Northeastern University

Revamped Prattle

By

Chandupatla Vinay Reddy
Yilin Xu
Chuhan Zhang
Arvindkumar Thiagarajan

CS5500 : Foundations of Software Engineering
Prof. Michael Weintraub
Omar Tuffaha

Table of Contents

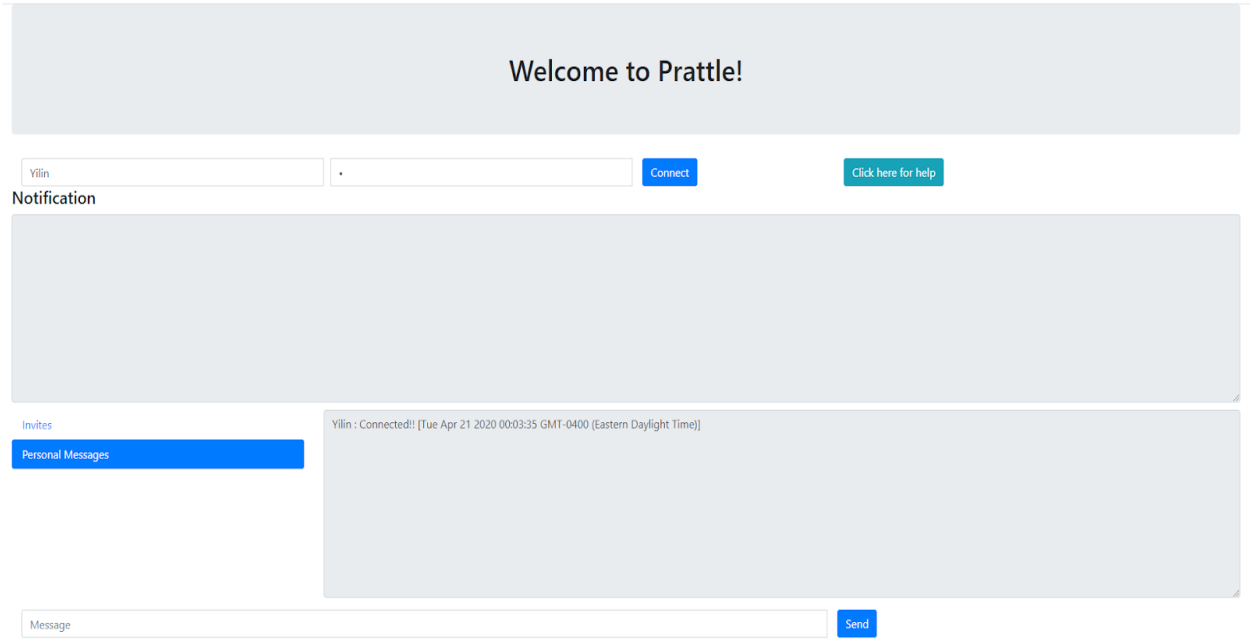
1. Project goals

The main goal of the project was to establish a straightforward and easy to use communication system that would enable a user to primarily use their created account to chat with their other users via group or personal messaging. The features of the application are summarized in the following points:

- Creating a user with a unique username and a password. Can also associate email-id with the user.
- Updating and deleting a user by their username.
- Direct communication with another created user.
- Broadcasting messages to all currently connected users.
- Attaching a timestamp to every message to view the time it was sent.
- Persisting all data including user information, messages, groups, follow-list, etc in a secure database hosted online.
- Messages that are sent to a user who is currently offline must persist and be sent to the user once they are connected with their order preserved.
- Creating a group with group name and moderator.
- Deleting and updating groups. (capable only by group moderator)
- Adding or removing a user from a group. (capable only by group moderator)
- Enable a user to follow and unfollow another user. Personal details of a user can be seen only when that user is being followed.
- Enable and persist invitations to join a group. A user would be able to accept or reject this invitation.
- A user can send messages to groups they are a part of. They can also choose to leave a group out of his/her own accord.
- Adding functionality for government usage, whereby the government can track user activities and their history with the application. They can also stop tracking said user. This functionality is implemented
- Logging the application to append all actions & events that occur in the app to log files that would persist locally.
- Improve the front-end design of the application to make the interface more appealing and easy to use for the user.
- Thoroughly testing the application to maintain excellent test coverage.
- Maintaining a clean and bug-free application, minimizing code smells, and refactoring code to improve readability and reduce redundancy.
- The application must handle warning/error cases such as trying to connect with a non-existent username, sending messages to non-existent users, etc well gracefully by notifying the user with appropriate messages.

2. Overview of the result

The previous section lists the goals we had for this project. In this section we explain how we met our goals and satisfied the base requirements for this project by implementing the following features and quality assurance processes. Below is an image showing the application after 2 months of our work when we were writing this report.



2.1. Application features:

a. Broadcast messaging

Any message sent normally on the prattle app without any special commands would be sent to each and every user registered on the app whenever they come online.

b. Group messaging

Users can create their own groups and add any users they want to their group. The owner of a group is known as the moderator, by default it is the creator of the group. A moderator can appoint any other user in the group as a moderator. As a moderator, a user can add other users or invite other users to the group.

c. Personal messaging

The application supports user-to-user messages as well which means that the messages sent using this feature will be visible only to the receiver and no one else. In other words, this is point-to-point messaging.

d. Connecting with other users

One interesting feature which makes this normal messaging app a social media app is the ability for users to follow each other. By following a user, one can view that user's information like name, email address etc. This feature could be extended to incorporate more information about the user like a profile picture, birthday, current status etc. A user can check his followers at any time by using the "/following" command.

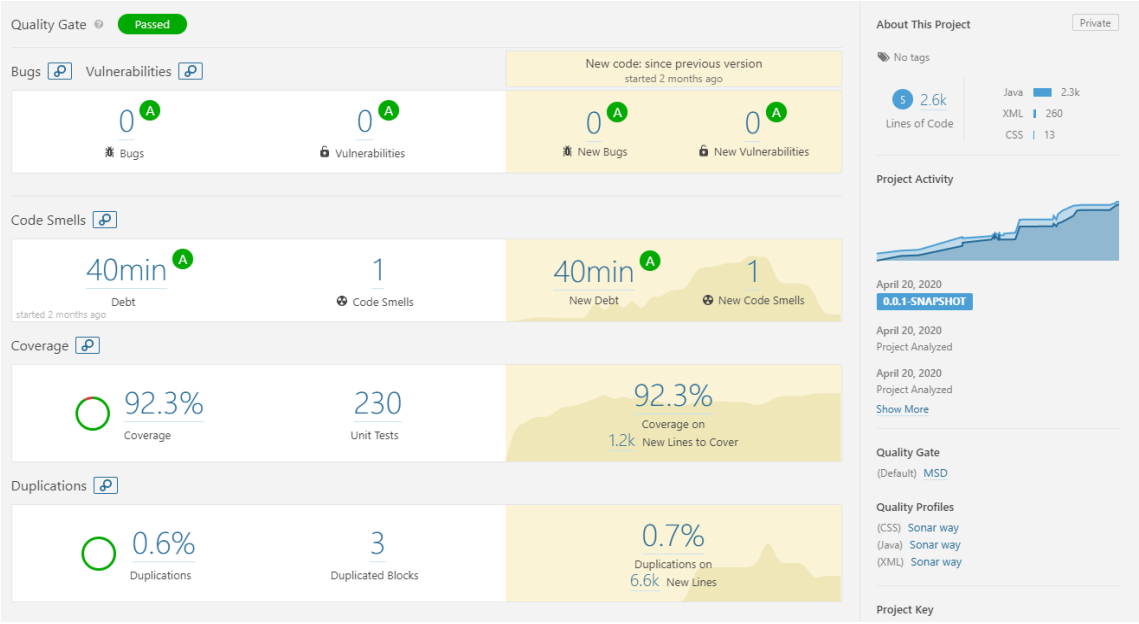
e. Government surveillance

A special feature of this application is the presence of a super-user called government user who can monitor all the activity/messages in the app. The government user can track specific users and be notified of their activity. The government user can check the users being tracked at any time and untrack them if he/she wishes to. Although this might seem like intrusion of privacy, this might be a feature that most corporate communication applications prefer nowadays.

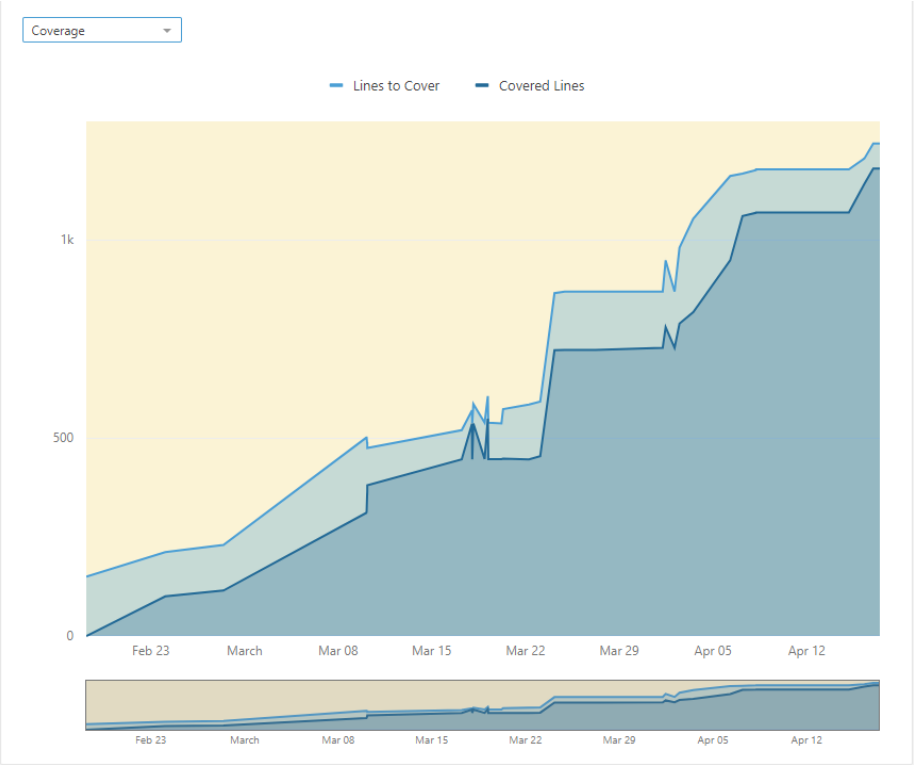
2.2 Quality assurance:

Quality must be prioritised over quantity when developing an application, or in Professor Mike's words "It's not the number of features that I care about, it's the quality of implementation of those features that's important". We believe in the same principle and hence, used tools like Jenkins, Sonarlint and Sonarqube to ensure that we followed the best practices to write every line of code.

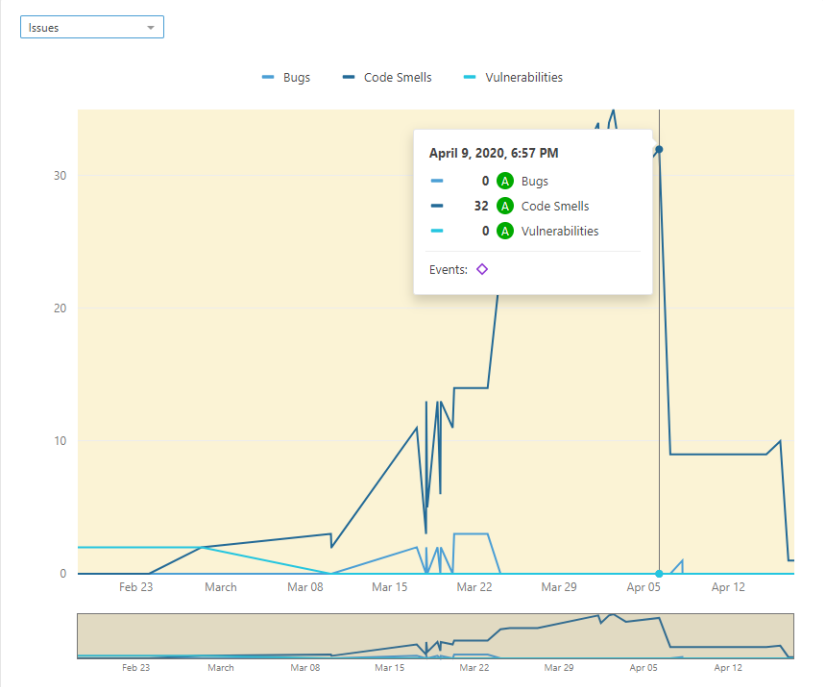
Below is a Sonarqube report of our final code which shows the test coverage (92.3%) and code issues (just 1 code smells). This analysis was done on April 20th.



The quality of the code has improved over time which is evident from the sonarqube reports for the line coverage and code issues below.



Above graph shows the test coverage over time



Above graph shows the code issues over time

3. Team's Development Process

The team applied the SCRUM method in the project. Each team member has the chance to be the SCRUM master of a sprint. In the beginning of each sprint has a planning session, which creates the sprint goals and story points from the product backlog. Then the team will begin to implement the story points. At the end of a sprint, there will be a team sprint review and personal sprint retrospective. The whole project has 4 scrum sprints in 8 weeks. In these 8 weeks, we used different forms and tools to increase our productivity, maintaining a high transparency inside the team. In the first two sprints,offline meetings were held in Snell. After moving online, we held a lot of online meetings. Software like Teams makes online meetings a convenient way for team discussion. The following paragraphs will review the development process in chronological order, among which for each tool we used there will be a discussion about it.

3.1. Review of the development process

In the first sprint, our major work is about understanding and analyzing the product requirement and then figuring out a reasonable backlog, including class diagram and sequence diagram. We used a lot of paper to express our ideas and design. Paper and pen are very straightforward which made our communication effective. The face-to-face brainstorming made everyone in the team understand the project quickly. Ideas, concepts and assumptions were discussed again and again between different teammates. When we worked on the SRS, Google Docs was very helpful as 4 of us can work simultaneously on the same documents. We used Google Docs a lot in the latter teamwork.

From the second sprint, implementation becomes a major part of sprints. Git is a cornerstone of teamwork. Our git branch model has a master branch, a develop branch and feature branches. This model saved us sometime from merge conflicts. Sonarqube was also used to measure the code quality. It's a powerful tool for structure testing. So we began to install Sonarqube to our local machine from the beginning. We have one working Sonarqube local installation from sprint 1. However, integration of Sonarqube and maven resulted in a problem that was hard to debug. It has something to do with the Jacoco plugin version. The team spent a lot of effort on it. We eventually solved the problem by trial and error. We learnt how to switch on maven debug information output and gained some insight about Jacoco initialization process.

In the third sprint, we used github issues and milestones to track story points and sprint goals. Kanban was also used during this sprint. The issues and milestones function well at keeping track of development. Kanban system was not used so often. There are two possible reasons. First, the project is small, there wasn't a lot of staff to track. Everyone could maintain a reasonable and effective mind model of our software in the mind. Second, teammates had sufficient communication with WhatsApp, Teams and phone calls. Once we have issues, we can directly discuss with the teammate who is more experienced at them. Maybe Kanban could be more useful for a more complicated project.

In the 4th sprint (also in the 3rd sprint), efforts on refactoring and debugging increased as the project accumulated a considerable amount of code. And our code becomes more intertwined with each other. As a result, we did many pair debugging/programming. More communication happened in the last 2 sprints. We tend to report our work status in our WhatsApp group. It's less formal but it's really convenient. A screenshot with a few sentences will make everyone aware of your current situation. With chat history, anyone can catch up the progress and contribute new ideas. If there is a way to summarize our discussion and push it to the github, then we will have good documentation about how our software was developed in those sprints. Sonarqube played an important role in removing code smells. When using Sonarqube, it also reveals how difficult it can be to reach a higher condition coverage. Some conditions are not reachable due to technical limitations.

We tried to deploy our application on different platforms: AWS, Heroku and Khoury Cloud. AWS's interface is powerful and professional, but also intimidating for newcomers. Heroku is free and easy to use. It provides git deployment and maven deployment. We tried both methods. Maven deployment requires less modification on the current code base than git deployment. Deploying on Khoury Cloud is just like deploying the application on a local Linux machine. After installing Tomcat, maven deployment can be used.

3.2. Summary of the teamwork

Overall, the team did very well on communication and collaboration. We find the whole team experience is a great opportunity to learn new staff, including software development methodology, design patterns, testing strategies, automation tools and Java skills. The teamwork helped us share our experience and knowledge. Discussions between members working on backend and frontend increased our understanding of the whole software system. The debug experiences (trial and error, discussion about possible reasons, explanation of solutions) usually gave us some insights about the code we were working on or the software tools we were using.

Automation is essential in teamwork. The team has 1 mac user, 1 linux user and 2 windows users. With the help of maven, we can build, test and measure test coverage of our project in the same manner. We also used maven to deploy the application to tomcat or heroku.

In terms of obstacles we meet during the development. Most issues were solved by pair debugging or group debugging. Those experiences were very helpful in terms of learning. However, in the end, we still have one code smell left. The reason is that this code smell is in the core part of the message parsing process. The original choice of command format was lack of thinking as none of us has command line interface design experience, which makes it hard to refactor in later development. Meanwhile, the team's later development mainly focused on implementing more features, which made the problematic method even more complicated. The refactor will create a lot of extra work on the existing test cases. A possible solution to avoid this situation is, we should have a balanced portion of adding functionality and refactoring the existing code in our daily development. Refactor should be done more frequently. As Martin Fowler said, refactoring should be an opportunistic activity, it should be "done whenever and wherever code needs to be cleaned up - by whoever."

4. Retrospective of the project

Areas we liked the most:

- Complicated yet well defined project requirement
  The client description of his requirement is although not technical, but very detailed and thorough on application features and behaviors he wants. This helps us to define SRS very quickly and start coding.

- Communication between develop team and product owner
  Throughout the 4 sprints, the product owner was readily available to clear our questions on expected outcomes. The short sprints help us maintain consistent communication with the product owner and make sure our product doesn't drift far from the client's requirement.

- Stricted tech stack but prototype application was provided
  Though we had restrictions on the technical stack we are allowed to use, a prototype application was provided to us for reference. It helps us a lot in understanding how websocket, json serialization works in java backend.

Area we don't like as much/ What we learnt:

- There were restrictions on the library and framework we can use to help our development. Unfortunately, Spring framework isn't allowed which costs us lots of time to write and test basic CRUD query to database. But we did deepen our understanding of JPA because of the lack of assistance of Spring framework.

- Teamwork is the key. Through the project, we sharpened our teamwork skill and learnt that consistent communication and collaboration increase our code quality and quantity by many fold.

- In the technical perspective, we learnt and conducted Agile scrum development. We had exercised continuous integration and ticketing systems throughout our project. We learnt and applied several design patterns in our project including builder pattern, composite pattern, observer pattern.

Changes to the course:

- Short introduction to how the prototype code works would be nice. Especially the part where the websocket.js interacts with java chatEndpoint.