CS673 Software Engineering

Team 1 – Get Active App

Project Proposal and Planning

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Jin Hao Li | Design and Implementation Lead, Team Lead | *Jin Hao Li* | 5/10/2025 |
| Jianing Li | Requirement Lead | *Jianing Li* | 5/10/2025 |
| Xiaoyao Yin | QA Lead | *Xiaoyao Yin* | 5/10/2025 |
| Arshdeep Dhillon | Security Lead  Design and Implementations  Team Lead | *AD* | 5/17/2025 |
| Shaohua Yue | Configuration Lead | *Shaohua Yue* | 5/10/2025 |
| Jie Shi | Requirement Lead | *Jie Shi* | 5/13/2025 |
| Xiaoyao Yin | QA Lead | *Xiaoyao Yin* | 5/15/2025 |

**Revision history**

| Version | Author | Date | Change |
| --- | --- | --- | --- |
| 1 | All team members | 5/10/2025 - 5/14/2025 | Filled the document for Iteration 0 |
| 2 | Jin Hao Li | 6/8/25 | Deleted blank page and Updated Timeline |
| 3 | Jin Hao Li | 6/9/25 | Updated Timeline |
|  |  |  |  |

[Overview 3](#_heading=h.ok1fg7e9dzfm)

[Related Work (Jianing) 4](#_heading=h.2qq070n0vhtw)

[Summary of Differentiation 4](#_heading=h.ukyywsnr28tn)

[Proposed High level Requirements (Jianing) 5](#_heading=h.3zpc00ezl8yk)

[Functional Requirements 5](#_heading=h.vloc48bm64pl)

[Essential Features 5](#_heading=h.7iq7uu6nll4i)

[Desirable Features 6](#_heading=h.5gv6wi9ymovx)

[Optional Features 6](#_heading=h.cysfd3n41iqw)

[Nonfunctional Requirements 7](#_heading=h.v6e6a4o7jc22)

[Management Plan (Jin Hao Li) 8](#_heading=h.p92thcv5mt6l)

[Objectives and Priorities 8](#_heading=h.h179zu2gvmye)

[Risk Management 8](#_heading=h.eiekb2n26p34)

[Timeline 9](#_heading=h.big18tklc5bc)

[Configuration Management Plan (Shaohua Yue) 10](#_heading=h.tfsx4x3mp8uj)

[Tools 10](#_heading=h.62runpm7ab69)

[Code Commit Guideline and Git Branching Strategy 10](#_heading=h.83bpt5o1l8so)

[CI/CD Plan 11](#_heading=h.oskiemo5ops)

[Quality Assurance Plan (Xiaoyao Yin) 12](#_heading=h.j9b2zimrs0l8)

[Metrics 12](#_heading=h.j48yhvauh73u)

[Coding Standard 12](#_heading=h.kcf6t3bgjkpi)

[Code Review Process 13](#_heading=h.c035l6gq12qi)

[Testing 14](#_heading=h.uvlsvdqlac79)

[Defect Management 15](#_heading=h.80l5xup95lz5)

[Tool for Defect Tracking 15](#_heading=h.3xfv0wivceag)

[Defect Tracking Process 16](#_heading=h.jlqqepaxirf)

[Types of Defects 16](#_heading=h.lcppti2etsq7)

[Defect Resolution Workflow 16](#_heading=h.lzy19ju6j09x)

[Defect Resolution Workflow 16](#_heading=h.2iwhnbh70pw)

[AI usage Log 17](#_heading=h.vyq9qy9q1ymj)

[References 17](#_heading=h.4ss4fax6y63y)

[Glossary 17](#_heading=h.elz0zkafsty5)

# 

# 

# 

# 

# Overview

(Please give an overview of your project. It should include the motivation, the purpose and the potential users of the proposed software system, the basic functionality of the proposed software system and the possible technology stack to be used.)

As students, we often become so focused on our studies that we overlook the incredible opportunity to build new friendships. To address this, we are developing a student activity engagement platform called “**GetActive**”, designed to encourage students to stay active and socially connected by participating in group activities. The motivation behind this project comes from the realization that students frequently struggle to find simple and effective ways to join social or recreational groups. *GetActive* aims to bridge this gap by offering a dedicated space for students to discover, create, and join various on-campus activities, ultimately helping them make new friends and foster a stronger sense of community.

The primary users of the system are university students, who will first register using their BU email and a chosen password through a registration page. A verification email will be sent to their email and once the user accepts it, they can log in using their email and password. After logging in, users can browse and join existing activities or create their own. When a user creates an activity, they automatically become its admin. Admin users can edit the activity name and delete their own activities. Users can join or leave multiple activities, regardless of their role. The system supports two types of users: normal (guest) users and activity admins, with smooth transitions between these roles based on user actions.

The platform will include both a registration page and a simple login page to support secure user access. The technology stack for *GetActive* includes a React frontend, a Spring Boot backend, and a relational database to manage user and activity data. Secure authentication will be implemented using JWT protocol. By making it easier for students to connect with others through shared interests, *GetActive* seeks to enrich the university experience and promote a more engaged campus life.

# 

# Related Work (Jianing)

(Please describe any similar software systems that you have found through the online research, and the differences between your software and those software systems.)

| **Platform** | **Core Positioning & Key Features** | **Principal Differences from GetActive** |
| --- | --- | --- |
| CampusGroups (Ready Education) | Turn-key campus life suite: single sign on, chat, event tickets/payments, enterprise analytics, mobile app, alumni networking. | Top down license sold to the entire institution; heavier footprint (payments, directories, CRM). GetActive aims for lightweight, student driven deployment with no billing or alumni functions. |
| Meetup / Facebook Groups / Discord | General purpose social or chat tools used informally by students. | Lacks BU SSO, role based admin controls, and on campus discovery UI. GetActive offers university verified identities plus purpose built roster & admin flows. |
| Anthology Engage | Enterprise “student involvement” hub tied to Anthology’s LMS/CRM stack, deep reporting, form workflows. | Requires institutional CRM integration and often multi year contracts. GetActive requires no LMS tie-in and can be piloted by a single club. |

## 

## Summary of Differentiation

* **Deployment Model**: GetActive can be adopted bottom up by one student org without a campus wide license, unlike most commercial suites.
* **Privacy by Design**: Public display names; BU ID never shown to peers, whereas other platforms often expose directory info or build public skill transcripts.
* **Feature Minimalism**: Core create/join/roster workflow ships in < 6 weeks; enterprise suites bundle payments, competencies, or AI that raise cost and complexity.

# 

# 

# Proposed High level Requirements (Jianing)

**Estimates include design, meetings, coding, add unit tests, testing, deployment(feature and production).**

**Estimate in Days can be converted to person-hours assuming this is a full time job with 8 hours per day.**

## Functional Requirements

### Essential Features

| Number | Feature Title | User Story | Est. Hours |
| --- | --- | --- | --- |
| E-1 | Account Registration | As a new student, I want to create an account with my BU email address, display name, and password, so that I can access the platform with an identity that’s shown publicly by name but secured by my BU email. | 16 hours |
| E-2 | Secure Login | As a returning student, I want to log in with my BU email and password, so that I can access my activities and profile securely. | 16 hours |
| E-3 | Activity Discovery | As a student, I want to browse activity cards (title, date/time, location) and search or sort them, so that I can quickly find events that interest me. | 16 hours |
| E-4 | Create Activity | As a student, I want to create a new activity and automatically become its admin, so that I can organize events for others. | 16 hours |
| E-5 | Join / Leave Activity | As a student, I want to join or leave any number of activities independently with one click so that I can manage my commitments easily. | 24 hours |
| E-6 | Roster Display | As an activity participant, I want to see a list of joined students by display name, so that I know who else is attending. | 24 hours |
| E-7 | Admin Edit & Save | As the activity admin, I want to edit details inline and save them, so that information stays current. Specifically, I want to be able to edit the time of the activity, the location, description and the name of the activity. | 24 hours |
| E-8 | Delete Activity | As the activity admin, I want to delete my activity with a confirmation dialog that shows the current join count, so that I can responsibly cancel events. Guests will not be notified when the activity is cancelled. | 24 hours |

### Desirable Features

| Number | Feature Title | User Story | Est.Days |
| --- | --- | --- | --- |
| D-1 | Popularity Sort | As a student, I want to sort activities on the home page by number of participants, so that I can join high-energy events. | 8 hours |
| D-2 | Report Activity | As a student, I want to report an inappropriate activity with a reason and description, so an email with the message can be sent to the admin who created the activity. | 16 hours |
| D-3 | User Profile | As a logged-in student, I want to see my profile avatar and my username, so that I’m sure I’m using the correct account. | 16 hours |
| D-4 | User Avatar | As a logged-in student, I want to upload my own profile avatar. | 16 hours |
| D-5 | Activity Comments | As a participant, I want to post questions or comments under each activity to clarify event details or connect with other participants. | 16 hours |

### Optional Features

| Number | Feature Title | User Story | Est. in Days |
| --- | --- | --- | --- |
| O-1 | Email / Push Notifications | As an activity participant, I want notifications when an activity changes, so that I don’t miss important updates. | 16 hours |
| O-2 | Max Capacity & Wait-List | As the activity admin, I want to cap attendance and auto-promote from a wait-list, so that events don’t overfill. | 24 hours |
| O-3 | Admin can remove guests | As the activity admin, I want the ability to remove guests from my activity so that I can manage who can participate. | 16 hours |
| O-4 | Unsaved-Change Guard | As the activity admin, I want a warning if I try to navigate away with unsaved edits, so that I don’t lose work by accident. | 8 hours |
| O-5 | Notify Guests | As an activity participant, I want to receive email notification when the activity is cancelled/deleted so that I can stay informed with activity info. | 24 hours |
| O-6 | Captcha on Registration | As the security team, I want a captcha on sign-up, so that bots can’t mass-create accounts. | 16 hours |
| O-7 | Event Feedback | As an activity participant, I want to leave a rating and feedback after an event so organizers can improve future events. | 8 hours |

## Nonfunctional Requirements

| Number | Category | Feature Title | User Story | Est. Days |
| --- | --- | --- | --- | --- |
| N-1 | Security | Monitor Code | As a developer I want to set up a SonarQube application so that I can monitor the quality of the code. | 24 hours |
| N-2 | Coding Style | Grammar Checker | As a devOps developer, I want to have a unified styled code. So that I can reduce complications and bugs | 16 hours |
| N-3 | Security | Protect Against Bad Actors | As a developer I want to add validation on all user entered fields so that I can prevent injection attacks. | 24 hours |
| N-4 | CI/CD | Feature branch pipeline | As a devOps developer, I want to create a feature branch pipeline to run tests, linter and produce artifacts. | 24 hours |
| N-5 | CI/CD | Main pipeline | As a devOps developer, I want to create a feature branch pipeline to run tests, linter, produce artifacts and deploy. | 24 hours |
| N-6 | CI/CD | Production Environment | As a devOps developer, I want to set up a production environment | 24 hours |
| N-7 | Performance | Performance Benchmarking | As a developer, I want to ensure the platform loads under 2 seconds for 90% of interactions so that users don't abandon the site. | 8 hours |

# 

# Management Plan (Jin Hao Li)

## Objectives and Priorities

(Please describe your project objectives with highest priority first. Project Goals can include but not limited to complete all proposed (essential) features, deploy the software successfully, the software has no known bugs, maintain high quality, etc )

The objective is to complete all proposed essential features with little or no known bugs. The software is able to deploy successfully to production.

## 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. Please provide the link to the sheet.)

Our team has identified several key risks that may impact the development of the *GetActive* platform. The first major risk is **time constraint**, as balancing coursework and project deadlines may limit the time available for implementation and testing. Another significant risk is **limited experience with React**, which could slow down frontend development as team members familiarize themselves with the framework. Additionally, we recognize a risk in our **unfamiliarity with Spring Security**, which is essential for securing backend endpoints and user authentication. To mitigate these risks, we have allocated time for focused learning sessions, plan to utilize official documentation and tutorials, and will follow a structured development timeline to ensure steady progress.

**Risk Management Sheet Link:** [CS673\_SPPP\_RiskManagement\_team1](https://docs.google.com/spreadsheets/d/1IYY1RN1G-WXDme-pEVlGNWM36LWxAaBh/edit?gid=1559991977#gid=1559991977)

## 

## 

## Timeline

(this section should be filled in iteration 0 and updated at the end of each later iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 1(2 weeks) | Ci/CD pipeline: N4-N6(72)  First Essential Features: E1-E4(64) |  | 136 hours/99 hours |
| 2 (2 weeks) | Last 4 Essential Features: E5-E8(96 hours)  E-6 is moved over to Iteration 3 since it is not done yet. |  | 172 hours/109.06 |
| 3 (1 weeks) | E-6(6)  Desirable Features(72 hours) |  | 78 person hours/ |

# 

# 

# 

# 

# Configuration Management Plan (Shaohua Yue)

## Tools

(In this project, we will use Git and Github as the version control tools. Please also specify any other tools to be used, e.g. IDE tools, CI/CD tools, container tools, SAST or DAST tools, and any other DevOps tools and **AI tools**)

Version control tools: Github

CI/CD tools: Github Action, Docker for building our frontend and backend.

IDE tools: Jetbrain, VScode, Curso

AI tools: ChatGPT, Claude

## Code Commit Guideline and Git Branching Strategy

(Please briefly describe criteria for the code commitment and the branching strategy used, e.g. what are the branches to be used, how the pull request will be used etc. Here is an article to give you some basic knowledge about different git branching strategies: <https://www.flagship.io/git-branching-strategies/>)

Git Branching Strategy: Github Flow

Code Commit Guideline:

1. Commit Message Format

Please follow the format below for each commit:

<type>(<scope>): <subject>

1. type: Type of the commit (required)
2. scope: Scope of the changes (frontend/backend/both)
3. subject: Brief description (required, within 50 characters)

Common Types

1. feat: New feature
2. fix: Bug fix
3. docs: Documentation changes
4. style: Code style changes (formatting, missing semi colons, etc.)
5. refactor: Code refactoring (neither new feature nor bug fix)
6. test: Adding or modifying tests

Examples

1. feat(login): add user login feature
2. fix(api): fix bug in user info API
3. Commit Requirements
   1. Each commit should focus on a single purpose; avoid mixing unrelated changes in one commit.
   2. Do not commit build files, dependencies, or sensitive information.
   3. **As we are using GitHub Flow, please pull the latest master branch and update your development branch before committing, ensuring your branch is up to date.**
   4. Make sure all lint checks and unit tests pass locally before committing.
4. Use Git Hooks to enforce code commit guidelines
5. Implement a bash script.
6. Modify Devcontainer’s configuration file to set up the git hooks when the container starts.

## CI/CD Plan

(Briefly describe how you plan to continuously integrate and deploy your application).

CI Plan:

Implement ci.yml,

1. Run linter/code analyze workflow on backend and frontend.
2. Build, package and deploy.
3. Deployment order: Database, Backend, Frontend.
4. Run tests by docker compose command.

CD Plan:

1. Triggered by tag on any commit.

# 

# Quality Assurance Plan (Xiaoyao Yin)

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Lines of Codes (LOC) | Measuring the total number of code written, used to track development progress |
| Test Case Pass Rate | Percentage of executed test cases that pass |
| Number of User Stories Completed | Tracks the number of completed user stories during each iteration. |
| Code Review Comments per PR | Number of meaningful comments per pull request |
| Defect Density | Number of bugs reported per 1,000 lines of code (KLOC). |

## Coding Standard

(Describe any coding standard to be used)

We will follow **industry-standard best practices** for each technology stack:

* **Frontend (React)**
  + Follow [Airbnb JavaScript Style Guide](https://github.com/airbnb/javascript)
  + Use ESLint and Prettier for formatting and linting.
* **Backend (Spring Boot)**
  + Follow [Google Java Style Guide](https://google.github.io/styleguide/javaguide.html)
  + Naming conventions for classes, methods, and variables.
  + Proper use of annotations and dependency injection.
* **Database**
  + Use consistent naming (snake\_case for table and column names).
  + Ensure primary/foreign keys and indexing follow best practices.

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

All code contributions will be submitted through GitHub pull requests (PRs). Each PR must be reviewed and approved by at least one other team member before merging into the main branch.

To ensure a consistent and enforceable review process, we will implement the following GitHub branch protection rules:

1. A GitHub pull request must be created for every change intended for the main branch — no direct commits will be allowed.
2. A pull request must receive at least one approval from another team member before it can be merged.
3. If there are any open discussions on a pull request, it cannot be merged until all conversations are resolved.
4. These rules will be enforced using GitHub’s branch protection settings, which will be configured at the repository level.

* **Reviewers**
  + Team members will review each other’s code for routine features and updates to ensure shared understanding and maintain code quality. For more complex or critical areas—such as authentication, database logic, or other security-sensitive modules—the implementation or design lead will conduct the review. This ensures that high-risk changes receive expert scrutiny, while also promoting knowledge sharing and accountability across the team.
* **Review Checklist**

The following checklist will be used when verifying the code reviews:

* Is the code logically correct and functionally complete.
* Are variable/method names meaningful and consistent.
* Does the code follow the agreed-upon coding standards.
* Unnecessary or duplicated code is not present.
* Is code secure and efficient.
* **Review Tools**
  + We will use GitHub's built-in review and comment features to conduct code reviews. This includes line-by-line comments, pull request discussions, and the ability to approve or request changes. These tools help facilitate clear communication, maintain a record of feedback, and ensure that all code changes are thoroughly reviewed before merging.
* **Feedback Style**
  + By praising well-written and clever code, it creates a friendly and encouraging learning environment. Positive reinforcement helps build confidence and motivates continued effort. At the same time, using clear, constructive comments and suggestions for improvement ensures that feedback is actionable and educational. This balanced approach supports both growth and morale, making it easier for learners to embrace challenges and refine their skills.

## 

## Testing

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

We will conduct both manual and automated testing to ensure code quality, functional correctness and overall system reliability. Our application consists of a backend built with the Spring Framework and a frontend built using React, so our testing approach will cover both layers accordingly.For the backend, we will use JUnit and Mockito to perform unit testing on individual components such as services, controllers and repositories. For the frontend, we will rely on JEST for unit and component-level testing to ensure the correctness of UI logic. Each developer will be responsible for writing and maintaining unit tests for their own code. The QA leader will take charge of coordinating integration and E2E testing, ensuring that different modules interact as expected and that system-level functionality is thoroughly tested. Manual testing will also be used to validate usability, UI consistency, and edge cases that automated tests may not cover.Testing will be an ongoing part of the development lifecycle, therefore developers will run unit tests during development and automated test suites will execute on every pull request through continuous integration tools such as GitHub Actions. Before each release of the iteration, a full regression and system test will be performed to catch any unforeseen issues and verify that the application remains stable.

* **Types of Testing**

To ensure comprehensive quality assurance, we will employ a mix of unit, integration, manual, and automated testing throughout the development process:

* + Unit Testing
    - For the backend, JUnit will be used to test individual components and services in the Spring application.
    - For the frontend, Jest will be used to test React components and logic.
  + Integration Testing
    - The QA lead will oversee integration testing to verify that modules interact correctly. This includes testing API interactions between the frontend and backend to ensure smooth data exchange and system behavior.
  + Manual Testing
    - Manual testing will focus on the user interface and end-to-end user journeys. This helps identify issues related to usability, layout, and workflows that automated tools may overlook. Exploratory testing and acceptance testing will also be part of this phase.
  + Automated Testing
    - Both frontend and backend automated test scripts will be incorporated into our continuous integration (CI) pipeline using GitHub Actions. This ensures that tests are executed consistently on every pull request, helping catch regressions early and maintain stability.
* **Testing Tools**
  + Backend: JUnit, Mockito
  + Frontend: Jest, React Testing Library
  + API Testing: Postman or Insomnia
* **Testing Objectives**

Our testing efforts are guided by the following key objectives:

* Validate feature requirements:
  + Ensure that all the implemented features align with the functional and non-functional requirements defined in the project scope.
* Identify edge case failures early:
  + Catch any potential unusual issues during the unit and integration testing to minimize the runtime errors and improve the robustness of the application.
* Ensure stable end-to-end workflows:
  + Verify that the core user flows such as user registration, user login, activity creation, activity deletion, etc work reliably across the frontend and the backend.

## 

## Defect Management

(Describe the tool to be used to manage the defect (e.g github issues). The types of defects to look at. The actions or personnel for defect management. )

Managing defects is critical to maintaining the quality and reliability of the *GetActive* application throughout the development lifecycle. This section outlines the tools, types of defects to monitor, and the workflow we will follow to ensure that bugs are tracked, resolved, and verified efficiently.

### Tool for Defect Tracking

To track and manage defects, we will use GitHub Issues. This tool provides a centralized platform for reporting, assigning, and tracking bugs and enhancements, ensuring team-wide visibility and accountability.

### Defect Tracking Process

All defects will be reported as issues on GitHub. Each issue will include a clear title, detailed description, steps to reproduce the problem, severity level, and supporting materials such as screenshots or logs when applicable. The **team lead** will review each report and assign it to a suitable developer based on expertise and workload. Once a fix is implemented, it will be submitted as a pull request (PR), and the **QA lead** will perform regression testing before closing the issue.

### Types of Defects

We will focus on identifying and resolving the following categories of defects:

* **Functional Bugs**: Issues that prevent users from joining or creating activities.
* **UI Issues**: Layout problems, unresponsive elements, or inconsistent user interface behaviors.
* **Security Flaws**: Unauthorized access to sensitive data or protected endpoints.
* **Performance Issues**: Slow load times or backend inefficiencies that affect user experience.

### 

### Defect Resolution Workflow

1. Identify and document the issue in the GitHub Issues.
2. Assign the issue to a developer based on expertise and availability.
3. Implement a fix and push the changes via a pull request (PR).
4. The QA lead performs verification via testing.
5. The QA lead closes the issue after successful testing.

### Defect Resolution Workflow

To ensure a consistent and thorough approach, we will follow this workflow:

1. Log the defect in GitHub Issues with all necessary details.
2. Assign the issue to an available developer.
3. Apply the fix and push it via a pull request (PR).
4. Conduct verification through QA testing.
5. Close the issue upon successful resolution and validation.

# 

# AI usage Log

(You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.)

| Tools | Who | Tasks | helpful | Evaluation/modification | links |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 

# 

# 

# References

(Any references/citations that you have used)

* <https://code.visualstudio.com/docs/devcontainers/containers>
* <https://docker-curriculum.com/>

# 

# Glossary

(Any acronym used in the document should be explained here)