CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G1
SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: User Registration & Authentication

Prepared By: Rentuma, Trixie Ann V.

Date of Submission: 2/06/2026

Version: 0.1

Table of Contents

1. Introduction

1.1. Purpose

This Functional Requirements Specification (FRS) document describes the requirements for a User Registration and Authentication system. The system will provide secure user account creation, login functionality, and session management capabilities. This document is intended for developers, system architects, quality assurance teams, project managers, and stakeholders involved in the development and deployment of the authentication system.

1.2. Scope

User Registration and Authentication system will:

- Allow new users to create accounts with unique credentials
- Authenticate existing users through secure login mechanisms
- Manage user sessions with appropriate timeout mechanisms
- Provide password reset and recovery functionality
- Validate user input to ensure data integrity and security
- Store user credentials securely using industry-standard encryption

The system will not include advanced features such as multi-factor authentication, social media login integration, or role-based access control in this version.

1.3. Definitions, Acronyms, and Abbreviations

- FRS - Functional Requirements Specification
- API - Application Programming Interface
- HTTPS - Hypertext Transfer Protocol Secure
- JWT - JSON Web Token
- SHA-256 - Secure Hash Algorithm 256-bit
- SQL - Structured Query Language
- UI - User Interface
- Authentication - The process of verifying the identity of a user
- Session - A temporary interaction between a user and the system
- Hash - A one-way cryptographic function that converts passwords into fixed-length strings
- Token - A unique identifier used to maintain user sessions or verify actions

2. Overall Description

2.1. System Perspective

The User Registration and Authentication system is a foundational component of a larger web-based application. It serves as the entry point for user access and security management. The system interfaces with a relational database for persistent storage of user credentials and session data. It communicates with the main application through RESTful APIs, providing authentication services to various modules and features. The system is designed as a standalone microservice that can be integrated with existing or future applications requiring user authentication capabilities.

2.2. User Classes and Characteristics

New Users:

- Description: Individuals creating an account for the first time
- Characteristics: Basic technical literacy, unfamiliar with the system, require clear instructions and guidance
- Frequency of Use: One-time registration process

Registered Users:

- Description: Existing users who log in to access the system
- Characteristics: Familiar with login process, expect quick access, value security and convenience
- Frequency of Use: Daily or multiple times per week

System Administrators:

- Description: Personnel responsible for managing user accounts and system operations
- Characteristics: High technical expertise, require monitoring and management tools, handle support requests
- Frequency of Use: Daily administrative tasks

2.3. Operating Environment

Specify the hardware, software, and tools required to operate the system.

2.4. Assumptions and Dependencies

Assumptions:

- Users have valid email addresses for registration and password recovery
- Users understand basic password security practices
- Database server has sufficient storage capacity for anticipated user growth
- Network infrastructure is reliable and secure
- Users have access to their email accounts for verification purposes

Dependencies:

- Email service (SMTP server) for sending verification and password reset emails
- SSL/TLS certificate for secure HTTPS connections
- Database management system availability and performance
- Third-party libraries for password hashing (bcrypt) and token generation (JWT)
- Web hosting infrastructure with adequate bandwidth and uptime guarantees
- DNS services for domain name resolution

3. System Features and Functional Requirements

3.1. Feature 1: User Registration

The user registration feature allows new users to create an account by providing necessary information including username, email address, and password. The system validates all inputs, checks for duplicate accounts, and securely stores user credentials in the database.

Functional Requirements:

- REG-001: The system shall provide a registration form with fields for username, email, password, and password confirmation
- REG-002: The system shall validate that the username is unique and contains only alphanumeric characters and underscores, with a length between 3-20 characters
- REG-003: The system shall validate that the email address follows standard email format (username@domain.extension)
- REG-004: The system shall validate that the email address is unique in the database
- REG-005: The system shall enforce password requirements: minimum 8 characters, at least one uppercase letter, one lowercase letter, one number, and one special character
- REG-006: The system shall verify that the password and password confirmation fields match
- REG-007: The system shall hash passwords using bcrypt algorithm with a cost factor of 10 or higher before storing in the database
- REG-008: The system shall send a verification email to the provided email address upon successful registration
- REG-009: The system shall display appropriate error messages for invalid inputs or registration failures
- REG-010: The system shall redirect users to the login page upon successful registration
- REG-011: The system shall prevent automated bot registrations using CAPTCHA or similar mechanisms
- REG-012: The system shall store user registration timestamp and IP address for audit purposes

3.2. Feature 2: User Authentication (Login)

The user authentication feature enables registered users to securely log into the system using their credentials. The system verifies the provided username/email and password against stored records and grants access upon successful authentication.

Functional Requirements:

- AUTH-001: The system shall provide a login form with fields for username/email and password

- AUTH-002: The system shall accept either username or email address as the login identifier
- AUTH-003: The system shall verify the provided credentials against stored user records in the database
- AUTH-004: The system shall use secure password comparison methods (bcrypt verify) to validate passwords
- AUTH-005: The system shall generate a session token (JWT) upon successful authentication
- AUTH-006: The system shall implement account lockout after 5 consecutive failed login attempts within 15 minutes
- AUTH-007: The system shall display a generic error message for failed login attempts without revealing whether the username or password was incorrect
- AUTH-008: The system shall redirect authenticated users to the main application dashboard or home page
- AUTH-009: The system shall provide a "Remember Me" option to extend session duration to 7 days
- AUTH-010: The system shall log all login attempts (successful and failed) with timestamp, IP address, and user agent
- AUTH-011: The system shall prevent login for unverified email accounts
- AUTH-012: The system shall implement rate limiting to prevent brute force attacks (maximum 10 attempts per IP per minute)

3.3. Feature 3: Password Management

The password management feature allows users to reset forgotten passwords and change existing passwords. It provides secure mechanisms for password recovery through email verification.

Functional Requirements:

- PWD-001: The system shall provide a "Forgot Password" link on the login page
- PWD-002: The system shall allow users to request password reset by entering their registered email address
- PWD-003: The system shall generate a unique, cryptographically secure password reset token (valid for 1 hour)
- PWD-004: The system shall send password reset instructions via email with a secure reset link containing the token
- PWD-005: The system shall verify the reset token's validity and expiration before allowing password change
- PWD-006: The system shall enforce the same password strength requirements for new passwords as during registration
- PWD-007: The system shall prevent users from reusing their last 3 passwords
- PWD-008: The system shall allow authenticated users to change their password from account settings

- PWD-009: The system shall require current password verification before allowing password change for authenticated users
- PWD-010: The system shall invalidate all existing sessions after a successful password change, requiring re-authentication
- PWD-011: The system shall send a confirmation email after successful password change
- PWD-012: The system shall expire unused password reset tokens after 1 hour

3.4. Feature 4: Session Management

The session management feature maintains user sessions after successful authentication, manages session timeouts, and provides secure logout functionality.

Functional Requirements:

- SESS-001: The system shall create a secure session upon successful user authentication
- SESS-002: The system shall implement session timeout after 30 minutes of inactivity for standard sessions
- SESS-003: The system shall extend the session timeout to 7 days if "Remember Me" is selected during login
- SESS-004: The system shall store session tokens securely using HTTPOnly and Secure cookie flags
- SESS-005: The system shall provide a logout function accessible from all authenticated pages
- SESS-006: The system shall invalidate and delete session tokens upon logout
- SESS-007: The system shall redirect users to the login page when attempting to access protected resources without a valid session
- SESS-008: The system shall support only one active session per user at a time (logging in from a new device/browser invalidates previous sessions)
- SESS-009: The system shall display a warning message 2 minutes before automatic logout due to inactivity
- SESS-010: The system shall refresh session tokens automatically during active use to prevent expiration
- SESS-011: The system shall store session metadata including IP address, user agent, and creation time
- SESS-012: The system shall allow users to view and manage their active sessions from account settings

4. Non-Functional Requirements

Performance:

- The system shall respond to registration requests within 2 seconds under normal load (up to 100 concurrent users)
- The system shall respond to authentication requests within 1 second

- The system shall support at least 1000 concurrent users without performance degradation
- Database queries shall execute within 500 milliseconds
- Page load time shall not exceed 3 seconds on standard broadband connections
- The system shall handle at least 10,000 user registrations per day

Security:

- All passwords shall be hashed using bcrypt with a minimum cost factor of 10
- All communication between client and server shall be encrypted using TLS 1.2 or higher
- The system shall protect against SQL injection attacks through parameterized queries and input sanitization
- The system shall implement CSRF (Cross-Site Request Forgery) protection for all state-changing operations
- Session tokens shall be cryptographically secure random values with minimum 128-bit entropy
- The system shall implement secure HTTP headers (X-Frame-Options, X-Content-Type-Options, Content-Security-Policy)
- The system shall sanitize all user inputs to prevent XSS (Cross-Site Scripting) attacks
- Sensitive data shall never be logged or stored in plain text
- The system shall comply with OWASP Top 10 security guidelines

Usability:

- The user interface shall be intuitive and require minimal training for basic operations
- Error messages shall be clear, specific, and guide users toward resolution without exposing security vulnerabilities
- The system shall be accessible and comply with WCAG 2.1 Level AA standards
- Forms shall provide real-time validation feedback to users
- The interface shall be responsive and work on desktop, tablet, and mobile devices
- Help text and tooltips shall be available for complex form fields
- The system shall support multiple languages (English as default, with capability for localization)

Reliability:

- The system shall perform automated database backups daily at midnight
- The system shall recover from failures within 15 minutes with minimal data loss
- The system shall log all errors and exceptions with sufficient detail for debugging
- The system shall include health monitoring and alerting for critical components
- The system shall gracefully handle database connection failures with appropriate retry logic

Scalability:

- The system architecture shall support horizontal scaling to accommodate user growth
- The database schema shall efficiently handle up to 100,000 user records initially, with capability to scale to 1 million users
- The system shall support load balancing across multiple application servers
- Session data shall be stored in a distributed cache (Redis) to support scaling
- The system shall be designed with microservices architecture for independent scaling of components

Maintainability:

- The code shall follow established coding standards and best practices (ESLint for JavaScript, PEP 8 for Python)
- The system shall be modular to facilitate updates and feature additions
- Comprehensive technical documentation shall be provided for developers and administrators
- The code shall include inline comments and API documentation
- The system shall use version control (Git) with proper branching strategy
- Unit tests shall cover at least 80% of the codebase
- The system shall include automated deployment scripts and CI/CD pipeline configuration

Compatibility:

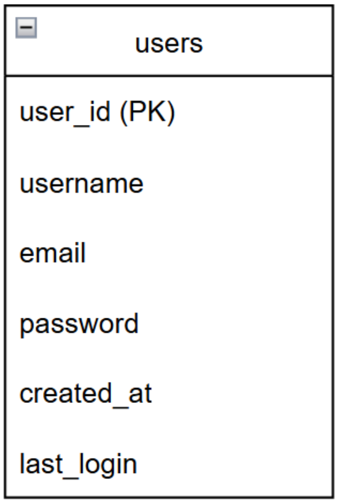
- The system shall work on all major web browsers (Chrome, Firefox, Safari, Edge) released within the last 2 years
- The system shall be compatible with both HTTP/1.1 and HTTP/2 protocols
- The system shall provide RESTful APIs that follow standard conventions for integration
- Database migrations shall be backward compatible to allow zero-downtime deployments

Availability:

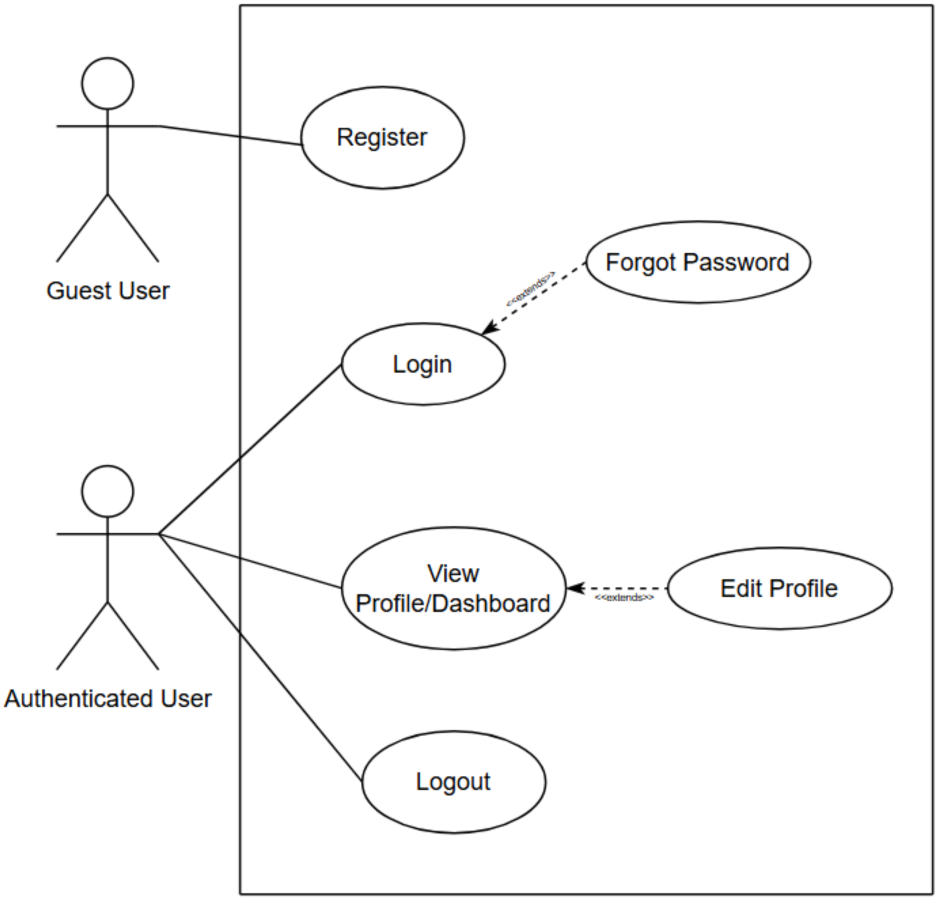
- The system shall be available 24/7 with planned maintenance windows communicated 48 hours in advance
- Planned maintenance shall not exceed 4 hours per month
- The system shall include redundancy for critical components (database, application servers)

5. System Models (Diagrams)

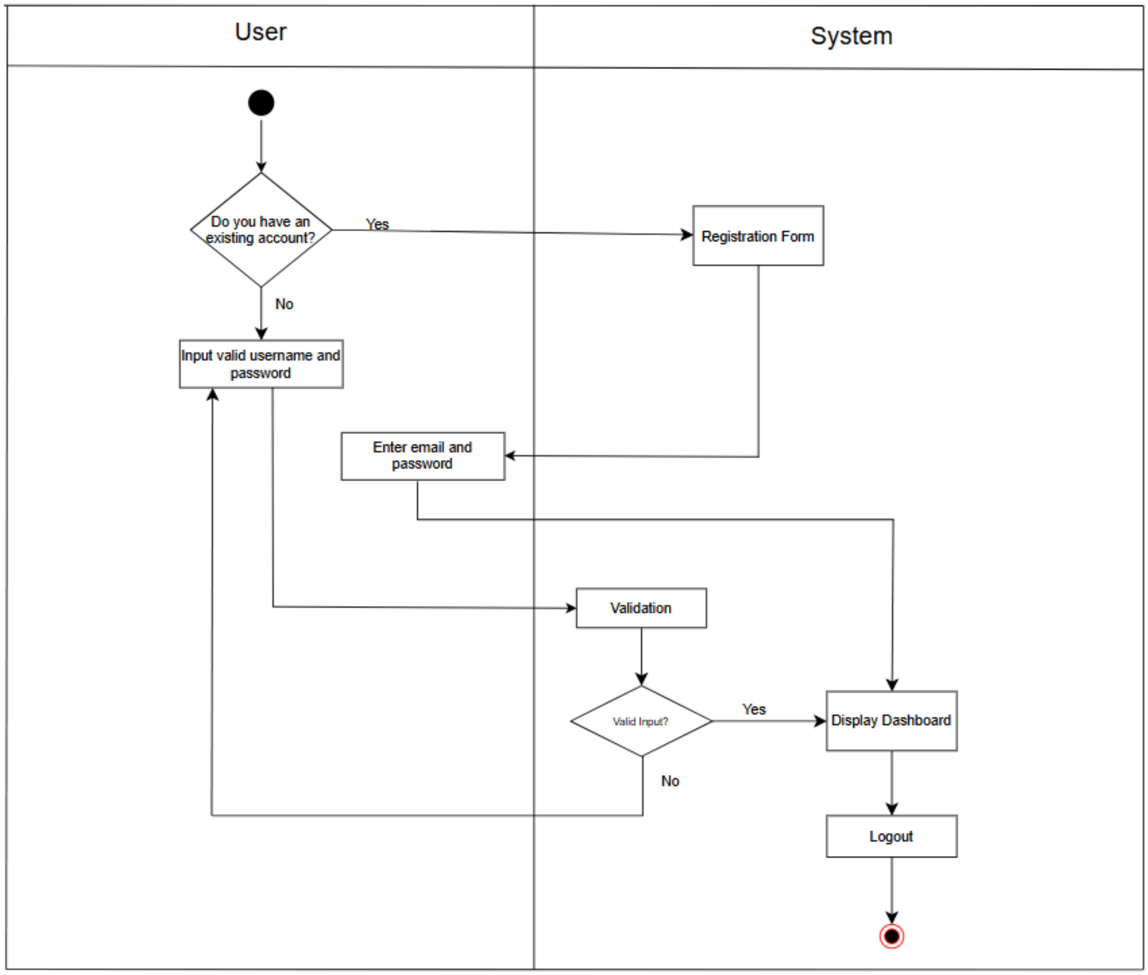
5.1. ERD



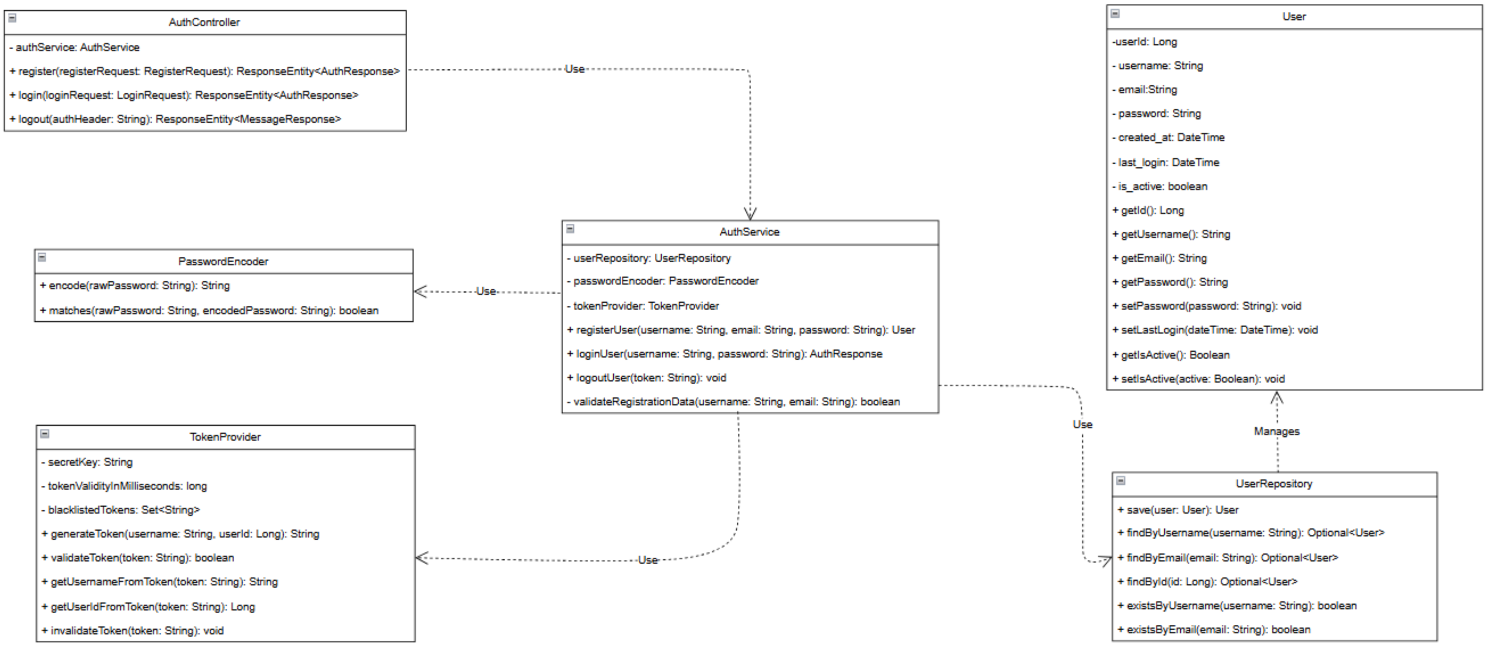
5.2. Use Case Diagram



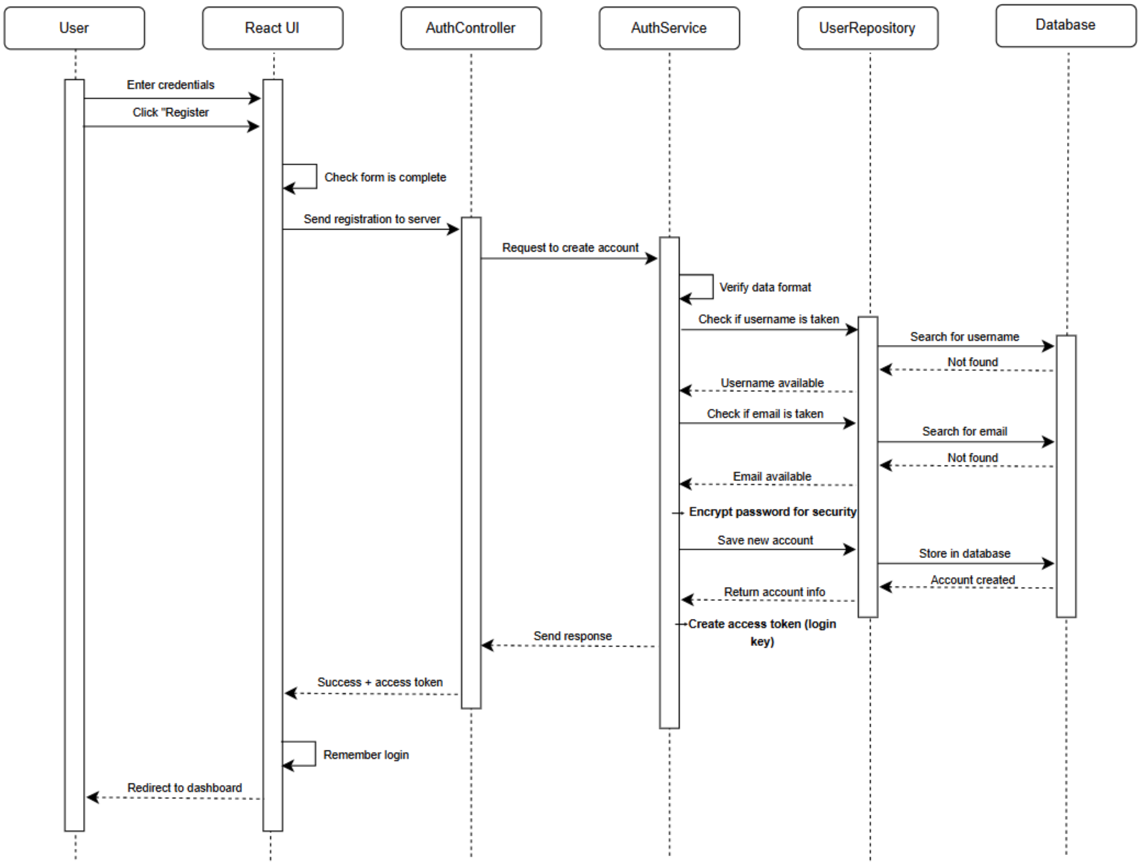
5.3. Activity Diagram



5.4. Class Diagram


AuthController
- authService: AuthService
+ register(registerRequest: RegisterRequest): ResponseEntity<AuthResponse>
+ login(loginRequest: LoginRequest): ResponseEntity<AuthResponse>
+ logout(authHeader: String): ResponseEntity<MessageResponse>
Use
AuthService
- userRepository: UserRepository
- passwordEncoder: PasswordEncoder
- tokenProvider: TokenProvider
+ registerUser(username: String, email: String, password: String): User
+ loginUser(username: String, password: String): AuthResponse
+ logoutUser(token: String): void
- validateRegistrationData(username: String, email: String): boolean
PasswordEncoder
+ encode(rawPassword: String): String
+ matches(rawPassword: String, encodedPassword: String): boolean
Use
TokenProvider
- secretKey: String
- tokenValidityInMilliseconds: long
- blacklistedTokens: Set<String>
+ generateToken(username: String, userId: Long): String
+ validateToken(token: String): boolean
+ getUsernameFromToken(token: String): String
+ getUserIdFromToken(token: String): Long
+ invalidateToken(token: String): void
Use
User
-userId: Long
- username: String
- email:String
- password: String
- created_at: DateTime
- last_login: DateTime
- is_active: boolean
+ getId(): Long
+ getUsername(): String
+ getEmail(): String
+ getPassword(): String
+ setPassword(password: String): void
+ setLastLogin(dateTime: DateTime): void
+ getIsActive(): Boolean
+ setIsActive(active: Boolean): void
Use
Manages
UserRepository
+ save(user: User): User
+ findByUsername(username: String): Optional<User>
+ findByEmail(email: String): Optional<User>
+ findById(id: Long): Optional<User>
+ existsByUsername(username: String): boolean
+ existsByEmail(email: String): boolean

5.5. Sequence Diagram


User
React UI
AuthController
AuthService
UserRepository
Database
Enter credentials
Click "Register
Check form is complete
Send registration to server
Request to create account
Verify data format
Check if username is taken
Search for username
Not found
Username available
Check if email is taken
Search for email
Not found
Email available
Encrypt password for security
Save new account
Store in database
Account created
Return account info
Create access token (login key)
Send response
Success + access token
Remember login
Redirect to dashboard