

MUJ Unite

A PROJECT REPORT

Submitted in Partial Fulfilment of the Requirement for the Award of the Degree  
of  
MASTERS OF COMPUTER APPLICATIONS (MCA)

By

Rohan Nigam

23FS20MCA00087



MANIPAL UNIVERSITY  
JAIPUR

Department of Computer Applications  
School of Computer Applications  
Faculty of Science

MANIPAL UNIVERSITY JAIPUR  
JAIPUR-303007  
RAJASTHAN, INDIA

May 2025

**DEPARTMENT OF COMPUTER APPLICATIONS**  
MANIPAL UNIVERSITY JAIPUR, JAIPUR – 303 007 (RAJASTHAN), INDIA

16<sup>th</sup> May, 2025

**CERTIFICATE**

This is to certify that the project titled **MUJ This Unite** is a record of the bonafide work done by **ROHAN NIGAM** (23FS20MCA00087) submitted in partial fulfilment of the requirements for the award of the Masters of Computer Applications (MCA) of **Manipal University Jaipur**, during the academic year **2023-2025**.

**Dr. Monika Jyotiyana**

*Project Guide, Dept of Computer Applications*

*Manipal University Jaipur*

**Dr. Shilpa Sharma**

*HOD, Dept of Computer Applications*

*Manipal University Jaipur*

## **ACKNOWLEDGMENTS**

In particular, I have my project manager Dr. Amit Hirawat. I want to thank Dr. Amit Hirawat for his constant guidance and valuable advice, which helped me to stay on the right path and improve my work at each stage.

My honest thanks to my project guide, Dr. Monika Jyotiyana also goes to her encouragement, thoughtful suggestions and continued support throughout the project.

To provide a supportive environment and provide the resources needed to do this work, I would like to thank the Head of Department Dr. Shilpa Sharma.

Finally, I am grateful for all the faculty members and staff at the department for their help and encouragement in my educational journey.

## ABSTRACT

During my time at Manipal University Jaipur, I often felt there was something missing—a common space where students, teachers, and alumni could all connect easily. We had social media, emails, and chats, but no single platform that truly brought everyone together. That gap sparked an idea: why not build something that does exactly that?

That's how **MUJ Unite** came into the picture. I wanted to create a simple and useful web application where people from the university—whether they're current students, faculty members, or alumni—can interact, share updates, and stay connected in one place.

I began by jotting down the key features I thought would be helpful: user login, profile pages, announcements, messaging, and posts. I used **Django** for the backend because it made things like user management and data storage easier. For the frontend, I stuck to familiar tools like **HTML, CSS, and JavaScript**, aiming for a clean and user-friendly interface.

I built and tested the platform step by step, making small improvements along the way. The result is a working version of MUJ Unite where users can sign up, update their profiles, chat with others, make announcements, and more—all from one place. The feedback I got during testing was encouraging. People found it easy to use and liked how everything was organized.

Overall, this project was both challenging and rewarding. I learned a lot, not just about coding, but also about how to build something practical from scratch that solves a real problem within my university community.

## LIST OF TABLES

<b>Table No</b>	<b>Table Title</b>	<b>Page No</b>
2.2.1	Database table	19
3.3.1	Agile Sprint Table for MUJ Unite Development	20
4.1.1	Implementation of each Module	31
4.1.2	Users Permissions	32
4.1.3	Modules listed in MUJ Unite	32
4.1.4	Details for QR code	37
4.2.1	Prototype v/s Final Implementation	56

## LIST OF FIGURES

Figure No	Figure Title	Page No
2.2.1	Code for manipulating data	12
2.2.2	Syntax for Django Templating Engine	12
3.2.1	Activity Diagram	23
3.2.2	Login Activity Diagram	24
3.2.3	Zero level DFD	26
3.2.4	Level 1 DFD	27
3.2.5	User Management Module Level 2 DFD	28
3.2.6	Communication Management Module level 2 DFD	28
3.2.7	Post Management Module Level 2 DFD	29
3.2.8	QR Code Module Level 2 DFD	29
4.2.1	Home Page	47
4.2.2	After login Home page	47
4.2.3	Login page	48
4.2.4	Sign up Page	48
4.2.5	Profile page	49
4.2..6	QR Code for scanning profile	50
4.2.7	Post option at profile page	50
4.2.8	Messaging option	51
4.2.9	New Chat option	51
4.2.10	Private Chat option	52
4.2.11	Announcement Page	53
4.2.12	Contact Us	54
4.2.13	About Us	55
4.2.14	Change password	55
4.2.15	Change Your password	56

# Contents

	Page No
Acknowledgement	i
Abstract	ii
List of Tables	iii
List of Figures	iv
<b>Chapter 1 INTRODUCTION</b>	
1.1 Introduction to work done/ Motivation ( <i>Overview, Applications &amp; Advantages</i> )	1
1.2 Project Statement / Objectives of the Project	1-2
1.3 Organization of Report	2-3
<b>Chapter 2 BACKGROUND MATERIAL</b>	
2.1 Conceptual Overview ( <i>Concepts/ Theory used</i> )	4-5
2.2 Technologies Involved	5-19
<b>Chapter 3 METHODOLOGY</b>	
3.1 Detailed methodology that will be adopted	20-22
3.2 Block diagrams	22-30
<b>Chapter 4 IMPLEMENTATION</b>	
4.1 Modules	31-46
4.2 Prototype	46 -57
<b>Chapter 5 RESULTS AND ANALYSIS</b>	
5.1 Conclusions	60
5.2 Future Scope of Work	61
<b>Bibliography</b>	<b>62-63</b>

# Chapter 1

## Introduction

MUJ-Unite is a web application made specifically for the Manipal University Jaipur community. "MUJ-Unite" wants to be an interactive platform for the student seeking career guidance, an alumni seeking to give back, or a teacher seeking to remain connected with former students—I designed MUJ Unite to help bridge the real challenges I observed as a student at MUJ—especially the lack of consistent alumni interaction. Through this platform, I aim to create practical connections and a user-friendly environment for growth and mentorship. The core idea of MUJ Unite is to help students, teachers, and alumni feel more connected in a shared space for communication and growth. Easy for all to remain connected, share their experiences, and assist one another to grow together.

### 1.1 Motivation

The motivation behind this project came from observing the communication gap that often exists in universities. Important announcements can get missed, students rarely connect with alumni, and faculty may find it difficult to engage with students outside the classroom. Social media platforms are not always suitable for official or academic communication. So, I wanted to create a platform that serves as a professional yet friendly space for university members to interact, collaborate, and stay connected.

➤ *Applications:* Many applications are discussed below:

- **Students** can get updates about events through posts or academic announcements.
- **Fresher's** can easily get guidance with the seniors.
- **Faculty members** can post class notices or departmental news and communicate directly with students.
- **Alumni** can share their experiences, post opportunities, and stay connected with the university.
- It can also be used for networking, event coordination, feedback collection, and building a sense of community.

➤ *Advantages:* Some of the advantages are listed below:

- Centralized platform for communication within the university.
- Improves engagement among students, faculty, and alumni.
- Simple user interface that's easy to use even for non-technical users.

### 1.2 Project Statement

So basically, the idea for MUJ Unite came from the fact that students, alumni, and teachers at MUJ didn't really have a proper space to talk or stay in touch. Everything was scattered—some stuff went on WhatsApp, some in emails, some people didn't even know where to check. It just wasn't working well. I thought, "Why not make one site where everyone can just log in and be part of the same thing?" That's how the project started.



One of the cool things I added was this QR scanning feature. Like, if you're at an event and wanna connect with someone, you just scan their code and boom—you get their profile right away. No need to ask for names or emails or whatever. It actually saves time and feels kinda modern too. It's super handy during college fests or alumni events.

The platform also lets people send messages one-to-one, which is useful when you wanna ask something or just talk without making it public. Say, a student wants to ask a doubt to a professor or maybe alumni wants to help a junior—it's just easier when you have a direct chat option. And yeah, there's an announcements part too, where important notices or updates can be shared. So everyone stays updated, instead of hearing things last minute or not at all.

Now, for security, I didn't wanna let just anyone in. So I set up a system where people need to use their MUJ email to sign up. That way, outsiders can't join. For alumni, I made it even more secure—they get a one-time password on their phone which they need to enter while signing up. This just adds that extra trust, like okay, this person really belongs here.

Altogether, with stuff like QR scanning, personal chats, updates, and a locked-down signup system, MUJ Unite kinda pulls everything together. It helps everyone stay connected in a neat, easy way. It's not super fancy, but it works. And honestly, it fixes a problem we all face—just staying in touch in a way that actually makes sense.

### *1.3 Organization of Report*

I've organized this report into separate chapters to keep everything clear and easy to follow. Each chapter walks through a different phase of creating MUJ Unite—from the initial planning stages to the actual development and testing. My goal was to keep it structured so that anyone reading it can get a complete picture of how the project took shape step by step.

#### ➤ Chapter 1 – Introduction

This first chapter sets the stage by sharing what led to the idea for this project. It looks at how MUJ Unite can help bring together students, alumni, and faculty in a meaningful way. I've also outlined the main goals I aimed to achieve with this platform, along with a quick guide on what to expect in the chapters that follow.

#### ➤ Chapter 2 – Background Material

This part of the report gives the background information and key concepts that helped shape the project. It includes an overview of the concepts and technologies used, such as web development frameworks (Django), frontend tools (HTML, CSS, JavaScript), databases (SQLite), and concepts of user authentication, database modelling, and client-server architecture.

#### ➤ Chapter 3 – Methodology

The methodology chapter details the systematic approach taken for the project's design and development. It includes planning strategies, block diagrams, architectural layouts, and the step-by-step process followed to build the system. This chapter explains how the requirements were translated into a working solution.

➤ Chapter 4 – Implementation

This chapter describes how the MUJ Unite portal was implemented in practice. It breaks down the system into various modules such as user authentication, post creation and interaction (like/unlike), messaging, announcements, and QR code generation. Screenshots, code structure, and component-level explanations are provided to illustrate how each feature works.

➤ Chapter 5 – Results and Analysis

This chapter presents the final output of the system along with screenshots of the working interface. It evaluates the system's performance based on testing results and analyses how effectively it meets the project objectives. If any user feedback or test results were collected, they are also discussed here.

➤ Chapter 6 – Conclusions and Future Scope

This section summarizes the key takeaways and achievements of the project. It highlights the project's contribution to solving the stated problem and suggests at least three possible areas where the system can be improved or expanded in the future, such as adding video calling, mobile app integration, or advanced analytics.

➤ Bibliography

This section list of cited academic papers, articles, websites, software documentation, and other resources referred in the preparation of the project.

## Chapter 2

### Background Material

#### 2.1 Conceptual Overview

The MUJ Unite project is built upon foundational theories in web development, information systems, and human-computer interaction.

➤ **Web Application Architecture (Client-Server Model):**

The project uses the client-server architecture, a standard model in web applications where the client (frontend) sends requests to a central server (backend) [1]. The server processes these requests (e.g., login, post, message) and responds with the appropriate data or confirmation. This architecture provides scalability, modularity, and centralized control over application logic and data storage.

➤ **MVC Pattern (Model-View-Controller – Django’s MTV Variant):**

In MUJ Unite, I organized the backend using Django’s MTV structure. Models handled user data, templates displayed it, and views acted as the bridge between user actions and the server, making it easier to handle database logic, page rendering, and user requests [2].

- **Model:** Represents the data structure and handles database interactions (e.g., Post, User, Message models).
- **Template:** Defines how data is displayed in the frontend using HTML, with placeholders for dynamic content.
- **View:** Handles logic between user input and database response. It acts as the controller that receives HTTP requests, processes data, and returns responses via templates.

This separation of concerns helps in maintaining and scaling the application efficiently.

➤ **User Authentication and Session Management:**

User authentication is a critical feature in MUJ Unite, ensuring secure access to the portal. Django’s built-in authentication system uses session management to store login states. Each user (student, alumni, admin) is assigned roles that define access permissions:

- Students can view and interact.
- Alumni can post, like, and message.
- Admins can moderate content and users.
- Session IDs are stored in browser cookies to maintain login status.

➤ **CRUD Operations:**

The project is centred around CRUD operations (Create, Read, Update, Delete). Examples include:

- Creating and deleting posts or announcements.
- Reading user profiles.
- Updating personal information.
- Deleting old messages or records.
- CRUD functionality is fundamental for dynamic, interactive web applications [3].

➤ **RESTful Communication:**

Even without a full REST API, the portal uses HTTP methods like GET and POST for functions such as posting updates or sending messages, making future integration with mobile apps easier. HTTP verbs like GET, POST, and DELETE are used during form submissions and dynamic content updates. This modular interaction allows for future integration with mobile apps or external APIs [4].

➤ **QR Code Generation Theory:**

Each user profile includes a dynamically generated QR code. This uses data encoding algorithms to convert user-specific data (e.g., name, registration ID, profile URL) into a scannable 2D matrix. Libraries like QR code and Pillow Use Reed–Solomon error correction and image encoding techniques to produce reliable codes for offline scanning or event check-ins.

➤ **Relational Database Concepts:**

The application uses a relational database (SQLite) based on ER modelling.

- Primary and Foreign Keys for relationships (e.g., each post is linked to a user).
- Normalization to avoid redundancy.
- Joins and Queries to fetch user data, posts, and messages.

➤ **Data Validation and Security Concepts:**

Django’s form system helps catch invalid or harmful inputs by checking them both in the browser and on the server. It also comes with built-in features that guard against several common security threats [5].

- Cross-Site Request Forgery (CSRF)
- SQL Injection
- Cross-Site Scripting (XSS)
- The use of Django’s secure framework reduces vulnerability risk and improves data integrity.

## *2.2 Technologies Involved*

The development of MUJ Unite was made possible by using a well-rounded set of modern tools and technologies. Each one played a specific role—whether it was for designing the frontend, running the backend, or handling the database.

➤ *Frontend Technologies*

» *HTML5 (Hypertext Markup Language)*

While working on MUJ Unite, HTML5 was basically the backbone of the website’s layout. I used it to set up the structure for different parts of the site—like login forms, profile pages, announcements, and more. It really helped me keep everything neat and organized. Since HTML5 works well across devices and browsers, it made things a lot easier when trying to build a responsive and accessible site [6].

One thing I found super helpful was the use of semantic tags. Stuff like `<section>` and `<article>` made the content way more readable, not just for me while coding, but also for browsers to understand what each part of the page was for. I used `<section>` to group related stuff like announcement areas, and `<article>` came in handy for individual posts so I could keep things styled properly and consistent.

For the user profile section, I kind of mixed `<div>` elements with semantic tags to arrange things like the user's name, profile picture, and bio. This setup worked well, especially on mobile screens, and gave me better control while designing the look and feel of the page.

Posts and announcements were each wrapped in `<article>` tags so I could style them separately and render them nicely using Django templates. It kept everything looking consistent and made updates easier whenever I had to change something [7].

## » *CSS3 (Cascading Style Sheets)*

In Developing MUJ Unite, I adjusted colors, fonts, spacing, and positioning to create a consistent look that felt clean and modern across all devices, whether viewed on a laptop or mobile screen [8].

One of the things I appreciated about CSS is how it keeps design separate from content. Rather than mixing visual styling with the HTML markup, I kept all the design rules in external stylesheets. This made the code easier to maintain, and any changes to the theme—like switching font sizes or background colors—could be made in just one place. I used classes and IDs to target specific elements, which gave me precise control over how different parts of the site appeared, from user profiles to buttons and message boxes.

When it came to laying things out on the page, I relied a lot on Flexbox. It really helped me arrange stuff like the navigation bar, post sections, and forms in a clean and organized way. Flexbox made it easy to line things up either in rows or columns, and the best part—it automatically adjusted when the screen size changed, which saved me a ton of time.

For some of the more detailed pages, like the admin dashboard or areas with multiple panels, I used CSS Grid. It gave me more control since I could set up rows and columns exactly how I wanted and place things right where they needed to be. This made everything feel a lot more structured and responsive, no matter what device it was viewed on.

CSS was also a big help when it came to making the site work well on different screen sizes. I used media queries to make sure everything looked right, whether someone was on a big desktop screen or a small phone. On smaller devices, I made a few tweaks—like cutting down the number of columns and bumping up the font size so things were easier to read. These changes really helped make the site more user-friendly across all kinds of devices [9].

I also made use of CSS variables to keep things like colors, spacing, and fonts consistent throughout the site. It really came in handy because once I set those values, I could reuse them across different pages. If I ever needed to change the design, I only had to tweak a few variables instead of editing

everything one by one. Sticking to the same style across the site also gave MUJ Unite a more polished, branded look, which made it feel more put-together and professional [10].

## » *JavaScript*

JavaScript plays an important role in making the responsive and attractive web design. I used it primarily to manage real-time interactions, update parts of the interface dynamically, and improve overall user engagement [11].

One of the key places where I used JavaScript was in implementing like/unlike functionality on posts. When a user clicked the "like" icon, the script instantly updated the icon's appearance and adjusted the like counter, giving immediate feedback. This action also sent a background request to the server so that the like could be stored in the database without the page reloading. This quick interaction kept the flow of the app smooth and more engaging.

JavaScript also helped improve form usability across the site. On pages like login, registration, and profile update, I used JavaScript to validate user input before sending it to the backend. It could detect if fields were left empty or if an email address was typed incorrectly. I also added dynamic features, like hiding or showing form sections based on what the user selected from dropdowns. For example, if a user selected "Alumni," additional fields like graduation year and department appeared automatically. This made the forms feel smarter and more personalized [12].

On top of that, I used JavaScript to manipulate elements on the page in real-time using the Document Object Model (DOM). Whether it was showing a success message after form submission or highlighting missing fields with a red border, these interactions made the site feel more responsive. In the messaging section, I plan to extend this by dynamically loading chat messages without needing to refresh the entire page.

While I didn't use a frontend framework like React or Vue in this version of MUJ Unite, I did rely on some of JavaScript's modern features to streamline code, such as event delegation and asynchronous data handling using `fetch()`. These features made the code more efficient and kept the interface responsive even as more content was added.

I also applied JavaScript for smaller enhancements like dropdown menus, confirmation popups, and modal windows. These were helpful in sections like account deletion, where a confirmation modal prevented accidental actions. Overall, JavaScript helped me improve the user experience without adding heavy libraries, and it allowed for a smoother interaction between users and the platform's features [13].

## » *Bootstrap*

During the development of MUJ Unite, I chose Bootstrap as the main front-end framework to speed up the design process and ensure a consistent user interface across all pages. One of the biggest advantages was how easily it let me create layouts that worked well on desktops, tablets, and smartphones without writing a lot of custom CSS.

The first feature I made use of was Bootstrap's grid system. It helped me organize page content into structured rows and columns. For instance, on the alumni dashboard, I used a two-column layout where user information appeared on the left and recent posts on the right. Because the grid system is responsive by default, I didn't have to worry about manually adjusting layouts for smaller screens—Bootstrap handled that automatically [14].

Beyond layout, Bootstrap offered a wide range of built-in components that saved development time. Elements like buttons, modals, navigation bars, and cards were already styled and responsive, so I could simply plug them into my HTML structure and move forward. This was particularly helpful when building the login page, profile view, and announcement cards, where consistency in design was important for usability.

## ➤ *Backend Technologies*

### » *Python*

Python is very important when it came to building the backend of MUJ Unite. I choose it because it's easy to understand, has a ton of helpful resources online, and works really well with Django, which I used throughout the project. Everything that happened in the background—like dealing with user login, saving data, generating QR codes, and managing the database—was handled using Python [15].

What I really liked about Python was how clean and simple the code looks. It just made everything easier—from writing the actual features to fixing bugs when something didn't work as expected. Since the platform needed to support different kinds of users like students, teachers, and alumni, Python made it easy to set up workflows for each group without making things overly complicated.

One cool part of the project was the QR code feature. It lets users scan each other's codes during events to view profiles. I used a Python library to create it, and honestly, it didn't take much—just a few lines of code to generate unique QR codes based on each user's ID. Then I stored those in the database and displayed them on their profile pages.

I also used Python to manage how the site handled things like form submissions. So when someone signs up or logs in, Python checks the info they entered, runs the right functions—like sending OTP emails or checking their credentials—and keeps things flowing smoothly on the backend.

Another thing I appreciated was how well Python worked with SQLite through Django's ORM. I didn't have to write complex SQL by hand. Instead, I just defined models in Python and used those to interact with the database. It saved me a lot of time and reduced the chance of running into random bugs due to messed-up queries.

Overall, Python was more than just a programming language in this project—it was the engine that powered the core functionality of MUJ Unite. From managing user sessions to customizing admin workflows, Python helped me turn the project's ideas into a working, scalable web application.

- **Simple and Readable Syntax:** Python's biggest advantage is its human-readable syntax, which closely mimics the English language. Statements like `if user.is_authenticated:` or `for post in posts:`

are self-explanatory, even for beginners. This feature greatly improves code readability and team collaboration, as developers can quickly understand each other's work with minimal explanation.

In MUJ Unite, this benefit enabled faster development, easier debugging, and efficient teamwork. Developers could focus more on solving the problem rather than dealing with complex syntax. The simplicity also helped during the testing and troubleshooting phases of the project.

- **Extensive Library and Module Support:** Python has an various types of libraries which plays an important role in MUJ Unite. Some of the libraries used are given below:
  - **QR code:** This library allows generation of QR codes that are automatically embedded into each user's profile.
  - **Pillow:** As an imaging library, Pillow helped save, manipulate, and render the generated QR codes in .png format.
  - **Date time:** Used to manage timestamps for posts, messages, announcements, and user registration.
  - **OS and uuid:** These were used to manage file storage and generate unique file names or tokens when saving images and other user-specific files.
- **Integration with Web Frameworks (Django):** Python works seamlessly with robust web frameworks like Django, which was used extensively in MUJ Unite. Django is written entirely in Python and follows the MTV (Model-Template-View) architectural pattern [16]. It provides tools and modules for:
  - Database management
  - Authentication
  - URL routing
  - Form handling
  - Session management
  - Template rendering

Python's synergy with Django ensured that even complex features like dynamic post feeds, role-based access control, and messaging systems could be implemented efficiently with clean and maintainable code.

- **Backend Processing and Data Management:** In MUJ Unite, Python was responsible for handling all server-side operations. Some examples include:
  - **User Authentication:** Validating login credentials, managing session cookies, and protecting pages with decorators like @login required.
  - **Form Submissions:** Processing form data for posts, messages, feedback, or event registration, with built-in validation and CSRF protection.
  - **Database Interactions:** Using Django's ORM, Python communicates with the SQLite database to perform CRUD operations. Every time a user adds a post or message, Python executes database transactions under the hood.



- Routing: Defining URL patterns and mapping them to views that process data and render the appropriate templates.

## » *Django Framework*

Django basically acted as the backbone for MUJ Unite’s backend. I went with it because it’s super practical and takes care of a lot of the usual web development stuff right out of the box. It handled things like setting up URLs, managing users, connecting to the database, and generating dynamic web pages—all without me having to build everything from scratch, which was a huge time-saver.

One thing I really liked about Django is its “batteries-included” approach. It gives you a lot of built-in tools, like authentication, session handling, form support, and even a whole admin panel that works straight away. For example, I used Django’s built-in forms and auth system to build the login and signup flow. I did tweak a few things to make sure it worked for different types of users like students, teachers, and alumni, but having the core features already there made life way easier [17].

The way Django is structured using the Model-Template-View (MTV) pattern also helped me stay organized. Models were great for defining my data—stuff like user profiles, posts, messages, etc. Views handled the main logic whenever a user did something like submit a form or open a page, and the templates took care of how everything looked on the screen. Using Django’s rendering tools, I was able to pass data into these templates and build dynamic pages without too much hassle.

Instead of writing SQL queries manually, I used Python classes to create, retrieve, and update data. This made the development process faster and reduced the chance of making database-related errors.

Another standout feature of Django was its built-in admin interface, which I customized to help manage announcements, user accounts, and feedback entries. This dashboard allowed me to test and monitor features in real-time, which made debugging and content management much easier during development.

Overall, Django was essential in bringing together all the moving parts of the MUJ Unite portal. It offered the structure, flexibility, and tools I needed to build a functional, secure, and scalable web application in a relatively short amount of time.

- MTV Architecture: Django follows a design pattern known as MTV (Model-Template-View), which is similar in concept to MVC (Model-View-Controller) but with different terminologies:
  - Model: Manages the data and business logic. In Django, each model is a Python class that maps to a database table using its powerful Object-Relational Mapper (ORM). For MUJ Unite, models such as User, Post, Message, and QR Code Data were used to handle the application's various data entities.
  - Template: Controls the presentation layer. Templates are HTML files with Django Template Language (DTL) syntax, which allows dynamic data to be inserted from the backend. In MUJ Unite, templates are used to render user profiles, announcements, message threads, etc.

- View: Handles request processing and returns a response. Views link the models and templates together, processing logic like login validation, form submissions, and QR code generation.
- Key Features Utilized in the MUJ Unite Project: Some of the key feature are discussed below:
  - Rapid Development: One of Django's strongest advantages is its ability to significantly reduce development time. It comes with a vast array of built-in features that allow developers to implement common functionalities without writing them from scratch. For example:
  - User Authentication System: Django provides a secure, extensible user authentication system out of the box. It supports login, logout, registration, password management, and user session tracking.
  - URL Routing: The URL dispatcher allows clear mapping between user-accessible URLs and the underlying view functions, making the website navigation logical and intuitive.
  - Form Handling: Django offers built-in form classes for rendering forms and validating inputs, which simplifies processes like user login, post creation, and feedback submission.
  - Static Files Management: Django simplifies the management and linking of static assets like CSS, JavaScript, and images—crucial for building a user-friendly frontend.
- Security: Security is a top priority for any application dealing with user-generated data. Django provides protection against the most common security threats:
  - SQL Injection: The ORM escapes inputs, making it virtually impossible for attackers to manipulate queries.
  - Cross-Site Scripting (XSS): Django templates auto-escape variables by default, preventing malicious JavaScript injection.
  - Cross-Site Request Forgery (CSRF): Django uses CSRF tokens to protect forms and sensitive operations from unauthorized access.
  - Clickjacking Protection: By setting proper HTTP headers, Django prevents framing attacks that trick users into clicking hidden buttons.
- Scalability and Reusability: Django encourages modular design through its app structure, where each component (like posts, messages, QR codes) is built as a self-contained app that can be reused or independently updated. This modularity provides:
  - Ease of Scaling: New features like event registration, alumni job boards, or profile badges can be added by creating new apps with minimal interference to the core system.
  - Code Reusability: Common utilities such as email notifications, form validations, or custom filters can be reused across multiple views and templates.

- Object-Relational Mapping (ORM): Django's ORM is one of its most valuable features. It abstracts the database layer, allowing developers to write Python code to query and manipulate data instead of SQL [18]. For instance, figure 2.2.1:

```
# Create a new post
Post.objects.create(author=user, content="Hello alumni!")

# Fetch messages sent by a user
messages = Message.objects.filter(sender=user)
```

*Figure 2.2.1 Code for Manipulating Data*

This simplifies database interaction, reduces the chance of errors, and ensures compatibility across different database backend. In MUJ Unite, the ORM was used extensively for storing user records, posts, messages, and QR code data.

- Templating Engine: The Django templating engine allows dynamic data to be embedded in HTML using simple and readable syntax. For example, refer figure 2.2.2:

```
<h2>Welcome, {{ user.username }}</h2>
<p>You have {{ messages.count }} new messages</p>
```

*Figure 2.2.2 Syntax for Django Templating Engine*

MUJ Unite's frontend pages' use templates to render content like:

- User dashboards
- List of posts and announcements
- Individual profiles with QR codes.

The templating engine plays a central role in delivering an interactive and personalized experience for each user.

- Django's Role in MUJ Unite: In the context of MUJ Unite, Django served as the backbone of the entire application. It was responsible for:
  - Handling user registration, authentication, and profile management.

- Allowing students, teachers, and alumni to post, like, or comment on updates.
- Supporting messaging and announcements, where users could interact privately or receive important updates.
- Generating and displaying QR codes linked to each user's profile.
- Providing administrators with full access through the Django Admin Panel to manage content and users efficiently.

## » *Django Admin Panel*

One of the most helpful tools I used during the development of MUJ Unite was Django's built-in admin interface. This web-based dashboard allowed me to view, edit, and organize all the key data in the application—such as user accounts, announcements, posts, and messages—without having to build a separate backend panel from scratch.

In short, the Django admin panel saved a significant amount of development time and gave me full control over the backend during the build and testing phases of MUJ Unite. It turned out to be an essential tool for keeping the platform organized and functional.

- Overview of Django Admin: In the MUJ Unite project, I made full use of Django's ability to auto-generate admin tools from the models I defined. By registering each model—such as users, posts, messages, and QR records—in the `admin.py` file, Django automatically created an interface that let me add, edit, and delete records through a web dashboard. This meant I didn't have to build a custom content management system to handle the database.

The admin panel supports complete CRUD operations, and I was able to perform all of them right through the browser. Whether I needed to update a user's role or delete a test post, I could do it without writing a single SQL query.

To secure the admin panel, Django required the creation of a superuser account using its command-line tools. After logging in at the `/admin/` route with this account, I gained access to all registered models and could manage them through a password-protected interface. This built-in security feature helped ensure that only authorized users could make changes to important site data.

Using Django's admin panel in this way gave me a major advantage during development. It allowed me to test features quickly, manage database records efficiently, and focus more on building user-facing functionality rather than backend dashboards.

- Role in MUJ Unite Project: In the context of MUJ Unite—a portal designed to connect students, alumni, and faculty—the admin panel was an essential part of the backend operations. Here are the primary roles it fulfilled:
  1. User Management: User accounts form the core of any web portal. In MUJ Unite, users belong to different roles such as students, alumni, and administrators.

The Django Admin Panel is allowed for:

- Adding New Users: Admins could manually create user accounts through the admin interface, assign them roles, and set permissions.
- Editing User Profiles: If a user needed to update their name, graduation year, contact info, or any field in their profile, admins could do so directly without involving the database or frontend code.
- Deactivating/Deleting Users: In cases of misconduct or spam, accounts could be deactivated or deleted to maintain a healthy digital environment.
- Role Assignment: By customizing the Django user model or extending it with user groups and permissions, the admin panel allowed the project team to define user roles (e.g., alumni, student, admin) and give access to different parts of the platform accordingly.

2. Content Moderation: In community-driven platforms like MUJ Unite, content moderation is critical. Users can post updates, share thoughts, and message each other. The admin panel enabled the moderation of this user-generated content through:

- Monitoring Posts: All user posts appeared in the admin panel as records, allowing moderators to view their content, author, timestamp, and status.
- Editing or Deleting Inappropriate Content: If any post violated community guidelines, admins could delete or edit it immediately from the admin interface.
- Flagged Content (optional customization): In advanced versions of the portal, a feature can be implemented where users report inappropriate content, and the flagged entries are highlighted in the admin dashboard for review.

This streamlined the moderation process and helped ensure that the community space remained respectful and constructive.

3. Announcement Management: One of the unique features of MUJ Unite is the ability for admins and faculty to broadcast announcements to all users. These could include:

- College event announcements
- Alumni meet updates
- Placement drives
- Holiday notices

The admin panel allowed authorized users to create, edit, publish, and delete announcements. A simple model (Announcement) was defined with fields like title, content, and date published, and once registered in the admin, it could be managed like any other database entry.

This eliminated the need for building a custom CMS (Content Management System) for posting announcements, saving both development time and resources.

4. Data Management and Integrity: Often, administrators need to correct data inconsistencies or manually input information. Django Admin made this possible by:

- Direct Data Access: Admins could browse, sort, and filter through large volumes of data using Django's search and filtering features.
- Batch Operations: Django's admin supports batch updating or deleting of records, useful in scenarios like deleting spam posts or activating a group of new users.
- Audit Trails (optional with extensions): With third-party packages like Django-simple-history or Django-auditlog, admins can track changes made to records for accountability.

This helped maintain data accuracy and reliability across the platform.

5. QR Code and Message Management: MUJ Unite includes features like QR code generation for user profiles and private messaging. Through the admin panel, administrators were able to:

- View and Regenerate QR Codes: If a user's QR code was lost or corrupt, it could be regenerated manually through the admin interface.
- Review Message Logs: For disciplinary or debugging purposes, admins could view past message records and handle complaints if needed.

This provided administrative oversight over crucial platform functionalities.

○ Customization of Admin Panel: Django allows developers to extensively customize the admin panel to suit project-specific needs. In MUJ Unite, some of the enhancements included:

- List Display Customization: Specifying which fields to show in the list view (e.g., usernames, post timestamps) for quicker access.
- Search Fields: Enabling search functionality within models, such as searching users by email or filtering posts by keywords.
- Model Inline Editing: Allowing related models (e.g., comments under a post) to be edited inline on the same page.
- Permission-based Access: Restricting access so that only certain users (e.g., faculty or admins) could post announcements or moderate users.

These enhancements improved usability and made the panel more tailored to the portal's unique requirements.

## » *SQLite Database*

For the MUJ Unite platform, I chose SQLite as the database solution because of its simplicity, low setup overhead, and strong compatibility with Django. One of the key benefits of using SQLite is that it doesn't rely on a server to function. Instead, the entire database lives in a single file, which makes it easy to manage and move between development environments—something especially useful in a student project with limited infrastructure.

Unlike traditional systems like MySQL or PostgreSQL that require server setup and configuration, SQLite runs directly from the file system. This allowed me to start building features right away without worrying about running background services or setting up user privileges. The database file (.db) automatically handled data storage for users, messages, announcements, and other content, all from within the Django project folder [20].

Since Django comes pre-configured to work with SQLite, I didn't have to spend time connecting external tools or writing SQL manually. Using Django's model system, I defined each table as a Python class. For example, I created models for alumni profiles, private messages, QR code data, and announcement posts. Django's Object-Relational Mapping (ORM) then handled everything in the background—from table creation to running queries—making the development process much faster.

I also took advantage of features like one-to-many and many-to-many relationships between models. These allowed me to connect users to posts, feedback to announcements, and even messages to specific user accounts. I didn't need to write raw SQL JOINS or worry about foreign key constraints; Django and SQLite handled all of it automatically.

Another practical advantage was SQLite's performance on low-resource machines. Since it doesn't need much memory or CPU power, I was able to run the entire project locally without any slowdown—even while managing multiple models and testing various features. SQLite also ensures data reliability through ACID compliance, so I never had to worry about losing data during testing or crashes.

For debugging, I appreciated how easy it was to inspect the database using tools like DB Browser for SQLite. The file-based setup made it simple to track down data issues or verify entries without setting up third-party systems.

Although SQLite is ideal for lightweight projects and development work, it won't be a limiting factor for MUJ Unite in the future. Django allows a seamless switch to more robust databases like PostgreSQL or MySQL when the time comes. The same models and queries will continue to work, with just a minor change in the database settings.

In short, SQLite provided a fast, simple, and stable foundation for database management during MUJ Unite's development. It let me focus on functionality rather than infrastructure and made collaboration with others easier thanks to its file-based structure and minimal setup requirements.

- Why SQLite for MUJ Unite?

The decision to use SQLite in the MUJ Unite project was driven by several practical and technical considerations. As a university-level web portal built for managing student, alumni, and faculty interactions, MUJ Unite required a reliable but easy-to-maintain database solution that could evolve with the project during development and testing phases. SQLite provided the perfect balance of functionality, simplicity, and flexibility.

Key Advantages of using SQLite: Some of the advantages are discussed below:

1. Lightweight and File-based: One of SQLite's defining characteristics is that it stores the entire database in a single file with a .db extension. This means that the complete dataset—including tables, indexes, and the schema—is contained in a portable file that can be easily copied, backed up, or transferred between systems.

For MUJ Unite, this portability was especially beneficial during the development and testing stages, where the database needed to be frequently migrated between systems or shared among team members.

2. Seamless Integration with Django ORM: Django's built-in Object-Relational Mapper (ORM) supports SQLite out of the box. This allows developers to define models in Python, which Django automatically translates into SQL statements to create the necessary database tables. CRUD (Create, Read, Update, Delete) operations are performed using python code rather than raw SQL.

In the MUJ Unite project, models were created for users, posts, messages, QR codes, announcements, and more. Each of these models was directly mapped to a table in the SQLite database. The ORM allowed:

- Automatic creation and migration of database schema
- Easy querying of data using python
- Use of relationships (e.g., foreign key) to link models such as users and their posts or messages
- Data validation and field constraints directly in model definitions

This tight integration helped ensure consistency between the application logic and the database schema, reducing errors and improving maintainability.

- Data stored in MUJ Unite using SQLite: SQLite served as the backbone for all persistent storage in MUJ Unite.

The key data components managed through the database included:

#### 1. User Data:

All registered users, including students, alumni, and admins, were stored in the User model. This included usernames, emails, passwords (hashed for security), user types, and related profile information. Authentication and session management were also dependent on this data.

#### 2. Posts and Interactions:

MUJ Unite featured a posting system where users could share messages or updates with the community. These were stored in a Post model with fields for content, timestamps, and the associated user. If implemented, likes or comments could be stored in related models linked through Foreign Key fields.

3. Messaging System: Users could send messages to one another, forming a simple messaging system. The Message model handled sender-receiver relationships, timestamps, and message content. These records were essential for enabling communication within the portal.



4. QR Code Metadata: For the QR code feature—used for linking to user profiles or for identification—associated metadata such as the encoded text or file path was stored in a QR code model. This ensured that the system could regenerate or retrieve codes efficiently.

5. Announcements: Faculty or administrators could post announcements to the portal, which were stored in the Announcement model. These entries included a title, body, and publication date, and were displayed to users on the homepage or announcements page.

6. Feedback and Support: If implemented, a feedback form allowed users to submit queries or suggestions. These were stored in a Feedback model and could be accessed by admins for response or analysis.

- Future Scalability and Transition: While SQLite is excellent for development and small-scale applications, Django’s database abstraction layer ensures that it can be easily upgraded to more robust databases like PostgreSQL or MySQL if needed.

For example, MUJ Unite is scaled to support thousands of users or hosted in a production-grade environment with concurrent requests and large data volumes, a more scalable RDBMS would be advisable. In such a case, the transition is relatively seamless.

- The models and ORM queries remain unchanged.
- Only the database engine and connection settings in settings.py need to be updated.
- Data migration tools like dump data and load data can be used to transfer records.

This ensures long-term sustainability and scalability of the project without locking developers into a single database solution.

SQLite played a foundational role in the development of the MUJ Unite portal. Its lightweight, server less nature made it ideal for academic and prototype projects, while its tight integration with Django enabled fast and consistent development. By handling essential data such as user accounts, posts, messages, announcements, and QR metadata, SQLite provided a reliable and flexible backend during all phases of development.

The database table is given on the next page as table 2.2.1:

***Table 2.2.1 Database Table***

<b>Model</b>	<b>Attributes (Fields)</b>
User (auth model)	username, email, password, is_active, is_staff
Profile	user (FK), bio, image, qr_code, contact_number
Post	user (FK), content, timestamp, likes, comments
Message	sender (FK), receiver (FK), message_text, timestamp
Announcement	title, content, posted_by, timestamp
Contact	name, email, message, timestamp
Verify / Active	Custom tables for email/phone verification and user activity tracking

## Chapter 3

### Methodology

#### 3.1 Detailed Methodology

When I started working on MUJ Unite, I decided to go with the Agile development method. It just felt like the right choice because it allowed me to split the whole project into smaller, more manageable tasks. That made everything a lot less stressful, and I could make changes on the go whenever I faced any technical issues or got new ideas.

Instead of trying to finish the entire project in one long stretch, I worked in short cycles. In each cycle, I picked a few features to work on—like the login system, the chat function, or profile updates—and once they were done, I tested them right away. This helped me catch problems early and fix them before moving on to the next part.

Working this way also kept me focused. I always had a clear goal for each phase, which made it easier to track progress and not feel overwhelmed. If something didn't work properly, I could simply tweak it in the next round rather than redoing everything from scratch. Agile made it easier to stay organized, handle issues quickly, and improve the portal bit by bit as I built it.

#### » Agile Methodology in Context of MUJ Unite

To manage the development of MUJ Unite efficiently, I adopted the Agile approach, which helped me to stay flexible and focused at the project success. Rather than trying to build everything all at once, we worked in short, focused phases—called sprints—where each sprint aimed to complete specific features or improvements [21].

At the end of every sprint, we had a working portion of the platform that we could test, review, and improve. This ongoing cycle of building and refining allowed us to adjust plans when new ideas came up or if something didn't work as expected. Since the project scope evolved during development—especially with feedback from mentors and changes in team direction—this method suited us well.

Here's a breakdown of how we structured our sprints and what tasks were discussed in table 3.1.1:

***Table 3.1.1 Agile Sprint Table for MUJ Unite Development***

<b>Sprint</b>	<b>Duration</b>	<b>Key Tasks &amp; Features Developed</b>	<b>Tools/Technologies Used</b>
Sprint 1	Week 1	- Requirement gathering - Planning core portal structure - Environment setup	VS Code, Python, Django

Sprint 2	Week 2	- User Authentication (signup/login/logout) - User model and profile structure	Django, SQLite
Sprint 3	Week 3	- Profile editing and display - Role assignment (student, alumni, admin) - Admin access	Django Admin, HTML/CSS, Bootstrap
Sprint 4	Week 4	- Posting system (create post, like/unlike, delete) - Designing feed layout	Django Views, Templates, JavaScript
Sprint 5	Week 5	- Messaging between users - Initial testing of message delivery and inbox UI	Django Models, Forms, SQLite
Sprint 6	Week 6	- Announcements (create via admin, display publicly) - Feedback form	Django Admin, Forms, Templates
Sprint 7	Week 7	- QR code generation for user profiles - Image storage using Pillow	QR code library, Pillow, Django Media
Sprint 8	Week 8	- Final testing & debugging - UI polishing and integration - Documentation	Postman, Manual Testing, GitHub

Each sprint in our development cycle ended with a brief review and testing session. For example, after completing Sprint 4, we tested the post creation and viewing feature using different types of accounts to confirm that access permissions were being enforced correctly. In Sprint 5, we ran into a problem where message timestamps weren't displaying accurately, especially in chats between alumni and students. We identified the issue and resolved it in the following sprint. This pattern of testing, fixing, and improving kept our development cycle productive and helped maintain the stability of MUJ Unite throughout the process.

One of the strengths of our approach was that we used modular development. Django supports this by letting developers split a project into smaller apps. We used this structure to separate core functionalities—like user management, messaging, announcements, and posts—into different Django apps. This division not only simplified debugging but also allowed us to work on multiple modules at the same time without causing code conflicts.

The Django admin panel was another tool that supported our workflow. It allowed us to quickly create and manage user accounts, test data entries, and model changes. Instead of building a custom dashboard, we relied on the default admin interface to handle many of the testing and content management tasks during development.

Our testing process combined manual checks and tool-assisted methods. For backend testing, we used Postman to test API endpoints and verify login and message submission behavior. On the frontend

side, we reviewed how pages looked and functioned across various screen sizes. During these reviews, we collected feedback from both teammates and faculty members, which we then used to make targeted improvements before the next sprint.

Throughout the project, we documented key elements such as user stories, database models, and design decisions.

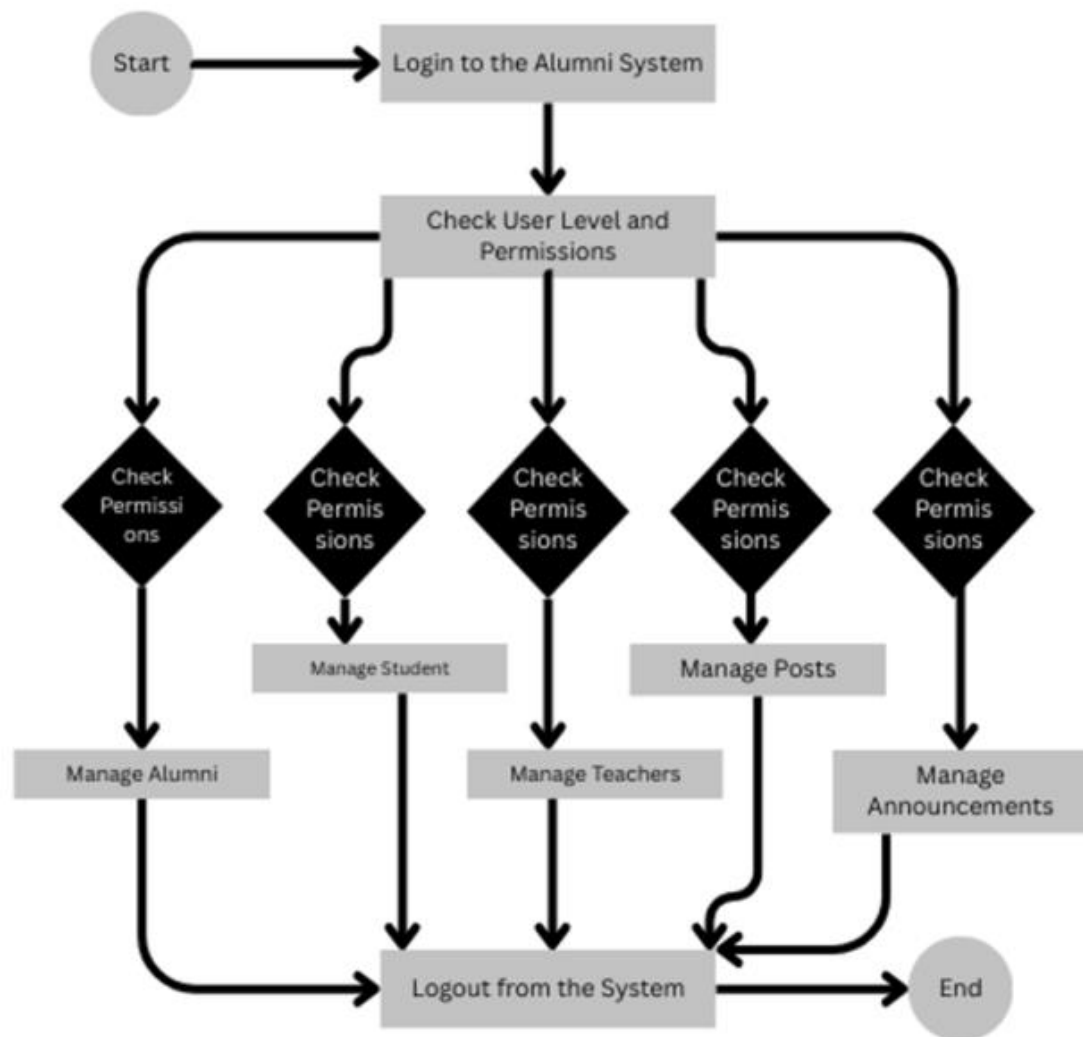
### *3.2 Block Diagrams*

#### » Activity Diagram – MUJ Unite

When I was mapping out how MUJ Unite would work, I created an activity diagram to show how different users—like students, alumni, and teachers—interact with the main features of the site. It's basically a visual guide that breaks down the steps each user typically takes while using the platform, from the moment they log in to how they engage with different parts of the system [22].

For alumni, I focused the flow on the things they'd most likely do after logging in. They can update their profile, check announcements, connect with others, and post updates. I also included options for them to share job opportunities or mentor students, which I thought would be really valuable. This kind of activity helps build a lasting bond between former students and the university, which was one of my key goals.

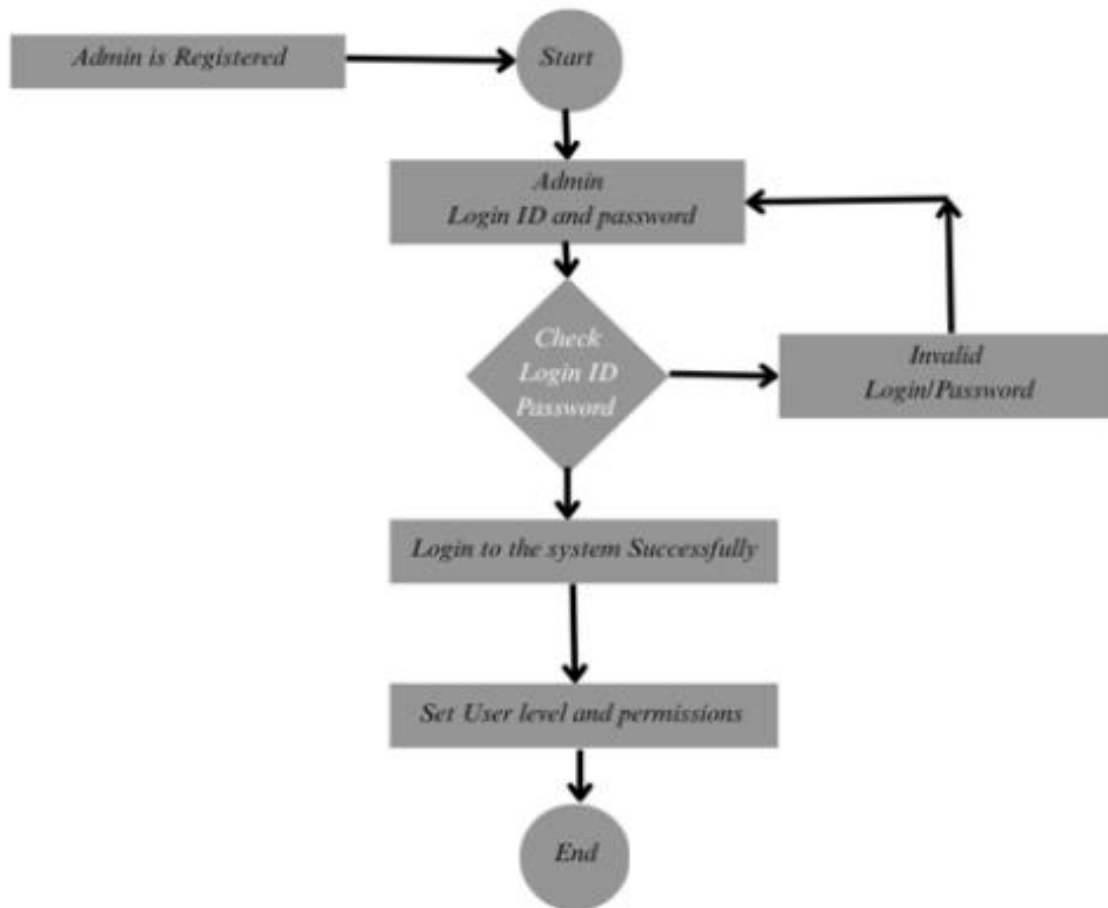
The Activity Diagram for MUJ Unite is given below:



*Figure 3.2.1 Activity Diagram*

» Login Activity Diagram - MUJ Unite

The Activity diagram for the admin login is discussed in next page:



*Figure 3.2.2 Login Activity Diagram*

» Data Flow Diagram –MUJ Unite

A DFD is a important tool for flow of the information between the various types of modules. This makes it easier to see how the system works as a whole without needing to get into the actual code or technical structure [23].

In the early stages of designing MUJ Unite, the DFD helped me to map out the system's key processes—like user registration, login, posting content, and managing announcements. It showed how these processes interact with various data elements such as user credentials, messages, and profile updates.

One of the reasons we used a DFD was to make the design easier to understand for everyone involved, including faculty mentors and teammates who might not be focused on the coding side. By focusing on the data's path through the system, we could quickly explain how different parts of the portal would work together and spot any missing elements before starting development.

- Key Components of a DFD: Key components of DFD are discussed in next subsections. A DFD uses four primary symbols: The four symbols are discussed below:
  - External Entity: Represented as a rectangle, it denotes sources or destinations of data that are outside the system. These could be users, other systems, or external organizations.
  - Process: Represented by a circle or rounded rectangle, it shows the transformation or manipulation of data. A process receives input data and produces output data.
  - Data Store: Represented as an open-ended rectangle or two parallel lines, it indicates where data is stored for use at a later time (e.g., databases, files).
  - Data Flow: Represented by arrows, it indicates the movement of data between entities, processes, and data stores.

Each of the components plays an important role in modelling how information is handled by a system.

- Types of DFDs

DFDs are organized into hierarchical levels to represent the system in increasing levels of detail.

- Level 0 DFD
  - Level 1 DFD
  - Level 2 DFD
- Level 0 Data Flow Diagram (DFD):

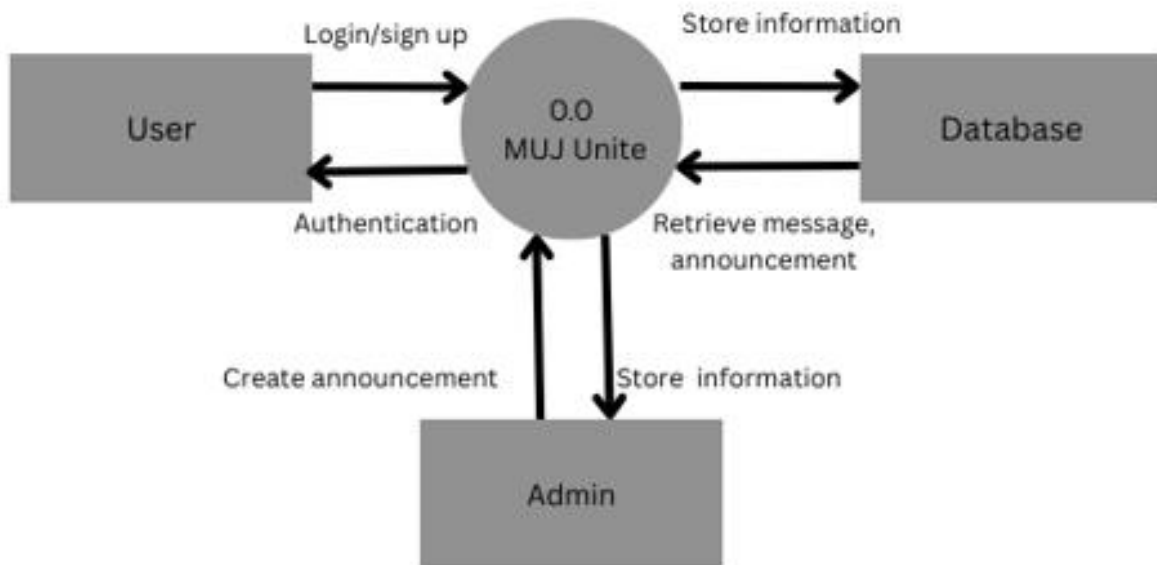
The Level 0 DFD, also called the context diagram, gives a basic snapshot of how the MUJ Unite platform works from the outside. It doesn't go into the technical stuff or what's happening behind the scenes—instead, it just shows the whole system as one big process and how it connects with different users. For example, students submit login credentials, alumni post updates, and admins manage announcements—all of which are visualized as inputs and outputs between the system and its users.

The main goal of this diagram is to define the system boundaries. It helps clarify which parts of the process are handled within the platform, and which interactions come from external sources. At this stage, we avoid technical details like data storage or internal sub-processes—keeping the focus on understanding who interacts with the system and what kind of data is exchanged.

This high-level view was especially useful during the early design stages of MUJ Unite. It gave both our team and project mentors a clear sense of the system's purpose, how it connects to different user roles, and what kind of information it processes. With this foundation in place, we were able to move on to more detailed diagrams that broke down the system further.

The Context Level DFD for the MUJ Unite is given below:





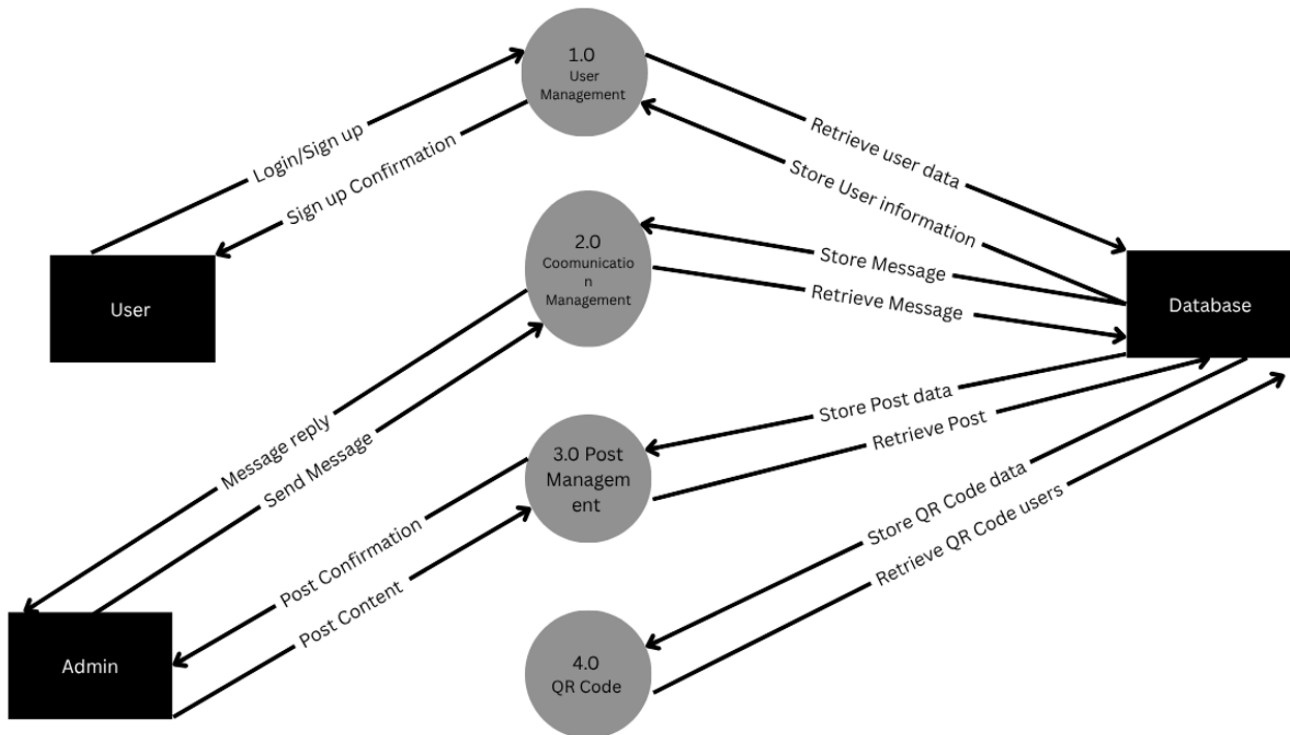
***Figure 3.2.3 Zero Level DFD***

- Level 1 Data Flow Diagram (DFD):

The Level 1 Data Flow Diagram for MUJ Unite provides a deeper view of how the platform operates by breaking down the overall system into its major functional parts. Unlike the high-level context diagram (Level 0), which shows the system as a single block, this diagram introduces multiple internal processes and data stores that handle specific tasks within the application.

At this level, each main function of MUJ Unite is treated as a separate process. Examples include user login and registration, profile management, creating and viewing posts, sending messages, and publishing announcements.

The Level 1 DFD for the MUJ Unite is given below:



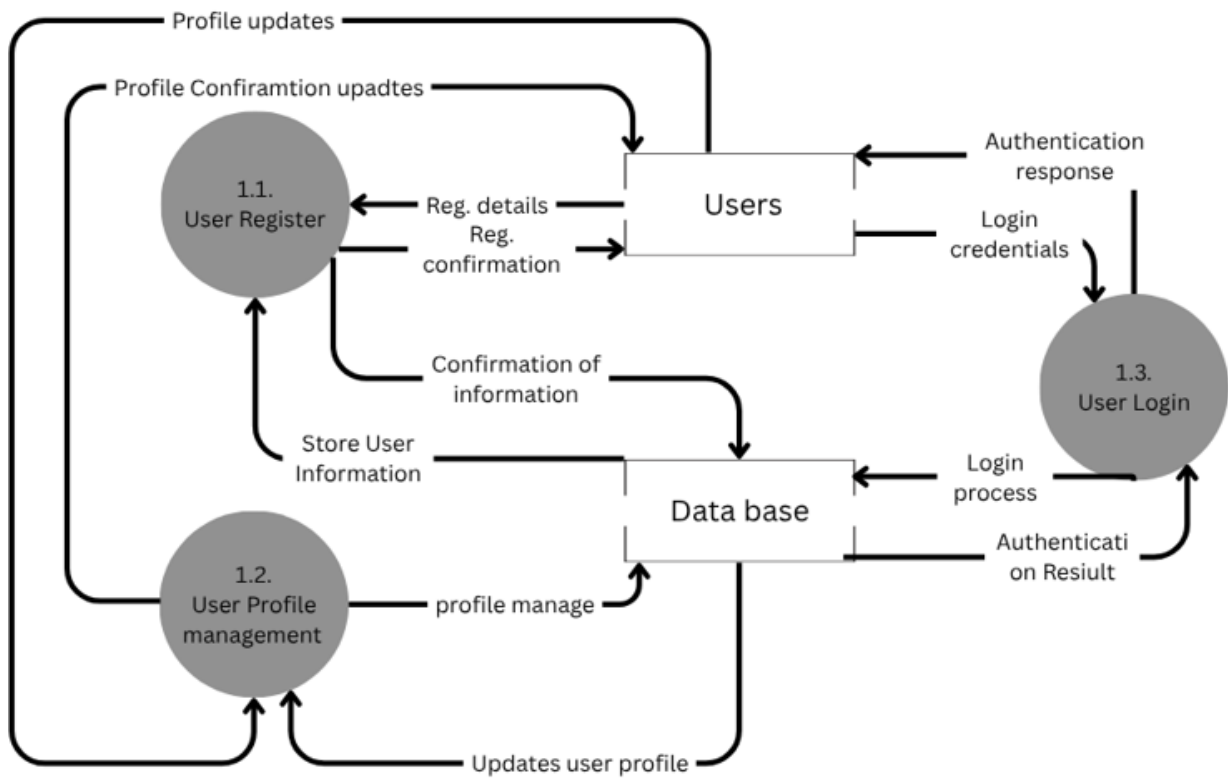
**Figure 3.2.4 Level 1 DFD**

- Level 2 Data Flow Diagram (DFD):

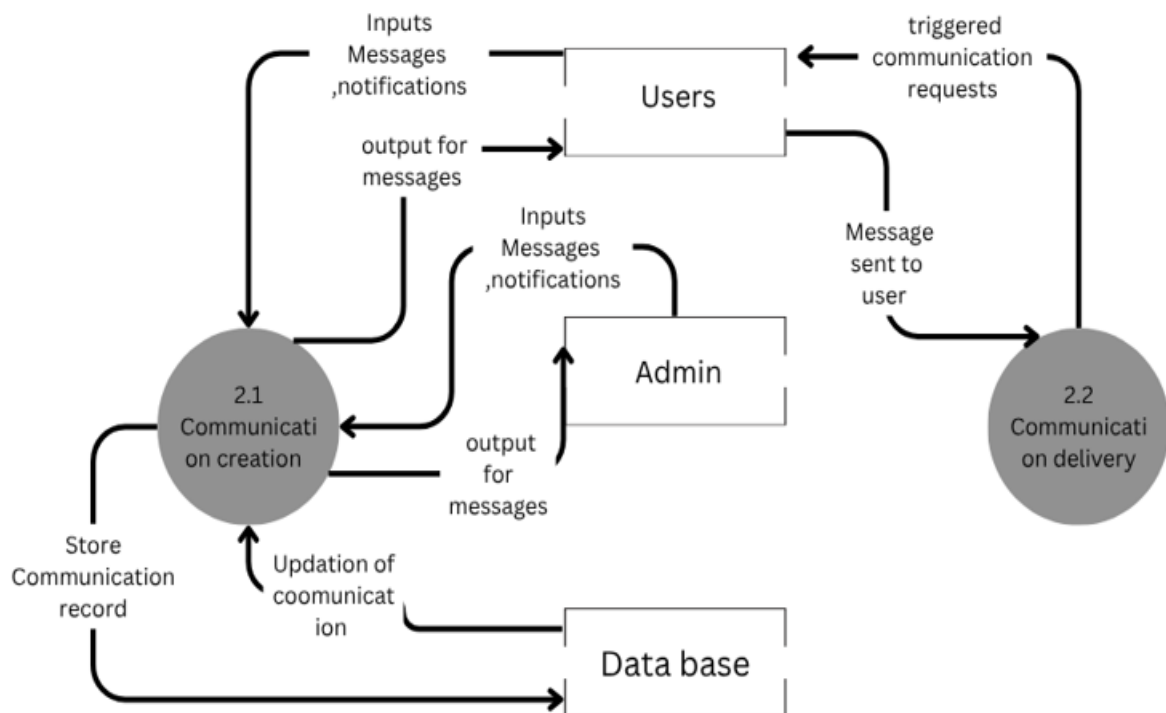
The Level 2 Data Flow Diagram basically goes deeper than Level 1 by breaking bigger tasks into smaller, more detailed steps. It gives a much clearer picture of how the actual operations work inside the system, especially when there are several decisions or actions involved.

In MUJ Unite’s case, I used Level 2 DFDs for parts like user sign-up and making posts. For example, instead of just saying “User Registration” like in Level 1, this level shows every step—checking if an email is already taken, making sure the password fits the rules, encrypting it, and then saving all that to the database. Every one of those steps was its own mini process in the diagram, which helped a lot when building and organizing things.

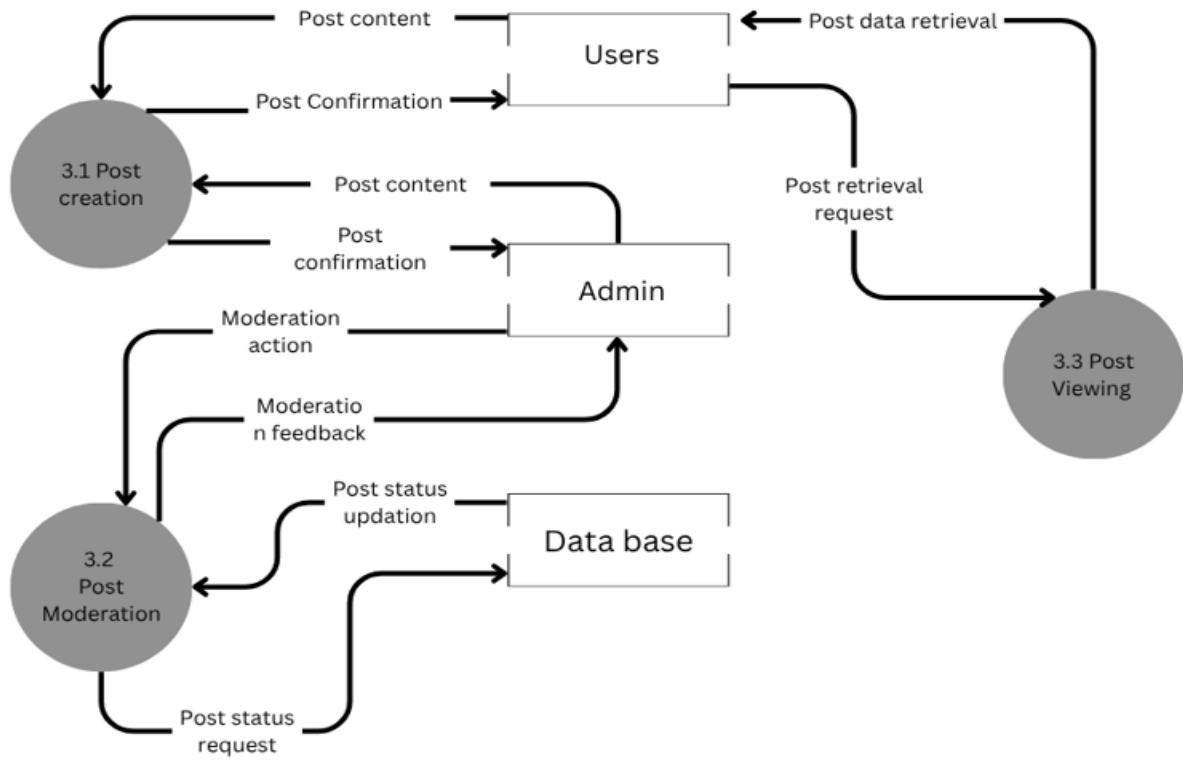
The Level 2 DFD for the MUJ Unite is given below:



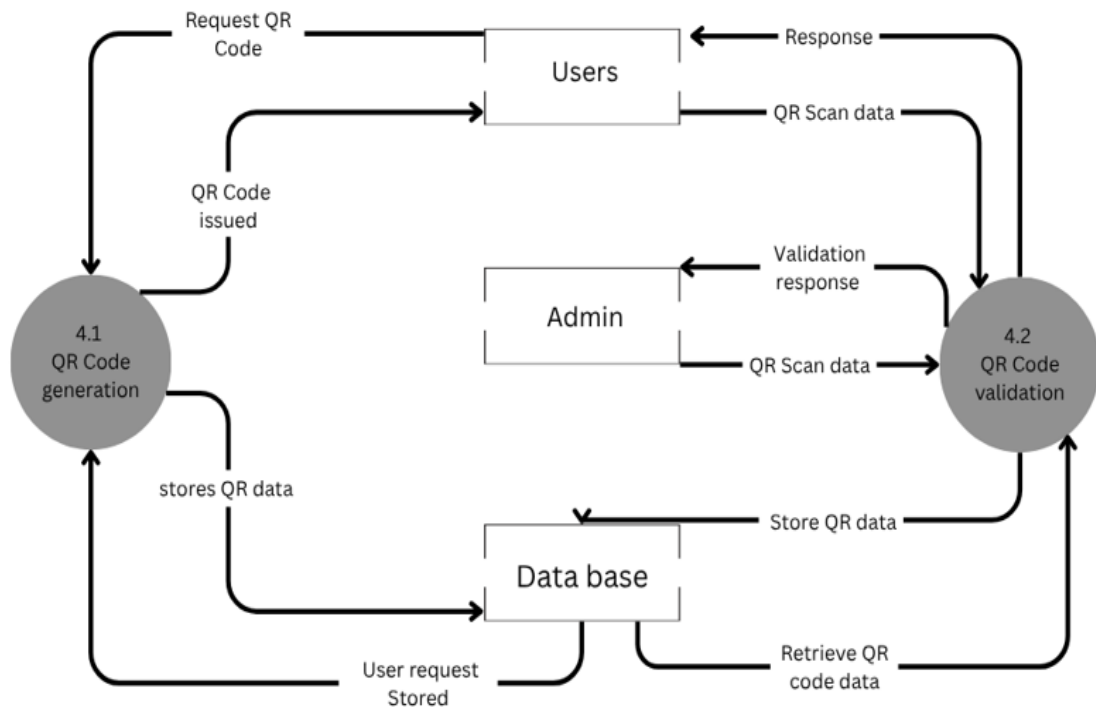
**Figure 3.2.5 User Management Module Level 2 DFD**



**Figure 3.2.6 Communication Management Module Level 2 DFD**



**Figure 3.2.7 Post Management Module Level 2 DFD**



**Figure 3.2.8 QR Code Module Level 2 DFD**

## » Entity Relationship Diagram- MUJ Unite

While working on MUJ Unite, one of the first things I did was create an Entity Relationship Diagram (ERD) to figure out how the database should be structured. It really helped me lay out all the important pieces of data—like users, posts, messages, and announcements—and how they're connected to each other.

I also added key fields to the user section, like their email, role (student, alumni, or teacher), and profile details. This matched up with what I needed for features like login, profile viewing, and messaging. Overall, designing the ERD early on made the backend a lot easier to manage because I already had a clear idea of how everything was supposed to fit together. [24].

The diagram helped clarify which pieces of data belonged together and which ones needed to be linked across different parts of the system. By defining relationships—such as alumni interacting with student posts or teachers managing announcements—we could design a database that supported all the key features of MUJ Unite efficiently.

This planning step made it easier to build models in Django later on, as the ERD acted like a blueprint for our database schema. It also helped us avoid redundancy, identify potential data issues early, and create a more reliable and scalable structure for the platform.

### ○ Purpose of an ERD

An ERD helps in planning and organizing data before it is actually implemented in a database. It serves as a blueprint that guides developers, database administrators, and analysts. In systems like alumni portals, e-commerce platforms, banking systems, or school management systems, an ERD ensures that the data relationships are clearly defined, consistent, and efficient.

### ○ Key Components of an ERD

ERDs use specific symbols to represent entities, attributes, and relationships:

- Entity: An entity is an object or concept about which data is stored. Entities are represented by rectangles. Examples include User, Post, Message, or Event. Entities can be:
  - Strong Entities: Exist independently (e.g., Student).
  - Weak Entities: Depend on other entities (e.g., Vote).
- Attribute: Attributes are the data fields that describe an entity. They are represented by ellipses (ovals). For example, the User entity may have attributes like User ID, Name, Email, and Role.
- Primary Key: An attribute that uniquely identifies each instance (e.g., User ID).
- Foreign Key: An attribute that links one entity to another (e.g., User ID in the Post entity).
- Relationship: A relationship shows how two or more entities interact. Relationships are represented by diamonds connecting the entities. For instance, a User “creates” a Post, or a Student “votes” in an Election.

## Chapter 4

### Implementation

#### 4.1 Modules

The MUJ Unite portal is composed of several core modules that handle various functionalities, ensuring smooth interaction among students, alumni, teachers, and administrators. Each module is developed using Django for the backend and HTML, CSS, and JavaScript for the frontend [25]. Key modules of the system are discussed below in table 4.1.1, table 4.1.2 & table 4.1.3:

**Table 4.1.1 Implementation of each Module**

Module	views.py Function	Template File	Model Used
Home	home(request)	home.html	Announcement, Post
Login	login user(request)	login.html	User
Sign Up	signup user(request)	signup.html	User, Verify
Profile	profile(request)	profile.html	Profile, User
QR Code	Generated in profile view	Embedded in profile.html	Profile
Messaging	message(request)	message.html	Message, User
Announcements	announcement(request)	anouncement.html	Announcement
Contact Us	contact(request)	contact.html	Contact
About Us	about(request)	about.html	Static (No model)
Change Password	change password(request)	change_password.html	User

**Table 4.1.2 Users Permissions**

<b>Users</b>	<b>User Roles and Permissions</b>
Student	View announcements Send messages View profile
Alumni	View announcements Post Messages Edit profile View profile
Admin	Post and delete announcements View all users Moderate content Access contact from submissions

**Table 4.1.3 Modules listed in MUJ Unite**

<b>Module Name</b>	<b>Description</b>
Home	Landing dashboard after login; displays announcements and navigation links.
Login & Sign Up	User authentication and account registration functionality.
Profile	Displays user details and QR code, allows editing of profile information.
Messaging	Allows users to send and receive internal messages.
Announcements	Displays official posts, admins can create, update, and delete announcements.
Contact Us	A form to contact the admin with queries or feedback.
About Us	Provides information about the project, its purpose, and the team.
Change Password	Enables users to securely update their account password.

1. **Home Module:** The Home module is the central landing page of the platform. It provides the users with an overview of the latest posts, announcements, and other important updates. The following functionalities are part of the Home module:
  - **Display Recent Posts:** Fetch and display the latest posts made by users.
  - **Announcements Section:** Show important announcements relevant to the community.
  - **Navigation Links:** Provide easy access to other parts of the platform such as profile, messages, and notifications.

» Implementation Details:

- The views.py file contains a home function that fetches recent posts and announcements.
- The home.html template is used to display these details dynamically.

The code for the above module are as follows:

```
def home(request):
    if request.session.has_key('id') and (request.method!="POST"):
        obj=request.session['id']
        data2=user.objects.get(uid=obj)
        data4 = Profile.objects.filter(Pr_id=obj).first()
        data=announcement.objects.all().order_by("-id")[:3]
        post = Post.objects.all().order_by("-id")
        data6=user.objects.all()
        print("if")
        if data2.urole=="Teacher" or data2.urole=="Alumini":
            result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
            result1=user.objects.exclude(uid=obj).order_by("uname")
            return render(request,"home
(2).html",{"msg":data,"msg1":data2,"msg2":data4,"post":post,"All_user":result1,"All_profile":result2,"msg6":data6})
        else:
            result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
            result1=user.objects.exclude(uid=obj).order_by("uname")
            return
render(request,"home(1).html",{"msg":data,"msg1":data2,"msg2":data4,"post":post,"All_user":result1,"All_profile":result2,"msg6":data6})
    elif (request.method=="POST"):
        print("else")
        ptext=request.POST["pmsg"]
        obj=request.session['id']
        data2=user.objects.get(uid=obj)
        data4=Profile.objects.get(Pr_id=obj)
        data=announcement.objects.all().order_by("-id")[:3]
        data6=user.objects.all()
        post = Post.objects.all().order_by("-id")
        if 'ppost' in request.FILES:
            file_data = request.FILES["ppost"]
            fs2=FileSystemStorage()
            ppost=fs2.url(fs2.save(file_data.name,file_data))
            print(data4.Pr_id)
            pid=Profile.objects.filter(Pr_id=obj).only("Pr_id")
            Post.objects.create(P_author=pid.first(),P_content=ptext,P_img=ppost,P_name=data2.uname,P_name_img=data4.Pr_img)
        else:
            pid=Profile.objects.filter(Pr_id=obj).only("Pr_id")
            Post.objects.create(P_author=pid.first(),P_content=ptext,P_name=data2.uname,P_name_img=data4.Pr_img)
        if data2.urole=="Teacher" or data2.urole=="Alumini":
            result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
            result1=user.objects.exclude(uid=obj).order_by("uname")
```



```

        return render(request, "home
(2).html", {"msg":data, "msg1":data2, "msg2":data4, "post":post, "All_user":result1, "All_profile":result2, "msg6":data6})
    else:
        result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
        result1=user.objects.exclude(uid=obj).order_by("uname")
    return
render(request, "home(1).html", {"msg":data, "msg1":data2, "msg2":data4, "post":post, "All_user":result1, "All_profile":r
esult2, "msg6":data6})
, "msg6":data6})

```

2. Login and Sign-Up Module: This module handles user authentication, allowing users to either log in or sign up.

- Sign-Up: Allows users to register by providing their username, email, and password.
  - Login: Allows users to log in using their credentials.
- Implementation Details:
- Sign-Up: Implemented in the signup view, which creates a new user and automatically logs them in upon successful registration.
  - Login: Implemented in the login view, which authenticates the user and logs them in as if the credentials are correct.

The code for the above module are given below:

```

def login(request):
    if request.method == "POST":
        arole = request.POST["member_level"]
        aid = request.POST["id"]
        apswd = request.POST["pass"]

        # Check if user exists with the provided uid and role
        if user.objects.filter(uid=aid, urole=arole).exists():
            data2 = user.objects.get(uid=aid, urole=arole)
            email = data2.uemail.lower()

            # 1. Email domain validation based on role
            if arole == "Teacher":
                if not email.endswith('@muj.jaipur.edu'):
                    messages.error(request, "Teachers must use an email ending with @muj.jaipur.edu.")
                    return render(request, 'login.html')
            elif arole in ["Student", "Alumini"]:
                if not email.endswith('@muj.manipal.edu'):
                    messages.error(request, "Students and Alumni must use an email ending with @muj.manipal.edu.")
                    return render(request, 'login.html')
            else:

```

```

messages.error(request, "Invalid role provided.")
return render(request, 'login.html')

# ✔ Password check using hashed password check
if check_password(apswd, data2.upswd):
    # Proceed with login session
    if active.objects.filter(aid=aid).exists():
        request.session['id'] = aid
    else:
        active(aid=aid).save()
        request.session['id'] = aid

messages.success(request, "Login successful")

# Fetch data
data = announcement.objects.all().order_by("-id")[:3]
data4 = Profile.objects.filter(Pr_id=aid).first()
data6 = user.objects.all()
post = Post.objects.all().order_by("-id")
result2 = Profile.objects.exclude(Pr_id=aid).order_by("Pr_name")
result1 = user.objects.exclude(uid=aid).order_by("uname")

# Redirect based on user role
if arole in ["Teacher", "Alumini"]:
    return render(request, "home (2).html", {
        "msg": data, "msg1": data2, "msg2": data4,
        "post": post, "All_user": result1,
        "All_profile": result2, "msg6": data6
    })
else:
    return render(request, "home(1).html", {
        "msg": data, "msg1": data2, "msg2": data4,
        "post": post, "All_user": result1,
        "All_profile": result2, "msg6": data6
    })
else:
    messages.error(request, "Incorrect password")
    return render(request, 'login.html')

else:
    messages.error(request, "Invalid user ID or role. Please check your details.")
    return render(request, 'login.html')

else:
    return render(request, 'login.html')

def signup(request):
    if request.method == "POST":
        urole = request.POST["member_level"]
        uname = request.POST["name"]
        uid = request.POST["id"]

```

```

uemail = request.POST["email"]
upswd = request.POST["pass"]
ucpswd = request.POST["cpass"]

# 1. Check if Name and ID match the verify table
if not verify.objects.filter(vid=uid, vname=uname).exists():
    messages.error(request, "***Only verified Manipalians can register (Name and ID must match).")
    return render(request, 'signup.html')

# 2. Enforce role-based email domain rules
if urole in ["Student", "Alumini"] and not uemail.endswith("@muj.manipal.edu"):
    messages.error(request, "***Students and Alumni must use an email ending with @muj.manipal.edu.")
    return render(request, 'signup.html')
if urole == "Teacher" and not uemail.endswith("@muj.jaipur.edu"):
    messages.error(request, "***Teachers must use an email ending with @muj.jaipur.edu.")
    return render(request, 'signup.html')

# 3. Confirm password match
if upswd != ucpswd:
    messages.error(request, "***Passwords do not match.")
    return render(request, 'signup.html')

# 4. Check for duplicates
if user.objects.filter(uid=uid).exists():
    messages.error(request, "***User ID already exists.")
    return render(request, 'signup.html')
if user.objects.filter(uemail=uemail).exists():
    messages.error(request, "***Email is already registered.")
    return render(request, 'signup.html')

# 5. Save user and create profile
data = user(urole=urole, uname=uname, uid=uid, uemail=uemail, upswd=make_password(upswd),
ucpswd=ucpswd)
data.save()
if not Profile.objects.filter(Pr_id=data).exists():
    profile = Profile(Pr_id=data, Pr_name=uname)
    profile.save()

messages.success(request, "***Signup successful. You can now log in.")
return render(request, 'login.html')

return render(request, 'signup.html')

```

3. Profile Module: The Profile module allows users to view and edit their personal profiles. The profile includes the information of the user name, user information. Additionally, this module supports:
  - Profile Picture Upload: Users can upload and update their profile pictures.

- **QR Code Generation:** A unique QR code is generated for each user, which can be scanned to access their profile.
- **Implementation Details:**
  - The Profile model stores user-related information.
  - The QR code is generated using the qrcode library, and it is displayed on the user's profile page.

The profile page is rendered using the profile.html template, and users can update their information via a form.

*Table 4.1.4 Details for QR Code*

Component	Description
Purpose	To generate a scannable QR code for each user that links to their profile.
Technology Used	Python QR code library, BytesIO, base64 for image encoding.
Integration Location	Integrated within the user profile view and displayed as an image.
Trigger	Automatically generated when the profile is created or updated.
Benefits	Easy profile access and sharing, especially during alumni events or meetups.

The code for the above module is given on next page:

```
def my_profile_view(request):
    print(request)
    if request.session.has_key('id') and request.method != "POST":
        # Retrieve profile and user data based on session
        obj = request.session['id']
        profile = Profile.objects.filter(Pr_id=obj).first() # Safe retrieval, returns None if no profile found

        current = user.objects.get(uid=obj)
        posts = Post.objects.filter(P_author=profile).order_by("-id")

        # Generate QR code for the profile URL
        profile_url = request.build_absolute_uri(reverse('my_profile_view'))
        qr_image = qrcode.make(profile_url)
        qr_buffer = BytesIO()
        qr_image.save(qr_buffer, format="PNG")
        qr_image_base64 = base64.b64encode(qr_buffer.getvalue()).decode("utf-8")
```

```

context = {
    'profile': profile,
    'current': current,
    'posts': posts,
    'qr_image': qr_image_base64,
}

return render(request, 'myprofile.html', context)

elif request.method == "POST":
    obj = request.session['id']
    obj2 = user.objects.filter(uid=obj)

    # Check if a profile picture is uploaded, otherwise use existing picture
    if 'picture' not in request.FILES:
        print("No new picture uploaded")
        obj3 = Profile.objects.get(Pr_id=obj)
        ppicture = obj3.Pr_img
    else:
        print("New picture uploaded")
        file_data = request.FILES["picture"]
        fs = FileSystemStorage()
        ppicture = fs.url(fs.save(file_data.name, file_data))

    # Retrieve form data
    pname = request.POST["name"]
    pemail = request.POST["email"]
    ptitle = request.POST["title"]
    pabout = request.POST["about"]

    print(ppicture)
    # Update Profile and User models with the new data
    Profile.objects.filter(Pr_id=obj).update(Pr_name=pname, Pr_title=ptitle, Pr_img=ppicture, Pr_about=pabout)
    user.objects.filter(uid=obj).update(uemail=pemail, uname=pname)
    announcement.objects.filter(anc_uid=obj).update(anc_img=ppicture)

    # Update posts with the new profile details
    profile = Profile.objects.get(Pr_id=obj)
    Post.objects.filter(P_author=profile).update(P_name=pname, P_name_img=ppicture)
    current = user.objects.get(uid=obj)
    posts = Post.objects.filter(P_author=profile).order_by("-id")

    # Generate QR code again after profile update
    profile_url = request.build_absolute_uri(reverse('profile_detail', kwargs={'idd': profile.Pr_id}))
    print("QR Code URL:", profile_url)
    qr_image = qrcode.make(profile_url)
    qr_buffer = BytesIO()
    qr_image.save(qr_buffer, format="PNG")
    qr_image_base64 = base64.b64encode(qr_buffer.getvalue()).decode("utf-8")

```

```

context = {
    'profile': profile,
    'current': current,
    'posts': posts,
    'qr_image': qr_image_base64,
}

messages.success(request, "Profile Successfully Updated")
return render(request, 'myprofile.html', context)

def profile_view(request, idd):
    if request.method == "GET":
        data1 = Profile.objects.get(Pr_id=idd)
        data2 = user.objects.get(uid=idd)
        posts = Post.objects.filter(P_author=data1).order_by("-id")

        # Generate QR code for the profile URL (replace with your own URL logic)
        profile_url = request.build_absolute_uri(data1.get_profile_url()) # Assuming you have a method to get profile URL
        qr_image = qrcode.make(profile_url)

        # Save the QR code as a base64 string
        qr_buffer = BytesIO()
        qr_image.save(qr_buffer, format="PNG")
        qr_image_base64 = base64.b64encode(qr_buffer.getvalue()).decode("utf-8")

        context = {
            'profile': data1,
            'userr': data2,
            'posts': posts,
            'qr_image': qr_image_base64, # Pass the QR code to the template
        }
        return render(request, "profile.html", context)
    else:
        return render(request, "view_ancmnt.html")

```

4. Messaging Module: The Messaging module enables users to communicate with each other through private messages. Users can send, receive, and read messages from other users on the platform.

The key features of this module include:

- Send Message: Users can send messages to other users.
- Inbox: Users can view their received messages.
- Message Notification: A notification is shown when a new message is received.

- Implementation Details:

- The Message model is used to store messages in the database.
- The inbox and compose message views handle message-related actions.
- inbox.html and compose.html templates render the messaging interface.

The code for the above module are as follows:

```
def Inbox(request):
    curr=request.session['id']
    messages = Message.get_messages(curr_user=curr)
    currentProfile=Profile.objects.filter(Pr_id=curr).first()
    active_direct = None
    active_direct_profile=None
    messages_count =0
    directs = None
    profile=Profile.objects.exclude(Pr_id=curr).order_by("Pr_name")
    allusers=user.objects.exclude(uid=curr).order_by("uname")
    if messages:
        message = messages[0]
        active_direct = message['user'].uid
        active_direct_profile=Profile.objects.filter(Pr_id=active_direct).first()
        directs = Message.objects.filter(m_user=curr, m_recipient=message['user'])|Message.objects.filter(
m_sender=message['user']).order_by("m_date")
        directs.update(m_is_read=True)
        messages_count = Message.objects.filter(m_user=curr, m_recipient=message['user']).count()
        messages_count=messages_count+Message.objects.filter(
m_sender=message['user']).order_by("m_date").count()

    print("Active Direct")
    print(active_direct)
    print("Directs")
    print(directs)
    print("Messages")
    print(messages)

    print("messages_count")
    print(messages_count)
    context = {
        'currentProfile':currentProfile,
        'directs': directs,
        'messages': messages,
        'active_direct': active_direct,
        'profile':profile,
        'allusers':allusers,
        'messages_count':messages_count,
        'active_direct_profile': active_direct_profile,
    }

    template = loader.get_template('direct.html')
    return HttpResponse(template.render(context, request))
```

5. Announcements Module: The announcements module allows administrators or authorized users to post announcements that are visible to all platform members.

The key features of this module include:

- Create Announcement: Admin users can create and post new announcements.
  - View Announcements: All users can view the list of announcements on the homepage.
- Implementation Details:
- The Announcement model is used to store the content of announcements.
  - The views.py file contains the logic for creating and displaying announcements.
  - Announcements are displayed using the announcements.html template.

The code for the above module are as follows:

```
def ancmnt(request):
    if request.session.has_key('id') and (request.method!="POST"):
        objj=request.session['id']
        data2=user.objects.get(uid=objj)
        context= {
            'info': data2,
        }
        return render(request,"add_ancmnt.html",context)
    elif request.method=="POST":
        objj=request.session['id']
        anc_uid=objj
        anc_dsc=request.POST["ancmnt"]
        obj=Profile.objects.get(Pr_id=anc_uid)
        anc_img=obj.Pr_img
        Announcement=announcement(anc_uid=anc_uid,anc_dsc=anc_dsc,anc_img=anc_img)
        Announcement.save();
        messages.success(request,'Announcement Has Been Made Successfully')
        data=announcement.objects.all().order_by("-id")[:3]
        data6=user.objects.all()
        data4=Profile.objects.get(Pr_id=objj)
        data2=user.objects.get(uid=objj)
        post = Post.objects.all().order_by("-id")
        if data2.urole=="Teacher" or data2.urole=="Alumini":
            result2=Profile.objects.exclude(Pr_id=objj).order_by("Pr_name")
            result1=user.objects.exclude(uid=objj).order_by("uname")
            return render(request,"home
(2).html",{"msg":data,"msg1":data2,"msg2":data4,"msg6":data6,"post":post,"All_user":result1,"All_profile":result2})
        else:
            result2=Profile.objects.exclude(Pr_id=objj).order_by("Pr_name")
            result1=user.objects.exclude(uid=objj).order_by("uname")
            return
render(request,"home(1).html",{"msg":data,"msg1":data2,"msg2":data4,"post":post,"msg6":data6,"All_user":result1,"
All_profile":result2})
        else:
```



```
messages.error(request, '*Please Check The Details')
return render(request, 'add_ancmnt.html')
```

```
def ancmnt_view(request):
    data=announcement.objects.all().order_by("-id")
    data2=user.objects.all()
    data3=User.objects.all()
    return render(request, "view_ancmnt.html", {"msg":data, "msg2":data2, "msg3":data3})
```

6. Contact Us: The Contact Us section in MUJ Unite gives users a way to get in touch with the platform admins if they have any questions, feedback, or problems. I felt this part was important because users should always have a direct way to reach out in case something isn't working right or if they simply want to ask something.

The key features of the module includes:

- Contact Form: A form in which users can input their name, email, and message.
- Implementation Details:
  - I created a basic HTML form for the Contact Us section, where users can enter their name, email, and message.
  - When the form is submitted, the details are handled by a function in views.py, which processes the data.
  - Depending on the setup, the message can either be saved in the database or directly emailed to the admin.

The code for the above module are as follows:

```
def contact_us_in(request):
    if request.method=="POST":
        cname=request.POST["name"]
        cemail=request.POST["email"]
        cmsg=request.POST["message"]
        Contact=contact(cname=cname,ceail=ceail,cmsg=cmsg)
        Contact.save();
        messages.success(request, 'Message has been sent Successfully')
    if request.session.has_key('id'):
        obj=request.session['id']
        data=announcement.objects.all().order_by("-id")[:3]
        data6=user.objects.all()
        data4=Profile.objects.get(Pr_id=obj)
        data2=user.objects.get(uid=obj)
        post = Post.objects.all().order_by("-id")
        result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
        result1=user.objects.exclude(uid=obj).order_by("uname")
        if data2.urole=="Teacher" or data2.urole=="Alumini":
```

```

        return render(request, "home
(2).html", {"msg":data, "msg1":data2, "msg2":data4, "msg6":data6, "post":post, "All_user":result1, "All_profile":result2})
    else:
        return
render(request, "home(1).html", {"msg":data, "msg1":data2, "msg2":data4, "msg6":data6, "post":post, "All_user":result1, "
All_profile":result2})
    else:
        return render(request, 'contact_us_in.html')

def contact_us_out(request):
    if request.method=="POST":
        cname=request.POST["name"]
        cemail=request.POST["email"]
        cmsg=request.POST["message"]
        Contact=contact(cname=cname, cemail=ceail, cmsg=cmsg)
        Contact.save();
        messages.success(request, 'Message has been sent Successfully')
        return render(request, 'first.html')
    else:
        return render(request, 'contact_us_out.html')

```

7. About Us: The About Us section gives visitors a clear idea of what the platform is all about— why it was created, what it aims to achieve, and who’s behind it. It plays an important role in building trust and helping users connect with the platform’s overall vision and goals.

- Implementation Details:

- The About Us page was built using a basic HTML file called about.html, which is connected to the backend through a Django view function defined in views.py.
- Most of the content on this page is written directly into the HTML file, but if there's ever a need to update it frequently, the info can also be pulled from the database using a model.

The code for the above module are as follows:

```

def home(request):
    if request.session.has_key('id') and (request.method!="POST"):
        obj=request.session['id']
        data2=user.objects.get(uid=obj)
        data4 = Profile.objects.filter(Pr_id=obj).first()
        data=anouncement.objects.all().order_by("-id")[:3]
        post = Post.objects.all().order_by("-id")
        data6=user.objects.all()
        print("if")
        if data2.urole=="Teacher" or data2.urole=="Alumini":
            result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
            result1=user.objects.exclude(uid=obj).order_by("uname")
            return render(request, "home
(2).html", {"msg":data, "msg1":data2, "msg2":data4, "post":post, "All_user":result1, "All_profile":result2, "msg6":data6})

```

```

else:
    result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
    result1=user.objects.exclude(uid=obj).order_by("uname")
    return
render(request,"home(1).html",{ "msg":data,"msg1":data2,"msg2":data4,"post":post,"All_user":result1,"All_profile":r
esult2,"msg6":data6})
elif (request.method=="POST"):
    print("else")
    ptext=request.POST["pmsg"]
    obj=request.session['id']
    data2=user.objects.get(uid=obj)
    data4=Profile.objects.get(Pr_id=obj)
    data=announcement.objects.all().order_by("-id")[:3]
    data6=user.objects.all()
    post = Post.objects.all().order_by("-id")
    if 'ppost' in request.FILES:
        file_data = request.FILES["ppost"]
        fs2=FileSystemStorage()
        ppost=fs2.url(fs2.save(file_data.name,file_data))
        print(data4.Pr_id)
        pid=Profile.objects.filter(Pr_id=obj).only("Pr_id")
        Post.objects.create(P_author=pid.first(),P_content=ptext,P_img=ppost,P_name=data2.uname,P_name_img=da
ta4.Pr_img)
    else:
        pid=Profile.objects.filter(Pr_id=obj).only("Pr_id")
        Post.objects.create(P_author=pid.first(),P_content=ptext,P_name=data2.uname,P_name_img=data4.Pr_img)
    if data2.urole=="Teacher" or data2.urole=="Alumini":
        result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
        result1=user.objects.exclude(uid=obj).order_by("uname")
        return render(request,"home
(2).html",{ "msg":data,"msg1":data2,"msg2":data4,"post":post,"All_user":result1,"All_profile":result2,"msg6":data6})
    else:
        result2=Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
        result1=user.objects.exclude(uid=obj).order_by("uname")
        return
render(request,"home(1).html",{ "msg":data,"msg1":data2,"msg2":data4,"post":post,"All_user":result1,"All_profile":r
esult2,"msg6":data6})

```

8. Change Password: The Change Password module give permission to authenticated users to securely update their account passwords. This is a critical component of user account management, helping maintain security and user autonomy.

The Key features of this module include:

- Password Validation: Verifies the current password before allowing the user to set a new one.
  - Matching Confirmation: Ensures that the new password and confirm password fields match.
  - Feedback Mechanism: Provides success or error messages based on the operation result.
- Implementation Details:

- The Change Password feature is implemented through a form rendered on a dedicated HTML page (change\_password.html), which includes fields for the old password, new password, and confirm new password.
- The form submission is handled in views.py, where Django queries the user model to verify the current password.
- If validation is successful, the password is updated in the database, and a success message is shown to the user.

The code for the above module are as follows:

```
def change_psw(request):
    if request.session.has_key('id'):
        obj = request.session['id']
        print(obj)
        data2 = user.objects.get(uid=obj)
        data1 = Profile.objects.filter(Pr_id=obj).first()

    if request.method == "POST":
        old_psw = request.POST["old_psw"]
        new_psw = request.POST["new_psw"]
        cnew_psw = request.POST["cnew_psw"]

    if user.objects.filter(uid=data2.uid, upswd=old_psw).exists():
        if new_psw == cnew_psw:
            data3 = user.objects.get(uid=data2.uid)
            data3.upswd = new_psw
            data3.save()
            messages.success(request, "Password Has Been Changed Successfully")
            data = announcement.objects.all().order_by("-id")[:3]
            data5 = Profile.objects.filter(Pr_id=obj).first()
            data4 = user.objects.all()
            post = Post.objects.all().order_by("-id")

            result2 = Profile.objects.exclude(Pr_id=obj).order_by("Pr_name")
            result1 = user.objects.exclude(uid=obj).order_by("uname")

        if data2.urole == "Teacher" or data2.urole == "Alumini":
            return render(request, "home (2).html", {
                "msg": data,
                "msg1": data2,
                "msg6": data4,
                "msg2": data5,
                "post": post,
                "All_user": result1,
                "All_profile": result2
            })
        else:
            return render(request, "home(1).html", {
                "msg": data,
                "msg1": data2,
```

```

        "msg6": data4, # fixed: was data6 (undefined)
        "msg2": data5,
        "post": post,
        "All_user": result1,
        "All_profile": result2
    })
    else:
        messages.error(request, "*Please Confirm The Password")
        return render(request, "change_password.html", {"msg": data2, "msg1": data1})
    else:
        messages.error(request, "*Please Enter The Correct User Password")
        return render(request, "change_password.html", {"msg": data2, "msg1": data1})
    else:
        return render(request, "change_password.html", {"msg": data2, "msg1": data1})
    else:
        # ✓ FIX: If session doesn't exist, redirect to login or show an error page
        messages.error(request, "You must be logged in to change your password.")
        return redirect('login') # or render a login page

```

## 4.2 Prototype

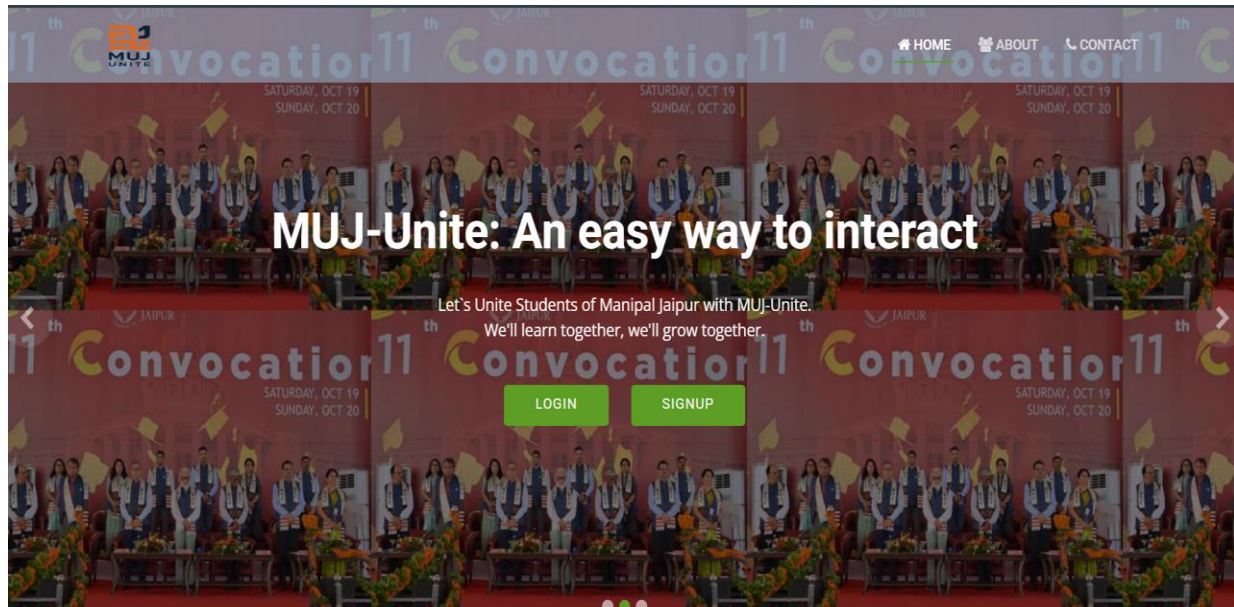
Before starting full-scale development, we created a working prototype of the MUJ Unite platform to visualize how the system would function and appear to users. This early version helped us shape the layout, user flow, and main features, making sure the platform would be easy to navigate and meet the expectations of its primary users—students, alumni, and faculty.

The prototype closely mirrored the planned structure of the final product. It included essential sections like the homepage, user profiles, messaging area, announcements board, login and registration screens, and utility pages such as “About Us” and “Contact.” We also incorporated features like QR code display and change password functionality to simulate real use cases.

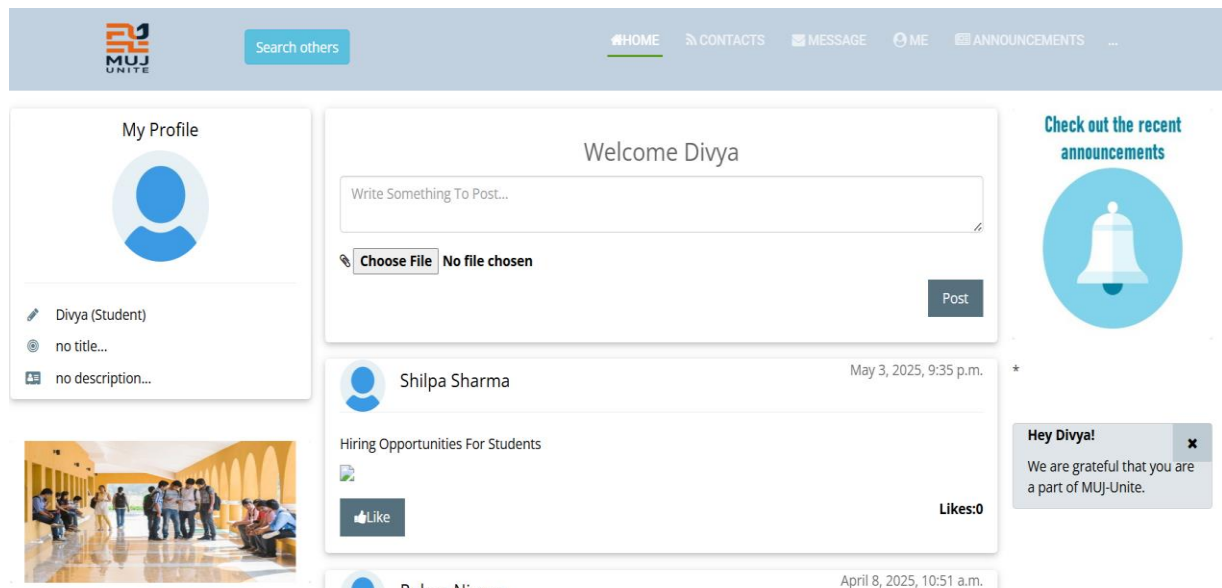
By building the prototype first, we were able to test the interface, get early feedback from peers and mentors, and make adjustments before investing time into detailed coding. It acted as both a design reference and a planning tool, guiding our decisions during the actual development phase and ensuring that the user experience remained a top priority.

- » The Home Page is central to user engagement and was prototyped to feature recent posts, announcements, and a navigation bar with quick links to all other modules. The layout follows a clean and organized structure, ensuring that users can access key information and perform primary actions without confusion. The header includes navigation buttons to the profile, messages, contact us form, and logout, while the body showcases updates in a card-based format.

A visual representation of how the landing page looks is given on the next page:

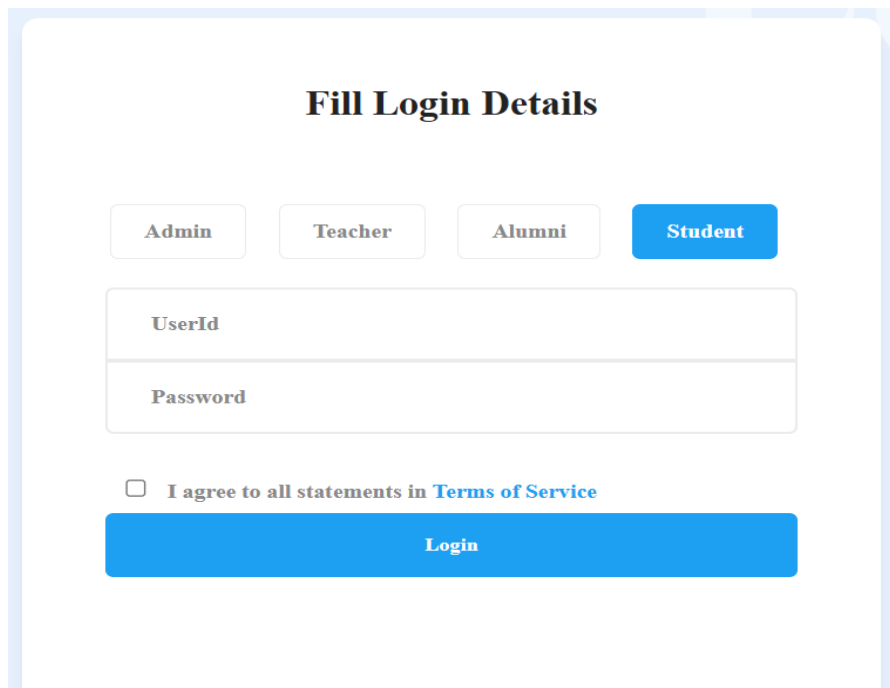


**Figure 4.2.1 Home Page**



**Figure 4.2.2 Home Page after Login**

- » The login and sign-up pages were designed with a focus on simplicity and usability. The login interface includes fields for username and password, along with proper error messages for invalid credentials. The sign-up form collects basic details such as name, email, and password, and includes password confirmation with validation checks.



The login page features a central white card with a light blue border. At the top, the title 'Fill Login Details' is centered in a bold, black font. Below the title, there are four role selection buttons: 'Admin', 'Teacher', 'Alumni', and 'Student'. The 'Student' button is highlighted in blue, while the others are white with light blue borders. Underneath the role buttons are two stacked input fields for 'UserId' and 'Password'. Below these fields is a checkbox followed by the text 'I agree to all statements in [Terms of Service](#)'. At the bottom of the card is a large blue button labeled 'Login'.

### Fill Login Details

Admin Teacher Alumni **Student**

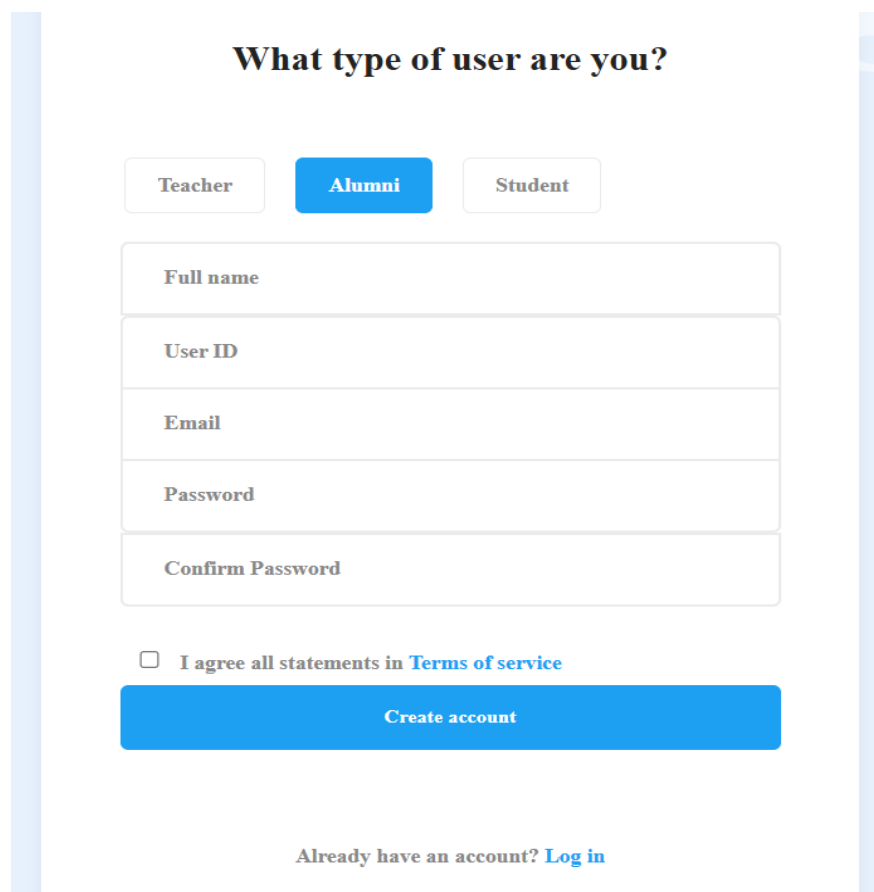
UserId

Password

☐ I agree to all statements in [Terms of Service](#)

Login

*Figure 4.2.3 Login Page*



The sign-up page features a central white card with a light blue border. At the top, the title 'What type of user are you?' is centered in a bold, black font. Below the title, there are three role selection buttons: 'Teacher', 'Alumni', and 'Student'. The 'Alumni' button is highlighted in blue, while the others are white with light blue borders. Underneath the role buttons are five stacked input fields for 'Full name', 'User ID', 'Email', 'Password', and 'Confirm Password'. Below these fields is a checkbox followed by the text 'I agree all statements in [Terms of service](#)'. At the bottom of the card is a large blue button labeled 'Create account'. Below the card, centered, is the text 'Already have an account? [Log in](#)'.

### What type of user are you?

Teacher **Alumni** Student

Full name

User ID

Email

Password

Confirm Password

☐ I agree all statements in [Terms of service](#)

Create account

Already have an account? [Log in](#)

*Figure 4.2.4 Sign up Page*

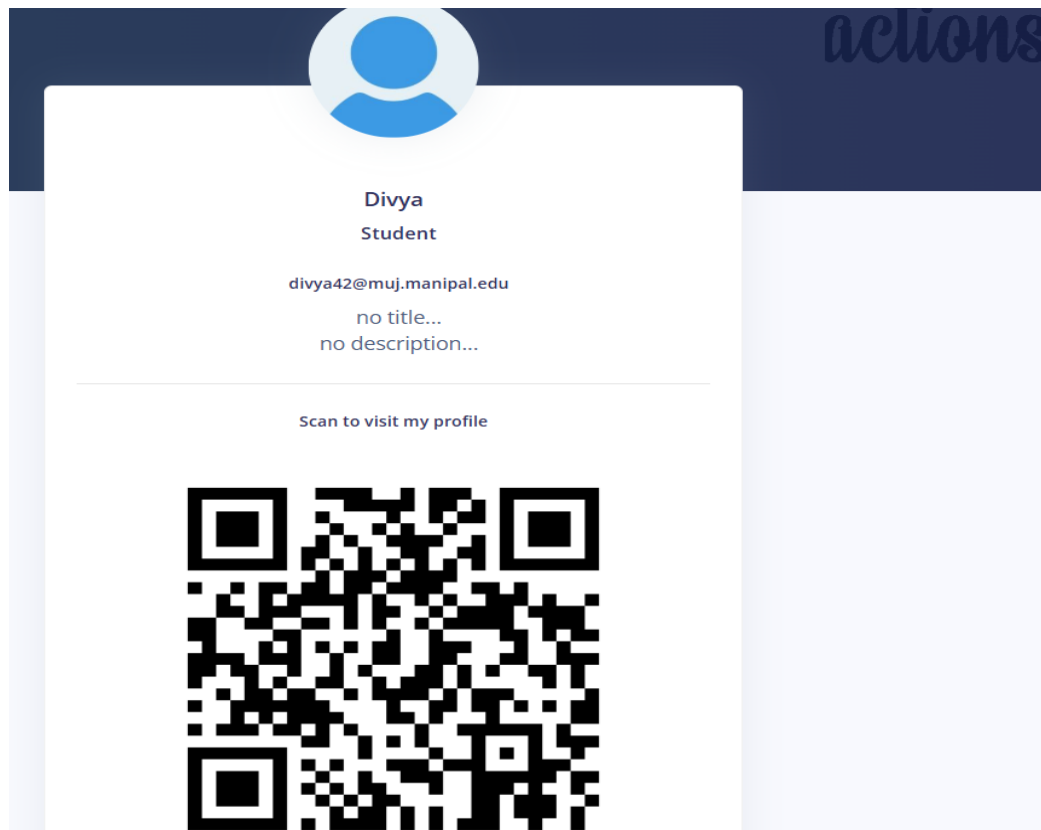
- » The profile page prototype shows the profile of the user which gets login successfully. It shows the name of the user and the user has the option to edit profile, edit post. This profile page also contains the QR Code option for each user.

The snapshot of the profile page is given below:

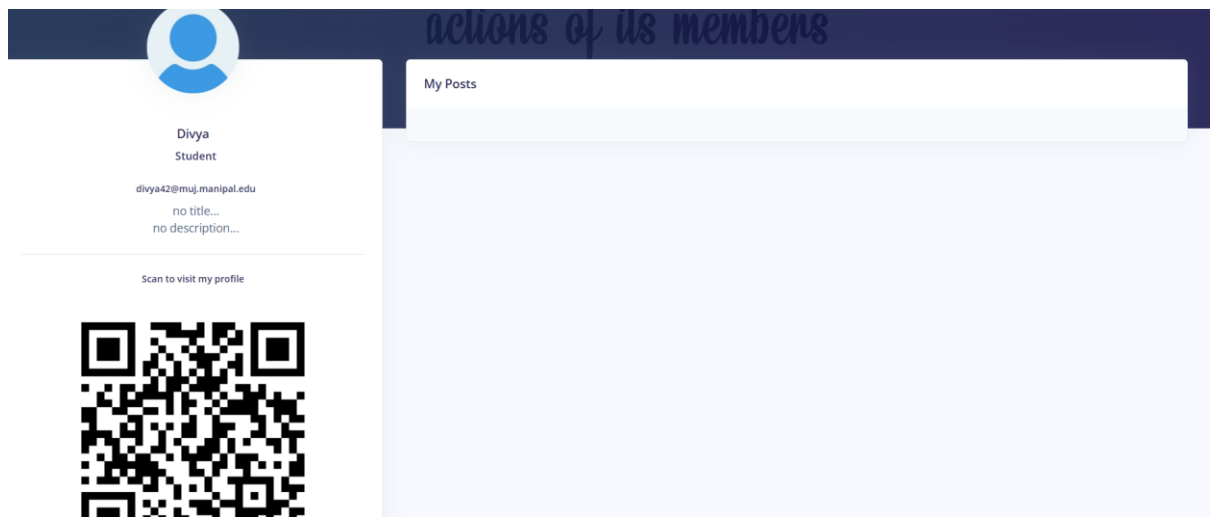


*Figure 4.2.5 Profile Page*



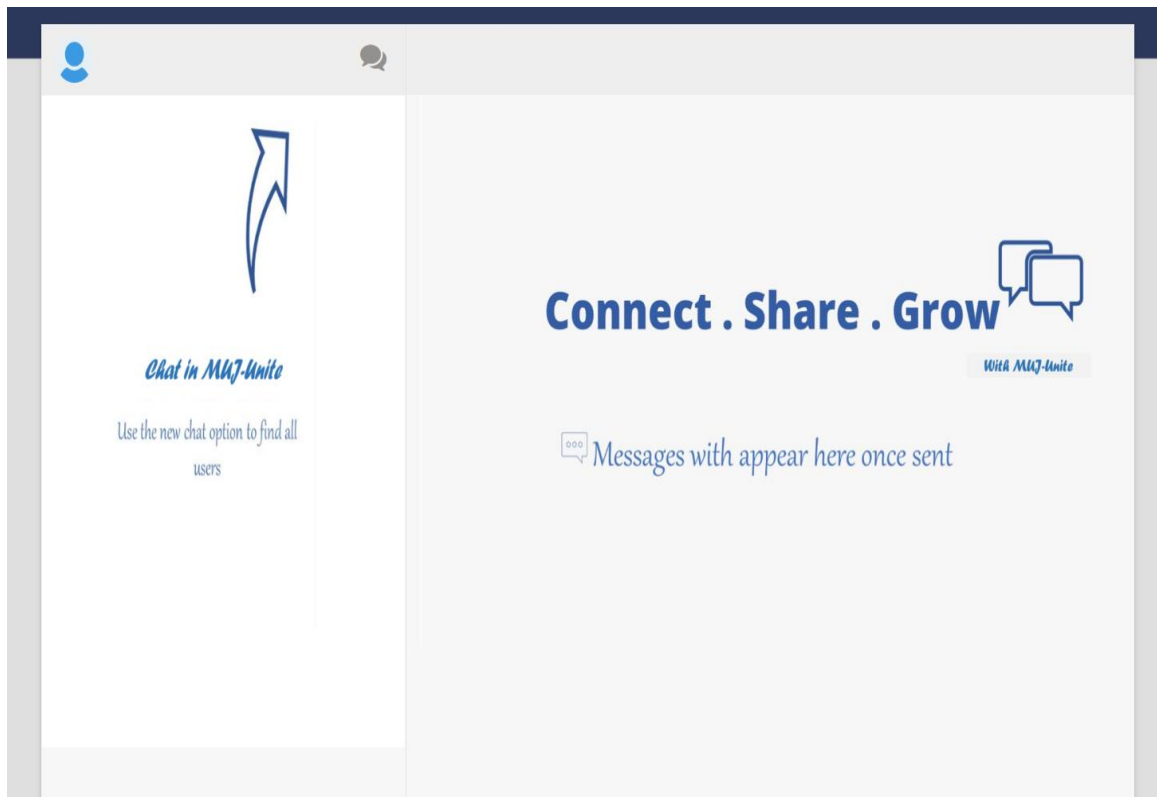


*Figure 4.2.6 QR Code for Scanning Profile*

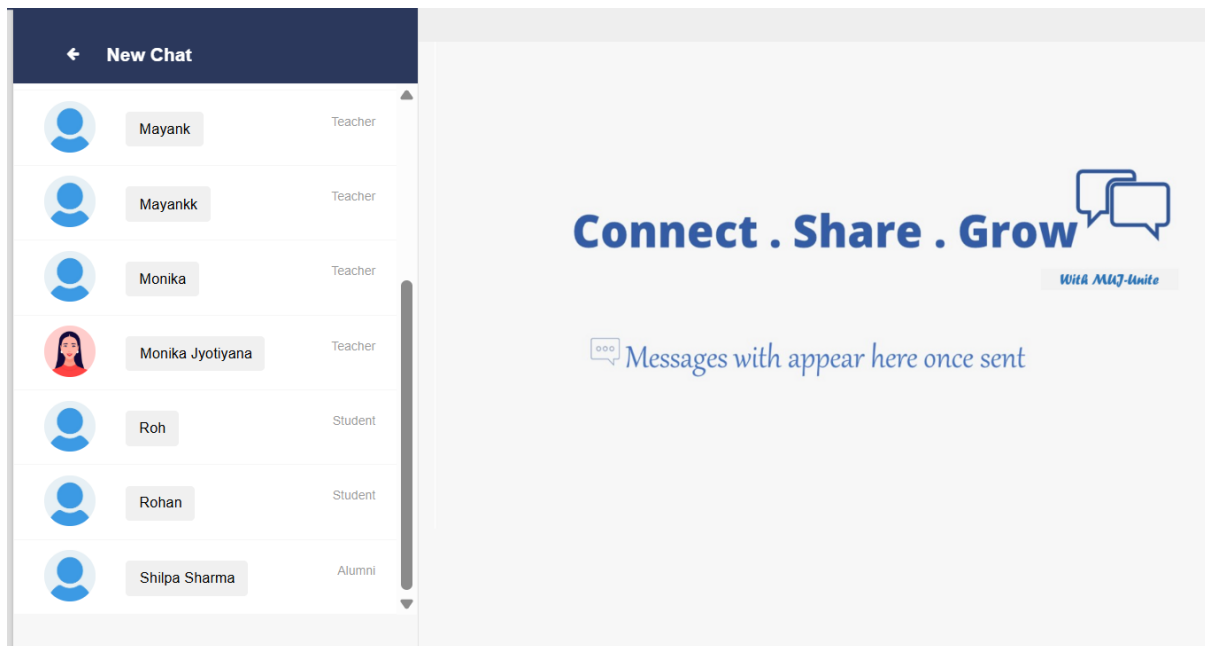


*Figure 4.2.7 Post Option at Profile Page*

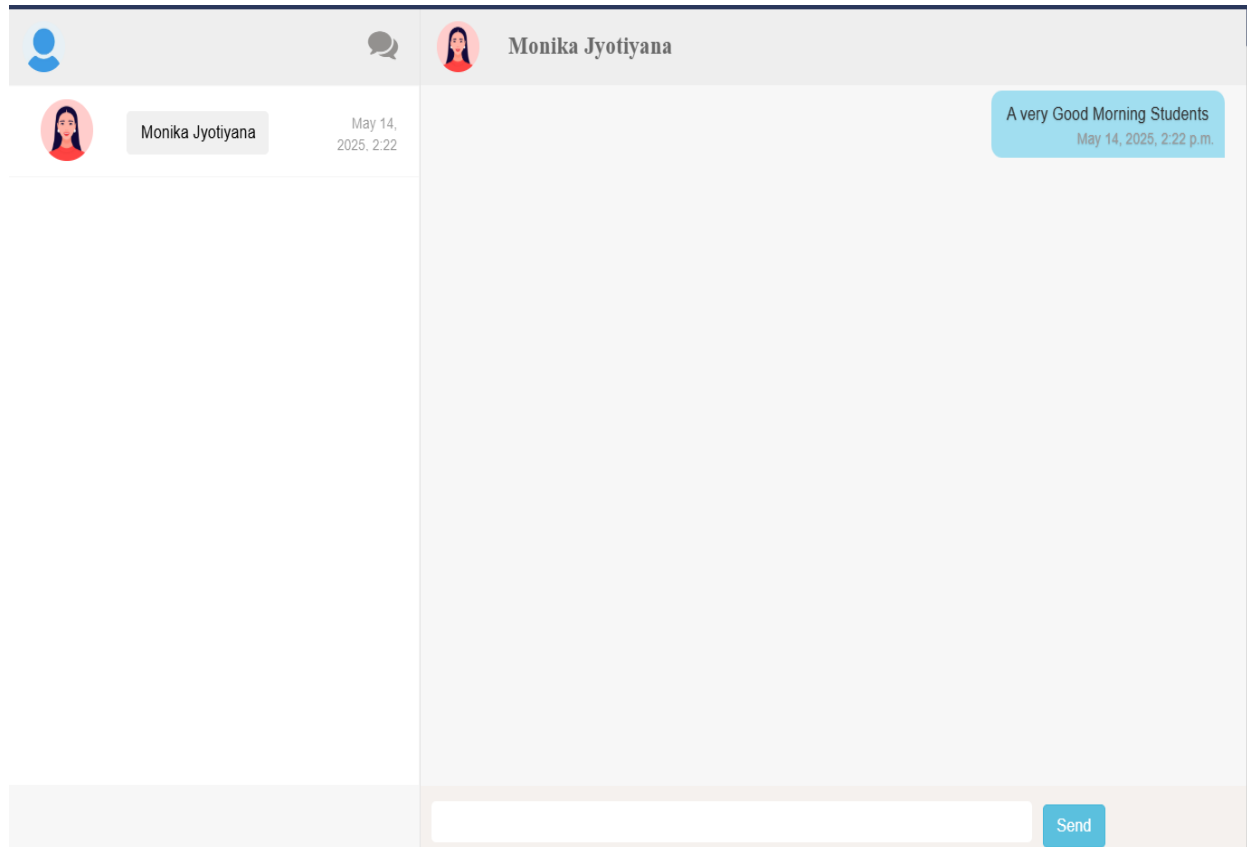
- » The Messaging Module includes the privately messaging option for the user.  
The snapshot of the message page is given below:



**Figure 4.2.8 Messaging Option**

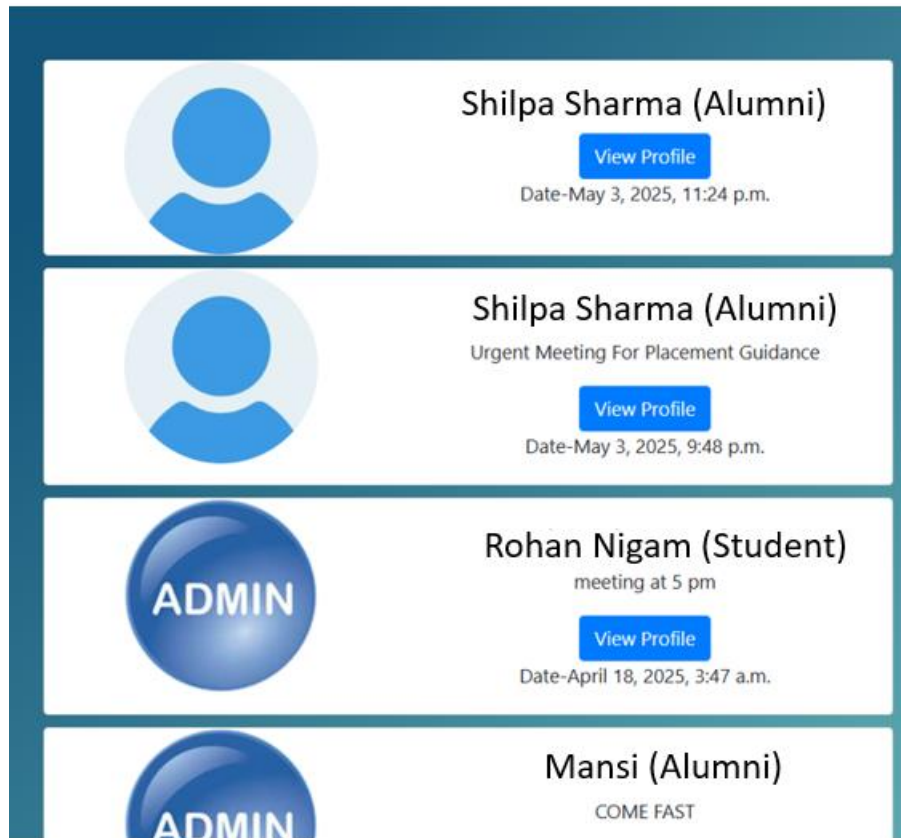


**Figure 4.2.9 New Chat Option**



***Figure 4.2.10 Private Chat Option***

- » The Announcements Module shows the user the announcement which is add by the other user. The announcement basically includes useful information which is posted by the other users. The snapshot of this module is given below:



*Figure 4.2.11 Announcement Page*

- » The Contact Us Page is designed as a communication between users and the platform administrators. This module is kept visually and simple but functionally complete, as it plays an important role in platform support.

The snapshot of this module is given on the next page:

**CONTACT  
US**

---

Name:  
Write your name here..

---

Email:  
Let us know how to contact you back

---

Message:  
What would you like to tell us..

---

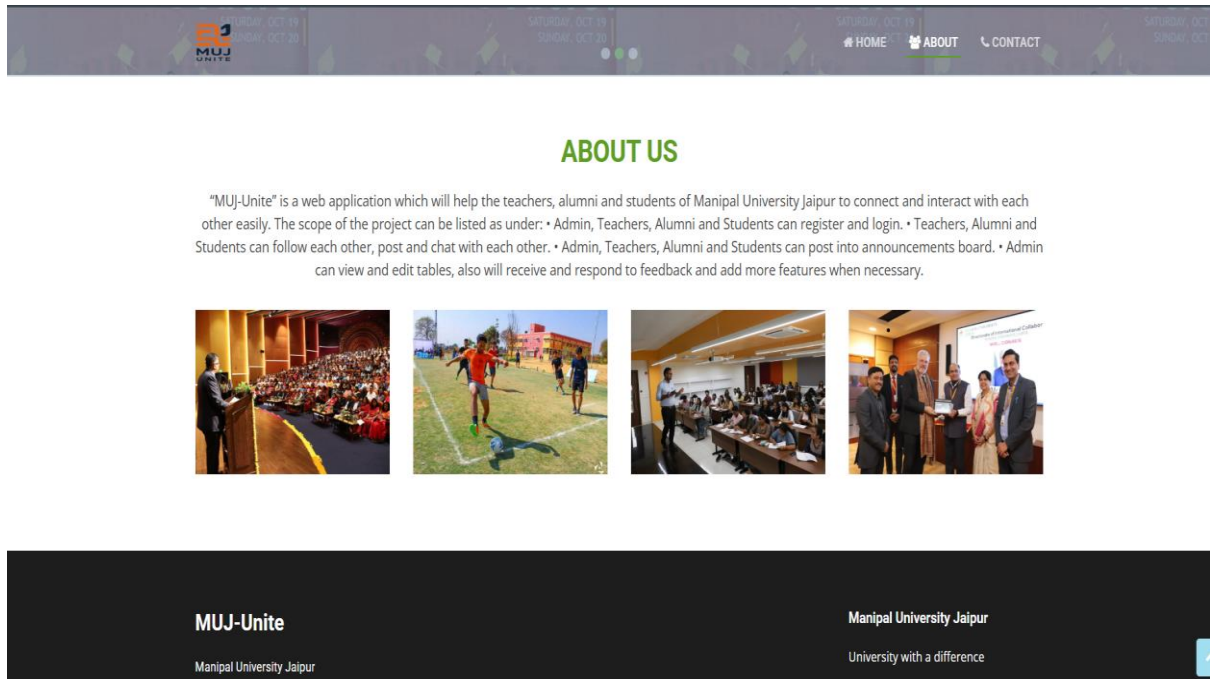
01438 228 456    MUJUnite@gmail.com

Send Message

***Figure 4.2.12 Contact Us***

» The About Us page tells the information of the MUJ Unite.

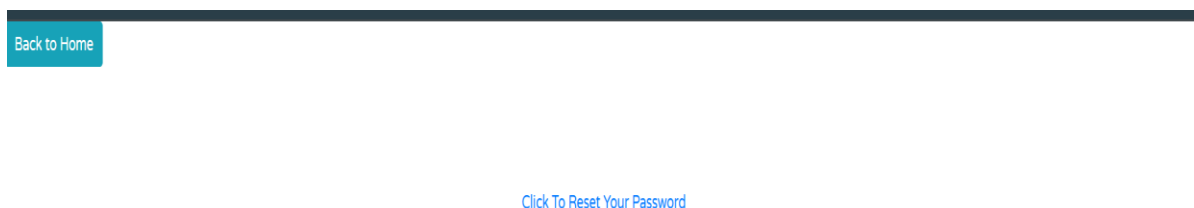
The snapshot of this module is given on the next page:



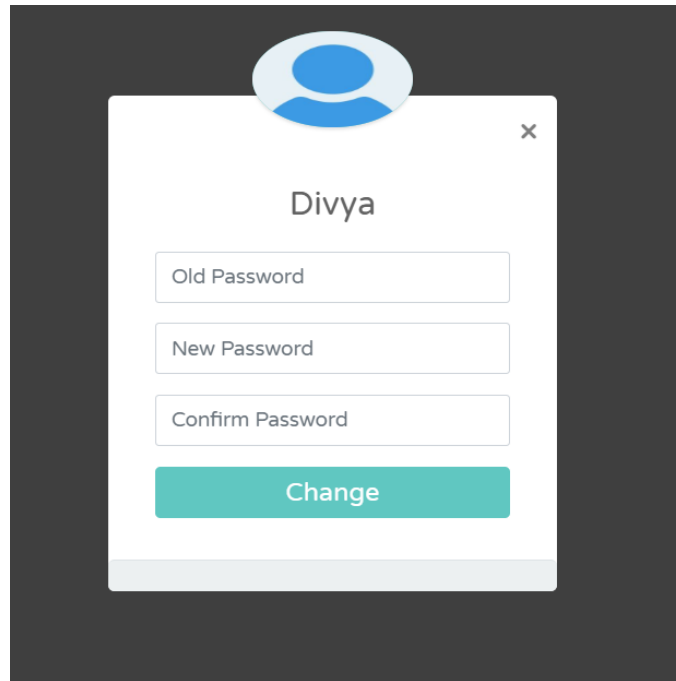
**Figure 4.2.13 About Us**

- » The Change Password Module, enhances user control and account security. It includes fields for entering the current password, new password, and password confirmation. Real-time validation ensures that the new password meets required criteria (e.g., minimum length, use of symbols or numbers). Feedback messages are displayed for both errors (e.g., incorrect current password) and successful updates. This module emphasizes security while maintaining a simple, accessible interface.

A snapshot of this module is given below:



**Figure 4.2.14 Change Password**



**Figure 4.2.15 Change Your Password**

Throughout the prototyping phase, feedback was regularly gathered from test users and mentors. Usability tests helped identify layout inconsistencies, unclear navigation paths, and areas for improvement. The prototype was updated iteratively to reflect changes, ensuring the final platform would be user-friendly and functionally complete. [26]

The prototype phase was instrumental in shaping the development of the MUJ Unite platform. Each module was carefully planned and visually represented to align with user expectations and technical feasibility. The inclusion of essential features like profile management, QR code integration, messaging, password control, and support pages ensures that the platform is comprehensive, efficient, and ready to serve the MUJ alumni network effectively.

» Comparison between Prototype Design and Final Implementation.

**Table 4.2.1 Prototype v/s Final Implementation**

Feature/Module	Prototype Design	Final Implementation
Home Page	Static layout with placeholder content.	Fully functional with dynamic announcements and posts pulled from the database.
Login & Sign Up	Basic UI with no validation logic.	Integrated with Django authentication; includes form validation, error handling, and session management.
Profile Page	Static user details layout; QR code not included.	Dynamic user info display; integrated QR code generated using qrcode and base64.
QR Code Feature	Not included in early design.	Fully implemented—QR code generated per user and embedded in the profile.

Messaging System	Designed with dummy user messages.	Real-time user messaging with sender/receiver relationships and database integration.
Announcements	Static announcement cards with mock text.	Fully dynamic; admins can post, edit, or delete announcements.
Contact Us Form	Placeholder form with no backend logic.	Fully functional; form data saved and accessible by admin.
About Us Page	Designed with lorem ipsum placeholder content.	Populated with actual project details and developer info.
Change Password	Not present in prototype.	Implemented with current password validation and secure password update.
UI/UX Enhancements	Basic layout, minimal CSS, limited responsiveness.	Improved UI using Bootstrap/custom styling; responsive design for desktop and mobile.
Error Handling & Feedback	Not considered during prototype.	User feedback via Django messages framework for success/failure scenarios.



## Chapter 5

### Results and Analysis

The objective of MUJ Unite is to develop a centralized web-based platform for students, alumni, and faculty of MUJ to communicate, share announcements, and stay connected. After completing the development and deployment of all modules, the system was rigorously tested to ensure all functionalities performed as intended. The results validate that the platform meets the functional and non-functional requirements outlined in the planning phase.

» Functional Results: The following modules were tested for functionality and performance and listed in the table 5.1:

*Table 5.1 Module Functionality*

Module	Result
Home	Loads dynamic announcements and navigation; verified user-specific content.
Login & Signup	Successfully authenticates users with hashed passwords and validation logic.
Profile & QR Code	Displays accurate user data; QR code generated dynamically and scannable.
Messaging	Messages sent and received between users accurately; database updates real-time.
Announcements	Admin can post and delete announcements; visible to all logged-in users.
Contact Us	Captures and stores user queries correctly in the database.
About Us	Static content loads correctly with no errors.
Change Password	Old password is validated; new password updated securely with hashing.

» Performance Analysis: The performance analysis of MUJ Unite is discussed below:

- Speed and Load Time: All modules respond within acceptable time limits under normal usage. Pages like Home and Profile load in under 2 seconds.
- Scalability: The system is built on Django, which is scalable for moderate to high usage with appropriate database optimization.
- Data Integrity: Inputs such as user registration, messages, and announcements were stored accurately in the database without loss or corruption.

» Security Considerations: The security features are discussed below:

- Passwords are hashed using Django's `make_password()` and `check_password()` functions, preventing plain text storage.
- Access control is implemented via role-based logic (admin, student, alumni), ensuring data privacy.
- The QR code is securely generated per user session and cannot be exploited without proper credentials.

» Usability and User Experience: The usability and user experience is discussed below:

- The UI is intuitive and easy to navigate. Users can locate features like messaging, announcements, and profiles effortlessly.
- Bootstrap and responsive design ensure cross-device compatibility (mobile and desktop).
- Django messages provide real-time feedback (e.g., "Login Successful", "Password Changed"), improving user interaction.

» Limitations and Observations: Some of the limitations and observations is discussed below:

- No real-time chat (Web Sockets or AJAX) is implemented; messaging is page-refresh based.
- No file/image sharing in messages.
- Admin backend could be further expanded for analytics or user management dashboards.
- QR code links are static and not integrated with external systems like LinkedIn or MUJ alumni databases.

## *5.1 Conclusions*

MUJ Unite was developed to strengthen the connection between teachers, students, and alumni through features such as private messaging, announcements, posts, notifications, and contact management. By using Python with the Django framework and integrating SQLite for data handling, the platform delivers a smooth, responsive, and efficient user experience tailored to the MUJ community. The project successfully met its objectives by providing a secure and user-friendly interface that supports key academic and social interactions. All major modules were tested thoroughly, and the system showed consistent performance across different user roles. Its structure also leaves room for future upgrades, making it suitable for potential institutional deployment or pilot testing.

Overall, MUJ Unite simplifies information sharing, encourages collaboration, and fosters a stronger bond within the university community. It lays the groundwork for a more connected and supportive academic environment, combining practical functionality with a meaningful purpose.

## 5.2 Future Scope

While MUJ Unite currently meets its foundational objectives, several improvements and expansions can be considered in future iterations to make the platform more powerful and user-centric:

- » Real-Time Chat Integration:
  - Implement Web Socket-based real-time messaging (using Django Channels).
  - Support for group chats and chat notifications.
- » Enhanced Admin Panel:
  - Create a dashboard with analytics (user activity, message counts, traffic statistics).
  - Allow admins to manage users, posts, and feedback more effectively.
  - Media Sharing in Messaging:
    - Add support for file, image, and document sharing in private messages.
    - Include a media viewer or downloader for attachments.
- » Notifications and Alerts:
  - Implement email or SMS notifications for new messages or announcements.
  - Real-time popups for in-site notifications.
- » Alumni Job Portal Integration:
  - Include a job board where alumni and companies can post opportunities.
  - Students can apply, save, or get notified about jobs relevant to their profile.
- » QR Code Extension:
  - Make QR codes scannable via mobile to link to external profiles (LinkedIn, GitHub).
  - Add event-based QR usage (e.g., for alumni event check-ins).
- » UI/UX Enhancements:
  - Improve accessibility features like dark mode, font resizing, and multilingual support.
  - Use animations and transitions for better visual experience.
- » Mobile Application:
  - Develop an Android/iOS version of the platform for better reach and portability.

## Bibliography

- [1] Conallen, Jim. "Modeling web application architectures with UML." *Communications of the ACM* 42.10 (1999): 63-70.
- [2] Rustagi, Palak, and Yugal Kumar. "Mvc architecture and its application." (2022).
- [3] Truica, Ciprian-Octavian, et al. "Performance evaluation for CRUD operations in asynchronously replicated document oriented database." *2015 20th International Conference on Control Systems and Computer Science*. IEEE, 2015.
- [4] Grüner, Sten, Julius Pfrommer, and Florian Palm. "RESTful industrial communication with OPC UA." *IEEE Transactions on Industrial Informatics* 12.5 (2016): 1832-1841.
- [5] Gancheva, Veska. "data security and validation framework for a scientific data processing SOA based system." *2011 Developments in E-systems Engineering*. IEEE, 2011.
- [6] Aronson, Larry. *HTML Manual of Style: A Clear, Concise Reference for Hypertext Markup Language (including HTML5)*. Pearson Education, 2010.
- [7] Sharma, T. N., Priyanka Bhardwaj, and Manish Bhardwaj. "Differences between HTML and HTML 5." *International Journal Of Computational Engineering Research* 2.5 (2012): 1430-1437.
- [8] Schengili-Roberts, Keith. *Core CSS: Cascading style sheets*. Prentice Hall Professional, 2004.
- [9] Bos, Bert, et al. "Cascading style sheets level 2 revision 1 (css 2.1) specification." *W3C working draft, W3C, June* (2005).
- [10] Lunn, Ian. *CSS3 foundations*. John Wiley & Sons, 2012.
- [11] Crockford, Douglas. *JavaScript: The Good Parts: The Good Parts*. " O'Reilly Media, Inc.", 2008.
- [12] Guha, Arjun, Claudiu Saftoiu, and Shriram Krishnamurthi. "The essence of JavaScript." *ECOOP 2010–Object-Oriented Programming: 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings 24*. Springer Berlin Heidelberg, 2010.
- [13] Jensen, Simon Holm, Magnus Madsen, and Anders Møller. "Modeling the HTML DOM and browser API in static analysis of JavaScript web applications." *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011.
- [14] Hesterberg, Tim. "Bootstrap." *Wiley Interdisciplinary Reviews: Computational Statistics* 3.6 (2011): 497-526.
- [15] Python, Why. "Python." *Python releases for windows* 24 (2021).
- [16] Thoutam, Vivek. "A study on python web application framework." *Journal of Electronics, Computer Networking and Applied mathematics* 1.01 (2021).
- [17] Burch, Carl. "Django, a web framework using python: Tutorial presentation." *Journal of Computing Sciences in Colleges* 25.5 (2010): 154-155.

- [18] Torres, Alexandre, et al. "Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design." *information and software technology* 82 (2017): 1-18.
- [19] Chandiramani, Ashish, and Pawan Singh. "Management of Django web development in Python." *Journal of Management and Service Science (JMSS)* 1.2 (2021): 1-17.
- [20] Kreibich, Jay. *Using SQLite*. " O'Reilly Media, Inc.", 2010.
- [21] Erickson, John, Kalle Lyytinen, and Keng Siau. "Agile modeling, agile software development, and extreme programming: the state of research." *Journal of Database Management (JDM)* 16.4 (2005): 88-100.
- [22] Bastos, Ricardo Melo, and Duncan Dubugras A. Ruiz. "Extending UML activity diagram for workflow modeling in production systems." *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, 2002.
- [23] Li, Qing, and Yu-Liu Chen. "Data flow diagram." *Modeling and Analysis of Enterprise and Information Systems*. Springer, Berlin, Heidelberg, 2009. 85-97.
- [24] Song, Il-Yeol, Mary Evans, and Eun K. Park. "A comparative analysis of entity-relationship diagrams." *Journal of Computer and Software Engineering* 3.4 (1995): 427-459.
- [25] Ganney, Paul S., Sandhya Pisharody, and Ed McDonagh. "Web development." *Clinical Engineering*. Academic Press, 2014. 171-190.
- [26] Sabale, R. Ganpatrao, and A. R. Dani. "Comparative study of prototype model for software engineering with system development life cycle." *IOSR Journal of Engineering* 2.7 (2012): 21-24

