



National Academy of Science and Technology

(Affiliated to Pokhara University)

Accredited by University Grants Commission (UGC), Nepal (2022)

Uttarbehadi -4, Dhangadhi, Kailali, Nepal

**A
Project Report
On
Notice Hub**

For the partial fulfilment of requirements for the Degree of Bachelor of Computer
Engineering under Pokhara University

Submitted to

Department of Computer Engineering
National Academy of Science and Technology

Under the Supervision of

Mr. Sunil Bahadur Bist
Lecturer, Department of Computer Engineering

Submitted by

Abishek Khadka (21070001)
Bikash Sunar (21070005)
Prem Rawal (21070023)
Sandesh Chaudhary (21070031)

BE Computer, 8th Semester
August 2025

ABSTRACT

The Notice Hub is a full-stack application designed and developed to centralize and streamline communication within educational institutions. It provides a unified digital platform for administrators, teachers, and students, addressing the challenges of fragmented and delayed notice dissemination.

The system's architecture is built on a robust backend utilizing the Spring Boot framework, which powers a RESTful API. This API serves both a dynamic web interface for administrative functions and a native Android mobile application for teachers and students. Security is a primary concern, and the backend leverages Spring Security with JWT (JSON Web Tokens) for stateless authentication on mobile clients and a sophisticated, role-based access control model across the entire platform. Data persistence is managed by JPA/Hibernate.

The application features distinct user roles with tailored functionalities. Administrators manage the entire system from a web portal, overseeing user accounts and publishing notices with granular targeting options (e.g., by department, program, or semester). On the mobile front, teachers and students use a Java-based Android application to receive notices. Teachers have the added ability to publish notices directly to students from their dashboard.

Key features include real-time push notifications powered by Firebase Cloud Messaging (FCM), which alerts intended recipients instantly when a new notice is published. The system also supports file attachments, ensuring comprehensive notice content. The project successfully demonstrates the integration of modern backend and mobile technologies to create a secure, scalable, and highly efficient communication platform, effectively connecting the institutional staff with the student body.

Keywords: *Notice Hub, Notice Management System, Notice Hub App, Notice*

ACKNOWLEDGEMENT

We would like to express our profound appreciation to our institution, **NAST, Dhangadhi**, for providing the foundational academic environment and resources essential for the completion of this project. The opportunity to develop the "**Notice Hub**" as part of our curriculum has been invaluable to our growth as computer engineering students.

We are immensely grateful to our respected project supervisor, **Mr. Sunil Bahadur Bist**, Lecturer in the Department of Computer Engineering, NAST Dhangadhi. His consistent mentorship, critical insights, and unwavering support were instrumental in guiding us through every phase of this project, from conceptualization to implementation. His expertise and encouragement were a constant source of inspiration.

Our sincere thanks also go to the entire faculty of the **Department of Computer Engineering** at NAST, Dhangadhi. Their dedication to providing us with a strong theoretical and practical knowledge base has been crucial to the success of this project. The skills and knowledge we acquired under their tutelage directly enabled us to tackle the technical challenges and deliver a robust final product.

Finally, we wish to thank our friends for their continuous support and encouragement throughout this journey. Their belief in our abilities has been a driving force behind our efforts.

Team Members

Abishek Khadka (21070001)

Bikash Sunar (21070005)

Prem Rawal (21070023)

Sandesh Chaudhary (21070031)

DECLARATION

We hereby declare that the work presented in this project report, titled "**Notice Hub**", is a result of our own original and diligent effort. The project was completed under the supervision of **Mr. Sunil Bahadur Bist** and is being submitted to the Department of Computer Engineering at the National Academy of Science and Technology (NAST), Dhangadhi, to partially fulfill the requirements for the degree of Bachelor of Computer Engineering.

We affirm that this report, including its design, implementation, and findings, is an authentic representation of our research and development. This work has not been presented previously, either in its entirety or in part, to any other academic institution or university for the conferment of a degree or diploma. All external sources of information, data, and conceptual ideas used in this document have been properly cited and acknowledged in the text and the bibliography. We accept full responsibility for the content of this report and its adherence to the academic standards of honesty and integrity.

Submitted By:

Abishek Khadka (21070001)

.....

Prem Rawal (21070023)

.....

Bikash Sunar (21070005)

.....

Sandesh Chaudhary (21070031)

.....

Date: August 29, 2025

TABLE OF CONTENTS

| | |
|---|------|
| ABSTRACT | i |
| ACKNOWLEDGEMENT | ii |
| LIST OF FIGURES | vi |
| LIST OF TABLES | viii |
| LIST OF ABBREVIATION | ix |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1 Background | 1 |
| 1.2 Objectives | 1 |
| 1.4 Achievements | 3 |
| 1.4 Organization of Report | 4 |
| CHAPTER 2 | 6 |
| SURVEY OF TECHNOLOGIES | 6 |
| 2.3 Technologies Chosen for Notice Hub | 7 |
| CHAPTER 3 | 10 |
| REQUIREMENTS AND ANALYSIS | 10 |
| 3.1 Problem Definition | 10 |
| 3.2 Requirements Specification | 11 |
| 3.2.1 Functional Requirements | 11 |
| 3.2.2 Non-Functional Requirements | 12 |
| 3.4 Software and Hardware Recommendations | 13 |
| CHAPTER 4 | 16 |
| DESIGN | 16 |
| 4.1 Introduction | 16 |
| 4.2 System Design | 16 |
| 4.3 Database Design | 20 |
| CHAPTER 5 | 21 |
| IMPLEMENTATION AND TESTING | 21 |
| 5.1 Implementation Approaches | 21 |
| 5.2 Coding Details and Code Efficiency | 22 |
| 5.5 Test Cases | 24 |
| CHAPTER 6 | 26 |

| | |
|--|-----------|
| RESULTS AND DISCUSSION | 26 |
| 6.1 Test Reports..... | 26 |
| 6.2 User Documentation..... | 27 |
| 6.2.1 Admin | 28 |
| 6.2.2 Teacher | 28 |
| 6.2.3 Student..... | 28 |
| CHAPTER 7 | 30 |
| CONCLUSION AND RECOMMENDATIONS..... | 30 |
| 7.1 Conclusion | 30 |
| 7.2 Recommendations | 32 |
| REFERENCES..... | 33 |
| ANNEX I..... | 35 |
| ANNEX II | 48 |
| ANNEX III..... | 53 |

LIST OF FIGURES

| | |
|---|----|
| Figure 4.2 System Design..... | 17 |
| Figure 4.2.1 Use Case Diagram..... | 18 |
| Figure 4.2.2.1 Level-0 Dataflow Diagram..... | 19 |
| Figure 4.2.2.2 Level-1 Dataflow Diagram..... | 19 |
| Figure 4.3 Database Design..... | 20 |
| Figure 5.1 Agile Methodology..... | 21 |
| Figure: Annex 1.1 Login..... | 35 |
| Figure: Annex 1.2 Dashboard..... | 35 |
| Figure: Annex 1.3 View Teacher..... | 36 |
| Figure: Annex 1.4 Add Teacher..... | 36 |
| Figure: Annex 1.5 Update Teacher..... | 37 |
| Figure: Annex 1.6 View Student..... | 37 |
| Figure: Annex 1.7 Save Student..... | 38 |
| Figure: Annex 1.8 Update Student..... | 38 |
| Figure: Annex 1.9 Notice to All..... | 39 |
| Figure: Annex 1.10 Notice to All Authority..... | 39 |
| Figure: Annex 1.11 Notice to Specific Authority..... | 40 |
| Figure: Annex 1.12 Notice to All Teacher..... | 40 |
| Figure: Annex 1.13 Notice to Specific Teacher | 41 |
| Figure: Annex 1.14 Notice to All Student | 41 |
| Figure: Annex 1.15 Notice to Specific Student | 42 |
| Figure: Annex 1.16 Setting | 42 |
| Figure: Annex 1.17 Add, Update and delete Level | 43 |
| Figure: Annex 1.18 Add, Update and delete Faculty | 43 |
| Figure: Annex 1.19 Add, Update and delete Department | 44 |
| Figure: Annex 1.20 Add, Update and delete Program..... | 44 |
| Figure: Annex 1.21 Add, Update and delete Semester | 45 |
| Figure: Annex 1.22 Add, Update and delete Authority | 45 |
| Figure: Annex 1.23 Add, Update and delete Designation | 46 |
| Figure: Annex 1.24 Add, Update and delete Job Type | 46 |
| Figure: Annex 1.25 Add, Update and delete Notice Category | 47 |
| Figure: Annex 1.26 Admin Profile..... | 47 |

| | |
|---|----|
| Figure: Annex 2.1 Login Module..... | 48 |
| Figure: Annex 2.2 Change Password..... | 48 |
| Figure: Annex 2.3 Forgot Password..... | 49 |
| Figure: Annex 2.4 Student Home..... | 49 |
| Figure: Annex 2.5 Student Notice..... | 50 |
| Figure: Annex 2.6 Student Profile..... | 50 |
| Figure: Annex 2.7 Teacher Home..... | 51 |
| Figure: Annex 2.8 Teacher Notice..... | 51 |
| Figure: Annex 2.9 Teacher Add Notice..... | 52 |
| Figure: Annex 2.10 Teacher Profile..... | 52 |

LIST OF TABLES

| | |
|--|----|
| Table 3.3 Planning and Scheduling..... | 13 |
| Table 5.5 Test Cases..... | 25 |
| Table 6.1 Test Reports..... | 27 |

LIST OF ABBREVIATION

API: Application Programming Interface
CRUD: Create, Read, Update, Delete
CSS: Cascading Style Sheets
DFD: Data Flow Diagram
DTO: Data Transfer Object
ERP: Enterprise Resource Planning
FCM: Firebase Cloud Messaging
GB: Gigabyte
GHz: Gigahertz
GIT: Global Information Tracker
GSON: Google Simple Object Notation
HTML: Hyper Text Markup Language
HTTP: Hyper Text Transfer Protocol
IDE: Integrated Development Environment
JPA: Jakarta/Java Persistence API
JWT: JSON Web Token
JSON: JavaScript Object Notation
LTS: Long Term Support
MVC: Model View Controller
MySQL: My Structured Query Language
NH: Notice Hub
ORM: Object-Relational Mapping
OTP: One-Time Password
REST: Representational State Transfer
SDK: Software Development Kit
SQL: Structured Query Language
UI: User Interface
URI: Uniform Resource Identifier
URL: Uniform Resource Locator
UX: User Experience
XML: Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Background

In many educational institutions, the communication of important administrative and academic information relies on a mix of traditional and digital methods, such as physical notice boards, email lists, and informal group messaging apps. While these methods serve a purpose, they are often plagued by inefficiencies, including delayed delivery, information fragmentation, and a lack of a centralized, accessible archive. A student might miss a crucial exam notice if they're not on campus, or an important update from a specific department could get lost in a sea of messages. The absence of a secure, single-point-of-truth for all official announcements creates an environment of inconsistency and potential miscommunication, impacting the timely flow of information to students, teachers, and other staff.

To address these challenges, the "**Notice Hub**" project was developed as a comprehensive, real-time notice management system. This full-stack application provides a centralized platform where notices can be created, targeted, and distributed efficiently. By moving away from fragmented communication channels, the system ensures that all stakeholders receive relevant announcements in a timely manner. The Notice Hub comprises a robust backend API, a web interface for administrators, and a native Android application for teachers and students, all working together to create a seamless and reliable communication ecosystem.

1.2 Objectives

The primary goal of the Notice Hub project is to design and implement a modern, secure, and user-friendly platform for institutional communication. To achieve this, we have established the following specific objectives:

- To design and develop a centralized, full-stack application for managing and publishing notices with a role-based access control system.
- To create a dedicated web-based dashboard for administrators to efficiently manage users (teachers and students) and oversee all notice-related activities.
- To develop a native Android mobile application that enables teachers and students to receive, view, and interact with notices in real time.

- To implement a robust notice-publishing mechanism that allows administrators and teachers to send announcements to specific user groups based on criteria such as program, semester, and department.
- To integrate Firebase Cloud Messaging (FCM) to provide instant push notifications to mobile users, ensuring timely delivery of urgent and important information.
- To build a secure and organized system that supports file attachments in notices and provides a searchable, centralized archive for easy retrieval of past announcements.

1.3 Purpose, Scope, and Applicability

1.3.1 Purpose

The primary purpose of the **Notice Hub** project is to develop a comprehensive and secure digital platform that revolutionizes how academic institutions handle internal communication. The system is designed to replace outdated and inefficient methods with a centralized, real-time solution. It aims to empower administrators with powerful tools to manage users and distribute notices with precision, while ensuring that teachers and students receive critical information instantly and reliably. A core objective is to eliminate communication gaps, reduce administrative overhead, and create a transparent, easily accessible source for all institutional announcements.

1.3.2 Scope

The scope of this project encompasses the development of a full-stack application with distinct, interconnected components. The system includes:

- **User Management:** A robust system for user authentication and role-based access control for administrators, teachers, and students.
- **Notice Management:** A centralized platform where notices can be created, edited, and published. This includes functionalities for categorization (e.g., exam, holiday, results) and searching.
- **Targeted Distribution:** A sophisticated mechanism for administrators and teachers to publish notices to specific user groups, filtering by criteria such as department, program, or semester.
- **Real-time Communication:** An instant push notification system using Firebase Cloud Messaging (FCM) to deliver notices to mobile app users.

- **Web Portal:** A web-based interface for administrators to perform all management tasks.
- **Mobile Application:** A native Android application for teachers and students to view notices, with teachers also having the ability to publish notices.

1.3.3 Applicability

The **Notice Hub** system is highly applicable to any educational or institutional setting that requires efficient and structured internal communication. This includes:

- **Schools and Colleges:** For communicating schedules, exam results, and general announcements to students and parents.
- **Universities:** For departments and faculties to publish research updates, seminar schedules, and administrative circulars.
- **Training Centers and Corporate Environments:** The core functionality of a centralized, role-based notice board can be easily adapted for internal company announcements, project updates, and training schedule.

1.4 Achievements

The development of the Notice Hub system has successfully culminated in the implementation of a fully functional and integrated platform. The following key achievements highlight the project's success in meeting its core objectives:

- **Comprehensive Web Portal:** We have successfully developed a robust and intuitive web portal that serves as the administrative backbone of the system. This portal provides administrators with complete control over user accounts and notice management. It enables the seamless creation, modification, and deletion of teacher and student profiles, as well as the publishing of notices with a variety of targeting options.
- **API-driven Communication:** The system's architecture is built on a RESTful API that facilitates a secure and efficient communication channel between the web system and the database. This API-based approach ensures that the frontend and backend are decoupled, allowing for greater flexibility and scalability. The API endpoints are protected using Spring Security and JWT, ensuring that all data exchange is secure and authorized.
- **User-friendly Interface:** A clean, intuitive, and responsive user interface has been designed and implemented for the web portal. The dashboard is

organized to simplify complex administrative tasks, making it easy for administrators to manage users and notices without extensive training.

- **Real-time Communication:** The system achieves instant communication by delivering notices to both teachers and students. This functionality has been enhanced with Firebase Cloud Messaging (FCM), providing push notifications to the mobile app for immediate delivery. This real-time alert system ensures that critical information, such as exam schedules or urgent announcements, is received by the intended audience without delay.

1.4 Organization of Report

This report is organized into seven main chapters to systematically present the planning, development, and current progress of the Notice Hub project. Each chapter focuses on a specific aspect of the system. Chapter 1: Introduction: This chapter sets the stage for the project, introducing the problem that the Notice Hub system is designed to solve. It highlights the inefficiencies of traditional communication methods in educational institutions, such as fragmented information and a lack of real-time delivery. Chapter 2: Survey of Technologies: This chapter provides a comprehensive review of the existing technological landscape for academic communication. It begins with an analysis of both informal platforms, like WhatsApp, and commercial systems, such as Fedena, detailing their features and limitations to justify the need for a new solution. Chapter 3: Requirement Analysis: This chapter formally defines the project by outlining the specific problems and requirements it addresses. The chapter then provides a thorough breakdown of both functional and non-functional requirements. Chapter 4: Design: This chapter serves as the blueprint for the entire system, detailing its architectural and structural design. It presents the system's architecture, which is a modular three-tiered client-server model. This section uses diagrams such as the Use Case Diagram and Data Flow Diagram (DFD) to visualize how different users and data flow through the system. Chapter 5: Implementation and Testing: This chapter documents the practical execution of the project, covering both the development and quality assurance phases. The section details the specific coding practices used, emphasizing code efficiency, modularity, and the use of the MVC pattern. Chapter 6: Result and Discussion: This chapter presents the outcome of the project. It provides a formal test report, using specific test cases to demonstrate that the system's core functionalities from user login to targeted notice publishing are working correctly.

Chapter 7: Conclusion and Recommendation: The final chapter summarizes the entire project. It begins with a conclusion that recaps the project's journey, reiterates its key achievements, and affirms that all initial objectives have been met.

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1 Introduction

This chapter presents a comprehensive review of existing systems and technologies used for academic notice sharing. It explores both commercial and informal platforms commonly utilized within educational institutions. The purpose of this survey is to analyse the features, advantages, and inherent limitations of these solutions. By examining the current landscape, we aim to identify critical gaps in functionality, security, and efficiency, thereby highlighting the clear need for a more robust, centralized, and purpose-built system like **Notice Hub**.

2.2 Survey of Technologies

2.2.1 Fedena (College Management System)

Fedena is a well-established and comprehensive ERP (Enterprise Resource Planning) system for schools and colleges. It offers an all-in-one suite of modules that streamline various institutional operations, including attendance, examination management, timetable generation, and a digital notice board.

Key Features:

- **Centralized Portal:** It provides a unified web-based platform for all stakeholders, including students, teachers, and administrators.
- **Integrated Notice Board:** The notice board is an integrated component of a much larger ERP, allowing it to function within a pre-existing management framework.
- **Role-Based Access:** Access to features is strictly controlled by user roles, ensuring that users can only view and interact with information relevant to their permissions.
- **Web-Based Interface:** It primarily operates through a web interface, which is accessible from any internet-connected device.

Limitations:

- **High Complexity and Cost:** As a full ERP, Fedena is often heavy, complex, and expensive to implement, making it a prohibitive solution for smaller institutions or those only seeking a focused communication tool.

- **Steep Learning Curve:** The system requires significant training and setup time for all users to become proficient, leading to low adoption rates.
- **Lack of Real-time Mobile Focus:** The notice module is typically not designed for instant, real-time push notifications, which is a critical requirement for urgent communication. Its web-first design often results in a sub-par mobile experience.

2.2.2 WhatsApp Groups (Informal Notice Sharing)

WhatsApp is widely adopted by educational institutions for informal notice sharing. Groups are often created by departments, faculty, and students to broadcast academic notices, updates, and general discussions.

Key Features:

- **Instant Messaging:** Messages are delivered instantly, providing a quick way to disseminate information.
- **Media and File Sharing:** It supports the sharing of documents, images, and other media, which is useful for distributing notice attachments.

Limitations:

- **Absence of Role-Based Access:** There is no formal access control. Anyone in a group can post, leading to a disorganized and cluttered environment where important notices can be easily missed.
- **Lack of Structure and Filtering:** The platform lacks a structured system for categorizing, tagging, or filtering notices, making it difficult to search for specific announcements.
- **Information Fragmentation:** Notices are scattered across multiple groups, making it impossible to have a single, centralized archive. This can lead to confusion and difficulty in retrieving past information.
- **Privacy and Security Concerns:** Sharing personal contact information in large groups poses a security risk, and the informal nature of the platform can lead to non-academic discussions, distracting from the main purpose of the group.

2.3 Technologies Chosen for Notice Hub

The technology stack for the **Notice Hub** was selected to ensure a robust, scalable, and secure platform that is both efficient for development and reliable for end-users. Our

choices reflect a commitment to widely-used, industry-standard tools that are well-suited for a modern, full-stack application.

2.3.1 Backend Technologies: Spring Boot & Java

Spring Boot was chosen as the core framework for the backend due to its ability to streamline development and deployment. It provides a powerful foundation for building micro services and RESTful APIs with minimal configuration.

- **Spring Security:** This framework was essential for implementing the project's sophisticated, role-based access control. It provides secure authentication using JWT for the mobile app, ensuring a stateless and secure experience.
- **Spring Data JPA:** We used this to manage database interactions, leveraging the power of Hibernate to provide a robust ORM (Object-Relational Mapping) solution. This simplifies data persistence, enhances data integrity, and abstracts complex SQL queries.
- **Firebase Admin SDK:** This was integrated to enable real-time push notifications. It allows the backend to securely communicate with the Firebase Cloud Messaging (FCM) service to instantly alert users of new notices.

2.3.2 Web Frontend Technologies: HTML, CSS, & JS

The admin web portal was developed using foundational web technologies. This approach ensures a light-weight, high-performance interface that is easy to maintain and deploy.

- **HTML:** Provides the core structure of the web pages.
- **CSS:** Handles the styling and layout, creating a clean and user-friendly administrative dashboard.
- **JavaScript (JS):** Manages dynamic and interactive elements on the web pages, facilitating a smooth user experience.

2.3.3 Mobile Application Technology: Native Android (Java)

A native Android application was built to provide an optimal user experience for teachers and students.

- **Java:** A mature and robust programming language that offers excellent performance and access to the full capabilities of the Android platform.
- **Retrofit:** The industry-standard library for making HTTP requests. It simplifies communication with the backend RESTful API, ensuring type-safe and efficient data exchange.

- **Firestore Database:** The core technology for delivering real-time push notifications, which is critical for instant notice dissemination.

2.3.4 Database Technology: MySQL

MySQL was chosen as the database to manage all system data. It is a powerful, reliable, and widely-supported relational database management system that provides the necessary integrity and performance for a project of this nature.

CHAPTER 3

REQUIREMENTS AND ANALYSIS

3.1 Problem Definition

The current communication landscape within educational institutions, as observed at NAST College, is characterized by a fragmented and inefficient mix of traditional and digital platforms. The heavy reliance on paper-based notices, informal platforms like WhatsApp and Facebook, and basic websites results in significant communication challenges. This approach creates a disorganized and unreliable ecosystem for sharing important information.

The key problems identified are:

- **Lack of Centralization:** Information is scattered across multiple platforms, making it difficult for students and teachers to find a single, authoritative source for all academic notices. Critical updates often get lost in a stream of informal messages and social media posts.
- **Administrative Inefficiency:** Manual processes, such as printing and physically distributing notices, are time-consuming and prone to human error. This administrative overhead diverts valuable time from core responsibilities.
- **Absence of Real-time Delivery:** Traditional methods lack real-time functionality. Students and teachers who are off-campus or not actively checking a specific platform may miss time-sensitive announcements, leading to confusion and delays.
- **Poor Access Control:** Informal platforms like WhatsApp do not offer structured, role-based access. Anyone in a group can post, leading to a cluttered environment and making it impossible to ensure that only relevant, verified information is shared with specific user groups.
- **Lack of a Structured Archive:** These methods provide no organized way to archive notices. Retrieving past announcements for reference is often a cumbersome process, relying on manual searches through message histories or outdated folders.

The **Notice Hub** is designed to solve these problems by creating a secure, integrated, and centralized platform that automates the entire notice-sharing lifecycle, ensuring that all stakeholders receive timely and accurate information.

3.2 Requirements Specification

To address the problems defined above, the **Notice Hub** system was designed with a clear set of functional and non-functional requirements.

3.2.1 Functional Requirements

The system is divided into three primary modules based on user roles: Admin, Teacher, and Student.

A. Admin Module (Web-Based)

- **FR1: User Management:** The Admin shall be able to create, view, update, and manage student and teacher accounts, including their personal and institutional details.
- **FR2: Notice Management:** The Admin shall be able to create, publish, and delete notices.
- **FR3: Targeted Notice Publishing:** The Admin shall be able to publish notices to specific groups, such as all users, all teachers, all students, or filtered by department, program, or semester.
- **FR4: System Updates:** The Admin shall have the ability to manage system-wide updates and resolve issues.

B. Teacher Module (Android App)

- **FR5: Notice Retrieval:** The Teacher shall be able to view a list of all notices that have been published to them by an Admin.
- **FR6: Notice Publishing:** The Teacher shall be able to publish notices directly to their students, filtered by program and semester.
- **FR7: Real-time Notifications:** The Teacher shall receive real-time notifications for new notices published by the Admin.
- **FR8: Profile Management:** The Teacher shall be able to view and manage their personal profile.

C. Student Module (Android App)

- **FR9: Notice Retrieval:** The Student shall be able to view a list of all notices that have been published to them by an Admin or a Teacher.
- **FR10: Real-time Notifications:** The Student shall receive real-time notifications for new notices.
- **FR11: Profile Management:** The Student shall be able to view and manage their personal profile.

D. Shared Functionality (Both Teacher and Student App)

- **FR12: File Handling:** Users shall be able to view and download files attached to notices.
- **FR13: Read Status:** Users shall be able to mark notices as read, and the system shall track this status.

3.2.2 Non-Functional Requirements

- **NFR1: Security:** All user data, including login credentials, must be securely stored and managed. Communication with the backend API shall be secured using JWT. Role-based access control must be strictly enforced.
- **NFR2: Usability:** Both the web interface and the mobile application must feature a clean, intuitive, and user-friendly design to enhance the overall user experience.
- **NFR3: Performance:** The system should be responsive. The app should load data quickly and notifications should be delivered in real-time.
- **NFR4: Reliability:** The system must provide a consistent and reliable service, handling network errors gracefully and providing clear feedback to the user. The notice database should have a backup mechanism.
- **NFR5: Maintainability:** The code on both the backend and mobile app must be well-structured and documented to allow for easy maintenance and future enhancements.

3.3 Planning and Scheduling

The project timeline spanned from the second week of May to the final week of August 2025. It began with the initial phases of problem identification, requirement collection, and proposal defense in May. This was immediately followed by a concentrated period of system design that extended through the first week of June. The majority of the project duration, from the beginning of June until the third week of August, was dedicated to coding and testing. A mid-term defense was conducted in July to assess the progress. The final weeks were allocated for testing and debugging with documentation, leading up to the final defense at the end of August.

| Task | May 2025 | | | June 2025 | | | | July 2025 | | | | August 2025 | | | |
|------------------------|----------|----|----|-----------|----|----|----|-----------|----|----|----|-------------|----|----|----|
| | W2 | W3 | W4 | W1 | W2 | W3 | W4 | W1 | W2 | W3 | W4 | W1 | W2 | W3 | W4 |
| Problem identification | | | | | | | | | | | | | | | |
| Requirement Collection | | | | | | | | | | | | | | | |
| Proposal Defence | | | | | | | | | | | | | | | |
| System Design | | | | | | | | | | | | | | | |
| Coding & Testing | | | | | | | | | | | | | | | |
| Mid-term Defence | | | | | | | | | | | | | | | |
| Testing and Debugging | | | | | | | | | | | | | | | |
| Documentation | | | | | | | | | | | | | | | |
| Final Defence | | | | | | | | | | | | | | | |

Table: 3.3 Planning and scheduling

3.4 Software and Hardware Recommendations

The following is a detailed breakdown of the recommended resources for developing and running the **Notice Hub** project.

3.4.1 Software Recommendations

- **IntelliJ IDEA:** This is the primary Integrated Development Environment (IDE) recommended for backend development. It provides advanced features like intelligent code completion, powerful debugging tools, and seamless integration with frameworks like Spring Boot and Git, which significantly boosts developer productivity.
- **MySQL Workbench:** This graphical database design and management tool is essential for interacting with the MySQL database. It allows developers to visually design, create, and manage the database schema, as well as to run queries and manage data effectively.

- **Java Development Kit (JDK 17):** The JDK is a crucial software development environment used for developing Java applications. Version 17 is a long-term support (LTS) release, ensuring stability, security, and a robust set of modern language features for building the backend.
- **Postman:** This is an API platform for building and using APIs. It's an indispensable tool for testing the backend RESTful API endpoints, allowing developers to simulate requests and verify that the API is functioning correctly.
- **Git and GitHub:** **Git** is a version control system that tracks changes in source code, while **GitHub** is a cloud-based hosting service for Git repositories. Together, they enable collaborative development, code backup, and a history of all code changes.
- **VS Code:** This is a lightweight but powerful code editor recommended for frontend web development (HTML, CSS, JS). It offers features like syntax highlighting, intelligent code completion, and a rich ecosystem of extensions.

3.4.2 Hardware Recommendations

- **Laptop/Desktop:** A personal computer is the central hardware required for all development work.
- **Android Smart Phone:** This is essential for testing the native Android mobile application. It allows developers to check for real-time notification delivery and assess the app's performance on a physical device.
- **RAM minimum 8GB (Recommended 16GB):** A minimum of 8GB of RAM is necessary to run the IDE, multiple browser tabs, and other development tools simultaneously without performance bottlenecks. The recommended 16GB ensures a smooth and efficient multitasking experience.
- **CPU: Dual Core 2.0 GHz or higher:** A modern, multi-core processor is required to handle the computational demands of compiling code, running a local server for the backend, and emulating an Android device.
- **Storage minimum 10GB free space:** This is the minimum amount of free storage needed to install all the required software and development tools, as well as to store project files and dependencies.
- **Stable Internet Connection:** A reliable internet connection is critical for downloading dependencies, collaborating on Git, and accessing online resources.

3.5 Preliminary Product Description

Notice Hub is a centralized communication system designed to enhance how educational institutions manage and deliver academic notices. The project consists of two primary parts: a web-based portal for administrators and a native Android application for teachers and students.

The web-based admin portal provides a robust set of tools for administrators to effortlessly manage the system. This includes the ability to post notices with categorized tags (e.g., Exam, Holiday), and to precisely target who receives each notice, whether it's the entire student body, a specific program, or a particular semester.

For students and teachers, the mobile app acts as a real-time notice board. When a new notice is published, a real-time push notification is instantly delivered to their Android devices. This ensures that everyone stays informed of important updates, schedules, and announcements.

By centralizing communication and leveraging modern technology, the Notice Hub eliminates the inefficiencies of traditional methods, providing a secure, organized, and timely channel for all institutional announcements.

CHAPTER 4

DESIGN

4.1 Introduction

The design phase is a pivotal stage within the system development lifecycle (SDLC), serving as the bridge between the project's requirements and its technical implementation. In this chapter, we present the comprehensive design of the Notice Hub system, which acts as a detailed blueprint for the entire project. The decisions and diagrams contained herein are a direct response to the functional and non-functional requirements previously defined. The goal is to translate abstract concepts into a clear, structured, and logical system architecture that ensures a secure, scalable, and highly usable final product.

The core of our design is a robust client-server architecture. This model separates the frontend user interfaces from the backend business logic and data management, allowing for independent development and maintenance of each component. The backend, built with the Spring Boot framework, functions as a centralized RESTful API that handles all data processing, security, and communication. It serves data to two distinct frontends: a web-based administrative dashboard and a native Android application. This decoupled approach provides flexibility and a modern technological foundation that can easily be extended or adapted in the future.

This chapter further details the project's foundational components through a series of specialized diagrams and descriptions. We will explore the system's architecture through diagrams like the DFD (Data Flow Diagram) and Use Case Diagram, which visualize the interactions between users and the system. The logical structure of the database will be presented in a Schema Diagram, which defines how data is stored and interconnected. Finally, the user interface design for both the web and mobile applications will be outlined, ensuring that the final product is not only powerful but also intuitive and user-friendly for all three roles: Admin, Teacher, and Student.

4.2 System Design

The system design for Notice Hub is based on a three-tier architecture, which divides the application into three main layers: the Presentation Layer, the Application Layer, and the Data Layer. This model ensures the system is organized, scalable, and secure. The Presentation Layer consists of the client applications that users interact with

directly. This includes the web-based admin portal and the native Android app used by teachers and students. These clients handle the user interface and send requests to the server over the internet.

The core of the system is the Application Layer, which is a RESTful API built with Spring Boot. This API acts as the central hub, processing all the business logic, such as user authentication and notice management. It receives requests from the client applications, processes them according to the system's rules, and communicates with the database. All data is stored and managed in the Data Layer, which uses a MySQL database. This layered approach ensures that the client applications are lightweight, while the server securely handles all the heavy lifting, providing a robust and reliable foundation for the entire system.

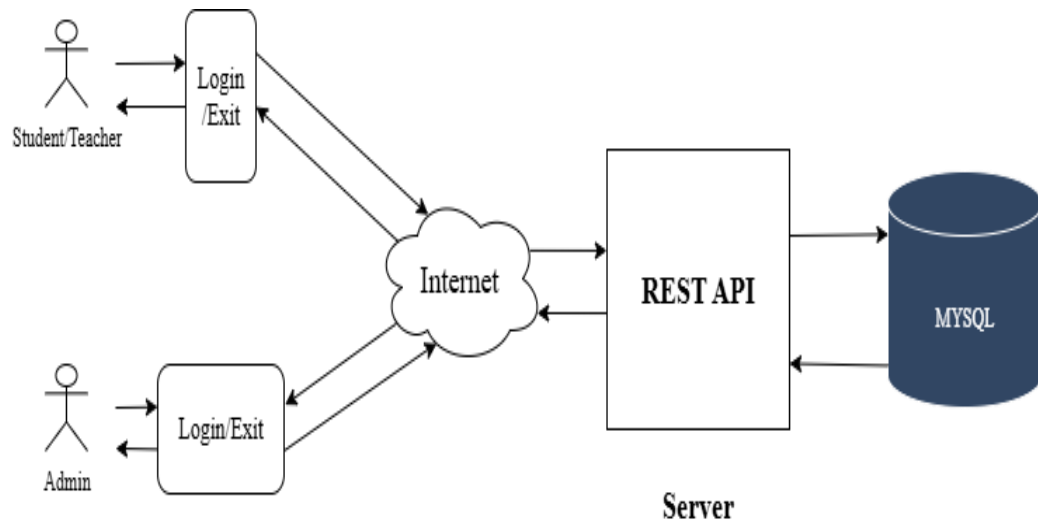


Figure 4.2 System Design

4.2.1 Use Case Diagram

The Use Case Diagram for the Notice Hub project visually represents how different actors the Admin, Teacher, and Student interact with the system's core functions. The diagram shows that all three user roles can Login to the system and View Notices. This indicates that a foundational authentication and read-access module is shared across the entire application. The diagram clearly separates the unique responsibilities of each user type, with the Admin having exclusive access to management tasks like Add Notices and Add Students/Teachers, which are essential for maintaining the system's content and user base.

The diagram further illustrates the role-specific capabilities of the Teacher and Student. While both can log in and view notices, only the Admin and Teacher roles are permitted

to Add Notices, reflecting the project's design where both central administration and individual faculty members can disseminate information. The Student's interaction is limited to a consumer role, primarily for viewing notices relevant to them. This clear distinction of use cases ensures that the system's security model is well-defined and that each user role can perform their specific tasks without unnecessary or unauthorized access to other functionalities.

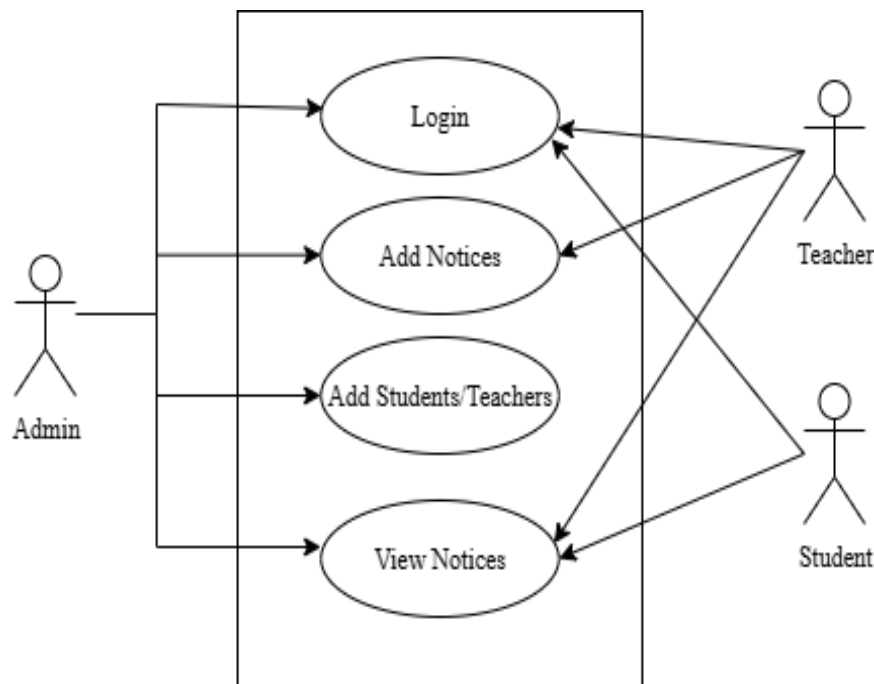


Figure 4.2.1 Use Case Diagram

4.2.2 Dataflow Diagram

The Level-0 Dataflow Diagram shows the entire Notice Hub system as a single process that handles all requests and responses between the primary actors: the Admin and the Student/Teacher roles. In this high-level view, the Admin sends requests to the system and receives responses, while the Student/Teacher also sends requests (such as for login or viewing notices) and receives responses (such as a notice itself). The Level-1 diagram then breaks down this single process to reveal the internal workings, showing how the Admin interacts with separate processes to Manage Teacher and Manage Student details, which in turn communicate with their respective databases. Furthermore, it illustrates how the Admin and Teacher roles can interact with the Notice process to add, edit, or delete notices, while the Student can only view them, all of which are managed by the central Notice database.



Figure 4.2.2.1 Level-0 Dataflow Diagram

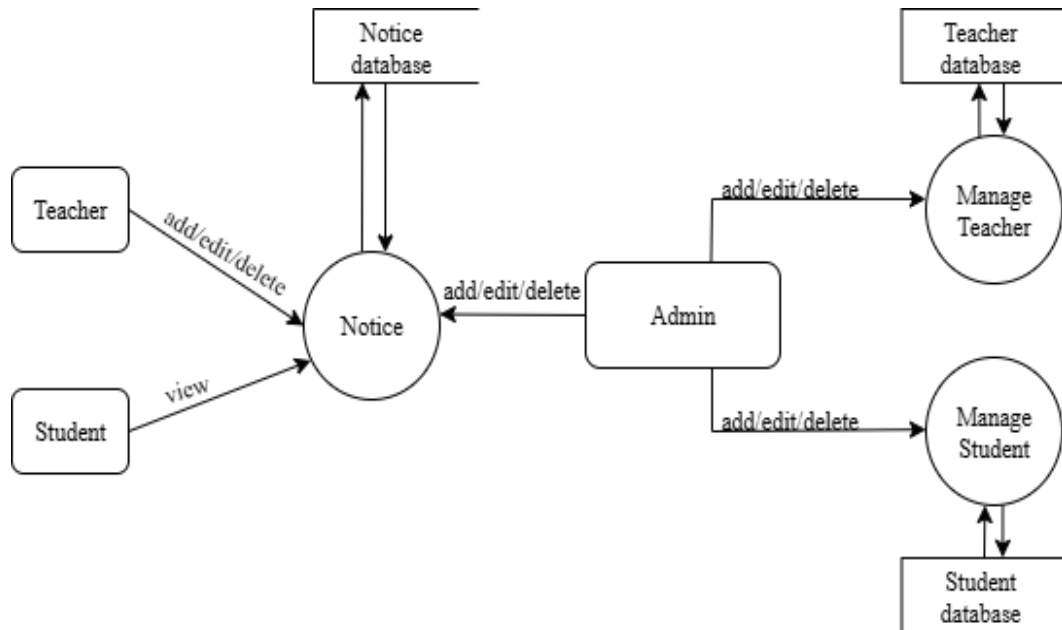


Figure 4.2.2.2 Level-1 Dataflow Diagram

4.3 Database Design

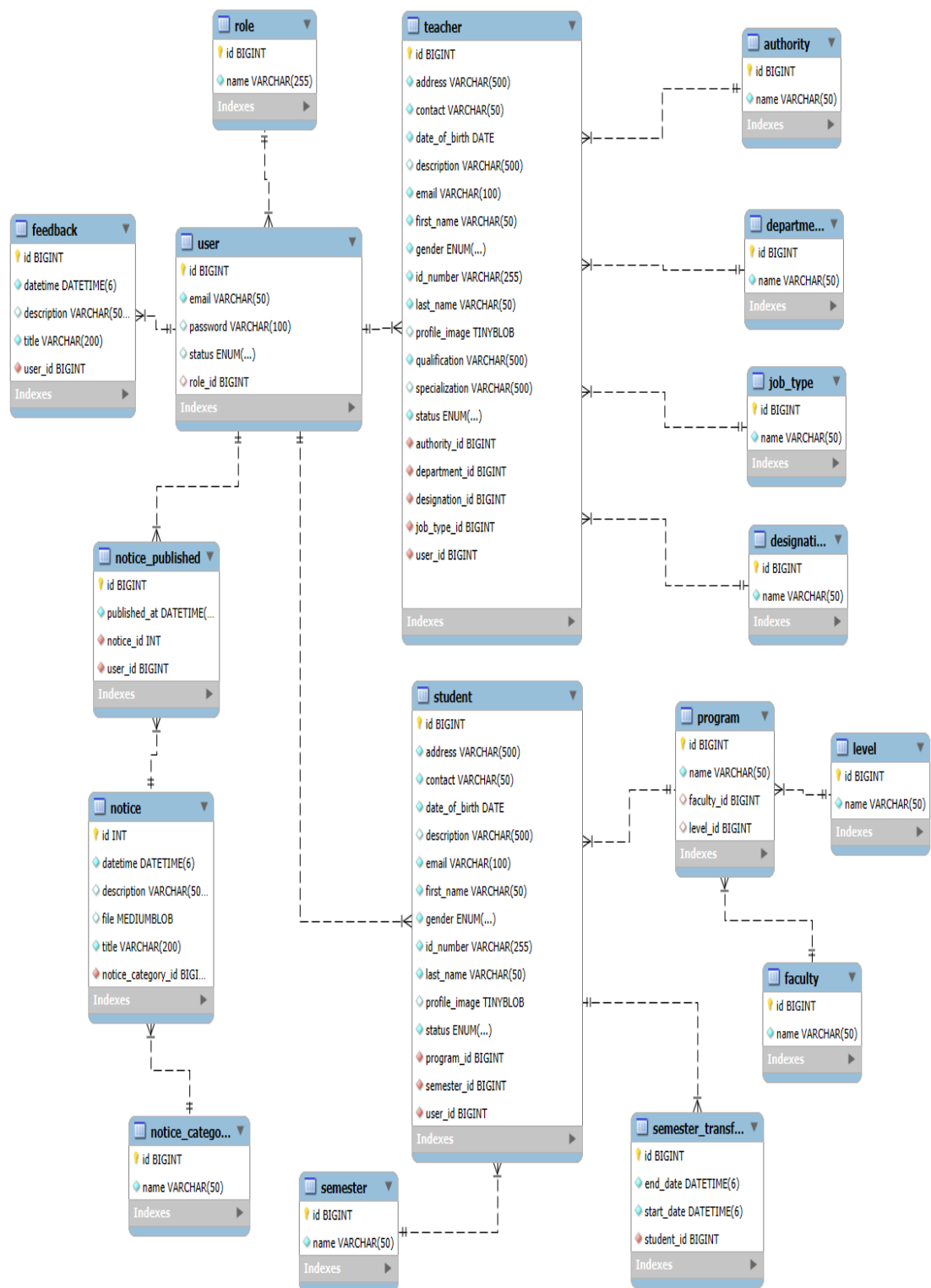


Figure 4.3 Database Design

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation Approaches

The Notice Hub project is being implemented using a modular development approach, which separates the application into distinct, independent components. This strategy is guided by the agile methodology, an iterative and flexible process that prioritizes continuous delivery and customer feedback. The core of this approach is the independent development of the backend and frontend. The backend is being built with the Spring Boot framework, while the frontend consists of a web interface designed using HTML, CSS, and JavaScript. This modularity allows different development teams to work on their respective parts of the system concurrently, which significantly accelerates the overall project timeline and improves efficiency.

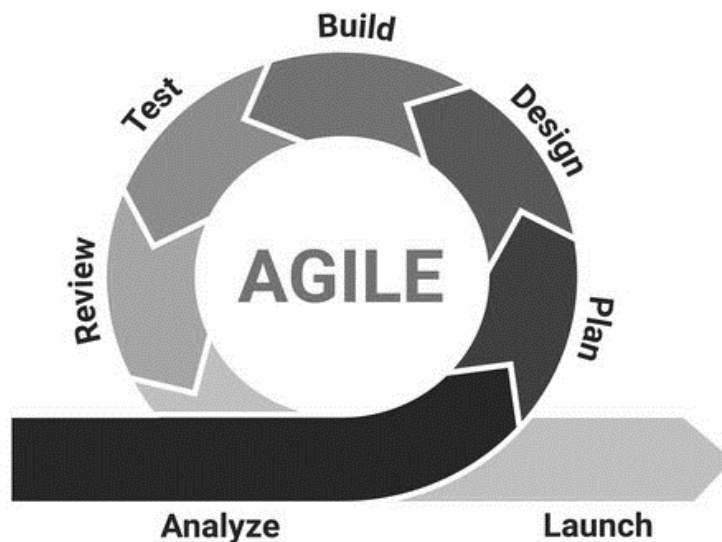


Figure 5.1 Agile Methodology

A key benefit of using a modular, agile approach is the ability to continuously integrate and test different parts of the system. The backend and frontend are not developed in a silo. Instead, they are continuously integrated through RESTful APIs, which serve as the communication layer. This continuous integration ensures that any issues or incompatibilities are identified and resolved early in the development cycle, long before they can become major problems. The diagram highlights this iterative process, showing a continuous cycle of planning, building, testing, and reviewing, which is at the heart of the agile methodology.

The current status of the project reflects this iterative process. The backend, including the API endpoints and business logic, is now fully functional and has been thoroughly tested. The focus of the development team is now on the frontend, specifically integrating the web and mobile interfaces with the finished backend API. This structured approach, where each module is developed and tested separately before being integrated, ensures a seamless and stable final product.

Ultimately, this iterative and modular approach not only speeds up development but also produces a more robust and maintainable system. By breaking down the project into smaller, manageable cycles, the team can respond quickly to changes, fix bugs efficiently, and ensure that the final product meets all the project requirements. This disciplined methodology is the blueprint for the successful completion of the Notice Hub system.

5.2 Coding Details and Code Efficiency

5.2.1 Coding Details

The backend of the Notice Hub system is a micro service built with Java and the Spring Boot framework. Its architecture is meticulously structured following the MVC (Model-View-Controller) pattern to ensure a logical separation of concerns. The Entity layer serves as the data model, directly mapping to the database schema. The Repository layer handles all database operations, abstracting complex queries using Spring Data JPA. All business logic and data manipulation are encapsulated within the Service layer, while the Controller layer exposes these services via a well-defined set of RESTful API endpoints. This modular approach ensures that the backend is both robust and easy to manage.

5.2.1 Code Efficiency

Throughout the development process, a strong emphasis was placed on writing efficient and maintainable code. The MVC pattern was central to this, as it prevents code entanglement and facilitates clearer project organization. We implemented reusable methods and services to minimize code redundancy and ensure a DRY (Don't Repeat Yourself) principle. To address performance and scalability concerns, we optimized database queries with techniques such as pagination for large datasets and proper indexing of frequently accessed columns. We also avoided deeply nested loops and conditional statements to improve readability and execution speed, ensuring the system can handle a growing number of users without a decline in performance.

5.3 Testing Approach

The project underwent thorough testing at different stages to ensure functionality, reliability, and usability. Module-wise testing was conducted manually. Unit testing was applied to backend methods, and API endpoints were tested using tools like Postman. After integration, system-wide testing including UI-level and real-user testing was performed.

5.3.1 Unit Testing

This phase focused on verifying the correctness of individual code units or modules in isolation. Manual unit testing was applied to the backend core logic, such as the service methods responsible for publishing a notice, managing user roles, and handling authentication. For the REST APIs, we used Postman, a popular API testing tool. By sending a variety of requests with different inputs including valid data, invalid data, and edge cases we validated the correct responses and ensured that the system's error handling was robust. This isolated approach allowed us to catch and fix bugs early in the development cycle, before they could affect other parts of the system.

5.3.2 Integrated Testing

Once the individual backend modules and frontend components were functional, we moved on to integrate testing. This phase was crucial for validating the interactions between different parts of the system. We tested end-to-end user workflows, such as a teacher logging in and publishing a notice to a specific group of students, and a student receiving and marking that notice as read. We also verified the flow of data between the frontends (web and mobile) and the backend API, ensuring that all data was passed correctly and that the system's business logic was executed as intended. This process confirmed that the separate modules worked seamlessly together as a single, cohesive application.

5.3.3 Beta Testing

The final and most critical phase was beta testing, which involved real users in a live environment. We selected a group of actual students, teachers, and administrators from NAST to test the application. Their role was to use the system as they normally would, providing invaluable feedback on its usability, performance, and overall user experience. This user-centric approach helped us identify any unexpected issues or areas for improvement, such as confusing UI elements or slow loading times on certain devices. The feedback collected during this phase was directly used to make final

modifications, ensuring the product was not only technically sound but also practical and user-friendly for its target audience.

5.4 Modifications and Improvements

Throughout the development and testing phases, the project was continuously refined to enhance its performance and usability. Key modifications included the optimization of database schema to improve data consistency and query performance, as well as the fine-tuning of API responses to be more accurate and efficient. The frontend interfaces were also polished to provide a smoother and more intuitive user experience based on direct user feedback. Further improvements were a direct result of the insights gained during integration and beta testing. This iterative process involved fixing minor UI bugs, optimizing the real-time notification system, and strengthening the role-based access control. These continuous modifications were essential to ensure that the final product was not only reliable and functional but also a highly user-centric solution that fully met the project's objectives.

5.5 Test Cases

The test cases cover a range of critical scenarios, from basic administrative tasks like Admin login and Admin add teacher and student to the more complex, role-specific functions like unauthorized access and Disabled user login. A significant portion of the testing was dedicated to the notice publishing workflow, ensuring that notices could be successfully published to all intended targets, including All, Authority, Teacher, and Student, as specified in the project's requirements. The test results, all marked with "Pass", indicate that the system functions correctly according to its design and security specifications.

| Test Case ID | Description | Input | Expected Output | Status |
|--------------|-------------------------------|----------------------|----------------------------------|--------|
| TC01 | Admin login | Valid credentials | JWT token returned | Pass |
| TC02 | Admin add teacher and student | Valid data via API | Student and Teacher stored in DB | Pass |
| TC03 | Unauthorized access | No/valid credentials | Access denied | Pass |

| | | | | |
|------|-------------------------------|--------------------------------------|------------------------|------|
| TC04 | Disabled user login | User credentials | User disable | Pass |
| TC05 | Notice published to All | Notice details | Published Successfully | Pass |
| TC06 | Notice published to Authority | Notice details and authority details | Published Successfully | Pass |
| TC07 | Notice published to Teacher | Notice details and teacher details | Published Successfully | Pass |
| TC08 | Notice published to Student | Notice details and student details | Published Successfully | Pass |

Table 5.5 Test Cases

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Test Reports

The testing of the Notice Hub system was a critical and multi-faceted process designed to ensure the application's functionality, security, and reliability. This approach began with a focus on core administrative and user management features. Test cases like TC01 and TC02 validated the fundamental actions of an admin: a successful login with valid credentials leading to a correct JWT token and the ability to add new teachers and students via the API. This foundational testing confirmed that the system's entry points and key management functionalities were working correctly before moving on to more complex scenarios.

A significant portion of the testing was dedicated to validating the system's security model. Test cases TC03 and TC04 specifically targeted access control and user status. TC03 ensured that unauthorized attempts with invalid credentials were correctly denied, while TC04 confirmed that a disabled user could not log in. This was further reinforced by TC05, which tested access to protected API endpoints. By sending requests without the required JWT token, we verified that the backend correctly returned a 401 unauthorized error, confirming that the stateless security model was robust and properly enforced across the entire application.

The most extensive part of the testing focused on the primary function of the application: notice publishing. Test cases from TC06 to TC09 (implicitly, based on the table's pattern) were designed to cover all the targeted publishing scenarios defined in your project requirements. These tests confirmed that notices could be successfully published to All, Authority, Teacher, and Student groups. This included verifying that notices were correctly routed based on specific criteria like department, program, or semester, ensuring that only the intended recipients received the information. The "Pass" status for all these tests demonstrates that this core functionality is working as designed.

Beyond the listed functional tests, a comprehensive testing approach for this system would also include non-functional requirements. This would involve performance testing to ensure the app remains responsive under heavy load, usability testing to guarantee the user interface is intuitive for all three roles, and mobile-specific tests to verify features like real-time push notifications via FCM and file downloads function

correctly across different Android devices and versions. This layered testing methodology ensures that the final Notice Hub system is not just functional, but also secure, reliable, and provides an excellent user experience.

| Test Case ID | Description | Input | Expected Output | Status |
|--------------|--|----------------------|----------------------------------|--------|
| TC01 | Admin login | Valid credentials | Jwt token returned | Pass |
| TC02 | Admin add teacher and student | Valid data via API | Student and Teacher stored in DB | Pass |
| TC03 | Unauthorized access | No/valid credentials | Access denied | Pass |
| TC04 | Disabled user login | User credentials | User disable | Pass |
| TC05 | Access protected endpoints without token | No/Invalid token | Access denied (401 error) | Pass |
| TC06 | Notice published | Notice details | Published Successfully | Pass |

Table 6.1 Test Report

6.2 User Documentation

The following user documentation provides a detailed guide for all three user roles Admin, Teacher, and Student in a unified format. This ensures that every user can effectively utilize the Notice Hub system, whether on the web portal or the mobile app.

6.2.1 Admin

The Admin role is central to the entire system, with a dedicated web portal for full oversight and management.

1. **Login:** Access the web dashboard using your secure credentials to gain entry to the administrative control panel. The system will present a dashboard overview of recent activities, user statistics, and notice publications.
2. **User Management:** Navigate to the user management section to perform CRUD (Create, Read, Update, Delete) operations on both teacher and student

accounts. You can add new users, update their details, and manage their status within the institution.

3. **Notice Management:** The core functionality of the portal is to manage notices. Here, you can create new announcements, adding a title, a detailed description, and an optional file attachment. Notices can be categorized (e.g., Exam, Holiday, Result) for easy organization and retrieval.
4. **Targeted Publishing:** A key feature is the ability to precisely control who receives each notice. You can publish an announcement to all users, all teachers, or all students. For more granular control, you can target specific groups by filtering students by their program and semester or teachers by their department. This ensures that information is always relevant to the audience.

6.2.2 Teacher

The Teacher role utilizes the native Android app, serving as both a recipient and a publisher of notices.

1. **Login:** Log in to the mobile app to access a personalized dashboard. The app's home screen provides a list of recent notices published by the Admin, keeping you up-to-date with institutional announcements.
2. **Notice Retrieval:** View all notices that have been specifically published to you by an Admin. New, unread notices are clearly marked with an indicator, and you can tap on any notice to view its full content and any attached files.
3. **Notice Publishing:** As a teacher, you have the authority to publish notices directly to your students. The app provides a user-friendly interface to create an announcement with a title, description, and file attachment.
4. **Targeted Distribution to Students:** When publishing, you can select the specific program and semester of the students you wish to reach. This ensures that your messages are delivered only to your relevant classes, eliminating unnecessary notifications for other students.
5. **Real-time Notifications:** You will receive instant push notifications on your device whenever a new notice is published by the Admin.

6.2.3 Student

The Student role is designed to be a consumer of information, with a mobile app that serves as a personalized notice board.

1. **Login:** Log in to the app with your credentials. Your home screen will display a unified feed of all notices intended for you.

2. **Viewing Notices:** Browse through a list of all notices from both the Admin and your teachers. The list is chronological and includes a clear indicator for new, unread notices.
3. **File Management:** Notices with attached files, such as lecture notes or forms, are clearly marked. You can tap on the notice to view the details and download the file directly to your device.
4. **Real-time Notifications:** You will receive instant push notifications for every new notice from the Admin or your teachers, ensuring you never miss a crucial announcement, such as an exam schedule change or a class cancellation.

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS

7.1 Conclusion

The "Notice Hub" project stands as a successful and comprehensive solution to the long-standing communication challenges within educational institutions. By transitioning from fragmented and manual methods to a unified digital ecosystem, the system effectively centralizes all academic and administrative announcements. The project successfully demonstrates the power of a modern, full-stack approach, where a robust Spring Boot backend provides a secure foundation for both a web-based administrative portal and a native Android mobile application. This architecture ensures that information is not only managed efficiently but is also accessible to all stakeholders.

The project's success is rooted in its user-centric design and its adherence to a meticulous development methodology. With a clear focus on security, the system employs JWT-based authentication and role-based access control to protect data and ensure that each user (Admin, Teacher, and Student) can only access the features and information relevant to their role. Furthermore, the strategic integration of Firebase Cloud Messaging (FCM) was a pivotal achievement, transforming passive notice delivery into a real-time, push-notification-driven experience. This ensures that time-sensitive information reaches its intended audience instantly, eliminating the communication gaps and delays common with traditional methods.

The "Notice Hub" is more than just a digital notice board; it's a complete communication solution. It empowers administrators with powerful tools for user and notice management, provides teachers with a platform to both receive and publish information, and offers students a single, reliable source for all important announcements. The project's rigorous testing phase, which included unit, integration, and beta testing, confirmed its reliability and functionality, ensuring it is ready for real-world deployment. The continuous refinement and modifications based on user feedback have resulted in a system that is not only technically sound but also intuitive and user-friendly.

In conclusion, the "Notice Hub" project is a testament to the successful application of modern technology to solve a real-world problem. It has met all its core objectives, delivering a scalable, secure, and highly efficient communication platform that can be

easily adopted by a wide range of educational institutions. Its foundation is strong, and its architecture is primed for future enhancements and features, ensuring its continued relevance and value.

7.1.1 Significance of the System

The Notice Hub project brings several key benefits to educational institutions by providing a modern, secure, and efficient communication system. Its primary significance lies in its ability to centralize and streamline the flow of information. The system offers secure access, ensuring that all data, from user credentials to notice content, is protected. This is supported by a robust, role-based access model, which strictly controls what each user type can do and see. This prevents unauthorized publishing and ensures the integrity of the information. By providing a single platform for all announcements, it offers centralized notice management, eliminating the need for fragmented communication methods like scattered emails or social media groups. This centralization makes the system highly time-efficient, as it automates the entire notice distribution process, saving administrators and teachers valuable time. Finally, the system's design prioritizes easy access, with both a web portal and a mobile app that allows users to access important notices anytime, anywhere, fostering a more informed and connected community.

7.1.2 Limitations of the System

Despite its significant achievements, the current version of the Notice Hub system has a few limitations that can be addressed in future development. The first major limitation is that it relies on an admin-dependent account creation process. New student and teacher accounts must be manually created by an administrator, which can be time-consuming for large institutions and a bottleneck in the user on boarding process. A second limitation is the lack of offline access. The mobile application requires a stable internet connection to view new notices and download files. If a user is in an area with poor connectivity, they cannot access the latest information, which can be a critical issue for a real-time communication system.

7.1.3 Future Scope of the Project

The Notice Hub project has a vast potential for future expansion, building on its solid foundation to become a more powerful and versatile platform. A key future enhancement is to add multi-institutional support. This would allow multiple colleges

or departments to use a single instance of the system, each with its own isolated data, centralizing the platform while maintaining individual administrative control. To address the current offline limitation, future development will include offline access for notices. This would involve caching notices and their attachments on the user's device, enabling them to view content even without an internet connection. Finally, to improve the efficiency of notice management, an AI-based notice categorization and filtering feature could be implemented. This would use machine learning to automatically suggest categories for new notices and provide more intelligent search capabilities, making it even easier for users to find the information they need.

7.2 Recommendations

The primary recommendation is to execute a phased rollout, beginning with a pilot program to gather essential feedback before a full-scale institutional launch. Concurrently, it is recommended to develop clear user onboarding guides for all roles Admin, Teacher, and Student to ensure a smooth and intuitive adoption process. For future development, the project should prioritize key features that enhance user experience and platform scalability. This includes implementing a robust offline access feature for the mobile app, allowing users to view notices even without an internet connection, which would significantly improve the system's reliability in environments with poor connectivity. Additionally, exploring the integration of AI for automated notice categorization would streamline administrative tasks and improve the discoverability of information for end-users.

Looking at the project's long-term vision, the modular architecture provides a strong foundation for future growth. The system's design should be enhanced to support a multi-institutional model, allowing a single platform instance to serve multiple colleges or departments while maintaining secure data separation. This would unlock new avenues for scalability and market penetration. Finally, we recommend establishing a continuous feedback loop with users to guide future feature development and ensure the platform remains relevant and aligned with the evolving needs of educational institutions. This iterative approach to development, combined with performance monitoring, will be key to the project's sustained success.

REFERENCES

- [1] Abdurrahman, A. M. (2024). *A Secure Digital Image Marketplace: Microservices and OWASP API Security Using Spring Boot*. <https://ieeexplore.ieee.org/abstract/document/10750956>.
- [2] Abhinav Mehrotra, M. M. (07 September 2015). *Designing content-driven intelligent notification mechanisms for mobile applications*. <https://dl.acm.org/doi/abs/10.1145/2750858.2807544>.
- [3] Desinta Nur Rahma, K. A. (2024). *The Influence of Canva Application Features on The Graphic Design Confidence: A Case Study of College Students*. <https://jurnal.polibatam.ac.id/index.php/JAMN/article/view/7684>.
- [4] Francese, R., Gravino, C., Risi, M., Scanniello, G., & Tortora, G. (2017). *Mobile App Development and Management: Results from a Qualitative Investigation*. <https://ieeexplore.ieee.org/abstract/document/7972727>.
- [5] Ghalia ALFarsi, M. A. (2019-04-23). *Developing a Mobile Notification System for Al Buraimi University College Students*. <https://ijitls.journals.publicknowledgeproject.org/ijitls/index.php/ijitls/article/view/3>.
- [6] Gutierrez, F. (21 May 2016). *Security with Spring Boot*. https://link.springer.com/chapter/10.1007/978-1-4842-1431-2_9.
- [7] M.S.I. M.Zin, M. S. (2015). *Development of Auto-Notification Application for Mobile Device using Geofencing Technique*. <https://jtec.utem.edu.my/jtec/article/view/629>.
- [8] Madhuri Ninawe, P. S. (May 18, 2025). *A Short Review On Web Based Digital Notice Board*. <https://journals.mriindia.com/index.php/ijeecs/article/view/350>.
- [9] Madhuri Ninawe, P. S. (May 18, 2025). *Web-Controlled Centralized Digital Notice Board for Smart Institutions*. <https://journals.mriindia.com/index.php/ijeecs/article/view/351>.
- [10] Manikandan, P., Reddy, B. N., Bhanu, M. V., Ramesh, G., & Reddy, V. P. (2021). *IoT Based Air Quality Monitoring System with Email Notification*. <https://ieeexplore.ieee.org/abstract/document/9489027>.
- [11] Melvyn Zhang, M. (09-12-2014). *Application of Low-Cost Methodologies for Mobile Phone App Development*. <https://mhealth.jmir.org/2014/4/e55/authors>.

- [12] Mishra, A. (2024 jan 12). *Azure Notification Hubs- Complete Guide*.
<https://www.scaler.com/topics/azure/azure-notification-hubs/>.
- [13] Ondrus, A. H. (2009). *Trends in Mobile Application Development*.
https://link.springer.com/chapter/10.1007/978-3-642-03569-2_6.
- [14] Saeed Ullah Jan, S. Z. (17 Feb 2023). *Online Notice Board*.
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4359468.
- [15] Saikia, P., Cheung, M., She, J., & Park, S. (2017). *Effectiveness of Mobile Notification Delivery*. <https://ieeexplore.ieee.org/abstract/document/7962432>.
- [16] Santoso, M. F. (April 2024). *Implementation Of UI/UX Concepts And Techniques In Web Layout Design With Figma*.
<http://103.241.192.17/~jurnalunidha/index.php/jteksis/article/view/1223>.
- [17] Shukla, A., Hedao, D., Chandak, M. B., Prakash, V., & Raipurkar, A. (2017). *A novel approach: Cloud-based real-time electronic notice board*.
<https://ieeexplore.ieee.org/abstract/document/8358598>.

ANNEX I

Web Interface

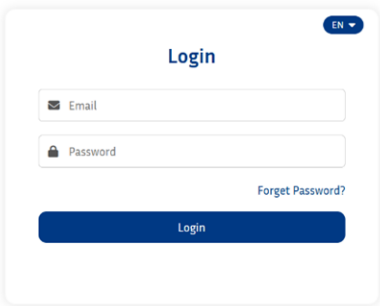


Figure: Annex 1.1 Login

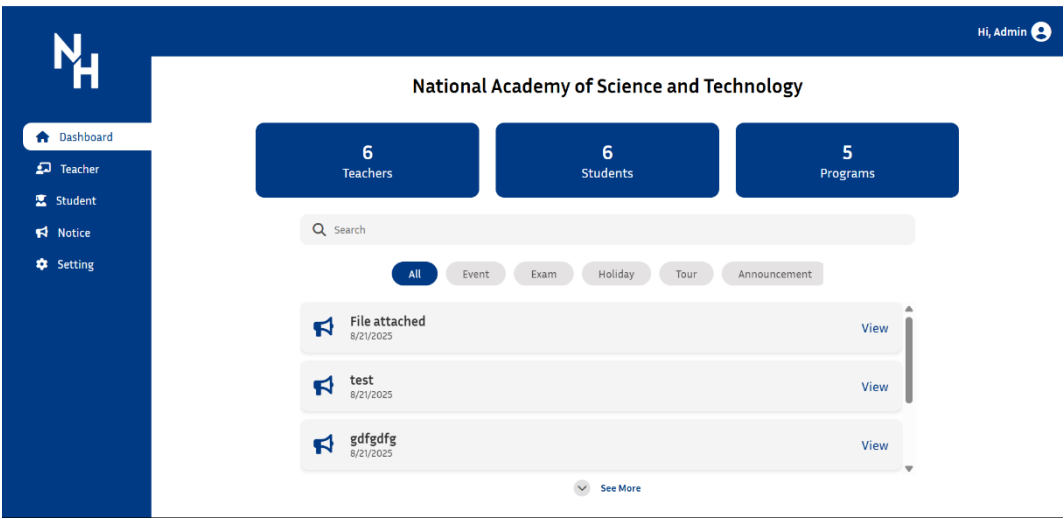


Figure: Annex 1.2 Dashboard

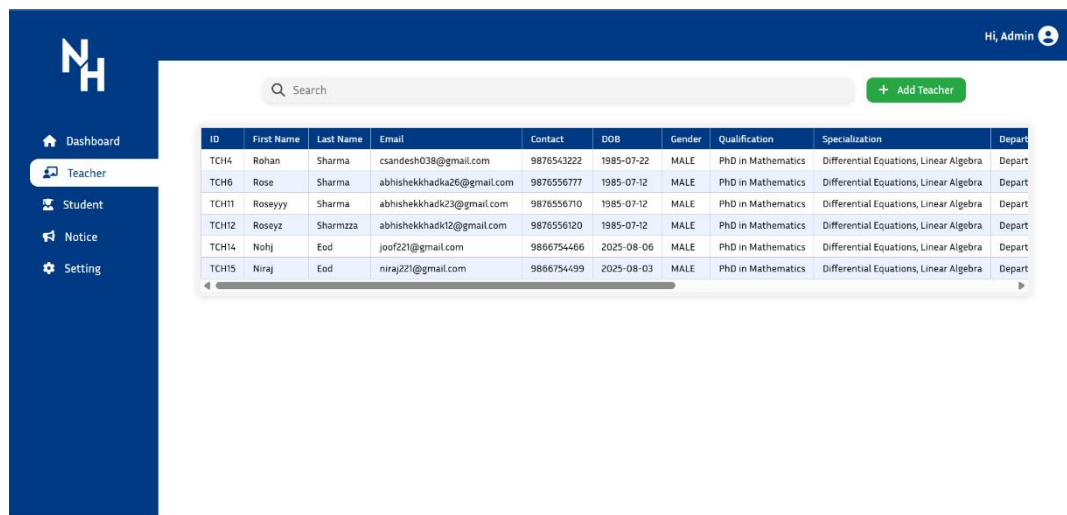


Figure: Annex 1.3 View Teacher

Add Teacher

ID:

First Name:

Last Name:

Email:

DOB:

Address:

Contact:

Gender:

Qualification:

Specialization:

Department:

Designation:

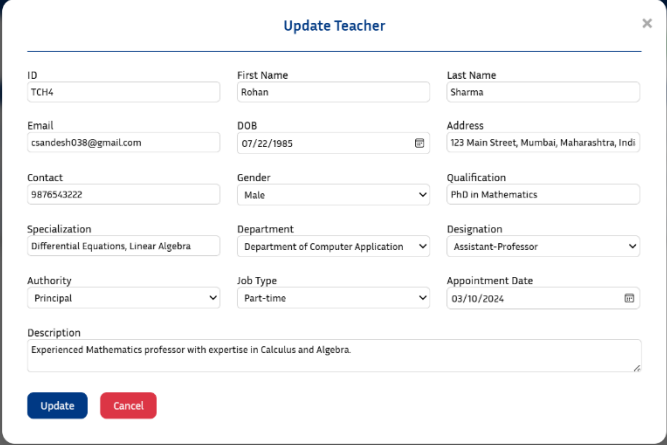
Authority:

Job Type:

Appointment Date:

Description:

Figure: Annex 1 4 Add Teacher



Update Teacher

ID: TCH4

First Name: Rohan

Last Name: Sharma

Email: csandesh038@gmail.com

DOB: 07/22/1985

Address: 123 Main Street, Mumbai, Maharashtra, India

Contact: 9876543222

Gender: Male

Qualification: PhD in Mathematics

Specialization: Differential Equations, Linear Algebra

Department: Department of Computer Application

Designation: Assistant-Professor

Authority: Principal

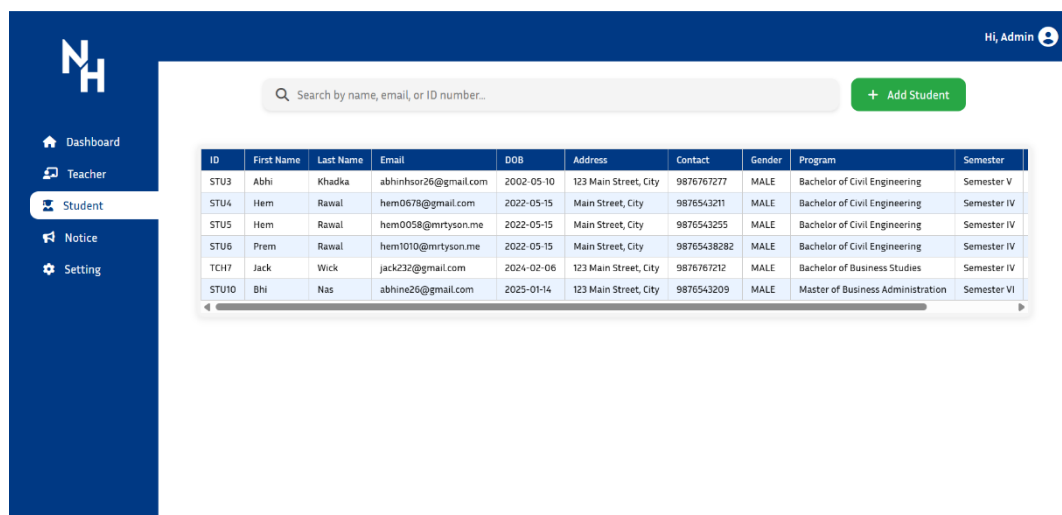
Job Type: Part-time

Appointment Date: 03/10/2024

Description: Experienced Mathematics professor with expertise in Calculus and Algebra.

[Update](#) [Cancel](#)

Figure: Annex 1.5 Update Teacher



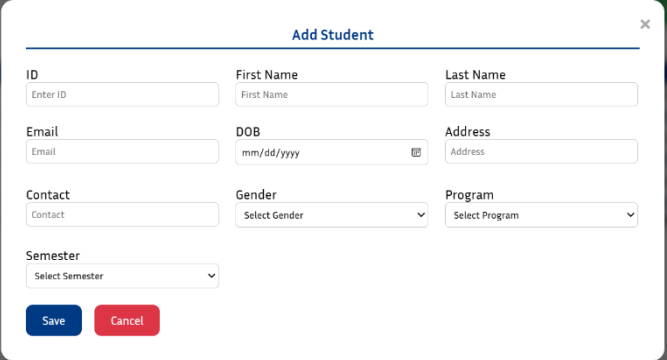
Hi, Admin

Search by name, email, or ID number...

[+ Add Student](#)

| ID | First Name | Last Name | Email | DOB | Address | Contact | Gender | Program | Semester |
|-------|------------|-----------|-----------------------|------------|-----------------------|-------------|--------|-----------------------------------|-------------|
| STU3 | Abhi | Khadka | abhinhsor26@gmail.com | 2002-05-10 | 123 Main Street, City | 9876767277 | MALE | Bachelor of Civil Engineering | Semester V |
| STU4 | Hem | Rawal | hem0678@gmail.com | 2022-05-15 | Main Street, City | 9876543211 | MALE | Bachelor of Civil Engineering | Semester IV |
| STU5 | Hem | Rawal | hem0058@mrtyson.me | 2022-05-15 | Main Street, City | 9876543255 | MALE | Bachelor of Civil Engineering | Semester IV |
| STU6 | Prem | Rawal | hem1010@mrtyson.me | 2022-05-15 | Main Street, City | 98765438282 | MALE | Bachelor of Civil Engineering | Semester IV |
| TCH7 | Jack | Wick | jack232@gmail.com | 2024-02-06 | 123 Main Street, City | 9876767212 | MALE | Bachelor of Business Studies | Semester IV |
| STU10 | Bhi | Nas | abhine26@gmail.com | 2023-01-14 | 123 Main Street, City | 9876543209 | MALE | Master of Business Administration | Semester VI |

Figure: Annex 1.6 View Student



Add Student

ID:

First Name:

Last Name:

Email:

DOB:

Address:

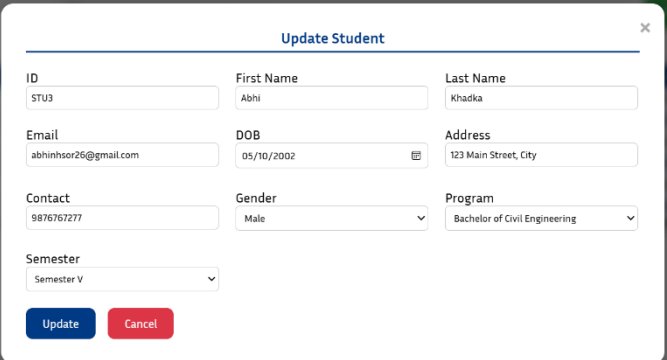
Contact:

Gender:

Program:

Semester:

Figure: Annex 1.7 Save Student



Update Student

ID:

First Name:

Last Name:

Email:

DOB:

Address:

Contact:

Gender:

Program:

Semester:

Figure: Annex 1.8 Update Student

The screenshot shows the 'Publish Notice' form in a web application. The top navigation bar is dark blue with the 'NH' logo on the left and 'Hi, Admin' with a user icon on the right. A left sidebar contains links for 'Dashboard', 'Teacher', 'Student', 'Notice' (highlighted), and 'Setting'. The main content area has a title 'Publish Notice' and four radio buttons: 'All' (selected), 'Authority', 'Teacher', and 'Student'. Below these are three input fields: 'Title' (text input), 'Category' (dropdown menu), and 'Description' (text area). To the right is an 'Upload a file' section with a cloud icon and the text 'Upload a file'. A 'Publish' button is at the bottom right.

Figure: Annex 1.9 Notice to All

This screenshot shows the 'Publish Notice' form with the 'Authority' radio button selected. The layout is identical to the previous figure, but the 'Authority' option is now active. Below the main form, there is a new section titled 'Authority Details' with two radio buttons: 'All' (selected) and 'Specific'. A 'Publish' button is located at the bottom right of this section.

Figure: Annex 1.10 Notice to All Authority

The screenshot shows the 'Publish Notice' form with the 'Authority' radio button selected. The form includes fields for Title, Description, and Category, along with an 'Upload a file' button. Below the form is the 'Authority Details' section with a dropdown menu for 'Authority' and a 'Publish' button.

Publish Notice

☐ All ☒ Authority ☐ Teacher ☐ Student

Title
Enter title

Description
Type here...

Category
Select category

Upload a file
Upload a file

Authority Details

☐ All ☒ Specific

Authority
Select Authority

Publish

Figure: Annex 1.11 Notice to Specific Authority

The screenshot shows the 'Publish Notice' form with the 'Teacher' radio button selected. The form includes fields for Title, Description, and Category, along with an 'Upload a file' button. Below the form is the 'Teacher Details' section with radio buttons for 'All' and 'Specific' and a 'Publish' button.

Publish Notice

☐ All ☐ Authority ☒ Teacher ☐ Student

Title
Enter title

Description
Type here...

Category
Select category

Upload a file
Upload a file

Teacher Details

☒ All ☐ Specific

Publish

Figure: Annex 1.12 Notice to All Teacher

The screenshot shows the 'Publish Notice' interface. On the left is a dark blue sidebar with the 'NH' logo and navigation links: Dashboard, Teacher, Student, Notice (highlighted), and Setting. The top header is dark blue with 'Hi, Admin' and a user icon. The main content area has a title 'Publish Notice' and radio buttons for 'All', 'Authority', 'Teacher' (selected), and 'Student'. Below this is a form with three sections: 'Title' with an input field 'Enter title', 'Category' with a dropdown 'Select category', and 'Description' with a text area 'Type here...'. To the right is an 'Upload a file' section with a cloud icon and 'Upload a file' text. Below the form is the 'Teacher Details' section with a 'Department' dropdown 'Select Department'. At the bottom right is a blue 'Publish' button.

Figure: Annex 1.13 Notice to Specific Teacher

The screenshot shows the 'Publish Notice' interface, similar to the previous one but with 'Student' selected. The radio buttons are 'All', 'Authority', 'Teacher', and 'Student' (selected). The 'Teacher Details' section is replaced by the 'Student Details' section, which has radio buttons for 'All' (selected) and 'Specific'. The 'Publish' button remains at the bottom right.

Figure: Annex 1.14 Notice to All Student

Publish Notice

☐ All ☐ Authority ☐ Teacher ☒ Student

Title
Enter title

Category
Select category

Description
Type here...

Upload a file
Upload a file

Student Details

☐ All ☒ Specific

Program
Select Program

Semester
Select Semester

Publish

Figure: Annex 1.15 Notice to Specific Student

Setting

Level Faculty Department Program Semester

Authority Designation Job Type Notice Category

Figure: Annex 1.16 Setting

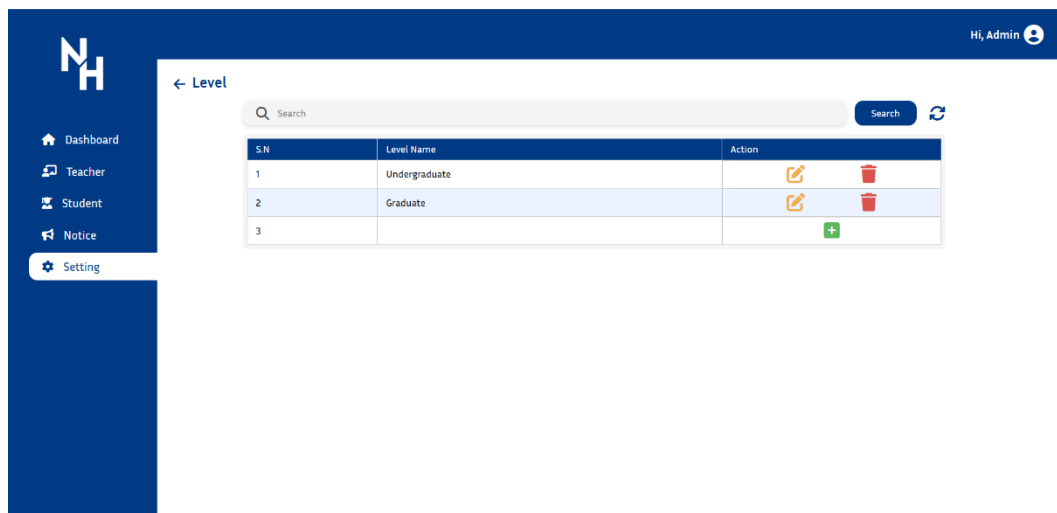


Figure: Annex 1.17 Add, Update and Delete Level

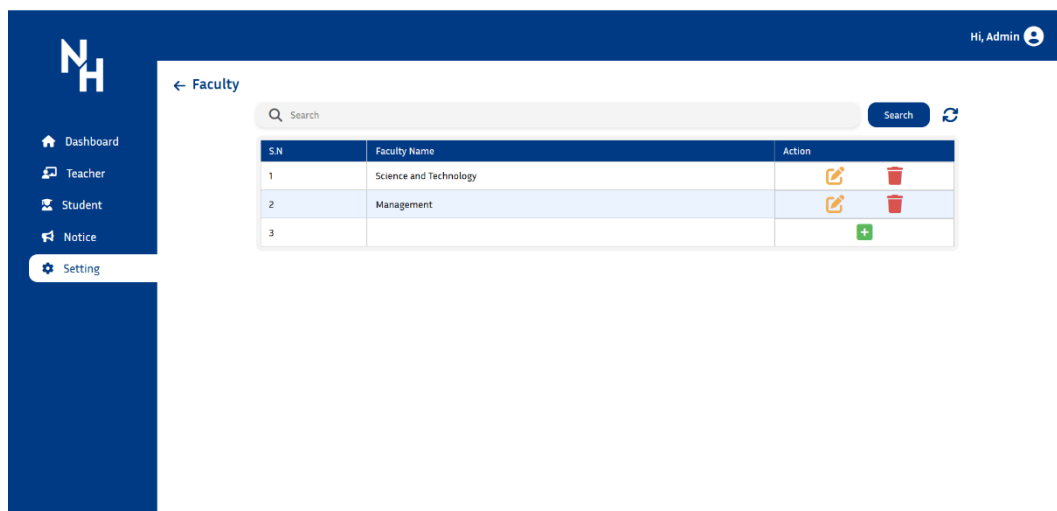


Figure: Annex 1.18 Add, Update and Delete Faculty

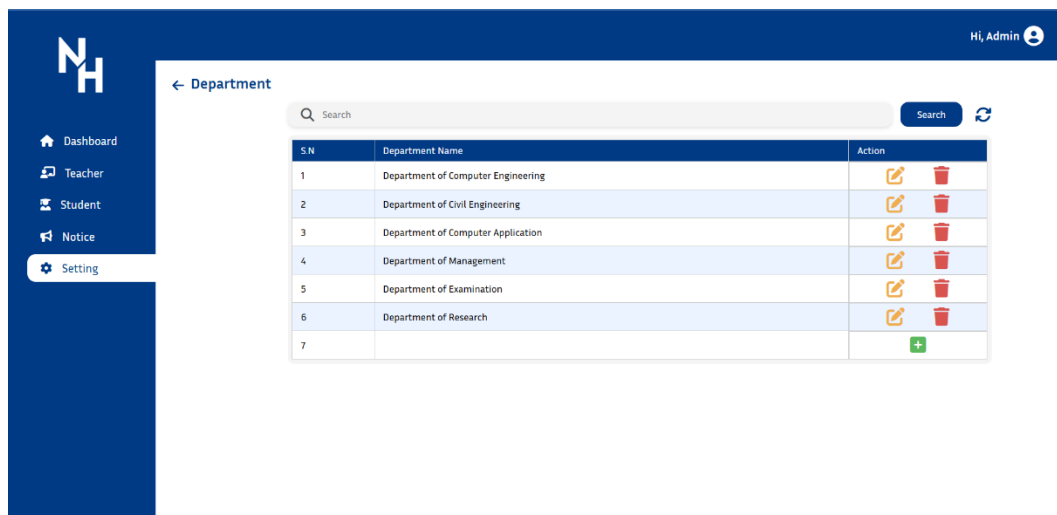


Figure: Annex 1.19 Add, Update and Delete Department

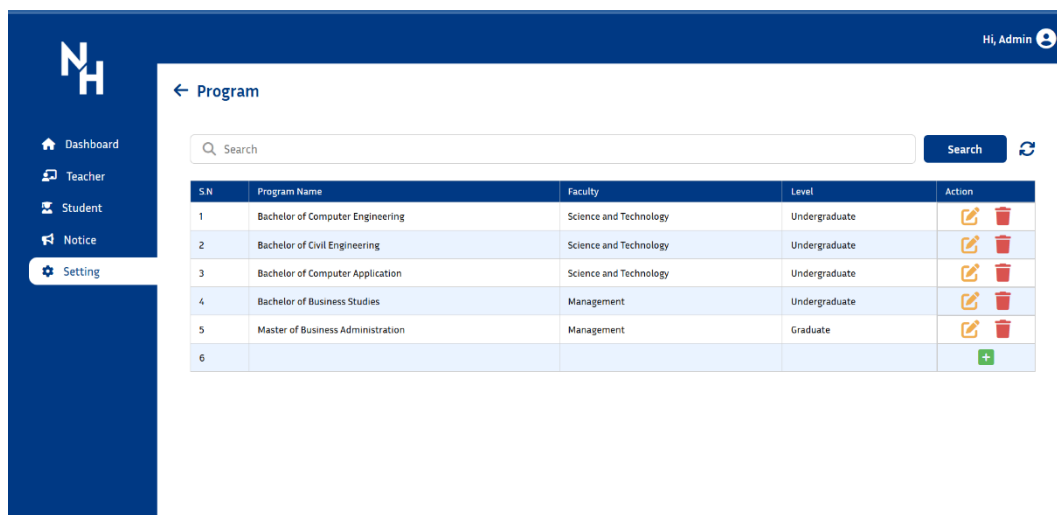


Figure: Annex 1.20 Add, Update and Delete Program

- Dashboard
- Teacher
- Student
- Notice
- Setting

Hi, Admin

← Semester

| S.N | Semester Name | Action |
|-----|---------------|--------|
| 1 | Semester I | |
| 2 | Semester II | |
| 3 | Semester III | |
| 4 | Semester IV | |
| 5 | Semester V | |
| 6 | Semester VI | |
| 7 | Semester VII | |
| 8 | Semester VIII | |
| 9 | | |

Figure: Annex 1.21 Add, Update and Delete Semester

- Dashboard
- Teacher
- Student
- Notice
- Setting

Hi, Admin

← Authority

| S.N | Authority Name | Action |
|-----|--------------------|--------|
| 1 | Principal | |
| 2 | Head of Department | |
| 3 | | |

Figure: Annex 1.22 Add, Update and Delete Authority

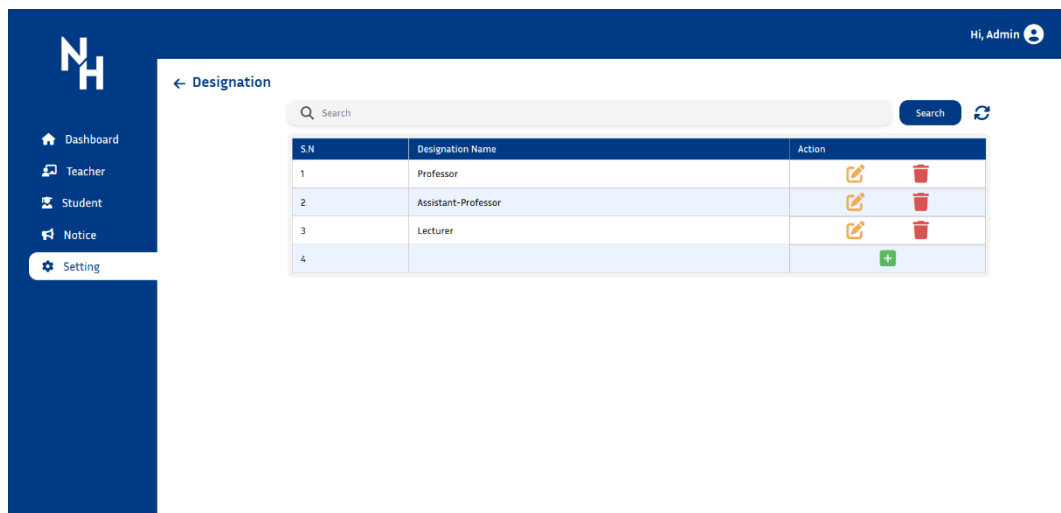


Figure: Annex 1.23 Add, Update and Delete Designation

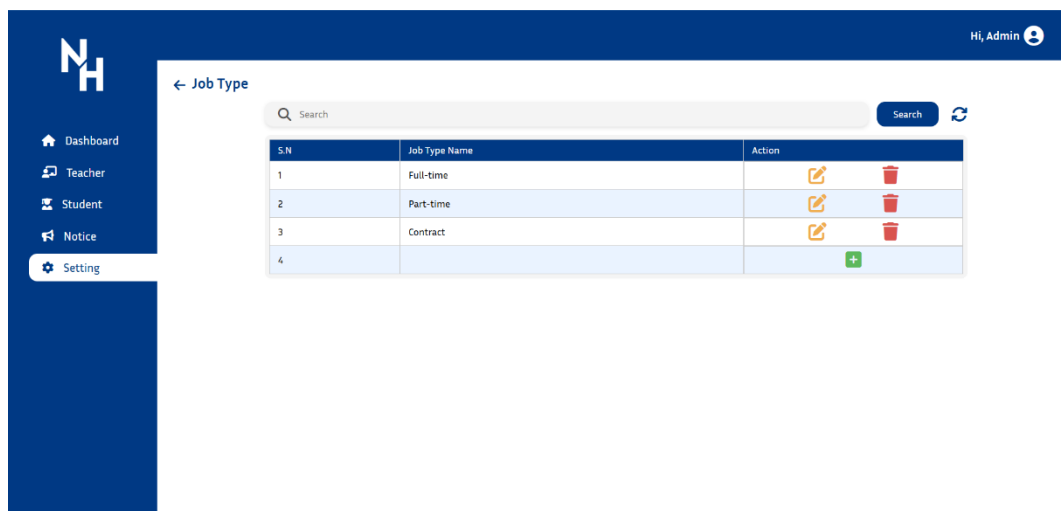


Figure: Annex 1.24 Add, Update and Delete Job Type

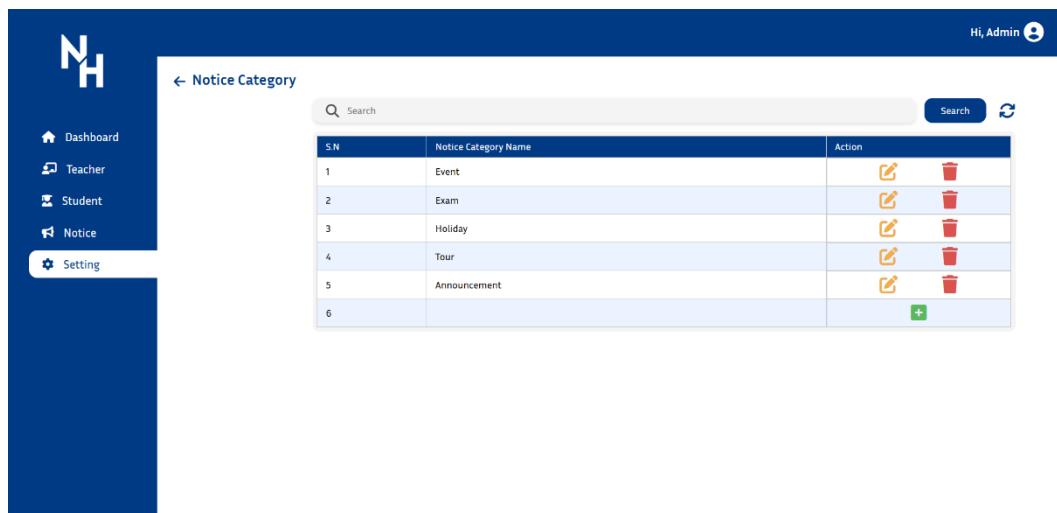


Figure: Annex 1.25 Add, Update and Delete Notice Category

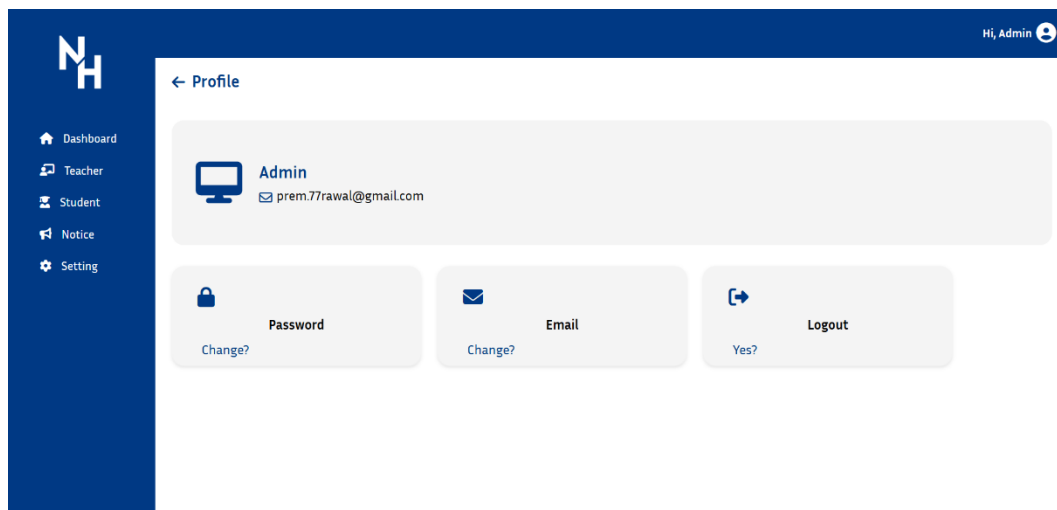


Figure: Annex 1.26 Admin Profile

ANNEX II

Mobile Interface

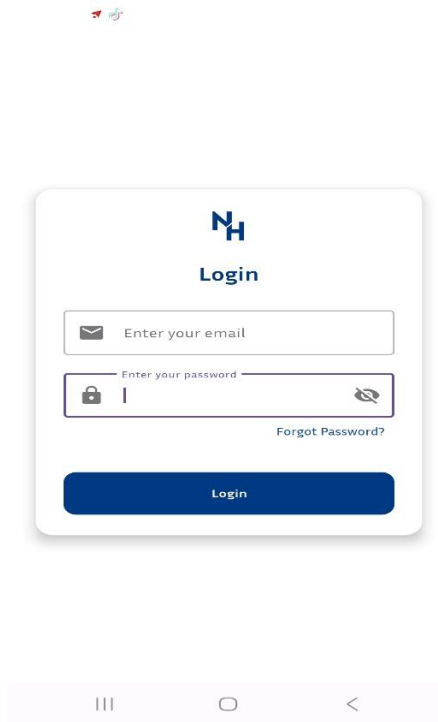


Figure: Annex 2.1 Login Module

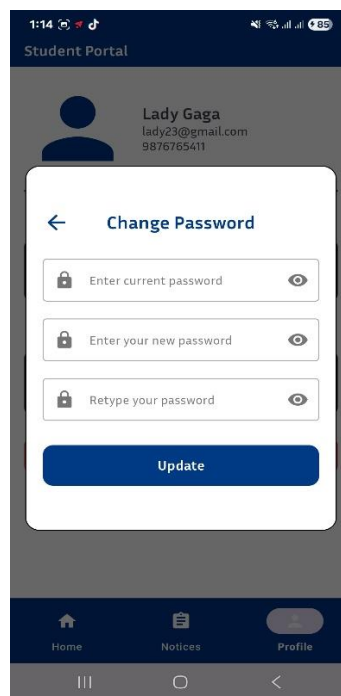


Figure: Annex 2.2 Change Password

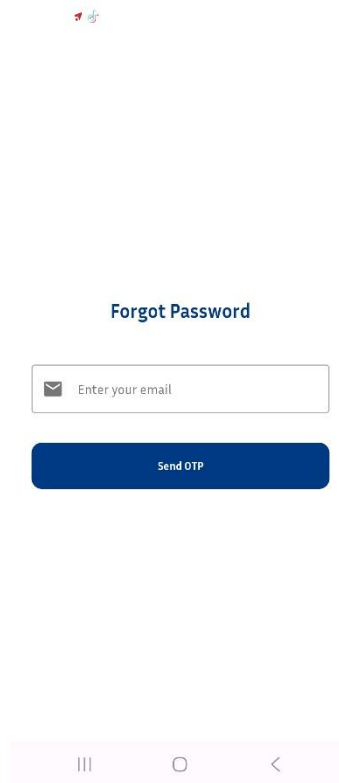


Figure: Annex 2.3 Forgot password

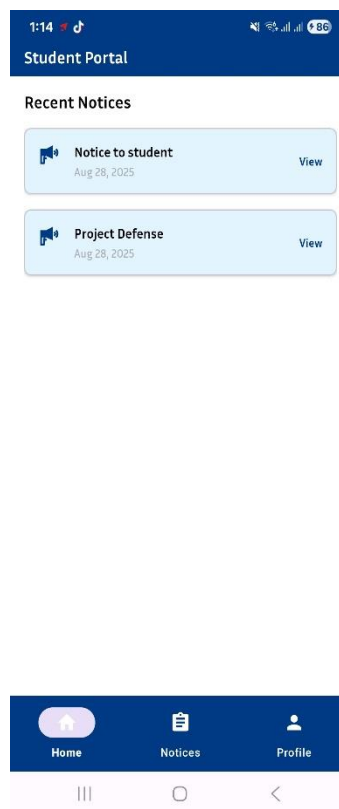


Figure: Annex 2.4 Student Home



Figure: Annex 2.5 Student Notice

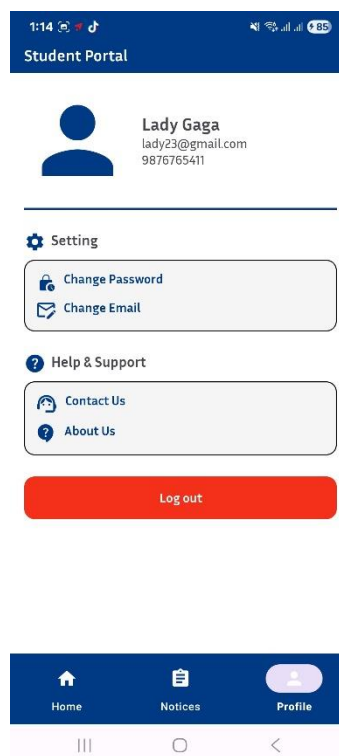


Figure: Annex 2.6 Student Profile



Figure: Annex 2.7 Teacher Home



Figure: Annex 2.8 Teacher Notice

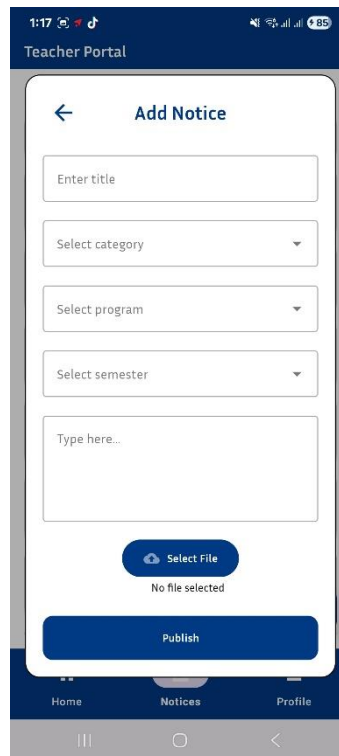


Figure: Annex 2.9 Teacher Add Notice

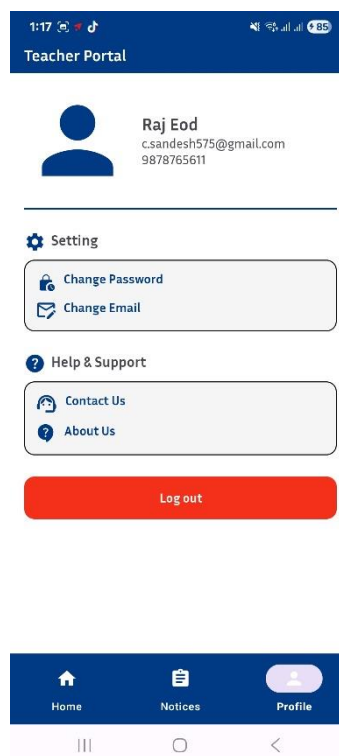


Figure: Annex 2.10 Teacher Profile

ANNEX III

Source Code:

```
@RestController
@RequestMapping("/api/admin/teacher")

public class TeacherController {

    private final TeacherService teacherService;

    public TeacherController(TeacherService teacherService) {
        this.teacherService = teacherService;
    }

    // REST API to create teacher
    @PostMapping
    public ResponseEntity<TeacherDto> createTeacher(@RequestBody TeacherDto
teacherDto) {
        TeacherDto created = teacherService.createTeacher(teacherDto);
        return new ResponseEntity<>(created, HttpStatus.CREATED);
    }

    // REST API to update teacher
    @PutMapping("/{id}")
    public ResponseEntity<TeacherDto> updateTeacher(@PathVariable Long id,
                                                    @RequestBody TeacherDto teacherDto) {
        TeacherDto updated = teacherService.updateTeacher(id, teacherDto);
        return ResponseEntity.ok(updated);
    }

    // REST API to get teacher by ID
    @GetMapping("/{id}")
    public ResponseEntity<TeacherDto> getTeacherById(@PathVariable Long id) {
        TeacherDto teacherDto = teacherService.getTeacherById(id);
        return ResponseEntity.ok(teacherDto);
    }

    @GetMapping("/all")
    public ResponseEntity<List<TeacherDto>> getAllTeachers() {
        return ResponseEntity.ok(teacherService.getAllTeachers());
    }

    // REST API to delete teacher
    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteTeacher(@PathVariable Long id) {
        teacherService.deleteTeacher(id);
        return ResponseEntity.ok("Teacher deleted successfully.");
    }

    // REST API to get teacher by userId
```

```

    @GetMapping("/by-user/{userId}")
    public ResponseEntity<TeacherDto> getTeacherByUserId(@PathVariable Long
userId) {
        TeacherDto teacherDto = teacherService.getTeacherDetailsByUserId(userId);
        return ResponseEntity.ok(teacherDto);
    }

```

```

    @GetMapping("/by-authority/{authorityId}")
    public ResponseEntity<List<TeacherDto>>
getTeachersByAuthority(@PathVariable Long authorityId) {
        List<TeacherDto> teachers =
teacherService.getTeachersByAuthorityId(authorityId);
        return ResponseEntity.ok(teachers);
    }
}

```

```

@RestController
@RequestMapping("/api/admin/student")
public class StudentController {

```

```

    private final StudentService studentService;

```

```

    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

```

```

    // REST API to create student

```

```

    @PostMapping
    public ResponseEntity<StudentDto> createStudent(@RequestBody StudentDto
studentDto) {
        StudentDto created = studentService.createStudent(studentDto);
        return new ResponseEntity<>(created, HttpStatus.CREATED);
    }

```

```

    // REST API to update student

```

```

    @PutMapping("/{id}")
    public ResponseEntity<StudentDto> updateStudent(@PathVariable Long id,
                                                    @RequestBody StudentDto studentDto) {
        StudentDto updated = studentService.updateStudent(id, studentDto);
        return ResponseEntity.ok(updated);
    }

```

```

    // REST API to get student by ID

```

```

    @GetMapping("/{id}")
    public ResponseEntity<StudentDto> getStudentById(@PathVariable Long id) {
        StudentDto studentDto = studentService.getStudentById(id);
        return ResponseEntity.ok(studentDto);
    }

```

```

    @GetMapping("/all")

```



```

    public ResponseEntity<List<StudentDto>> getAllTeachers() {
        return ResponseEntity.ok(studentService.getAllStudent());
    }

    // REST API to delete student
    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
        return ResponseEntity.ok("Student deleted successfully.");
    }
}

@RestController
@RequestMapping("/api/shared/notices")
public class NoticeController {

    private final NoticeService noticeService;
    private final ObjectMapper objectMapper;

    public NoticeController(NoticeService noticeService, ObjectMapper
objectMapper) {
        this.noticeService = noticeService;
        this.objectMapper = objectMapper;
    }

    // New endpoint to handle file uploads for all users.
    // The notice data is now a String, which we parse from the request part.
    // The file is an optional MultipartFile.
    @PostMapping("/all")
    public ResponseEntity<NoticeDto> createNotice(
        @RequestPart("notice") String noticeJson,
        @RequestPart(value = "file", required = false) MultipartFile file
    ) throws JsonProcessingException {
        NoticeDto noticeDto = objectMapper.readValue(noticeJson, NoticeDto.class);
        NoticeDto created = noticeService.publishNoticeToAll(noticeDto, file);
        return new ResponseEntity<>(created, HttpStatus.CREATED);
    }

    // New endpoint to publish to teachers with an optional file
    @PostMapping("/teacher")
    public ResponseEntity<NoticeDto> publishToTeachers(
        @RequestPart("request") String requestJson,
        @RequestPart(value = "file", required = false) MultipartFile file
    ) throws JsonProcessingException {
        PublishNoticeRequestDto requestDto = objectMapper.readValue(requestJson,
PublishNoticeRequestDto.class);
        NoticeDto created = noticeService.publishNoticeToTeachers(requestDto, file);
        return new ResponseEntity<>(created, HttpStatus.CREATED);
    }
}

```

```

// New endpoint to publish to students with an optional file
@PostMapping("/student")
public ResponseEntity<NoticeDto> publishToStudents(
    @RequestPart("request") String requestJson,
    @RequestPart(value = "file", required = false) MultipartFile file
) throws JsonProcessingException {
    PublishNoticeRequestDto requestDto = objectMapper.readValue(requestJson,
PublishNoticeRequestDto.class);
    NoticeDto created = noticeService.publishNoticeToStudents(requestDto, file);
    return new ResponseEntity<>(created, HttpStatus.CREATED);
}

// New endpoint to publish to authorities with an optional file
@PostMapping("/authority")
public ResponseEntity<NoticeDto> publishToAuthority(
    @RequestPart("request") String requestJson,
    @RequestPart(value = "file", required = false) MultipartFile file
) throws JsonProcessingException {
    PublishNoticeRequestDto requestDto = objectMapper.readValue(requestJson,
PublishNoticeRequestDto.class);
    NoticeDto created = noticeService.publishNoticeToAuthorities(requestDto, file);
    return new ResponseEntity<>(created, HttpStatus.CREATED);
}

// Existing GET endpoints remain the same
@GetMapping("/{id}")
public ResponseEntity<NoticeDto> getNoticeById(@PathVariable Long id) {
    return noticeService.getNoticeById(id)
        .map(ResponseEntity::ok)
        .orElse(ResponseEntity.notFound().build());
}

@GetMapping("/all")
public ResponseEntity<List<NoticeDto>> getAllNotices(
    @RequestParam(required = false) Long categoryId,
    @RequestParam(required = false) String searchTerm,
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "5") int size) {
    Pageable pageable = PageRequest.of(page, size,
Sort.by("datetime").descending());
    List<NoticeDto> notices = noticeService.getAllNotices(categoryId, searchTerm,
pageable);
    return ResponseEntity.ok(notices);
}
}

@RestController
@RequestMapping("/api/admin/department")
public class DepartmentController {

    private final DepartmentService departmentService;

```

```

public DepartmentController(DepartmentService departmentService) {
    this.departmentService = departmentService;
}

//REST API to add Department
@PostMapping
public ResponseEntity<DepartmentDto> addDepartment(@RequestBody
DepartmentDto departmentDto) {
    return new
ResponseEntity<>(departmentService.createDepartment(departmentDto),
HttpStatus.CREATED);
}

@PutMapping("/{id}")
public ResponseEntity<DepartmentDto> updateDepartment(@PathVariable Long
id,
                                                    @RequestBody DepartmentDto
updateDepartment) {

    DepartmentDto departmentDto = departmentService.updateDepartment(id,
updateDepartment);
    return ResponseEntity.ok(departmentDto);
}

// REST API to get department
@GetMapping("/{id}/{id}")
public ResponseEntity<DepartmentDto> getDepartmentById(@PathVariable Long
id) {
    DepartmentDto departmentDto = departmentService
        .getDepartmentById(id);
    return ResponseEntity.ok(departmentDto);
}

@GetMapping("/name/{name}")
public ResponseEntity<DepartmentDto> getDepartmentByName(@PathVariable
String name){
    DepartmentDto departmentDto = departmentService
        .getDepartmentByName(name);
    return ResponseEntity.ok(departmentDto);
}

@GetMapping("/all")
public ResponseEntity<List<DepartmentDto>> getAllDepartments() {
    return ResponseEntity.ok(departmentService.getAllDepartment());
}

// REST API to delete department
@DeleteMapping("/{id}")
public ResponseEntity<String> deleteDepartment(@PathVariable Long id) {
    departmentService.deleteDepartment(id);
}

```

```

        return ResponseEntity.ok("Faculty deleted successfully.");
    }
}
@RestController
@RequestMapping("/api/admin/noticeCategory")
public class NoticeCategoryController {

    public final NoticeCategoryService noticeCategoryService;

    public NoticeCategoryController(NoticeCategoryService noticeCategoryService) {
        this.noticeCategoryService = noticeCategoryService;
    }

    //REST API to add notice category
    @PostMapping
    public ResponseEntity<NoticeCategoryDto> addNoticeCategory(@RequestBody
    NoticeCategoryDto noticeCategoryDto) {

        return new
        ResponseEntity<>(noticeCategoryService.createNoticeCategory(noticeCategoryDto),
        HttpStatus.CREATED);
    }

    //REST API to update notice category
    @PutMapping("/{id}")
    public ResponseEntity<NoticeCategoryDto>
    updateNoticeCategory(@PathVariable Long id,
                        @RequestBody NoticeCategoryDto
    updateNoticeCategory) {

        NoticeCategoryDto noticeCategoryDto =
        noticeCategoryService.updateNoticeCategory(id, updateNoticeCategory);
        return ResponseEntity.ok(noticeCategoryDto);
    }

    //REST API to retrieve notice category
    @GetMapping("/{id}")
    public ResponseEntity<NoticeCategoryDto>
    getNoticeCategoryById(@PathVariable Long id){

        NoticeCategoryDto noticeCategoryDto = noticeCategoryService
        .getNoticeCategoryById(id);
        return ResponseEntity.ok(noticeCategoryDto);
    }

    @GetMapping("/name/{name}")
    public ResponseEntity<NoticeCategoryDto>
    getNoticeCategoryByName(@PathVariable String name) {
        NoticeCategoryDto noticeCategoryDto = noticeCategoryService
        .getNoticeCategoryByName(name);
    }
}

```

```

        return ResponseEntity.ok(noticeCategoryDto);
    }

    @GetMapping("/all")
    public ResponseEntity<List<NoticeCategoryDto>> getAllNoticeCategory() {
        return ResponseEntity.ok(noticeCategoryService.getAllNoticeCategory());
    }

    //REST API to delete notice category
    @DeleteMapping("/{id}")
    public ResponseEntity<String> deletedNoticeCategory(@PathVariable Long id) {
        noticeCategoryService.deleteNoticeCategory(id);
        return ResponseEntity.ok("Notice Category deleted successfully.");
    }
}

```