CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-Section SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: Mini App – User Registration & Authentication

Prepared By: Vinci F. Villanueva

Date of Submission: February 2, 2026

Version: 1

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this Functional Requirements Specification (FRS) document is to design the functionality of the Mini App - User Registration & Authentication system. This document serves as a guide for the development of said system.

1.2. Scope

The system will include:

- **User Registration**: Allows the user to create a unique account.
- **User Sign-in**: Using a username/email and password, the user can access their account.
- **Session Management**: Maintains a secure state across protected routes (using JWT).
- **Protection within designated pages (Dashboard/User Profile)**: Only a logged-in user can view said pages.
- **User Logout**: Mechanism that invalidates the active session/token.

1.3. Definitions, Acronyms, and Abbreviations

Term/Acronym	Definition
FRS	Functional Requirements Specification: A document detailing the functions a software system must perform.
User	Any person interacting with the system, either as a Guest or an Authenticated User.
Guest User	A user who has not logged in and has limited access to the application.
Authenticated User	A user who has successfully logged in and can access protected features.
JWT	JSON Web Token: A standard for securely transmitting information between parties as a JSON object. Used for session management.
API	Application Programming Interface: The set of routines, protocols, and tools for building software applications. The Spring Boot backend will serve as the API.
UI	User Interface: The React-based frontend application.

2. Overall Description

2.1. System Perspective

Describe how the system fits into a larger context or environment.This authentication system is designed as a standalone Mini App serving as a foundational layer for a future, larger application. It operates as a three-tier architecture:

1. **Presentation Tier (React UI)**: The client-side interface where users interact for registration and login.
2. **Application Tier (Spring Boot API)**: The backend server that handles business logic, including password hashing, token generation (JWT), and communication with the database.
3. **Data Tier (Database)**: The persistent storage for user records, including usernames, emails, and hashed passwords.

2.2. User Classes and Characteristics

Guest User and **Authentication**

2.3. Operating Environment

The system requires the following technologies for development and operation:

1. **Frontend**: React (with Node.js for development environment).
2. **Backend (API)**: Spring Boot (Java) with an embedded application server (e.g., Tomcat/Jetty).
3. **Database**: A relational database (e.g., MySQL, PostgreSQL, or H2 for development) accessed via Spring Data JPA.
4. **Authentication**: Spring Security for core authentication mechanisms, utilizing JWT for stateless sessions.
5. **Development Tools**: IDE (e.g., IntelliJ, VS Code), Maven/Gradle for dependency management.

2.4. Assumptions and Dependencies

Type	Item	Description
Assumption	User Identity	It is assumed that the username and email fields will be sufficient for uniquely identifying a user.

Assumption	Password Security	It is assumed that the system's security is directly dependent on the correct implementation of password hashing (e.g., using bcrypt).
Dependency	Database Connectivity	The system is dependent on a fully
		functional and available database for all registration and login operations.
Dependency	Frontend-Backend Contract	The development of the React UI is dependent on the final definition and implementation of the REST API endpoints in Spring Boot.

3. System Features and Functional Requirements

Describe each major feature of the system and its functional requirements.

3.1. Feature 1: User Registration

Description: This feature allows a new user to create an account in the system by providing a unique username, a valid email address, and a password.

Functional Requirements:

- The system MUST accept a username, email, and password from the user.
- The system MUST validate that the submitted username is not already in use. - The system MUST validate that the submitted email is not already in use and is in a valid format.
- The system MUST hash and salt the password before storing it in the database. - The system MUST store the new user's credentials and metadata (e.g., creation date) in the Users table.

3.2. Feature 2: User Authentication (Login/Logout)

Description: This feature provides secure access control, allowing an authenticated user to gain access to protected pages and securely exit their session.

Functional Requirements:

- The system MUST allow an existing user to submit their username/email and password to log in.
- The system MUST compare the submitted password against the stored password hash using the configured encoding algorithm.

- Upon successful authentication, the system MUST generate a unique JWT and return it to the client.
- The system MUST protect the User Profile/Dashboard endpoint, requiring a valid JWT in the request header for access.
- The system MUST invalidate the user's session/token when the user triggers the logout function, denying access to protected pages.

4. Non-Functional Requirements

Specify system quality attributes such as performance, security, usability, reliability, etc.

4.1. Security

- **Password Storage:** All user passwords must be hashed using **BCrypt** before being stored in the database. Plain text passwords must never be saved.
- **Authentication Token:** The system must use **JSON Web Tokens (JWT)** for stateless session management. Tokens should expire after a set duration (e.g., 24 hours).
- **Data Protection:** Sensitive data (such as passwords and tokens) must be transmitted over secure channels (HTTPS) to prevent interception.
- **Access Control:** Protected routes (e.g., Dashboard) must strictly deny access to requests without a valid JWT in the HTTP header.

4.2. Performance

- **Response Time:** Login and Registration requests should be processed and respond within **2 seconds** under normal load conditions.
- **Database Efficiency:** Database queries (finding a user by email) should be optimized using indexing on the email column to ensure fast lookups.
- **Concurrency:** The system should support multiple users logging in simultaneously without performance degradation.

4. 3. Usability

- **User Interface:** The React frontend must be responsive, adapting layout for both desktop and mobile web browsers.
- **Error Feedback:** The system must provide clear, human-readable error messages for failed actions (e.g., "Email already in use" or "Invalid credentials") rather than raw system codes.
- **Form Validation:** Client-side validation (React) should prevent the submission of empty fields or invalid email formats before the request reaches the server.

4.4. Reliability & Availability

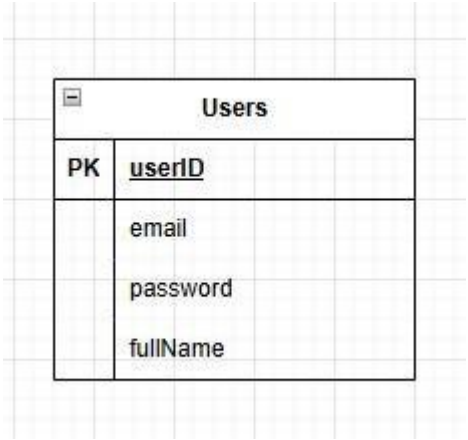
- **Availability:** The system should be available 99.9% of the time during business hours.

- **Data Integrity:** The system must ensure that no duplicate accounts can be created with the same email address (enforced via database unique constraints).

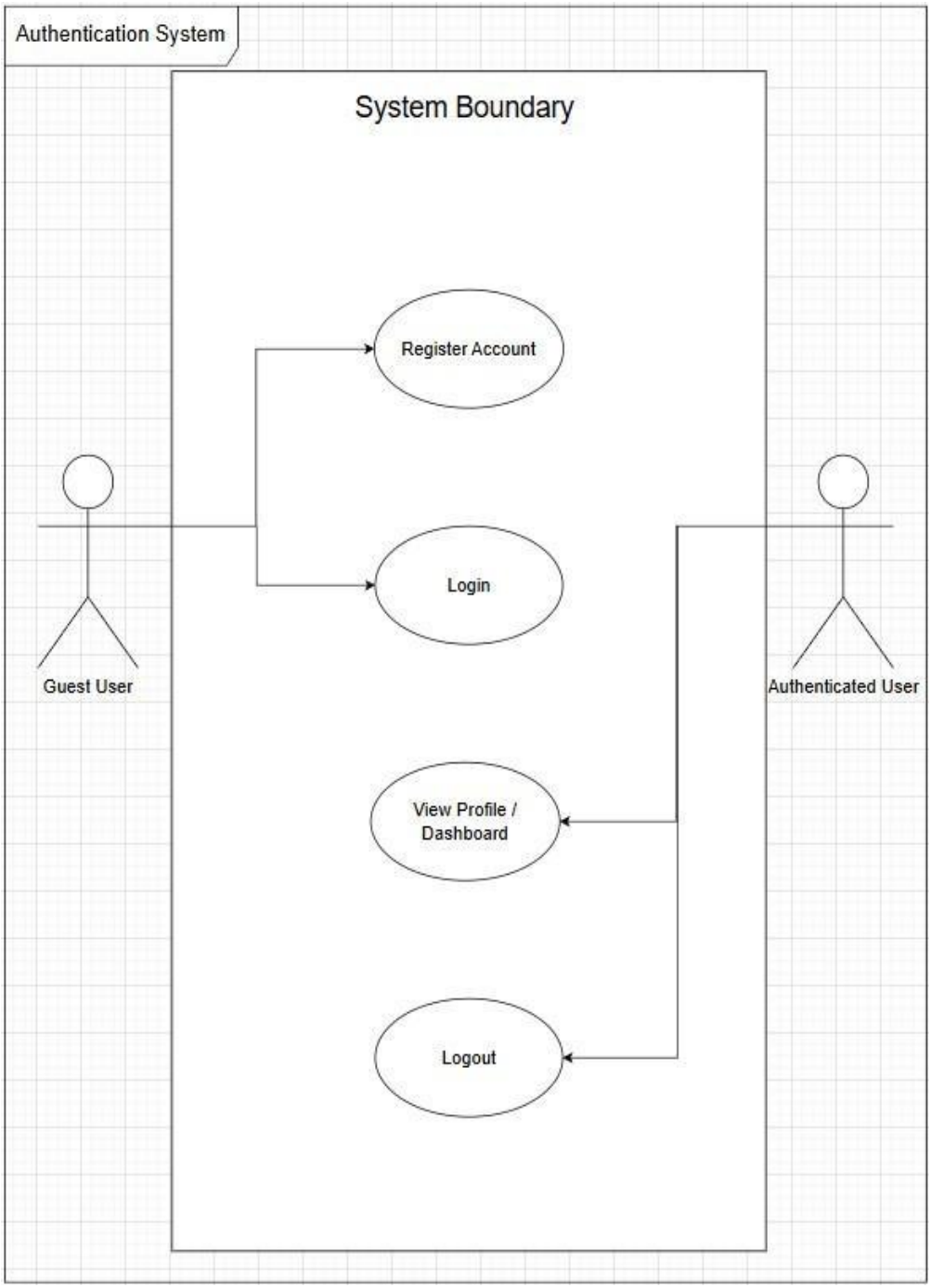
5. System Models (Diagrams)

Insert the necessary diagrams for the system:

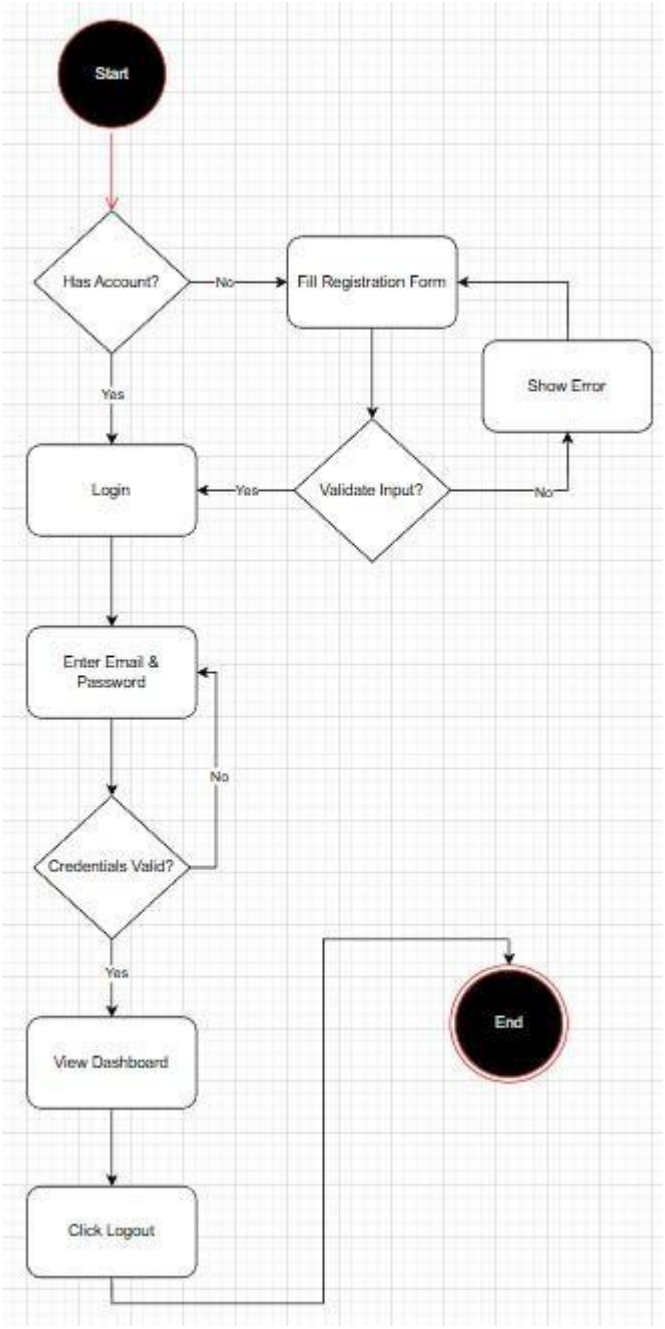
5.1. ERD



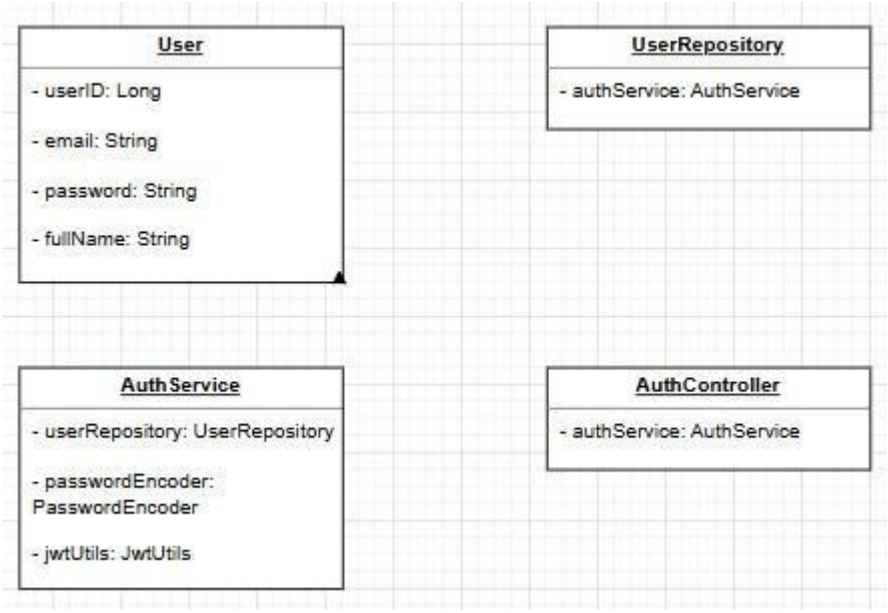
5.2. Use Case Diagram



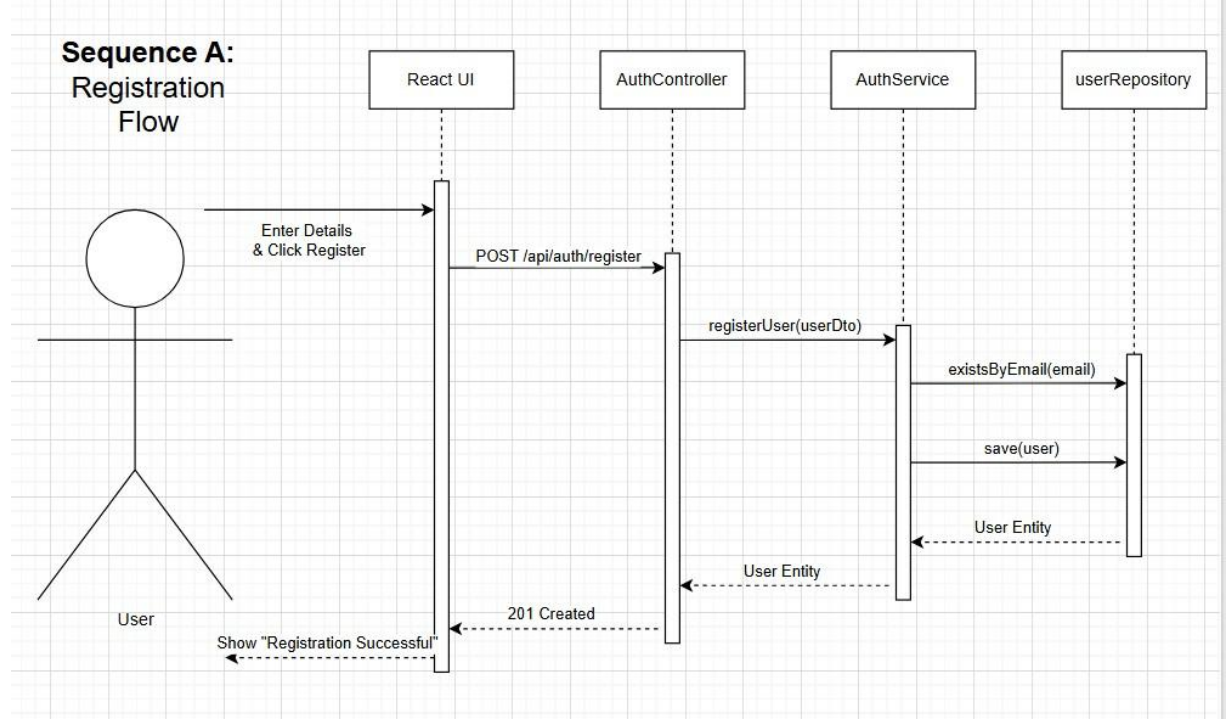
5.3. Activity Diagram

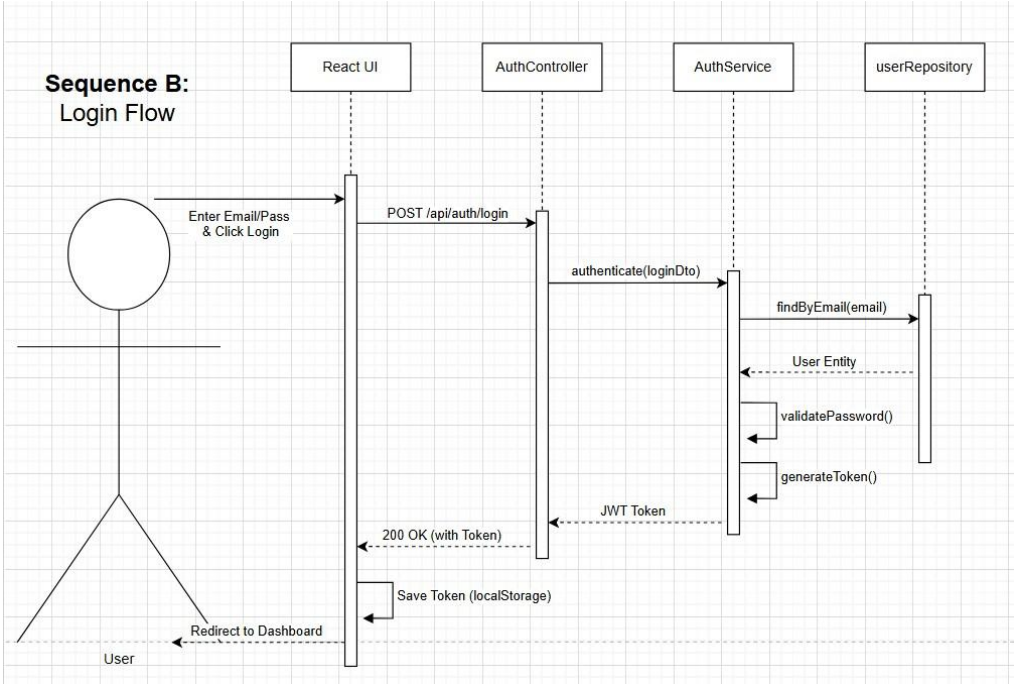


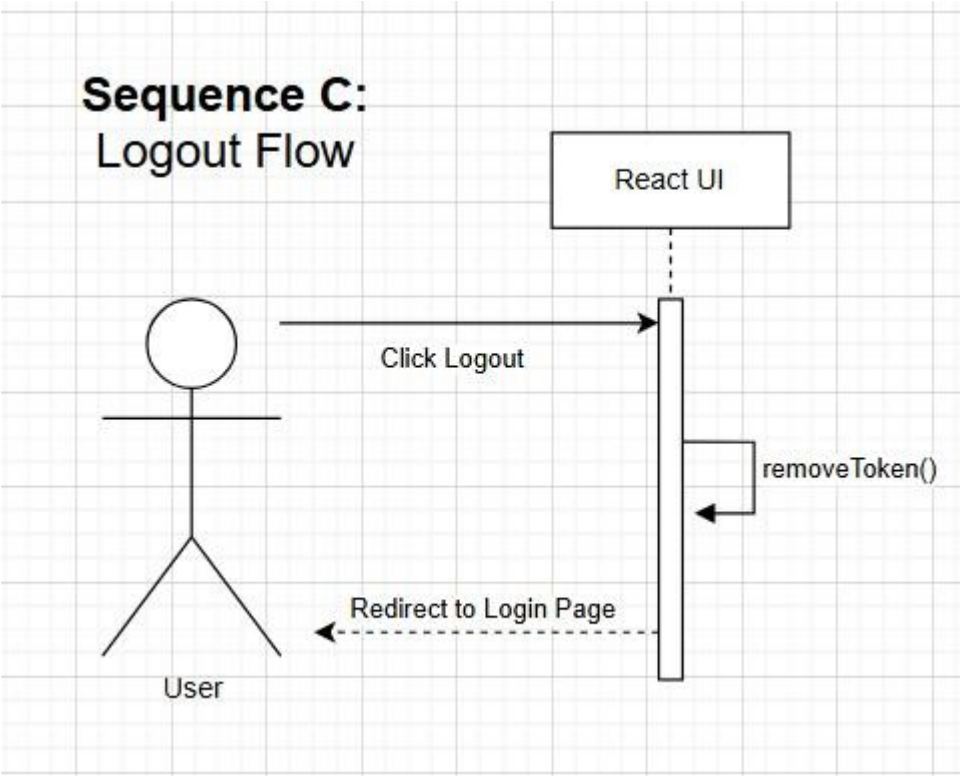
5.4. Class Diagram



5.5. Sequence Diagram






Sequence C:
Logout Flow
React UI
Click Logout
removeToken()
Redirect to Login Page
User

6. Appendices

Include any additional information, references, or support materials.

Appendix A: Database Schema Script (MySQL)
The following SQL script defines the structure of the users table required for the authentication module.

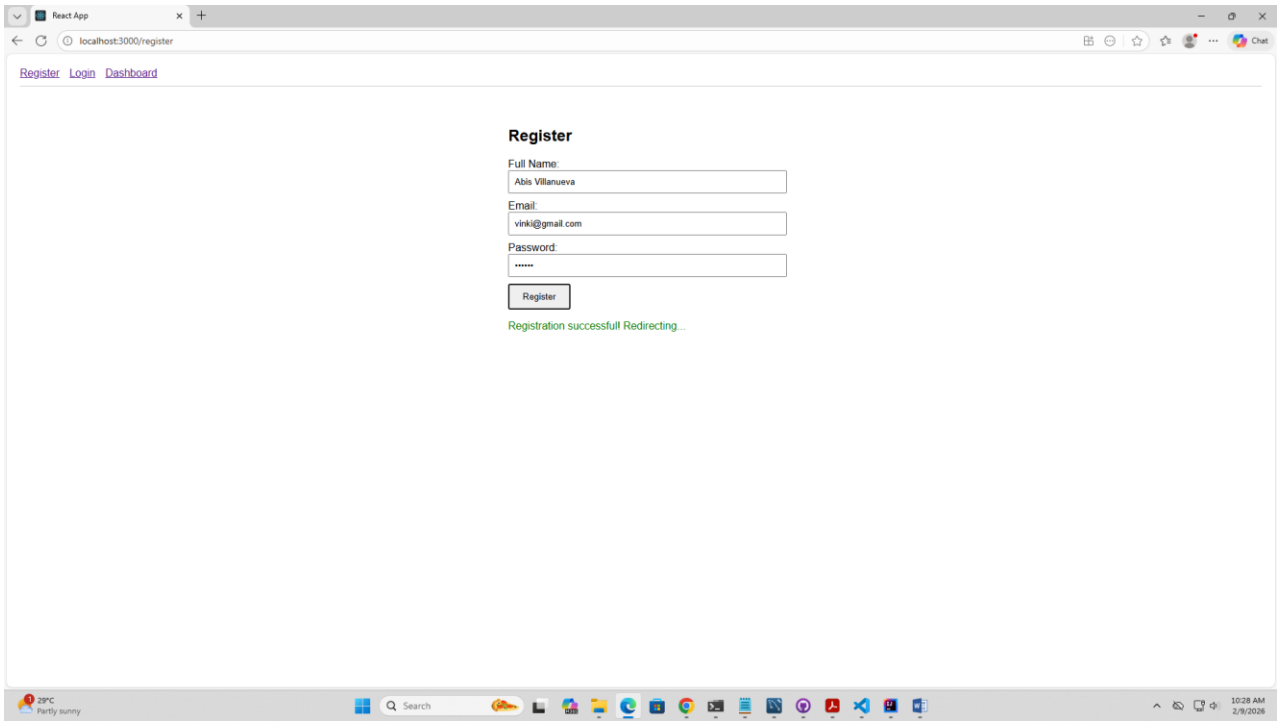
Appendix B: API Endpoints Reference
The following REST API endpoints are exposed by the Spring Boot backend (AuthController) to support the frontend actions.

Appendix C: Technology Stack
The following technologies and libraries are utilized in the implementation of this system:

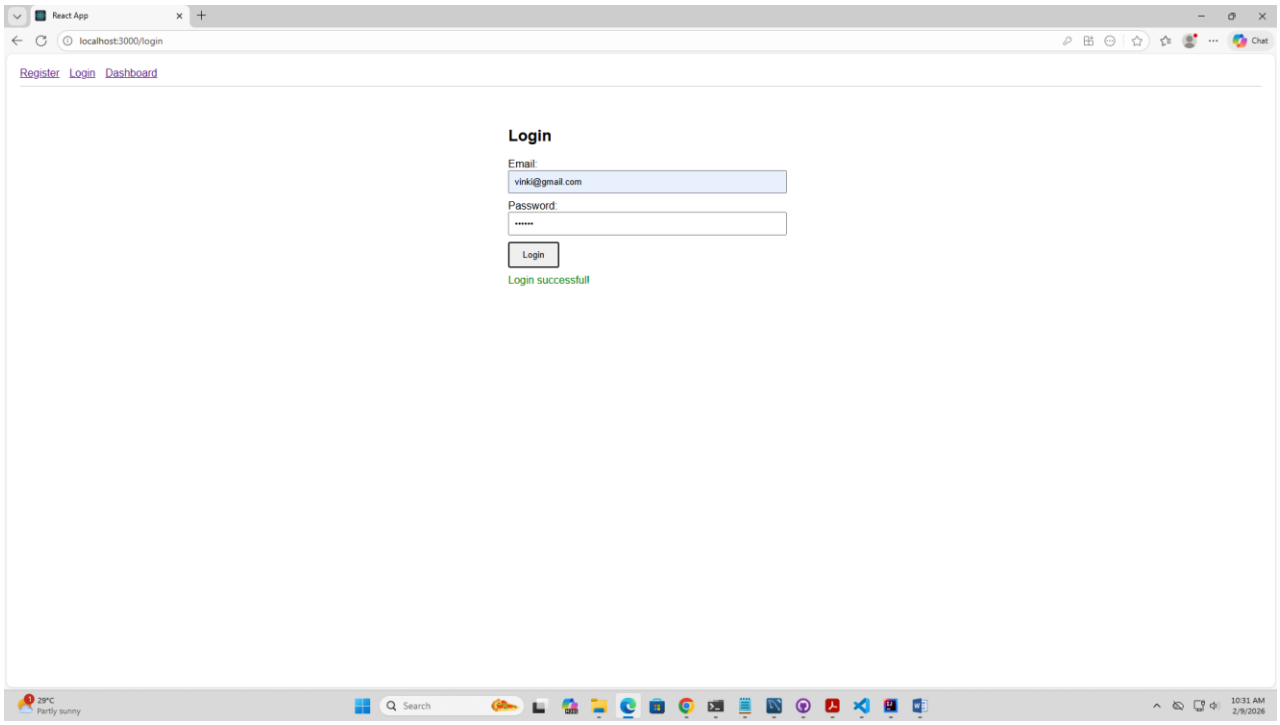
- **Frontend:** React.js, Axios (for API requests), React Router (for navigation).
- **Backend:** Java Spring Boot, Spring Security (for authentication logic).
- **Database:** MySQL (relational data storage).
- **Security:** BCrypt (password hashing), JSON Web Tokens (JWT) (stateless session management).
- **Tools:** Draw.io (Diagramming), Postman (API Testing), Maven (Dependency Management).

Appendix D: Sceenshots:

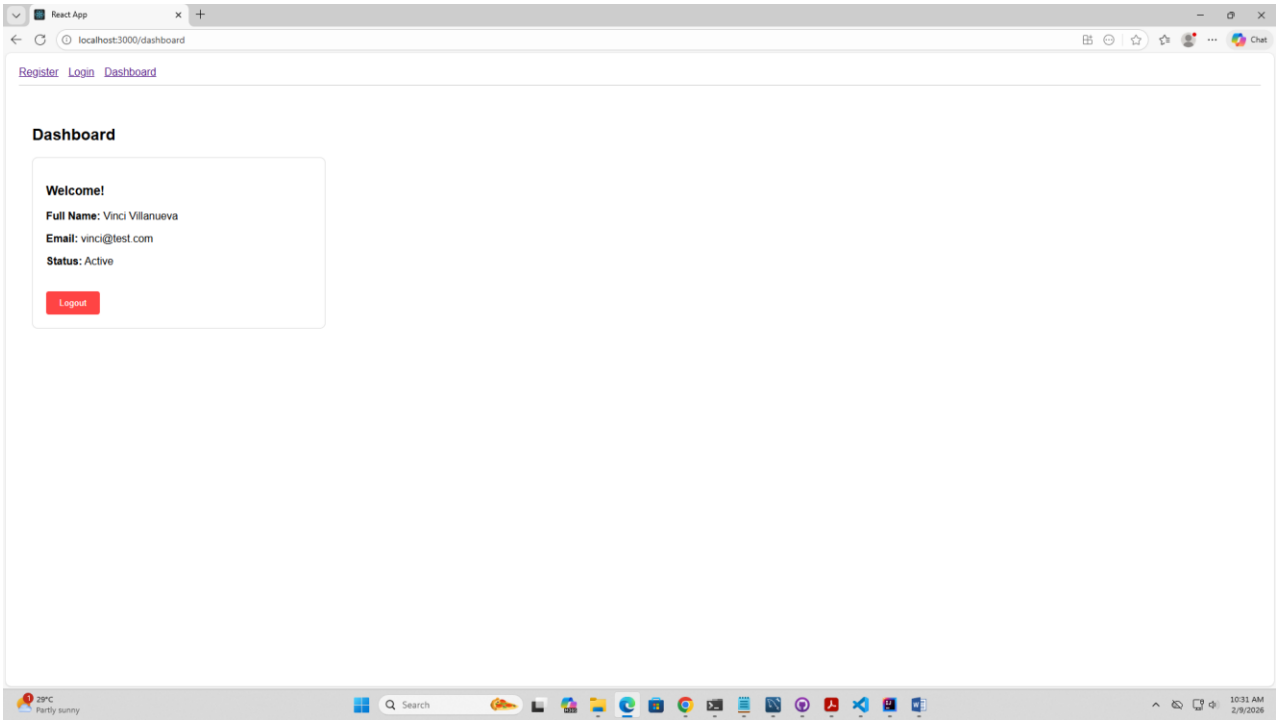
REGISTER


React App
localhost:3000/register
Register Login Dashboard
Register
Full Name:
Abis Villanueva
Email:
vinki@gmail.com
Password:
Register
Registration successfull Redirecting...

LOGIN


React App
localhost:3000/login
Register Login Dashboard
Login
Email:
vinki@gmail.com
Password:
Login
Login successfull

DASHBOARD



DATABASE