Final Project Report: Slick Communications

Team-108

Manikantha Dronamraju, Sean Fitzpatrick, Justine Lo, Priyaranjan Panda

CS5500: Managing Software Development
Professor Michael Weintraub
11 December 2018

Summary

Our system is designed as a communication server that provides fast messaging services without any delay. Project goals included user messaging, group messaging, persistence of messages and users, encrypting passwords, recalling and searching of messages, adding MIME types to messages, storage of unread messages, the capability to wiretap a user or a group of users, parental control, and wrapping messages with sender's and receiver's IP.

To access the system, the user has to first register with the system. After the initial registration is successful, they can login anytime with the username and password they created during registration. The system stores all the passwords in an encrypted manner. After logging in, they can see the list of active users in the system. They can send a message to any user whether they are online or offline. They can also receive messages from other users. The user can also update their name and delete his account from the system. Additionally, if the user receives messages when they are offline, the system will give the user the backlog of those messages the next time they log on. These messages will be sorted in order based on the timestamps of when they were sent.

A user can also form groups and add other users to the group. This allows for group messaging capabilities, so a single message to a group goes to all present users in the group at once. Similarly, messages that are replies to messages sent to a group go to all other members in the group. Group members can also delete users from the group or delete the group entirely.

Aside from sending simple text messages, users can send different MIME type messages, such as .mp3, .vlc, .png, and .jpeg. Another feature of the system is that all the records of users, user groups and messages sent by users are persisted in a database so that nothing is lost. This provides the flexibility to users to recall their messages. Users can recall any message that they have sent previously to other users. Recalled messages are not delivered to the receiver as part of the message backlog delivery from the server. Each message that gets stored in the system is wrapped with sender's and receiver's IP address. The receiver's IP with respect to a message remains empty when the receiver is offline but gets recorded immediately the next time the receiver comes back online and gets the backlog of those messages. Besides these functionalities, the user can also search his messages based on receiver's name , his name and input timestamp. The system will display a list of messages the user had previously sent based on the user's search conditions. Both recall and search messages were done for CALEA compliance.

In addition to these messaging capabilities we implemented a Parental Control feature which allows the system to flag inappropriate messages. The user can decide whether or not to use parental control upon logging in, should they want to see sensitive content. This entire

process is performed offline and does not cause any delay for sending messages. This functionality is currently only working for text messages.

Apart from user functionalities, the system allows for agencies to wiretap any user's or group of users' communications provided that the agency has a subpoena. It is done without the user knowing they are being wiretapped. The agency can only read the messages in which the user or group sends or receives and cannot reply or alter these messages in anyway. This wiretapping session remains active for 3 hours after which the communication link between the agency and the target gets disrupted.

A notion of an administrator was also created. This allows for the capability of dynamically turning on and off a logger to view any errors thrown during the runtime of the system.

Overview of the Result

The product overall is a moderate success. In terms of feature completion, it was a great success. The prior listed features were all completed on time and within the constraints of requirements given. Sprint-over-sprint we increased our output by a considerable amount as we learned more about the system, and improved our processes. With the exception of sprint 1, we did not have any feature carry-over between sprints. With respect to actual work completed, our data was a little scarce. We focused more on implementation of features and thus could have had a more robust issue tracking process. For instance, we neglected to estimate point values for tickets in JIRA, which would have given us a better picture of our velocity. Instead, we must rely on the number of tickets we completed each sprint, and accounting for the fact that there may have been some issues here or there that did not end up being tracked in JIRA. The first sprint we completed around 20 issues, the second completed just over 30 issues, and in the third completed 20 tracked issues, however we believe had we been more diligent in tracking issues, we would see that we completed more work than in the second sprint. That said, our we are confident we tracked major features and requirements effectively, and we can point to our backlog being empty as evidence. As a result, the system is fully featured, deployed, and usable.

With the velocity that we had in feature completion, our code quality suffered. The codebase as it stands is covered by 310 unit tests resulting in 85.1% code coverage. While this was our target throughout the project, this number could increase and our test suite could be made more robust with some refactoring to the codebase. Continuing with technical debt, SonarQube is currently tracking 90 code smells and 5% code duplication overall. Both of these could have been avoided by favoring a little more care instead of velocity up front, however the work required to fix both is relatively little. That said, there is a non-negligible amount of effective duplications that SonarQube doesn't identify - for instance many methods across data access object classes are extremely similar, and moving forward we would refactor these into a handful of more generic methods. Firstly, this would ease some of the difficulty we faced in testing, and probably increase our overall test coverage. Secondly, having a handful of generic methods would increase the system's scalability and maintainability, making it much easier to add new features.

With the system deployed we were able to run stress and load tests using JMeter. Our findings were that the system in its current instance can safely handle 1500 concurrent users sending and receiving messages at once. Among our tests we found failures would begin to occur in the 1600-1800 user range. We suspect this is at least in part bound by the amount of computational power we have on AWS, but more testing would be needed to determine this. Around maximum load the average round trip time for a message was typically around 355ms. At lower levels of load, e.g. 100 - 1000 users, the round trip time averaged 249 - 266ms. These times are roughly what we expected, and are short enough to provide a fast and seamless user experience to any front-end used. For next steps, we would also like to test sending MIME-type messages with associated files, track multiple users over time, and with mixed types of messages.

Team's Development Process

Over the past several weeks, we demonstrated our abilities to work together as a team and produced quality work. Our team was able to overcome all issues we encountered and improved on teamwork and communication, leading to a successful final product that fulfilled all necessary requirements, functionality, and more.

We adopted the Agile methodology and implemented Scrum processes to help us organize and distribute tasks. Initially, our group had a few issues with following these processes which lead to misorganization and poor communication. During Sprint 1, some of us were not as familiar with how to best utilize Git and Jira. This lead to poor distribution of work, and team members were unsure of who was doing which tasks. There was also poor time management due to busy schedules. Members in the group were either working full time, had interviews, and/or overloaded with other coursework. As a result, there was a decrease in work quality due to last minute work. Although expectations and stretches were all completed for Sprint 1, there was a poor understanding of project goals, as well as misorganization and lack of communication.

For Sprint 2, we reflected on issues that occured in Sprint 1. To improve organization and communication, we had to first make sure everyone was familiar with the technologies we were using, such as Git, Jira, and Slack. We went over proper protocols on how to use Git; a reference guide on Git workflow and a template on creating pull requests for merging branches were written and pinned in the Slack group. These documents described basic commands such as creating branches, pushing, pulling, and committing work. We also did daily "stand ups" via Slack. Everyday, each member would post what they will be doing that day and what issues or problems that came up. This helped inform the group on everyone’s progress and when to expect certain features to be completed. Utilizing these technologies drastically improved work quality, workflow, and communication. Group members also had a much better comprehension of project goals and overall flow of the code. In addition to this, group members became more familiar with one another’s work habitats and skill sets.

By Sprint 3, the team had a very good understanding of how to best incorporate new features in a very efficient manner. Tasks were easily and quickly divided up based upon each other’s scope of knowledge and understanding of how new features should be implemented. Each team member was able to take ownership of different aspects of the project, and we were able to successfully integrate all of our work together.

Aside from communication and addressing issues as soon as possible, another key aspect to the success of the team was understanding each other’s shortcomings. Everyone has a different skill set, so by understanding each other’s strengths and weaknesses, we were able to drastically improve the efficiency and quality of our work in a short amount of time. For example, a couple members were more familiar with implementing the database and creating persistence for users, groups, and messages. These members focused on these features of the project, and afterwards, reviewed and explained the code to others so that everyone had a good understanding of how the features work and were implemented. For features that none of the group members were familiar with, at least two members will work together and figure out how to implement these features. For example, we were not familiar with how to use Mockito for testing. Two of the group members took the time to look into the implementation and figured out how to best utilize Mockito for testing.

Overall, our group had great teamwork; everyone was reliable and responsible, did their best to be flexible in their schedules, listened to one another, kept the team informed of progress and issues, always ready to help, and showed genuine commitment to the team and project. We were successful as a team and in accomplishing all project goals in an organized and timely manner.

Experience as a Team

From our perspective the project had a strong learning curve; beginning in sprint 1 we had quite a ride till the end of it. At the beginning we had growing pains adjusting to each team member’s style and schedule, but ultimately everyone fulfilled their responsibilities. Over the course of the following six weeks we gelled as a team and that helped us achieve better results overall.

The part of the project was that we enjoyed the most was having the liberty to implement things the way we wanted. Even though there was a limitation of not using http we still enjoyed figuring out how to do things without it. We learned the most with using thread manipulation, running multiple threads in parallel, socket programming, reflection, and testing using a mocking framework like Mockito. There was so much to learn in relatively little time and we made the most of it. As a team and as individuals we are proud of what we achieved and learned from the project.

The worst part of the project would be the legacy code that we had to modify and add on to. Developing a project from scratch is simpler than reading over the code written by a third person and having to work on top of it. Initial days of our sprint were used at debating whether we should modify the legacy code or not. With the limitation of socket programming we thought a lot of time was not as best as it could have been, and may have been better if we had been given a walk-through of the existing flow.

We learned how to work as a team together, coordinate the work and make sure systematic development was happening. Every sprint someone would volunteer to learn something new that could be used to better our implementation. Apart from the technical details mentioned we learned to develop a sense of trust and belief for our team members and take initiative and support them if they could not accomplish a task. Also, the most important learning we had was to never leave anything for the last day.

As a team we understood that the idea of the project is to have some constraints, e.g. socket programming, but we believe that the liberty to choose a project should the team's. This gives us something that more achievable. Also, before beginning the agile process it would be beneficial to have a sprint focused on doing research, gathering requirements, familiarizing with systems, etc. As at the beginning of the sprint there was a negative vibe with no http and socket programming a lot about legacy code. To develop a more positive environment and encourage new ideas we believe that we should have a little more freedom in choosing tools and methodology (e.g. http), and this would improve the experience even more.