**CS673 Software Engineering** 

**Team 2 - CareerForge**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Gopi Rayini | Configuration Leader | *GR* | 9/7/25 |
| Pedro Ramirez | Design and Implementation | *PR* | 9/7/25 |
| James Rose | Team Leader | *JR* | 9/7/25 |
| Qi Chen | Requirement Leader | *QC* | 9/7/25 |
| Stacey Burns | QA Leader | *SB* | 9/7/25 |
| Yongxiang Chen | Security Leader | *YC* | 9/9/2025 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0.0** | **Gopi, Pedro, James, Qi Chen, Stacey** | **09/7/25** | **Initial Draft** |
| **2.0.0** | **Stacey** | **09/21/2025** | **Updates based on feedback from Iteration 0:**   * **Related work section** * **Code Commit Guideline and Git Branching Strategy section** * **CI/CD plan** * **Timeline** |
| **3.0.0** | **Stacey** | **10/06/2025** | **Updated based on feedback:**   * **Added graph to related work section** * **CI/CD plan** * **Timeline** |
| **4.0.0** | **Qi Chen** | **10/11/2025** | **Finalized the document:**   * **Timeline** |
| **5.0.0** | **Stacey Burns** | **10/12/2025** | **Added details to what was completed in Iteration 3 timeline** |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management (need to be updated constantly)](#_tk7yixobah8p)

[Timeline (need to be updated at the end of each iteration)](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_dzly5b9kz982)

[Deployment Plan if applicable](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Code Review Process](#_hx3eaiwb8v3m)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[References](#_pd9euov6m4du)

[Glossary](#_ty3i2nqffhtc)

# Overview

This project is a job search application aimed at helping individuals stay organized in their career search by saving, applying to, and tracking jobs in one place. The motivation is to provide users with a simple system to lead on applied jobs instead of relying on scattered tools. Its purpose is to connect employers and employees, with potential users being anyone on the job market. Core functionality includes creating user accounts, viewing and searching job posts, saving and applying to jobs, and tracking the status of applications. The proposed technology stack for the system is React with TypeScript for building the user interface, supported by a Java backend that manages job posts, authentication, and application tracking.

# Related Work

Through our research, we identified several established job search and application-tracking platforms:

* **LinkedIn Jobs** – Provides a large professional network with job postings, “Easy Apply,” and a personal “Applied Jobs” tracker. It also monetizes through sponsored job ads, which are given higher visibility.
* **Indeed** – Offers a broad job board and a “My Jobs” feature for saving and tracking applications. However, tracking is limited when applications are submitted on external employer sites, and the platform relies heavily on paid postings and ads.
* **Glassdoor** – Combines job search with company reviews and salary insights. Candidates can save and track applied jobs, but its interface is more complex because of the added focus on employer branding and advertising.
* **ZipRecruiter** – Focuses on ease of applying (“1-Click Apply”) and distributing resumes widely. The platform prioritizes employer-sponsored listings and marketing features.

**CareerForge distinguishes itself from these existing platforms in the following ways:**

* **No Advertisements or Sponsored Jobs**Unlike LinkedIn, Indeed, or ZipRecruiter, CareerForge does not surface paid placements. Every job is displayed on equal footing, making results more unbiased and transparent.
* **Free of Charge**While major platforms are free for job seekers, they often monetize indirectly (ads, upsells, premium tiers). CareerForge is entirely free of charge and free of hidden monetization, ensuring accessibility for all users.
* **Streamlined Application Tracking**CareerForge emphasizes first-class tracking of application status across all jobs, regardless of where the application originated. This provides a more reliable and centralized timeline of progress compared to LinkedIn or Indeed, which offer only partial or manual tracking for off-site applications.
* **Simplicity and Focus**Existing platforms include social networking, employer branding, AI-driven discovery tools, and other add-ons that can overwhelm users. CareerForge intentionally avoids these layers to deliver a clean, simple, and distraction-free experience focused only on job search organization and tracking.

| **Platform** | **Ads/Sponsored Jobs** | **Off-Site Tracking** | **Cost for Job Seekers** | **Focus Area** |
| --- | --- | --- | --- | --- |
| LinkedIn Jobs | Yes | Partial | Free + Premium Up sell | Networking and Branding |
| Indeed | Yes | Limited | Free with ads | Broad Job Board |
| Glassdoor | Yes | Limited | Free | Reviews and Salary Insights |
| ZipRecruiter | Yes | Very Limited | Free with ads | Resume Distribution |
| CareerForge | No | Yes (all jobs) | Completely Free | Tracking and Simplicity |

While existing systems optimize for employers, ads, or premium tiers, CareerForge is designed solely for job seekers. It provides a free, unbiased, and simple way to manage the often chaotic process of job searching and application tracking.

# Proposed High level Requirements

* 1. Functional Requirements  
     (For each functional requirement, please give a feature title and a brief description using the following format: As (a role), I want to (action), so that (value).)
     1. Essential Features (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or a range of person hours)

1. ***Requirement 1:*** As a potential employee, I want to be able to create an account so that I can login - 40 person hours
2. ***Requirement 2:*** As a potential employee, I want to be able to enter and save a job title and employer so that I can track and monitor the status - 20 person hours
3. ***Requirement 3:*** As a potential employee, I want to be able to search for a job using job title so that I can apply for it. Will be using mock data - 10 person hours
   * 1. Desirable Features (the nice features that you really want to have too):
4. ***Requirement 4:*** As a potential employer, I want to be able to create job postings for people to apply to - 30 person hours
5. ***Requirement 5:*** As a potential employer, I want to be able to delete job postings for people to apply to - 30 person hours
   * 1. Optional Features (additional cool features that you want to have if there is time):
        1. ***Requirement 1:*** As a potential employee, I want to add personal notes (deadlines, contacts, comments) for each job to help organize thoughts - 40 person hours
        2. **Requirement 2:** As a potential employee, I want to be able to upload my resume so that I can attach that to a job application - 20 person hours
   1. Nonfunctional Requirements
      1. Security requirements
         1. **Requirement 1:** Passwords must be hashed
         2. **Requirement 2:** MFA

# Management Plan

## 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 primary objective of this project is to build a platform that connects employers and employees through job listings. The highest priority in the first iteration is to implement secure user account creation for employees, followed by creating a database with mock data for job postings so the employees can view them. Later iterations will expand the system to include job applications, search and filtering, notifications, and eventually advanced features like user profiles, dashboards, and job recommendations. The overall goal is to deploy a working, high-quality system with no major bugs that supports smooth interaction between job seekers and employers.

## 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.)

The main risks identified for this project include potential technical issues such as security vulnerabilities in user accounts, bugs in job listing functionality, and integration challenges between the Java backend and JavaScript frontend. There is also a risk of scope creep if too many features are added early, as well as time management risks if tasks are underestimated. To manage these risks, we will prioritize building and testing core features first, perform regular code reviews and testing to catch bugs early, and use an incremental iteration plan to keep the project on track. Risks will be reassessed after each iteration to ensure issues are addressed promptly.

**Risk Management Sheet Link:** [**https://docs.google.com/spreadsheets/d/1CwiQTPFikWo0synuCMQeg9UZLMs-61kC3Y73ReoJNI8/edit?usp=drive\_link**](https://docs.google.com/spreadsheets/d/1CwiQTPFikWo0synuCMQeg9UZLMs-61kC3Y73ReoJNI8/edit?usp=drive_link)

## 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 total person hours |
| --- | --- | --- | --- |
| 1 | -Project skeletons (backend and frontend running locally)  1. Build login/register components  2. Definition and maintenance of database schema  3. Set up component library  4. Set up Docker and Create Dockerfile  5. Initialize React and TypeScript project  6. JIRA and workflow setup  7. Initialize Java Spring Boot  8. GitHub Actions for deployment set up  9. Set up SAST tools  10. Implement User Authentication  11. Create class diagram  12. Integrate layout components into main app  13. Create APIs for User registration/login  14. Create aside component with filters and theme toggle  15. Set up APIs for saving/retrieving jobs  16. Create jobsview with jobcard components and API integration | 1.1 Build Login form  1.2 Build Register form  1.3 Add basic validation  1.4 Task 4: Style forms using Bootstrap  2.1 Initialize AWS RDA DB Instance  2.2 Design DB schema  2.3 Create Migration Scripts  2.4 Implement models in backend code  2.5 Setup schema versioning  2.6 Write unit tests for models  3.1 Configure Bootstrap styles and create style files  3.2 Create layout wrapper component  3.3 Create reusable header component and footer  3.4 Create reusable button components  5.1 Create Vite React + TypeScript project  5.2 Install dependencies  5.3 Organize project folder structure  5.4 Verify local run and push to GitHub  8.1 Create GitHub actions workflow file  8.2 Configure deployment environment  8.3 Write deployment steps in workflow  8.4 Test deployment pipeline  8.5 Integrate with CI/CD Pipeline  9.1 Evaluate and Select SAST Tool  9.2 Configure SAST Tool locally  9.3 Integrate SAST into CI/CD pipeline  9.4 Set up reporting and alerts  9.5 Define security rules and thresholds  9.6 Test and Validate Integration  10.1 Configure spring security and JWT filters  10.2 Protect endpoints with authentication middleware  10.3 Write tests  12.1 Integrate Header, JobList, Aside and Footer  12.2 Enhance the design  13.1 Implement/register endpoint with validation  13.2 Add error handling and unit tests  14.1 Build aside component  14.2 Add filter buttons  14.3 Build theme toggle button  15.1 Define job entity and SQL schema  15.2 Implement REST endpoints  15.3 Write integration tests  16.1 Build JobCard Component  16.2 Build JobsView Layout  16.3 Integrate Rise Jobs API  16.4 Add Loading, empty, and error states | **Estimated**: 40 person hours  **Real person hour**s: 267  **Link to Week 1 JIRA Requirements and Tasks:** <https://bu-team-l7qpgfrr.atlassian.net/browse/SCRUM-5>  **Link to Week 2 JIRA Requirements and Tasks:** <https://bu-team-l7qpgfrr.atlassian.net/browse/SCRUM-98> |
| 2 | -Collaborate with frontend for API integrations  1. Implement error handling and response standardization  2. Setup Postman collection for API testing  3. Test Suite for Auth, MyJobs, and Jest framework  4. Fetch and Manage MyJobs  5. Collaborate with frontend API integrations  6. Refactor and clean up code to fit UML design  7. Fix and enable unit tests  8. Add testing to frontend  9. Set up selenium framework  10. Connect frontend to backend (login, register, content) | 1.1 Define global exception handler in Spring Boot  1.2 Create a common API response format  1.3 Write tests  2.1 Create postman collection  2.2 Add environment variables  2.3 Document instructions  3.1 Install Jest  3.2 Login and Registration Pages for testing  3.3 API client for testing  3.4 401/404 routing components for testing  3.5 Update README  4.1 Extend API client with MyJobs methods  4.2 Dropdown toggle for save and applied  4.3 Fetch and render list  4.4 Delete single saved job  4.5 Delete single applied job  4.6 Bulk deletes  4.7 Unit tests  5.1 Setup test data in DB  5.2 Integrate API fields with frontend | **Estimated**: 20 person hours  **Real person hour**s: 247.5  **Link to Week 3 JIRA Requirements and Tasks:** <https://bu-team-l7qpgfrr.atlassian.net/browse/SCRUM-167>  **Link to Week 4 JIRA Requirements and Tasks:** <https://bu-team-l7qpgfrr.atlassian.net/browse/SCRUM-200> |
| 3 | Clean Up, get frontend deployed using CI/CD pipeline, implement more tests   1. Fix Aside bug within MyJobs 2. Connect frontend to backend 3. Setup/Merge Selenium Framework 4. Add testing to frontend 5. Fix and enable unit tests | 1.1 Extend Field and Type options to My Jobs filter  2.1 Build typed API client and auth header injection  2.2 Implement Login Flow  2.3 Implement Register flow  2.4 Save job action  2.5 Apply job action  3.1 Install Selenium and dependencies  3.2 Configure Test Environment  3.3 Write test cases  4.1 Testing for RegisterForm  4.2 Testing for Field.tsx  4.3 Testing for MyJobsButton.tsx  4.4 Testing for Aside.tsx  4.5 Testing for [validation.ts](http://validation.ts)  5.1 Fix errors existing in unit tests  5.2 Add new unit tests for complete cover coverage | **Estimated**: 40 person hours  **Real person hour**s: 174  **Link to Week 4 JIRA Requirements and Tasks:** <https://bu-team-l7qpgfrr.atlassian.net/browse/SCRUM-200> |

# Configuration Management Plan

## 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)

* GitHub Actions - CI/CD
* ChatGPT
* Docker
* IntelliJ IDEA
* VSCode
* SAST/DAST - CodeQL GH Scanning
  1. 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/>

We are following a GitFlow-inspired branching strategy to keep development organized, reduce conflicts, and ensure quality before code reaches production. Our main branches will be:

* **main**This branch represents the production-ready code. Only fully tested, reviewed, and approved code is merged here.
* **dev**The development branch where all active development work is merged first. This ensures that new features and fixes can be tested together before moving toward release.
* **qa**Dedicated for testing and quality assurance. Once work is stable in dev, it is merged into qa for system-level or acceptance testing.   
    
  All changes must go through a pull request before being merged into dev, qa, or master. A minimum of one approval is required (ideally from a teammate who reviews code quality, functionality, and adherence to standards). PRs serve as both code review checkpoints and documentation of changes, allowing teammates to understand what’s being added or fixed. Each commit should represent a logical unit of work (e.g., “Add JWT utility class” instead of “fix stuff”). Where applicable, commits should include unit or integration tests. Lint/format checks pass before committing (to ensure consistency). The promotion flow works as follows - Developer works on a feature branch and opens a PR into dev. After approval, the code is merged into dev. Once a group of features is stable, dev is merged into qa for testing. After successful QA, qa is merged into master for release.

## CI/CD Plan

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

We are using GitHub Actions to automate the build, test, and deployment pipeline for our application. This ensures that every code change is validated early and deployed consistently. Every time a developer pushes a commit or opens a PR, GitHub Actions automatically runs the pipeline. The code compilation/build verifies the application compiles without errors. It will execute the unit and integration tests to confirm that new code doesn’t break existing functionality. It also ensures coding standards and styles are consistent across the team. Security and dependency scans are performed to catch vulnerabilities early. If any step fails, the PR is blocked until the issue is resolved, ensuring only high-quality code reaches shared branches.  
  
 When changes are merged into the qa branch, GitHub Actions automatically deploys the application to a QA/staging environment. This allows testers to validate new features before production.  
  
Once changes pass QA and are merged into master, GitHub Actions triggers a deployment job that pushes the latest stable build to the production environment. This guarantees that the code running in production always comes from the approved main branch.

We have environment variables stored in GitHub’s encrypted secrets manager (ie APP\_JWT\_SECRET, AWS\_DEPLOY\_ROLE\_ARN, EC2\_SSH\_KEY, and JDBC\_URL. These secrets are injected at runtime to keep sensitive information secure. Successful builds generate Docker images. Images are pushed to a container registry for consistent deployments across environments. All deployments are monitored for stability and performance. Rollback procedures are in place to restore production quickly if issues occur.

# Quality Assurance Plan

## 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 |
| --- | --- |
| Defect rate | Keep track of number of bugs found per KLOC tracked via GitHub issues |
| Test case pass rate | % of passing tests/total number of tests |
| # of test cases | Number of automated, manual, and unit tests |
| # of user stories completed | Number of user stories completed/user stories in total |
| Person Hours | Logged per iteration for estimation accuracy |
| Code coverage | Percentage of code covered by tests, measure using Gradle or similar tools |
| Lines of Code | Used to track codebase growth and complexity |
| Cyclomatic Complexity | Measures the complexity of code logic to identify high-risk areas |

* 1. Coding Standard

(Describe any coding standard to be used)

* Frontend: React with TypeScript strict mode enabled.
* Backend: Java code follows Google Java Style Guide.
* General:
  + Consistent naming conventions
  + Code formatting enforced via Prettier and linters
  + All code must be readable, maintainable, and well-documented

## 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 changes must go through Pull Requests.
* Design/Implementation Leader (Pedro) reviews frontend/backend logic.
* Peer reviews are mandatory before merging.
* A review checklist includes:
  + Code readability and clarity
  + Maintainability and adherence to standards
  + Adequate test coverage
  + Proper documentation and comments
* If feedback is provided, team members should at least acknowledge the feedback even if no changes are made

## 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)

* Tools and Framework:
  + Unit Testing: JUnit and Jest
  + Integration testing: Postman, custom features
  + End-to-End testing: Selenium with java
  + Manual Functional Testing: Conducted during each sprint for exploratory and UI testing
* Roles and responsibilities:
  + QA Lead: Designs and executes automated end-to-end tests
  + Developers: Write unit tests for their own code
  + Team: Participates in testing
* Testing objectives:
  + Ensure all user flows are intuitive and bug free
  + Validate API responses and performance
  + Confirm seamless integration between frontend, backend, and database
  + Maintain high test coverage and low defect escapes

## 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. )

* Use GitHub issues to track defects
* Types of defects to look at are functional bugs, UI/UX issues, performance bottlenecks, and integration failures
* All defects are logged and prioritized
* Critical issues are addressed first
* Team discusses each defect; ownership is assigned based on expertise
* Fixes are verified through regression testing

# 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 |
| --- | --- | --- | --- | --- | --- |
| ChatGPT | Entire Team | Drafting proposal sections, refining requirements, risk mgmt ideas | Yes | Team reviews and edits to match project goals | *(chat history link to be provided)*  [*https://chatgpt.com/share/68bdb8d0-16d4-8007-8072-6fe5ea465dc0*](https://chatgpt.com/share/68bdb8d0-16d4-8007-8072-6fe5ea465dc0) |
| Visual Studio Copilot | Pedro and Jimmy | Front end integration | Yes | Code creation |  |
| Copilot | Stacey | Refining QA requirements | Yes | Created a first draft and then used AI for refinement. Reviewed based on real life experience and the requirements of the team | Added to appendix below |

# References

(Any references/citations that you have used)

* LinkedIn Jobs –<https://www.linkedin.com/jobs>
* Indeed –<https://www.indeed.com>
* Google Java Style Guide –<https://google.github.io/styleguide/javaguide.html>
* Git Branching Strategies –<https://www.flagship.io/git-branching-strategies/>

# Glossary

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

* **CI/CD**: Continuous Integration / Continuous Deployment.
* **QA**: Quality Assurance.

**Appendix**

How can you make this better? Quality Assurance Plan

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

Defect rate

Keep track of number of bugs found - JIRA/pivotal tracker

Test case pass rate

% of passing tests/total number of tests

# of test cases

Number of automated, manual, and unit tests

# of user stories completed

Number of user stories completed/user stories in total

Person Hours

Logged per iteration for estimation accuracy

Code coverage

ie using Gradle to see how many of our tests are passing

Lines of code versus test coverage that covers that code

Coding Standard

(Describe any coding standard to be used)

React with TypeScript strict mode enabled.

Java code follows Google Java Style Guide.

Consistent naming conventions and linting (Prettier).

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 changes must go through Pull Requests.

Design/Implementation Leader (Pedro) reviews frontend/backend logic.

Peer reviews required for all PRs before merging.

Checklist includes readability, maintainability, test coverage.

If feedback is provided, team members should at least acknowledge the feedback even if no changes are made

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)

QA leader will create and execute automated tests using java and selenium framework

This can be used for our end to end testing. The objective will be to ensure our user flow is free from bugs and is easy to use.

API testing. The objective here will be to ensure our API is responsive and provides the expected response based on data received.

Manual testing

Integration testing - Make sure our database, API, front and back end systems are integrated successfully.

End to end user flow as well. The objective will be to ensure our user flow is free from bugs and is easy to use.

Each developer will create their own unit tests

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. )

Use GitHub issues to track defects

Focus on critical defects reported first, then any non-priorities

For each defect raised, this will be raised with the team initially. If someone is aware of what the root cause might be, that person can take ownership to resolve.

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

ChatGPT

Entire Team

Drafting proposal sections, refining requirements, risk mgmt ideas

Yes

Team reviews and edits to match project goals

(chat history link to be provided)

https://chatgpt.com/share/68bdb8d0-16d4-8007-8072-6fe5ea465dc0

Visual Studio Copilot

Pedro and Jimmy

Front end integration

Yes

Code creation

Copilot said:

Copilot

Here’s a refined and more professional version of your Quality Assurance Plan, with improved clarity, structure, and consistency. I’ve preserved your original content but enhanced it for readability, completeness, and alignment with best practices.

✅ Quality Assurance Plan

📊 Metrics

Metric Name Description

Defect Rate Number of bugs reported per KLOC (thousand lines of code), tracked via Jira or Pivotal Tracker.

Test Case Pass Rate Percentage of test cases that pass: # of passing tests / total # of tests.

Number of Test Cases Total number of unit, integration, manual, and automated tests.

User Stories Completed Ratio of completed user stories to total planned stories per iteration.

Person Hours Total hours logged per iteration to assess estimation accuracy.

Code Coverage Percentage of code covered by tests, measured using Gradle or similar tools.

Lines of Code (LOC) Used to track codebase growth and complexity.

Cyclomatic Complexity Measures the complexity of code logic to identify high-risk areas.

📌 Note: These metrics will be reported in each iteration summary and progress report.

🧑‍💻 Coding Standards

Frontend: React with TypeScript, strict mode enabled.

Backend: Java code adheres to the Google Java Style Guide.

General:

Consistent naming conventions.

Code formatting enforced via Prettier and linters.

All code must be readable, maintainable, and well-documented.

🔍 Code Review Process

All code changes must go through Pull Requests (PRs).

Design/Implementation Lead (Pedro) reviews all frontend and backend logic.

Peer reviews are mandatory before merging any PR.

A review checklist includes:

Code readability and clarity.

Maintainability and adherence to standards.

Adequate test coverage.

Proper documentation and comments.

Review feedback must be acknowledged, even if no changes are made.

🧪 Testing Strategy

🧰 Tools & Frameworks

Unit Testing: JUnit (Java), Jest (React).

Integration Testing: Postman, custom test suites.

End-to-End Testing: Selenium with Java.

Manual Testing: Conducted during each sprint for exploratory and UI testing.

👥 Roles & Responsibilities

QA Lead: Designs and executes automated end-to-end tests.

Developers: Write unit tests for their own code.

Team: Participates in manual and integration testing.

🎯 Testing Objectives

Ensure all user flows are intuitive and bug-free.

Validate API responses and performance.

Confirm seamless integration between frontend, backend, and database.

Maintain high test coverage and low defect leakage.

🐞 Defect Management

Tool: GitHub Issues.

Defect Types: Functional bugs, UI/UX issues, performance bottlenecks, integration failures.

Process:

All defects are logged and prioritized.

Critical issues are addressed first.

Team discusses each defect; ownership is assigned based on expertise.

Fixes are verified through regression testing.

🤖 AI Usage Log

Tool Who Tasks Helpful? Evaluation/Modification Link

ChatGPT Entire Team Drafting proposal sections, refining requirements, risk management ideas Yes Team reviewed and edited to align with project goals Chat History

Visual Studio Copilot Pedro & Jimmy Frontend integration and code generation Yes Code reviewed and modified as needed N/A

Would you like this exported as a formatted PDF or Word document for submission?