Goals

The goal of the project was to build a messaging platform for users. This messaging platform mocks modern day messaging platforms like Skype, Slack minus the GUI. We began to include possible features into sprints and that journey will be discussed later. Every sprint had a set of tasks to be completed and we managed to have a fully functional messaging system in three sprints, lasting two weeks each.

We started with the notion of users and groups and sending messages between users and groups. This is most basic functionality of any messaging system. With every Sprint we enhanced the system to add features that make the system trustworthy and easier to use. Apart from software features we also setup environment functionalities like automated build, checks and deploy to our cloud resource AWS. The intention was to build a fully functional messaging system by covering all the functionalities mentioned in each sprint whilst ensuring user friendliness and experience.

A few notable features of our system are mentioned below:

Login: We had the capability to login as particular user. This was the first step to start using the messaging system. The password were all stored in the db in a one way encryption format.

User and Group Messaging: We had all CRUD operations in place for users and groups. Users were allowed to change their password, delete their account and read their messages both sent and received. We formatted our messages to display sender and receiver and also group name for group messages. Our messages also provided feedback to the user on successful delivery or failure. This elevated the user experience. Using our system you can also send messages of different MIME types like images, gifs and videos.

Message features: We enhanced certain message features before the system was ready to go live. We maintained a parental control switch to avoid the use of vulgarity over messages. This is something every user can control by turning it on/off. We added the capability to recall a message sent in the past. Users can also search for messages in their feed based on sender or receiver.

Compliance: We ensured industry standards by incorporating CALEA compliance and the ability for an agency to intercept messages if they brought forward a legal subpoena. All our messages our encrypted over the network and will be sent as is(without being decrypted) to the agency that requested a wiretap.

The goals were developed keeping in mind that we could have an enhanced system that performed smoothly as a messaging system capable of the above mentioned features which enabled users to have a positive experience.

Results

We started out by splitting tasks during our first scrum meeting. We also prioritized backlog tasks for the nex sprint if any. We would meet every odd day to check on progress, blockers and any anticipated changes in understanding of the requirements. This format worked great for the team and also helped us work parallelly as shown by the graph in Fig 1.

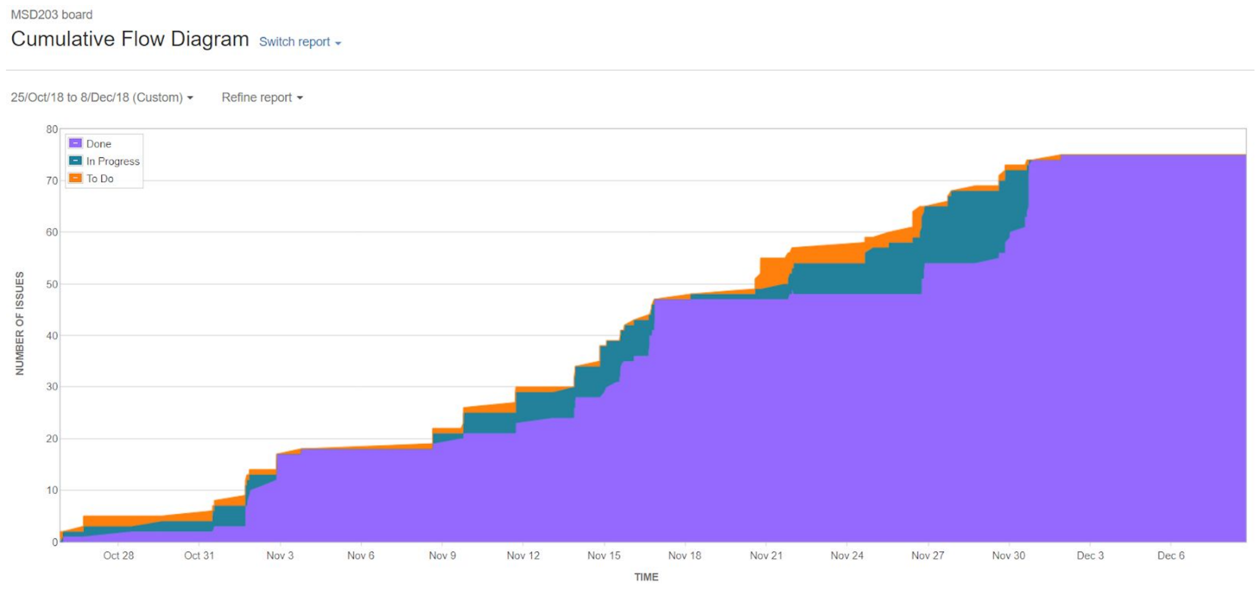
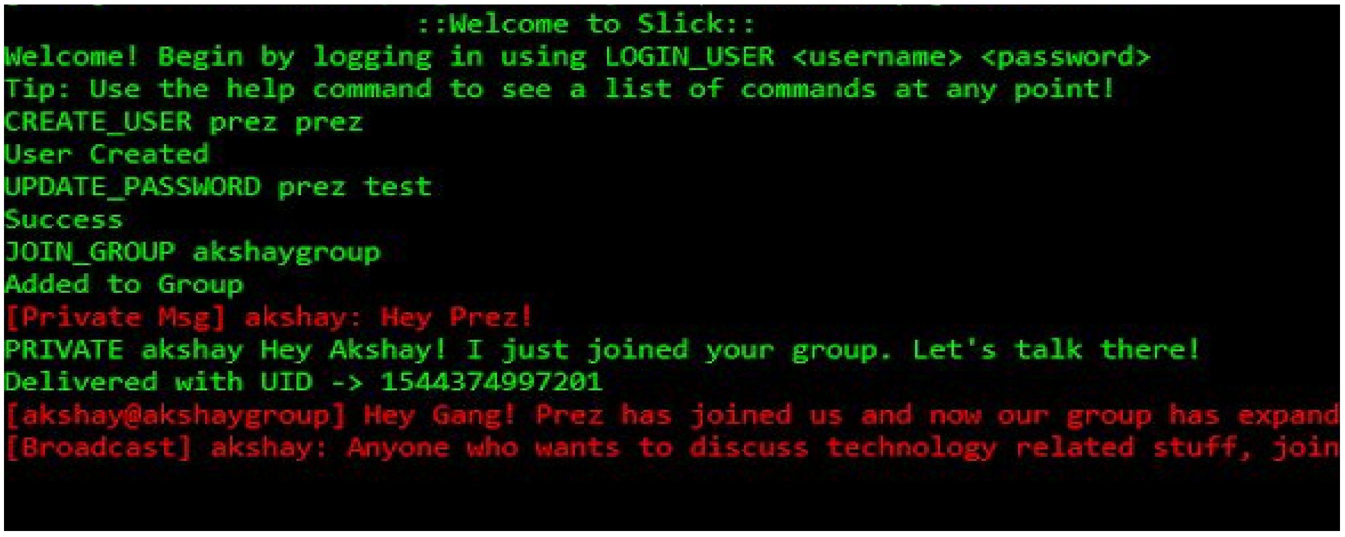


Fig 1. Jira tasks from creation to completion

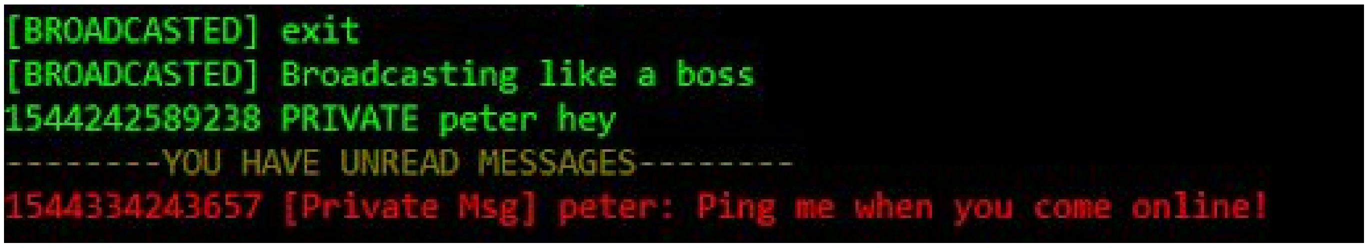
The feedback to the user was well received since this was a terminal based application and showing UI components was not an option we could consider. We ensured the user could distinguish between messages and was able to view his history upon logging in as shown in Fig2.



```
                        ::Welcome to Slick::
Welcome! Begin by logging in using LOGIN_USER <username> <password>
Tip: Use the help command to see a list of commands at any point!
CREATE_USER prez prez
User Created
UPDATE_PASSWORD prez test
Success
JOIN_GROUP akshaygroup
Added to Group
[Private Msg] akshay: Hey Prez!
PRIVATE akshay Hey Akshay! I just joined your group. Let's talk there!
Delivered with UID -> 1544374997201
[akshay@akshaygroup] Hey Gang! Prez has joined us and now our group has expand
[Broadcast] akshay: Anyone who wants to discuss technology related stuff, join
```

Fig 2. Color coded messages and feedback on login and CRUD operations

We also showed the user the messages he received when he was offline. This was split up fro his history as “Unread Messages”. This can be seen in Fig 3.



```
[BROADCASTED] exit
[BROADCASTED] Broadcasting like a boss
1544242589238 PRIVATE peter hey
--------YOU HAVE UNREAD MESSAGES--------
1544334243657 [Private Msg] peter: Ping me when you come online!
```

Fig 3. Unread Messages view

The development with regards to compliance went smoothly and we now offer the facility of tapping into communication between users provided we’re presented with a valid legal subpoena.
We received feedback from the TA on numerous occasions and constantly worked on making our system better. We changed our bag of words approach for parental control to a 3rd party API which was faster for the system. We also added more MIME types like gifs and videos. Parental control was a toggle switch whose control was with the user rather than modifying messages for the entire system on the fly. These tips definitely enhanced our system to a greater extent.

Development process

The development process was split into modules such that each team member owns a module and becomes an expert in his module. This helped in assigning issues to owners and thus tasks and bugs were solved faster. This also helped in assigning tasks and splitting work more easily.
The process was split in three sprints, each running over two weeks. Each sprint had minimum expectations, stretch goals and environment goals to be completed in that time. If a task turned out to be incomplete or deviating from the ideal functionality then these tasks were moved to a backlog and prioritized for the next sprint. Tasks were split up during our first scrum meeting and we would meet regularly to talk about progress, blockers and change in ideas. If a team member completed his task sooner than expected he would help another member out or pick up another task from the queue.

Test-Driven development is something our team had not encountered before and this was a wake-up call especially when we realised how hard it was to achieve test coverage during Sprint 1. The codebase was initially taken with a pinch of salt since none of us were familiar with networks and its jargon. So we set out to clear the 85% coverage mark. Finalizing on the DB was not done with confidence since we did not know what was in store for the future.

Our decision to select MongoDB was based on this unsurety. Unstructured DB for unsure specs was a safe bet, We split the tasks into achieving code coverage for different classes. During this process we realised socket errors, bind exceptions and did everything possible to resolve these failures. Setting up a slack notifications and AWS environment was easier since we were well versed with these technologies. Using smart commits with Git was initially not followed stringently but it became a habit eventually. We were not able to do most of our stretch goals since we underestimated how difficult achieving test coverage would be.
Looking back, we would explore other tools like Mockito to test a network flow which would have avoided bind exceptions and socket errors. We would also want to look at running Junit 5 test cases serially. The team worked hard and work was split equally. Help was provided wherever and whenever anyone could.

We had pending CRUD operations for Group which was prioritized in Sprint 2. During this phase we were fairly familiar with the code and had ideas for what needs to go into what module. We split tasks during our first scrum meeting based on familiarity with modules such that everyone had equal amounts of work to be completed. We used Jira for task management and each one was aware of what other team members were working on. MongoDB was finalised and test cases were written on the fly to avoid problems we faced with test coverage in Sprint 1. We built our messaging system and it worked flawlessly. We persisted users and groups, their messages and encrypted passwords. We did not save our POJO objects in the DB but rather using our own json format. We hit a roadblock with MIME messages and were unable to send videos and gifs. We did accomplish sending images over the network.
We then looked to make the system more user friendly given our learning from the UX/UI slides.This was fairly restricted since we were using a Terminal instead of a GUI.
We incorporated

- different colored messages for sent and received messages. See Fig 2.
- a HELP command that showed all the commands and their syntax. This was to avoid user recall from memory.
- User feedback to keep the user informed of messages sent and actions completed. See Fig 3.
- History messages to be shown on successful login. See Fig 3.

Looking back I think we did much better with more organisation and control. The codebase felt familiar for similar tasks of User and Group. The UI/UX changes was welcomed and we received a bonus for the same. History messages was helpful on revisit.
MIME messages for types apart from images were prioritized for the next sprint. Smart commits became a good habit and code smells were kept in mind while coding.These improvements kept our codebase cleaner and easier to maintain keeping stretch completion as a priority. Environment changes for Slack, Jenkins and AWS were easily integrated.

In our next sprint we had functionality principally involved around routing of messages, Compliance factors and subpoenas. By this time we the codebase felt like our own and we didn't stress too much about the requirements. We tried a new approach this time to split functionality amongst us and complete them end to end instead of based on module familiarity. This was done so that other team members learn the codebase as well and no one member is the sole point of knowledge. This worked well in terms of gaining knowledge of the codebase but reduced our work rate by a considerable amount. We switched back to handlings modules we were experts in so that we could complete the sprint sooner.
We made a few assumptions in this sprint for Recall and parental control and this had us do some last minute rework. We assumed Recall to be for last sent message only and parental control was not a switch that could be controlled by the user. Compliance tasks had a few changes to be handled by our existing system. These were changes we expected and were prepared for. Creating and handling Subpoenas was an extension and hence this didn't interfere with existing functionalities which gave us the freedom of implementation. We changed our implementation of parental control from a bag of words to using a third party api. This earned us a few brownie points.
Looking back I think we should have stuck to splitting tasks based on familiarity and expertise. We should have confirmed our thought process in Parental Control and Recall before implementing. We slacked a little bit on building test cases as we developed and hence was a little harder to achieve code coverage in the end. JMeter had a steep learning curve. Environment stretches were again easy to handle.

Retrospect

We our amazed by how we started out this project to where we stand right now. We really didn't expect to build a fully functional Messaging system. This was a great learning experience whilst following good coding practices. Managing software is something we're pretty fluent in now given our experience with Jenkins, Smart commits, CI/CD, Slack integrations for monitoring and security. We found this project definitely improved our skill set in other aspects of the SDLC apart from coding. We also gained a lot of Network knowledge from this project, something that unfortunately nobody in our team was familiar with. We have now overcome our fear of sockets and bind exception barring the occasional nightmares. We also valued the knowledge we gained in terms of compliance which broaden our scope of thought for future projects.

Through the course of this project we paid attention to subtle aspects of software design like UX/UI. We realized sooner than later that having fully functional code is not enough for a welcoming system. We provided user feedback for every action taken like successful login, successful message delivery etc. We also color coded messages to differentiate between received and sent messages. We tried to incorporate Nielsen's design principles as much as we could for ease of use especially since we were using a terminal.

A few important realisations are mentioned below:

1. *Testing is important for a developer:* We are now aware of the effectiveness of test-driven development. We could say with confidence that a particular functionality we developed, works fine since we had test cases in place. We also realised that we caught a few extra bugs that generally went unnoticed by covering branches and and loops.
2. *Code smells and practices:* We definitely improved writing quality code with SonarLint in place. Silly issues with building Strings, using curly brackets for if-else statements, catching exceptions etc. are now avoided and we're glad we learnt these.
3. *CI/CD*: Automation of deployments and having checks in place before code is committed was very resourceful and these industry standards have definitely given us an edge in building robust software. We also reduced the risk of having a bad codebase online.
4. *Team coordination*: This is by far a great learning experience as we established a lot of communication over slack and monitoring of task on Jira. Git merges were a hassle in the start but with better communication we improved by a big margin. Communication was key. We're definitely better listeners than we were before and also ask better questions when we receive a technical specification document.

A few roadblocks we faced along our way:

1. *Jenkins queue:* We faced long wait times in the jenkins queue and that wasted our time. We tried to solve these by starting early but we still ended up facing longer wait times as we got closer to the deadline.
2. Knowledge Transfer: We had a couple of issues in understanding the codebase in the beginning and since this was setup by someone else a knowledge transfer or a walkthrough of the code would have been helpful.

Conclusion

Overall, we had a positive project experience implementing weekly in-class knowledge to a messaging system that became more sophisticated over time. We learnt a few important things in software development that we've added to our bag of tricks. We have also improved our communication in terms of clarifying technical doubts with someone who is not very technical. We feel that we have more skills to show on our resume than before and also take away a little network knowledge. At the end of it all we can now proudly brag about our new Slick messaging system.