

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-messging>

The branch you should use is **master** or **milestone-4b**.

Note that milestone-4b will always be the same from this point on (so we recommend checking out this branch).

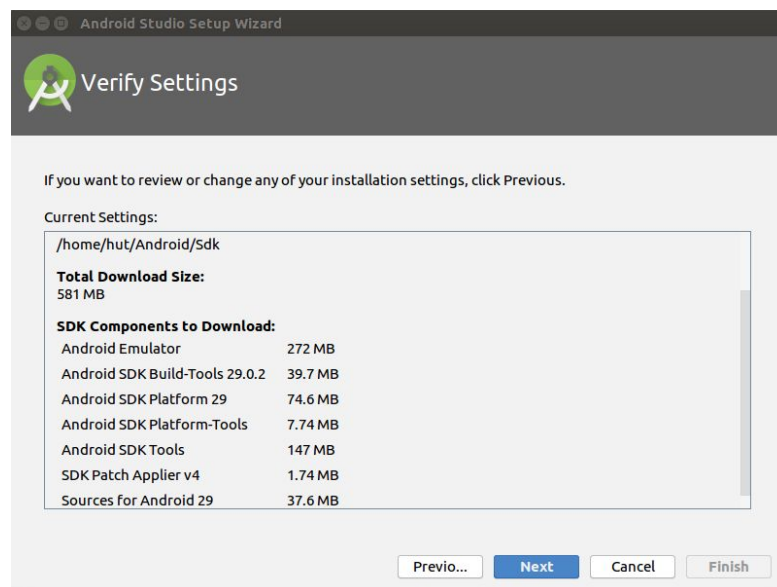
How to Compile

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



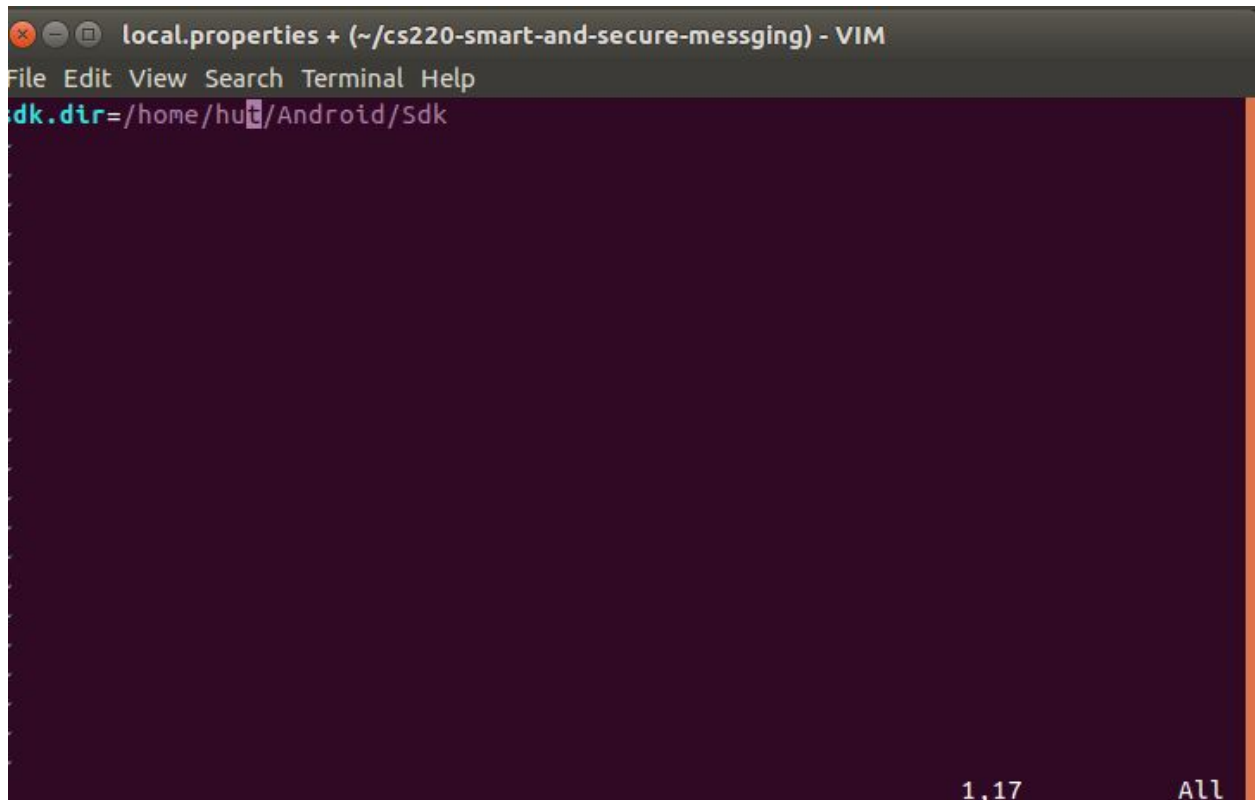
- b. Keep a note of the path given there! It is the path to your Android SDK folder.
- c. Keep clicking next, install, and finish.


- 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:
- Go to the path shown in the image in 3a. In my case it is `/home/hut/Android/Sdk`
 - `cd` into `tools/bin`
 - 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.
- Go to top level directory. That is `cs220-smart-and-secure-messaging`.
 - Create a file called `local.properties`
 - within the file write ONLY:
 - `sdk.dir =` the path you noted for android SDK
 - 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$
```

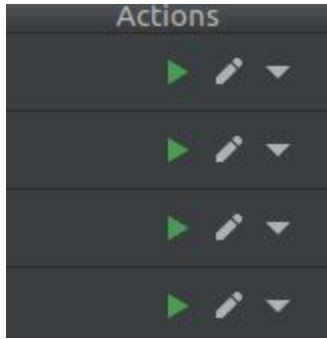



6. Next, open the project in android studio. At the top bar you should see a green hammer, and if you press that the project should build.
 - a. You might need to run a gradle sync before you build. To do the sync, look in the upper right hand corner of the IDE and press this button 

How to Run the Code/Application

STEP 0: If you have already installed an emulator before, please wipe the data on the emulator. Go to AVD (Tools > AVD Manager) like so, and click the rightmost triangle. You should see a “Wipe Data” option. Click it.

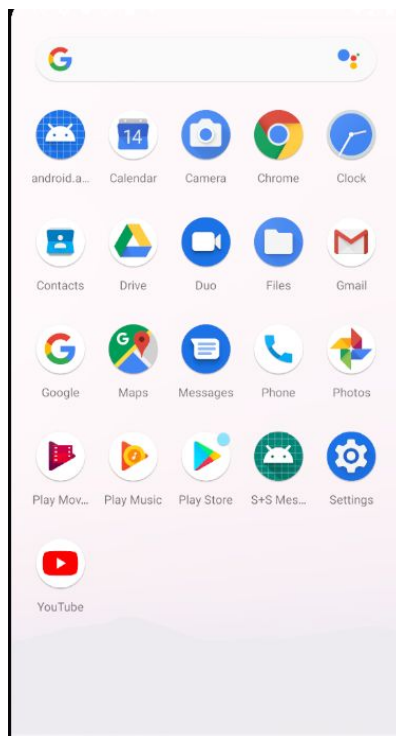
Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 3 API 29		1080 × 2160: 440dpi	29	Android 10.0 (Google Play)	x86	1.6 GB	  
	Nexus 5 API 29		1080 × 1920: xxhdpi	29	Android 10.0 (Google Play)	x86	1.8 GB	  
	Pixel 2 API 29		1080 × 1920: 420dpi	29	Android 10.0 (Google Play)	x86	582 MB	  
	Pixel 2 XL API 29		1440 × 2880: 560dpi	29	Android 10.0 (Google APIs)	x86	1.1 GB	  



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 the following three methods (unfortunately, firebase auth is unstable on emulator):**
 - i. **Wait for a few seconds and try again three times:**
 1. Press the “Sign In” button again with the code “123456” entered
 - ii. **Restart the Application.**

1. Exit the emulator.
 2. Press the green play button again. You will now be shown the login page again.
- iii. **Reinstall the Application through Android Studio.** Note that this may cause problems if you have already signed up as a user. But if this is your first time logging in, it is fine. See [“Key Corruption “Bug” Problem”](#) section for more information.
1. In the toolbar at the top of the studio, press Build -> Clean Project.
 2. Then press Build -> Build Project.
 3. Finally press the green play button to start the emulator and reinstall the app.
- iv. **Reinstall the Application after deleting the application.** Note that this may cause problems if you have already signed up as a user. But if this is your first time logging in, it is fine. See [“Key Corruption “Bug” Problem”](#) for more information.
1. Click the white circle at the bottom of the phone and drag upwards. You will be presented with this screen



2. Look for the green app icon with android figure with name S+S Messaging
3. Hold down the icon and drag it until you see the option to uninstall
4. Drag icon into uninstall trash can. Uninstall the app.

5. 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

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.

1. To run the unit tests, look for the folder `app/src/androidTest/java`. In this folder, you should see a folder named **`com.cs220.ssmessaging.clientBackendUnitTests`**.
2. Before running the tests, make sure you have built the code.
3. Then, simply right click the **`com.cs220.ssmessaging.clientBackendUnitTests`** folder and look for an option with the green play button that says 'Run Tests in...
4. Also, to run an individual set of tests, you can right click that individual folder and select the run test option.

Note: All 84/84 test cases should pass.

Important: Key Corruption “Bug” Problem *READ THIS*

The Problem

This problem is not necessarily a bug in normal use case circumstances (you don't do a major reinstall of the application). But, there is a chance that when you delete and reinstall the application OR do a major change in the code, rebuild, and then (do a major) reinstall of the application, the previously logged in user can no longer see previous conversations' messages.

We did not plan for this to happen, nor did we have a use case for this. We envisioned our app as a one keypair per device app and one app per user. We also viewed that if our app is reinstalled, the same user cannot be used again. Note that there can still be multiple users per app (For example, if you want to be seen as different accounts to different people).

This is not an issue on non-emulators since you won't be rebuilding the code and reinstalling the app all the time.

Why This Problem Happens

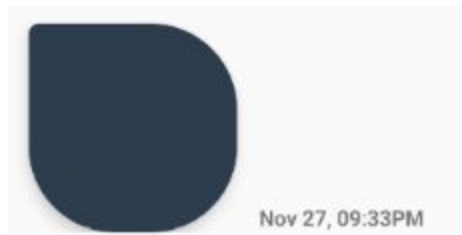
When a user is created, a key pair is generated and stored on the local device. Note that if a key pair is already on the device, the user uses that key pair. The public key of the key pair is also uploaded to the database with the user as the owner.

When you do a major reinstall or delete the app and then reinstall, the key pair files are deleted. This means that the next time you log into the device, new key files will be generated for you completely different from the old key files. However, the keys will not be updated in the database.

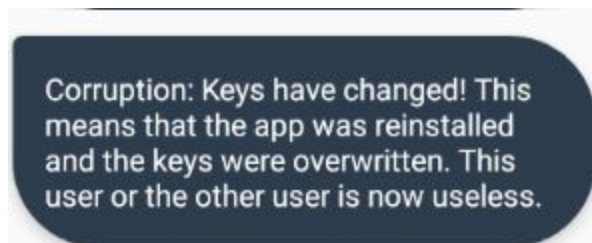
After this key change, whenever you receive a message, the encrypted message assumes that your private key is the OLD private key and NOT the new private key. This is because the message was encrypted with your OLD public key. Thus, the message is not decryptable for you and decryption will throw an exception if not caught (we catch such exceptions).

What You Will See if This Problem Happens

Image Message



Text Message



Mitigations

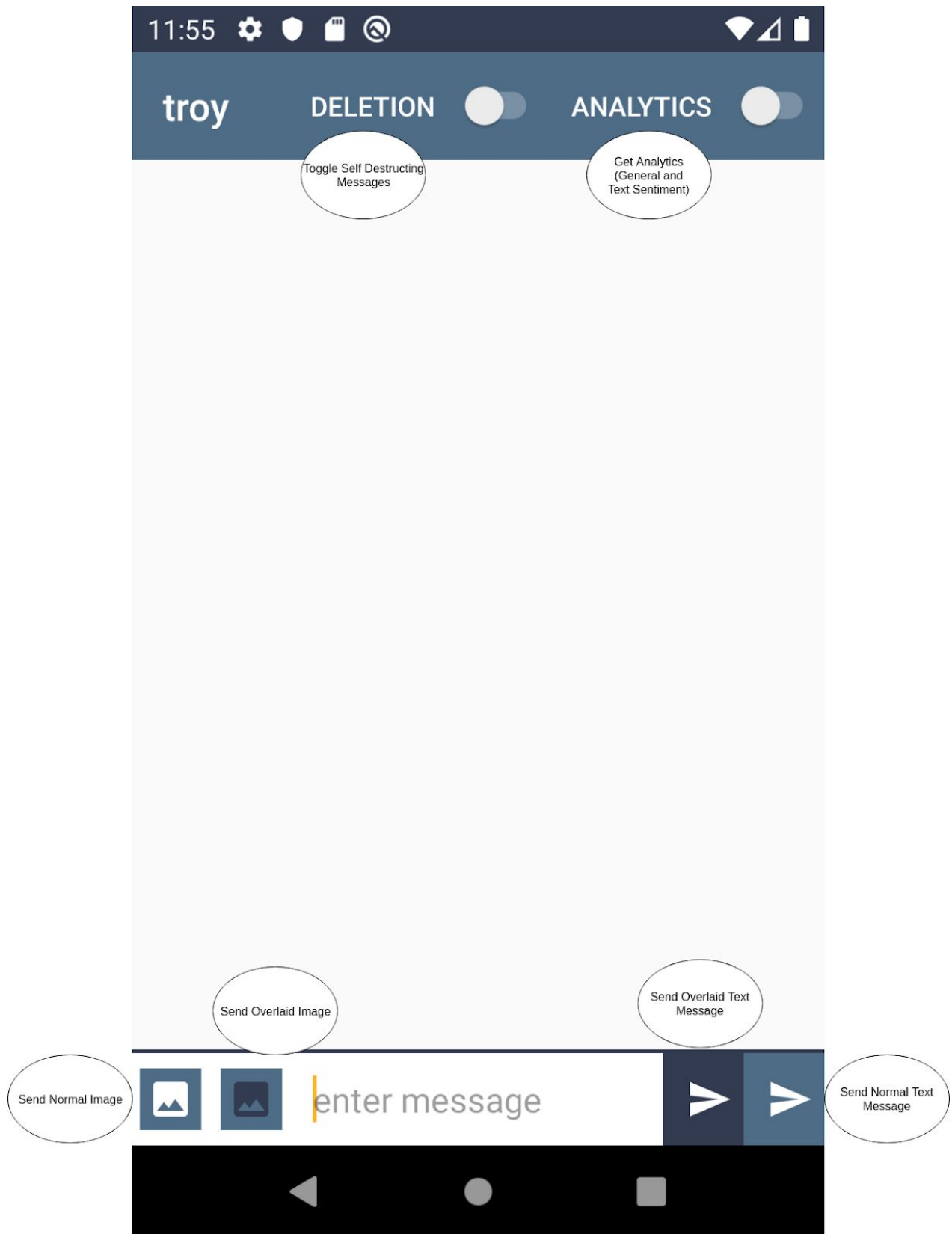
See the “Other” section at the end of the document.

Suggested Acceptance Tests

Notes:

- *The username MUST consist exclusively of alphanumeric characters.*
- *Some functions of our app will take time to process as they connect to foreign servers, please be patient if a function does not appear to be running.*
- *Acceptance tests for features implemented in Milestone 3b can be found in the Milestone 3b README. They are referenced a couple of times in the following tests.*

How to use Message View UI



Send an Image Message:

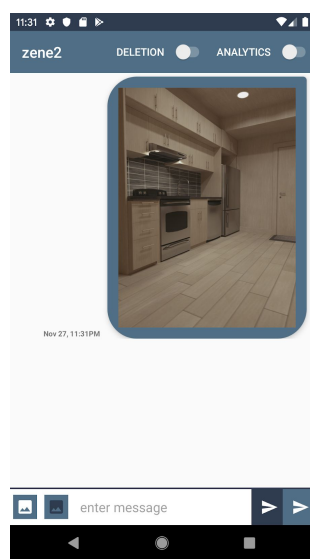
1. Open the app.
2. Follow the steps to “Start Conversation” as some *userA* with *userB*.
From the Milestone 3b acceptance tests.
3. Click on the normal (light) image button on the bottom left of the screen.
4. Navigate to the desired image from the device’s memory.
5. Click on the image you would like to send.
6. **Expected behavior: Image 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.

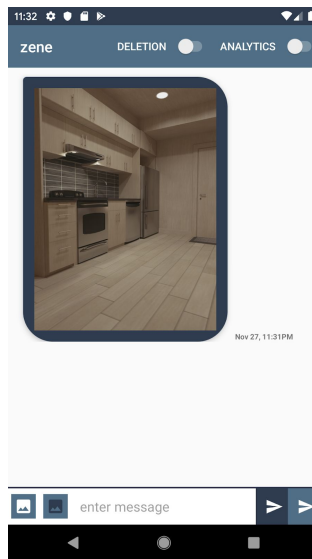
Receive an Image Message:

1. Open the app.
2. Follow the steps for “Send Image Message”.
3. Login as *userB* (the recipient of the image message).
4. Click on the conversation with *userA* (the sender of the image message).
5. **Expected Behavior: The image(s) sent in “Send an Image Message” appear in the conversation on the left hand side of the screen.**

Acceptance Criteria: The images sent from *userA* are available and visible to *userB*.



SEND IMAGE MESSAGE

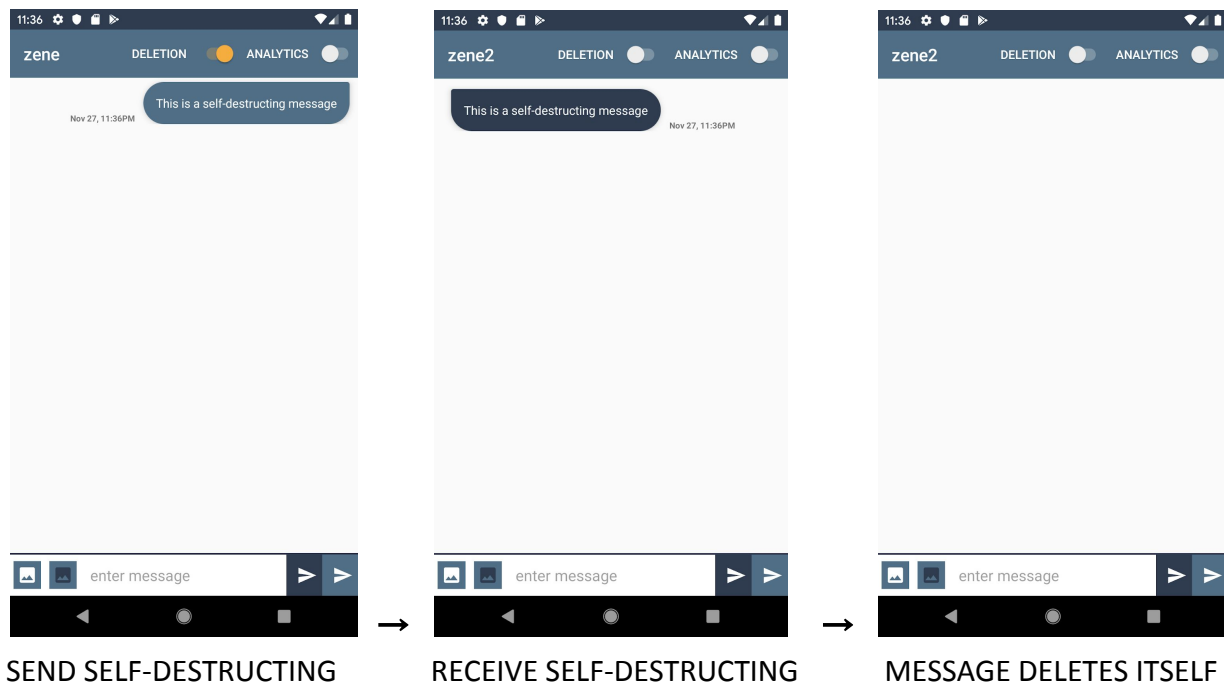


RECEIVE IMAGE MESSAGE

Send a Self-Destructing Text Message:

1. Log in to the app as some *userA*.
2. Select or create a conversation with some *userB*.
3. Turn on the toggle to the right of DELETION (top bar of conversations screen).
4. Type out and send a message using the normal (light blue) send button.
5. Log in to the app as *userB*.
6. Open the conversation with *userA*.
7. **Expected Behavior: the message sent by *userA* using the red button will disappear after five seconds.**

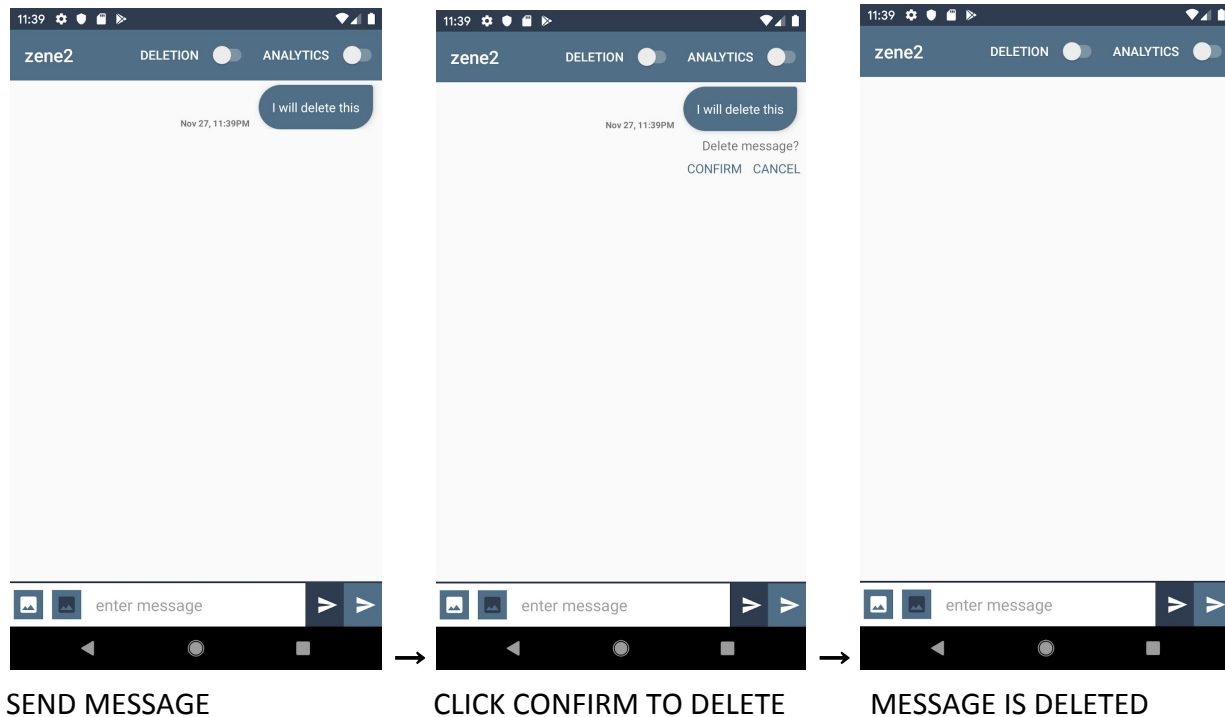
Acceptance Criteria: The self-destructing text message, once opened by the recipient, disappears after a five seconds.



Delete a Sent Message:

1. Log in to the app as some *userA*.
2. Select or create a conversation with *userB*.
3. Type out a message and press the normal (light blue) send button.
4. Click to the left of the sent message bubble, above the timestamp.
5. **Expected Behavior: options appear below the message giving the option to delete message or cancel.**
6. Click confirm.
7. **Expected Behavior: the sent message disappears from the screen.**

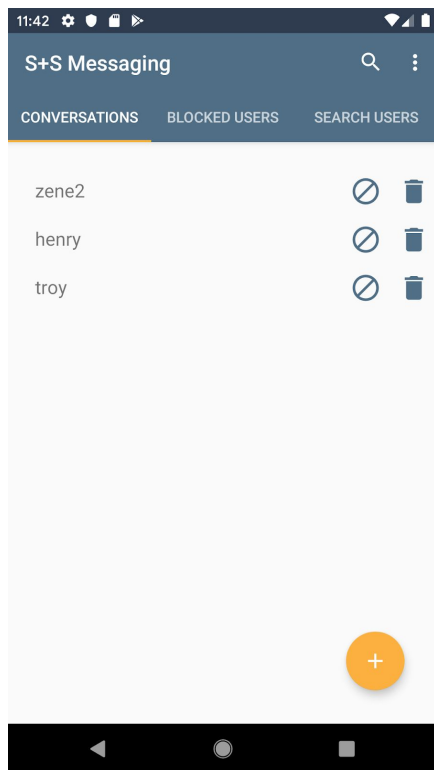
Acceptance Criteria: The sent message is deleted from the database as well as *userA*'s and *userB*'s conversation screens.



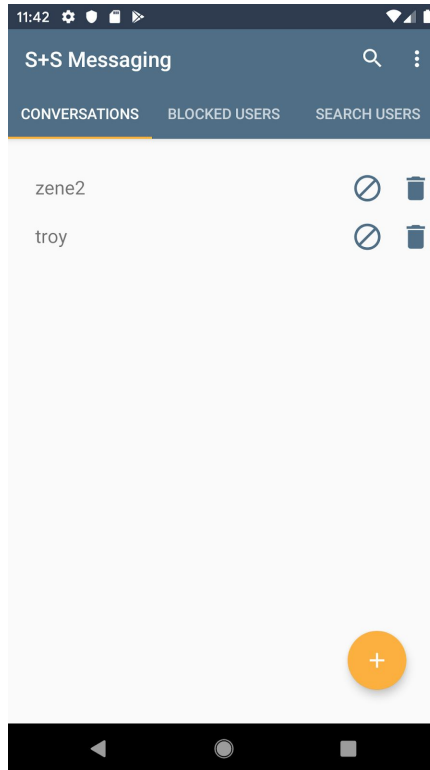
Delete a Conversation:

1. Log in to the app as some *userA*.
2. Select or create a conversation with *userB*.
3. Go back to main page.
4. Find the conversation and press the trash can.
5. **Expected Behavior: Conversation removed from conversations list.**

Acceptance Criteria: The conversation is deleted and no longer appears on your conversation list.



CLICK DELETE ICON

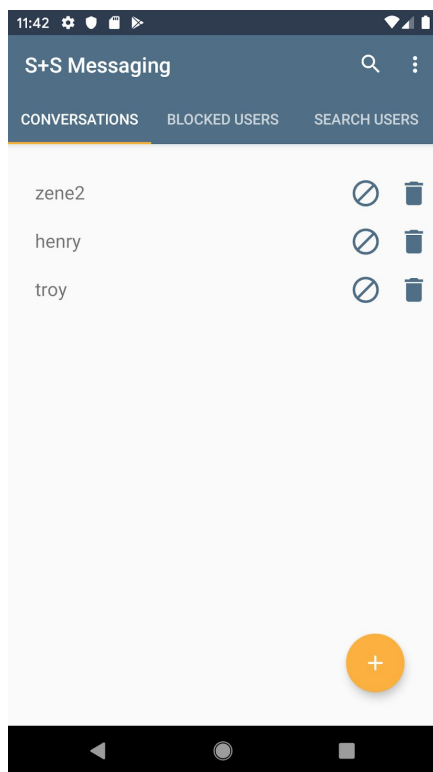


CONVERSATION NO LONGER APPEARS

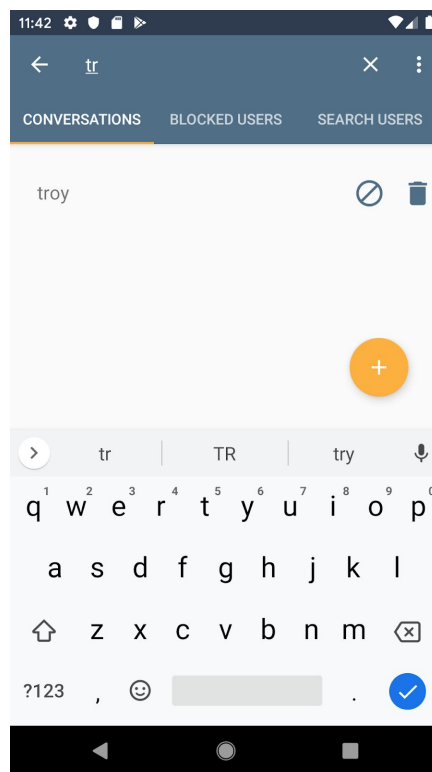
Search for Conversation

1. Log in to the app as some *userA*.
2. Create conversations with other users until *userA* is a member of at least 2 conversations.
3. Click on the magnifying glass at the top of the conversation select screen.
4. Enter a string that is a substring of one conversation name but not the others.
5. **Expected Behavior: all conversations but the one searched for will disappear from the conversation list.**
6. Enter a string that is a substring of no conversation names.
7. **Expected Behavior: all conversations will disappear from the conversation list.**
8. Enter a string that is a substring of all conversation names (if one exists).
9. **Expected Behavior: no conversations will disappear from the conversation list.**

Acceptance Criteria: The conversations that don't contain the string entered into the search bar will be filtered out of the visible conversation list.



CLICK ON SEARCH ICON

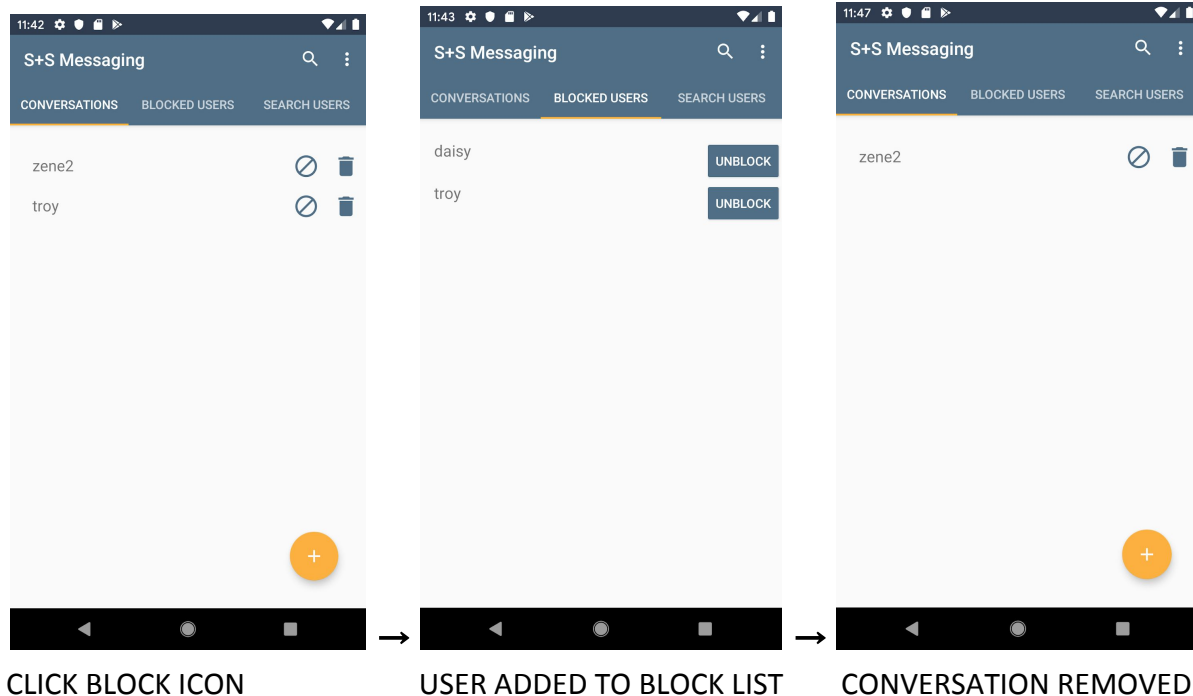


FILTERS LIST BASED ON "TR"

Block a User:

1. Log in to the app as some *userA*.
2. Create a conversation with some *userB*.
3. Return to the main navigation screen, and click the “block” button next to the *userB*.
4. **Expected Behavior: *userB* should disappear from the conversation select screen.**
5. Click on the plus button at the bottom left of the conversation select screen and attempt to create a conversation with *userB*.
6. **Expected Behavior: the app returns an error message.**
7. From the conversation select screen, click on the “blocked users” tab towards the top of the screen.
8. **Expected Behavior: *userB* is listed as a blocked user**

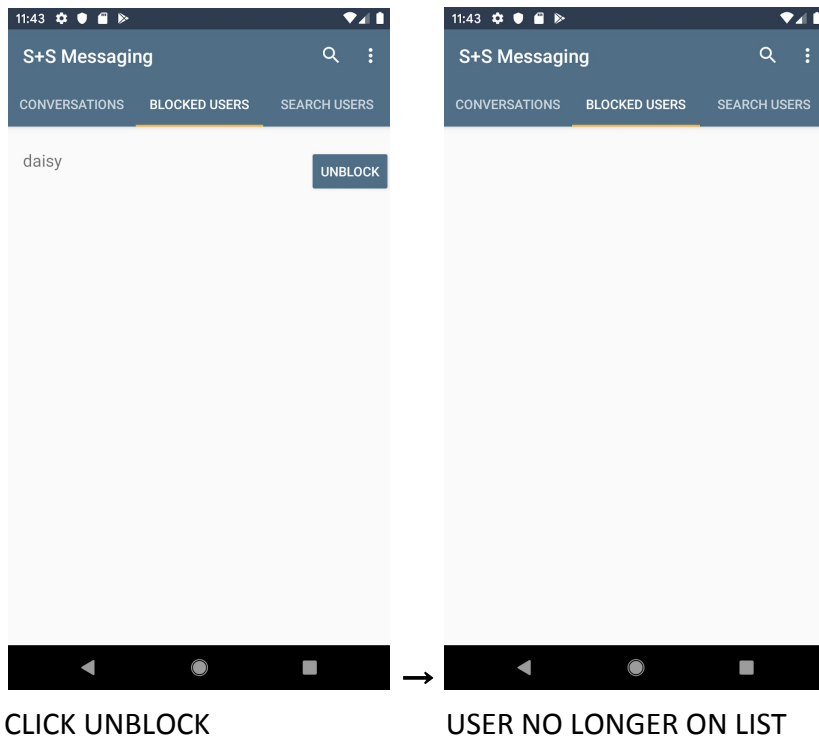
Acceptance Criteria: UserB should be listed under userA’s blocked list and userA should not have any conversations with userB while userB.



Unblock a User

1. Follow the steps of “Block a User” as *userA* for some user *userB*.
2. Navigate to the Blocked Users page of the app.
3. Click the Unblock User button next to *userB*.
4. **Expected Behavior: *userB* should disappear from the blocked users page.**
5. Start a conversation with *userA*.
6. **Expected Behavior: Conversation (with previous messages) is back.**

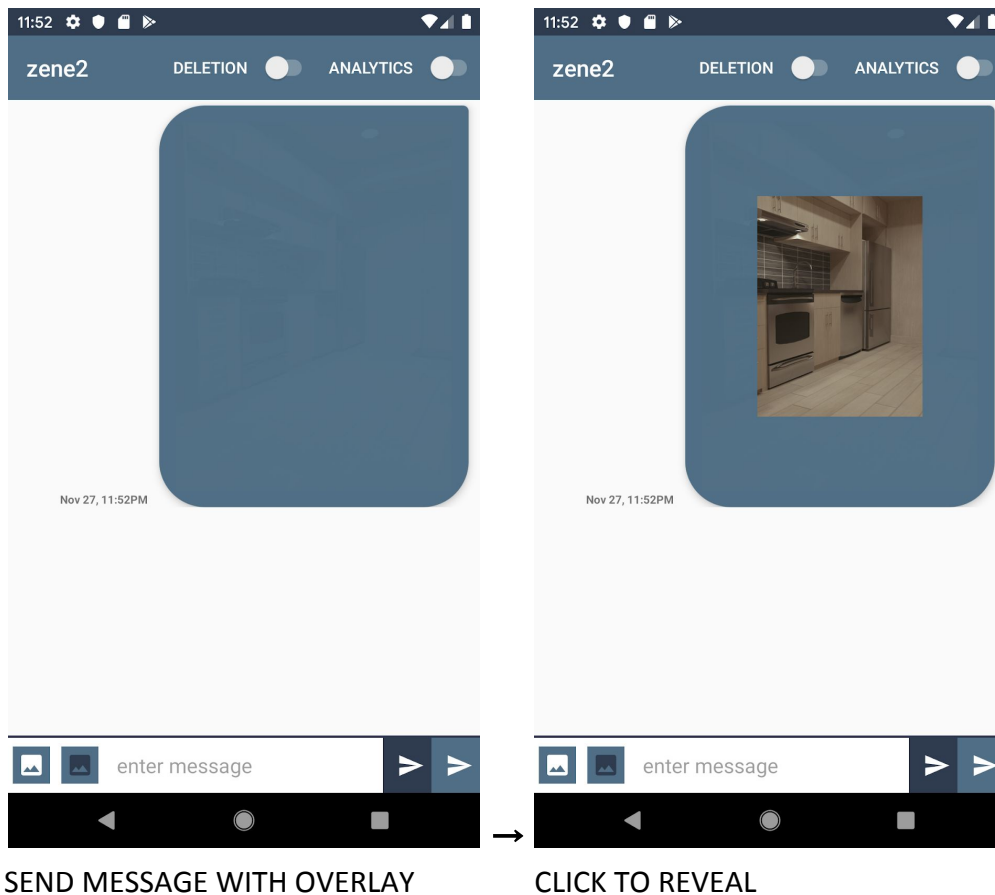
Acceptance Criteria: UserB should now have the same status as userA’s regular contacts and be able to start convo with UserA.



Send an Overlay-Image Message:

1. Open the app
2. Start a conversation as some *userA* with some *userA*.
3. Click on the dark image button on the bottom left of the screen.
4. Navigate to the desired image from the device's memory.
5. Click on the image you would like to send.
6. **Expected behavior: Image is sent and rendered with an overlay.**

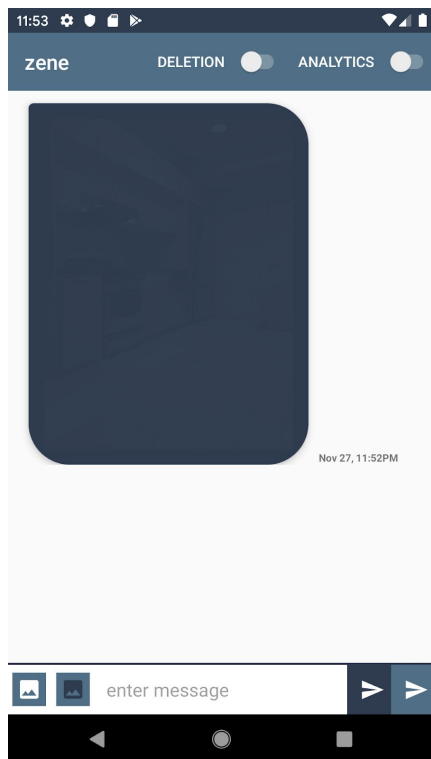
Acceptance Criteria: User is able to log in, start a conversation, send an image with overlay, and have it render for them in their conversation screen.



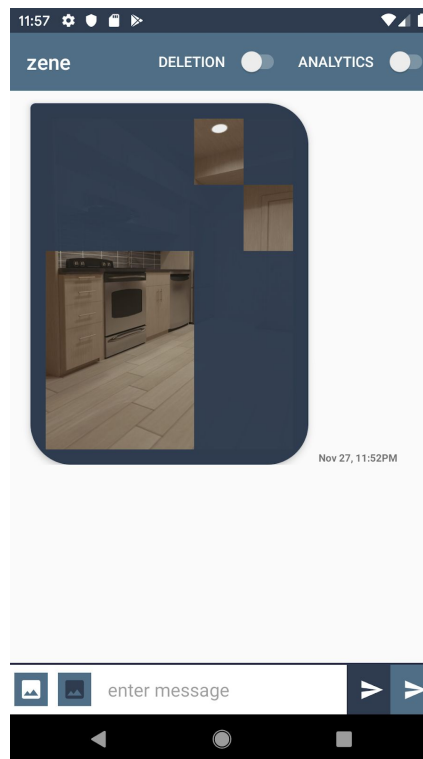
Receive an Overlay-Image Message:

1. Open the app.
2. Follow the steps for “Send an Overlay-Image Message”.
3. Click on various portions of the overlay.
4. **Expected Behavior:** clicked portions of the overlay disappear for a short time, revealing the image beneath it.

Acceptance Criteria: Portions of the image sent are visible for a short time after the corresponding portions of the overlay are clicked on.



RECEIVE MESSAGE WITH OVERLAY

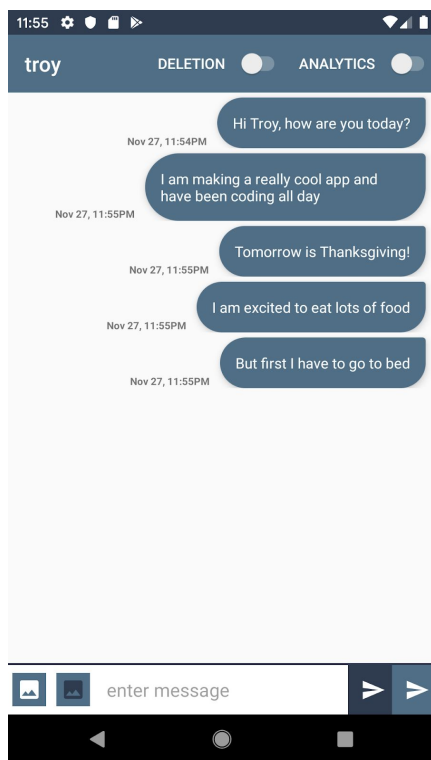


CLICK TO REVEAL

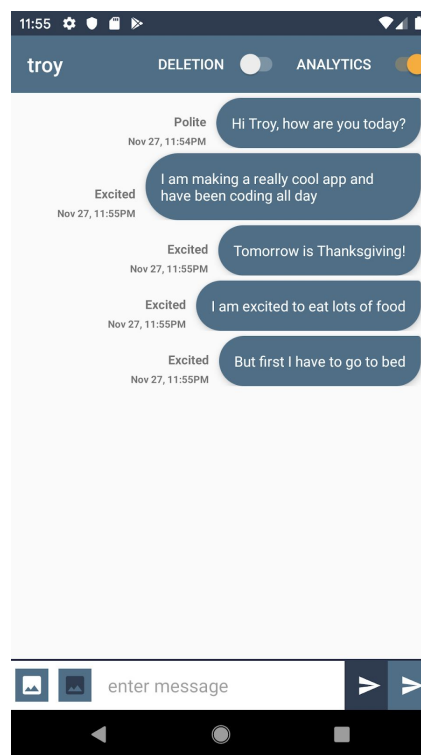
Display and Toggle Tone (aka Get Analytics):

1. Open and sign-in to the app.
2. Navigate to a specific conversation with text messages.
3. Click on the switch in the upper right corner of the screen.
4. **Expected Behavior: Text messages should go between having only timestamps displayed next to them and having both a tone (ex: polite, angry, etc.) and a timestamp displayed next to them (some will have no sentiment).**

Acceptance Criteria: While the switch is on, the words displayed next to text messages should reflect the tone of the user who sent them. While the switch is off, there should only be timestamps next to messages.



SEND MESSAGES



TOGGLE ANALYTICS ON TO DISPLAY SENTIMENTS

Log Out

1. Open the app and sign in.
2. Try to press the back button.
3. **Expected Behavior: Nothing happens. You are not allowed to press the back button to leave the home page.**
4. Click the vertical three dots in the upper right hand corner.
5. Click settings.
6. Press log out.

Acceptance Criteria: You will be logged out and sent back to login screen.

[Cannot be Emulated] Screenshot Blocking

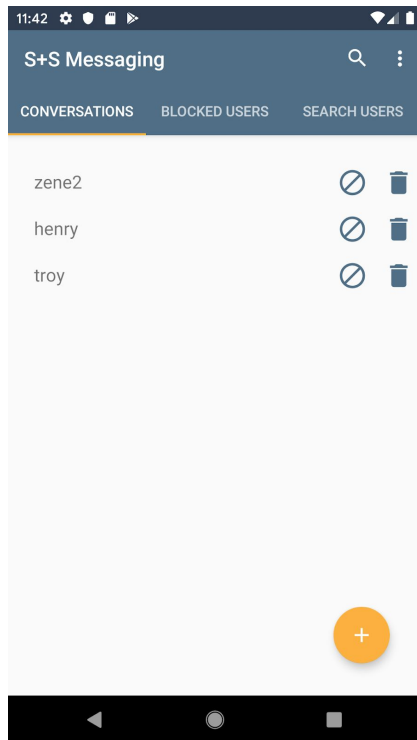
Note: This feature cannot be used in the present testing environment. Screenshot blocking does not work on the Android emulator; the functionality requires the use of a physical Android device. For this reason, we omit this testing procedure from the general suite of acceptance tests. However, we have provided testing guidelines below, which outline how this feature might be tested with the use of a physical device.

1. Open the app on a physical android phone and sign in.
2. Enter into a conversation.
3. Attempt to screenshot the contents of that conversation.
4. **Expected Behavior: No screenshot is taken, and an error message to that effect is displayed.**

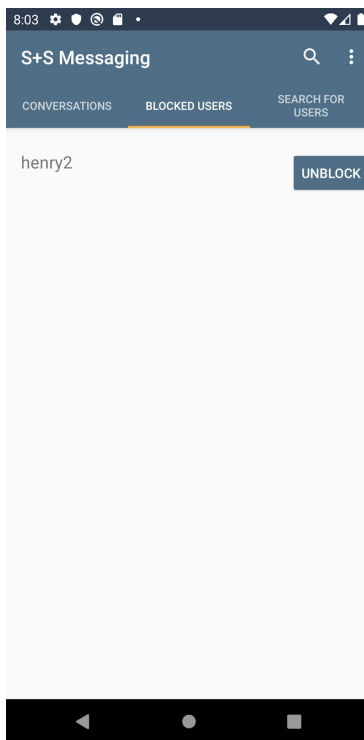
Acceptance Criteria: Screenshots of the contents of a conversation cannot be taken.

Example Screens

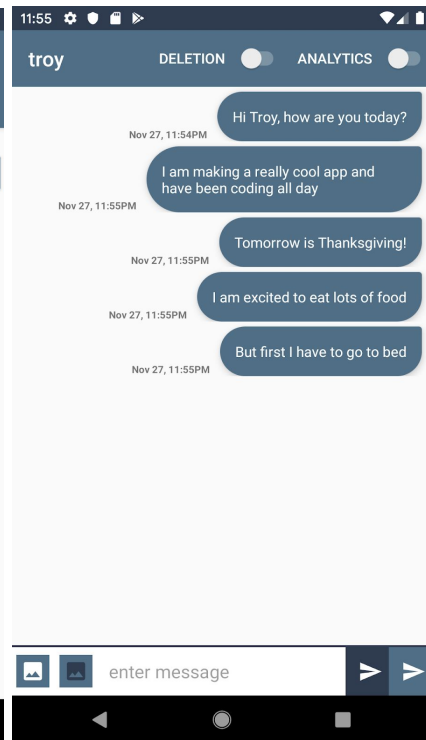
Conversation Viewer



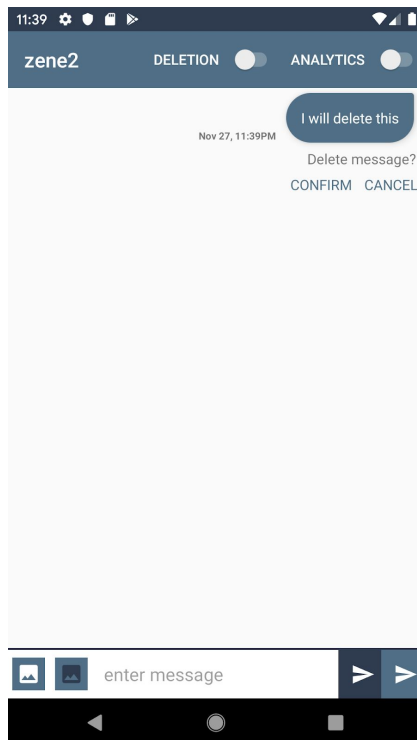
Blocked User List



Chat Screen



Delete Message Popup



Use Cases We Implemented

From the guaranteed use cases in Milestone 4a:

Search for Conversation:

Users can search their list of conversations by the name of the other member of the conversation. This is done via a search bar/magnifying glass at the top of the main navigation screen. Note that this search only filters through a list of the user's conversations in order to find the one they are looking for. It is not intended to search within individual conversations.

Note: users can also start conversations or block users from this page.

Manage Block List:

Users can add and remove other users from a block list. Conversations with blocked users will be deleted and cannot be restarted by either party while one is on the other's blocked list. To add someone to block list, click the circle with slash next to the conversation or in the search for user list.

Delete Sent Message:

Users are given the option to delete messages from the conversation viewing UI. To do this, click to the left of a message and a confirmation will pop up. Once the request to delete is confirmed, the client sends a message to the server destroying the message from the database. Users can only delete messages that they sent.

Delete Conversation:

Users are given the option to delete conversations from the conversation manager UI. Press the trash can next to a conversation to delete it. Once the request to delete is confirmed, the client sends a message to the server destroying the conversation from the database.



End-to-End Encryption:

All text and image messages are encrypted on the sender's device, are encrypted when they are logged by the database, and are only decrypted once they reach the recipient's device.

We can give you access to the database if you request it.

Drag to View Hidden Image Message --> Press to View Hidden Image Message:

This use case has been changed somewhat. It is too difficult to do a draggable overlay and thus, we split each image into 25 boxes. If you press a box, that area of the image is viewable for a few seconds before being covered again by the box. You can press multiple boxes at once.

Note: this use case was mistakenly called “Drag to View Hidden Message” in milestone 4a.

“Smart” Conversation Analytics:

Users have the option to view the tone of text messages and the overall sentiment of the conversation by toggling it in a conversation. To do this, toggle the “Analytics” button:



Screenshot Message Prevention:

When screenshot message prevention is enabled, the recipient is blocked from taking a screenshot on their mobile device.

Note: Screenshot blocking works by setting a flag that interferes with the physical machinery of android phones and thus only works on a physical Android device; this feature cannot be tested using the emulator.

End-to-End Information Management:

Users can now change their information: first name and last name.

From the “extra use cases” in Milestone 4a:

Send Message With Destruction Timer:

Users have an option to send messages with a destruction timer. Once such a message is opened by the recipient, it expires and is expunged from both the device and the server after 5 seconds.

You can do this by toggling the Deletion toggle. Any message you send will be self destructing while the toggle is toggled.



Send Drag-and-Reveal (Text) Message --> Send Click-to-Reveal (Text) Message:

In our previous iterations, this use case was an overlay in which the user could drag their finger over to reveal. Given the constraints of the project, it is now a clickable overlay rather than a draggable one. When a text message is sent with the click-to-reveal feature, the message appears as a blank message bubble for both the sender and the receiver. When the message bubble is clicked on, the text becomes visible for a few seconds before being covered again. Note that unlike with the image overlay, one click reveals the whole text.

Note: a few use cases listed in Milestone 4a were merged, deprecated, or otherwise changed; please see the *Use Case Changes* section for full details.

Who Did What

User Interface: Daisy, Miranda, Zené

Backend: Troy, Ryan, Henry

Watson Smart Analytics: Donna, Gray

When needed, team members helped to integrate their part of their code with the overall app. Moreover, teams helped other teams when necessary.

What We Implemented vs. What We Planned in the Design Document

Original Use Cases

- Login
 - Implemented
- Logout
 - Implemented
- Register
 - Implemented
- Manage Account Information
 - Implemented
- Start Conversation (Between 2 People)
 - Implemented
- View/Leave a Conversation
 - Implemented
- Send/Receive Message Without Image, Message Encrypted
 - Implemented
- Send/Receive Message With Viewable Image, Message Encrypted

- Implemented
- Delete Sent Message
 - Implemented
- Search For Conversation
 - Implemented based on conversationId and userIds
 - Did not implement search criteria such as conversation content (too difficult)
- Manage Contacts
 - Did not implement. Explained above that our conversation list doubles as contact management/list.
- Search For Contact/User
 - In contacts list - Redundant, search for contact same as search for conversation
 - Globally - Implemented
- Manage Block List
 - Implemented
- Look at Analytics
 - Implemented
- Drag to View Part of Hidden Message With Overlay
 - Implemented, but instead of dragging, we implement click and reveal.
- Send 'Drag and Reveal' (Text) Message (Extra use case)
 - Implemented, but instead of dragging, we implement click and reveal.
- Send Message With Destruction Timer (Extra Use Case)
 - Implemented
- Send Message With Picket Overlay (Extra use case)
 - NOT implemented
- Start Group Conversation
 - Not implemented, dropped.
- Screenshot Message
 - Not implemented, dropped.
 - Instead, implemented "prevent screenshot message"

Use Case Changes

- **Search for Conversation**
 - Now we only search based on conversationId and userIds. It is too difficult to implement other searching methods such as content.
- **Send 'Drag and Reveal' (Text) Message -> Send 'Click and Reveal' (Text) Message**
 - We now use click/pressing since it is much easier and less error prone to implement.

- **Drag to View Part of Hidden Message With Overlay -> Click to View Part of Hidden Message With Overlay**
 - We use click for the same reasons as the previous use case change.
- **Contacts List Use Cases Dropped**
 - We noticed that our contacts list use cases were basically the conversations list use cases. We therefore dropped the contacts list in favor of the conversations list.

Changes In Unit Tests

Frontend Unit Testing

Our reason for not unit testing frontend is the same as in Milestones 4a and 3b. We manually tested the frontend with user input.

Backend Unit Testing

All changes have been marked in the code with a comment beginning with “FIX”. Unfortunately, many changes have been made to the unit tests in response to changing design and simple errors in the test code.

- **BlockListTests**
 - General fixes for improper test values.
- **CipherExtensionTests**
 - Fixes for improper test values in response to changing algorithm
 - Also added some test parameters in testEncryptAndDecryptMessageText because of modified encryption method. We now use AES to encrypt messages and RSA to encrypt the AES keys.
- **ConversationTests**
 - General fixes for improper test values.
- **EncryptedMessageUnitTests**
 - New fields required changed constructor values and more checks within test cases.
- **UserUnitTests**
 - General fixes for improper checks and test values in response to changing code.

Other

getAnalytics

Last milestone, you recommended that we test our getAnalytics function, though we have refrained from doing so because getAnalytics returns the sentiments of various messages by connecting to an outside server (called Watson). It is too difficult to test this as there is

no easy way to mock the Watson server API. Another reason why we cannot test the `getAnalytics` and related call back functions because our Watson account has a limited number of times it can be accessed before we must pay to implement it. Thus, we have decided to refrain from testing this function from the backend, choosing instead to test both `getAnalytics`' functionality and implementation simultaneously through front-end acceptance tests. While we understand the risks of not testing this function on the backend, we have conserved our calls to Watson and the greater front-end testing has indicated that `getAnalytics` functions as expected anyway.

Extra Small Tasks

- As discussed above, if Android Studio does a major reinstall of the application (which may happen), the keys for the user are deleted by the system and new random keys are generated. This will render the previously logged in user useless since the keys are permanently unsynced with the server and conversations. The previously logged in user will always see corrupted messages and images. The user has to be deleted or not used anymore.

We did not plan for this to happen, nor did we have a use case for this. We envisioned our app as a one keypair per device app and one app per user. We also viewed that if our app is reinstalled, the same user cannot be used again.

- Note that there can still be multiple users per app. For example, if you want to be seen as different accounts to different people.
- We will make a small change in the code to either:
 - Stop people who have logged in on one device from logging in on another device.
 - Delete the user if the app is reinstalled.
 - Update the user's keys in the database. This makes the user viable for future messages and conversations, but previous conversations and messages are null.
- Bug and stability fixes if any arise during system testing.
- Refactoring of the code to make it more readable.
- Making the UI more readable.