****

**CS673 Software Engineering**

**Team 1 - Spring Alert**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| James (Jim) Penner | Team Leader | *Jim Penner* | 05/15/2021 |
| Shenxiao (Sherry) Li | Requirement Leader | *Sherry Li* | 05/15/2024 |
| Jordany Arnaud | Design & Implementation Leader / Security | *Jordany A.* | 05/27/2024 |
| Chaozeng Zhu | Configuration Leader | *Chaozheng Zhu* | 05/23/2024 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **0** | **Team1:Spring\_Alert** | **5/15** | **v1.0** |
| **1** | **Team1:Spring\_Alert** | **05/27** | **v1.1** |
| **2** | **Team1:Spring\_Alert** | **06/17** | **v1.2** |
|  |  |  |  |

[Overview](#_heading=h.gjdgxs)

[Related Work](#_heading=h.30j0zll)

[Proposed High level Requirements](#_heading=h.1fob9te)

[Management Plan](#_heading=h.3znysh7)

[Objectives and Priorities](#_heading=h.2et92p0)

[Risk Management (need to be updated constantly)](#_heading=h.tyjcwt)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.3dy6vkm)

[Configuration Management Plan](#_heading=h.1t3h5sf)

[Tools](#_heading=h.4d34og8)

[Deployment Plan if applicable](#_heading=h.2s8eyo1)

[Quality Assurance Plan](#_heading=h.17dp8vu)

[Metrics](#_heading=h.3rdcrjn)

[Code Review Process](#_heading=h.26in1rg)

[Testing](#_heading=h.lnxbz9)

[Defect Management](#_heading=h.35nkun2)

[References](#_heading=h.1ksv4uv)

[Glossary](#_heading=h.44sinio)

# Overview

Our main contribution to the Online Learning Platform involves creating a user management service which is responsible for handling user-related operations, such as user registration, user profile management, and group management, including the security design for the UI as well as the backend APIs via Auth0. In addition to that, we have committed to implementing a simple “Notification Service.” This service is important for our project-based learning platform.

Purpose and Motivation

In a microservice-based architecture with numerous service APIs and a single-page application (SPA) frontend, maintaining robust security is both crucial and challenging. This complexity arises due to the need to protect data integrity and user privacy across distributed components while ensuring seamless user interactions.

Our primary goal is to establish a comprehensive security framework that encompasses both the user interface (UI) and backend APIs. We aim to implement a solution that not only safeguards user data during registration, profile management, and group operations but also ensures consistent and scalable security practices across our platform. By doing so, we enhance user trust and platform resilience against potential threats.

Our second purpose is to make communication and information sharing easier among everyone on the online platform, allowing notifications to be sent through emails Our aim is to make sure users have a better experience, stay engaged, and collaborate more effectively by keeping them informed about what's happening in the learning platform.

Potential users

The platform is designed to cater to a diverse set of end users, each with distinct roles and needs:

* *Students*: Individuals who use the platform for learning. They access educational content, participate in courses, complete assignments, and interact with instructors and peers.
* *Instructors/Tutors*: Qualified professionals who create and deliver educational content. They facilitate learning by providing lectures, grading assignments, and supporting students through their educational journey.
* *Administrators*: Staff members responsible for managing the platform and its content. This includes university personnel who oversee the delivery of online learning programs, manage user access, handle technical issues, and ensure that the platform operates smoothly and securely.

From a technical standpoint, the Single Page Application (SPA) serves as the primary client interface for our APIs. The SPA acts on behalf of the users, facilitating interactions with the backend services. This approach enables a seamless and dynamic user experience by leveraging the SPA to handle most of the client-side logic and interface updates, thereby reducing the need for full-page reloads.

Features Overview

User Management Service:

* *User Profile Registration & Management*: The User Profile Registration & Management feature facilitates the registration of new users by allowing them to create accounts with necessary personal details. Once registered, users can manage their profiles by updating personal information and changing settings. This feature also integrates access control mechanisms to ensure that users can securely access and modify their profiles according to their assigned permissions.
* *User Group Management*: The User Group Management feature enables the creation and administration of user groups to support organizational and role-based structuring within the platform. Administrators or authorized users can create groups, assign roles, and define permissions, allowing users to be associated with one or more groups based on their roles and functions (e.g., students, instructors, administrators).

Notification Service

* *Template Management*: Users can create standard templates for email notifications. These templates can be in HTML or plain text and include placeholders for things like recipient names or event details. This ensures that notifications look consistent and professional, making them more engaging for users.
* *Sending Notifications*: The main function of the service is to send notifications to specific recipients or groups through email. Users can choose who gets the notifications.

Potential Technology Stack

* *Database: PostgreSQL* will be our database of choice, offering flexibility and scalability for storing data in a relational-oriented format.
* *Microservices Framework*: We plan to utilize Spring or Spring Boot for building our microservices, providing a robust and scalable framework for development.
* *Messaging Service*: RabbitMQ will serve as our messaging service, facilitating communication between microservices in an asynchronous and reliable manner.

In addition to the Notification service, we will also implement an API gateway using Spring Cloud Gateway if we have time. This gateway will serve as a centralized entry point for the notification service and all our other services. It will provide the following functionalities:

* *Dynamic Request Routing Configuration*: The API gateway will allow for dynamic configuration of request routing, enabling us to direct requests to the appropriate microservices based on defined rules and conditions.
* *Rate Limiting*: We will implement rate limiting within the API gateway to control the rate of incoming requests, preventing overload and ensuring optimal performance of our services.
* *Authentication/Authorization Enforcement*: The gateway will enforce authentication and authorization policies, ensuring that only authorized users can access our services and resources using spring security.
* *Resiliency and Circuit Breakers*: To enhance the reliability and fault tolerance of our system, the API gateway will incorporate resiliency patterns such as circuit breakers. These mechanisms will help prevent cascading failures and maintain system stability during periods of high load or failure.

# Related Work

One of the notification systems for schools based on our research is Apptegy's notification system, Thrillshare. It is a comprehensive notification system designed for schools, allowing alerts and emergency notifications to be sent via text messages, voice calls, social media, websites, and apps from a unified platform. It features easy-to-use functionalities like recipient customization, voice call recording, and text-to-speech. Thrillshare integrates seamlessly with various school information systems, ensuring accurate targeting and efficient communication. [Reference1](https://thrillshare.com/). [Reference2](https://medium.com/geekculture/push-notification-for-order-placed-using-observable-design-pattern-with-spring-boot-websocket-9203874f3ebb)

However, our system is being developed for academic purposes, focusing specifically on implementing essential functionalities such as templates and notifications.

# Proposed High level Requirements

Functional Requirements

User Management Service**：**

* *Register a new user*: As a potential student or instructor joining the online learning platform, I want to register with my details so that I can access courses and participate in educational activities.
* *Get All/Searchl user profiles*: As an administrator of the online learning platform, I need to retrieve all user profiles so that I can manage student and instructor accounts effectively and ensure smooth platform operation.
* *Update a user profile*: As a user (student or instructor), I want to modify my profile information so that my educational background and contact details are up-to-date.
* *Get user profile by ID*: As a system administrator or support staff, I need to fetch specific user details by ID so that I can provide personalized assistance and resolve user queries efficiently.
* *Get current user profile*: As a logged-in user (student or instructor), I want to view or update my own profile information for better interaction and engagement on the platform.
* *View my current user profile*: As a logged-in user, I want to view or update my own profile information.
* *Create a user group*: As an administrator or course organizer, I want to create a new course group so that students and instructors can collaborate on specific course topics or projects.
* *Add users to a group*: As an administrator or course coordinator, I want to add students or instructors to a course group so that they can collaborate on course projects and discussions.
* *Add a user to a group*: As an administrator or group manager, I need to assign a student or instructor to a specific course group so that I can manage group enrollments and participation effectively.
* *View all groups*: As an administrator or platform manager, I need to retrieve all course groups so that I can monitor course enrollments, assignments, and discussions across the platform.
* *View group by ID*: As an administrator or support staff member, I need detailed information about a specific course group by its ID to assist in resolving group-specific issues and managing group dynamics effectively.
* *Leave group*: As a user, I want to leave a group, so that I can remove myself from group activities and resources
* *View group members*: As an administrator or course coordinator, I want to retrieve members of a specific course group so that I can manage group memberships, track participation, and ensure group activities run smoothly. / As a current group member I want to view the entire list of group members and potentially some basic details (short Bio) so that I can start to know/connect to the people in the group.

Notification Service

* *Create template:* As a user, I want to create email notification templates, so that notifications have a standardized format and professional appearance.
* *Message everyone in the group*: As a current group member, I want to email/sms/discord with each member in my group(s) in order to communicate a message effectively with all members simultaneously.
* *Track Notification delivery status:* As a user, I want to track the delivery status of notifications, so that I can ensure timely communication with recipients.
* *View Notification:* As a sender, I want to view details of sent notifications, such as timestamps and delivery outcomes, so that I can troubleshoot any issues and improve notification delivery.

Optional Features

* *Send email notification with attachments:* As a user, I want to include attachments in email notifications, so that additional context or supplementary information can be provided.

Non-functional requirements

Security requirements

We will implement security measures using Spring security. Additionally, we will leverage OAuth2 for authentication and authorization using Auth0, providing secure access to our services and APIs.

# Management Plan

## Objectives and Priorities

We will prioritize the core functionalities first, followed by desirable and optional features if time permits.

**Iteration Breakdown Overview:**

**Iteration 1:**

* Set up Development Environment*–shore in pivot tracker*
* User Sign Up
* User Log In

**Iteration 2:**

* View Profile
* Create Group
* Add user to group
* View Group member

**Iteration 3:**

* Send notification
* Update notification
* Read notification status
* Buffer time for testing and bug fixes
* Aggregation & Deployment

Security Features spread across iterations. Optional features can be implemented if there is remaining time or during buffer periods in the iterations.

## Risk Management

(*Please write a summary paragraph about the main risks your group identified and how you plan to manage these risks. Then use the separate google sheet for detailed risk management. The template is provided in the same folder with this file*) \_Jim

[**Risk Management Sheet Link**](https://docs.google.com/spreadsheets/d/14pns9ijnIF713gwWw10GxbhO24Xk8EWN/edit#gid=1663912720)

## Timeline

### Iteration Plan(Update)

| Iteration | Functional Requirements | Tasks (Cross requirements tasks) | Estimated per person hours |
| --- | --- | --- | --- |
| 1 | User Registration  User Login  Set Auth0.com as the authorization server | UML diagram  install and configure Spring Boot, RabbitMQ  Set up version control with Git.  Implement registration logic.  Integrate with RabbitMQ for messaging.  Configure Spring Security for the application.  Set up OAuth2 authentication with Auth0.  Design resource/role-based access control | Estimate:  20 ~30h/person |
| 2 | View profile  Create Group  Add user to group  View group member  Send notifications | Database design for retrieval in PostgreSQL  Integrate permission setting with group  Implement backend logic for storing preferences in PostgreSQL  Integrate with RabbitMQ for delivering notifications through chosen channels.  Store notifications in PostgreSQL  Unit tests  Test end-to-end functionality. | Estimate:  40 ~50h/person |
| 3 | Refine key features  End-to-end testing  Document Refine | Add security code to authorize user to send notification;  Buffer Time for Testing and Bug Fixes | Estimate:  25 ~35h/person |

## 

# Configuration Management Plan

## Tools

In this project, we are committed to leveraging a robust set of tools to streamline development, ensure code quality, and enhance deployment efficiency

* *Git and GitHub*: Utilized for version control to manage code changes and facilitate collaboration. [See](https://github.com/BUMETCS673/project-teamfirst)
* *IntelliJ IDEA*: Chosen as the integrated development environment (IDE) for its powerful coding and debugging capabilities.
* *GitHub Actions*: Employed for CI/CD automation to automate build, test, and deployment workflows.
* *Docker*: Used for containerization to create, deploy, and manage application containers.
* *CockroachDB*: Selected as the database system for its scalability and resilience in handling data. [Website](https://www.cockroachlabs.com/)
* *Rencher*: Utilized for container orchestration and deployment management to optimize containerized applications' deployment and scalability.[Website](https://render.com/)
* SAST/DAST (TBD): Tools for Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) are yet to be determined to ensure robust security measures throughout the development lifecycle.

## b. Code Commit Guideline and Git Branching Strategy

In collaborative software development, the Code Commit Guideline and Git Branching Strategy serve as foundational frameworks for organizing code contributions and structuring Git branches. This section of the document elucidates the branching strategy designed to streamline workflows, mitigate conflicts, and ensure the methodical integration of new features and fixes.

Branch Types:

* *Main Branch*: This branch serves as the main line of development and should only be used to pull changes from upstream.
* *Iteration Branch*: We create a new branch for each iteration or feature development. Developers work on these branches and merge them into the main branch at the end of the iteration after review.

Workflow and Guideline

* Connect your local repository to the project repository hosted on GitHub.
* Before starting work, ensure your local main branch is updated with the latest changes from the remote repository.
* Create a new branch dedicated to your current iteration or feature development.
* Stage and commit your changes locally, then push them to your remote iteration branch.
* Initiate a request on GitHub to merge your changes from the iteration branch into the main branch after completing your feature development.

## c. Deployment Plan if applicable

We plan to deploy our application using Render to establish a CI/CD pipeline for building and deploying the application automatically. Render will be configured to receive notifications upon new changes being pushed to the repository, triggering automatic redeployment of the project. This approach ensures continuous integration and deployment, streamlining the process of updating and maintaining our web application in a scalable and efficient manner. [Render Link](https://render.com/)

# Quality Assurance Plan

## Metrics

The table provides a comprehensive view of metrics essential for evaluating the quality of a user management service and notification service. Here's an analysis and summary of the included metrics:

| **Metric** | **Description** | **Target** | **Tool** |
| --- | --- | --- | --- |
| Lines of Code (LOC) | Measure codebase size | Track growth over time | SonarQube |
| Cyclomatic Complexity | Identify complex code | Average < 10 | SonarQube |
| Code Coverage | Ensure test coverage | > 80% | Jacoco |
| Code Duplication | Identify redundant code | < 5% | SonarQube |
| Number of Test Cases | Track test coverage | Increase per iteration | JUnit |
| Defect Density | Measure defect rate per KLOC | < 0.5 defects/KLOC | Pivotal tracker |
| Defect Detection Rate | Track defects per phase | High in development | Pivotal tracker |
| User Stories Completed | Track feature development | Defined per iteration | Pivotal tracker |
| Sprint Burndown Rate | Measure task progress | Complete by sprint end | Pivotal tracker |
| Velocity | Measure rate of work completion | Maintain or increase | Pivotal tracker |
| Cost | # of Person-hours expended | 8 hours per iteration | Pivotal tracker |
| Stories | #of User stories completed |  | Pivotal tracker |
| Bugs/LoC | Bugs found per LoC |  |  |
| Dependencies | Number of dependencies |  |  |

b. Coding Standard

Our team will adhere to the below guidelines to ensure that the codebase remains consistent, maintainable, and aligned with industry best practices. These standards should be regularly reviewed and updated to reflect new learnings and evolving best practices.

*Overall Principles*

* Adopt the Google Java code style.
* Follow the standard Java naming conventions.
* Adhere to the SOLID principles to ensure maintainable and scalable code.
* Utilize design patterns appropriately to solve common problems.
* Follow refactoring guidelines as described by Martin Fowler.
* Keep methods short and focused on a single responsibility.
* Prefer fewer arguments. Use parameter objects for complex parameter lists.
* Favor composition over inheritance unless there is a clear "is-a" relationship.
* Use interfaces or abstract classes for methods that can have multiple implementations.

*Error Handling*

* Prefer creating custom exceptions for specific error scenarios.
* Ensure error messages are clear and descriptive.

*Logging*

* Use SLF4J with Logback for logging.
* Avoid logging sensitive information.

*API Design*

* Use appropriate HTTP methods (GET, POST, PUT, DELETE).
* Implement JWT or OAuth 2.0 for secure authentication.
* Validate all inputs using libraries like Hibernate Validator.
* Use environment variables instead of hardcoded properties for configuration.
* Use nouns instead of verbs in endpoints to represent resources.
* Use URL versioning for clarity.
* Use Swagger/OpenAPI for documenting APIs.

## c. Code Review Process

The following is a structured code review process to ensure quality, consistency, and knowledge sharing within the team:

* Each feature has a designated owner responsible for writing the initial code and ensuring its completeness and quality.
* The owner documents the feature's design, key decisions, and known limitations in the code repository.
* A code review meeting is held via Zoom three days before each iteration, with all team members invited to participate.
* The code owner presents their code, explaining the purpose, implementation details, and notable design choices or challenges.
* Team members provide constructive feedback, ask questions, and suggest improvements.
* The feature owner resolves identified issues and updates the code accordingly.
* Once all feedback has been addressed and the team is satisfied with the changes, the code is approved for merging.

## d. Testing

For unit testing, we will utilize JUnit 5 along with either WebClient or REST Assured to develop our tests. Each developer will be responsible for writing unit tests, while the QA lead (Chao) will focus primarily on functional and end-to-end (integration) testing.

Unit tests will be executed daily to ensure continuous integration. Functional testing will be conducted as components reach a feature-complete state. Integration testing, likely using tools such as Postman, will commence as other teams (such as UI) begin delivering their beta products.

Our testing efforts primarily aim to verify all fundamental functional requirements. Additionally, we prioritize ensuring user data security and maintaining backend database consistency.

## e. Defect Management

Bugs will be tracked on Pivotal Tracker, along with all of the rest of the project. Defects and bugs in our project will be tracked using Pivotal Tracker, which serves as our primary tool for managing project tasks and issues. Pivotal Tracker allows us to create and prioritize tasks, track their progress, and ensure timely resolution of defects.

*Types of Defects*: We will categorize defects based on severity and impact on functionality, including critical bugs that affect core features, major issues impacting usability, and minor bugs with minimal impact.

*Defect Management Actions*:

* *Identification*: Team members will identify defects during development, testing, or through user feedback.
* *Reporting*: Defects will be reported promptly in Pivotal Tracker, detailing symptoms, environment, and steps to reproduce.
* *Assignment*: Each defect will be assigned to a responsible team member, ensuring accountability for resolution.
* *Prioritization*: Defects will be prioritized based on severity and impact, ensuring critical issues are addressed first.
* *Resolution*: Developers will fix defects following the coding standards and best practices outlined in our project guidelines.
* *Verification*: Testers will verify defect fixes through rigorous testing, ensuring the issue is fully resolved without introducing new problems.
* *Closure*: Defects will be closed in Pivotal Tracker once verified and confirmed as resolved by the QA team.

# References

[Google Java Coding Style](https://google.github.io/styleguide/javaguide.html)

[Gang of Four design patterns](https://springframework.guru/gang-of-four-design-patterns/)

[Git Configuration and Commands](https://www.git-scm.com/)

[Apptegy.com – School alert system](http://apptegy.com)

[Role-Based Access Control: Case Studies and Lessons Learned (secureidentityhub.com)](https://www.secureidentityhub.com/access-control-case-studies-lessons/)

[Building a Notification Framework for Microservice-based Application | by Sohom Majumdar](https://medium.com/walmartglobaltech/building-a-notification-framework-for-microservice-based-application-6fe5ac9dfcee)

[CockroachDB, Distributed SQL database](https://www.cockroachlabs.com/lp/mission-critical-gs/?utm_source=google&utm_medium=cpc&utm_campaign=g-search-na-bofu-pipe-brand&utm_term=e-cockroachdb-c&utm_content=lp660564545020&utm_network=g&_bt=660564545020&_bk=cockroachdb&_bm=e&_bn=g&gad_source=1&gclid=CjwKCAjwmrqzBhAoEiwAXVpgokDf2fGfHIQeyp8TS8FRx4Hs-3j_CfQHPOQavZu-oOAbCY7vLxPadBoChRkQAvD_BwE)

[Auth0, Authentication & authorization platform](https://auth0.com/)

# Glossary

* Auth0: Identity management platform providing authentication and authorization as a service.
* *Single Page Application (SPA)*: Web app that dynamically updates the current page, offering a seamless user experience without full page reloads.
* *PostgreSQL*: Powerful open-source relational database system known for its reliability and support for SQL standards.
* *Microservices*: Architectural style where applications are structured as independent, loosely coupled services.
* *Spring Boot*: Java-based framework for developing standalone, production-grade Spring applications with minimal configuration.
* *RabbitMQ*: Message broker facilitating communication between microservices using the AMQP protocol.
* *API Gateway*: Server handling API requests, enforcing policies like throttling and security, and routing requests to microservices.
* *OAuth 2.0*: Authorization framework enabling third-party applications to access a user's resources without exposing credentials.
* *Swagger/OpenAPI*: Specification for documenting REST APIs, defining endpoints, request/response formats, and authentication methods.
* *Git and GitHub*: Version control system and platform for hosting Git repositories, facilitating collaborative software development.
* *IntelliJ IDEA*: Integrated development environment (IDE) for Java development with advanced coding assistance and productivity tools.
* *CI/CD (Continuous Integration/Continuous Deployment)*: Practices automating software development processes to deliver code changes frequently and reliably.
* *Docker*: Platform for developing, shipping, and running applications in containers for easy deployment and scalability.
* *CockroachDB*: Distributed SQL database providing scalability, resilience, and strong consistency across multiple nodes.