Team 9: Tanner Huynh, David Sadowsky, Alice Tilles, Dian Wang CS5500-Weintraub

Final Project Report

Purpose Statement (Goals)

"BOWSHOT will allow users to communicate with other users through direct, private messages or in group channels. Users will be able to create and join channels that are moderated by other users." - *Team 9's SRS*

We wanted the frontend to be fresh and the backend to be reliable. For the timespan allotted, we expected to be able to build a minimally-viable product, a thoroughly useable, Slack-inspired chat application, with no special capabilities provided.

Desired Functionality Offering

At the specified url (hosted on an AWS ec2 instance), a user should expect to see a landing page that offers the user the options to either log in or register as a new user. This portion of the site should be username/password protected, have limited login attempts per user, and provide feedback to the user about the current status of their registration/login attempts. All pages should be responsive to window resizing, so that a sub-optimal size will alert the user and prompt them to resize correctly.

Upon entry to the application, the user will be shown a nav-bar on the left, featuring some basic information personal to the logged-in user, the names of other users in their network, and the group chats ("channels") the logged-in user is a member of (#general being the only automatically subscribed channel). Users can customize their profile by writing a bio and choosing an avatar.

From the initial logged-in view, users can search for groups by typing the name of the group, then clicking a "Join" button. Users can also start or open a direct message (1:1) conversation with any users displayed in the nav-bar. These conversations persist even after the user closes the app window. A conversation history is shown in the center dominant portion of the window, with messages from the logged in user being right-aligned, and all other users, left-aligned, as is convention in most contemporary messaging apps. The conversation recipient is displayed in the top portion of the window, and the text-entry area displayed below the conversation history.

Messages appear alongside to the sender's chosen avatar, the sender's username, and the message's associated timestamp. Messages are sorted so that the most recent messages appear at the bottom.

A community as a whole has some super-mods, capable of deleting user accounts. Channel mods, who only have extended capabilities within the channels they moderate, can promote/demote other users to/from channel modship.

Hypothetical Forecast

We intended to make sure our product was easily extensible, scalable, and modifiable, so that future releases will be more efficiently developed, and offer more value to the customer after significantly shorter development periods. Operationally, the initial development period for this prototype served as a synchronization and homogenization of team work styles, tool-utilizations, and shared project knowledge. Simply, it doubled as a team-building exercise. The sky's the limit for future iterations of functionality, now that our foundation has been manifested.

Result Overview

We achieved the main goals of what the team set out to accomplish. By the end of the four sprints, our project consists of a live and functional chat app, with a modern, clean interface, capable of handling group and direct chats. We are confident enough in the quality of our project to allow the client to test it out and we don't expect any major bugs to occur. The biggest achievements/milestones of the project included: (1) Deployment to AWS so the app is accessible to any user/device with internet. (2) Ability to private message other users. (3) Ability to message within a group chat ("channel"). (4) A striking UI with considerable thought put into the UX design.

There are some small behavior issues we would fix in the next sprint, such as the required refresh to login to a new account or see an avatar change, and the 'locked out' message to appear after a refresh (if the account is indeed locked out). The team could add improved and more advanced functionality indefinitely (as demonstrated by the constant releases of established chat apps, such as Slack). There are still many features missing between BOWSHOT today and BOWSHOT with all the SRS functionalities, but the codebase now provides a great jumping off point, with the bulk of the difficult, back-end work in place, and ready to be extended.

The team took many steps to ensure a handover to another development team would be smooth and simple, and they could immediately start rolling out releases without wasting time on tech debt or untangling spaghetti code. Some of the measures we took included: (1) A great deal of documentation -- we even used Jira tickets to require this. (2) No single team member being in 'charge of' a section of the code. Having it be passed around allowed many eyes to see it and review, refactor, improve it. (3) Adhering to industry-standard architecture and patterns. This included: common design patterns (e.g., MVC architecture), adhering to the principles of object-oriented design (e.g., modularity), and good naming conventions and file structure.

Completion Statistics

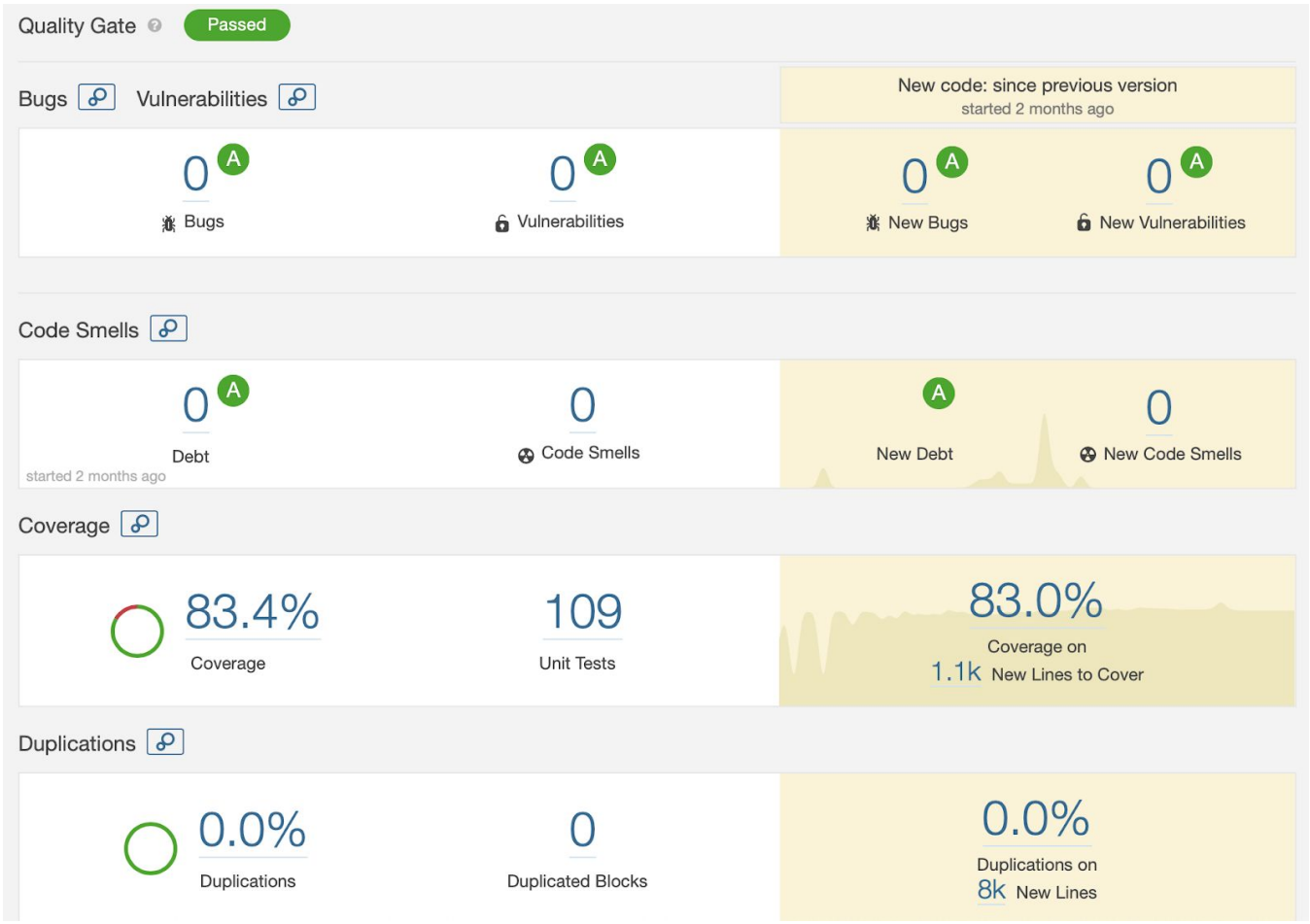
Each bullet point in our sprint plan goals was represented by a ticket in Jira. This is a breakdown of point commitments vs. points completed:

Sprint	Commitment	Completed
FSE9F19 Sprint 1	124	109
Sprint 2	60	49
Sprint 3	80	108
Sprint 4	113	122

- Sprint 1: 1 ticket decreased in complexity estimation; 1 ticket incomplete
- Sprint 2: 1 ticket incomplete
- Sprint 3: 1 ticked increased in complexity estimation; 7 tickets added
- Sprint 4: 3 tickets decreased in complexity estimation; 2 increased; 1 added

Overall, we completed 388 points out of 337 committed, which is a **115%** completion rate.

Quality Statistics



What Worked, and What Didn't

What worked

The more we talked, the more we progressed. This included discussions regarding smaller questions and comments about progress (which took place in frequent chatting in the Slack channel and during group troubleshooting), and during larger discussions and meetings. When team members spoke up about feeling lost or behind, this allowed the rest of the team to help catch that person help, building up their ability to contribute to the project. After every sprint reflection with the client, we sat and reflected as a team for several hours, and planned our next sprint goals. This ensured the team was all aligned and we had consensus. Finally, the scrum master being always available & flexible with their time for help diagnosing a bug or working on issues was crucial in completing tickets.

We found that when we leaned into team member strengths and expertise, we made a lot of progress. The team also decided it was our priority to complete fewer functionalities in a robust, high-quality way, than to complete more, but buggier, functionaties. This strategy set us up for a more professional and dependable final deliverable.

What didn't (+ how to fix)

The biggest hurdle was team member knowledge gaps in parts of the program. While we made amazing progress when a team member leaned into their strengths and produced amazing work, this led to other team members being somewhat clueless, unable to contribute to that part of the system in any meaningful way. By the end of the project, we were working towards closing these gaps. Given more time, we would continue to do so by sharing knowledge in Slack or in person, having more peer reviews (e.g., instead of only person reviewing as a merge-to-master requirement, all must review), and delegating tickets to members unfamiliar with the part of the system the ticket dealt with. Another good idea is to pair up on difficult tickets, having a newbie and an expert team up. Instead of the expert working alone and asking for a review of 200 lines of new code, alienating the newbie, the newbie would be able to ask questions and comprehend the new code during the development process.

Our second biggest hurdle was some team members starting late (frequently due to balancing personal homework for the class). To work around this, the team could figure out the dependencies within the tickets, and give people who start early tickets with the fewest or no dependencies, and give those who start late the tickets that have dependencies. This way, team members would not hold each other up based on personal habits, and there would not be as much conflict. Another solution we implemented during the final sprint and worked out well was to have soft/internal due dates for midway through the sprint for some of the tickets.

Retrospective

The project was a considerable commitment that set high expectations. We tackled this challenge head-on and supported each other using our different backgrounds and expertise. The tech stack proved to be a learning curve and time sink, so the opportunity to learn was immense. In the end, we created a product that we were proud to present to the stakeholders even with all the mentioned challenges.

Due to the relatively short length of the semester and relatively large portion of time spent learning new technologies, we weren't able to complete our commitment list in the initial SRS. In reality, the client backlog was too big of a list for the four sprints. Stakeholders should give a shorter list to set more realistic expectations for the team. We were encouraged to develop "nice to have" features for the app, when in reality, the timeframe and scope should have been focused more on basic features and making sure those features work. We focused our efforts on making the core functionality of the product looks nice and works really well.

Part of our shared exercise in focus was to reduce the amount of new technology we needed to learn in order to make the product work. We did manage to avoid adopting unnecessary frameworks, like React for the front end, to reduce complexity; however, we eventually discovered we had to use or benefitted from using additional tools (e.g., Mockito and Hibernate). We were lucky that each person on the team had their specialization for different parts of the program. We were able to divide and conquer, but also to help other members of the team when they didn't understand the code or necessary tools.

The nature and subject matter called for a substantial time commitment, and we had other responsibilities that made it difficult to meet together. Therefore, our efforts to have remote standup meetings, and lengthy in-person meetings once a week helped us communicate better and stay on top of the commitments. As the semester progressed, we felt more confident as a team, and our development speed increased as our processes were solidified. Branch nomenclature, fibonacci ticket estimation, and a better level of overall organization really helped us produce quality results for each sprint.

Regardless of all the speedbumps and knowledge debt, we are really proud of the fact that we maintained efficient, clean code. We maintained a formidable level of quality and made sure there was always appropriate documentation for the code we wrote. We were diligent to check the local SonarQube and Maven build before merging it to our individual branches for peer review. The team finally had momentum and the project was reaching a critical point where we felt like we could vastly improve the experience in the final sprint. Ultimately, the semester ended before we could complete the SRS commitment list we had initially promised.