Final Project Report
Team 208

Ram Prakash Arivu Chelvan, Shashwat Sanghavi, Himanshu Budhia, John Goodacre

1 Goals

- The primary goal of this project is to build a service that allows people to message each other directly or in groups, easily and securely.
- This service must be deployed on the cloud (AWS instance), that the client can communicate with from anywhere on the planet (with an Internet connection).
- Stretch Goal: Build a robust Front-End interface that works on all smart devices with a modern web browser.

The service should have the following features. User of this service should be able to:

- Find each other with ease and create new groups with other users in the service.
- Access this service from their phones, tablets or their laptops, both public and their private devices.
- Retrieve old messages from their conversations when online.
- Start a private converation with other users in the service.
- Mark themselves as non-discoverable or private users.
- Broadcast a message to all the user conversations.
- Add other users to a group, delete groups if they are a moderator.
- Stretch Goal: Have all their messages translated to a language of their choice.
- Stretch Goal: Users should be able to update their display picture.
- Stretch Goal: Users should be able to send Emojis and GIFs.
- Stretch Goal: Users should be able to send and receive images and videos and view them in the client.
- Stretch Goal: Users should be able to see if other users are online.

2 Overview

The team was able to achieve all of the goals discussed in the previous section, both base expectations and the stretch goals.

2.1 Stretch Goal: Front-End built with React Framework

A fully functional and stable front-end has been built with the React Framework and JavaScript. This Front-end client runs on multiple platforms, from smartphones to personal computers, basically any smart device with a modern browser running on it. This client supports all of the features that were initially planned to be implemented and the client is hosted on AWS instance for remote access.

2.2 Stretch Goal: Features

All of the stretch goals that we were planned in terms of features were implemented in the final sprint and they were successfully demoed during the review as well. Users would now be able to see translated messages in their language, send/receive Emojis, GIFS, pictures, videos and also view these multimedia messages in the client itself. The users can also view the Online status of other users using the service.

2.3 Completion claims from JIRA

The team used JIRA as the Project Management Tool and all of the backlog items were recorded and delegated at beginning of each sprint by the corresponding Scrum Master.

Sprint #	Issues Assigned	Issues Completed
1	14	14
2	14	14
3	37	33
4	19	19

NOTE: For Sprint 3, the team took on an extraordinary amount of work and was able to complete most of them except a few issues (stretch goals) which were completed in the following sprint.

2.4 Test Coverage and Code Quality

- The team used Sonarqube to assess test coverage and code quality, as was mandated for this course.
- Tested all functionality in the Prattle code (legacy and old) to test and assess code behaviour.

Unit Tests	135
Errors	0
Failed	0
Success	100 %

Overall Code Coverage	81.8 %
Condition Coverage	68.6 %

- Removed all bugs and vulnerabilites
- Ensured duplication of code is below 3% (1.1%)
- Removed major code smells (A rating)

Bugs	0
Vulnerabilities	0
Code Duplication	1.1%

3 Development Process

The team's development process at the beginning of each sprint, normally started with the discussion of what needs to be done and who can take care of what issues. We met during the beginning of each sprint to discuss this and we met a couple of times towards the end of each sprint to merge all our individual work and push it to the Master branch. We used Slack to constantly communicate what needed to be done, and depending on the kind of work that needed to be done in that sprint we would meet more often to get things done. Since two amongst the four of us are full-time employees, this development workflow worked best for us.

Initially, we tried to have daily standup meetings over the phone some time in the night, but the general workload did not give us that much time to speak over the phone every day. So we decided to communicate over text so that the rest of the team is constantly updated of the work progress.

Here are some of the things that worked well:

3.1 Github

All of the code had to be merged with the Master branch at the end of each sprint. Each of us had our own working branch and we also had a branch called "develop" which was working master. When we finish working on a feature or an issue, we merged the code on our individual branch with "develop" branch. Conflicts while merging were normally resolved during our team meetings. Towards the end of the sprint, once the code in "develop" passes the Quality Gates set by Sonarqube, we created Pull Requests and merged our code from the working Master branch to the real Master branch.

3.2 JIRA

At the beginning of each sprint, the scrum master created and assigned issues on JIRA after a team meeting where we discussed who works on what issues. These issues were then assigned to the current sprint and work begins. We also decided to use smart commits that essentially ties a Github Commit to a specific issue on JIRA. This helped us to review the work done on specific JIRA issues.

3.3 Communication

This was the key element that worked really well in our team. Constant communication and support from teammates enabled productive teamwork. If a teammate was stuck on an issue, we sat and tried to resolve them together. Everyone was kept in the loop about active changes in the codebase and everyone contributed during our brainstorming sessions.

3.4 Standardized Development Practices

- We ensured thorough documentation by using comments for classes and methods, increasing readability
- Followed standard naming conventions for variables, classes, and methods
- Used static constant variables for code maintainability
- This would help our future developers to read and understand our code easily, if this was passed on to them directly as base code.

3.5 Incorprating feedback

At the end of each sprint, we were able to look back at the work done in the previous two weeks and make better decisions in the following sprint. The feedback given by the Product (TA) had useful suggestions and since Agile allows to incorporate changes easily, we could incorporate the feedback while planning the following sprint.

4 Retrospective

The team really enjoyed working on this project. It's not just another project that can be showcased in our portfolio, it was also a great learning experience. These were some of the things that the team liked and learned from the project:

- The project was designed to be a learning experience, right from the start when we were given legacy code. Understanding and working with legacy code was challenging, especially testing different components from the legacy code. But we were able to learn new testing techniques in Java and implement them in this project.
- The Agile Project Management style was something new and exciting for the team. Although it was a little overwhelming in the beginning, we grew into it and got comfortable with the process by the end of second sprint.
- The project encouraged us to design the system keeping scale in mind and that positively influenced many of our design decisions.
- Learning design patterns in class allowed us to implement them in this project, and we found that to be really interesting as the project allowed us to demonstrate what we learnt in class.
- We learnt new testing strategies to be able to test certain components of the codebase.
- We also learnt to use new Java based libraries that interfaced with Cloud based services like AWS.
- As part of the Agile Methodology, we learnt to work remotely but in sync with continuous team communication.
- The role of a Scrum Master helped us learn leadership qualities that would be useful in future managerial tasks.

Suggestions to support a greater experience:

- When we were handed the legacy Prattle code, better support in terms of explaning how everything worked in the old code at the beginning of the sprint itself could have been really helpful at that time. The videos from the Prof. Jump came in a little later than we'd expected but it was still really helpful.