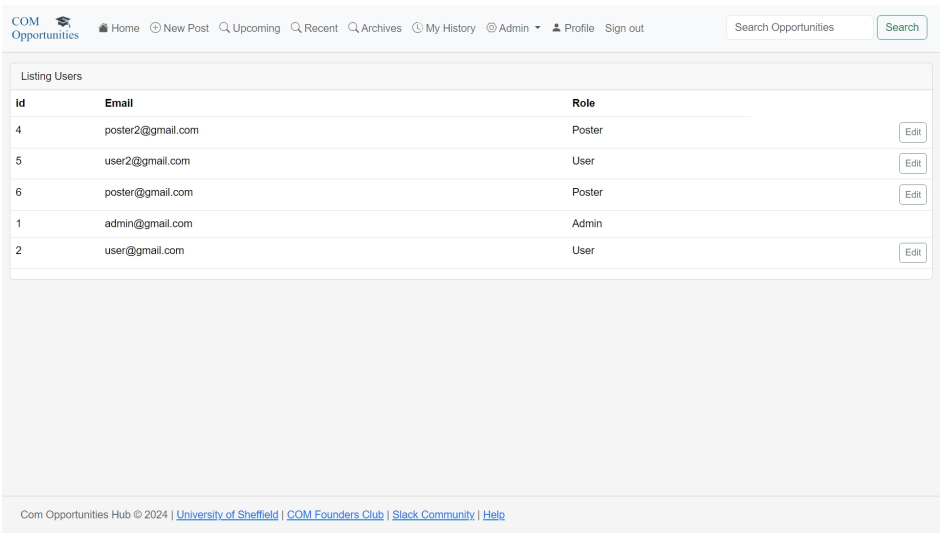


Figure 5.18: Users Page

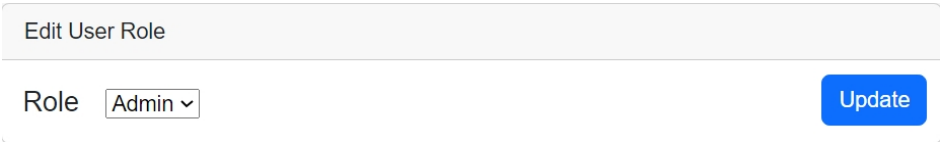


Figure 5.19: Edit User Role Form

Chapter 6

Testing

This chapter will discuss the testing of the system. The testing was done using a combination of automated testing and manual testing. Various levels of testing were done for each type to ensure that the system was working as expected.

6.1 Automated Testing

Automated testing was done using RSpec, a testing tool for Ruby. As discussed in the requirements and analysis chapter, the main forms of automated testing done were unit testing and integration testing. A total of 122 tests were written to test various features of the system. Additionally, after the implementation of the system, another level of automated testing, browser testing, was decided to be done to measure the effectiveness of the web app on the browser across different devices.

6.1.1 Unit Testing

Abiding by the agile scrum approach, after the completion of each sprint, unit tests were written for the models and controllers of the components developed in that sprint. The unit tests were written to check the correct implementation of each unit, and to ensure that the code was working as expected. An example of a simple unit test for the user model is shown in Figure 6.1. The test validates the length of the password is greater than or equal to 6 characters when signing up.

```
30 RSpec.describe User, type: :model do
31   describe 'validations:' do
32     it 'should validate the length of password' do
33       visit "/users/sign_up"
34       fill_in 'Email', with: 'abc@gmail.com'
35       fill_in 'Password', with: '12313'
36       fill_in 'Full name', with: 'abc'
37       click_button 'Sign up'
38       expect(page).to have_content("Password is too short (minimum is 6 characters)")
39     end
40   end
41 end
```

Figure 6.1: Unit Test for validating the length of the password

6.1.2 Integration Testing

After writing various unit tests for each piece of code in the system, integration tests were written to check the interaction between these units and components for achieving the desired functionality. The integration tests were written to check the correct implementation of the features and functionalities, and to ensure that the system was working as expected. An example of a complex integration test for the post controller is shown in Figure 6.2. The test checks the correct implementation of the approval functionality of the Admin.

```

2   RSpec.describe "Posts features:", type: :feature do
899   describe "Approve posts:" do
900     it "admin should approve a post" do
901       # Create an Admin account
902       user = User.create(email: 'test@example.com', password: 'password', full_name: 'Test User', user_role: '2')
903       # Login as an Admin
904       visit new_user_session_path
905       fill_in "Email", with: user.email
906       fill_in "Password", with: 'password'
907       click_button "Log in"
908       # Create a new post (Admin and Poster can create posts)
909       visit new_post_path
910       fill_in "Title", with: "Test Post"
911       fill_in "Location", with: "Test Location"
912       fill_in "post_start_date", with: Date.today
913       fill_in "post_end_date", with: Date.today
914       fill_in "Organiser", with: "Test Organiser"
915       fill_in "Description", with: "Test Description"
916       fill_in "post_deadline", with: Date.today
917       fill_in "start_time", with: Time.now
918       fill_in "end_time", with: Time.now + 1.hour
919       fill_in "URL", with: "http://example.com"
920       check "AI" # Assuming 'AI' is one of the tags
921       check "ML" # Assuming 'ML' is another tag
922       select "No", from: "Recurring"
923       click_button "Save"
924       # Once a new post is created, Go to pending page for approval
925       click_link "Admin"
926       click_link "Pending"
927       expect(page).to have_content("Test Post")
928       expect(page).to have_content("Awaiting Approval")
929       click_link "Approve"
930       # Approved posts must be displayed on the upcoming page
931       click_link "Upcoming"
932       expect(page).to have_content("Test Post")
933     end
  end

```

Figure 6.2: Integration Test for the approval functionality of the Admin

6.1.3 Browser Testing

After the completion of the development of the system, and writing various unit and integration tests, the web app was tested using the lighthouse tool. The lighthouse tool is a browser tool that evaluates the performance, accessibility, best practices, and SEO of a website. The web app (<http://localhost:3000/>) was tested on a variety of devices to ensure that the system was working as expected. The scores of the web app on desktop and mobile are shown in the Results and Discussion chapter.

6.2 Manual Testing

Manual testing included a combination of system testing and usability testing. System testing was done to test the fulfillment of the requirements, while usability testing was done to test the effectiveness, efficiency, and usability of the system.

6.2.1 System Testing

After the development of the system, and conducting various levels of automated testing, the system was tested as a whole against the requirements. The system was tested to ensure that the functional and non-functional requirements were met, and that the system was working as expected. As discussed in the previous chapter, and shown in the figures, the majority of functional requirements and the non-functional requirements of the web app were implemented successfully. The web app could be launched smoothly, and various functionalities were working as expected.

The functional requirements of the system were found to be well-implemented and robust. For non-functional requirements, the web app was responsive, and could be viewed on mobile as well as desktop. The interface of the web app was user-friendly, easy to use, and introduced automation to reduce manual work. The system abided by the Data Protection Act by keeping the data requirement from the users to a minimum, and by providing the option to delete the account. The web app was also secure, with the password encryption done using bcrypt gem. Additionally, all the personal data was stored securely, with most of it only accessible to the user. Accessibility of the web app was also ensured by making the web app responsive.

6.2.2 Usability Testing

Usability testing was done using cognitive walkthrough. The web app was tested by the developer from the end-user perspective to ensure that the system was user-friendly, easy to use, and met the requirements. During the cognitive walkthroughs, each user type was considered, and the user flow diagrams were followed to test the system. Although the usability testing was done by the developer, the system was tested as if it was being tested by the end-users. In the future, user acceptance testing will be done by volunteers and real users to get feedback and improve the system.

Chapter 7

Results and Discussion

This chapter presents the results of the project and discusses the findings. The chapter is divided into several sections. The first section presents the results of the Testing and Evaluation. The second section discusses the requirement evaluation. The final section discusses the future work that can be done to improve the system.

7.1 Testing and Evaluation Results

The system was evaluated using the strategies discussed in the Testing chapter. The Automated Testing, using the RSpec framework covered 248 / 258 LOC (96.12%) of the code. Out of 122 tests, 121 tests passed, and 1 test failed. Additionally, few tests were chosen not to be written due to the features they tested were either not implemented or deemed unnecessary. Summary of the failed or not written tests are as follows:

- **Failed Test:** When creating a post, deadline cannot be in the past.
Reason: The test failed because of the lack of mechanism to prevent the user from creating a post with a deadline in the past. The system allows the user to create a post with a deadline in the past, which should not be allowed. Although the system missed this feature, the test was written to ensure that the feature would be implemented in the next sprint.
- **Tests not written:** The following tests were not written:
 - Test 1:** The user should be able to suspend notifications for a time period.
 - Test 2:** The poster should be able to use web scraping to get the details of the post from a URL when creating a post.

Note: Summary of all the tests written and the results can be found in the Appendix section of the report.

After the completion of the automated testing, although the system had a good test coverage, and most of the features were implemented correctly, the failed test and the tests not written show a limitation of the testing strategy chosen for the project. TDD (Test Driven Development) could have been a better choice. TDD would have ensured that all the features were tested and implemented correctly, as the tests would have been written before the code, and the code would have been written to pass the tests. This would have also ensured that all the tests were written, and the system was fully tested with no bugs.

Further testing was done manually, which included system testing and usability testing. Manual evaluation of the system, especially the cognitive walkthroughs, which were conducted by the developer showed that the system is easy to use and fulfills the main requirements and expectations of the end-user (as outlined in the requirements gathering phase). While this evaluation is subjective, it is a good indicator that the system is user-friendly and effective. For a more objective evaluation, the system was evaluated using the lighthouse tool, another level of automated testing, to evaluate the performance, accessibility, best practices, and SEO of the website on the browser across different devices. The scores of the web app (<http://localhost:3000/>) on desktop and mobile are shown in Figures 7.1 and 7.2.

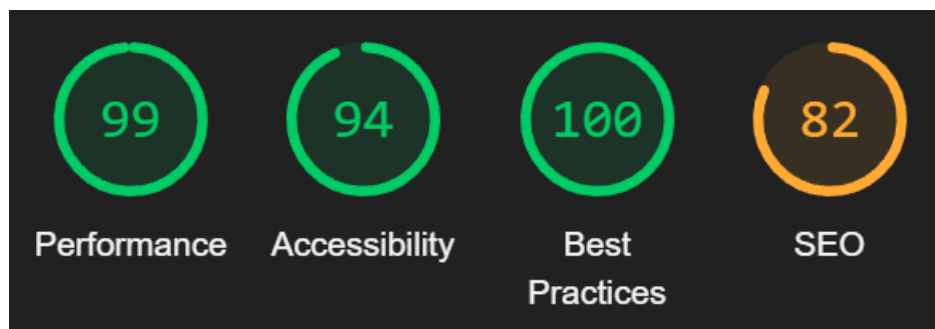


Figure 7.1: Desktop Lighthouse Score

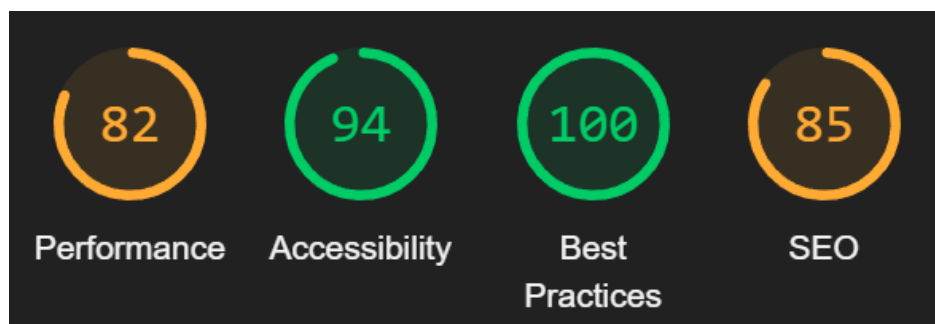


Figure 7.2: Mobile Lighthouse Score

As shown in the figures, the system has good performance, accessibility, best practices, and SEO scores for both desktop and mobile. The performance score on the desktop is better than

the mobile, which is expected because the system was designed as a desktop-first application. Although the system is responsive and works well on mobile, the scores can be improved by optimising the images and the code. This should be done in the next sprint.

7.2 Requirement Evaluation

The level of completion of the user stories mentioned in the Functional Requirements section of the Requirements and Analysis chapter is outlined below. The User Story No. links to the user stories in Table 3.1, 3.2, and 3.3.

User Story No.	Priority	Completion
1.1	M	Completed
1.2	M	Completed
1.3	M	Completed
1.4	M	Completed
1.5	M	Completed
1.6	M	Completed
1.7	M	Completed
1.8	M	Completed
1.9	M	Completed
1.10	M	Completed
1.11	M	Completed
1.12	M	Completed
1.13	M	Completed
1.14	M	Completed
2.1	S	Completed
2.2	S	Completed
2.3	S	Completed
2.4	S	Completed
2.5	S	Completed
2.6	S	Completed
2.7	S	Completed
2.8	S	Completed
3.1	C	Not Completed
3.2	C	Completed
3.3	C	Not Completed
3.4	C	Completed
3.5	C	Completed
3.6	C	Completed

Table 7.1: Requirement Evaluation

Table 7.1 shows that all of the Must-Have and Should-Have requirements were completed. However, some of the Could-Have requirements could not be completed. User Story 3.1 was not completed because the feature was deemed unnecessary and out of scope for the project.

The system being a web application, the only way for a User (Browser) to receive notifications is by logging in, hence it would not cause any inconvenience to them as they could always stop receiving notifications by not logging in, or unsubscribing the tags they are following. User Story 3.3 of the poster on the other hand was not completed due to time constraints, and shall be implemented in the next sprint.

The Non-Functional Requirements defined in Table 3.5 of the Requirements and Analysis chapter were also considered. The system adheres to all the non-functional requirements, as evaluated through manual testing in the previous chapter.

All major requirements expected by the end-user were completed. The main limitation of the old system which was the lack of automation was addressed along with the new features that were requested by the end-user. Additional features were added to make the system more user-friendly and easy to use. Some features that were supposed to be implemented were not implemented solely due to the lack of time. After the implementation of major features, spending time on minor ones would not have been a good use of time. Hence, that time was spent on testing and evaluation to ensure that the system was robust and well-tested.

7.3 Future Work

Looking ahead, there are several enhancements planned to further improve the system's functionality. Firstly, completing the integration of Google login remains a priority, despite encountering some technical hurdles, such as unexpected errors, and time constraints. This feature will provide users with an additional sign in option, enhancing accessibility.

Furthermore, the addition of filters for tags and dates will facilitate quicker and more efficient post discovery, enhancing the overall browsing experience, and streamlining post navigation. Enhancements in the post creation process are also planned, with the aim of preventing users from creating posts with deadlines in the past. This will ensure that all posts are relevant and up-to-date, improving the overall quality of the system.

Addressing complex recurring posts is another area for improvement. While the system adequately supports simple and straightforward recurring posts with consistent time intervals, more complex and inconsistent intervals are not yet supported. As the system can not accurately predict the dates of posts after each cycle, further design investigation is needed to improve this feature for handling diverse and irregular recurrence patterns effectively.

Optimising the system for improved performance, accessibility, best practices, and SEO scores is also planned. This will involve refining the system's code and image assets to enhance the overall user experience, particularly on mobile devices.

Deployment of the system to a server and conducting user acceptance testing with volunteers is also planned. This will provide valuable feedback on the system's usability and performance, enabling further refinements to be made, and ensuring the readiness of the system in a real-world setting.

Exploring the addition of web scraping functionality to the system is another area for future development. This feature will allow users to extract post details from URLs when creating posts, further enhancing efficiency and user convenience, aligning with our commitment to delivering an intuitive user experience.

Investigating the use of AI to automate tasks such as automated tag suggestion, auto categorisation, also holds the potential to further enhance the system's efficiency and functionality. This exploration aligns with our long-term vision of creating an advanced, user-friendly system that leverages cutting-edge technology to deliver an exceptional user experience.

Chapter 8

Conclusion

This report has outlined the research, analysis, design, implementation, and testing of the COM Opportunity Hub, an efficiency based web application developed to streamline the management and dissemination of opportunities within the Department of Computer Science. The project aimed to replace manual processes with automated ones, improve communication between opportunity posters and seekers, and introduce additional features to enhance the user experience.

Primarily, the project offered intuitive and efficient platforms for staff or external posters to post and manage opportunities while presenting them in an organised manner for interested students or staff. Additionally, administrators were provided with features for managing users and posts. The incorporation of features for browsing, searching, and tracking opportunities, along with personalised notifications based on user preferences, aimed to enhance communication and browsing experience.

The project began with understanding the background and motivation behind the project, followed by a literature survey and formulation of research questions. Findings from the survey were used to inform the project design and implementation. Following this, requirements were gathered from stakeholders, shortcomings of the existing system were analysed, and key features required for the new system were identified. The requirements were then prioritised based on their importance and expected impact.

The design phase involved planning the system architecture, technology stack, user flow, and database design. The project was categorised into components representing the different aspects of the system, each denoting the major functionality of various user types: Poster, User (Browser), and Admin. Following the design, implementation began, where the system was developed using Ruby on Rails, with the MVC pattern being the core. Database was implemented using PostgreSQL, and front-end using Bootstrap. Scrum for one, a modified version of the agile scrum methodology, was used for project management. The system was developed in sprints, with each sprint focusing on components described in the design phase.

Thorough testing, including automated and manual methods, ensured the system's robustness and correctness. Unit tests, integration tests, and browser tests were used for automated testing, while system testing and usability testing were done manually. The system was found to be well implemented, with all major functional and non-functional requirements being met.

Overall, the project successfully developed a web application that addressed the shortcomings of the old system, providing an efficient platform for managing, announcing, and browsing opportunities. It lays the foundation for future enhancements and expansions, such as integrating Google login, adding filters for tags and dates, and improving the post creation process. The project holds potential for further development and adaptation to cater to the needs of other departments and institutions, serving as a valuable tool for opportunity management and dissemination.

Bibliography

- Agile-Modeling (2023), ‘User stories: An agile introduction’. (Accessed: 22-11-2023).
URL: <https://agilemodeling.com/artifacts/userStory>
- alexa.amazon.co.uk (2023), ‘Amazon alexa’. (Accessed: 10-11-2023).
URL: <https://alexa.amazon.co.uk/>
- Alwagadani, B. & Alomar, K. (2017), ‘Personalizing university email toward user acceptance: An initial study’, *International Journal of Computer Applications (0975 – 8887)* **160**(5).
- apple.com/siri (2023), ‘Siri- apple’. (Accessed: 10-11-2023).
URL: <https://www.apple.com/siri/>
- assistant.google.com (2023), ‘Google assistant’. (Accessed:10-11-2023).
URL: <https://assistant.google.com/>
- BrainStation (2024), ‘Online data science intro day’. (Accessed: 11-05-2024).
URL: <https://www.eventbrite.co.uk/e/online-data-science-intro-day-tickets-883304957907>
- Carayannopoulos, S. (2017), ‘Using chatbots to aid transition’.
- Chodos, D. & Cohen, R. (2006), ‘A method combining email and web pages for announcing research opportunities to researchers’, p. 304–310.
- Choudhary, S., Landge, M., Salunke, S., Sutar, S. & Mhamunkar, K. (2016), ‘Advanced training and placement web portal’, *International Journal of Technical Research and Applications* **4**(2), 75–77.
- Collective, C. (2024), ‘Virtual reality workshop’. (Accessed: 11-05-2024).
URL: <https://www.eventbrite.co.uk/e/virtual-reality-workshop-tickets-891479468087>
- COM-Opportunities (2023), ‘Computer science opportunities hub, department of computer science, university of sheffield’. (Accessed: 25-10-2023).
URL: <http://bit.ly/uoscomopps>
- Consortium, A.-B. (2023), ‘Moscow prioritisation’. (Accessed: 22-11-2023).
URL: <https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html>

- equalityhumanrights.com (2011), ‘Services, public functions and associations: Code of practice’. (Accessed: 20-11-2023).
URL: <https://www.equalityhumanrights.com/equality/equality-act-2010/codes-practice/services-public-functions-and-associations-code-practice>
- Filipe, F., Pires, I. M. & Gouveia, A. J. (2023), ‘Why web accessibility is important for your institution’, *Procedia Computer Science* **219**, 20–27.
- gov.uk (2023), ‘Data protection’. (Accessed: 22-11-2023).
URL: <https://www.gov.uk/data-protection>
- Heitmayer, M. & Lahlou, S. (2021), ‘Why are smartphones disruptive? an empirical study of smartphone use in real-life contexts’, *Computers in Human Behavior* **116**, 106637.
- Heryati, A., Yulianti, E., Faradillah, Sartika, D., Dhamayanti, Saluza, I. & Sanmorino, A. (2019), ‘The design of smart notification on android gadget for academic announcement’, *Telkomnika* **17**(1), 147–152.
- Hong, I. B. (2007), ‘A survey of web site success metrics used by internet-dependent organizations in korea’, *Internet Research* **17**(3), 272–290.
- import.io (2023), ‘Import.io’. Accessed: 15-11-2023.
URL: <https://www.import.io/>
- ISO (2018), ‘Ergonomics of human-system interaction - part 11: Usability: Definitions and concepts (iso 9241-11:2018)’.
- Jamil, M. A., Arif, M., Abubakar, N. S. A. & Ahmad, A. (2016), ‘Software testing techniques: A literature review’, pp. 177–182.
- Khder, M. A. (2021), ‘Web scraping or web crawling: State of art, techniques, approaches and application’, *Int. J. Advance Soft Compu. Appl.* **13**(3).
- Kofod-Petersen, A. (2014), ‘How to do a structured literature review in computer science’.
- Larusdottir, M. (2009), ‘Listen to your users: The effect of usability evaluation on software development practice’.
- Lawson, R. (2015), *Web Scraping with Python*, Packt Publishing Ltd.
- Lazar, J., Dudley-Sponaule, A. & Greenidge, K.-D. (2004), ‘Improving web accessibility: a study of webmaster perceptions’, *Computers in Human Behavior* **20**(2), 269–288. *The Compass of Human-Computer Interaction*.
- legislation.gov.uk (2010), ‘Equality act 2010’. (Accessed: 20-11-2023).
URL: <https://www.legislation.gov.uk/ukpga/2010/15>
- Lewis, C. (1982), ‘Using the “thinking aloud” method in cognitive interface design’.

- Lin, J. (2019), 'An event-based social web app'.
- Lou, T. (2016), 'A comparison of android native app architecture mvc, mvp and mvvm'.
- Lucidchart (2024), 'Scrum for one: A tutorial on adapting agile scrum methodology for individuals'. (Accessed: 28-04-2024).
URL: <https://www.lucidchart.com/blog/scrums-for-one>
- monster.co.uk (2023), 'Monster'. (Accessed: 6-11-2023).
URL: <https://www.monster.co.uk/>
- Nielsen, J. & Molich, R. (1990), 'Heuristic evaluation of user interfaces'.
- octoparse.com (2023), 'Octoparse'. (Accessed: 15-11-2023).
URL: <https://www.octoparse.com/>
- Onyesolu, M. O., Odoh, M. O. & Ositanwosu, O. E. (2013), 'Short message service (sms) user interface system to support university portal services', *JOURNAL OF COMPUTER SCIENCE AND ENGINEERING* **18**.
- openai.com/chatgpt (2023), 'Chatgpt'. (Accessed: 10-11-2023).
URL: <https://openai.com/chatgpt>
- Orlova, M. (2016), 'User experience design (ux design) in a website development: Website redesign'.
- Paz, F. & Pow-Sang, J. A. (2016), 'A systematic mapping review of usability evaluation methods for software development process', *International Journal of Software Engineering and Its Applications* **10**(1), 165–178.
- reed.co.uk (2023), 'Reed.co.uk'. (Accessed: 6-11-2023).
URL: <https://www.reed.co.uk/>
- Ritter, M. & Winterbottom, C. (2017), *UX for the Web: Build websites for user experience and usability*, Packt Publishing Ltd.
- Robson, L., Cook, L. & Habgood, N. (2016), 'Student experience of university email communication', pp. 6118–6124.
- Sanmorino, A. & Fajri, R. M. (2018), 'The design of notification system on android smartphone for academic announcement', *Int. J. Interact. Mob. Technol.* **12**, 192–200.
- scrapy.org (2023), 'Scrapy'. (Accessed: 15-11-2023).
URL: <https://scrapy.org/>
- sheffield.ac.uk/accessibility (2023), 'Accessibility'. (Accessed: 20-11-2023).
URL: <https://www.sheffield.ac.uk/accessibility>

SIAN (2024), ‘Barnsley startup meetup’. (Accessed: 10-05-2024).

URL: <https://www.eventbrite.co.uk/e/barnsley-startup-meetup-tickets-685597349127>

Sikder, M. F., Halder, S., Hasan, T., Uddin, M. J. & Baowaly, M. K. (2017), ‘Smart disaster notification system’, pp. 658–663.

Stolz, C. (2005), ‘Guidance performance indicator - web metrics for information driven websites’.

Stone, D., Jarrett, C., Woodroffe, M. & Minocha, S. (2005), *User Interface Design and Evaluation*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA ©2015.

Subbe, C. P., Duller, B. & Bellomo, R. (2017), ‘Effect of an automated notification system for deteriorating ward patients on clinical outcomes’, *Crit Care* **21**, 52.

Tech, Y. P. I. (2024), ‘Young people in tech launch event’. (Accessed: 11-05-2024).

URL: <https://www.eventbrite.co.uk/e/young-people-in-tech-launch-event-tickets-878508591847>

Thakkar, J., Raut, P., Doshi, Y. & Parekh, K. (2018), ‘Erasmus - ai chatbot’, *International Journal of Computer Sciences and Engineering* **6**.

TK (2018), ‘Understanding the basics of ruby on rails: Http, mvc, and routes’. (Accessed: 22-11-2023).

URL: <https://www.freecodecamp.org/news/understanding-the-basics-of-ruby-on-rails-http-mvc-and-routes-359b8d809c7a>

W3C (2021), ‘Wcag 2.2 aa: Summary and checklist for website owners’. (Accessed: 20-11-2023).

URL: <https://wcag.com/blog/wcag-2-2-aa-summary-and-checklist-for-website-owners/>

W3C (2023), ‘Web content accessibility guidelines (wcag) 2.2’. (Accessed: 19-11-2023).

URL: <https://www.w3.org/TR/2023/REC-WCAG22-20231005/>

Waycott, J., Bennett, S., Kennedy, G., Dalgarno, B. & Gray, K. (2010), ‘Digital divides? student and staff perceptions of information and communication technologies’, *Computers and Education* **54**(4), 1202–1211.

Weinschenk, S. (2011), *100 Things Every Designer Needs to Know About People*.

Wolff, R. M. V., Nörtemann, J., Hobert, S. & Schumann, M. (2019), ‘Chatbots for the information acquisition at universities - a student’s view on the application area’.

Wong, E. (2016), ‘Simplicity in design: 4 ways to achieve simplicity in your designs’.

Zahran, D. I., Al-Nuaim, H. A., Rutter, M. J. & Benyon, D. (2014), ‘A comparative approach to web evaluation and website evaluation methods’, *International Journal of Public Information Systems* **10**(1).

Appendices

Appendix A

Summary of RSpec Tests

The automated testing using RSpec as discussed in the Testing chapter was done using a combination of unit testing and integration testing. Summary of a total of 122 tests written to test various features of the system are as follows:

RSpec Code Examples		122 examples, 1 failure Finished in 19.26634 seconds
<input checked="" type="checkbox"/> Passed <input checked="" type="checkbox"/> Failed <input checked="" type="checkbox"/> Pending		
Admin features:		
Users page		
should not show users to non-admin		0.28673s
should show all users to admin		0.13786s
should edit user role to admin		0.24977s
should edit user role to user (browser)		0.31901s
should edit user role to poster		0.45557s
ApplicationDecorator		
.collection_decorator_class		
returns PaginatingDecorator		0.00209s
#delegate_all		
delegates methods to the object		0.01102s
responds to methods defined on the object		0.00356s
does not respond to methods not defined on the object		0.00235s
PaginatingDecorator		
inherits from Draper::CollectionDecorator		0.00140s
includes Draper::AutomaticDelegation		0.00354s
ApplicationMailer		
default from address		
uses the correct default from address		0.00916s
PagesController		
GET #home		
returns http success		0.00922s
renders the home template		0.00574s

Figure A.1: Summary of RSpec Tests (1)

Posts features:	
creating a post-	
poster/admin creates a post	0.42252s
poster can't create a post without a title	0.26809s
poster can't create a post without a location	0.22169s
poster can't create post without a start date	0.27139s
poster can't create post without an end date	0.18029s
poster can't create post without an organiser	0.25401s
poster can't create post without a description	0.28377s
poster can't create post without a URL	0.31408s
poster can't create post without tags	0.22115s
poster can't create post without selecting recurring	0.24041s
poster can't create post with recurring but without recurring interval and interval unit	0.21194s
poster can't create post with recurring but with recurring interval num as negative or zero and interval unit as blank	0.19884s
poster can't create post with start date in the past	0.18164s
poster can't create post with end date before start date	0.16236s
poster can't create post with deadline before today	0.22434s
<pre> Failure/Error: expect(page).to have_content("Deadline can't be before today") expected to find text "Deadline can't be before today" in "Home New Post Recent Archives My History Profile Sign out" # ./spec/features/posts_spec.rb:589:in `block (3 levels) in <top (required)>' ./spec/features/posts_spec.rb:589:in `block (3 levels) in <top (required)>' 587 click button "Save" 588 # Expectations 589 expect(page).to have_content("Deadline can't be before today") 590 end 591 592 # Install the coderay gem to get syntax highlighting </pre>	
user (browser) can't create a post	0.12938s
editing a post-	
poster/admin edits a post	0.61126s
poster/admin can't edit a post by leaving a required field blank	0.54849s
only admin and post creator can edit a post	0.44408s
deleting a post-	
admin deletes a post	0.69340s
viewing a post-	
user (browser) views a post	0.46474s
searching for a post-	
user (browser) searches for a post	0.39386s
poster can't search for a post	0.12806s
Upcoming posts:	
should show all upcoming posts	0.37099s
should not show upcoming posts to posters	0.14763s
Recent posts:	
should show all recent posts	0.13224s
Archived posts:	
should show all archived posts	0.21706s

Figure A.2: Summary of RSpec Tests (2)

Approve posts:		
admin should approve a post		0.41029s
posters or user (browser) should not be able to approve a post		0.13191s
Email posts:		
admin should email a post		0.43773s
admin should not be able to email a post info if clear selection is clicked		0.52644s
poster or user (browser) should not be able to email a post		0.15284s
Save posts:		
user (browser) should be able to save a post as favourite		0.51652s
user (browser) should be able to unsave a post from the saved		0.83668s
if saved, show that it is already saved and give an option to unsave		0.78087s
posters and admin should not be able to see saved page		0.14599s
my history page:		
should show the history of all the posts the poster/admin has posted		0.34770s
user (browser) should not be able to view my history page		0.12533s
Notifications page:		
should show notifications for the user (browser) if a post's tags match the user's interests		0.59459s
should give Show and Save options for new notification of posts		0.75568s
admins or posters should not be able to view the notifications page		0.17735s
Profile features:		
Profile page		
should show profile		0.16528s
should edit profile		0.33006s
should not edit profile with invalid or taken email		0.25463s
should destroy profile		0.14002s
ApplicationRecord		
abstract class		
is an abstract class		0.00089s
Post		
validations:		
is valid with valid attributes		0.01108s
is not valid without a title		0.01445s
is not valid without a location		0.01188s
is not valid without a start_date		0.01150s
is not valid without a end_date		0.01200s
is not valid without a organiser		0.01144s
is not valid without a description		0.01293s
is not valid without a url		0.01084s
is not valid without recurring as true or false		0.01163s
is not valid with start_date in the past		0.01338s
is not valid with end_date in the past		0.01072s
is not valid with end_date before start_date		0.01234s
is not valid with recurring_interval_num as negative		0.01198s
is not valid with recurring_interval_unit as blank if recurring_interval_num is present		0.01178s
is not valid with recurring_interval_num and recurring_interval_unit as blank if recurring is true		0.01272s
is not valid with custom_recurring_info as blank if recurring is true		0.00150s
is not valid without tags		0.01101s

Figure A.3: Summary of RSpec Tests (3)

User	
validations:	
should validate presence of email	0.06695s
should validate presence of password	0.06960s
should validate the uniqueness of email	0.09256s
should validate the length of password	0.11315s
should validate the format of email	0.14589s
should not be valid when email is nil	0.01345s
should not be valid when password is nil	0.01616s
authenticate:	
should return the user when email and password are correct	0.15757s
should return error when email is incorrect	0.08504s
should return error when password is incorrect	0.08559s
Database:	
should have a table	0.00203s
should have columns	0.00150s
should have a primary key	0.00133s
should have indexes	0.01039s
should have a unique index	0.00713s
should have default values	0.00286s
should have not null constraints	0.00183s
Factory:	
should create user (browser)	0.00827s
should create poster	0.00925s
should create admin	0.00771s
Insertion:	
should insert user	0.01406s
Deletion:	
should delete user	0.01282s
Update:	
should update user	0.01747s
Registration:	
should register user	0.09202s
should not register user with invalid email	0.05523s
should not register user with too short password	0.07799s
should not register user with already taken email	0.06216s
should not register user with empty email or password	0.04641s
Sign in:	
should sign in user	0.07841s
should not sign in user with invalid email	0.06839s
should not sign in user with invalid password	0.06199s
should not sign in user with empty email or password	0.10381s
Sign out:	
should sign out user	0.13816s

Figure A.4: Summary of RSpec Tests (4)

Forgot password:	
should send reset password email	0.24541s
should not send reset password email with invalid email	0.07794s
should not send reset password email with empty email	0.06001s
should send email with reset password link	0.14390s
Devise Configuration:	
uses database authenticatable	0.00131s
is registerable	0.00123s
is recoverable	0.00135s
is rememberable	0.00128s
is validatable	0.00118s
.from_omniauth:	
when user does not exist	
creates a new user from omniauth data	0.02098s
when user already exists	
does not create a new user	0.01670s
returns the existing user	0.01368s

Figure A.5: Summary of RSpec Tests (5)