**CS673 Software Engineering** 

**Team 1 - Chit Chat**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Masih Vahida | Team Leader | *Masih Vahida* | 09/07/2025 |
| Robin Roeoesli | Requirement Leader | *Robin Roeoesli* | 09/07/2025 |
| Jordyn Lipsey | Configuration Leader | *Jordyn Lipsey* | 09/07/2025 |
| Deasia Little | QA Leader | *Deasia Little* | 09/07/2025 |
| Ardit Briskaj | Design and Implementation Leader | *Ardit Briskaj* | 09/07/2025 |
| All | Security Leader |  | 09/07/2025 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[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 chat interface that allows users to either interact with the entire community or engage in a private conversation with an AI. The interface includes a chat box where users can type their messages. Community messages are broadcasted to all online users, enabling anyone to respond. Alternatively, users can choose to chat directly with the AI. In this mode, the message is sent only to the AI, and the response is visible exclusively to the user who initiated the conversation.

The idea is to create a portal where users can choose to ask questions either from the AI or from live users in real time.

The front end will be a web application built with React, while the back end will be developed using Node.js. Real-time communication will be handled through the Socket.IO protocol.

# Related Work

Similar software available online are messaging apps; however, these are typically designed for peer-to-peer communication or require users to create groups in order to chat with each other. They also usually require account creation with a username and password. In contrast, this project removes that barrier—users can simply open the app and start communicating immediately, without registration. Additionally, the ability to ask questions directly to an AI is a feature not commonly found in traditional messaging apps.

# 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. **Message Input & Sending System - 30 hours**
   1. As a user, I want to type a message in a text box, so that I can see and edit what I want to send before sending it.
   2. As a user, I want to be able to send this message by either clicking the Send button or on Enter press, so that my message is delivered.
2. **Chat Display and Container - 15 hours**
   1. As a user, I want to see all chat messages displayed inside a scrollable chat container, so that I can review recent and past messages.
   2. As a user, I want the chat message container to automatically update when a new message is received, so that I don’t need to refresh the app.
   3. As a user I want a different message type between mine, community and AI responses, so that I can easily tell who sent what.
3. **Navigation and Chat Modes Menu - 20 hours**
   1. As a user, I want a sidebar menu with options to switch between community and ai modes
   2. As a user I want to access Settings from sidebar, so that I can adjust my preferences
   3. As a user I want the chat box container to contain the relevant messages depending on the mode I have selected, so that I don’t get confused between different chats.
      1. Desirable Features (the nice features that you really want to have too):
         1. **Account Creation System - 20 hours**
            1. As a user, I want to create a new account so that I can login with my account next time and not lose my chat history
         2. **Login System - 5 hours**
            1. As a user, I want to be able to log in into my account, so that I can access my chat history and setting preferences.
         3. **Save System - 5 hours**
            1. As a user, I want my chats to be saved to my account, so that I can view past conversations whenever I return
      2. Optional Features (additional cool features that you want to have if there is time):
         1. **Profile Picture - 5 hours**
            1. As a user, I want to be able to upload my image and customize my profile.
   4. Nonfunctional Requirements
4. The system should be able to handle a few dozen users active at a time.
5. The application UI should be responsive and work for all screen sizes
6. [Socket.io](http://socket.io) needs to be integrated with both front and back end and handle all events coming from UI.
7. Users should not be able to crash the application with malicious inputs like query injections, unreadable chars, or insanely large amounts of text.
8. LLM has been integrated with application.
9. Account credentials are encrypted before saved to db.

# Management Plan

## Objectives and Priorities

1. The team will first complete all essential features such as message input, message display, and mode switching to ensure the chat system is functional.
2. A functional prototype will be deployed that allows both community and AI conversations by the end of the first iteration.
3. The system will be kept stable and reliable so that no critical bugs remain in the deployed version.
4. Quality assurance practices such as unit testing, integration testing, and peer code reviews will be applied to maintain a high level of code quality.
5. The application interface will be designed to be responsive and intuitive so that it works well across different screen sizes.
6. Security and privacy will be treated as priorities by preventing malicious input, encrypting sensitive data, and following secure coding practices.
7. Once the core system is stable, desirable features such as accounts and chat history will be added.
8. Optional features such as profile pictures and personalization will only be implemented if there is time remaining after the essential and desirable features are completed.

## 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 integration of different technologies such as React, Node.js, Socket.IO, and the OpenAI API may lead to compatibility or performance issues. This risk will be managed by integrating components step by step and testing frequently.
* The team may underestimate the complexity of certain features or face scheduling conflicts that cause delays. This risk will be managed by breaking down work into smaller tasks, tracking progress with Jira, and reviewing timelines at the end of each iteration.
* Insufficient testing may lead to bugs or security issues appearing in the deployed system. This risk will be managed by automating tests with PyTest and Jest, enforcing peer reviews before merges, and validating changes with GitHub Actions.
* Security vulnerabilities such as injection attacks or unsafe credential handling may appear. This risk will be managed by sanitizing inputs, encrypting stored data, and applying authentication best practices with Passport.js.
* The reliance on external services such as the OpenAI API may lead to failures or downtime if the API is unavailable or rate limits are reached. This risk will be managed by implementing error handling and ensuring efficient usage of API calls.

**Risk Management Sheet Link:** https://docs.google.com/spreadsheets/d/1nO7hR5ZuQhj\_ezcA37vpGulzMwAt3tELZfF\_y9OKClk/edit?gid=0#gid=0

## 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 | Essential: Message input and sending, chat display, mode switching | Set up GitHub repository and branching strategy; configure CI/CD pipeline; build message input box and chat container; implement basic backend server with Socket.IO; integrate first AI API call | 65 hours (estimated) |
| 2 | Essential: Full community and AI chat functionality, non-functional requirements such as responsiveness and security | Implement navigation sidebar; differentiate between user, community, and AI messages; improve error handling and input sanitization; run integration testing led by QA leader; deploy first working prototype | 70 hours (estimated) |
| 3 | Desirable and optional: Account system, chat history, profile pictures | Implement authentication with Passport.js and PostgreSQL; save and retrieve chats from the database; add profile picture upload; perform final testing and defect fixing; complete documentation and project report | 72 hours (estimated) |

# 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**)

**IDE:** Visual Studio (VS Code)

**CI/CD Tools:** GitHub Actions

**Container Tools (if needed):** Docker & Podman

**Testing:** Jest and Playwright

**Database**: PostgreSQL for data storage.

**Project Management:** Jira

**AI Tools:** OpenAI API

**Authentication:** Passport.js

* 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/>

Our Git standards of practice for the CS673 Project will be based on: scope, branch management, and review process.

1. Understanding Scope:

When branching off of the main branch, there should be clear scope that is identified. The scope should be concise and any team member should be able to read the branch name and understand what the branch's purpose is.

Ideally, scope should be concise; if you require 50 commits to achieve the goal in mind, your scope is too large.

Once scope is defined, changes should be made to that branch only with scope in mind.

1. Branch Management:

Branch naming is very important to defining scope and to understand what purpose each branch holds.

These are the following naming conventions we should be using for our branching -

1. Use grouping words at the beginning of our branch name, this should define the purpose of the branch
2. A short description of the branch should be given after identifying information

List of grouping words -

* feature - Feature that is being added or expanded on
* bugfix - Fix a bug in the code

*Branches should be named in lowercase letters separated by hyphens.*

For example, to create an additional feature to the app, the branch would have the following name: *feature/add\_textbox.* To fix a bug related to the app:*bug/bugfix*

This will allow consistent naming conventions that are descriptive. It will allow everyone to understand the purpose of each branch in GitHub/Git.

Branches should be kept up to date with main whenever possible. Once a branch has been merged with main, the stale branch will then be deleted.

1. Review Process:

A review process will be set in place for anyone creating a PR onto the main branch. Depending on the complexity of the pull request, the number of reviews might change.

* For a standard pull request (less than 10 commits/less than 4 files changed)
  + Only one review is required for merge
* For a large pull request (more than 10 commits/more than 4 files changed)
  + Two reviewers are required to approve prior to merging. Due to the added complexity of commits, multiple reviewers should be needed to make sure that everything looks correct.

The above is to incentivize small pull requests. When a pull request is too large it causes the following problems:

1. It is very difficult to review
2. There are higher chance of errors flowing through

You should assign your pull request to member(s) of the team that the review is most appropriate for.

When creating a pull request, there should be a strict template that should be followed for the description of the pull request -

1. What is the purpose of this pull request?
   * Give a quick high level summary to whoever will review the PR on what the scope of the pull request is
2. What are the added benefits we will achieve from the pull request?
3. Quick summary of the changes made in each file
4. Are there any sections in particular that should be given a more thorough review?
5. Additional thoughts/considerations

Pull requests should include any additional unit tests along with documentation changes for the application. This will help ensure that team members will understand what has been changed and be reassured that it should work.

## CI/CD Plan

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

Every time code is pushed to GitHub, GitHub Actions will automatically install our project’s dependencies and run both Node.js and Playwright tests (using Playwright for JavaScript). If the tests pass, it confirms our app is stable; if they fail, GitHub will flag an error. This ensures our main branch always has a tested, working version of the project.

# 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 |
| --- | --- |
| Lines of Code (LOC) | Count of total lines of code at the end of each iteration, to show how the project grows. |
| Cyclomatic Complexity | Tracks how many decision paths are in the code to keep functions simple and maintainable. |
| Number of Classes | Total classes created in the project, showing how the design scales over time. |
| Test Case Pass Rate | Percentage of test cases that pass in each iteration to measure stability. |
| Automated Test Coverage | Percentage of code run by automated tests to show how much of the system is tested. |
| Defects Logged Per Iteration | Number of bugs found and recorded each cycle to track quality trends. |

* 1. Coding Standard

For this project we are using JavaScript (ES6+) for both the frontend and backend, with React on the frontend and Node.js with Express on the backend. All code will be written in VS Code and stored in a shared GitHub repository, using feature branches for new work.

We will use an automatic code formatter to keep spacing and quotes consistent. Variables and functions will follow camelCase, while React components will use PascalCase. On the frontend, we will use functional components with hooks and add data-testid attributes where needed for testing. On the backend, routes will be organized by feature and sensitive values like API keys will be kept in environment variables.

PostgreSQL will be our database, managed through migrations, with passwords stored securely using hashing. Testing will be done with Jest for unit tests and Playwright for end-to-end flows like login and chat. Branches will follow feature/... and fix/... naming, with short descriptive commit messages and pull requests reviewed before merging.

## 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?)

*Please see configuration management plan for detailed pull request review process.*

A review process will be set in place for anyone creating a pull request onto the main branch.

You should assign your pull request to member(s) of the team that the review is most appropriate for. For general pull requests, any member of the team can review.

Given the template for pull requests, reviewers should easily be able to understand the purpose of the code, and any areas that require extra attention. Reviewers can accept the pull request if everything looks acceptable or reply with questions before accepting the pull request.

## 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 rely mainly on automated testing and use manual checks only when needed. Each developer will write unit tests for their own code using Jest—this covers both backend (Node/Express) logic and frontend React components (with React Testing Library). The QA lead owns integration and end-to-end testing with Playwright to validate core user flows such as account creation, login, joining a room, and sending/receiving messages. Defects will be tracked in GitHub.

When and what types: unit tests run locally and on every pull request in GitHub Actions; integration/API tests (Jest) and end-to-end tests (Playwright) also run in CI on PRs and on the main branch as smoke checks. Integration testing will include making real requests to the API and database to confirm routes, authentication, and chat events work together correctly. Before any class demo or milestone, we will run a full automated regression (unit, integration, E2E). Manual testing is limited to quick sanity/exploratory passes after major changes.

Our testing objective is to maintain a high pass rate, expand automated coverage as features are added, and catch issues early in the pipeline so broken code does not make it into the main branch.

## 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. )

The team will be using Git Hub Issues to track defects and improvements.

Below are some labels for tagging issues:

Priority (shades of red)

* 0 (critical) - must be fixed immediately for users to properly use our application(s)
* 1 (high) - significant, potential negative impacts to users
* 2 (medium) - all other
* 3 (low) - nice to have, no timeline, just documenting ideas

Improvements (green):

* Efficiency (runs faster, less memory, less storage)
* Functionality (new cool thing, made something more reusable/robust)
* Reliability (better testing, logging, security, consistency/reproducibility)
* Sustainability (easier to use/update/add, easier to understand, less technologies/packages)

Once an issue is found, a Git Hub issue will be logged with priority and improvement, and a team member will be assigned to it (if applicable). Git Hub Issues will be attached to pull requests and closed once completed. Issues can include bugs in the code, new features for the app, or any ideas for improvement that are not being tackled on your current branch.

# 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 | Ardit | Ask ai to create a visual representation of what our app could look like as a point of start. | Yes |  |  |
| ChatGPT | Masih | I explained the project overview to the AI and asked for suggestions for the GUI | Yes |  |  |
| ChatGPT | Masih | I asked for recommendations on the tool sets and the languages to be used for the front end and the backend | Yes | It recommended [Socket.io](http://socket.io) for communication which aligns with our selection as well.  For frontend it recommended React which better connects with socket.io .  For backend recommended [node.js](http://node.js) which is also aligns with our selection |  |
| ChatGPT | Deasia | I asked for best course of testing frameworks since we changed from Streamlit to React | Yes | Recommended Playwright with JS so it will integrate seamlessly with the project |  |
| ChatGPT | Jordyn | Can you create a logo for a chat app called chit chat? | Yes |  |  |
| Chat GPT | Masih | Asked for different recommendations on how to implement a button and text box on the GUI in react since i had trouble with the way i implemented it | Yes | It recommended multiple different ways and I modified my code based on the recommendations and it finally worked! |  |
| Copilot Claude | Ardit | Asked Claude to create a logo | Yes |  |  |
| Copilot | Masih | Used to put a presentation together in Power Point | Yes | It made the template design for the slides easier |  |
| ChatGPT | Masih | Used to make the documentation cleaner | Yes | It made the sentences better, simpler and more comprehensive |  |

# References

(Any references/citations that you have used)

# Glossary

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