CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342 - G4
SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: Mini App

Prepared By: Leana Mariflor A. Belaguas

Date of Submission: 02/02/26

Version: 1

Table of Contents

1. Introduction

1.1. Purpose

The objective of this system is to furnish a secure and dependable authentication module for a web application. This document is intended for developers, quality assurance testers, and stakeholders to facilitate a comprehensive understanding of the system's operational flow and database architecture before implementation.

1.2. Scope

The system manages User Registration, Login, Profile Viewing, and Logout. It establishes boundaries by protecting specific dashboard and profile routes, ensuring that protected pages cannot be accessed when a user is logged out.

1.3. Definitions, Acronyms, and Abbreviations

ERD: Entity Relationship Diagram.

JWT: JSON Web Token used for secure, stateless authentication.

UML: Unified Modeling Language used for the Activity, Class, and Sequence diagrams.

SPA: Single Page Application (referring to the React UI).

2. Overall Description

2.1. System Perspective

This module acts as the "Gatekeeper" for the application. It manages the interaction between the React UI, the Spring Boot API, and the Database.

2.2. User Classes and Characteristics

Guest User: Unauthenticated individuals who can only access the Register and Login use cases.

Authenticated User: Users who have successfully logged in and are granted access to View Profile/Dashboard and Logout.

2.3. Operating Environment

Frontend: ReactJS.

Backend: Spring Boot.

Database: MySQL.

Tools: draw.io / diagrams.net.

2.4. Assumptions and Dependencies

3. System Features and Functional Requirements

3.1. Feature 1: User Registration

Description: Allows new users to create an account by providing a username, email, and password.

Functional Requirements:

- The system shall provide a Register Page for user input.
- The system shall validate if the Username or Email already exists in the Database via the existsByEmail method.
- If unique, the system shall Hash/Encrypt the password using the PasswordEncoder.
- The system shall save the user record with fields including userID, username, email, password, and role.

3.2. Feature 2: Authentication (Login/Logout)

Description: Validates user credentials and initiates an authenticated session.

Functional Requirements:

- The system shall verify credentials using the authenticate method in the AuthService.
- Upon successful verification, the TokenProvider shall Generate a JWT.
- The React UI shall store the token (e.g., in localStorage) to maintain the authenticated state.
- The system shall return a 401 Unauthorized error for invalid credentials.

3.3. Feature 3: Profile and Dashboard Access

Description: Allows authenticated users to view protected data.

Functional Requirements:

- The system shall allow access to the Dashboard and Profile only if a valid JWT is provided.
- The TokenProvider must validate the token for every protected request.
- The system shall fetch the current user's data from the Database using findByUsername.

3.4. Feature 4: Logout

Description: Terminates the user session.

Functional Requirements:

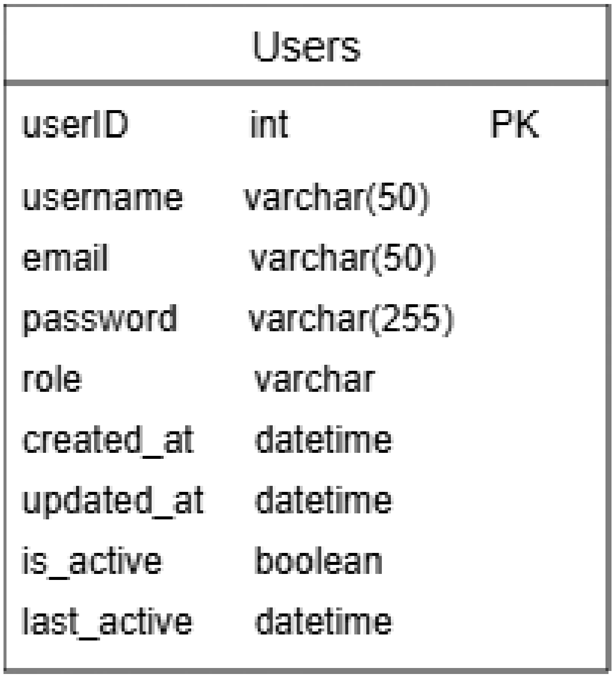
- The system shall invalidate the token via the TokenProvider.
- The React UI shall clear local storage and reset the authentication state.
- The system shall redirect the user back to the login or landing page.

Non-Functional Requirements

- Security: Passwords must be hashed; protected pages must remain inaccessible to unauthenticated users.
- Usability: The system must provide clear feedback, such as "Email already taken" or "Invalid Credentials".
- Traceability: Every user record must include created_at and updated_at timestamps for auditing.

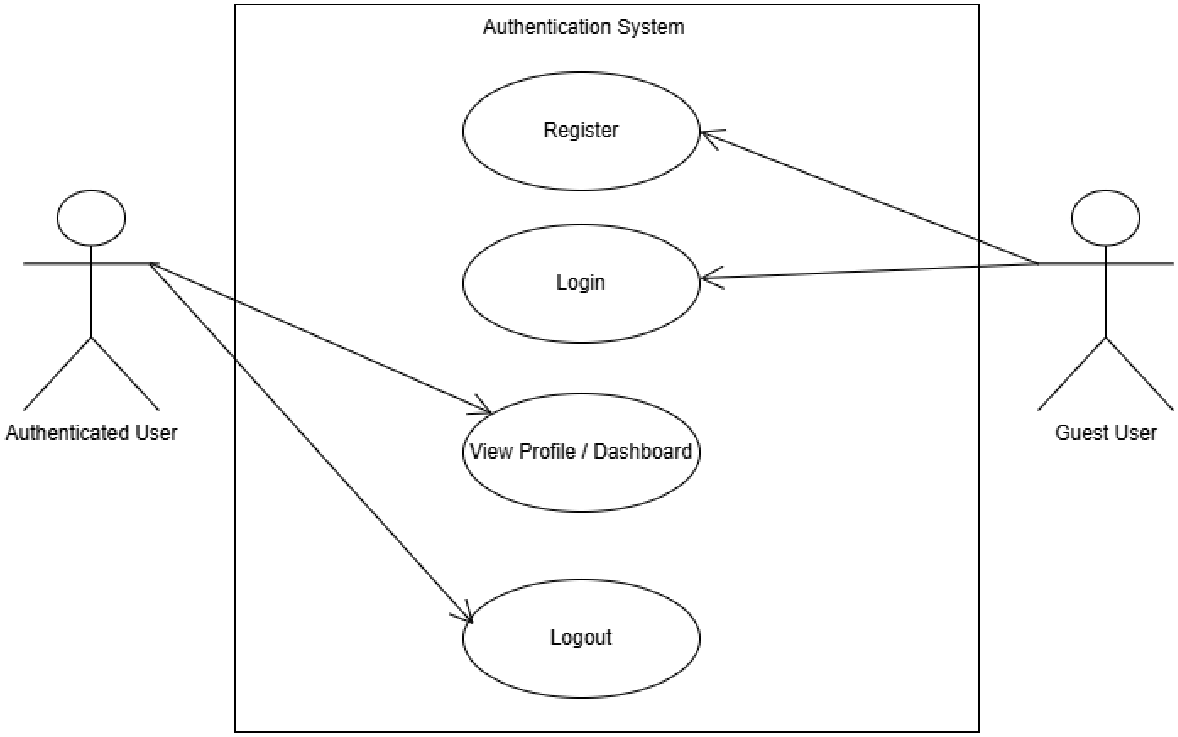
4. System Models (Diagrams)

4.1. ERD

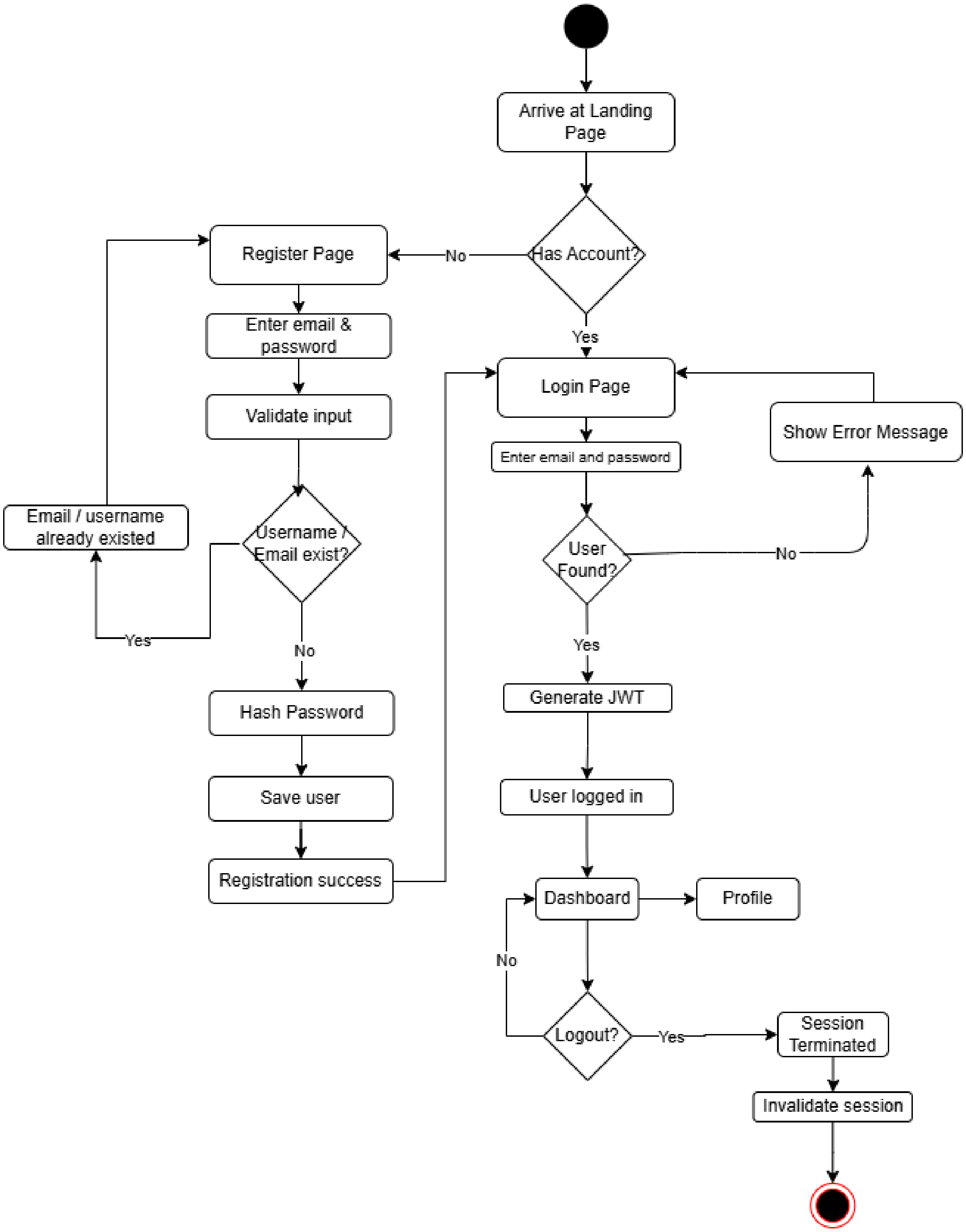


Users		
userID	int	PK
username	varchar(50)	
email	varchar(50)	
password	varchar(255)	
role	varchar	
created_at	datetime	
updated_at	datetime	
is_active	boolean	
last_active	datetime	

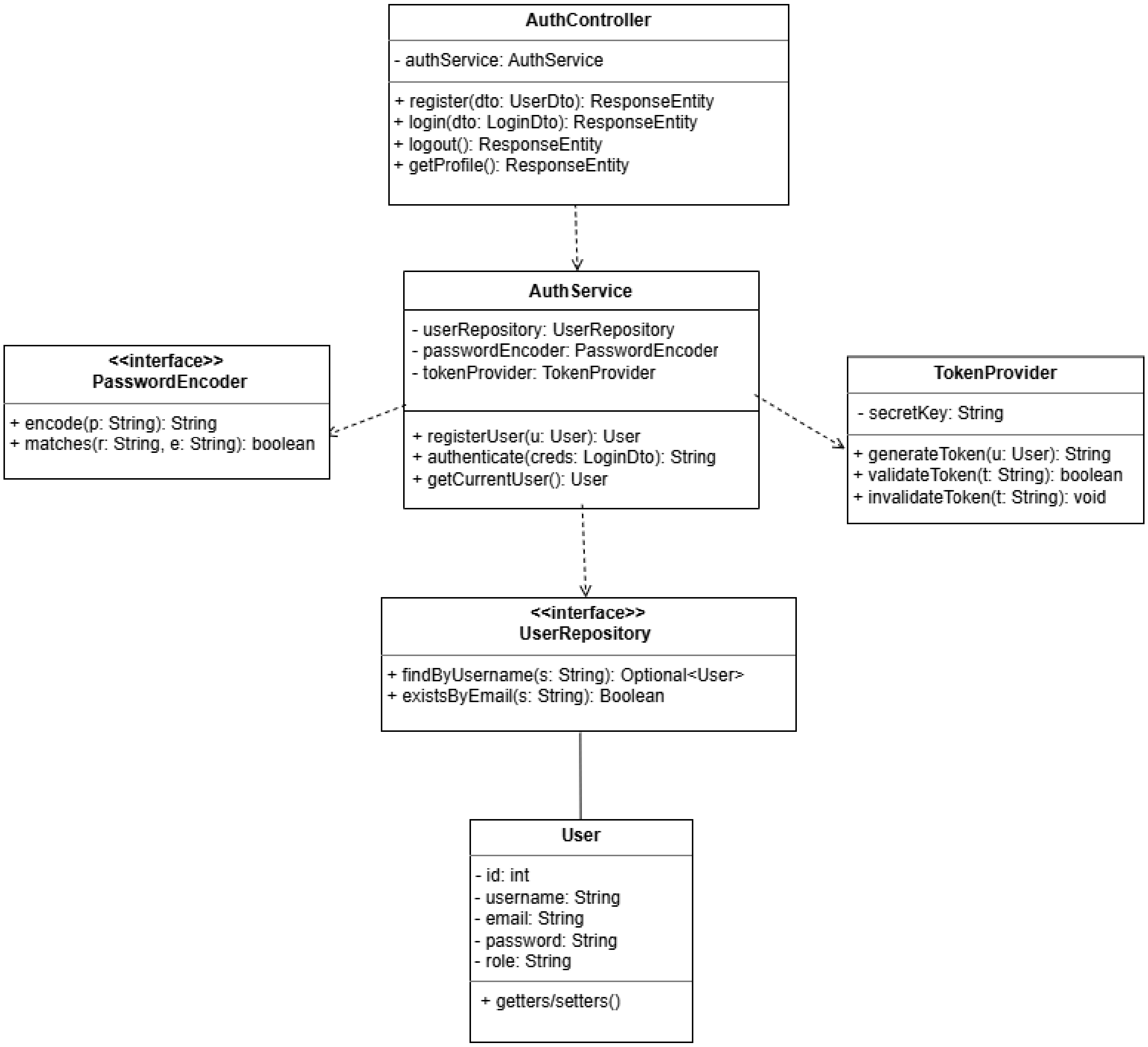
4.2. Use Case Diagram



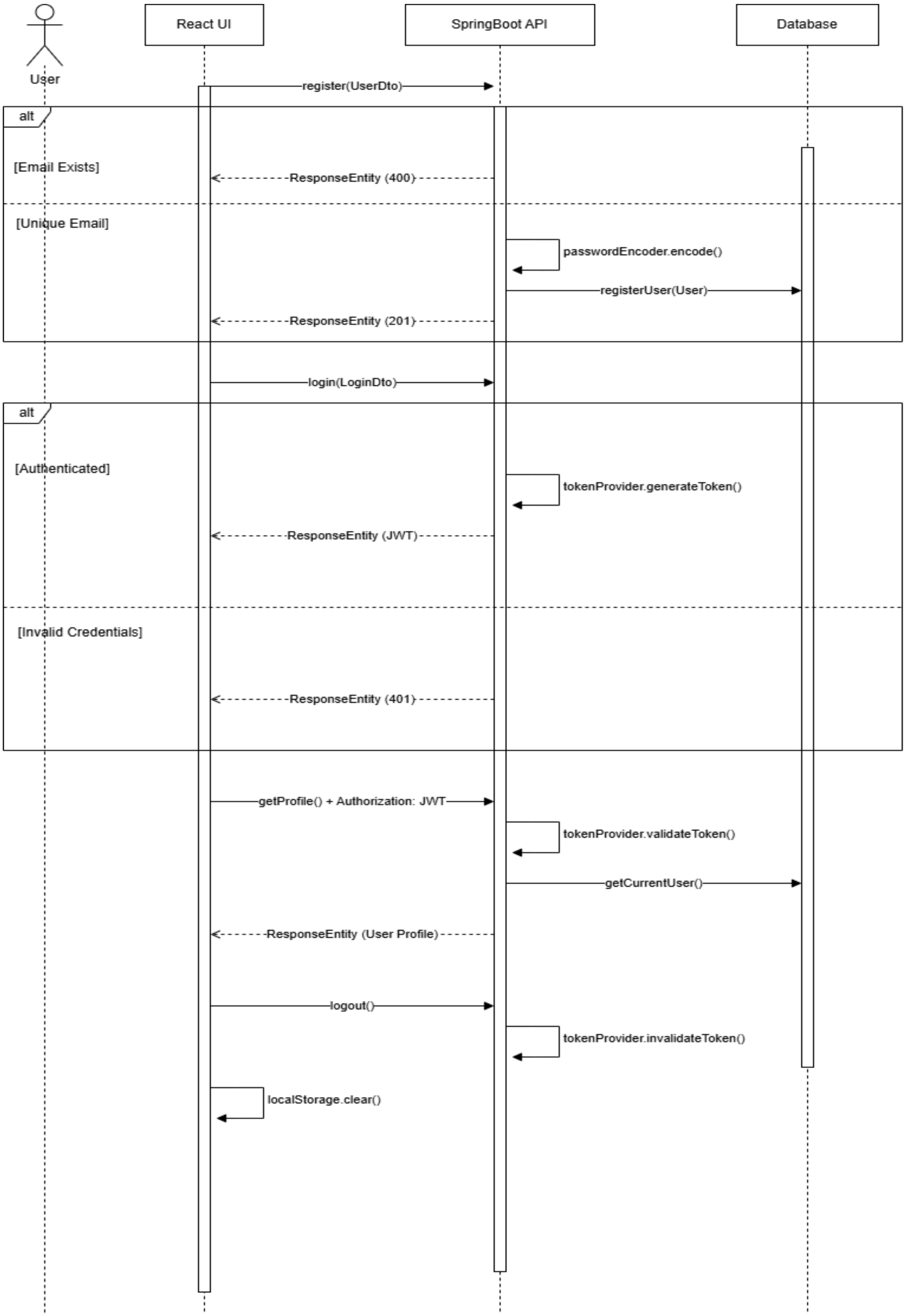
4.3. Activity Diagram



4.4. Class Diagram


AuthController
- authService: AuthService
+ register(dto: UserDto): ResponseEntity
+ login(dto: LoginDto): ResponseEntity
+ logout(): ResponseEntity
+ getProfile(): ResponseEntity
AuthService
- userRepository: UserRepository
- passwordEncoder: PasswordEncoder
- tokenProvider: TokenProvider
+ registerUser(u: User): User
+ authenticate(creds: LoginDto): String
+ getCurrentUser(): User
<<interface>>
PasswordEncoder
+ encode(p: String): String
+ matches(r: String, e: String): boolean
TokenProvider
- secretKey: String
+ generateToken(u: User): String
+ validateToken(t: String): boolean
+ invalidateToken(t: String): void
<<interface>>
UserRepository
+ findByUsername(s: String): Optional<User>
+ existsByEmail(s: String): Boolean
User
- id: int
- username: String
- email: String
- password: String
- role: String
+ getters/setters()

4.5. Sequence Diagram



5. Appendices

Include any additional information, references, or support materials.