CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G1
SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: Laboratory Activity: Mini App

Prepared By: Zydric Abel

Date of Submission: 02/06/2026

Version: 1.1

Table of Contents

1. Introduction 3
    1.1. Purpose 3
    1.2. Scope 3
    1.3. Definitions, Acronyms, and Abbreviations 3
2. Overall Description 3
    2.1. System Perspective 3
    2.2. User Classes and Characteristics 3
    2.3. Operating Environment 3
    2.4. Assumptions and Dependencies 3
3. System Features and Functional Requirements 3
    3.1. Feature 1: 3
    3.2. Feature 2: 3
4. Non-Functional Requirements 3
5. System Models (Diagrams) 4
    5.1. ERD 4
    5.2. Use Case Diagram 4
    5.3. Activity Diagram 4
    5.4. Class Diagram 4
    5.5. Sequence Diagram 4
6. Appendices 4

1. Introduction

1.1 Purpose

This document describes the requirements for a simple User Authentication Mini App that allows users to register, login, view their profile, and logout.

Intended Audience:

- Course instructor evaluating the design
- Developer implementing the system

1.2 Scope

What the system will do:

- Allow new users to register with email and password
- Authenticate users through login mechanism
- Provide secure access to protected dashboard/profile pages
- Enable users to logout and clear their session
- Prevent unauthorized access to protected resources when logged out.

System Boundaries:

- **In scope:** User registration, login, logout, and basic profile viewing
- **Out of scope:** Password reset, email verification, social media login, role-based permissions, multi-factor authentication

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
API	Application Programming Interface - endpoints for communication between frontend and backend
JWT	JSON Web Token - secure method for transmitting information between parties
BCrypt	Password hashing algorithm used for secure password storage
SPA	Single Page Application - React frontend architecture

REST	Representational State Transfer - architectural style for web services
CRUD	Create, Read, Update, Delete - basic database operations
JPA	Java Persistence API - Java framework for managing relational data
DTO	Data Transfer Object - object used to transfer data between processes
HTTP	Hypertext Transfer Protocol - foundation of data communication on the web
CORS	Cross-Origin Resource Sharing - mechanism for allowing restricted resources
LocalStorage	Web browser storage mechanism for storing data locally
Token	Authentication credential returned after successful login
Protected Route	Application page that requires authentication to access
Guest User	User who has not logged in
Authenticated User	User who has successfully logged in

2. Overall Description

2.1 System Perspective

The User Authentication Mini App is a standalone full-stack web application consisting of:

Frontend (React):

- Single Page Application (SPA)
- Communicates with backend via RESTful API
- Manages user interface and client-side state

Backend (Spring Boot):

- RESTful API server
- Handles business logic and authentication
- Manages database operations

Database (MySQL):

- Stores user credentials and information
- Maintains data persistence

Architecture:

```
[React Frontend] <-- HTTP/REST --> [Spring Boot Backend] <-- JPA --> [MySQL Database]
```

2.2 User Classes and Characteristics

1. Guest User (Unauthenticated)

- **Description:** Visitors who have not logged in
- **Technical Expertise:** Basic computer skills
- **Frequency of Use:** One-time registration, periodic login
- **Privileges:** Can register new account, can attempt login
- **Characteristics:**
  - No access to protected pages
  - Can view login and registration pages
  - Automatically redirected when attempting to access protected content

2. Authenticated User (Logged In)

- **Description:** Users who have successfully logged in
- **Technical Expertise:** Basic computer literacy
- **Frequency of Use:** Regular application usage
- **Privileges:** Full access to dashboard/profile, can logout
- **Characteristics:**
  - Has valid JWT token stored in browser
  - Can access all protected application features when authenticated
  - Can view and use dashboard/profile page when authenticated
  - Session persists until logout or token expiration

2.3 Operating Environment

Client-Side Requirements:

- Modern web browser (Chrome, Firefox, Safari, or Edge)
- JavaScript enabled
- Internet connection required
- Browser localStorage support

Server-Side Requirements:

- **Operating System:** Windows, macOS, or Linux
- **Java:** JDK 17 or higher
- **Database:** MySQL 8.0 or higher

Development Tools:

- **Backend:** Spring Boot 3, Maven, IntelliJ IDEA
- **Frontend:** Node.js 16+, npm or yarn, VS Code
- **Database:** MySQL, PostgreSQL
- **API Testing:** Postman

Network Requirements:

- Backend runs on http://localhost:8080
- Frontend runs on http://localhost:3000
- CORS configured to allow cross-origin requests between frontend and backend

2.4 Assumptions and Dependencies

Assumptions:

1. Users have valid email addresses
2. Users have stable internet connection
3. Browser supports localStorage and modern JavaScript
4. Development environment is trusted (localhost)

Dependencies:

1. MySQL database must be running
2. Java JDK 17+ required for backend
3. Node.js required for frontend development
4. Spring Security for password hashing
5. JWT library for token generation

6. React Router for frontend navigation
7. Axios for HTTP requests

---

3. System Features and Functional Requirements

3.1. Feature 1: User Registration

Description:
Allows new users to create an account by providing their email, password, first name, and last name. The system validates the input, checks for duplicate emails, hashes the password, and stores the user information in the database.

Functional Requirements:

- **FR-1.1:** System shall provide a registration form with the following fields:
  - Email address (required, must be valid email format)
  - Password (required, minimum 6 characters)
  - First Name (required, alphabetic characters only)
  - Last Name (required, alphabetic characters only)
- **FR-1.2:** System shall validate email format on the client side before submission
- **FR-1.3:** System shall check if the email already exists in the database
  - If email exists, display error: "Email already registered"
  - If email is unique, proceed with registration
- **FR-1.4:** System shall hash the password using BCrypt algorithm before storing in database
- **FR-1.5:** System shall store user information in the database with the following fields:
  - Auto-generated unique ID
  - Email address
  - Hashed password
  - First name
  - Last name
  - Created timestamp
- **FR-1.6:** System shall display success message upon successful registration
- **FR-1.7:** System shall redirect user to login page after successful registration
- **FR-1.8:** System shall display appropriate error messages for:
  - Invalid email format
  - Duplicate email
  - Empty required fields
  - Server errors

3.2. Feature 2: User Login

Description:
Allows registered users to authenticate themselves by providing their email and password. Upon successful authentication, the system generates a JWT token and grants access to protected resources.

Functional Requirements:

- **FR-2.1:** System shall provide a login form with the following fields:
  - Email address (required)
  - Password (required)
- **FR-2.2:** System shall retrieve user record from database based on provided email
- **FR-2.3:** System shall verify the provided password against the stored hashed password using BCrypt
- **FR-2.4:** System shall generate a JWT token upon successful authentication containing:
  - User email as subject
  - Issue timestamp
- **FR-2.5:** System shall return the JWT token and user information to the client upon successful login
- **FR-2.6:** System shall store the JWT token in browser localStorage on the client side
- **FR-2.7:** System shall redirect authenticated user to dashboard/profile page
- **FR-2.8:** System shall display error message "Invalid credentials" for:
  - Non-existent email
  - Incorrect password
  - Empty fields
- **FR-2.9:** System shall not reveal whether email or password was incorrect (security measure)

3.3. Feature 3: View Dashboard/Profile

Description:
Allows authenticated users to view their profile information on a protected dashboard page. The system verifies the JWT token before granting access.

Functional Requirements:

- **FR-3.1:** System shall require valid JWT token in the Authorization header for dashboard access
- **FR-3.2:** System shall validate token signature.

- **FR-3.3:** System shall retrieve user information from database based on token data
- **FR-3.4:** System shall display user profile information:
  - First name
  - Last name
  - Email address
  - Account creation date
- **FR-3.5:** System shall redirect to login page if:
  - Token is missing
  - Token is invalid
  - Token is expired
- **FR-3.6:** System shall maintain authentication state across page refreshes if valid token exists
- **FR-3.7:** System shall automatically include JWT token in all API requests to protected endpoints

3.4. Feature 4: User Logout

Description:
Allows authenticated users to terminate their session and clear authentication credentials from the browser.

Functional Requirements:

- **FR-4.1:** System shall provide a logout button/link accessible from the dashboard
- **FR-4.2:** System shall remove JWT token from browser localStorage upon logout
- **FR-4.3:** System shall clear any cached user information from the client
- **FR-4.4:** System shall redirect user to login page after logout
- **FR-4.5:** System shall display logout success message
- **FR-4.6:** System shall prevent access to protected pages after logout.

3.5. Feature 5: Protected Routes

Description:
Prevents unauthorized access to protected application pages by implementing route guards that check for valid authentication tokens.

Functional Requirements:

- **FR-5.1:** System shall check for valid JWT token before rendering protected pages
- **FR-5.2:** System shall redirect unauthenticated users to login page when attempting to access protected routes

- **FR-5.3:** System shall allow authenticated users to access protected routes without additional login
- **FR-5.4:** System shall define the following route access levels:
    - Public routes: Login, Register (accessible to all)
    - Protected routes: Dashboard, Profile (require authentication)
- **FR-5.5:** System shall automatically redirect to appropriate page based on authentication state:
    - Authenticated user accessing login page ⟶ Redirect to dashboard
    - Unauthenticated user accessing protected page ⟶ Redirect to login

---

4. Non-Functional Requirements

4.1 Performance Requirements

- Login/Register complete in under 3 seconds
- API responses under 2 seconds

4.2 Security Requirements

- Passwords hashed with BCrypt
- JWT tokens expire after 24 hours
- HTTPS in production (HTTP ok in development)
- Passwords shall never be stored in plain text
- System shall not expose sensitive information in error messages
- SQL injection prevention through parameterized queries (JPA)
- XSS prevention through proper input sanitization

4.3 Usability Requirements

- Simple, clear interface
- Helpful error messages
- Form validation shall provide immediate feedback
- Navigation shall be consistent across all pages

4.4 Maintainability Requirements

- Code shall follow standard naming conventions
- Layered architecture (Controller ⟶ Service ⟶ Repository)
- Frontend shall use component-based architecture
- System shall separate concerns between presentation, business logic, and data access

4.5 Data Integrity

- Email addresses shall be unique in database
- All required fields shall be validated
- Timestamps shall be automatically generated
- Database constraints shall prevent invalid data

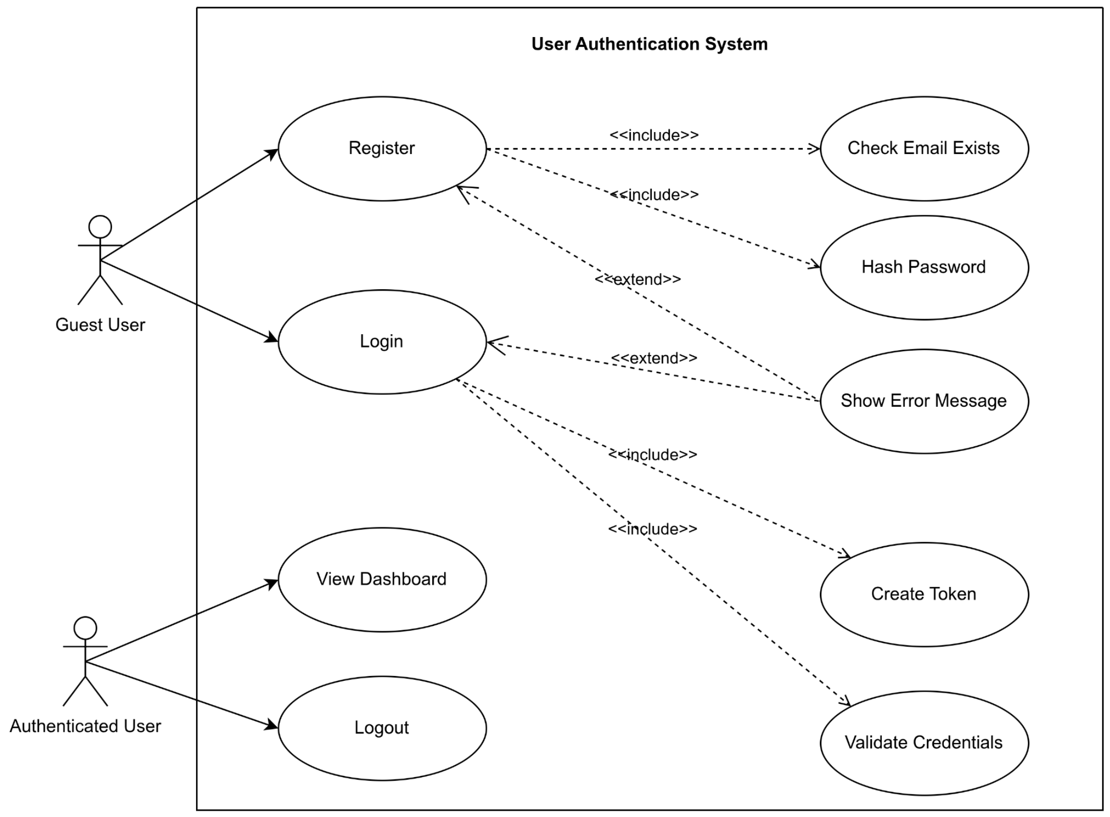
---

5. System Models (Diagrams)

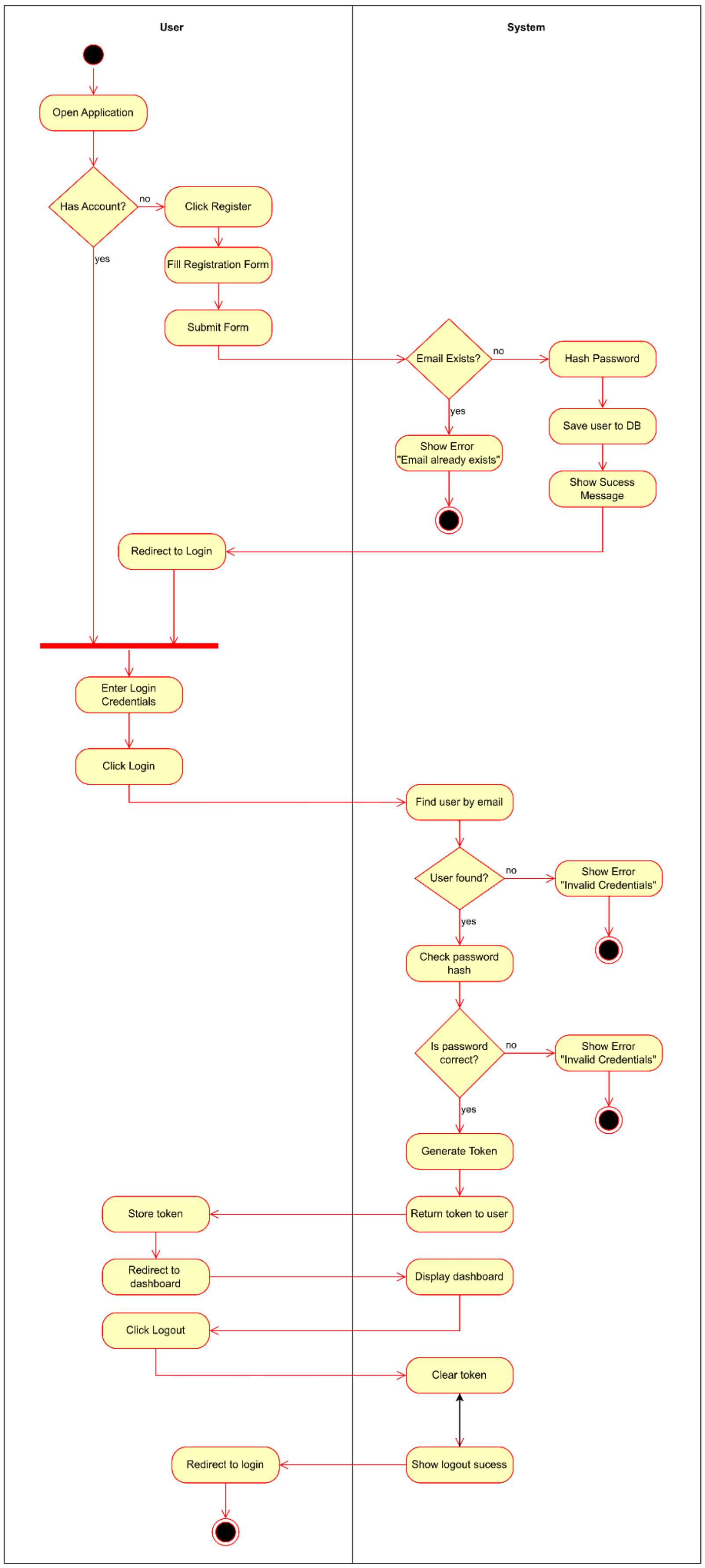
5.1 ERD (Users table)

Users	
PK	id INTEGER
	email: VARCHAR(255)
	password: VARCHAR(255)
	firstName: VARCHAR(100)
	lastName: VARCHAR(100)
	createdAt: TIMESTAMP

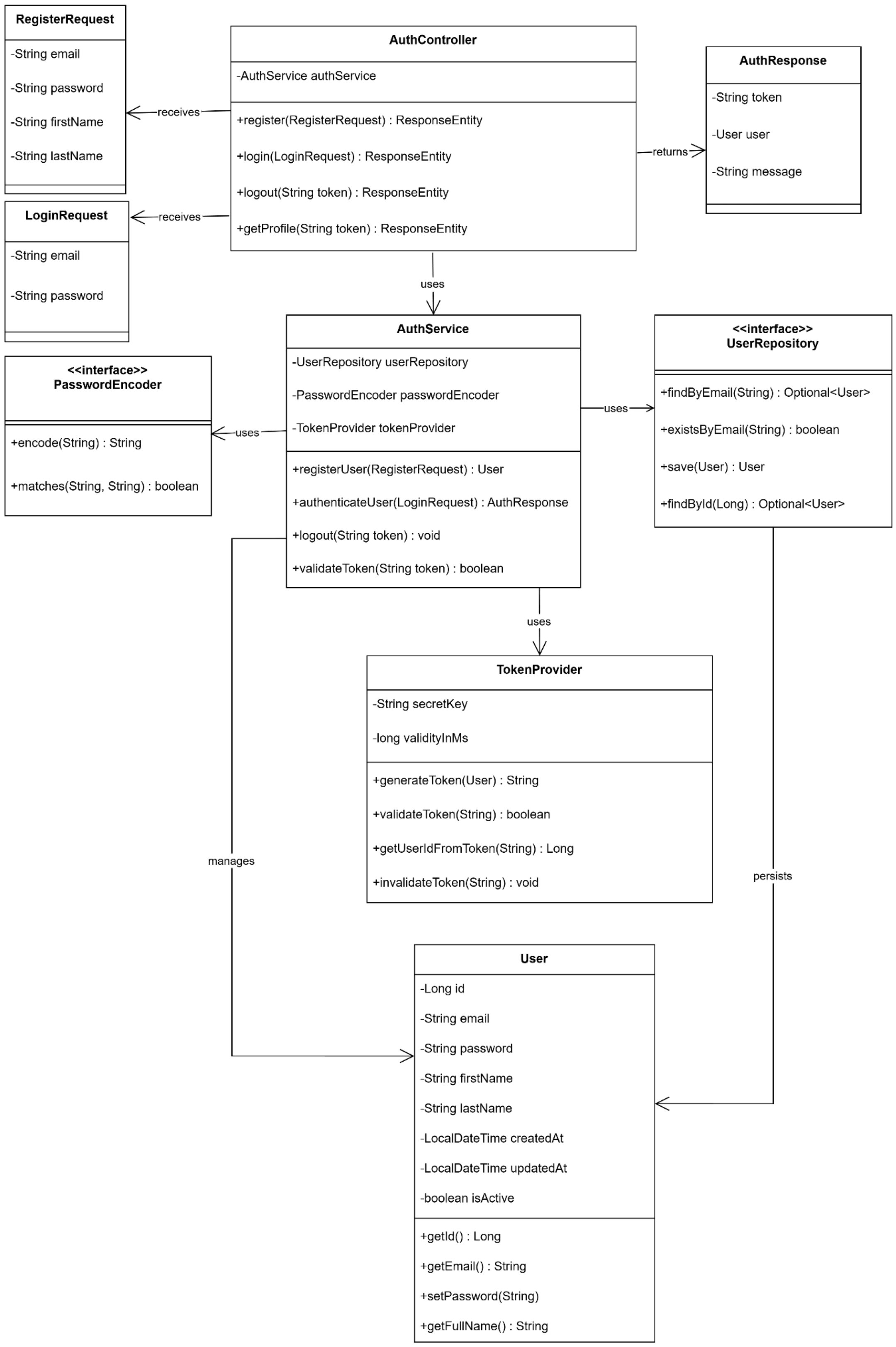
5.2 Use Case Diagram



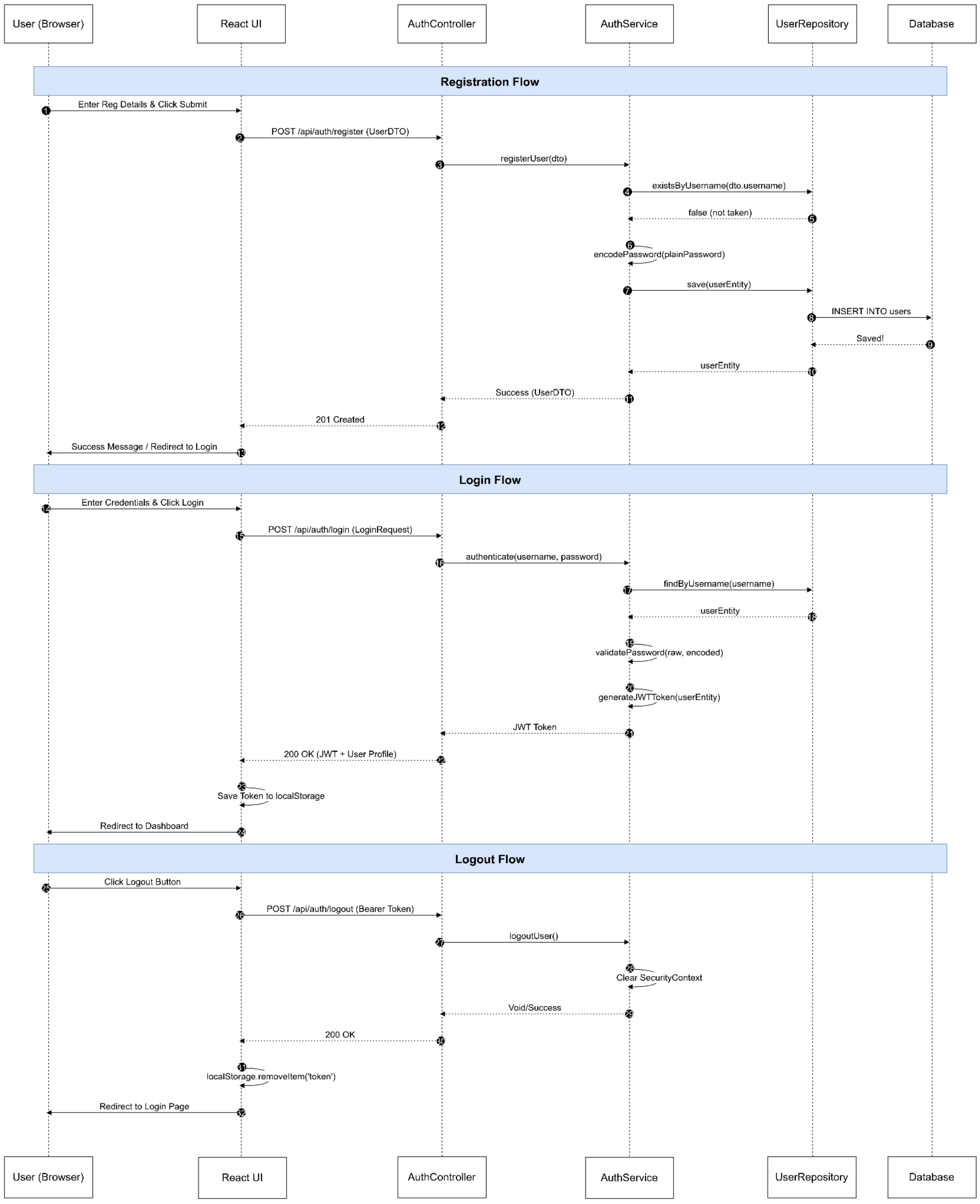
5.3 Activity Diagram



5.4 Class Diagram


RegisterRequest
-String email
-String password
-String firstName
-String lastName
receives
AuthController
-AuthService authService
+register(RegisterRequest) : ResponseEntity
+login(LoginRequest) : ResponseEntity
+logout(String token) : ResponseEntity
+getProfile(String token) : ResponseEntity
returns
AuthResponse
-String token
-User user
-String message
LoginRequest
-String email
-String password
receives
uses
AuthService
-UserRepository userRepository
-PasswordEncoder passwordEncoder
-TokenProvider tokenProvider
+registerUser(RegisterRequest) : User
+authenticateUser(LoginRequest) : AuthResponse
+logout(String token) : void
+validateToken(String token) : boolean
<<interface>>
PasswordEncoder
+encode(String) : String
+matches(String, String) : boolean
uses
uses
<<interface>>
UserRepository
+findByEmail(String) : Optional<User>
+existsByEmail(String) : boolean
+save(User) : User
+findById(Long) : Optional<User>
uses
TokenProvider
-String secretKey
-long validityInMs
+generateToken(User) : String
+validateToken(String) : boolean
+getUserIdFromToken(String) : Long
+invalidateToken(String) : void
manages
persists
User
-Long id
-String email
-String password
-String firstName
-String lastName
-LocalDateTime createdAt
-LocalDateTime updatedAt
-boolean isActive
+getId() : Long
+getEmail() : String
+setPassword(String)
+getFullName() : String

5.5 Sequence Diagram


User (Browser)
React UI
AuthController
AuthService
UserRepository
Database
Registration Flow
1 Enter Reg Details & Click Submit
2 POST /api/auth/register (UserDTO)
3 registerUser(dto)
4 existsByUsername(dto.username)
5 false (not taken)
6 encodePassword(plainPassword)
7 save(userEntity)
8 INSERT INTO users
9 Saved!
10 userEntity
11 Success (UserDTO)
12 201 Created
13 Success Message / Redirect to Login
Login Flow
14 Enter Credentials & Click Login
15 POST /api/auth/login (LoginRequest)
16 authenticate(username, password)
17 findByUsername(username)
18 userEntity
19 validatePassword(raw, encoded)
20 generateJWTToken(userEntity)
21 JWT Token
22 200 OK (JWT + User Profile)
23 Save Token to localStorage
24 Redirect to Dashboard
Logout Flow
25 Click Logout Button
26 POST /api/auth/logout (Bearer Token)
27 logoutUser()
28 Clear SecurityContext
29 Void/Success
30 200 OK
31 localStorage.removeItem('token')
32 Redirect to Login Page

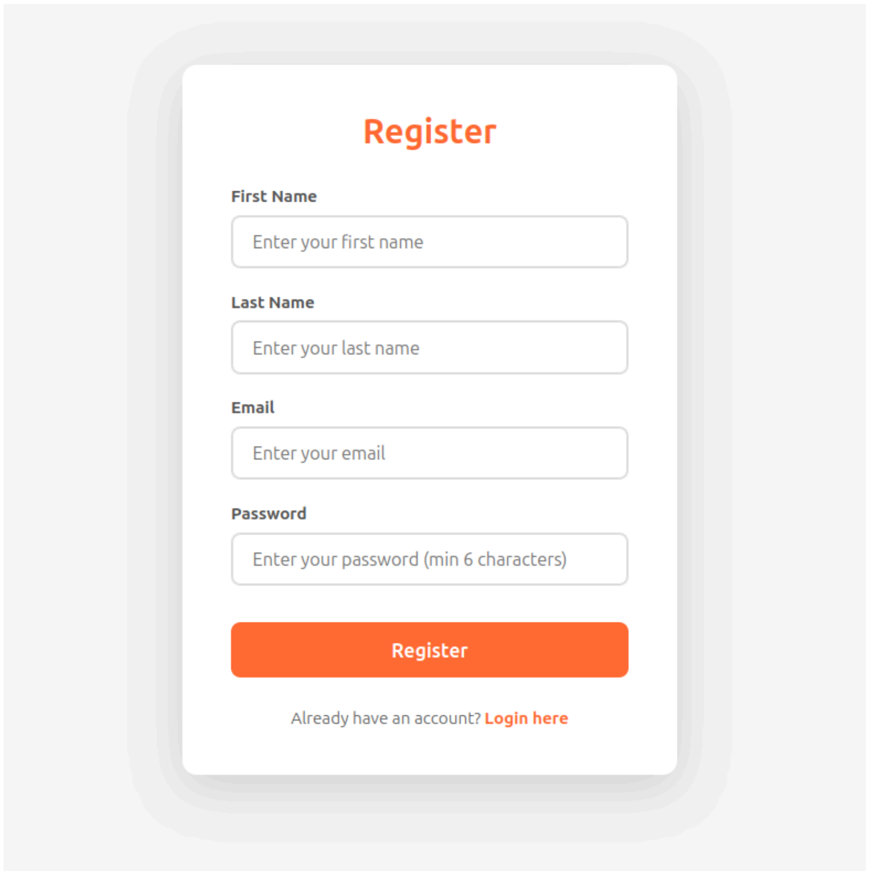
6. Appendices

6.1 API Endpoints Summary

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	Register new user	No
POST	/api/auth/login	Login user	No
GET	/api/auth/profile	Get user profile	Yes

6.2 Web Screenshots

- **Register**



Register

First Name

Enter your first name

Last Name

Enter your last name

Email

Enter your email

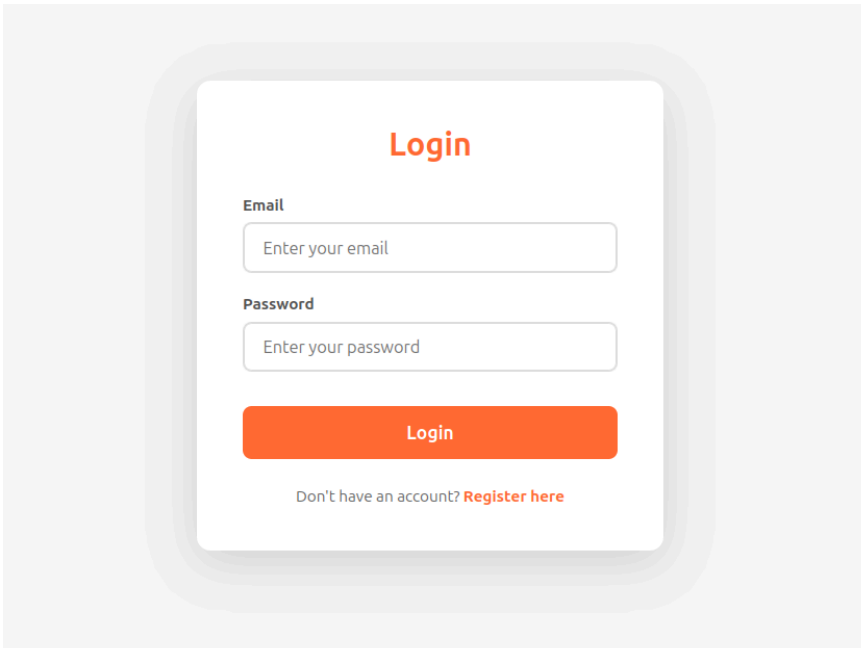
Password

Enter your password (min 6 characters)

Register

Already have an account? **Login here**

- **Login**



Login

Email

Enter your email

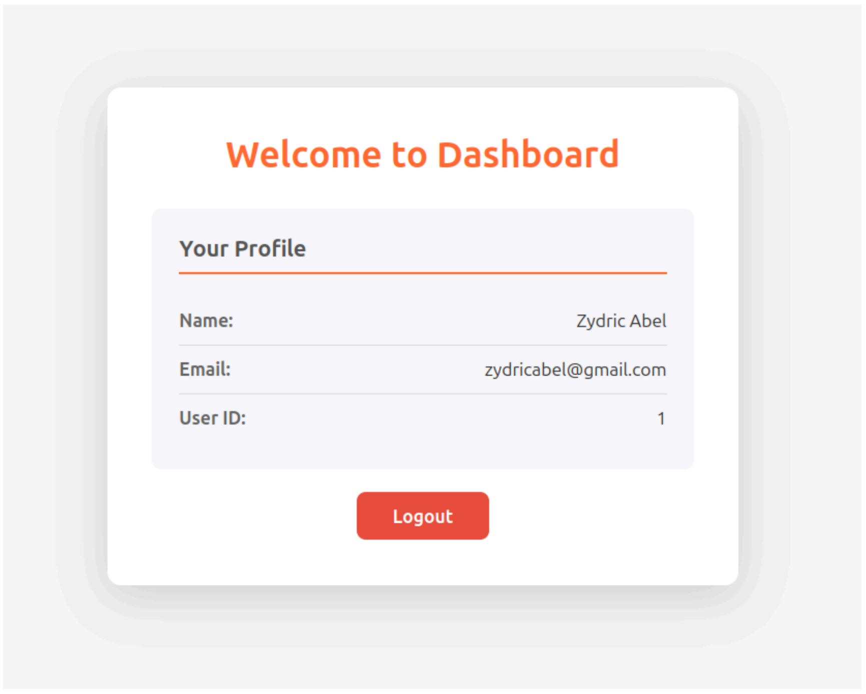
Password

Enter your password

Login

Don't have an account? Register here

- **Dashboard/Profile**



Welcome to Dashboard

Your Profile

Name:	Zydric Abel
Email:	zydricabel@gmail.com
User ID:	1

Logout

- **Logout**