Daisy Barbanel, Davaajav Ganzorig,
Miranda Grisa, Connor Hopcraft, Troy Hu,
Ryan Li, Gray Mackall, Zené Sekou

Milestone 4a README

Submission Location:

https://github.com/ze-ne/cs220-smart-and-secure-messging

The branch you should use is **master** or **milestone-4a**.
Note that **milestone-4a** will always be the same from this point on (so you might want to check out this branch).

Running Tests

Tests should be run following the same instructions for compiling and running unit tests outlined for milestone 3b (in the README).

Second Iteration Implementation Plan

For this iteration we will implement the following use cases:

- Search For Contact/User
- Search For Conversation
- Manage Block List
    - addToBlockList
    - removeFromBlockList
- Delete Sent Message
- Delete Conversation
- End to End Encryption
- Drag to View Part of Hidden Message With Overlay
- End to End Contact Management
- “Smart” functionality (“Look at Analytics” in the original design document)
- Screenshot Message Prevention
- End to End manage information

If we have time, we will implement the following use cases:

- Send Message With Destruction Timer
- Send Image Message With Picket Overlay
- Send ‘Drag and Reveal’ Text Message

These use cases can be seen as "extra" use cases that we will try to implement, but do not guarantee due to constraints. We, however, intend on trying to implement at least one of these use cases.

We are no longer implementing the following use cases:

- Group messaging
  We feel that group messaging functionality will be unnecessarily time-consuming to implement and is ultimately non-essential to the "smart and secure" vision of this application.

Division of Labor

Following our last iteration, we restructured how the work would be divided. This was motivated by the fact that the server team no longer had a significant amount of work to do for iteration two. Instead, remaining server work will be done by teams as needed. Another member was added to the UI team as there is more UI work in this iteration and the remaining two members will be responsible for implementing the smart features of the app. The new division of labor is as follows:

User Interface: Miranda, Zené, Daisy
Backend: Troy, Ryan, Connor
Smart : Dona, Gray

New Unit Test Locations

We added 19 new unit tests; 2/19 pass while 17/19 fail.
There are a total of 84 unit tests; 67/84 pass and 17/84 fail.

/app/src/androidTest/java/com/cs220/ssmessaging/clientBackendUnitTests/UserUnitTests.kt

- `testDeleteSentMessage()`
- `testDeleteConversation()`
- `send_image_message()`

/app/src/androidTest/java/com/cs220/ssmessaging/clientBackendUnitTests/ImageHandlerUnitTests.kt

- *All 10 tests in this file*

/app/src/androidTest/java/com/cs220/ssmessaging/clientBackendUnitTests/ConversationUnitTests.kt

- `testRemoveMessage()`
- `testGetSubConversation()`

/app/src/androidTest/java/com/cs220/ssmessaging/clientBackendUnitTests/BlockListUnitTests.kt

- *All 4 tests in this file*

Untested Functions

As was the case with our milestone 3a submission, we were not able to create unit tests for functions which depend heavily on calls to the database. We leave such functionality to integration testing, where the communication between client and server can be usefully assessed. The functions omitted from unit testing for this reason are marked with code comments and are additionally listed below:

In *User.kt*:

- `getUserIdsByFirstName()`
- `getUserIdsByLastName()`
- `doesUserExistByUserId()`
- `getBlockList()`
- `sendEncryptedMsg()`
- `receiveConversation()`
- `addPublicKeyToServer()`

In *Conversation.kt*:

- `getAnalytics()`

Frontend Unit Testing

Please note that we did not implement front-end unit tests for the same reason as the last Milestone.

- Many of our previous ideas regarding the front end and user interface of the software were unnecessarily cumbersome, and not as streamlined as they could be.
- Many of the frontend unit tests focus on functions that we have decided to implement in the backend. Frontend functionality is thus tested with the backend unit tests as the frontend just calls the backend functions.
- Furthermore, frontend unit tests rely on on building mock ups of various dependencies and classes. Kotlin treats classes as final, so those tests were ultimately unsuited for our needs and have been replaced.

Instead, we run the emulator and operated frontend features, comparing the observed behavior with the expected behaviors of the use cases we planned as a comprehensive form of manual testing.