Prattle

By

Matthew Sobkowski, Ben Steele, Jeremiah Holbrook and Evgenii Shatokhin

The requirement was to build the next big messaging application. At the applications core, users can retrieve and initiate conversations with one another all while independent of any device. The marketing team provided some ideas for the application and its development which governed the applications overall goals. The main goals of the application can be broken down into 4 different areas. User and groups, communication, government/security, and scalability.

User and Group goals: Clients needed to be able to create users. Following creation of a user, the client should be able to designate a password which requires log in functionality. If a user can create a password, they also must be able to change their password. Users must also be capable of changing statuses (online, offline, busy etc.). Users must be able to create groups. Groups are composed of at least one individual and one moderator. The moderator within a group needs to have the ability to add and remove users from the group. The moderator can also disband a group as well as promote other group users to moderators in which they will have moderator powers over the group.

Communication goals: Two types of messaging need to be offered. Direct messages also known as private messages contains exactly 2 users. Group messages contain a minimum of one user with essentially no maximum number of users. Messaging must be able to send and receive without calling the database. Messages must persist in a queue so messages can be revisited upon loading a message channel. Messaging needs to be time stamped and loaded within the order it is received.

Government/Security goals: The application must be calea compliant. All data stored must be easily accessible in real time upon government order. All employee changes and user changes to the database must be logged and available to produce for potential government subpoenas. Aside from abiding to government calea rules, user data should be secure. End to end encryption must be used, and encryption at rest should also be used for all user messages and information.

Scalability goals: The application must be capable of handling a few hundred to a few thousand users upon completion. This will require a system and database capable of meeting the demands of a high read and write environment.

After four sprints the team met all necessary goals set forth as well as additional stretch features. The system supports user creation and password protection. Users persist in the database allowing for proper login functionality. Users can create groups. When a group is created the creator is set as the moderator of the group. The moderator can add, remove, and promote users. The moderator is also capable of disbanding a group. Direct message and group messages can be created by any user. Messages persist in the database and upon selection of a channel, past messages are loaded onto the screen. Messages are encrypted in transit and encrypted at rest. Messages contain time stamps and load in order of creation. Currently the team uses AWS services such as Cloud Trails and Cloud Watch to monitor our systems to keep track of user and employee interactions with our database. AWS services such as Dynamodb, IAM, and S3 are currently configured to provide real time data of specific users in case of a potential government subpoena. The application uses DynamoDB which is a NoSQL database created by AWS which is currently set to elastically scale based on the amount of user calls.

Upon completion of major goals, additional features were added to the application. The application contains the ability for users to filter out explicit content. Additionally, the application can translate messages to 12 different languages. The testing results are a testament to the quality of the features created. The final SonarQube results showed an 86.4%-line coverage, and an 85.8% conditional coverage.

The development process of the application roughly followed the same process each sprint. Each Saturday following a sprint review the team would meet up to discuss the upcoming sprint goals. During the meetup sprint goals would be created, and each team member would take a specific goal to accomplish for the current sprint. Throughout the week team members would keep in touch to ensure developers were not working on the same code base to avoid merge conflicts. The team would meet up for a long session meeting one week after sprint goals are created to check up on sprint goal progress. Group members would continue working on their assigned goal until Wednesday night which would be the groups deadline for all code completion and testing. Thursday, all team members code is combined into a single branch and final changes to pass SonarQube requirements are completed. Application is pushed to master and the database is staged for demo on Friday concluding the sprint.

Deciding on goals early was both good and bad. It worked out well because members had clear goals to progress throughout the sprint but deciding on two weeks' worth of work upfront lead to some group members receiving a larger workload than others due to the unknown challenges of certain goals. The best aspect of the team's development process were the deadlines. Having the Wednesday deadline to have all work other than the final master push complete, lead to a stress-free environment. The early deadline allowed group members to take an entire day to properly test their work and ensure SonarQube requirements were complete for the final master push. Constant instant message communication was probable the least effective aspect of our development process. It became too easy for some members to become overwhelmed with work without asking for help, or it being identified that the member needed help. Instant messages often result in instant responses. Members may say "things are going well" but in fact they may need help. Verbal communication was much more effective because it allowed for more long form discussion and follow up on progress as opposed to the quick responses of instant messaging. In terms of addressing issues the team was very collaborative in the communication of problems within the group. When issues would arise, the team would often schedule a voice call meeting to come up with a solution.

Looking back on the project the most enjoyable aspect was the autonomy. This was one of the first applications in which we were all able to experiment and try out a wide range of technologies. This not only resulted in the learning of the technologies used within the creation of the application, but the also the learning of the many tradeoffs of alternative technologies. We learned a lot about operations and the use of AWS services, front end development, NoSQL databases, http requests, and design patterns that allow for multiple developers to work on a small code base. The least enjoyable aspect was getting started. Having almost complete autonomy over the design, operations, and features of an application results in having an unlimited number of options to choose from. Although the autonomy made the beginning of the project unenjoyable, it was worth the suffering. In the future to support a great experience, having students get ahold of the project earlier in the semester might be more beneficial. That way as students are progressing through the beginning of the semester course work, they can start to formalize ideas into how the classwork can support the design of their application.