

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

Submission Location:

<https://github.com/ze-ne/cs220-smart-and-secure-messaging>

The branch you should use is **master** or **milestone-3b**.

Note that **milestone-3b** will always be the same from this point on (so you might want to check out this branch).

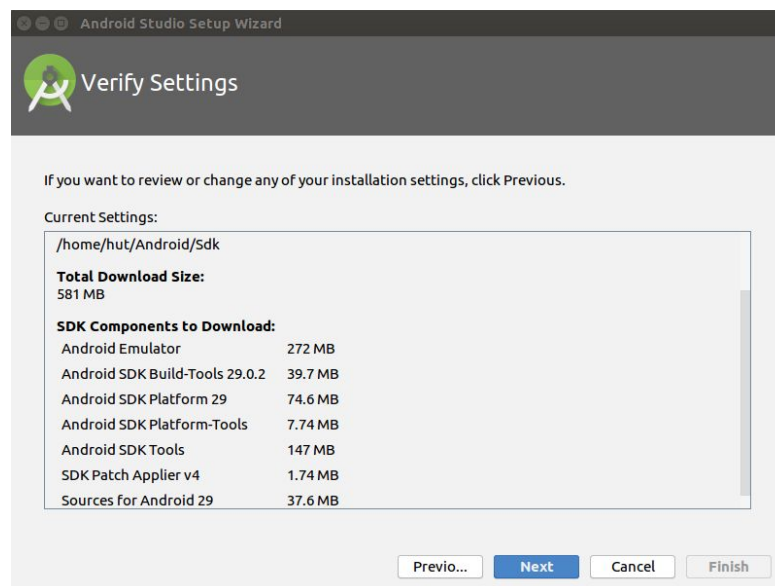
How to Compile

Much of this will be copied and pasted from the last document, as things are pretty similar.

IMPORTANT: In our experience, it is no longer possible to run on the CSIL machines, as the emulator requires too much memory. We therefore suggest that you run this on a Mac/Linux environment with sufficient memory.

Setup

1. First download Android Studio: <https://developer.android.com/studio>
2. Then extract it someplace and go to android-studio/bin and run ./studio.sh
3. You will now be presented with options.
 - a. Choose standard setup keep clicking next until you get to this page



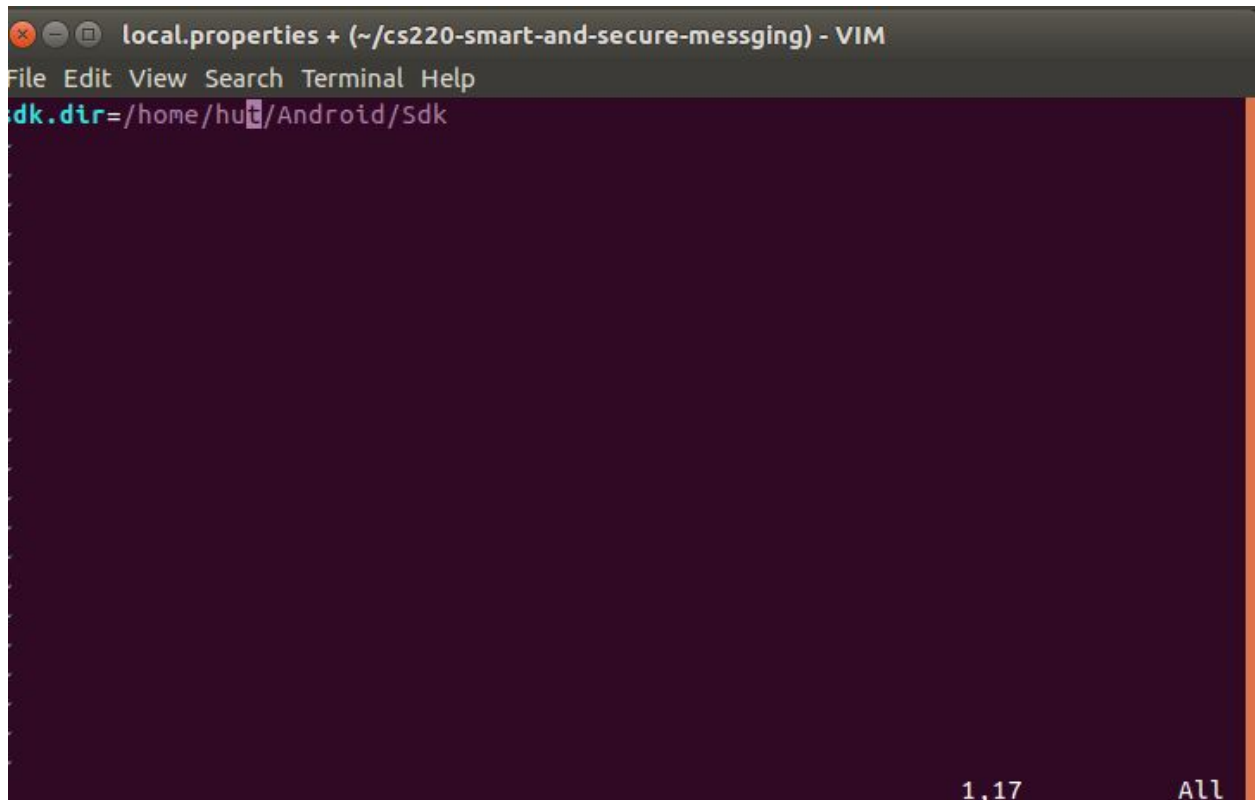
- b. Keep a note of the path given there! It is the path to your Android SDK folder.
 - c. Keep clicking next, install, and finish.


- d. Exit the pop up for the studio. We will use the command line from here on.
- 4. Now you have to accept the license for Android SDK for our tests to work. What you'll want to do is:
 - a. Go to the path shown in the image in 3a. In my case it is `/home/hut/Android/Sdk`
 - b. `cd` into `tools/bin`
 - c. run the command: `yes | ./sdkmanager --licenses`

```
BUILD FAILED in 1m 34s
hut@starmie:~/cs220-smart-and-secure-messging$ cd ..
hut@starmie:~$ cd Android/Sdk
hut@starmie:~/Android/Sdk$ cd tools/bin
hut@starmie:~/Android/Sdk/tools/bin$ yes | ./sdkmanager --licenses
```

- 5. Find a place to git clone the files from the repository.
 - a. Go to top level directory. That is `cs220-smart-and-secure-messaging`.
 - b. Create a file called `local.properties`
 - c. within the file write ONLY:
 - i. `sdk.dir =` the path you noted for android SDK
 - ii. For example: `sdk.dir = /home/hut/Android/Sdk`

```
hut@starmie: ~/cs220-smart-and-secure-messging
File Edit View Search Terminal Help
hut@starmie:~$ cd ~/Android/Sdk/tools/bin
hut@starmie:~/Android/Sdk/tools/bin$ ./sdkmanager
Warning: File /home/hut/.android/repositories.cfg could not be loaded.
[=====] 100% Computing updates...
hut@starmie:~/Android/Sdk/tools/bin$ cd
hut@starmie:~$ cd cs220-smart-and-secure-messging/
hut@starmie:~/cs220-smart-and-secure-messging$ touch local.properties
hut@starmie:~/cs220-smart-and-secure-messging$ vi local.properties
hut@starmie:~/cs220-smart-and-secure-messging$
```



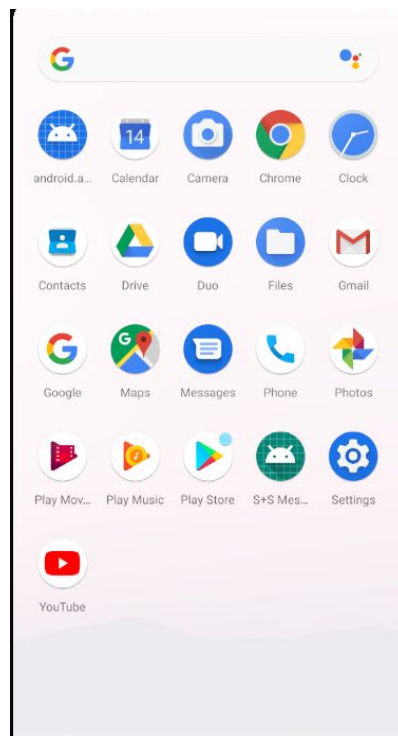
6. Next, open the project in android studio. At the top bar you should see a green hammer, and if you press that the project should build.
 - a. You might need to run a gradle sync before you build. To do the sync, look in the upper right hand corner of the IDE and press this button 

How to Run the Code/Application

1. Once you have successfully built the code, you must run it. In android studio, you can do this either on a physical device connected by USB, or on an emulator. We recommend that you use an emulator, as it is easier and more standardized.
2. To add an emulator in android studio, go to Tools > AVD Manager and then add a virtual device. We have run the app on a **Pixel 2 API 29/Pixel 3 API 29**, and running the most recent version of Q (Android 10.0). Once you have added a virtual device and built the code, you can select that device and then click the green play button to run the app. The emulator will launch and launch the app with it.
 - a. Note: In the rectangle next to the green hammer, you will need to select app before pressing the green play button like so:



- b. Also note that it will take some time to build, start up the emulator, and then install the app.
3. In order to use the app, you must create an account. There should be a button at the main screen asking if you need an account. It will prompt you to enter a username, first name, last name, and phone number. The username, first name, and last name need to be alphanumeric. Furthermore, when using an emulator you must use the phone number 7203201044 (we have specifically set this up for you, there are others throughout the documentation that we have also set up, but numbers not listed here will **not** work). When prompted for a code, enter 123456. For more details, see the acceptance case related to logging in.
 - a. **Important: If you have trouble logging in due to firebase authorization issues, try reinstalling the application. That is, (for Pixel emulator phone)**
 - i. **go to the home page**
 - ii. **click the white circle at the bottom of the phone and drag upwards. You will be presented with this screen**



- iii. **Look for the green app icon with android figure with name S+S Messaging**
- iv. **Hold down the icon and drag it until you see the option to uninstall**
- v. **Drag icon into uninstall trash can. Uninstall the app.**
- vi. **Finally, press the green play button again and the app will be reinstalled to your device**

- b. If you can't get the app running, please contact us and we can get you on track.
4. From here you should be logged in and free to use the interface.

How to Run Unit Tests (Unit tests have changed a lot!)

1. **Important:** In order to run the unit tests, you *must* have an emulator installed. Please see the last section if you have not done this yet.
2. To run the unit tests, look for the folder app/src/androidTest. In this folder, you should see clientBackendUnitTests.
3. Before running the tests, make sure you have built the code.
4. Then, simply right click the **com.cs220.ssmessaging** folder and look for an option with the green play button that says 'Run Tests in...
5. Also, to run an individual set of tests, you can right click that individual folder and select the run test option.

Suggested Acceptance Tests

Note: The following tests make reference to the different screens in the app. We have provided screenshots of these different screens at the end of this section to ensure you are seeing the same thing as us.

Note: The username MUST be alphanumeric characters only!!!!

Create a User:

1. Open the app
2. Click "Need a new account?"
3. **Expected behavior: registration screen is rendered.**
4. Type in a username, e.g. "Bob" or one of your choice
5. Type in your first name
6. Type in your last name
7. Type "6303920518" for a phone number
8. Click register
9. **Expected behavior: loads code screen**
10. Enter "123456" for your code
11. Sign in
12. **Expected behavior: loads main conversations screen**

Acceptance Criteria: User is able to load a registration screen, enter their info, register, and load main conversations screen.

Valid Login:

1. Open the app
2. Type "Henry" for username

3. Type "6303920518" for phone number
4. Click "Login"
5. **Expected Behavior: loads code screen**
6. Enter "123456" for your code
7. Click "Sign In"
8. **Expected Behavior: loads main conversations screen**

Acceptance Criteria: Login is successful and conversations screen load

Invalid Login

1. Open the app
2. Type "Jason" for the username
3. Type "6303920518" for the phone number
4. Click "Login"
5. **Expected Behavior: a notification appears informing the user that the username entered is invalid**
6. Type "Henry" for the username
7. Type "222" for the phone number
8. **Expected Behavior: a notification appears informing the user that the phone number entered is invalid**

Acceptance Criteria: All login attempts are unsuccessful and user is unable to proceed to the conversations screen.

Invalid Code

1. Open the app
2. Type "Henry" for the username
3. Type "6303920518" for the phone number
4. Click "Login"
5. Enter an invalid code e.g. "123" or "abcde"
6. Click "Sign In"
7. **Expected Behavior: The app warns the user to enter a code and does not allow them to proceed to the next screen.**

Acceptance Criteria: User is unable to proceed to the next screen.

Logout

1. Follow the steps for Valid Login
2. Click the three dot icon in the top right corner

3. **Expected Behavior: A settings button appears**
4. Click the “Settings” button
5. **Expected Behavior: Settings screen is rendered**
6. Click the “Logout” button in the top left corner
7. **Expected Behavior: User is returned to the login screen**

Acceptance Criteria: User is able to click the logout button and return to login screen.

Start Conversation:

1. Open the app
2. Follow the steps for valid login where expected behaviors are the same as before, i.e.
 - a. Type “Henry”
 - b. Type “6303920518” for phone number
 - c. Click login
3. Where it says “enter-username”, type “Connor” and click START CONVERSATION
Note that existing conversations will only be visible once the “enter username” field is clicked
4. **Expected behavior: Messaging screen is rendered.**

Acceptance Criteria: User is able to log in, enter the name of an existing user to start a conversation with, and start and render a conversation screen.

Send Message:

1. Open the app
2. Follow the steps to “Start Conversation” as Henry with Connor. Alternatively, you could follow the steps in “Create a User” to make a user, log out, make a second user, and start a conversation with the first (i.e., using the provided phone number for both accounts).
3. Type a message and press send.
4. **Expected behavior: Message is sent and rendered.**

Acceptance Criteria: User is able to log in, start a conversation, send a message, and have it render for them in their conversation screen.

Note: We are aware that it is odd to require pressing the back button to send and render the message, and will fix this in the next iteration.

Receive Message:

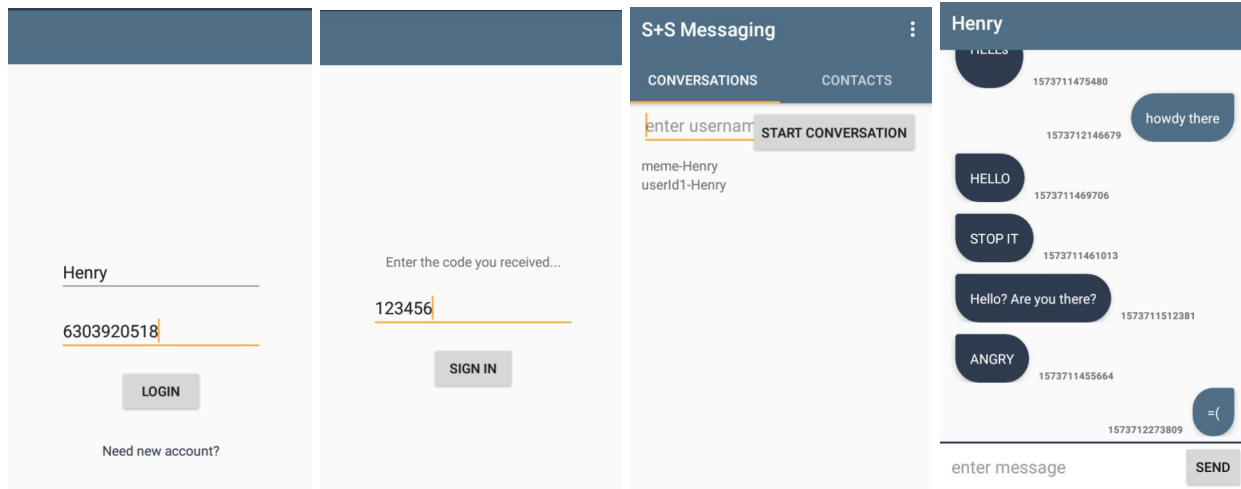
1. Open the app
2. Follow the steps for “Send Message”
3. Follow the steps for “Valid Login” but this time use login as the user that you sent the message to in step 2

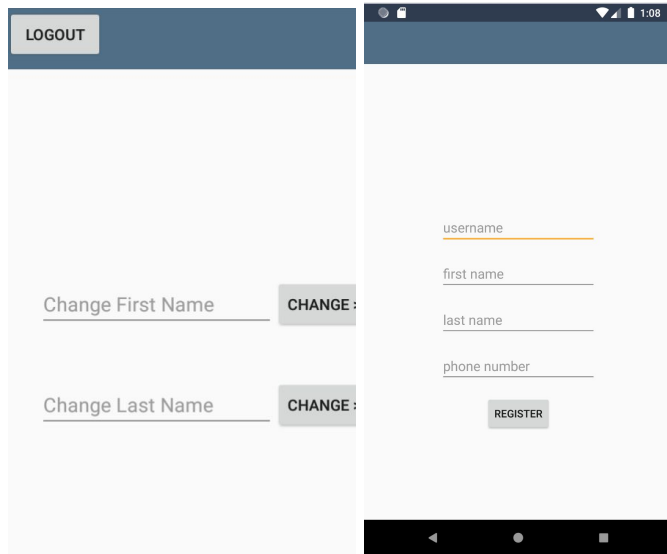
4. Click “enter username”
5. **Expected Behavior: Conversations screen shows a conversation with the names of the two users involved in the conversation from step 2**
6. Click the conversation with names of the two users involved in the conversation from step 2
7. **Expected Behavior: The message(s) sent in part 2 are appear in the conversation on the left hand side of the screen.**

Acceptance Criteria: The conversation and messages created in step 2 are available and visible to the user.

Example Screens

In order of appearance: Login, Code, Conversations, Conversation, Settings, Registration





Use Cases We Implemented

Login/Logout/Register:

New users can create enter their information (first name, last name, username, and phone number) to create an account that will sync with the app's underlying database and register them as a new user.

Phone Authentication:

While the emulators run through Android Studios do not actually text the authentication messages through the simulated phone, the

Starting Conversations and Viewing them:

When a user enters another user's name, a conversation is created and both users can see the newly created conversation in the conversations list. Then, all the user has to do is press the conversation to view it.

Sending and Receiving Text Messages:

Once a message is sent, the message is reflected in both the firestore database and users' conversation interface. Remember that for now, in order to view changes, you will have to click the back button and reenter the conversation.

Contact List:

Contacts have been implemented locally in both the back and front ends of our code, though we plan on integrating the two with the server in the next iteration.

Settings:

Allows users to change their first name, last name, or logout. Logout feature is synced with the underlying database, but changes to first and last names only link to the UI for now.

Encryption:

Implemented on the backend; will be integrated end to end with the database in the future iteration.

Cloud Conversation Storage:

Allows a user to log in on one device, start a conversation, send a message, and then log in to a different device and see that conversation history on the new device.

Database Management:

Store information about users and conversations on the database. Send messages to be stored on the database. Listen on client side to messages that were sent to the database and then pass the received messages on to the frontend.

Parts of Use Cases Moved Iteration 2

We had trouble with working with the asynchronous database component of the app. More specifically, integrating the backend+frontend client with the server proved very difficult. Thus, we wanted to focus more on core messenger features of our app, like sending and receiving text messages, instead of bonus features like Encryption/Decryption + Sending/Receiving Image Messages.

The vast majority of us have little to no experience working with Android Studios, mobile programming, or Kotlin; we've found it rather challenging to learn the new software, hardware and language simultaneously - please forgive our inexperience!

End to End Encryption/Decryption

Sending and Receiving Image Messages

End to End Contact Management

Who Did What

User Interface: Miranda, Zené

Backend: Troy, Ryan, Henry

Database Integration/Backend: Donna, Daisy, Gray

- Originally this was supposed to be a server team, but we moved to a serverless architecture. Instead we used Firestore to handle all communication with clients.
- As such, this team ended up working heavily with the backend team to help setup the integration with the Firestore database.

Changes (Unit Test and Code)

Passwords:

Last iteration, you recommended that we implement a password system alongside our user registration. Instead of a password, however, we have decided on using a phone authentication to verify any given user - in essence, your *device* is your password.

Front-End Testing:

As we built this iteration of our app, we realized that 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 our previous unit tests focused on functions that we have decided to implement elsewhere in the backend; thus, we have moved and consolidated several of our tests, and removed a few others. This functionality is thus tested with the backend unit tests as the frontend just calls the backend functions. Furthermore, many of our previous unit tests relied on building mock ups of various dependencies and classes, though 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.

Backend Unit Testing (Major changes)

Note that all 65 backend unit tests pass (and should pass). Also note that because of major changes, the old unit tests from milestone_3a will not run on the code from milestone_3b.

Unfortunately, our backend tests had to greatly change changed for a few major reasons:

- We now run the tests on the emulator (moved the test files to androidTest directory) because objects like files and database API work differently on Android than locally on Unix machines.
- Refactoring code introduced more functions and thus more code to unit test.
- Changing design resulted in objects' and functions' properties changing. This required the tests' inputs and expected values to change greatly.
 - We specifically changed the Conversation class such that it no longer takes two Users in its constructor but two userIds. This change was necessary because otherwise, we would be instantiating the firebase class inside thousands of time (which is not good for database limits)
 - As a result, many backend tests for Device, User, Conversation, and Message have been modified to use the correct parameters and assertion comparisons.
- We made errors in choice of assertion functions, class inputs, or expected values.
 - E.g. we often compared arrays using assertEquals, which is wrong. We need assertEquals.
- Firebase class kept causing our tests to crash because of certain test structures.

- We had to split up the Message unit tests into 4 files: MessageUnitTests, EncryptedMessageUnitTests, ImageMessageUnitTests, and TextMessageUnitTests. After doing this, we no longer had crashes.

As with last time, many functions in User are not able to be unit tested because they completely deal with working with the server/Firebase. Firebase as of now lacks proper support for mocking and thus unit testing. The following functions cannot be unit tested in User class:

- sendTextMsg
- getUserPublicKey
- addPublicKeyToServer
- addSelfToDatabase
- receiveConversation

Specific unit test changes

For future extensibility, we made the design decision to move property checking (e.g. if name is alphanumeric) for the User, Conversation, and Message classes to static helper functions (companion objects in kotlin). We therefore added 15 unit tests for these new helper functions. Note that these functions are simple and do not change the behavior of the existing functions and unit tests from milestone 3a.

- isValidUser
 - isValidName
 - isValidUserId
- isValidConversation
 - isValidConversationId
 - isValidLastTimeSynched
 - Uses isValidUser()
- isValidMessage (For each of EncryptedMessage, TextMessage, and ImageMessage)
 - isValidMessageBody (For each of EncryptedMessage, TextMessage, and ImageMessage)
 - isValidTimestamp (For Message ONLY)
 - isValidMessageType (For EncryptedMessage ONLY)
 - isValidPathToImage (For ImageMessage ONLY)
 - Uses isValidUser()
 - Uses isValidConversationId()

We also added another constructor for User that allows for setting of contacts and conversation list. This only requires one more unit test.

There is a function called mEquals that we added that compares messages. We added unit tests for their implementations.

We specifically changed the Conversation class such that it no longer takes two Users in its constructor but two userIds. This change was necessary because otherwise, we would be instantiating the firebase class inside thousands of time (which is not good for database read and write limits). As a result, many backend tests for Device, User, Conversation, and Message have been modified to use the correct parameters and assertion comparisons.

Changes to the unit tests have also been marked and explained with a comment that begins with "FIX"

Other

N/a