System Test Documentation

Sentiscape

CA400

Jason Boylan: 18342986
Kelan Smyth: 18342973

Dr. Hyowon Lee

Date Completed: 24/04/2022

Abstract

Thorough and quality testing is an aspect of software development that is vital in ensuring that the software works as intended and is important in avoiding and correcting any bugs or errors encountered. There are a number of different testing methods that can be used to

ensure different aspects of the software are up to standard and many of them have been implemented throughout the course of the development process.

Overview

Throughout the project, we did our best to test our project and validate that it worked as intended. Thanks to our use of Android Studio's Gradle build tool, we were able to view how our program was compiled. Having Gradle available to automate the build process helped make sure that every configuration was error-free and it quickly installed the application onto our mobile devices, making it easy to implement ad hoc testing.

We ensured that user testing was completed thoroughly by asking a broad range of questions to a large number of groups. The results of the user testing were then analysed.

Unit testing was conducted to test how the application worked between screens and on smaller functions that did not require the application to run.

This variety of testing methods was required to ensure that we had developed a robust and reliable application.

Ad hoc Testing

Ad hoc testing was routinely undertaken throughout the software development process. While working on the application, we were provided with ample opportunity to attempt to break the application's functionality and this allowed us to further understand where any issues in our code may be located. We incorporated this style of testing across various aspects of the application.

Login System

Sentiscape's login system was a perfect place to conduct ad hoc testing. We tried various cases where we believed that users shouldn't be able to successfully log in. One example of this is when the password isn't entered correctly, or if it's not even entered at all. Before doing this, we had various conditional statements ready with messages to display to the user if they hadn't correctly entered their details. Thanks to the ad hoc testing we uncovered a couple of cases that we might have otherwise missed, for example entering a 1 character password was something that was originally possible as we hadn't considered a minimum character limit for passwords.

Dark Mode

Another example of ad-hoc testing being conducted was when dark mode was implemented. The application was tested in various states to make sure that the light/dark mode setting stayed consistent between screens, and on sign-in/sign out.

Online Status

The online status functionality would be shown as “online” when users were logged into the app. When users log out, the status would be displayed as “offline”. Ad hoc testing was suitable here as the simplest approach to testing this was to log out as a user to set their status to “offline” and then log in as a different user to check that the status displayed “offline” as intended.

Account Deletion

When implementing a “Delete Account” button, we ran into some issues that were resolved with the help of ad hoc testing. The first attempt at implementing this feature comprised of removing the data from the real-time database, finishing the current activity and starting the Login Activity. This removed the user from the database, meaning that any login attempt using those credentials would fail.

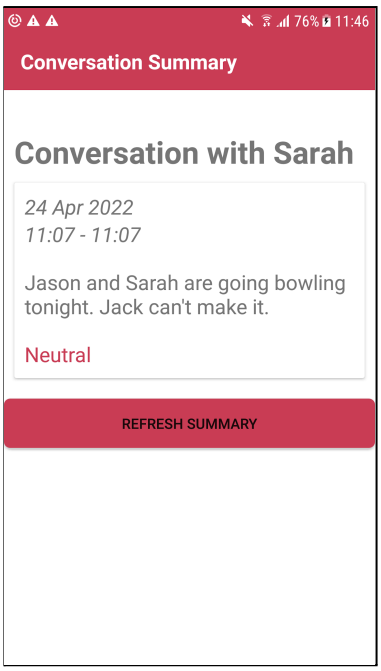
We then tried ad hoc testing by creating accounts, deleting them and trying to log back in. This all worked as intended. One issue was discovered, though. We logged in as another user to check that the deleted user wouldn’t display on the Chat Fragment. However, deleted users were still displayed here even though their accounts were deleted. This was because we hadn’t removed the users’ information from the Firestore. This was an issue we may not have noticed if it weren’t for ad hoc testing.

Messaging

When testing the messaging aspect, we decided the best way to do this was to log in as different users and message each other back and forth to make sure things worked smoothly. A shortcoming of this though was that there was only ever a maximum number of two users logged in at any given time. This was also a shortcoming of the user testing as only one pair of users would use the application at the same time.

Text Summarisation

When testing the Summarisation feature, we would send messages back and forth and then analyse the resulting text summarisation. Here’s an example of this:



Then a POST request would be sent to the server containing the conversation and the Python script would send a GET request and summarise it. After exchanging the summary with the application the data would be displayed and we could analyse the details.

This allowed us to test that only messages from the correct date would be summarised, that the messages between users would show up in the right location and that the summaries and sentiment analysis aligned.

Sentiment Analysis

When testing the sentiment analysis model an activity was created that users could input messages into and the model would return a sentiment value. This page was used to test different aspects of the sentiment analysis features implemented in the application such as the reactive chat bubbles and the reactive chat background.

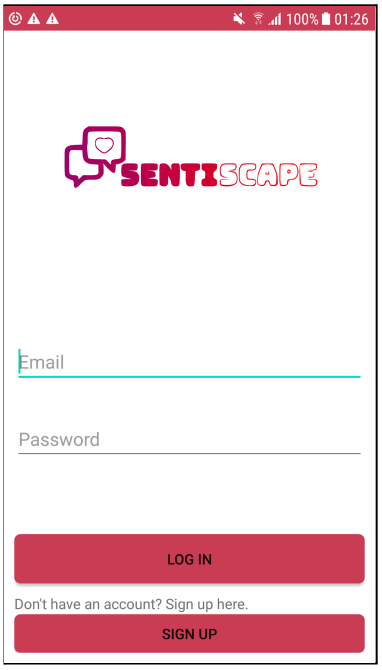
A POST request would be sent through the Ngrok tunnel to the Stanford CoreNLP server to return sentiment results, this allowed us to assess the adequacy of the detected sentiment from a users perspective and assess the speed at which the application could return a result to the user and determine if the model was suitable to use for an instant messaging application.

There was also testing in the Specific Chat Activity to ensure users could send and receive messages with sentiment detection enabled, this was done by having users send messages back and forth and confirming that the detected sentiment actively changed the background of the chat bubbles for the sender and receiver as well as changing the background for both users.

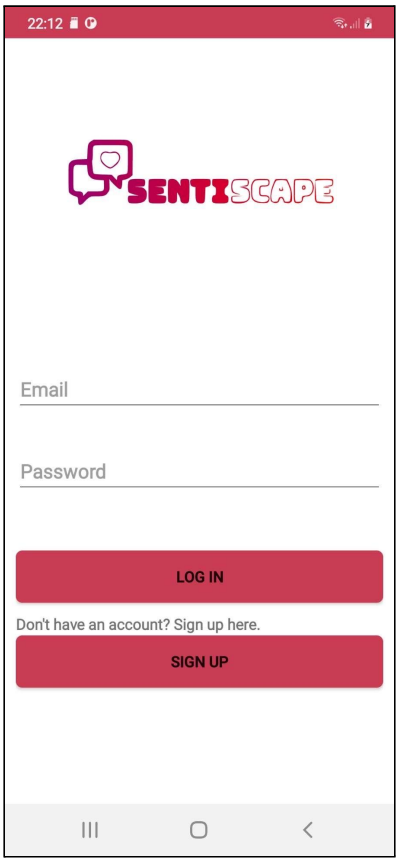
Hardware Testing

Sentiscape supports a large number of Android devices, including legacy systems. We both had access to *Samsung Galaxy S6* smartphones to test the application, but the issue with this was that the screen size was the same for both devices (5.1 inches). In order to test that the display worked on different sized screens, we attained access to a *Samsung Galaxy A40* (screen size: 5.9 inches). By using these devices, we had access to different size screens as well as older and newer devices. The *Samsung Galaxy S6* was released in 2015 and the *Samsung A40* was released in 2019.

Here is how the Log In Activity displays on each device:



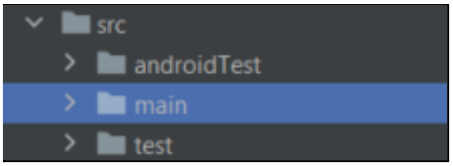
Samsung Galaxy S6



Samsung Galaxy A40

Testing the Code

Android Studio provides two separate directories for testing an application.



The “test” directory is for unit tests that can be conducted in isolation without needing to run the application. The “androidTest” directory is for instrumented tests. These tests must run the application in order to be conducted.

Unit Tests

Unit tests were conducted on the Python summarisation script using PyUnit. These tests were performed on some of the different methods contained in `summarisation.py`.

An example of this is the `test_sentiment_analysis()` which asserts if certain values result in sentiment values as they should.

```
def test_sentiment_analysis(self):
    self.assertEqual(summarisation.sentiment_analysis('1.6'), "Neutral")
    self.assertEqual(summarisation.sentiment_analysis('2.9'), "Positive")
    self.assertEqual(summarisation.sentiment_analysis('1.26'), "Negative")
```

Another example tests that the summary is returned as a string and not as a list with a nested dictionary.

```
def test_summary(self):
    test_conversation = "Amanda: I baked cookies. Do you want some? Jerry: Sure! Amanda: I'll bring you tomorrow :-)"
    self.assertEqual(type(summarisation.get_summary(test_conversation)), str)
```

Instrumented Tests

Various instrumented tests were conducted on the application. Tests were created using JUnit and Espresso. JUnit provided most of the functionality needed but Espresso was used when performing actions on each activity (such as entering text in an EditText field).

For each class of tests, the activity was instantiated before the test is run.

```
@Rule
public ActivityScenarioRule<SignUpActivity> activityScenarioRule = new ActivityScenarioRule<>(SignUpActivity.class);

private ActivityScenario<SignUpActivity> signUpActivity = null;

Instrumentation.ActivityMonitor monitor = InstrumentationRegistry.getInstrumentation().addMonitor(OnboardingActivity.class.getName(), result: null, block: false);
```

```
@Before
public void newActivity() { signUpActivity = activityScenarioRule.getScenario(); }
```

One primary example that featured a lot was a test to make sure that all the elements exist in the activity. Here's an example of this in the SignUpActivity:

```
@Test
public void testSignUp() {
    assertNotNull(signUpActivity);

    signUpActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.usernameSignUp)));
    signUpActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.emailSignUp)));
    signUpActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.passwordSignUp)));
    signUpActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.signUp)));
}
```

Here, the test runs the current activity and asserts that the elements exist.

The second type of test was one to assert that one activity successfully moves to a new activity. Again using the SignUpActivity as an example, here is a test that checks if the "Sign Up" button will bring the user to the OnboardingActivity once the correct credentials have been entered:

```
@Test
public void testSignUpButton() {
    signUpActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.signUp)));

    onView(withId(R.id.usernameSignUp)).perform(clearText(), typeText( stringToBeTyped: "tester6"), ViewActions.closeSoftKeyboard());
    pauseTestFor( milliseconds: 500);
    onView(withId(R.id.emailSignUp)).perform(clearText(), typeText( stringToBeTyped: "tester6@gmail.com"), ViewActions.closeSoftKeyboard());
    pauseTestFor( milliseconds: 500);
    onView(withId(R.id.passwordSignUp)).perform(clearText(), typeText( stringToBeTyped: "tester6"), ViewActions.closeSoftKeyboard());

    onView(withId(R.id.signUp)).perform(click());

    Activity onboardingActivity = InstrumentationRegistry.getInstrumentation().waitForMonitorWithTimeout(monitor,  timeOut: 5000);
    assertNotNull(onboardingActivity);
}
```

Tests were also sometimes run if there were occurrences where a button wouldn't always bring users to the next screen. Here in the LogInActivity, if no credentials were entered, the activity shouldn't advance to the MainActivity, and so asserts null:

```
@Test
public void testLogInButtonNotEntered() {
    logInActivity.onActivity(activity -> assertNotNull(activity.findViewById(R.id.logIn)));

    onView(withId(R.id.logIn)).perform(click());

    Activity mainActivity = InstrumentationRegistry.getInstrumentation().waitForMonitorWithTimeout(monitor,  timeOut: 5000);
    assertNull(mainActivity);

}
```

Due to time constraints, it was not possible to thoroughly test the entire application, but an effort was made to try and cover as much as possible.

Logcat

Logcat is a tool in Android Studio that allows users to view the logs of the running application. Logs were written throughout the development of the application to test if certain variables returned what they were expected to.

```
JSONArray summaryArray = tokenList.getJSONArray( name: "summary");
String jsonSummary = summaryArray.toString();
Log.i( tag: "JSON SUMMARY", jsonSummary );
updateSummary(jsonSummary);
```

User Testing

In order to test the usability of the system, we designed and conducted user testing conforming to the ethics guidelines (approved). We recruited 5 pairs (each pair is 2 participants), undertaking a series of pre-defined text messaging tasks (see below) taking an average of 10 minutes each, using our prototype app installed on 2 Andriod phones in a room with the 2 project members. Each member sat beside the participant and guided, observed and chatted while the user completed the tasks. After all the tasks were completed, they filled in a post-task questionnaire. Findings were analysed and a number of insights for future refinements were suggested.

Questions

To do this, we filled in an ethics approval form where we explained our reasoning and included the survey questions. The survey was created using Google Forms and consisted of the following questions:

How would you rate the usability of the mobile application? *

1 2 3 4 5

Did you think it was easy to navigate the application and achieve what you wanted to do? *

- Yes
- No

Did you think it was easy to send and receive messages? *

- Yes
- No

Did you like the Conversation Summary feature? *

- Yes
- No

How accurate did you think the conversation summaries were? *

1 2 3 4 5

Did you like the Affection feature? *

- Yes
- No

How accurate did you think the Affection feature was at reading your emotions? *

1 2 3 4 5

Would you continue to use this application if it was made available permanently? *

- Yes
- No

Did you think the group chat feature worked as intended (Group chat testers only)?

- Yes
- No

This brief series of questions was aimed at collecting answers for the most high-level questions. The main questions were designed to find out if users found the app to function at a high enough standard and to discover if the Summarisation and Affection features were worthwhile additions to a messaging application.

At first, the questionnaire was designed with group chats in mind. This meant there was going to be a group of four to test out the group chat functionality and they would answer an extra question. In place of this, two extra pairs were included.

The questionnaire was sent to five different groups. Each group consisted of two users. In order to participate in this user testing, participants were given Android devices and a test username and email address. They were also given a topic of conversation to message each other about. Each topic could be broken down into different emotions (positive, neutral and negative). This gave us the opportunity to see if their conversation would return an expected sentiment.

These were the topics that were provided to each group:

- Group 1: Discuss what you're looking forward to in the summer (Positive).
- Group 2: Discuss what you did this week (Neutral).
- Group 3: Discuss your least favourite television shows (Negative).
- Group 4: No topic (to see what a natural conversation would be like).
- Group 5: No topic (same reason as group 4).

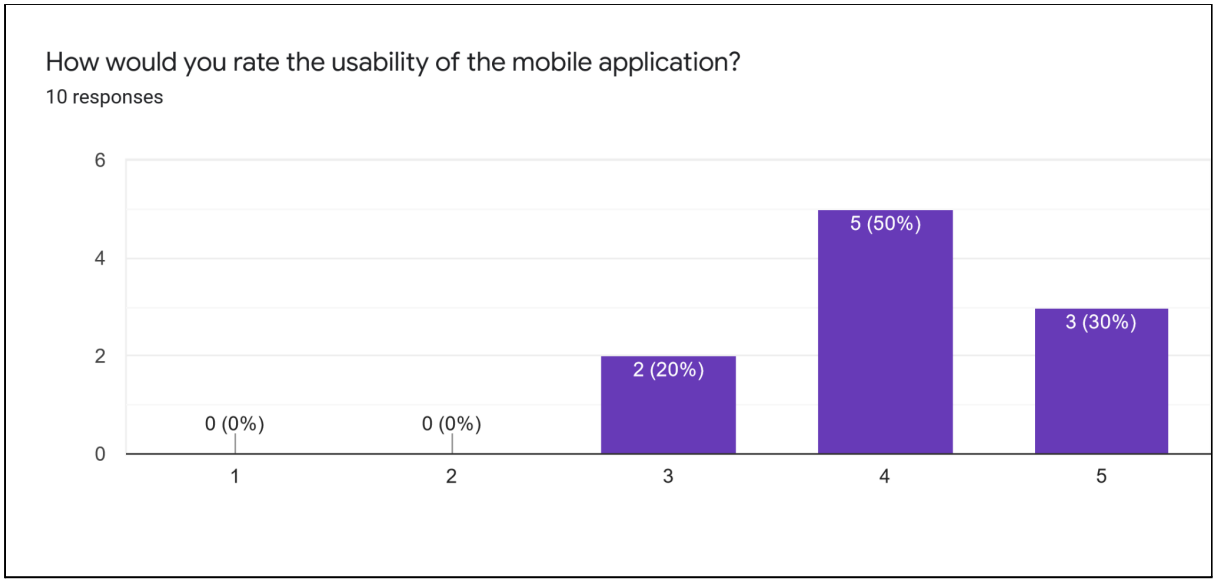
They were to test the application for fifteen minutes. This allowed them time to message each other, to see the sentiment analysis functionality and at the end of their conversation, they could see a summary of their conversation. Providing users with fifteen minutes to test the application allowed them to have a more comprehensive understanding of the application. At the end of the testing period, they were sent the link to the questionnaire.

All five of the groups consisted of pairs. In these pairs, users could message each other back and forth and could also explore the other aspects of the application individually.

Answers

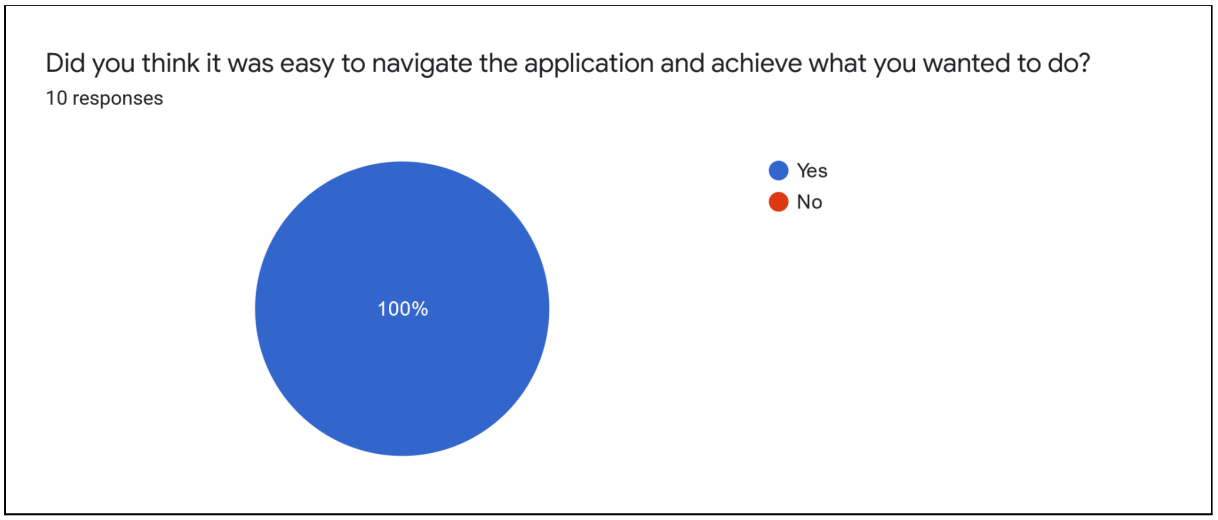
Once the participants had finished testing the application and filled in the questionnaire, we were able to view the results and analyse them. Here are the results of each question:

How would you rate the usability of the mobile application?



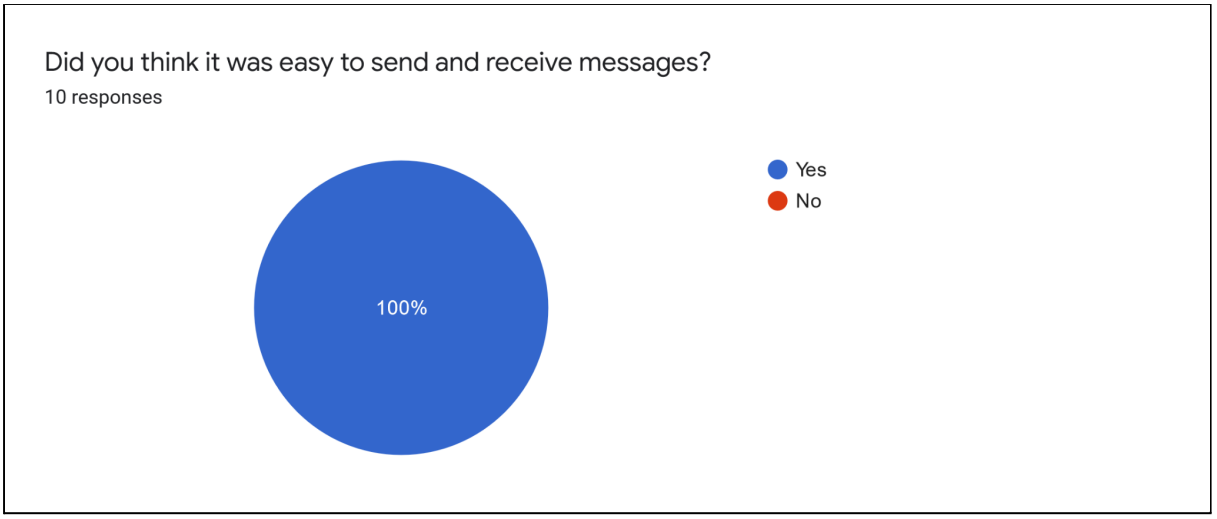
As seen above, there was an overall positive reaction to the application's usability. Of the 10 responses recorded, there was an average score of 4.1 which was a strong result.

Did you think it was easy to navigate the application and achieve what you wanted to do?



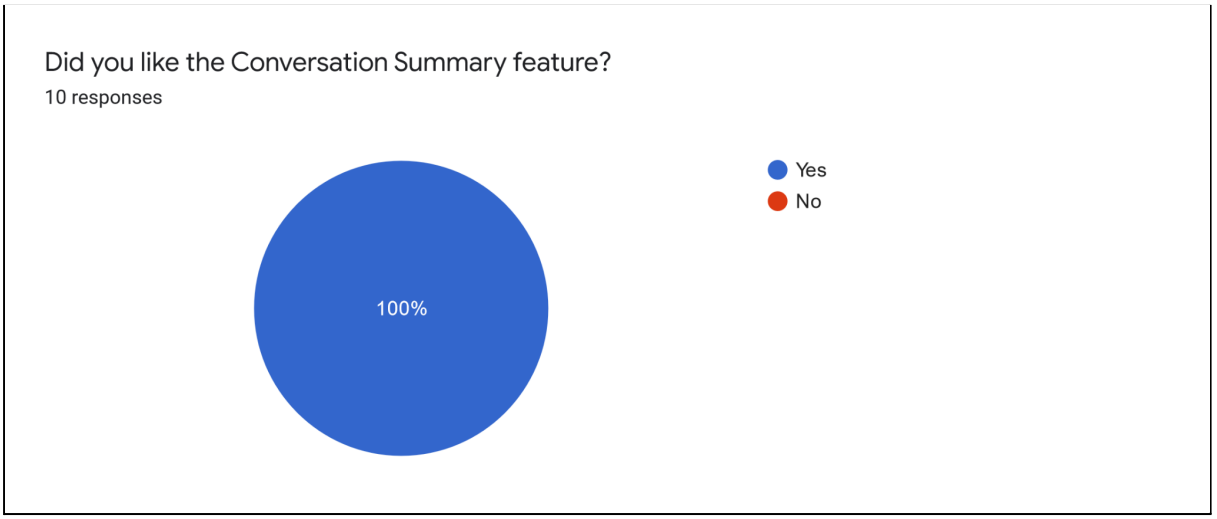
For this question, 100% of the responses said "yes" which was an overwhelmingly positive response. This indicated that the application was intuitive to use and that the onboarding was of useful assistance when signing up.

Did you think it was easy to send and receive messages?



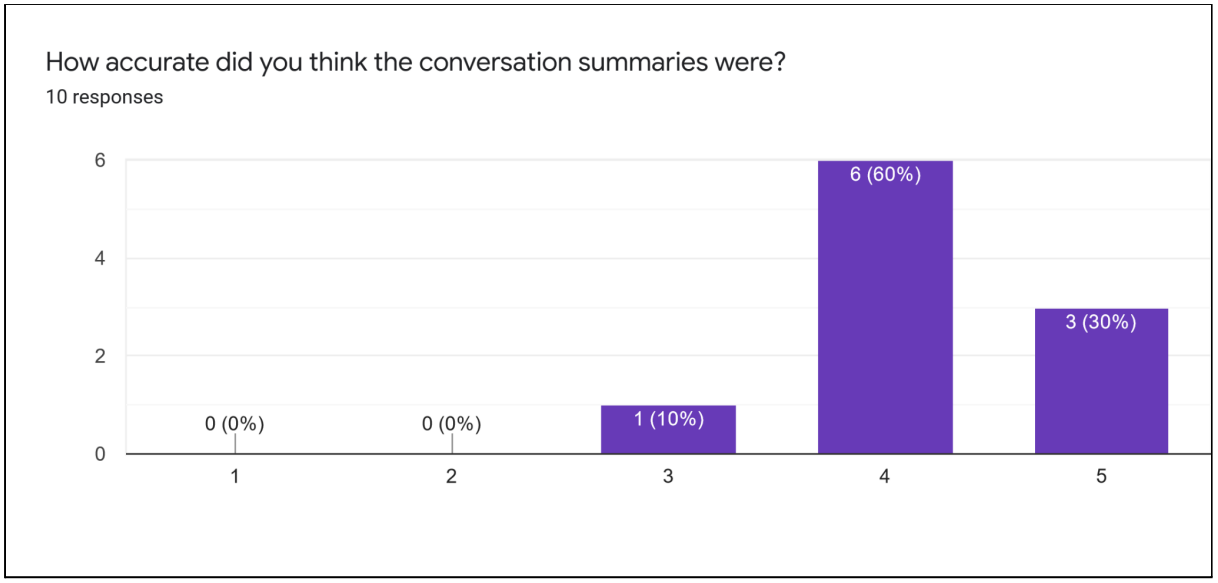
This question received 100% “yes” responses, indicating that the messaging feature worked as appropriate. This made sense, as there was an emphasis on making sure that sending and receiving messages was instant.

Did you like the Summarisation feature?



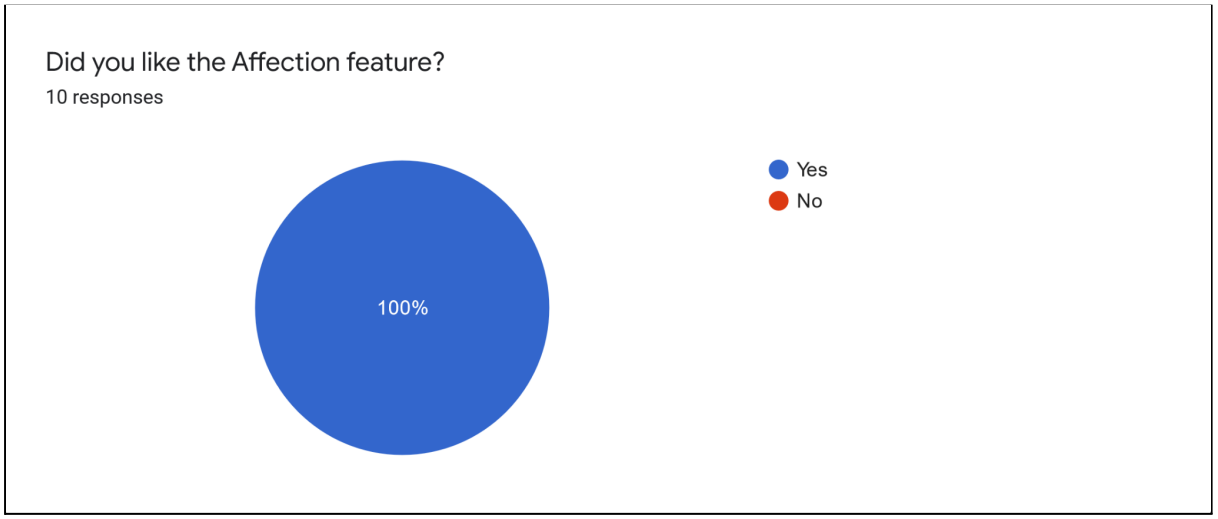
All of the responses to this question were “yes”. Overall, all users enjoyed the feature.

How accurate did you think the conversation summarisations were?



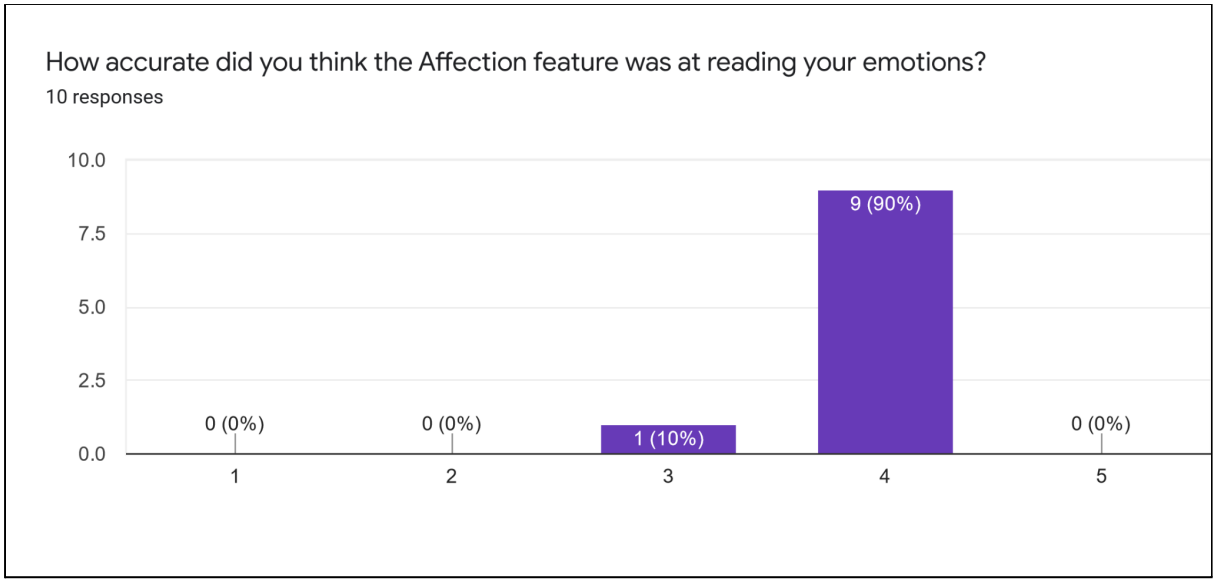
Of the 10 responses, there was an average score of 4.2 out of 5. As a percentage, this works out to be 84%. This meant that the majority of the users thought the summaries were quite accurate.

Did you like the Affection feature?



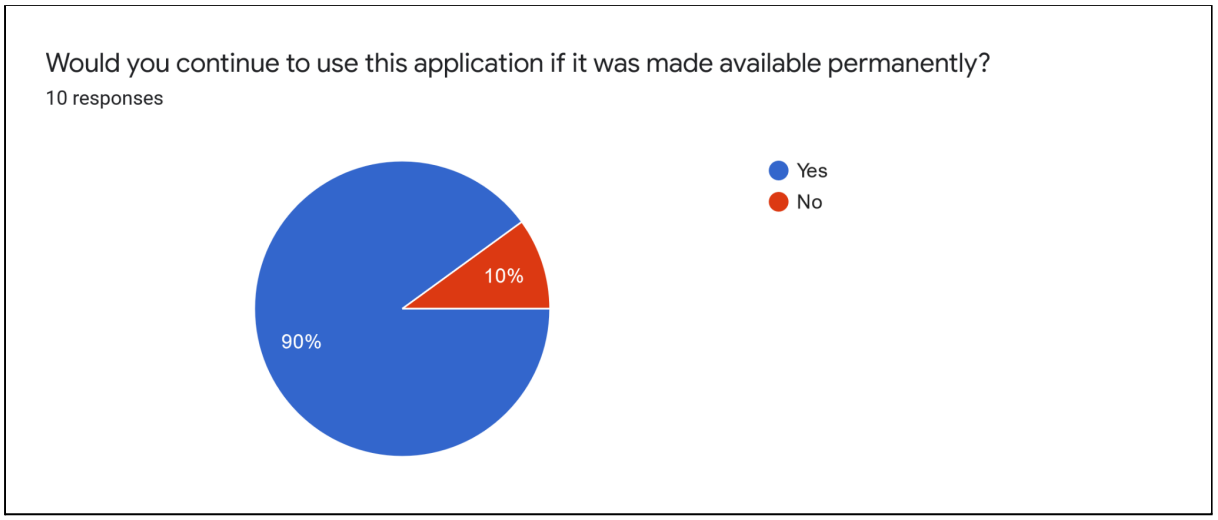
All of the responses to this question said “yes”. The users enjoyed sending messages and seeing what affection would be attached to them.

How accurate did you think the Affection feature was at reading your emotions?



The user testers were asked to answer with a score out of 5. The average score was 3.9 (78%). This indicated that the overall reaction to this feature was positive.

Would you continue to use this application if it was made available permanently?



9 out of 10 of the testers responded to this question with “yes”. They enjoyed the emotion detection and summarisation.

Are there any improvements you would like to see implemented?

Although this question was optional, all 10 of the respondents answered it. One popular request was to include notifications in the future so users can see when they have received a message.

Another popular request was the addition of multimedia, such as voice messages and photos. This was a good idea as the idea of sentiment analysis could be incorporated into the photos using image classification.

Analysis

The user testing proved to be a worthy time investment as it allowed us to bring Sentiscape to a larger audience and see how the Summarisation and Affection features worked on people who had no prior knowledge of the application.

From the results, it appears that the basic functionality of the application works with no noticeable bugs. The additional features were also popular and when asked if they'd continue to use the application in the future, everyone said yes. This suggests that the Summarisation and Affection features were unique "hooks" that make people want to come back to the application as no other application is offering the same functionality.

Users provided some valuable feedback in terms of improvements and they are additions we would like to consider if future work is to take place on the application.