FSE Spring 2020 Team 11 Final Report

Mitali Shroff, Thomas Dohle, Yanchi Li, Timothy Gladyshev

Section 1: Summary of Project Goals

We set out to create a server-based application that allows for sharing messages between users, as well as between users and groups. The high level functionality we originally wanted to achieve included registering as a user, joining groups, sending messages to users/groups, searching for users/groups, and CALEA compliance.

We wanted to approach this project from the point of view of test driven development, and a commitment to quality. All features were pushed with testing, we generally favored testing over maximum feature completion, and we mostly kept our code coverage consistent.

We were also interested in exploring and implementing some interesting features that are not unanimous in messaging services. After communicating with the product owner, we decided that setting an alias on messages for private discussions is an exciting and useful direction.

In terms of our original vision for the project structure we drew up a rough UML diagram of necessary classes and their relationships. The reality of development, as well as a lack of familiarity with the inherited code mostly took us in a different direction, but still we found that a useful reference throughout the project.

After extensive communication with the product owner, we created an SRS document (included) detailing possible features and priorities. To accomplish as many of these as possible over the short project timeline, we created our team goals at the start of every sprint as follows. The first coding sprint goals consisted of testing the inherited code, supporting basic communication, and establishing a database connection. The second 2 weeks were devoted to expanding basic communication, adding search and friend functionality, and adding basic group functionality. During the last sprint, we planned to expand friending capability, add status functionality, improve the database, introduce system logging, and add anonymous communications to messaging. Work was broken up based on past experience and interest in the stories.

The project goals were extensively discussed, represented our interests, and as a result we were able to meet and surpass our goals at every stage, as well as deliver a quality final product with great value to the owner.

Section 2: The Product

Our team's goal was to develop a system that fulfilled as many client requirements as possible without sacrificing quality, maintainability, or reliability. We achieved this goal by focusing first and foremost on the integrity of the code that was produced. Before implementing any new features, we spent a good portion of the first sprint writing extensive unit tests for the code that was handed to us (Issues 5-9, 21). From then onward, we utilized test-driven development to ensure that our coverage remained high. We did experience a slight dip in coverage between sprints 3 and 4. Upon noticing this, we dedicated time after sprint 4 to enhance test coverage and eliminate warnings/smells pointed out by SonarQube (Issues 94, 98, 99). In addition to unit tests, we also implemented extensive, standardized logging to help track errors that occur on our production system (Issue 66). By linking our system with site24x7, we have a convenient dashboard to search logs and be notified of issues in real time.

In addition to supporting the following described features on the server, we also thought it was important to provide a practical and simple user interface so that our clients don't need to rely on clunky tools such as postman that are not intuitive to the non-technical user.

User features:

- A person can register as a user. Once they register, they can connect to the system from any browser with an internet connection
- Users can send messages directly to other users or broadcast messages to all users who are online (Issue 12)
- Users can send messages to other users or to groups anonymously with a custom alias (Issue 70)
- When a User logs in, they can see all messages that have been sent to them (Issue 35)
- Users can set and update a status that is visible to other Users (Issue 24)
- A User can add another User as a friend and see their status (Issues 39, 40, 63, 64, 69)
- A User can see which other Users are online (Issue 68)

Group features:

- Users can form groups with other users (Issue 36)
- Users can join groups that already exist (Issue 41)
- Users can add other users to existing groups (Issue 65)

- If a User is a Moderator of a Group, that User can remove Users from the Group or disband the Group (Issue 62)
- Users can send a message to all members of a Group that they are a member of (Issue 37)

Section 3: The Process

The Team-11 development process started off with a **Product Backlog**, a prioritized list of features suggested by the product owners. We created a Software Requirements Specification (SRS) and also developed UML diagrams to gain a better understanding of the project being undertaken.

We decided to build the system in increments, making sure we targeted the important parts, but also ordering them such that having one functionality in place would set us up for the next task.

During our sprint planning, we took into account what needed to be added to the sprint based on what had the highest priority in the **Product Backlog**, and the availability and capacity of each team member.

We found that grooming the stories by voting and assigning story points to each task, we could estimate the complexity better. **Grooming** each story also allowed us to discuss the feature being tackled and briefly discuss how to tackle it.
Based on the items chosen for the sprint, a **Sprint Backlog** was created on GitHub Project. Story pointing helped to make sure we were not over committing. We also prioritized the tasks in the sprint backlog so that we could develop in increments and prevent dependency on each other.

The table below summarises the total story points we picked up in each sprint, and how many stories we covered.

Sprint	Story points undertaken	Stories covered
Sprint 2	46	10
Sprint 3	32	8
Sprint 4	45	11

Each sprint spanned over a two week period. The scrum master scheduled meetings and regular check-ins to make sure no one was blocked and to have the team give a

progress update. This held us accountable, as well as gave us a chance to reach out in case we needed anything, or if we could be of assistance to anyone.
We also heavily used Slack to track progress, communicate blockers and for note-taking. Creating pull requests allowed us to gain valuable feedback from each other, and we took time out to test each other's code changes. We worked in a self-organizing manner. The team was highly available and approachable throughout the duration of the project. We were even able to squeeze in some extra features in each sprint due to our planning and prioritization.

At the end of every sprint, we performed a **Sprint Retrospective**. This allowed us to glean what went well during the sprint, what could have been improved, and cheers for peers that did well during that sprint. The outcome of the sprint retrospective is a list of actionable items that should be implemented in coming sprints.

The last step in the Scrum process was the **Sprint Review** meeting, where we went over the deliverables achieved, demoed new features. This gave the product owner a sense of the sprint increment.

Before the final submission, we worked on our test coverage, cleaned up code smells, created Javadocs and improved the UI.

The team was highly available throughout the sprints, and displayed enthusiasm for the product being developed. There was collaboration not just at a team level, but even amongst individuals on different features of the project. The team was receptive to feedback, as well as adept at giving good feedback. We did not shy away from reaching out if we needed help. We were able to estimate well, the amount of work in each sprint, and everyone took charge of the tasks taken on. Implementing features over and above the intended goals boosted team morale. There was knowledge sharing on various levels, as different team members handled different parts of the project. This ranged from database operations, to back end, to front end, and operations tasks.

An issue we faced was delaying code pushes to the end of the sprint. We resolved this by setting a hard deadline for code merges a day prior to the end of the sprint. Adjusting to purely online communication mid-way through the project was another interesting challenge, but thanks to the flexibility and dedication of the team, we were able to continue to deliver features. Resolving merge conflicts as a team proved to be a good way to avoid erroneous merges.

Some good practices we implemented were to maintain code quality above quality gates, to first merge changes into a pre-master branch to resolve merge conflicts, and to use a sprint retrospective board to develop action items for coming sprints.

Section 4: Retrospective

What the team liked the most

- It was realistic exposure to what it's like to work on a development team
- Collaborating with good people
- It reinforced that I enjoy product development and the structure of Agile

What the team liked least

- It was difficult to understand what the actual expectations were from the project

What the team learnt

- Learning to trust your teammates to do good work
- Communicating honestly about my strengths and weaknesses
- What it's like to be a scrum master

What should change?

- Messaging seems to be a solved problem. It would be more interesting to work on something more unique
- There wasn't much formal guidance about accessing/configuring the Mass Open Cloud instance
- It would be nice to learn how to use a real world cloud service like AWS or Azure, although I did learn a lot from working with a bare bones server