A logo for a university

Description automatically generated A green and black logo

Description automatically generated

**6CS007 – Project and Professionalism**

**Milestone 3 – Artefact Design**

University Id: 2330520

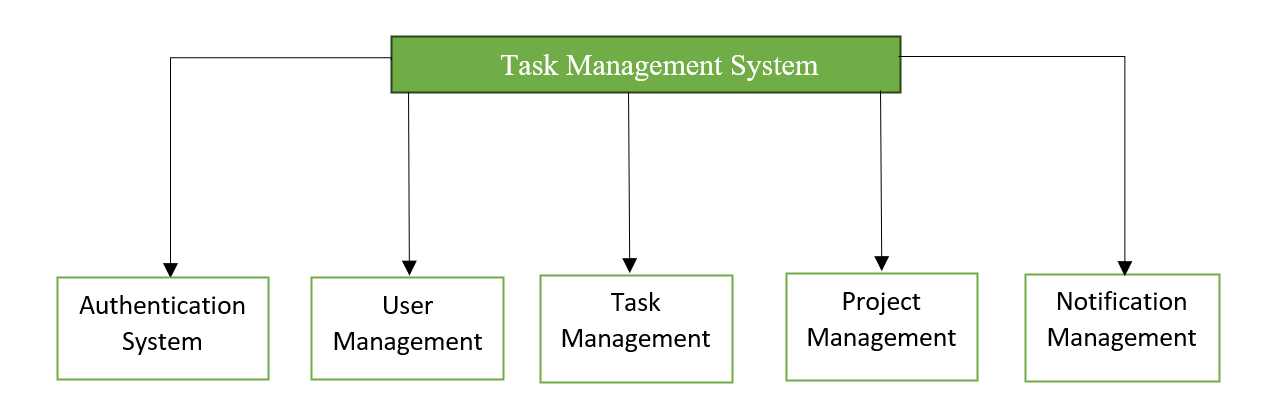
Group: L6CG22

Supervisor: Dipti Gyawali

Submitted by: Shrota Ghimire

Submitted on: 2nd February, 2025

# Functional Decomposition Diagram (FDD)



# Subsystem 1 – Authentication System

## 2.1 Introduction

This document outlines the artifact design for the Authentication Subsystem as part of the final year project. User authentication gets secure management through this subsystem, so it protects authorized access to the Task Management System.

This subsystem provides users with secure identification procedures which offer both flexibility and defensive integrity. User registration functionality and login authentication services and role-based access control capabilities will be managed by this subsystem to realize data security and system integrity across all platform elements.

## 2.2 System Architecture and Design

## 2.2.1 Activity Diagram

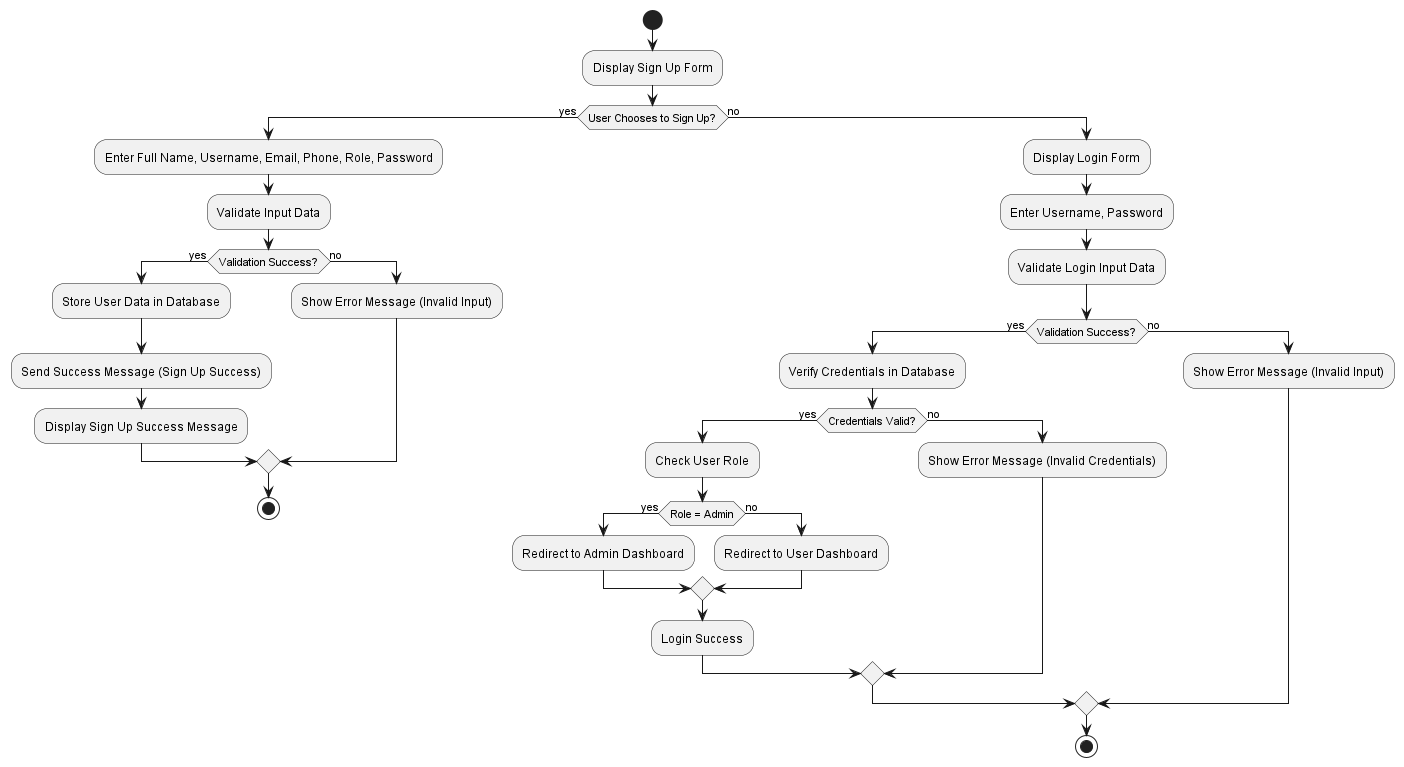


Figure 1: Activity Diagram

## 2.2.2 Use case Diagram

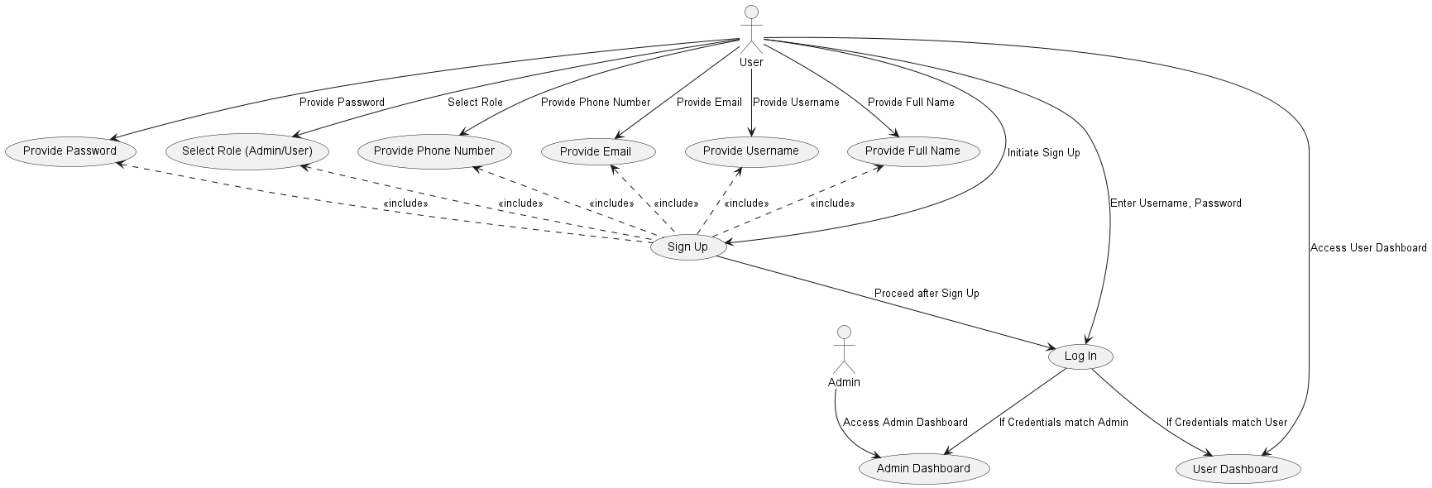


Figure 2: Use Case Diagram

## 2.2.2 Entity Relationship Diagram (ERD)

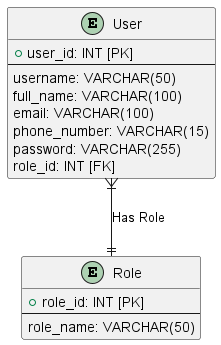


Figure 3: Entity-Relationship Diagram

## 2.2.3 Class Diagram

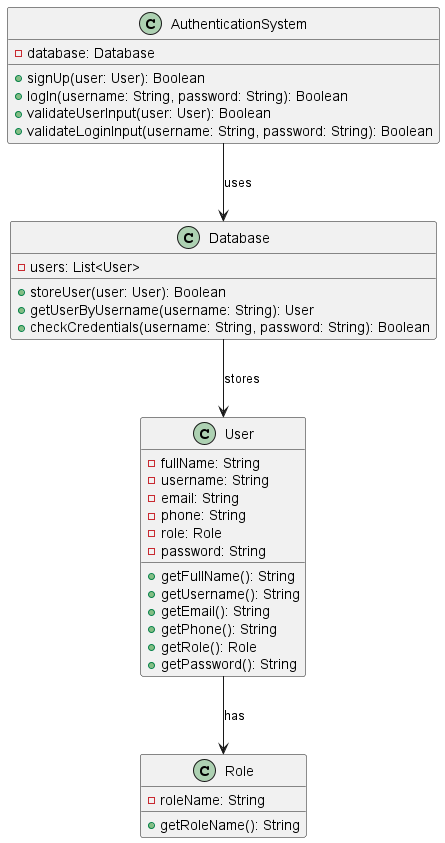


Figure 4: Class Diagram

## 2.2.4 Sequence Diagram

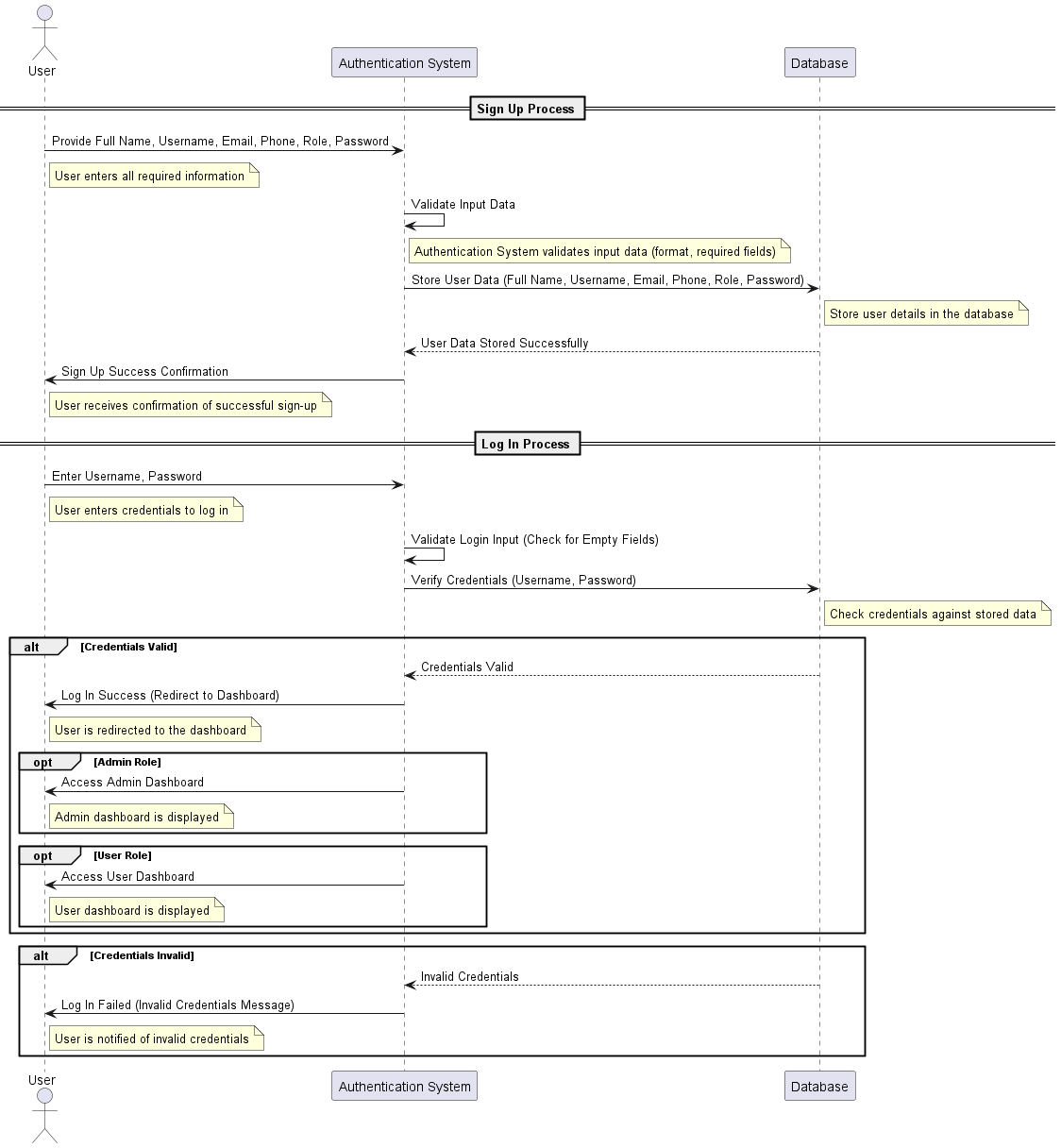


Figure 5: Sequence Diagram

## 2.2.5 Wireframes

* **Sign Up**

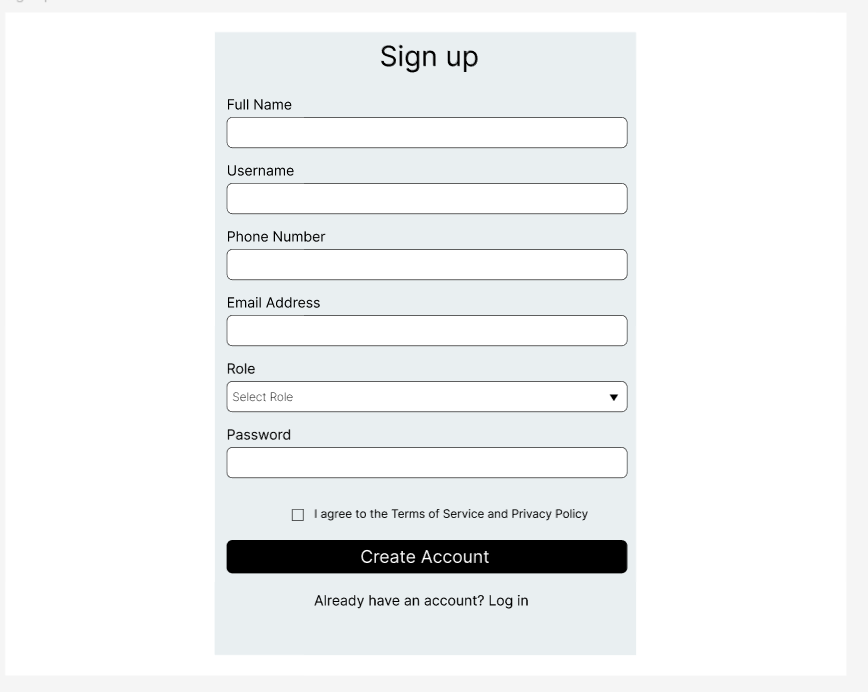
****

Figure 6: Signup

* **Log In**

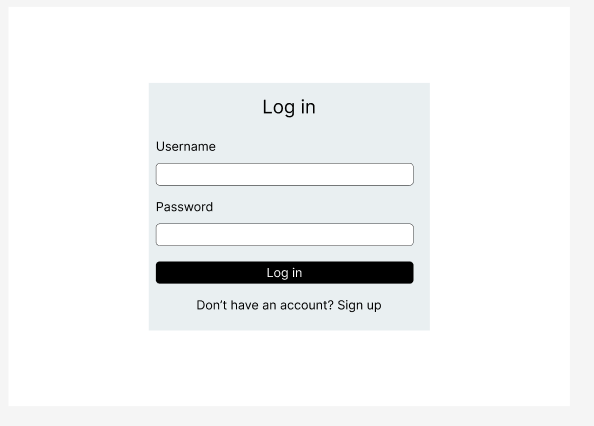
****

Figure 7: Log In

## 2.3 Critical Features and Implementation

## 2.3.1 Key Functionalities

## Feature 1: User Sign-up

Through the Sign-Up interface users can establish new accounts through submitting their account information. Users who need system access must register while the application maintains the secure storage of their information.

**Implementation:**

* Users need to complete a registration form containing full name, username, email, phone number, password and their designated role. Users complete a form which contains mandatory input validations to verify both mandatory field entry and proper data formatting.
* The Authentication System class processes user input before validating it for accuracy then stores user information in the database.
* Security tests verify that passwords fulfill requirements regarding minimum length and complexity and confirm uniqueness of both usernames and emails addresses.
* The system must perform an encryption process when storing passwords into its database to enhance safety.
* The implementation of bcrypt hashing serves as the standard approach to secure store passwords.

## Feature 2: User Login

Through Login functionality users can access the system by providing their username along with their password. The system authenticates user credentials by checking them against its database then enables access according to user roles that include Admin or User.

**Implementation:**

* Users need to input their username together with password on a login form.
* Both required fields must have content before continuing validation tests.
* The Authentication System executes a search against the Database for users matching the given username.
* A match check occurs for the user's stored hashed password versus their supplied password which they provide after our system hashed it. Successful authorization results in system access.
* The system reviews the assigned role of the active user. The system sends regular Users toward the User Dashboard however it directs Admins straight to the Admin Dashboard.
* When credentials fail authentication, the system shows an error message which prompts users to try again.
* A secure password verification algorithm (bcrypt) helps compare hashes stored in the system with entered passwords.

## 2.3.2 API and Database Implementation

## 2.3.2.1 API Endpoints

Through its RESTful API system, the authentication structure handles both new user registration alongside user login operations as well as access restriction functions. Below is a structured list of the key API endpoints:

* **User Signup**
* Endpoint: POST/api/signup
* Description: Allow new users to register an account.
* Request Body (JSON):



Figure 8: JSON code

* Response:

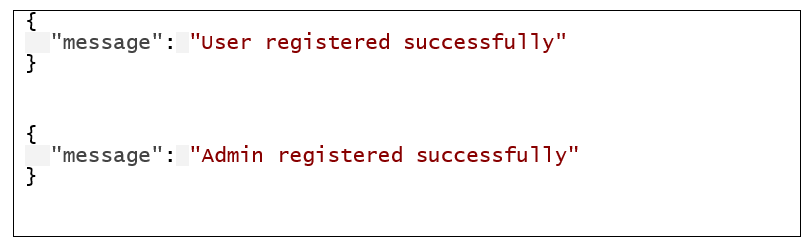


Figure 9: Success (201 created)

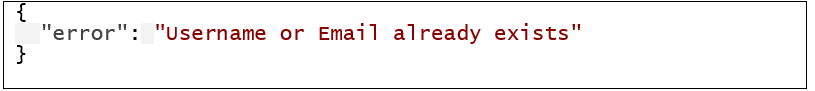


Figure 10: Error Message

* **User Login**
* Endpoint: POST /api/login
* Description: Authenticates a user or admin based on username and password.
* Request Body (JSON):

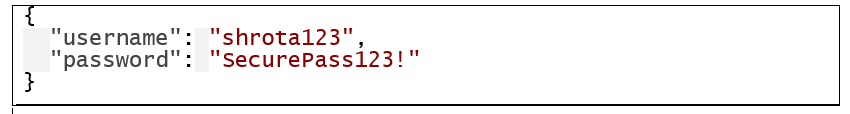


Figure 11: JSON code

* Response:

A white background with red text

Description automatically generated

Figure 12: Success (200 OK)

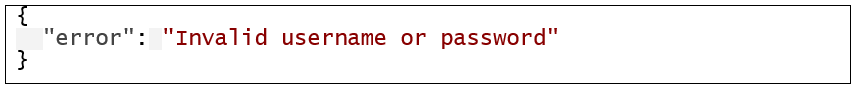


Figure 13: Error Message

## 2.3.2.2 Database Schema

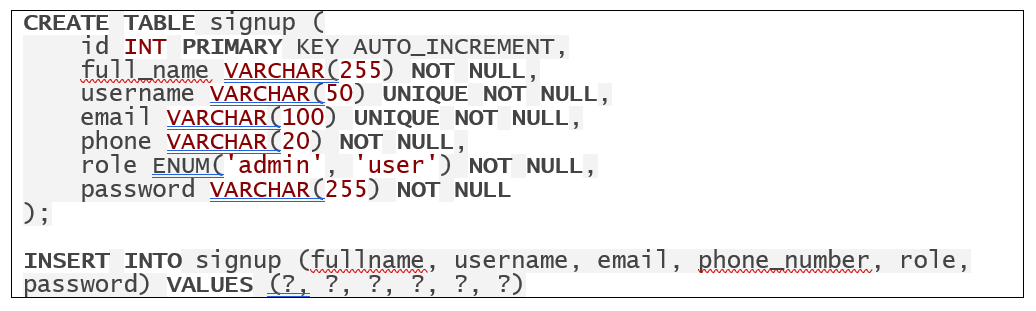


Figure 14: Signup Database Schema

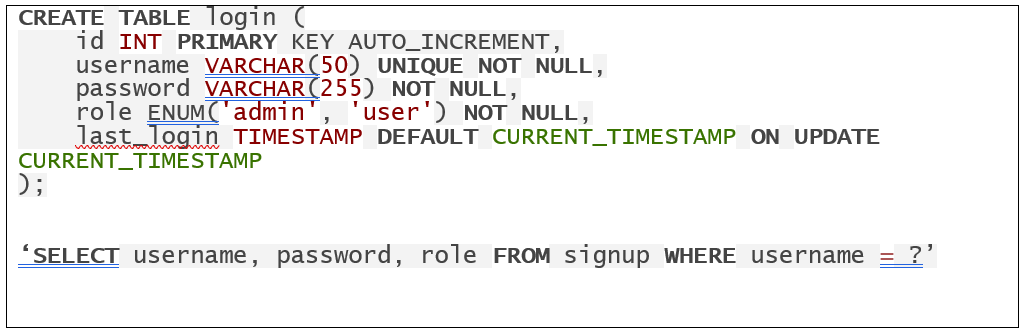


Figure 15: Login Database Schema

## 2.4 Development Process

## 2.4.1 Sprint Planning

This subsystem ‘Authentication System’ is my sprint 1. The duration of this sprint was 1 week. During this time, I developed an authentication system with frontend and backend integration. I have listed the tasks I did, and they’re on the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Days** | **Tasks** | **Goals** | **Deliverables** |
| Day 1 | Frontend setup | Initialize frontend project structure | HTML, CSS, JavaScript setup |
| Day 2 | Development of Signup and Login Page | Build UI of Signup and Login Page | Signup and Login UI |
| Day 3 | Error handling | Connect Frontend forms with APIs | Error handling and Frontend-Backend Communication |
| Day 4 | Backend setup | Initialize Backend project structure | Node.js setup |
| Day 5 | Implement Signup and Login APIs | Develop backend logic for signup and login | Signup and Login API |
| Day 6 | Role-Based Access | Implement role-based redirection for Admin and user | Role-based Redirection |
| Day 7 | Error handling | Fix any issues in the backend | Error-free backend |

## 

## 2.4.2 Execution Details

I executed this work through Agile methodology to create and optimize separate elements in the system. The development approach implemented systematic workflow beginning with frontend layout construction which then integrated backend application programming interfaces.

* **Frontend Development**

The central priority at first was to assemble an engaged user interface that interacted with users. The frontend used HTML and CSS along with JS to become a fast and modular development environment. Frontend components received integration treatment to work with form validation alongside error handling functionality as well as API integration. The developers moved on to creating application responsiveness while sustaining a natural user experience after completing the UI stage.

* **Backend Development**

The backend development started with Node.js after frontend development finished. A set of APIs was designed to manage user authentication at different levels through login and signup and role-based authentication services. The database architecture received architectural patterns to store credentials alongside valuable operational information. User data management demands dependable performance which led to MySQL selection.

* **Integration**

Integration of frontend design with backend APIs followed development completion of frontend and backend components independently. The authentication system performed requests at the frontend which were transmitted to the backend functions.

**Tools and Technologies used:**

|  |  |
| --- | --- |
| **Category** | **Tools/Technology used** |
| Frontend | HTML, CSS, JavaScript |
| Backend | Node.js |
| Database | MySQL |
| Version Control | GitHub |

## 2.5 Testing Approach

Testing stands as an essential developmental component for verifying that every system component performs to specifications. Research-based testing extends across unit testing and integration testing and user testing for complete system examination.

## 2.5.1 Unit Testing

Unit testing analyzes individual system components while keeping them independent of other parts. System components must operate as expected to achieve this objective. Below are the strategies for testing function:

* **Testing Function**

Dedicated tests need to exist for every backend function such as password hash operations and stored data queries. Next, we apply tests which confirm that password hashing correctly matches stored passwords during the comparison process.

* **Test Coverage**

The essential functionality of signup and login needs complete testing for valid data together with invalid inputs and edge conditions. Password hashing along with validation checks and error handling and their respective test cases need to be tested.

## 2.5.2 Integration Testing

During integration testing different system modules and components must communicate well to perform as expected. Proper data exchange verification between system components and their collective functionality depends on this testing. Below are the strategies for integration testing:

* **Testing API Endpoints**

Putting different API routes along with the login and signup endpoints through a full integration test. A user should be able to successfully create an account before successfully authenticating into the system with their original credentials.

* **Database Integration**

Verify that database operations process data correctly for both insertion and retrieval. Check that user data succeeds in storing properly in the signup table immediately after successful account creation and examine login credentials against the database content at login time.

## 2.5.3 User Testing

User testing realizes objective assessment of how the system works to satisfy end-user needs. User testing demonstrates that the system should be easy to use while fulfilling all user requirements.

* **User Interface testing**

Testing all the frontend components alongside forms (signup and login) buttons and navigation to verify functionality while maintaining visibility.

* **Usability Testing**

Obtaining user feedback by observing their experiences working with the interface alongside system functionality. Annotating your user’s experience by observing their performance while they execute the steps of creating an account, entering login credentials and reaching their dashboard. Recording any issues they face.

* **Performance Testing**

Testing the system under maximum operational stress to assess performance specifically on user authentication as well as API request and response behaviors.

* **End-To-End Flow**

The team conducts end-to-end testing for real-world operations including user signup procedures and system authentication with account logins to access platform resources. The system must behave exactly as expected while users should be able to navigate through it without obstacles.

## 2.6 Conclusion

The Authentication System provides three main features which secure user registration activities while enabling safe access login and controlling access based on user roles throughout the Task Management System. An Agile development process allowed me to create this system in multiple phases that started with frontend development and proceeded to backend development before finishing with integration work. Proper validation and bcrypt hashing served as mechanisms to enhance system security. System reliability together with performance were confirmed through the combination of extensive unit testing and integration testing and user testing. An effective authentication system produces a secure authentication framework that ensures user access administration with system protection maintained.

# Subsystem 2 – User Management

## 3.1 Introduction

This document outlines the artifact design for the subsystem User Management as part of the final year project Task Management System. Through the User Management subsystem of the Task Management System, administrators achieve control over user accessibility within different projects. This security protocol grants enrollment safety by using a two-step verification system for users and provides fast user disconnection options when needed. The subsystem manages user access through correct role definitions and controls allowing teams to work together efficiently across projects.

It operates to provide efficient user access control through functional user administration for project-based access. The system enables administrators to produce special codes for user admission while enabling them to end user access privileges when required. A secure system which protects project integrity and role responsibilities exists to stop unauthorized access.

## 3.2 System Architecture and Design

## 3.2.1 Activity Diagram

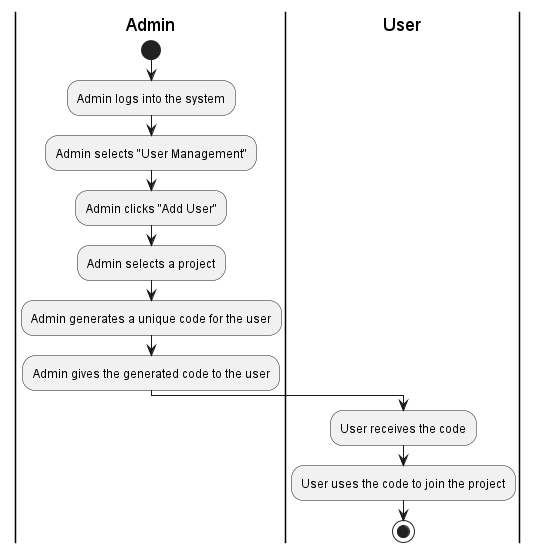


Figure 16: Activity Diagram - 1

## 3.2.2 Use Case Diagram

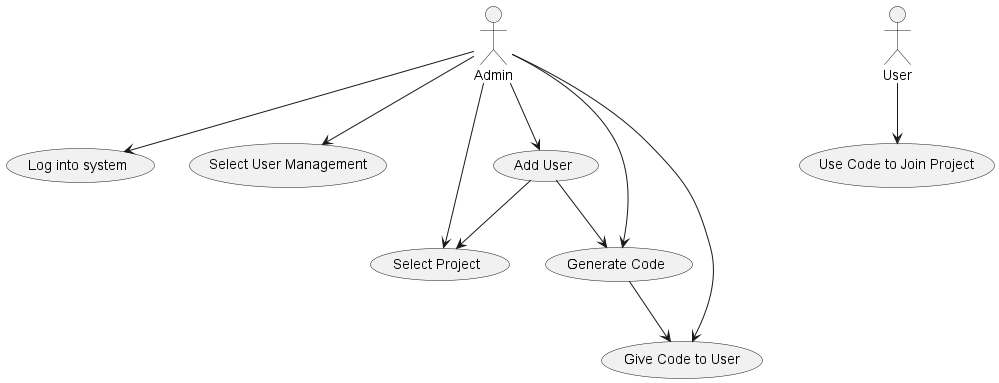


Figure 17: Use Case Diagram – 2

## 3.2.3 Entity Relationship Diagram (ERD)

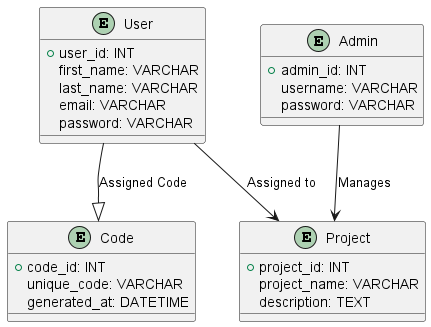


Figure 18: Entity Relationship Diagram

## 3.2.4 Class Diagram

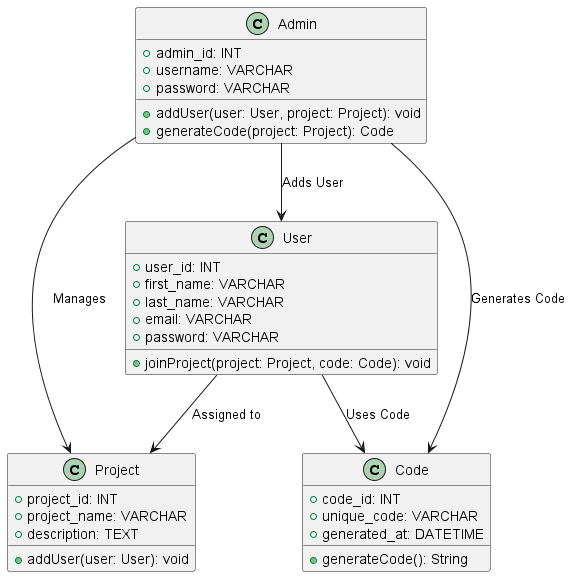


Figure 19: Class Diagram

## 3.2.5 Sequence Diagram

A diagram of a program

Description automatically generated

Figure 20: Sequence Diagram

## 3.2.6 Wireframes

* **Adding and Removing User**

**A screenshot of a computer

Description automatically generated**

Figure 21: Adding and Removing User

A screenshot of a computer

Description automatically generated

Figure 22: Add User

## 3.3 Critical Features and Implementation

## 3.3.1 Key Functionalities

## Feature 1: Add User

Through the system administrators obtain the capability to invite users to project through code generation. Staff can choose a project to code generation while the resultant code becomes available to platform users. The system enables users to join projects through a specified code which provides both controlled and secure entry.

**Implementation:**

* Use of the "Add User" action by an admin leads to the selection of a preferred project.
* The system creates a distinct alphanumeric identification code which belongs to the project.
* The user performs entry of the code to prove authorization and becomes part of the project database.
* The database system adds the user to the project user list as part of its automatic database process.

## Feature 2: Remove User

Active administrators possess the ability to eliminate users from projects which leads to terminated access permissions. Project data protection happens through this system since it eliminates unauthorized users' access.

**Implementation:**

* The administrator selects a project before viewing a list that displays allocated users.
* The system includes a removal option for all users within its interface.
* The database gets modified with access revocation after the system gets confirmation for removal.

3.3.2 API and Database Implementation

3.3.2.1 API Endpoints

Through its RESTful API system, the authentication structure handles both add user alongside remove operations. Below is a structured list of the key API endpoints:

* **Add User:**
* Endpoint: POST /api/addUser
* Description: Add users to the project
* Request Body (JSON):



Figure 23: Add User - Request Body

* Response:

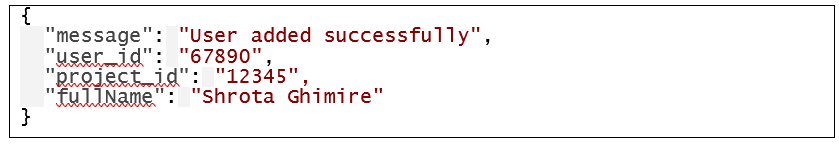


Figure 24: Success Message

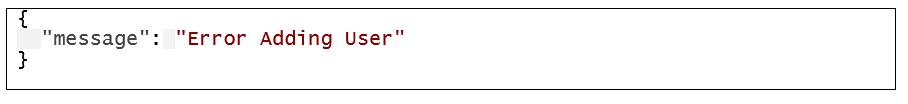


Figure 25: Error Message

* Remove User:
* Endpoint: POST /api/removeUser
* Description: Removing users from the project
* Request Body (JSON):



Figure 26: Remove User – Request Body

* Response:

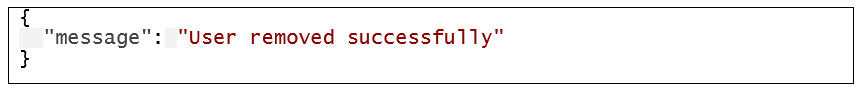


Figure 27: Success Message

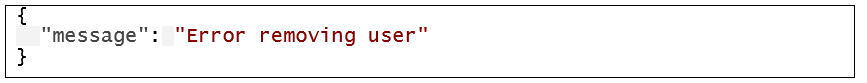


Figure 28: Error message

## 3.4 Development Process

## 3.4.1 Sprint Planning

This subsystem ‘User Management’ is my sprint 2. The duration of this sprint is 2 weeks. During this time, I will develop a user management system with frontend and backend integration. I have listed the tasks I will do, and they’re on the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Days** | **Tasks** | **Goals** | **Deliverables** |
| Day 1 | Create frontend UI for adding users | User form design | HTML form with CSS |
| Day 2 | Create frontend UI for removing users | Display users and remove option | User list with delete button |
| Day 5-6 | Design database schema for users | Define user attributes | MySQL table for users |
| Day 7-8 | Develop API or adding and removing users | Handle user addition and deletion in backend | Endpoints |
| Day 9-10 | Integrate frontend with backend | Connect UI with API | Working add/ remove features |
| Day 10-11 | Implement input validation | Prevent errors | Frontend form validation |
| Day 13-14 | Improve the UI layout and styling | Better user experience | CSS improvements |

## 3.4.2 Execution Details

I executed this work through Agile methodology to create and optimize separate elements in the system. The development approach implemented systematic workflow beginning with frontend layout construction which then integrated backend application programming interfaces.

* **Frontend Development**

Development begins with creating an attractive user interface system that lets users communicate with the system. A fast and modular development environment will be built from HTML and CSS and JS components at the front end. The frontend components need to integrate with form validation as well as error handling functions and API integration services. The developers will build application responsiveness before focusing on maintaining user experience naturalness after finishing the UI development stage.

* **Backend Development**

The Node.js backend development launches following completion of frontend development. The system will require APIs to perform adding user and removing through user management. The database architecture requires architectural patterns to keep operational information and credentials in the storage system. MySQL will be selected because performing data management on users demands dependable system performance.

* **Integration**

The front-end design integration with backend APIs will commence after the independent development of both front-end and back-end components. The user management executes front-end requests which are transmitted to backend operations.

**Tools and Technologies used:**

|  |  |
| --- | --- |
| **Category** | **Tools/Technology used** |
| Frontend | HTML, CSS, JavaScript |
| Backend | Node.js |
| Database | MySQL |
| Version Control | GitHub |

## 3.5 Testing Approach

Testing stands as an essential developmental component for verifying that every system component performs to specifications. Research-based testing extends across unit testing and integration testing and user testing for complete system examination.

## 3.5.1 Unit Testing

Unit testing performs analysis on individual system components while keeping these components separate from the remaining parts of the system. The successful operation of system components leads to this objective. The following strategies serve as testing functions:

* **Testing Functions**

Tests will be devoted to each backend operating functionality including user addition and removal processes. The tests will check that user data insertion through the add user function functions properly and that user removal operations are executed correctly via remove user function. Tests will verify that all database interactions proceed as expected through API requests and responses to prevent unusual behavior during system usage.

* **Test Coverage**

Testing will be carried out for the core features of user addition and removal which includes both appropriate and inappropriate data scenarios as well as unexpected situations. The system tests user input validity through inspection procedures to validate appropriate database data entry combined with unapproved entry denial. The test will verify the error handling system activates proper error messaging when failed operations occur, for instance nonexistent users or incorrect input values. A full set of tests covering all essential components will confirm proper system performance.

## 3.5.2 Integration Testing

Integration testing needs different system modules and components to exchange data correctly to fulfill their expected performance. The success of data exchange verification between system components and their collective functionality depends on the results of this test. The following strategies represent the approach for integration testing:

* **Testing API Endpoints**

I will perform integration test operations on the user management API routes which contain both add user and remove user endpoints. This test will confirm that both user addition succeeds and that appropriate responses are generated. I will test the API's user management feature by confirming it allows for both successful user addition and frontend-to-backend communication to refresh the user data in the user interface. A back-end process that effectively removes user data from the database must function in parallel with user interface updates to reflect the deletion.

* **Database Integration**

The database operations need verification for accurate processing during insert and delete operations. During the addition of new users, the system must properly insert data into the users' table and during removal it must correctly delete associated user data. I will verify which user data selects from the database while also checking for storage and retrieval accuracy within database processes. The testing procedure confirms smooth operation between database functions and backend APIs.

## 3.5.3 User Testing

The realistic evaluation of system performance based on end-user requirements takes place during user testing. Testing verifies both usability and user requirements satisfaction in one system.

* **User Interface Testing**

I will conduct tests for all frontend elements that include both the add user form and remove user form as well as buttons and navigation items. The application seeks to maintain working forms alongside an intuitive interface which provides users with reliable information about each input. Users will be able to successfully perform system tasks related to user management through testing that confirms their ability to do so without confusion.

* **Usability Testing**

Users' system interaction regarding user management actions will serve as the basis for collecting feedback through observation methods. The researcher will document test participant encounters and log their struggle points while filling in user information and completing the submission requirements as well as user verification procedures. The gathered data will point out opportunities to enhance user interaction.

* **Performance Testing**

The maximum operational stress test for the system will concentrate on performing user management functions. My assessment will evaluate how fast the API requests execute together with the user enrollment and de-enrollment system functions. The system evaluation aims to demonstrate stability and peak performance as users submit increasingly high numbers of requests.

* **End-To-End Flow**

Testing will be executed across the system to imitate actual user operations involving the addition and removal of users. I will test the system to ensure it works properly during these procedures thus allowing users to complete their tasks without hurdles. The test checks that both the logic and interface of user management operations flow smoothly throughout the process.

## 3.6 Conclusion

The User Management contains essential management features which enable an efficient process to add or remove users to the Task Management System. The system development can be finished through an Agile process by starting with frontend work and backend development until it reaches integration completion. The system will guarantee a pleasant user experience through both robust validation controls and excellent error reporting coupled with secure SQL database methods. A combination of unit testing and integration testing and user testing will validate the system functionality leading to reliability and stability verification. Through the User Management System users will access a safe and effective platform for managing user administration operations which enables both user addition and removal to function without disrupting system security.