**CS673 Software Engineering** 

**Team 1 - TeamFirst**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| James (Jim) Penner | Team Leader |  |  |
| Shenxiao (Sherry) Li | Requirement Leader | *Sherry Li* | 2024.05.15 |
| Hunter Washburne | Design & Implementation Leader / QA |  |  |
| Jordany Arnaud | Design & Implementation Leader / Security |  |  |
| Chaozeng Zhu | Configuration Leader |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **0** | **Team1** | **5/15** | **Initial draft** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[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 contribution to the Online Learning Platform involves creating a “Notification Service.” This service is important for our project-based learning platform. It helps to share information easily within the university, making communication smoother and increasing student engagement.

**Purpose and Motivation**

In a learning environment where activities and interactions are always changing, it's important to have updates and alerts in real-time. Our goal is to meet this need by creating a strong notification system. This system will give users timely information about important events, changes, or updates related to their roles in the learning platform.

Our purpose is to make communication and information sharing easier among everyone in the university community. We plan to do this by using microservices architecture and RESTful APIs. Our service will be flexible and scalable, allowing notifications to be sent through various channels like SMS and email. 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 notification service will cater to various users within the university community, including staff, program administrators, instructors, and students.

* *Staff members*: They can use the service to stay updated on administrative tasks, like enrollment changes or schedule adjustments.
* *Program administrators*: This group can disseminate information about program-related events or announcements through the service.
* *Instructors*: They can send notifications to students about course materials, assignments, or upcoming deadlines.
* *Students*: Students will receive notifications about their enrollment status, project updates, or group collaboration activities.

**Features**

* *Service Registration*: Microservices in the learning platform can sign up with the notification service to send notifications. The notification service keeps a list of registered services, ensuring only authorized ones can send notifications.
* *Notification Rule Management*: Administrators or authorized users can set up rules for when notifications should be sent. These rules define conditions like event types, who should get the notifications, and how they should be delivered. By setting rules, users can automate notifications, making communication smoother.
* *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. Users can choose who gets the notifications and how they're delivered, whether it's email, SMS, push notifications, or in-app messages. Group notifications make it easy to send updates to multiple people at once.
* *Email Attachment*: Users can include attachments in email notifications, like files or images. This adds more context to the messages and makes them more useful for recipients. Users can upload attachments from their devices or select them from cloud storage services.
* *Notification Tracking*: The service keeps track of notifications sent, recording details like when they were sent, who sent them, who received them, and if they were delivered successfully. Users can check this data to see if notifications were received and troubleshoot any issues. Tracking helps ensure timely communication with users.

**Potential Technology Stack**

* *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.
* *Database: MongoDB* will be our database of choice, offering flexibility and scalability for storing data in a document-oriented format.

In addition to the Notification service, we will also implement an API gateway using Spring Cloud Gateway. 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 enables schools to send alerts and emergency notifications through text messages, voice calls, social media, websites, and apps from a single platform. It offers easy-to-use features like customizing recipients, recording voice calls, and text-to-speech. Thrillshare integrates with various school information systems, ensuring accurate audience targeting and streamlined communication.

However, there are some key differences compared to our notification service:

* Microservice Integration: Our system allows microservices to register and interact with the notification service.
* Rule-Based Notifications: Our system enables administrators to create specific rules for notification timing.
* Customization and Tracking: Our system offers more options for customization (templates) and detailed tracking of delivery and outcomes.
* Preferred Channels: Users can select their preferred notification channels in your system.

The Design pattern we gonna use is the observer pattern, and here is the similar stock project implemented with the observer pattern on Spring Boots. It enables push notifications while someone places an order and it will give notification.

This project only performs self notification while the user doing some action, giving sth like confirmation notification, our project will have something like another user perform some action, notification to its subscribers.

<https://medium.com/geekculture/push-notification-for-order-placed-using-observable-design-pattern-with-spring-boot-websocket-9203874f3ebb>

# Proposed High level Requirements

**Functional Requirements**

*Register a service*

As a microservice developer, I want to register my microservice with the notification service, so that I can send notifications to users within the learning platform.

*Create Notification Rule*

As an administrator, I want to create rules specifying when notifications should be sent, so that users receive timely and relevant information.

*Create template*

As a user, I want to create email notification templates, so that notifications have a standardized format and professional appearance.

*Configure Notification Preferred Channels*

As a user, I want to choose the delivery method for notifications, such as email, SMS, or push notifications, so that recipients receive notifications through their preferred channels.

*Send notification*

As a user, I want to send notifications to specific recipients or groups, so that important information can be shared efficiently.

*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.

**Nonfunctional Requirements**

Security requirements

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

# Management Plan

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*
* Register a Service
* Create Notification Rule
* Create Template

**Iteration 2:**

* Configure Notification Preferred Channels
* Send Notification & View Notification
* Track Notification Delivery Status(Optional Feature)

**Iteration 3:**

* 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.

## a.Timeline & Objectives:

### Iteration Plan

#### **Iteration 1: Core Infrastructure and Basic Features**

Duration: 2 weeks

Goals:

1. Set up the Development Environment (4 hours)
   * Install and configure Spring Boot, RabbitMQ, and MongoDB.
   * Set up version control with Git.
2. Register a Service (10-15 hours)
   * Define API endpoints for service registration.
   * Implement service discovery and registration logic.
   * Integrate with RabbitMQ for messaging.
   * Create unit tests for registration functionality.
3. Create Notification Rule (15-20 hours)
   * Design rule creation REST API endpoints using Spring Boot.
   * Implement rule creation logic and store rules in MongoDB.
   * Integrate rule engine for evaluating conditions.
   * Write unit and integration tests.

Total Estimation: ~30-35 hours

#### **Iteration 2: Template and Channel Configuration**

Duration: 2 weeks

Goals:

1. Create Template (12-18 hours)
   * Develop a REST API for template creation using Spring Boot.
   * Implement template storage and retrieval in MongoDB.
   * Ensure templates support variables/placeholders.
   * Test template rendering.
2. Configure Notification Preferred Channels (10-15 hours)
   * Create UI for channel configuration (frontend technology of choice, e.g., React, Angular).
   * Implement backend logic for storing preferences in MongoDB.
   * Integrate with RabbitMQ for delivering notifications through chosen channels.
   * Test end-to-end functionality.

Total Estimation: ~25-33 hours

#### **Iteration 3: Notification Sending, Tracking, and Viewing**

Duration: 2 weeks

Goals:

1. Send Notification (15-20 hours)
   * Design and implement notification sending API using Spring Boot.
   * Integrate with RabbitMQ for message delivery.
   * Implement batch sending capabilities and store notifications in MongoDB.
   * Test sending notifications to various channels.
2. Track Notification Delivery Status (12-18 hours)
   * Implement tracking logic for each delivery method.
   * Create database schema in MongoDB for storing status updates.
   * Develop a dashboard or API to view status.
   * Test delivery tracking for reliability.
3. View Notification (10-15 hours)
   * Implement a UI or API to retrieve and display sent notification details using Spring Boot.
   * Ensure detailed logging for troubleshooting.
   * Write tests for viewing and logging functionality.
4. Buffer Time for Testing and Bug Fixes (5-10 hours)

Total Estimation: ~42-63 hours

### Optional Features and Nonfunctional Requirements

Optional Features (if time permits):

Send Email Notification with Attachments (10-15 hours)

* + Extend email sending functionality to support attachments.
  + Implement file upload and storage in MongoDB.
  + Ensure attachment security and size constraints.
  + Test email notifications with various attachments.

**Nonfunctional Requirements** (spread across iterations):

Security Requirements (20-25 hours)

* + Configure Spring Security for the application.
  + Set up OAuth2 authentication with Keycloak.
  + Implement role-based access control.
  + Conduct security testing and code reviews.

## b. Risk Management (need to be updated constantly)

(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. Please provide the link to the sheet.)

**Risk Management Sheet Link:**

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration) See above iteration timeline for more details.

# Configuration Management Plan

## Tools

We will use git and github for version control

Intellij will be IDE tool

CI/CD will use github action

Docker as container

sast/dast TBD

* 1. Code Commit Guideline and Git Branching Strategy

1. Each person should have a sub branch ( normally is their own name)
2. Each developer link their local code with our cloud one use code:

git remote add upstream https://github.com/BUMETCS673/project-teamfrist

1. Every time begin coding, make sure pull code from upstream to main branch

Main branch should onlys used to pull, push is based on your name branch

git pull upstream main

1. Then align name branch code same as main branch

Git merge main

1. When completed feature,
   1. Git add.
   2. Git commit -m’ your change message’
   3. Git push origin <name-branch\_defined\_by\_yourself>
2. Go to github repo page, find and click the pull request button
3. Click and send pull request
   1. Deployment Plan if applicable

(If you plan to deploy your application (e.g. your web application), briefly describe how you plan to deploy your application).

# Quality Assurance Plan

## Metrics

| Metric Name | Description |
| --- | --- |
| Complexity | Number of lines of code written, with fewer lines meaning lower complexity which is good |
| Cost | # of person-hours expended, with fewer hours meaning lower cost which is good |
| Stories | Number of user stories completed, with more stories meaning more value provided for the customer which is good |
| Bugs/LoC | Bugs found per LoC in each release |
| Dependencies | Number of dependencies (other than Java Standard Library) used (fewer is better) |

Coding Standard

Our team will be using the Google Java code style, with Java standard naming conventions:

<https://google.github.io/styleguide/javaguide.html>

<https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>

We will discuss and possibly add to this section architectural considerations such as the project package structure.

## Code Review Process

Owner of each feature writes code, then weekly we meet and owners explain their code and everyone comments on Zoom (recorded).

## Testing

For unit testing we will be using JUnit5, with webclient or RESTassured to write the tests. Every developer will write unit tests, while the QA lead (Hunter) will focus more on functional and end-to-end (integration) testing.

Unit tests will be run daily. Functional testing will be run as components are brought into a somewhat feature complete state. Integration testing (probably using Postman) will be performed as Goothe other teams (UI, etc.) start to deliver their beta products.

The primary objective for testing is to ensure that the basic functional requirements are met. The next objective is to maintain security of user data and consistency of the backend database.

## Defect Management

Bugs will be tracked on Pivotal Tracker, along with all of the rest of the project.

# References

Google github style guide

Gang of four design patterns

aboutgitlab.com - git configuration and other methods

Apptegy.com – school alert system

# Glossary

JUnit5: Latest iteration of testing framework for Java and JVM;

REST: REpresentational State Transfer: architectural style for providing standards between computer systems on the web.

RESTAssured: Open source test automation tool for testing the REST services