Contents

I Introduction

A Project Overview and Group Solution

With the increasing emphasis on collaborative learning and teamwork in university environments, the demand for efficient and user-friendly room booking systems has significantly grown. However, the existing campus room booking systems often provide only basic functionalities such as simple login, registration, and manual reservations. These systems lack advanced features like dynamic room filtering, detailed room information, and comprehensive administrative oversight, leading to inefficient resource utilization, reduced user satisfaction, and management difficulties.

To address these challenges, our group developed an online meeting room booking system named "***WeMeet***". This web-based application provides a secure, scalable, and intuitive platform tailored specifically for academic environments. The key functionalities implemented by our group include:

- User registration with email verification, secure login, and profile management.
- Room searching and filtering based on criteria such as capacity and IT facilities.
- User booking management for creating, updating, viewing, and canceling reservations.
- Role-based access control distinguishing between regular users and administrators.
- Administrator functions including room catalog management, booking review, user account control, and real-time monitoring dashboards.

The system was developed using a robust technology stack (see Table 1 in Appendix A), including ***Spring Boot*** for backend development, ***Thymeleaf*** for frontend templating, ***MySQL*** for data storage, and ***Docker*** for containerized deployment on ***Alibaba Cloud***. Through iterative development, regular team meetings, and structured testing, our group successfully delivered a modular, maintainable, and user-centric solution.

B Individual Contributions

Within this group project, my primary responsibility was designing and implementing the backend of a user authentication and management subsystem (in collaboration with another team member). My subsystem provided comprehensive backend functionalities including user registration, secure login, identity verification, and permission management. Specifically, my individual contributions included:

- **User Registration and Verification**: Developed secure user registration with email-based verification, username uniqueness checking, password strength validation, and avatar upload integrated with ***Alibaba Cloud OSS***.
- **Secure Login and Authentication**: Implemented ***JWT (JSON Web Token)***-based stateless authentication combined with ***Redis*** for centralized token management to enhance security and scalability.
- **Password Management**: Created secure password reset functionality with email verification for convenient account recovery.
- **Security Enhancements**: Integrated graphical **CAPTCHAs** to prevent brute-force attacks, **Role-Based Access Control (RBAC)** for fine-grained permission management, and secure password encryption techniques.
- **File Management**: Developed backend file upload and storage solutions for user avatars using ***Alibaba Cloud OSS***.
- **Frontend Test Pages**: Created several simple ***HTML*** pages as a basic frontend interface for testing and demonstrating backend functionalities (the formal frontend pages were developed by another team member specifically responsible for all frontend tasks).

Figure 1 illustrates the backend code structure of my subsystem. Throughout development, I followed best practices for security, maintainability, and scalability. My subsystem was thoroughly tested and integrated smoothly with teammates' modules, enhancing the reliability and usability of the final solution.

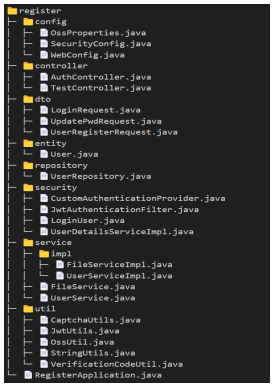


Figure 1: Backend Code Structure of the Subsystem

C Report Structure

The remainder of this report is structured as follows: **Section II** discusses how the Scrum methodology guided both

our team's and my individual software development process. **Section III** illustrates and justifies the software design by detailing the implementation of all PBIs in my individual work. **Section IV** discusses the change management strategies applied when handling project changes. **Section V** addresses the legal, social, ethical, and professional issues considered throughout the project. Finally, **Section VI** concludes the report by summarizing my achievements and reflecting on future improvements. **References** and **Appendix** sections are provided at the end of the report, with the Appendix containing supplementary materials such as PBIs, UML diagrams, relevant code snippets, and other supporting documentation.

II Software Development Process

A Scrum Methodology

Scrum is an agile software development methodology emphasizing iterative progress, incremental delivery, and continuous improvement [1]. It enables teams to adapt flexibly to changing requirements and deliver high-quality software efficiently.

In Scrum, clearly defined roles including **Product Owner**, **Scrum Master**, and **Development Team** collaborate closely through structured events such as **Sprint Planning** for task selection, **Daily Scrum** meetings for progress updates, **Sprint Review** for demonstrating completed work, and **Sprint Retrospective** for continuous improvement. All these activities occur within short iterative cycles called **Sprints**. Figure 2 clearly illustrates the Scrum framework process.

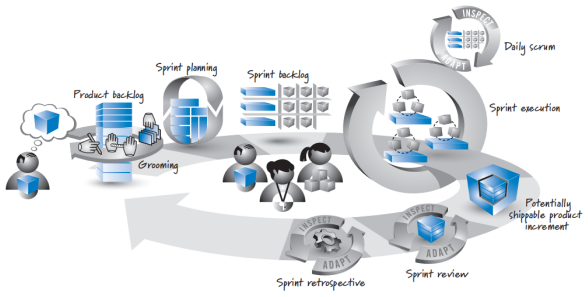


Figure 2: Scrum Framework Process

The main advantages of Scrum include increased flexibility to adapt to changing requirements, enhanced communication and collaboration within the team, improved transparency of project progress, and regular delivery of functional software increments.

B Group Implementation of Scrum

Our group strictly followed the Scrum framework, organizing our software development into three consecutive Sprints, each lasting two weeks. The detailed schedule is shown in Table 2.

Table 2: Two-Week Scrum Sprint Schedule

A Sprint		
	Week 1	**Week 2**
Mon	Sprint Planning + Daily Scrum	PBI Grooming + Daily Scrum
Tue	Daily Scrum	Daily Scrum
Wed	PBI Grooming + Daily Scrum	PBI Grooming + Daily Scrum
Thur	Daily Scrum	Daily Scrum
Fri	Daily Scrum	Daily Scrum + Sprint Review + Sprint Retrospective

Each Sprint begins with a Sprint Planning meeting on Monday of Week 1, clearly defining tasks and goals to align team members on objectives. Daily Scrum meetings held every weekday allow us to track progress and quickly identify and resolve issues collaboratively. Regular **Product Backlog Item (PBI) Grooming** sessions on both Wednesdays and Monday of Week 2 clarify upcoming tasks and priorities, reducing ambiguity and enhancing productivity.

On Friday of Week 2, the Sprint concludes with a Sprint Review, where completed features are demonstrated to stakeholders for timely feedback. Immediately afterward, the Sprint Retrospective provides our team an opportunity to reflect on challenges, discuss improvements, and further strengthen collaboration.

The structured implementation of Scrum events effectively facilitated our software development process by enhancing transparency, enabling rapid adaptation to changing requirements, and significantly improving communication and collaboration among team members, ultimately ensuring timely delivery of high-quality software increments.

C Individual Scrum Practices

In our group collaboration, each member defined their own PBIs based on their responsibilities. The PBIs I undertook specifically targeted my user authentication and management subsystem. The complete list of these PBIs is summarized in Table 3 in Appendix B.

During the implementation of these PBIs, I encountered several technical challenges. A representative example was the implementation of the PBI "**SP-01 Encrypt User Passwords Before Storing Them in the Database**".

Initially, I planned to use ***Base64*** encoding to store user passwords. However, during a Daily Scrum meeting in Sprint 3, I raised concerns about the security implications of this approach. ***Base64*** is merely a simple encoding method, easily reversible, and provides no genuine security protection for sensitive data such as passwords[2].

In response, our team held an immediate discussion during the Daily Scrum meeting. One team member pointed out that ***Base64*** encoding is not an encryption algorithm but rather a straightforward encoding scheme, making it vulnerable to reverse engineering attacks. Another teammate recommended adopting a dedicated password hashing algorithm such as ***BCrypt***, which incorporates salting and adaptive hashing rounds, significantly enhancing resistance to brute-force attacks[3].

Following this discussion, I conducted further research into security best practices and confirmed the suitability and security advantages of ***BCrypt***. In the subsequent Daily Scrum meeting, I presented my findings and proposed switching from ***Base64*** encoding to ***BCrypt*** hashing. After a brief discussion, our team quickly reached a consensus to adopt ***BCrypt*** for password encryption.

Through effective communication and rapid decision-making facilitated by our Daily Scrum meetings, our team promptly identified and resolved this security issue, significantly enhancing the application's security. This experience clearly demonstrated the value of Scrum's daily communication practices in efficiently addressing technical challenges and improving overall software quality.

III Software Design

A Individually Completed PBIs

As previously mentioned, the list of PBIs I completed is summarized in Table 3 in Appendix B. These PBIs were specifically designed to fulfill the requirements of the user authentication and management subsystem. To provide a clear and structured overview, the PBIs were grouped into five main functional categories:

- **User Registration**: Handle the creation of new user accounts, ensuring data validity and security. For example, checking username availability (UR-01) and verifying email addresses (UR-05).
- **User Login**: Focus on secure user authentication, such as verifying user credentials (UL-01) and preventing automated login attempts using CAPTCHA (UL-02).
- **Password Reset**: Address scenarios where users need to reset forgotten passwords, involving email verification (PR-01) and password updating (PR-02).
- **Email Verification**: Provide email-based verification functionality, such as sending verification codes (EV-01) and validating these codes (EV-02).
- **Security Protection**: Ensure sensitive user data is securely managed, for instance, encrypting user passwords before storage (SP-01).

Collectively, these PBIs ensure a comprehensive, secure, and user-friendly authentication and management subsystem.

B Technology Stack

As previously mentioned, our team adopted a robust and consistent technology stack (see Table 1 in Appendix A), such as ***Spring Boot***, ***Thymeleaf***, ***MySQL***, and ***Docker***. My subsystem was developed using technologies fully aligned with the team's overall stack. The detailed technology stack specifically used in my subsystem is summarized in Table 4.

Table 4: Technology Stack for My Subsystem

Category	**Technology / Tool**
Programming Language	Java
Backend Framework	Spring Boot
Security Framework	Spring Security
Authentication	JWT (JSON Web Token)
Database	MySQL
ORM	JPA (Hibernate)
Caching	Redis
Email Service	JavaMail
File Storage	Alibaba Cloud OSS
Build Tool	Maven
Version Control	Git
API Testing	Postman

C System Architecture

The system adopts a **Layered Architecture** as its overall structure. Within the presentation and business logic layers, the **Model-View-Controller (MVC) Pattern** is applied to further decouple data, user interface, and control logic. This combination enhances maintainability, scalability, and testability.

1 Layered Architecture

The project is organized into several main layers:

- **Presentation Layer**: Handles user interaction and HTTP requests/responses, including static web pages (e.g., `register.html` in `static/`) and RESTful controllers (in `controller/`). Controllers receive user input, validate it, and forward requests to the business logic layer.
- **Business Logic Layer**: Implements core business rules and processes through service interfaces and their implementations (in `service/` and

`service/impl/`). This layer processes data, coordinates business operations, and interacts with the persistence layer.

- **Persistence Layer**: Responsible for data access and storage, using repositories (in `repository/`) to interact with the database. For example, `UserRepository` provides CRUD operations for user entities.
- **Domain Layer**: Defines core business entities (in `entity/`), such as `User`, which are mapped to database tables and used throughout the application for data transfer and persistence.
- **Utility & Configuration Layer**: Provides supporting utilities (in `util/`, e.g., for string operations, JWT, OSS, captcha) and configuration classes (in `config/`, e.g., for security, CORS, OSS properties).

This layered structure ensures a clear separation of concerns and improves the maintainability of the system. Figure 3 in Appendix C illustrates the layered structure.

2 MVC Pattern

The **MVC Pattern** is a software architectural pattern that separates an application into three main logical components: the model, the view, and the controller [4]. Within the Presentation and Business Logic layers, the **MVC Pattern** is used to further structure the code:

- **Model**: Located in `entity/` and `dto/`, the model represents the application's data and business objects, such as `User` and various request/response DTOs. These classes define the data structure, validation rules, and are mapped to database tables via JPA annotations.
- **View**: Located in `static/` (e.g., `register.html`), the view provides the user interface for registration and other interactions. It presents forms and feedback to users, and sends requests to backend controllers via HTTP.
- **Controller**: Located in `controller/`, the controller handles HTTP requests, validates input, invokes business logic via services, and returns responses. It acts as the intermediary between the view and the model, ensuring user actions are processed correctly.

The use of the **MVC Pattern** further decouples data, interface, and control logic, resulting in a robust and extensible application structure. Figure 4 in Appendix C illustrates the **MVC Pattern** adopted in this project.

D Database Design

The overall database schema of the project is designed around several core entities, including `User`, `MeetingRoom`, `Booking`, `UserRole`, `Notification`, `BookingRejectLog`, and `AuditLog`. These entities are interconnected through clearly defined relationships, ensuring data integrity, scalability, and efficient querying. Figure 5 in Appendix D provides a high-level overview of the database schema and entity relationships.

In my subsystem, the primary database table involved is the `users` table. This table stores essential user information required for authentication, authorization, and profile management. Below is the ***SQL*** definition for the `users` table:

```sql
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    avatar VARCHAR(255),
    role ENUM('Admin', 'User') NOT NULL,
    is_locked BOOLEAN DEFAULT FALSE,
    email_verified BOOLEAN DEFAULT FALSE,
    lock_until DATETIME DEFAULT NULL,
    failed_attempts INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT
    CURRENT_TIMESTAMP,
    last_login_at TIMESTAMP NULL,
    last_password_change_at TIMESTAMP NULL
);
```

The design of the `users` table ensures secure and efficient user management, supporting critical features such as authentication, role-based access control, account security, and profile customization.

E Essential PBI 1: UL-02 Verify Graphical Verification Code (CAPTCHA) During Login

1 Logical Flow with Key Codes

This PBI is implemented to prevent automated login attempts and enhance security. The logical flow is as follows:

(1) **Frontend Requests CAPTCHA**: When the login page is loaded, the frontend sends a GET request to `/allowlist/captchaImage` to obtain a CAPTCHA image and a unique key.

(2) **Backend Generates CAPTCHA**: The backend uses the `CaptchaUtils` component to generate a random 4-character alphanumeric code. This code is rendered as an image with noise lines for security, using **Java's Graphics API**. The code and a unique key are stored in ***Redis*** with a 5-minute expiration.

```java
// AuthController.java
@GetMapping("/captchaImage")
public ResponseEntity<?> getCaptchaImage() {
    Map<String, Object> captchaMap =
    captchaUtils.generateCaptcha();
    BufferedImage image = (BufferedImage)
    captchaMap.get("image");
    String captchaKey = (String) captchaMap.
    get("key");
    // ...convert image to Base64 and return
    with key...
```

```
}
```

```
// CaptchaUtils.java
public Map<String, Object> generateCaptcha()
    {
    String captchaCode = generateRandomCode()
    ;
    String captchaKey = UUID.randomUUID().
    toString();
    redisTemplate.opsForValue().set("captcha:
    " + captchaKey, captchaCode,
    CAPTCHA_EXPIRATION, TimeUnit.MINUTES);
    // ...draw image...
}
```

(3) **Frontend Displays CAPTCHA**: The backend returns the image (as a ***Base64*** string) and the key. The frontend displays the CAPTCHA image and provides an input field for the user to enter the code.

(4) **User Submits Login Form**: The user enters their username, password, and the CAPTCHA code, then submits the form. The frontend sends these, along with the CAPTCHA key, to the backend.

(5) **Backend Validates CAPTCHA and Feedback**: The backend retrieves the code from ***Redis*** using the key and compares it with the user input. If the code matches and is not expired, the backend deletes the code from ***Redis*** and proceeds with authentication. Otherwise, an error is returned.

```
// AuthController.java
@PostMapping("/login")
public ResponseEntity<?> login(@Valid
    @RequestBody LoginRequest loginRequest) {
    if (!captchaUtils.validateCaptcha(
    loginRequest.getCaptchaKey(),
    loginRequest.getCaptchaCode())) {
        return ResponseEntity.badRequest().
    body(Map.of("code","0","message", "The
    verification code is incorrect or has
    expired."));
    }
    // ...proceed with authentication...
}
```

```
// CaptchaUtils.java
public boolean validateCaptcha(String key,
    String code) {
    String storedCode = redisTemplate.
    opsForValue().get("captcha:" + key);
    if (storedCode != null && storedCode.
    equalsIgnoreCase(code)) {
        redisTemplate.delete("captcha:" + key
    );
        return true;
    }
    return false;
}
```

Figure 6 in Appendix E shows the sequence diagram for this process. Furthermore, the complete code for generating CAPTCHA can be seen in the `CaptchaUtils.java` file in Appendix F.

2 Database Support

The database design for this PBI leverages ***Redis*** as a temporary store for CAPTCHA codes. When a CAPTCHA is generated, it is stored in ***Redis*** with a unique key and a short expiration time (e.g., 5 minutes). This approach ensures that the main user database remains unaffected, and the stateless nature of ***Redis*** allows for scalable and efficient verification.

3 Object States and State Transitions

Several key objects are involved in this PBI, each with distinct states. For CAPTCHA code, the entire process has the following states:

- `Created`: When the frontend requests a CAPTCHA, a code is generated and stored in ***Redis***.
- `Validated`: Upon user submission, the backend checks the code. If correct, the code is deleted from ***Redis*** (single-use).
- `Expired`: If not used within the expiration time, ***Redis*** automatically removes the code.

The corresponding state machine diagram is illustrated clearly in Figure 7 in Appendix E.

For user login, the entire process has the following states:

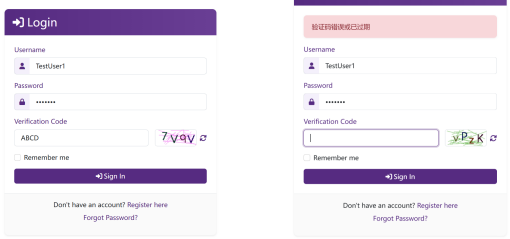
- `Awaiting_CAPTCHA`: The user cannot proceed without solving the CAPTCHA.
- `CAPTCHA_Validated`: The user is allowed to proceed with authentication.
- `CAPTCHA_Invalid_or_Expired`: The user must retry with a new CAPTCHA.

The corresponding state machine diagram is illustrated clearly in Figure 8 in Appendix E. These state transitions ensure that each CAPTCHA is single-use and time-limited, directly affecting user behavior and system security.

4 UI Design

The UI is designed to clearly present the CAPTCHA image and provide an input field for the code. If the user enters an incorrect or expired CAPTCHA, the UI displays an error message and prompts the user to try again. The interface also allows users to refresh the CAPTCHA if needed, ensuring a smooth and user-friendly experience while maintaining security requirements.Figure 9 demonstrates the UI disign of the CAPTCHA.

By integrating these UI elements, the system effectively guides users through the verification process, reducing confusion and supporting the successful delivery of this PBI.



(a) Enter the Code (b) Retry the Code

Figure 9: UI Design for the CAPTCHA

F Essential PBI 2: SP-01 Encrypt User Passwords Before Storing Them in the Database

1 Logical Flow with Key Codes

This PBI is implemented to ensure that user passwords are never stored in plaintext in the database, thereby protecting user privacy and enhancing system security. The logical flow is as follows:

(1) **User Submits Registration Form**: The user fills in the registration form with a username, password, email, and other required information, then submits the form to the backend.

(2) **Backend Receives and Validates Data**: The backend receives the registration request and validates the input fields, including checking the password format and uniqueness of the username/email.

(3) **Password Encryption**: Before saving the user data, the backend uses the `BCryptPasswordEncoder` to hash the plaintext password. This ensures that the password is never stored or transmitted in plaintext.

```
// SecurityConfig.java
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

```
// UserServiceImpl.java
@Autowired
private PasswordEncoder passwordEncoder;

@Override
public ResponseEntity<?> register(User user)
    {
    // ...validation...
    user.setPassword(passwordEncoder.encode(
    user.getPassword()));
    // ...save user to database...
}
```

(4) **Store Encrypted Password in Database**: The encrypted (hashed) password is stored in the `users` table in the database, replacing the original plaintext password.

(5) **Password Verification During Login**: When a user attempts to log in, the backend retrieves the hashed password from the database and uses `BCryptPasswordEncoder.matches()` to verify the submitted password against the stored hash.

```
// CustomAuthenticationProvider.java
@Override
protected void additionalAuthenticationChecks
    (UserDetails userDetails,
        UsernamePasswordAuthenticationToken
    authentication) throws
    AuthenticationException {
    String presentedPassword = authentication
    .getCredentials().toString();
    if (!this.getPasswordEncoder().matches(
    presentedPassword, userDetails.
    getPassword())) {
        throw new BadCredentialsException("
    Username or password is incorrect");
    }
}
```

Figure 10 in Appendix E shows the sequence diagram for this process. Furthermore, the complete code for ***BCrypt*** of the password can be seen in the `BCryptPasswordEncoder.java` file in Appendix F.

2 Database Support

The users table in the database only stores the hashed version of the password. For example, when registering, suppose the user's password is `"myPassword"`. After hashing with ***BCrypt***, the `password` field will contain a string like `"$2a$10..."`, as shown in Figure 11.This design ensures that even if the database is compromised, attackers cannot obtain users' plaintext passwords.

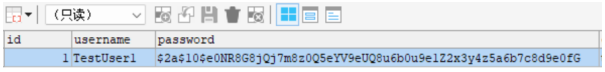


Figure 11: Encrypted Password Stored in the Database

3 Object States and State Transitions

The password object is involved in this PBI, with the following distinct states:

- `Plaintext`: When the user submits the registration or password reset form, the password is initially in plaintext in memory.
- `Hashed`: Before being saved to the database, the password is hashed using ***BCrypt*** and only the hashed value is stored.
- `Verified`: During login, the submitted plaintext password is compared with the stored hash using ***BCrypt***'s verification function.

The corresponding state machine diagram is illustrated clearly in Figure 12 in Appendix E.

IV Change Management

A Managing Requirement Changes in Scrum

Scrum provides a flexible framework designed to effectively accommodate requirement changes throughout the project lifecycle[5]. In practice, the **Product Owner** is responsible for regularly updating and prioritizing the **Product Backlog** to reflect new or revised requirements. When requirement changes arise, the Scrum team collaboratively evaluates their impact and feasibility during **PBI Grooming** sessions. Typically, these changes are not immediately included in the ongoing **Sprint** to avoid disrupting team productivity. Instead, the team integrates accepted changes into future **Sprint Planning** meetings, ensuring smooth adaptation while maintaining steady progress.

B Examples of Requirement Changes

During my development process, I encountered several cases where requirements needed adjustments. The previously mentioned "**SP-01**", where I abandoned ***Base64*** encoding in favor of ***BCrypt*** encryption for user passwords, is a typical example. Below are two additional representative examples:

1 UR-06 Allow Users to Upload Avatar Pictures

Initially, I implemented a straightforward feature allowing users to upload any image as their avatar. However, during a Daily Scrum meeting in Sprint 2, another developer pointed out that users might upload images with varying sizes and aspect ratios, potentially causing distorted or inconsistent profile displays. After discussing and analyzing this issue together, we concluded that adding an image cropping functionality would effectively solve this problem, as shown in Figure 13. Therefore, I adjusted the implementation to allow users to crop their avatars into a standardized square format, ensuring consistent visual presentation across user profiles.

2 UR-09 Check Administrator Secret Key on Administrator Registration

When defining this requirement in Sprint 3, administrator registration was designed to require a secret key. At first, I considered generating multiple unique secret keys, each valid for a single use. However, during the PBI Grooming session in Sprint 3, our group realized that generating multiple single-use secret keys would require an additional database table to store these keys, an extra field to track their usage status, and regular maintenance to clear expired or used keys. To avoid this unnecessary overhead, our group decided to implement a single universal administrator secret key, simplifying database design, reducing backend complexity, and streamlining the registration process.

Figure 13: UI Interface for the Avatar Cropping Function

```java
// AuthController.java
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody User user) {
    String role = user.getRole();
    String adminKey = user.getAdminKey();
    // If the role is an administrator, verify the unique key "Cpt202Group50"
    if ("admin".equalsIgnoreCase(role) && !"Cpt202Group50".equals(adminKey)) {
        return ResponseEntity.ok().body(Map.of("code","0","message", "The administrator key is incorrect. Please re-enter it."));
    }
}
```

V Legal, Social, Ethical and Professional Issues

A Legal Issues

- **Issue 1 - Data Protection and Privacy**: During user registration, our system collects users' personal information, such as email addresses. There is a risk that improper handling of this data could violate the ***PIPL (Personal Information Protection Law)***[6].
- **Solution**: We only collect necessary information for registration. Passwords are encrypted before storage, and sensitive data is transmitted over secure channels. Users are informed about data collection and must provide explicit consent.
- **Issue 2 - Use of Third-party Libraries**: Our project uses open-source libraries such as ***Spring Boot*** for backend development. There is a risk of violating software license agreements if these libraries are used improperly.

- **Solution**: We review all third-party library licenses, such as ***Apache 2.0***[7], and provide proper attribution in our documentation to ensure compliance.

B Social Issues

- **Issue 1- Accessibility and Inclusivity**: Some users such as those with disabilities, may find it difficult to complete the registration process, especially when uploading avatars. This could reduce the usability for certain groups.

- **Solution**: We design the registration interface to be simple and accessible, with clear instructions and error messages. We avoid collecting unnecessary information to lower the barrier for all users.

- **Issue 2 - User Trust**: If users feel uncertain about how their email addresses and passwords are stored, or if they perceive the system as insecure, this could erode user trust and discourage participation.

- **Solution**: We publish a transparent privacy policy explaining how user data is used and protected. Security measures such as email verification and strong password requirements are implemented to build user trust.

C Ethical Issues

- **Issue 1 - Responsible Data Handling**: If user data is mishandled or accessed by unauthorized personnel, it could violate ethical standards and user trust, as outlined in the ***ACM (Association for Computing Machinery) Code of Ethics***[8].

- **Solution**: We restrict access to personal data to only authorized administrators. Regular audits are conducted to ensure ethical handling of all user data.

- **Issue 2 - Prevention of Misuse**: This system may be misused by malicious users, such as brute-force login attempts, causing harm to legitimate users and affecting system security.

- **Solution**: We implemented CAPTCHA verification and activity monitoring to detect and prevent suspicious behavior, ensuring user safety and system reliability.

D Professional Issues

- **Issue 1 - Secure Coding Practices**: If user input is not properly validated, it could introduce security vulnerabilities like injection attacks. This could compromise the professionalism and reliability of the system.

- **Solution**: We follow secure coding standards, validate all user input, and conduct regular code reviews and automated testing to ensure code quality and security.

- **Issue 2 - Documentation**: Missing critical documentation, such as API documentation, can cause difficulties in future maintenance and auditing.

- **Solution**: We maintain comprehensive documentation for all system components, including API usage (see Appendix G), data flow, and security practices.

VI Conclusion

A Reflections

Throughout this project, I have gained a wealth of valuable experience in both software engineering and team collaboration.

From a software engineering perspective, I learned how to conduct effective requirements analysis, break down complex functionalities into manageable modules, and design robust backend systems. Implementing features such as user registration and email verification gave me hands-on experience with real-world development challenges. I also became more familiar with best practices in code organization, and compliance with legal and ethical standards.

From a teamwork perspective, practicing Scrum methodology was especially rewarding. By dividing our work into Sprints and holding regular meetings, our team maintained clear communication and adapted quickly to changes. This agile approach improved our productivity, and ensured everyone contributed to the project's success. Working closely with teammates also taught me the value of mutual support, open communication, and collaborative problem-solving.

B Future Improvements

Looking ahead, there are several ways I can improve my performance in future projects.

From a software engineering perspective, I plan to strengthen my project management skills, such as setting clearer milestones, managing risks proactively, and ensuring better alignment between requirements and implementation. I will also focus on improving time estimation and prioritization to deliver features more efficiently.

Regarding teamwork, I aim to foster a more collaborative and supportive team environment. This includes encouraging open feedback, sharing knowledge regularly, and taking initiative in resolving conflicts or challenges. I will also work on enhancing team coordination through clearer role definition and more effective use of Scrum practices.

References

[1] K. S. Rubin, *Essential Scrum: A practical guide to the most popular agile process.* Upper Saddle River, Nj: Addison-Wesley, 2012.

[2] R. Rahim, S. Sumarno, M. T. Multazam, S. Thamrin, and S. H. Sumantri, "Combination Base64 and GOST algorithm for security process," *Journal of Physics: Conference Series*, vol. 1402, p. 066054, Dec. 2019, doi: https://doi.org/10.1088/1742-6596/1402/6/066054.

[3] J. B. Alimpia, "An Enhanced Hash-based Message Authentication Code using BCrypt," *International Journal for Research in Applied Science and Engineering Technology*, vol. 6, no. 4, pp. 1429–1432, Apr. 2018, doi: https://doi.org/10.22214/ijraset.2018.4240.

[4] R. S. Pressman and B. R. Maxim, *Software engineering : a practitioner's approach.* New York: Mcgraw-Hill, 2015.

[5] A. Pham and Phuong Van Pham, *Scrum in action : agile software project management and development.* Boston: Course Technology Ptr, 2012.

[6] *Personal Information Protection Law of the People's Republic of China*, Standing Committee of the National People's Congress, Aug. 20, 2021.

[7] Apache Software Foundation, "Apache License Version 2.0," Jan. 2004. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed: May 12, 2025].

[8] Association for Computing Machinery (ACM), "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: May 12, 2025].

Appendix

Appendix A: Project Technology Stack

Table 1: Technology Stack of WeMeet System

Technology/Framework/Tool	Description/Usage	Version
Java	Primary programming language for backend development	17
HTML5	Page structure and semantic markup	-
CSS3	Styling and layout design	-
JavaScript (ES6+)	Dynamic interactions and business logic	-
jQuery	DOM manipulation and AJAX request simplification	-
Bootstrap	Responsive UI components and grid system	5
Font Awesome	Icon library for visual elements	-
Thymeleaf	Server-side templating engine with Spring Security integration	(Spring Boot)
Fetch API	Native browser API for AJAX communication	-
Spring Boot	Core application framework with embedded Tomcat	3.4.4
Spring Security	Authentication and authorization framework	3.4.4
Spring Data JPA	ORM implementation with Hibernate	3.4.4
JJWT	JWT token generation and validation library	0.11.5
Spring Mail	Email notification service implementation	3.4.4
Bean Validation	Input data validation framework	3.4.4
MySQL	Primary relational database (Alibaba Cloud RDS)	8.0.41
Redis	Distributed caching for sessions/hot data	7.0.15
HikariCP	High-performance JDBC connection pool	(Default)
Alibaba Cloud ECS	Elastic Compute Service for hosting	-
Alibaba Cloud RDS	Managed MySQL database service	-
Alibaba Cloud OSS	Object storage for user-uploaded files	3.17.4
Alibaba Cloud SLB	Server Load Balancer for traffic distribution	-
Docker	Containerization platform for deployment	-
Docker Compose	Multi-container orchestration tool	-
Maven	Build automation and dependency management	-
Git	Version control system	-
Postman	API testing and documentation tool	-
JUnit 5	Unit testing framework	-
Mockito	Mocking framework for isolated testing	-
Lombok	Code reduction through annotation processing	1.18.24
Spring Boot DevTools	Hot reloading for development efficiency	3.4.4
Apache Commons IO	File handling utilities	2.14.0
OpenCSV	CSV file export implementation	5.7.1
BCrypt	Password hashing algorithm	(Spring Security)

Appendix B: Individually Completed PBIs

Table 3: Individually Completed Product Backlog Items

PBI ID	Description	Sprint
User Registration		
UR-01	Check if username is already taken	1
UR-02	Check password strength	1
UR-03	Confirm password matches the original password	1
UR-04	Validate email address format	1
UR-05	Verify user's email address using email verification (see EV module)	2
UR-06	Allow users to upload avatar pictures	2
UR-07	Provide a default profile picture if none uploaded	3
UR-08	Allow users to select their role (regular user or administrator)	2
UR-09	Check administrator secret key on administrator registration	3
User Login		
UL-01	Verify username and password during login	1
UL-02	Verify graphical verification code (CAPTCHA) during login	2
UL-03	Lock account after multiple failed login attempts	3
Password Reset		
PR-01	Verify user's email address using email verification (see EV module)	3
PR-02	Allow users to reset their password after successful email verification	3
Email Verification		
EV-01	Send verification code to user's email address	1
EV-02	Check if the entered email verification code is correct	1
EV-03	Set expiration time for email verification codes	2
Security Protection		
SP-01	Encrypt user passwords before storing them in the database	3

Appendix C: System Architecture Diagrams

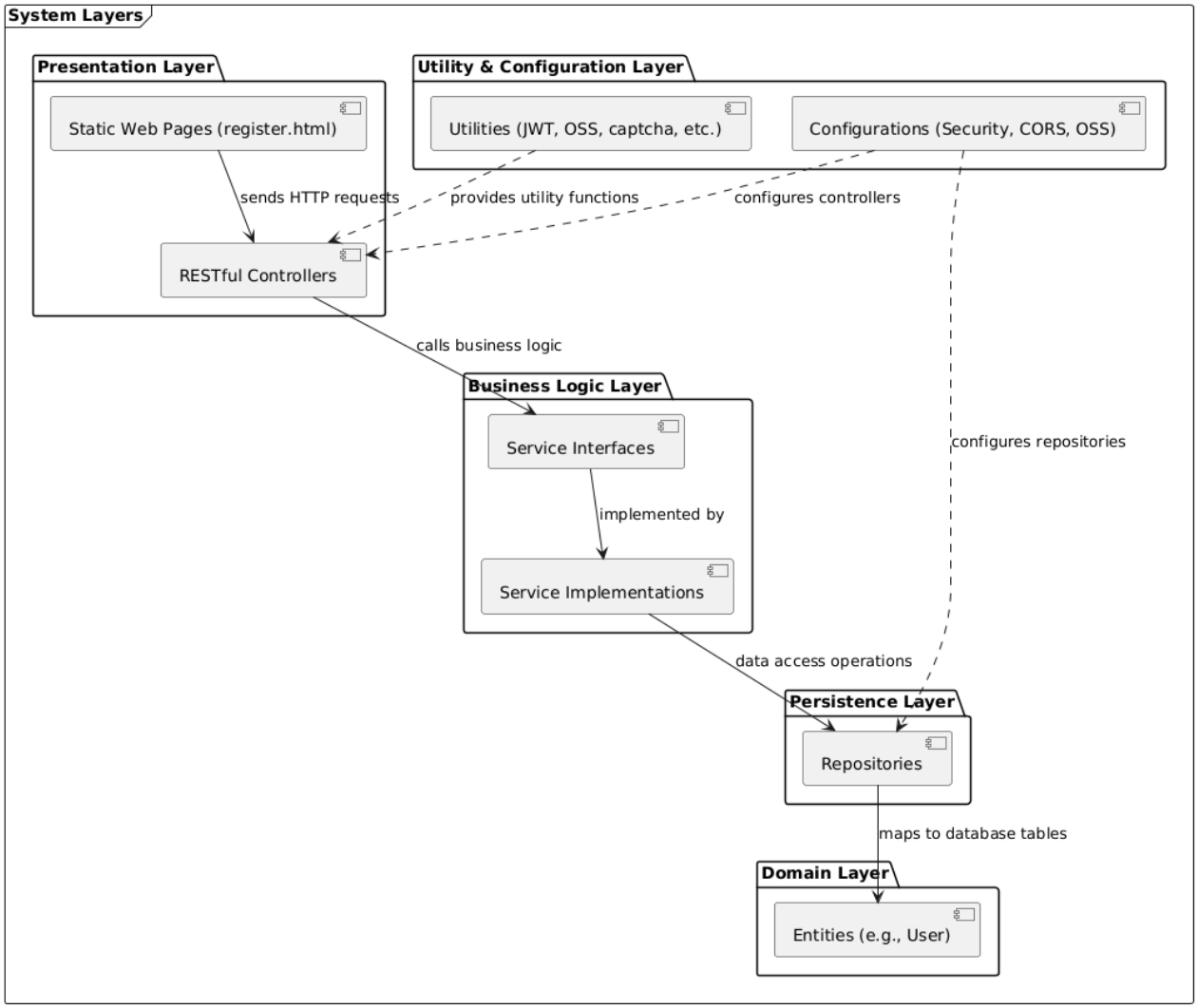


Figure 3: System Layered Architecture

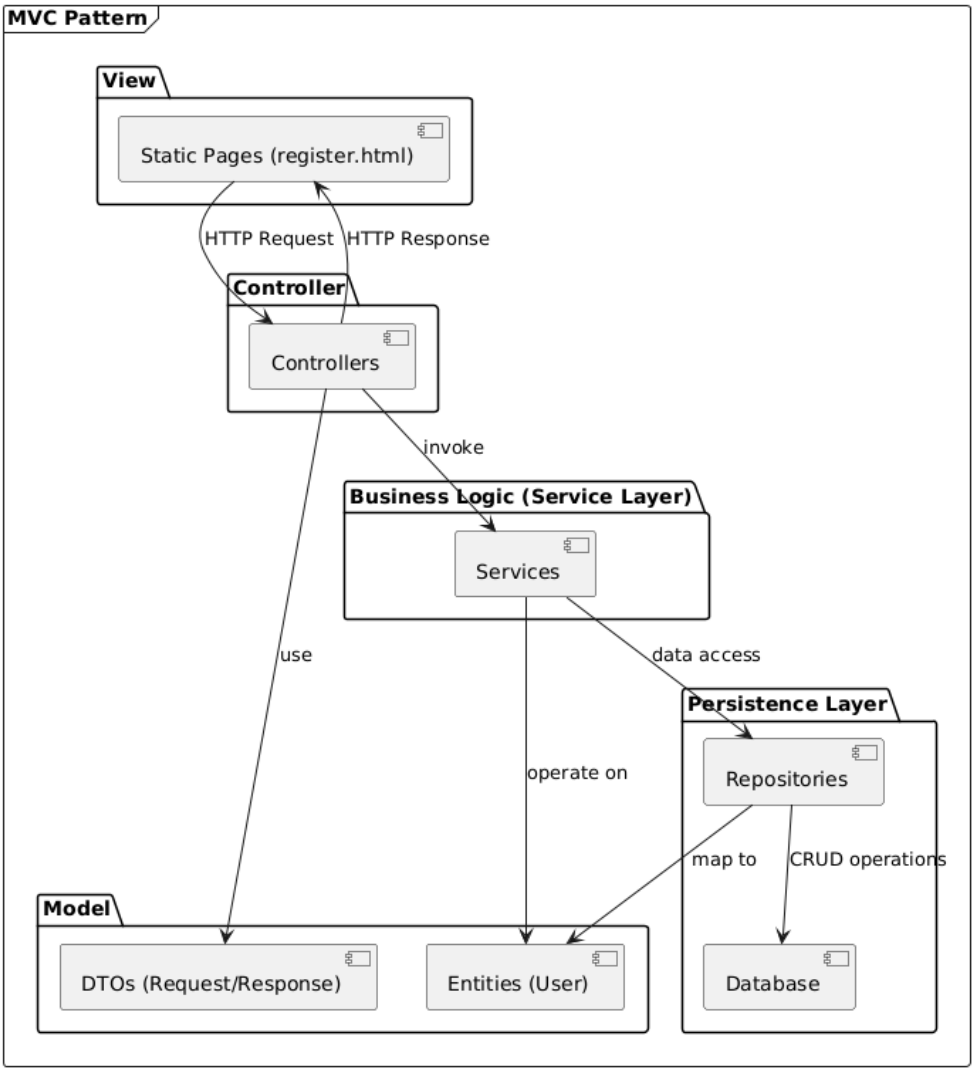


Figure 4: System MVC Pattern

Appendix D: Entity-Relationship Diagram

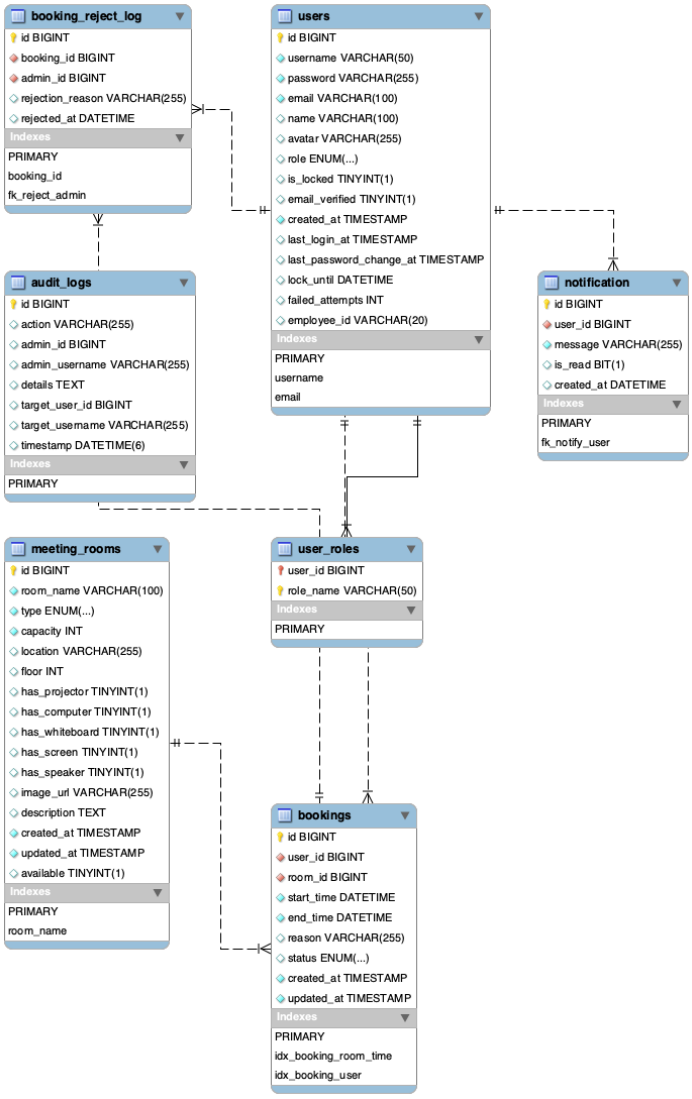


Figure 5: ER Diagram for Database of WeMeet System

Appendix E: UML Diagrams

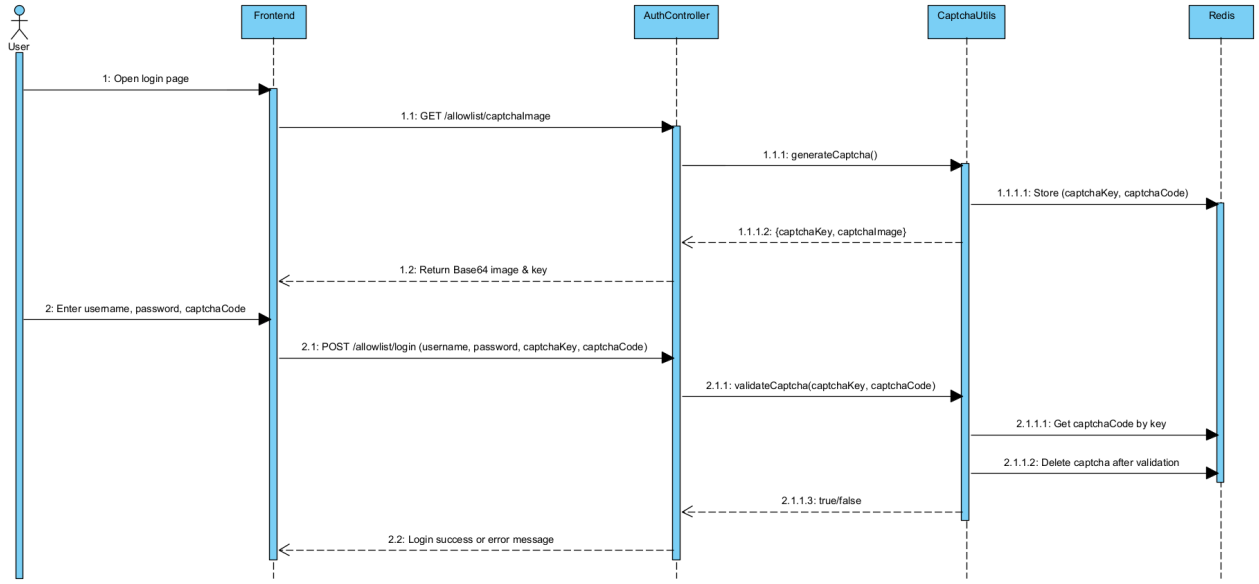


Figure 6: Sequence Diagram for PBI UL-02 (Verify Graphical Verification Code (CAPTCHA) During Login)

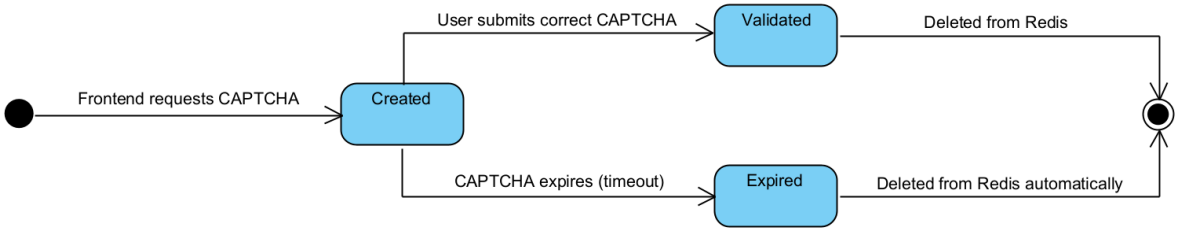


Figure 7: State Machine Diagram for CAPTCHA Code State

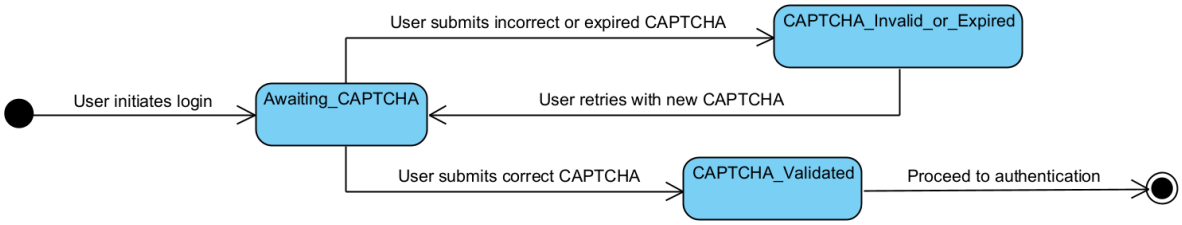


Figure 8: State Machine Diagram for User Login State

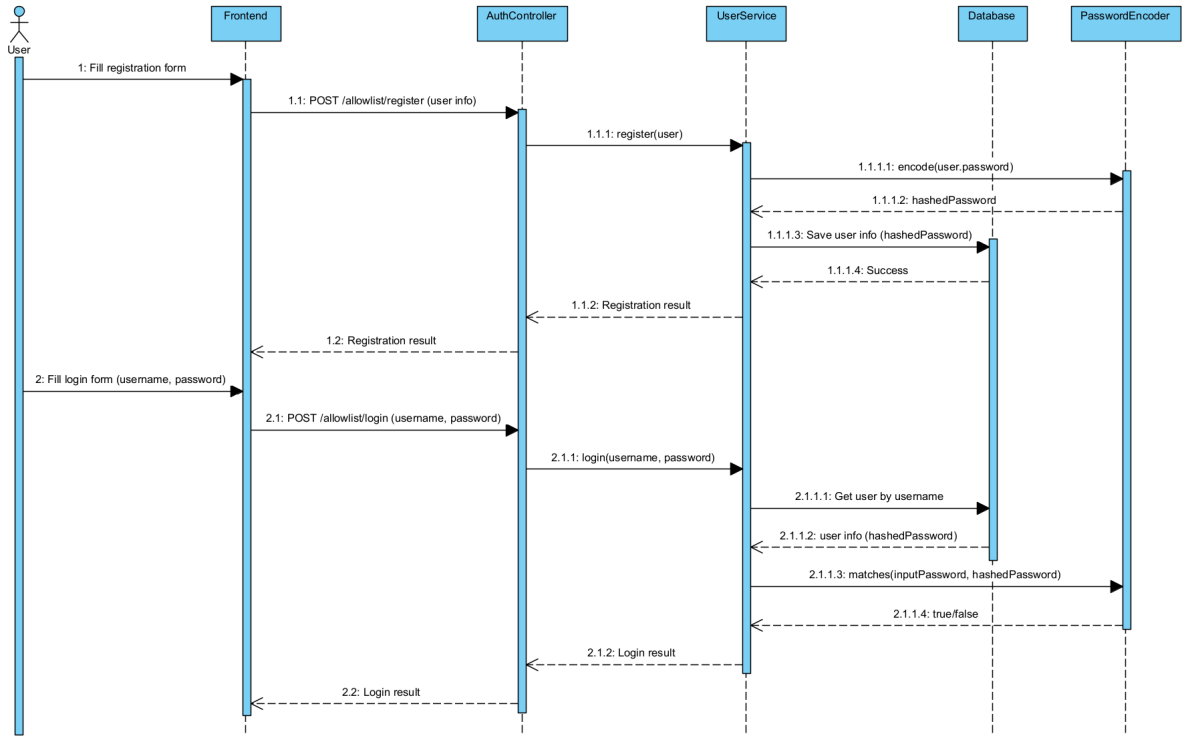


Figure 10: Sequence Diagram for PBI SP-01 (Encrypt User Passwords Before Storing Them in the Database)

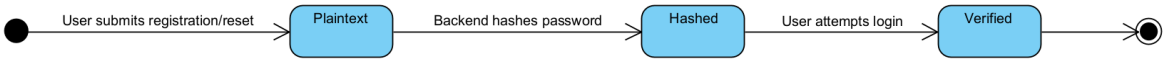


Figure 12: State Machine Diagram for Password State

Appendix F: Source Code (Part)

```java
// CaptchaUtils.java

package com.example.register.util;

import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.UUID;
import java.util.concurrent.TimeUnit;

@Component
public class CaptchaUtils {

    private final RedisTemplate<String, String> redisTemplate;
    private static final int CAPTCHA_EXPIRATION = 5; // Captcha expiration time (minutes)

    public CaptchaUtils(RedisTemplate<String, String> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    // Generates a captcha image and stores the captcha code in Redis
    public Map<String, Object> generateCaptcha() {
        int width = 100;
        int height = 40;
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        Graphics g = image.getGraphics();

        // Set background color
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, width, height);

        // Generate random captcha code
        String captchaCode = generateRandomCode();
        String captchaKey = UUID.randomUUID().toString();

        // Store captcha code in Redis with expiration
        redisTemplate.opsForValue().set(
            "captcha:" + captchaKey,
            captchaCode,
            CAPTCHA_EXPIRATION,
            TimeUnit.MINUTES
        );

        // Draw captcha code onto the image
        g.setColor(Color.BLACK);
        g.setFont(new Font("Arial", Font.BOLD, 20));
        g.drawString(captchaCode, 20, 25);

        // Add random interference lines to the image
        Random random = new Random();
        for (int i = 0; i < 4; i++) {
            g.setColor(new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)));
            g.drawLine(random.nextInt(width), random.nextInt(height),
                       random.nextInt(width), random.nextInt(height));
        }

        // Prepare the result map containing captcha key and image
        Map<String, Object> result = new HashMap<>();
        result.put("key", captchaKey);
        result.put("image", image);

        return result;
    }

    // Validates the provided captcha code against the stored value in Redis
```

```
    public boolean validateCaptcha(String key, String code) {
        if (key == null || code == null) {
            return false;
        }

        // Retrieve stored captcha code from Redis
        String storedCode = redisTemplate.opsForValue().get("captcha:" + key);

        // Validate and delete captcha code if correct
        if (storedCode != null && storedCode.equalsIgnoreCase(code)) {
            redisTemplate.delete("captcha:" + key);
            return true;
        }
        return false;
    }

    // Generates a random 4-character alphanumeric captcha code.
    private String generateRandomCode() {
        Random random = new Random();
        StringBuilder sb = new StringBuilder();
        String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

        for (int i = 0; i < 4; i++) {
            sb.append(chars.charAt(random.nextInt(chars.length())));
        }

        return sb.toString();
    }
}
```

```
// BCryptPasswordEncoder.java

package org.springframework.security.crypto.bcrypt;

import java.security.SecureRandom;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.security.crypto.password.PasswordEncoder;

// Implementation of PasswordEncoder that uses the BCrypt hashing algorithm
public class BCryptPasswordEncoder implements PasswordEncoder {
   private Pattern BCRYPT_PATTERN;
   private final Log logger;
   private final int strength;
   private final BCryptVersion version;
   private final SecureRandom random;

   // Constructors
   public BCryptPasswordEncoder() {
      this(-1);
   }

   public BCryptPasswordEncoder(int strength) {
      this(strength, (SecureRandom)null);
   }

   public BCryptPasswordEncoder(BCryptVersion version) {
      this(version, (SecureRandom)null);
   }

   public BCryptPasswordEncoder(BCryptVersion version, SecureRandom random) {
      this(version, -1, random);
   }

   public BCryptPasswordEncoder(int strength, SecureRandom random) {
      this(org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder.BCryptVersion.$2A, strength
    , random);
   }

```

```
41     public BCryptPasswordEncoder(BCryptVersion version, int strength) {
42        this(version, strength, (SecureRandom)null);
43     }
44
45     public BCryptPasswordEncoder(BCryptVersion version, int strength, SecureRandom random) {
46        this.BCRYPT_PATTERN = Pattern.compile("\\A\\$2(a|y|b)?\\$(\\d\\d)\\$[./0-9A-Za-z]{53}");
47        this.logger = LogFactory.getLog(this.getClass());
48        if (strength == -1 || strength >= 4 && strength <= 31) {
49           this.version = version;
50           this.strength = strength == -1 ? 10 : strength;
51           this.random = random;
52        } else {
53           throw new IllegalArgumentException("Bad strength");
54        }
55     }
56
57     // Encodes the raw password using BCrypt hashing algorithm
58     public String encode(CharSequence rawPassword) {
59        if (rawPassword == null) {
60           throw new IllegalArgumentException("rawPassword cannot be null");
61        } else {
62           String salt = this.getSalt();
63           return BCrypt.hashpw(rawPassword.toString(), salt);
64        }
65     }
66
67     // Generates a salt string for BCrypt hashing
68     private String getSalt() {
69        return this.random != null ? BCrypt.gensalt(this.version.getVersion(), this.strength, this.random
       ) : BCrypt.gensalt(this.version.getVersion(), this.strength);
70     }
71
72     // Verifies if the raw password matches the encoded password
73     public boolean matches(CharSequence rawPassword, String encodedPassword) {
74        if (rawPassword == null) {
75           throw new IllegalArgumentException("rawPassword cannot be null");
76        } else if (encodedPassword != null && encodedPassword.length() != 0) {
77           if (!this.BCRYPT_PATTERN.matcher(encodedPassword).matches()) {
78              this.logger.warn("Encoded password does not look like BCrypt");
79              return false;
80           } else {
81              return BCrypt.checkpw(rawPassword.toString(), encodedPassword);
82           }
83        } else {
84           this.logger.warn("Empty encoded password");
85           return false;
86        }
87     }
88
89     // Checks if the encoded password should be re-encoded with a stronger hashing strength
90     public boolean upgradeEncoding(String encodedPassword) {
91        if (encodedPassword != null && encodedPassword.length() != 0) {
92           Matcher matcher = this.BCRYPT_PATTERN.matcher(encodedPassword);
93           if (!matcher.matches()) {
94              throw new IllegalArgumentException("Encoded password does not look like BCrypt: " +
       encodedPassword);
95           } else {
96              int strength = Integer.parseInt(matcher.group(2));
97              return strength < this.strength;
98           }
99        } else {
100          this.logger.warn("Empty encoded password");
101          return false;
102       }
103    }
104 }
```

Appendix G: API Documentation

1. General Information

Authentication: No authentication required for public endpoints. Some endpoints require authentication via JWT token (obtained after login).

2. API Endpoints

1) Upload Avatar

URL: `POST localhost:8080/allowlist/uploadAvatar`
Content-Type: `form-data`
Request Parameters: `avatar` (File, required): Image file to upload as avatar.
Response Example:

```
{
    "code": "1",
    "message": "Upload successful",
    "url": "https://cpt202.oss-cn-shanghai.aliyuncs.com/uploads/a60a39ac-2a60-4db0-9ca1-57
    d3bd0cdeac.png"
}
```

2) Send Registration Email Verification Code

URL: `POST localhost:8080/allowlist/sendRegisterCode`
Content-Type: `application/json`
Request Parameters:

```
{
    "email":"Mingxuan.Hu22@student.xjtlu.edu.cn"
}
```

Response Example:

```
{
    "code":"1",
    "message":"Verification code sent. Please check your email."
}
```

3) User Registration

URL: `POST localhost:8080/allowlist/register`
Content-Type: `application/json`
Request Parameters:

```
{
    "username":"TestUser1",
    "password":"123ABCD",
    "email":"Mingxuan.Hu22@student.xjtlu.edu.cn",
    "registerCode":"274507",
    "role":"admin",
    "avatar":"https://cpt202.oss-cn-shanghai.aliyuncs.com/uploads/ecc43517-4021-4e4a-9b25-11
    b94e31354b.png",
    "adminKey":"Cpt202Group50"
}
```

Response Example:

```
{
    "code":"1",
    "message":"Registration successful"
}
```

4) Send Forgot Password Email Verification Code

URL: POST localhost:8080/allowlist/sendForgetPwdCode
Content-Type: application/json
Request Parameters:

```
{
    "email":"Mingxuan.Hu22@student.xjtlu.edu.cn"
}
```

Response Example:

```
{
    "code": "1",
    "message": "Verification code sent. Please check your email."
}
```

5) Reset Password

URL: POST localhost:8080/allowlist/forgetPwd
Content-Type: application/json
Request Parameters:

```
{
     "email":"Mingxuan.Hu22@student.xjtlu.edu.cn",
     "forgetPwdCode":"412094",
     "password":"1234ABCD"
}
```

Response Example:

```
{
    "code": "1",
    "message": "Password reset successful"
}
```

6) CAPTCHA Image

URL: GET localhost:8080/allowlist/captchaImage
Content-Type: None
Response Example:

```
{
    "code":"1",
    "captchaKey":"5884795a-7710-4ca8-b80f-1766e5bb5c0d",
    "captchaImage":"base64-encoded image data"
}
```

7) User Login

URL: POST localhost:8080/allowlist/login
Content-Type: application/json
Request Parameters:

```
{
    "username":"TestUser1",
    "password":"1234ABCD",
    "captchaKey":"5884795a-7710-4ca8-b80f-1766e5bb5c0d",
    "captchaCode":"N6LV"
}
```

Response Example:

```
{
    "code": "1",
    "data": {
        "user": {
```

```
            "id": 8,
            "username": "TestUser1",
            "email": "Mingxuan.Hu22@student.xjtlu.edu.cn",
            "avatar": "https://cpt202.oss-cn-shanghai.aliyuncs.com/uploads/ecc43517-4021-4e4a-9b25-11b94e31354b.png",
            "createdAt": "2025-04-19T13:12:03",
            "lastLoginAt": "2025-04-19T13:15:04.4281639",
            "lastPasswordChangeAt": "2025-04-19T13:13:13",
            "isLocked": false,
            "emailVerified": true,
            "role": "admin"
        },
        "token": "JWT token returned upon successful login"
    }
}
```


8) Authorization Test Endpoint

URL: `GET localhost:8080/test/test1`
Content-Type: `application/json`
Request Header Parameters: `Authorization` (String, required): `Bearer <JWT token>`
Description: This endpoint has no actual business logic, it is used only to test authorization functionality.

3. Common Response Code Explanation

`code`:

- `"1"`: Success
- `"0"`: Failure