CS5500 - Foundations of Software Engineering
Spring 2020
Team 2 - Final Project Report

(Purvil Bambharolia, Dipen Patel, Devanshi Ganatra, Jainam Sheth)

Introduction:
The overall goal of the project was to develop/build a secure messaging system which focused on features similar to the popular messaging application - *Slack*. Features like group messaging, secure messaging, exchanging files, thread based conversations, and easily accessible front-end clients were at the centre of the project.

Project Goals:
Our team at *Slick Communications Inc*., aimed at completing all the requirements from the product backlog provided to us by our client. We set out a 2-month timeline consisting of 4 sprints, each comprising 2 weeks in order to accomplish this. The product backlog/project goals can be divided into the following components -

1. **Messaging** - Messaging should be secure with the correct encryption technique in place. Messaging must support emojis, and any other UTF-8 supported character. Also, messaging should support exchange of any type of media files such as images, videos, document files, etc. Each message should also be associated with timestamps and expiration timestamps. Also, the messaging system should be capable of deleting/hiding some messages. Users of the system should be provided with an ability to reply to a message directly, thus forming a thread of conversation.

2. **Users** - Users should be able to sign-up, and then login into the system. A user should be able to initiate a conversation with any other user seamlessly by searching them using their username. Users should be able send any type of message to any other user. Users should be able to set their profile picture. Users can also set custom profile pictures for a specific user. Users should be able to create, invite other users to a group.

3. **Groups** - Groups are a set of users communicating with each other through a privately accessible channel. The group can have a secure passcode set by its moderator. Users should be able to access the groups they are part of using group passcodes. Moderators should be able to accept/reject invites to the groups made by other users. Any user inside a group should be able to create a poll inside a group, to which all the users part of that group can respond. Moderators should be able to kick a user from the group they moderate.

4. **Government** - After receiving/issuing a subpoena, the system must provide the government the ability to track a user. Tracking a user involves logging and providing all

user activity including the private messages that the user has had with other parties of the system.

Results:

After a total of 4 sprints and 8 weeks, we managed to create a well-working Slick Communications back-end system using Java, websockets and many other technologies and our front-end client using Angular. Our server backend application supports all of the above state requirements and possibly also layouts possible extensions to the project. We utilized a Model-View-Controller architecture where in our controllers exposed APIs for the front-end to interact. Also, to note, we also kept in mind scalability and hence the way we architected our backend server, its possible to convert it to a microservice architecture with minimal effort.

Talking about our tech stack, we choose MongoDb as our database instance due to the fact it provides high availability and scalability right out of the box. We hosted our MongoDb instance on MongoDb Atlas due to its self-managed cluster system. We also managed to develop a front-end client using Angular.

We ended up using a lot of popular design patterns within our code like Singleton, Command, Builder pattern. We also devised our own socket messaging architecture allowing the client and server to connect seamlessly using sockets and perform commands on each other like sending messages.

Talking about our deployment processes, whenever we pushed a commit to a branch, Jenkins would be responsible for building, testing and validating the codebase for this commit against a certain predefined set of metrics using SonarQube. SonarQube quality gate includes - Bugs, Vulnerability, Coverage, Unit tests, duplications, code smells. We managed to keep our code smells and duplications as low as possible, however, we faced a very particular issue with one of our socket messaging classes. We tried resolving duplicates using abstraction, however, we weren't able to do it 100% as we were running into some Tomcat server issues with the abstractions proposed.

We also managed to keep our unit tests steady. We didn't merge any feature requests until there were unit tests for it. This allowed us to keep our coverage steady and also prevented any bugs.

We used Amazon S3 to manage/store user data that is being exchanged between two or more users.

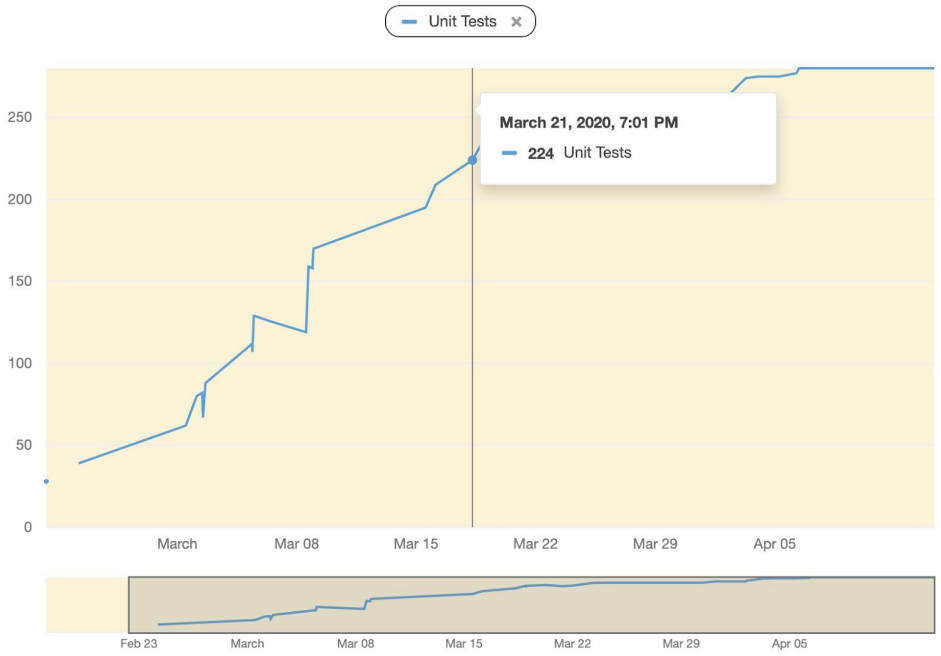

Fig: Number of unit tests against time.

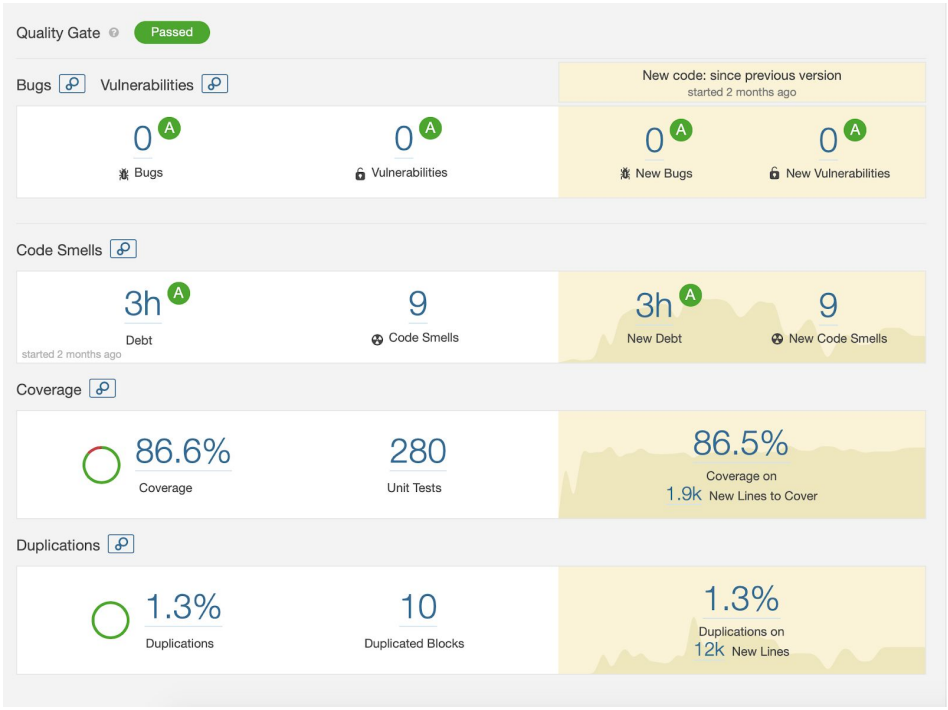

Fig. Quality gate metrics for our final code.

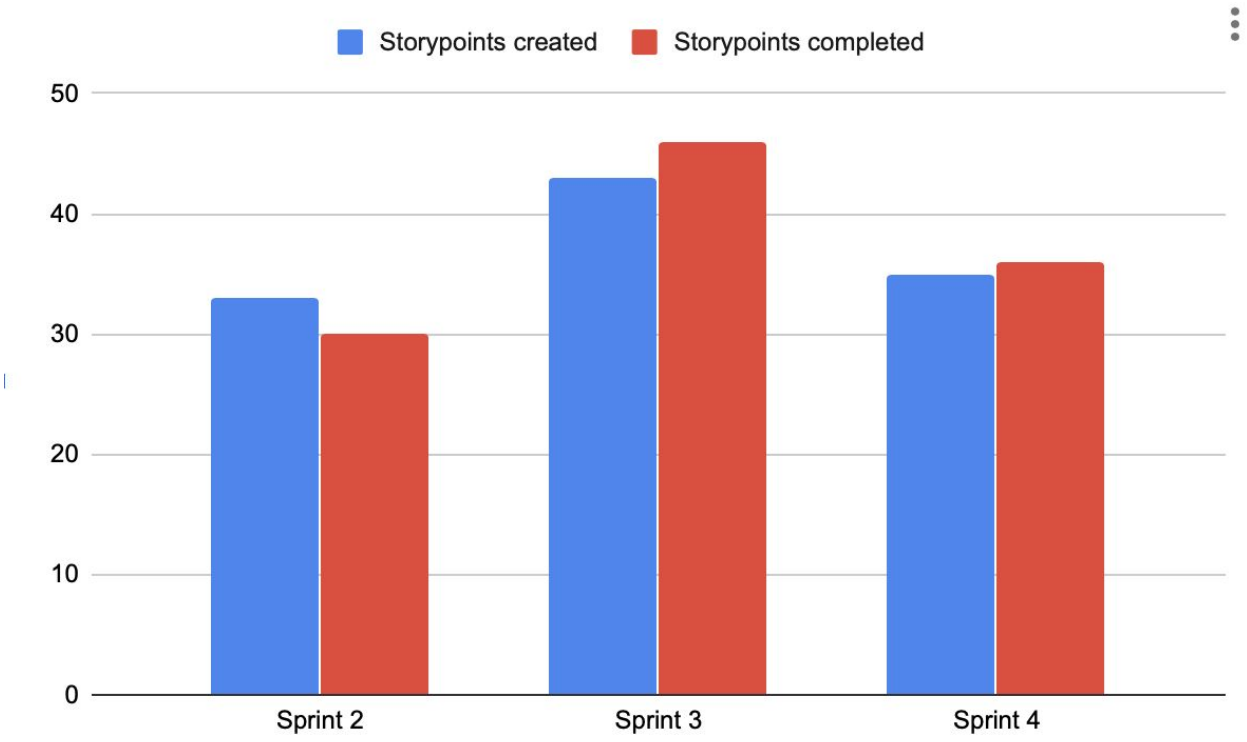

Fig. Sprint story points

Development Process:
Our first sprint consisted of meeting and getting to know each of our team mates. We collectively carried out small team bonding activities to have a solid head start. We discussed each of our strengths and weaknesses. We also emphasized on how important daily standup and weekly meetings are, and also set up calendar invites for the same. We had regular daily standup meetings upto 15 min to sort out issues and generally resolve any conflicts.

We already had Github, Jenkins, and SonarQube setup for our project. We used SonarQube to track how our application development performed against quality gates such as bugs, vulnerabilities, unit testing, coverage, etc.

At the start of every sprint, we laid out objectives for the sprint and used Github Projects feature layout for our sprint plan. We added/created issues for these sprint objectives and distributed them within ourselves. Distributing issues within our team prevented team members from getting lost within the development process. At the start of the first sprint, we laid out project objectives and created documentation such as SRS, use-case diagrams, etc. We also understood the initial codebase and tried playing out with the codebase given to us by our instructor. We also set up our slack channels to include plugins for Github and Jenkins.

During the second sprint, we worked on making services to handle database CRUD operations, create model classes, set up user/message controllers to expose APIs to the front-end client,

and started development work on a CLI based client. We made significant use of various design patterns such as Singleton, Command, Builder, etc to prevent anti-patterns. We also managed to create our own socket messaging based architecture, which we were able to implement for simple one-to-one messaging.

During our third sprint, we managed to implement group controllers, expand socket messaging to group messaging, and started development on Angular-based web application frontend. And also expanded our database services to include aggregation functions. We implemented profile pictures and other user preferences for the users.

During our fourth and final sprint, we focused on integrating everything together. We made group messaging work with our front-end client. Also, we managed to complete government related tasks.

We also managed to keep code review sessions during our sprints. Also, we bumped up the number of reviews per PR required to 2 to enforce consistency and code knowledge within the team. We also made sure the branches were deleted after they are merged into the master branch in order to prevent cluttering of branches.

We followed SOLID principles during our entire development process.

Retrospective:
Overall, the project has been a great learning experience. We have learned alot from this course and project. It has not only enhanced our coding skills and pushed them beyond our limits, but also our collaborative skills. All of us on the team are very proud of the team and project/product that we have built within just 2 months of time. We really enjoyed working with different design patterns that were involved across our whole codebase.

The project also helped honing our programming skills by actually implementing popular design patterns in real-life projects. We also did a lot of code reviews which are one of the most essential things in the technical team. We are sure that all of this learning would be ultra-helpful to us during our full-time opportunities. Also, the project has been a significant plus on our resumes.

Room for improvement includes - each of the teams should be given the tasks to set up Jenkins and other operations on their own. This would allow the students to learn how the operations team works. Also, we do not feel very confident to admit that we learnt any part of Jenkins or Devops technologies with this course. It was all development that we learnt.

Apart from that, we feel very accomplished and great about this course and project.

Thank you
Slick-client-team-2