**1 Introduction**

Our project aimed to create an **"online meeting booking system"** to fix the common problems of inefficiency and booking conflicts with how meeting rooms are managed at universities. The final product is a web application with a separate front end (built with Vue.js) and back end (using Spring Boot). This setup allows for real-time updates and better management of campus meeting rooms.

Our team dedicated to provide a comprehensive booking platform for two main targeted primary user groups: **students** and **administrators**. Students are normal users, they can utilize the platform to perform a series of operations, functions including account creation and personal information maintenance, to intelligent filter meeting rooms (based on capacity, facilities, location), time slot booking, and subsequent management of their booking records including viewing, modification, or cancellation. For administrators, the system grants more extensive control, encompassing the management of meeting room resources, comprehensive user account administration, such as locking or unlocking accounts, centralized approval and status tracking of all booking requests, and timely processing of user feedback.

Within this project, my key individual responsibilities revolved around the **construction of the authentication mechanism** and the **user profile modification**. To fulfill these duties, the final module I delivered incorporates **a series of core features** aimed at enhancing security and user experience. Firstly, the authenticity of new user's identity during registrations is ensured through **CAPTCHA via email**. Secondly, to safeguard user logins, all user passwords are encrypted using a **"hashing and salting (Group33)" encryption method**. Thirdly, I developed a **login authentication process integrated with session management as well as adding a request time limit to CAPTCHA request**, which can effectively maintain user login status and flow cosntrol of system. Furthermore, in their profile page, the system empowers users to **autonomously update their login passwords and registered email addresses**, providing essential self-management capabilities for their accounts. A significant achievement in my work was the design and implementation of an **AOP (Aspect-Oriented Programming)-based authorization control scheme**. This scheme, through the introduction of custom annotations **(@AuthCheck)**, enables **role-based access control (RBAC)**, thereby ensuring that various functional modules of the system are accessible only to users who have been authenticated and possess the accessible operational permissions.

The subsequent sections of this report will provide a detailed review of the software development methodology adopted for the project, with a specific focus on the tasks I accomplished within the S**crum agile development framework**. Following this, an in-depth analysis of the software design implementation details for the various **Product Backlog Items (PBIs)** I was responsible for will be presented, along with a discussion of the change management strategies employed during the project. The report will then examine the legal, social, ethical, and professional considerations encountered during project development, particularly in relation to my individual contributions. The concluding part of the report will summarize the overall project experience, distill personal insights, and propose constructive suggestions for improvement in future endeavors.

**2 Software Development Process**

Throughout the development lifecycle of the online meeting booking system, our team fully adopted the Scrum framework. As a mainstream agile development methodology, Scrum's core characteristics emphasized **iterative progress**, encouraging **close collaboration** and **continuous communication** among team members, and maintaining a flexible responsiveness to evolving requirements, which highly adhered to the nature of this project, and developer can use these plans to explain and classify what project needs. Our motivation for choosing Scrum is the expectation that its systematic practice will enable us to more effectively navigate the various uncertainties in software development. Through fixed Sprint cycles, we were able to deliver operational software functional modules and integrate feedback after each iteration into subsequent development iterations. This ensured that the final product could meet users' core needs and successfully achieve the project's established strategic objectives.

**2.1 My Individual Work Planning within the Scrum Process**

Throughout the team's Sprint Iteration framework, the **“Authentication”** and **“User Profile Modification”** modules that I was responsible for developing were also systematically planned and driven according to Scrum's iterative principles. My development work was mainly organized around the following phased objectives across the four Sprints, and the details could be seen in the *Appendix 1*.

**2.2 Rationale Behind the Scrum Plan Design**

This Scrum plan for my modules was designed with several considerations to ensure logical progression and alignment with the team's overall developing progress. Each sprint was structured to build incrementally on the previous one.

* **Sprint 1** focused on laying the groundwork. The priority was to establish the core user entity and a basic, albeit incomplete, registration pathway. This allowed foundational data structures to be agreed upon early.
* **Sprint 2** aimed to make the system usable from an authentication perspective by implementing login and session management, and crucially, adding email verification to bolster the security and validity of registrations. This provided an early testable authentication loop.
* **Sprint 3** shifted towards enhancing user control over their accounts (password/email updates) and introducing a critical security layer with AOP-based authorization. The decision to introduce AOP at this stage was based on having stable core authentication features and a growing number of APIs that would require protection.
* **Sprint 4** was dedicated to ensuring the robustness, security, and maintainability of my modules through comprehensive testing, refinement, and documentation. Extending AOP application also ensured that as the wider system functionalities stabilized, security was consistently applied.

PBIs were allocated to sprints based on their logical dependencies, a desire for early feedback on core functionalities, and the technical readiness to tackle more complex features like AOP once foundational elements were in place.

**2.3 Scrum's Impact on Communication and Collaboration Efficiency**

The application of Scrum principles and its structured cycle significantly enhanced our team's communication and collaborative efficiency:

* **Sprint Planning:** The **Sprint Planning Meeting** marks the **start of each new Sprint**, designed to define the Sprint Goal and the specific tasks to achieve it. At the beginning of each Sprint, our team prioritizes incomplete features based on current project progress and existing completed modules. These **highest-priority features** are then refined into **smaller, more easily traceable Sprint Backlog items**, which are subsequently assigned fairly to all team members. By the meeting's conclusion, we generate a **clear blueprint for action** to guide development for the entire Sprint cycle.

For instance, in planning for Sprint 2, a key discussion involved the **exact request/response JSON format for the Login API** and the email verification API, covering details like parameters, response structures, and error codes. This required **direct communication with frontend developers** to ensure a unified transmission message. Such proactive clarification was crucial in **preventing significant rework** from misaligned API specifications, directly boosting overall efficiency.

* **Implementation & Daily Scrum:** In this phase, our team works from the **Sprint Backlog** to develop a **product increment**, engaging in design, coding, and testing. To maintain progress and address issues, we hold **Daily Scrums, typically every Wednesday**, where we synchronize progress and discuss impediments. This ensures we can quickly adapt and meet Sprint goals.

For instance, while developing the **user authentication feature in Sprint 3**, a disagreement arose on the implementation. Some suggested simple 'if' statements for role checks. However, leveraging **Spring's AOP capabilities**, I proposed creating a custom **@Around annotation for authentication and permission control**. This approach offered better reusability and extensibility for future interfaces. **My proposal was ultimately adopted by the team**. These regular discussions are crucial for identifying cross-task issues and collaboratively finding solutions, thereby maintaining the team's overall efficiency.

* **Sprint Review:** We conducted this section at the end of each cycle, provides a valuable platform for **demonstrating our completed product increment to stakeholders**, which includes course instructors, TAs, and classmates, and our target was to gather their direct feedback. This input is essential for me to assess if the features I developed meet expectations. Based on this, I can refine the **Product Backlog**, making necessary adjustments to guide future development.

For instance, **after Sprint 2**, a TA’s concern about **preventing fraudulent email API requests** was noted. I ensured this became a PBI for the next Sprint, and its implementation enhanced system stability. **Later, after Sprint 3**, I demonstrated the new **'user password modification'** feature and showcased how our custom **@AuthCheck annotation** effectively secured the admin-only **'delete user'** API, preventing unauthorized access. Through such demonstrations, we receive **immediate feedback**, enabling continuous, **collective wisdom-driven iteration** and optimization of the product's user experience.

* **Sprint Retrospective:** We schedule the **Sprint Retrospective immediately after the Sprint Review** and before the next Sprint Planning meeting. It serves as an internal team activity dedicated to **self-reflection and continuous improvement**. The primary objective is to collectively review the just-concluded Sprint, examining our technology stack, team collaboration, and project progress. **We openly discuss what aspects were successful and should be maintained, identify any problems or shortcomings, collaboratively explore their root causes, and formulate specific, actionable improvement measures** to apply in subsequent Sprints.

For example, **after Sprint 2**, as our project had grown, our team jointly identified a significant issue: the **lack of consistent**, **easily accessible API documentation** for backend services, including my authentication endpoints. This led to frequent questions from frontend developers and somewhat slowed down integration efficiency. To address this pain point, in the following Sprints, I prioritized allocating time to **generate and update API documentation** for my modules, utilizing tools like **Knife4j**. This decision, stemming from collective team reflection, markedly **improved our workflow** and effectively **reduced communication barriers** and friction between backend and frontend developers.

This cyclical process of planning, executing, inspecting, and adapting, inherent in Scrum, fostered a highly communicative and efficient development environment. It ensured that my work on the critical authentication and user profile components was not only technically sound but also well-aligned with the team's progress and evolving needs.

**3 Software Design**

This section details the design philosophy and specific implementation of the **Product Backlog Items (PBIs)** for which I was individually responsible. These PBIs primarily revolve around the core functional areas of the **User Authentication Module** and **User Profile Management**, as well as include the implementation of a practical **Aspect-Oriented Programming (AOP)-based** authorization mechanism. Throughout the design and development process, **system security**, **usability**, and **subsequent maintainability** were the fields I mainly focus on.

**3.1 General Design and Implementation of My PBIs**

My individual contributions can be clearly demarcated into several key functional areas, each comprising a series of interconnected PBIs aimed at collaboratively delivering a complete and user-friendly feature set.

PBIs-ONBOARDING: User Registration & Email CAPTCHA Verification

* Design & Implementation Overview: This PBI group constructed the user registration pathway. Core components include: the /user/register​ API for handling registration requests (including optional avatar and verification code); the /user/ask\_code​ API, which, in conjunction with EmailSender​ and a Caffeine cache, implements email OTP sending and temporary storage; and a RateLimiter​ to restrict OTP request frequency. User passwords are encrypted using an MD5 with salt method (UsersServiceImpl.getEncryptPassword​). Avatar uploads are processed via QiniuService​ and image’s URL will be stored in the users​ table.
* Justification: Email OTP verification and request rate limiting ensure the authenticity of registrations and prevent abuse. Password hashing secures credentials. Cloud storage for avatars reduces server load and optimizes access speed.
* UML Placeholder:
  + [UML Placeholder: State Machine Diagram for User Registration Flow]

PBIs-AUTH: User Login, Session Handling & Logout

* Design & Implementation Overview: This PBI set implemented secure user access and session management: the /user/login​ API validates users by comparing hashed passwords; upon success, an HttpSession​ is established server-side, storing the user entity (keyed by UserConstant.USER\_LOGIN\_STATE​). The /user/me​ API allows authenticated users to retrieve their sanitized personal information. The /user/logout​ API invalidates the session for a secure exit.
* Justification: A secure login mechanism is fundamental to system access control. Session management maintains user context. A clear logout function ensures user account security.
* UML Placeholder:
  + [UML Placeholder: State Machine Diagram for User Session Lifecycle]

PBIs-PROFILE: User Profile & Credentials Management

* Design & Implementation Overview: These PBIs empower users with account management capabilities: changing usernames (/user/change-username​); updating email addresses (/user/change-email​, which reuses email OTP verification for the new email); and resetting passwords (/user/reset-password​, also relying on email OTP for identity confirmation). User avatars can be updated via /user/upload-avatar​ (through QiniuService​), and administrators can delete user avatars via /user/delete-avatar​.
* Justification: User self-management of profile information and credentials is standard and essential for data accuracy and account security. Secondary OTP verification for critical changes enhances security.
* UML Placeholder:
  + [UML Placeholder: State Machine Diagram for Email Update Process]

PBIs-AOP: Fine-Grained Role-Based Access Control via AOP

* Design & Implementation Overview: To achieve fine-grained permission control, an AOP authorization mechanism was designed: the @AuthCheck​ annotation (AuthCheck.java​) was created with a mustRole​ parameter to define access rights; an AuthInterceptor​ aspect (AuthInterceptor.java​) uses @Around​ advice to intercept annotated methods. The interceptor compares the current logged-in user's role (obtained from the session via UsersService.getLoginUser(request)​) with the role required by the annotation, allowing or denying access (throwing a BusinessException(ErrorCode.NO\_AUTH\_ERROR)​) based on predefined logic (e.g., admins have all permissions). This is applied to several UsersController​ methods.
* Justification: AOP decouples authorization logic from business logic, improving code readability and maintainability. Declarative permission configuration simplifies security management.
* UML Placeholder:
  + [UML Placeholder: Conceptual Diagram for AOP Authorization Flow]

**3.2 Detailed Design and Implementation of My Core PBIs**

Among all the PBIs I was responsible for developing, the following two are particularly prominent due to their critical importance to the system's functional integrity, security assurance, and demonstration of software engineering best practices: 1) PBI-REG-DETAIL: Implementing a Secure User Registration Process with Email Verification, and 2) PBI-AOP-DETAIL: Constructing an AOP-based Authorization Mechanism for API Endpoints. Next, I will elaborate on the design and implementation details of these two core PBIs.

**3.2.1 PBI-REG-DETAIL: Implementing a Secure User Registration Process with Email Verification**

This PBI serves as the entry point for users into the system, and the robustness and security of its design directly impact the initial quality of user data and the overall security of the system.

* Logical Flow: User registration starts with form submission to the **/user/register endpoint**. Before this, users request an email OTP via **/user/ask\_code**, involving rate limiting, OTP dispatch, and temporary caching. The core registration logic then **validates the OTP**. Upon successful validation, it **checks for username and email uniqueness** and **securely encrypts the password**. A new user entity is created (default role 'USER'), an optional avatar is processed (uploaded to cloud storage), and finally, the **user data is persisted** to the database. The OTP cache is then cleared, and a user object (UserVO) is returned. Validation failures trigger a BusinessException.
  + UML Placeholder:
    - [UML Placeholder: Activity Diagram for User Registration]
    - [UML Placeholder: Sequence Diagram for User Registration]
* Database Support: This PBI heavily relies on the users​ table. The email​ field's NOT NULL UNIQUE​ constraint is fundamental for email verification and user uniqueness. The password​ field stores the encrypted digest for security. The username​ is ensured unique at the business logic layer. avatar\_url​ stores the avatar link, and role​ defaults to 'user'. Timestamps like created\_at​ and updated\_at​ aid auditing.
* States of Objects Involved:
  + ​Users​ Entity Object: Transitions from a transient in-memory state (during data population, e.g., setting username, hashed password, role to 'USER') to a persistent state (after successful database insertion, gaining a user\_id​ and timestamps).
  + CAPTCHA in LOCAL\_CACHE​: Has two main states: "Active & Valid" (exists in cache, not expired) and "Expired/Invalidated" (removed due to TTL or usage). The RateLimiter​ also maintains cached states for IP request times.
* UI Discussion:
  + UI

**3.2.2 PBI-AOP-DETAIL: Constructing an AOP-based Authorization Mechanism for API Endpoints**

This PBI introduces a declarative, non-invasive authorization solution for the system's API endpoints by leveraging Aspect-Oriented Programming (AOP).

* Logical Flow: When a request targets a controller method annotated with **@AuthCheck** (e.g., an admin-only endpoint), **Spring AOP intercepts the call** before method execution. An interceptor then retrieves the required role (mustRole) from the annotation and the current user's role from their session. **Authorization logic is then applied**: access is granted if mustRole isn't specified or if the user is an admin. If roles mismatch (e.g., a regular user attempts an admin action), a **BusinessException is thrown**, preventing unauthorized access. If authorization succeeds, the original controller method executes; otherwise, an appropriate error response is returned.
  + UML Placeholder:
    - [UML Placeholder: Activity Diagram for AOP Authorization]
    - [UML Placeholder: Sequence Diagram for AOP Authorization]
* Database Support: The core data for AOP authorization originates from the role​ field in the users​ table. The AuthInterceptor​ uses the logged-in user's role value (corresponding to constants in UserConstant.java​) for its decisions. This role, defaulting to "USER" on registration, and its accuracy in the database, are crucial for the authorization logic.
* States of Objects Involved:
  + ​@AuthCheck​ Annotation Instance: Its mustRole​ attribute value is determined at compile/deployment time and read by the interceptor at runtime.
  + ​Users​ Entity Object (Logged-in User): Its role​ property (established at login and stored in the session) is the key state for authorization checks.
  + ​ProceedingJoinPoint​ (AOP Concept): Represents the intercepted method call; the interceptor controls its execution flow via the proceed ()​ method.
* UI Discussion:
  + [UI Discussion to be added based on provided UI details.]

Appendix:  
Appendix 1: Scrum’s sprint planning table

| **Sprint Cycle** | **My Key PBIs** | **Sprint Theme** | **Corresponding Group Sprint Phase** |
| --- | --- | --- | --- |
| **Sprint 1** | 1. Design users database schema and entity.  2. Set up project structure for Authentication module.  3. Implement initial User Registration API (core fields).  4. Implement password hashing (hash + salt). | Establishing User Identity Foundations & Basic Registration | Initial Design and Setup |
| **Sprint 2** | 1. Implement User Login API.  2. Implement basic session management.  3. Add email verification (OTP(Once Time Pasword) send & verify) to registration.  4. Design API for fetching user profile data. | Enabling Secure Login & Initial Profile Access with Email Verification | Foundational Backend Structures |
| **Sprint 3** | 1. Implement Update Password functionality. 2. Add a time limit for requesting to send an Email.   2. Implement Update Email Address functionality.  3. Design and implement core AOP authorization aspect ( @AuthCheck).  4. Apply AOP to critical user-specific APIs.  5. Document the interfaces I have implemented. | Enhancing Profile Management & Introducing API Authorization | Core Logic Implementation & API Integration |
| **Sprint 4** | 1. Conduct unit and integration testing for all module features.  2. Refine error handling and API responses.  3. Extend AOP authorization to relevant new/other team members' endpoints.  4. Complete the documentation on Authentication APIs. | Module Hardening, Comprehensive Testing & Broader Authorization Integration | Deployment & Comprehensive Testing |

