Unit Test Report – Voice Capsule by the Voice Capsulators, Revision Date 11/21/2021

List Of Modules and performed unit tests:

1. **Authentication** -- Tested by Marianna Marcelline
    a. Landing page: Opens app:
        i. Expected behavior: The user should be directed to the landing page
    b. Sign up screen: Press ‘SIGN-UP’ button:
        i. Expected behavior: The user should be directed to the ‘Create account’ page
    c. Enter valid Email, full name and password fields:
        i. Expected behavior: User should be able to input text into the fields
    d. Sign up screen: Press ‘SAVE’
        i. Expected behavior: a new entry in the “users” document in Cloud Firestore is created that is named after the particular user’s ID with subfields corresponding to the sent and pending capsules (which start off with no subentries)
    e. Log-in screen: Press ‘LOGIN button on landing page:
        i. Expected behavior: User should be directed to the ‘Sign in with email’ page
    f. Sign in with email screen: User inputs valid email into the email field and presses ‘NEXT’ button:
        i. Expected behavior: User email should be checked against existing emails in Firestore to see if it exists. As it is a valid email, the user will be directed to ‘Password’ page after pressing ‘NEXT’ button
    g. ‘Sign in’ page: User inputs valid password into password field and presses ‘SIGN-IN’ button
        i. Expected behaviour: User password and email combination should be checked against credentials in firebase to see if they match. As this is a valid password, the user should have successfully logged in, they will be directed to the record button page.

2. **Recording/Playback** -- Tested by Kyle Won
    a. Recorder: start and then immediately stop recording (double click)
        i. Expected behavior: recording starts and immediately stops
    b. Recorder: start and then stop recording anytime before recording time limit
        i. Expected behavior: recording starts and then stops when the stop button is clicked
    c. Recorder: start and then wait until recording time limit
        i. Expected behavior: recording starts and then stops when the time limit is reached
    d. Player: with playback in progress, press “pause” at any point in recording
        i. Expected behavior: playback pauses

e. Player: with playback in progress, press “skip backwards 5 seconds”
    i. Expected behavior: playback goes back 5 seconds and continues
f. Player: with playback in progress, press “skip forwards 5 seconds”
    i. Expected behavior: playback goes forwards 5 seconds and continues
g. Player: with playback in progress, press “restart”
    i. Expected behavior: playback goes to beginning of recording and start playing
h. Player: with playback in progress, press “stop”
    i. Expected behavior: playback stops, goes to beginning of recording and does not start playing
i. Player: with playback paused at any point in the recording, press “play”
    i. Expected behavior: playback starts from current point in recording
j. Player: with playback paused, press “skip backwards 5 seconds”
    i. Expected behavior: playback goes back 5 seconds and doesn’t play
k. Player: with playback paused, press “skip forwards 5 seconds”
    i. Expected behavior: playback goes forwards 5 seconds and doesn’t play
l. Player: with playback paused, press “restart”
    i. Expected behavior: playback goes to beginning of recording and doesn’t play
m. Player: with playback paused, press “stop”
    i. Expected behavior: playback goes to beginning of recording and doesn’t play

3. **Capsules**: The capsules page, time opened, etc -- Tested by Jovita Martinez
    a. Regarding the functions for this module, the first class is the “CapsulesSlide”. This class is tested manually when opening the Capsule app page itself. The next function “_CapsulesSlideState” loads the list of available voice capsules with “initState” which initializes animation and loads capsules from local storage. Another function “checkForNewCapsules”, checks if any are capsules available, if so it is downloaded from the database. These functions were tested incrementally with the testing approaches listed below.
    b. Capsule Page: Standard test case, using a new account, press “capsules” from the bottom navigation bar.
        i. The Capsule page should be empty, no voice capsules should be there.
        ii. After pressing the refresh button, no voice capsules should be there.
    c. Capsule Page: Sending a recording to myself for the next day.
        i. The Capsule page should be empty, no voice capsules should be there.
        ii. After pressing the refresh button, no voice capsules should be there.
        iii. The next day, after pressing the refresh button, voice capsules should be there. Download button should download the recording to local storage. Delete button should delete the recording from the capsule page.
    d. Capsule Page: Sending receiving a recording for the next day.
        i. The Capsule page should be empty, no voice capsules should be there.

        ii. The next day, after pressing the refresh button, voice capsules should be there. Download button should download the recording to local storage. Delete button should delete the recording from the capsule page.
4. **Friends**: Adding friends… Displaying contacts -- Tested by Ricardo Gonzalez
    a. List of friends:
        i. Expected behavior: for every friend, there is a list element displaying the friend's full name and email address.
        ii. If there are pending friend requests, there should be an indication of so.
    b. Refresh button:
        i. Expected behavior: on press, the friend's list should display the user's most recent list of friends. If there are pending friend requests, there should be an indication of so.
    c. Add Friends screen (a form for sending friend requests):
        i. Expected behavior: Takes a friend's email and sends a friend request to a user who:
            1. Exists
            2. Is not the user sending the request
            3. Has not sent the current user a friend request
            4. Is not already a friend.
        ii. If one of these requirements are not met, the user is prompted to re enter the email address.
    d. Friend Requests screen:
        i. Expected behavior: for every friend request, there is a list element displaying the friend's full name, email address, and buttons to accept and decline requests.
        ii. When an action is taken, there is feedback confirming the action.

5. **Uploading to database (Firebase)** -- Tested by Daniel Zuniga
    a. Sign-Up Screen: after a user has input a valid email address, name, and password, press the "SIGN-UP" button
        i. Expected behavior: a new entry in the "users" document in Cloud Firestore is created that is named after the particular user's ID with subfields corresponding to the sent and pending capsules (which start off with no subentries)
    b. Send Screen: after the recording has finished being processed by the recording module, the voice recording is temporarily saved to local storage in a cache
        i. Expected behavior: the locally saved recording is saved as "/data/user/0/com.ucsc.voice_capsule/cache/recorded_file.mp4" and is confirmed by a popup on the bottom of the screen
    c. Send Screen: while here, press "Click to select open date"
        i. Expected behavior: calendar picker pops up followed by a time picker in order to specify when to allow the Voice Capsule to be listened to
    d. Send Screen: while here, press on the dropdown menu next to "Select recipient"

        i. Expected behavior: the list of the currently logged in user's friends appears for selection
    e. Send Screen: while here, pressing the "SEND" button without specifying a date
        i. Expected behavior: an error dialog appears to inform the user that they must select a date and time for which the Voice Capsule can be opened
    f. Storage: after a user has chosen a date/time and a contact to send the Voice Capsule to, press the "SEND" button
        i. Expected behavior: the locally named "recorded_file.mp4" is uploaded to the target user's directory on Firebase Storage in the following file path: "<receiver_id>/outgoing_<sender_id>_<send_date_and_time>.mp4"
    g. Database: after a user has chosen a date/time and a contact to send the Voice Capsule to, press the "SEND" button
        i. Expected behavior: update the "sent_capsules" document within the "capsules" collection for the sending user and update the open date time, receiver name, receiver user ID, and send date time fields
        ii. Expected behavior: update the "pending_capsules" document within the "capsules" collection for the receiving user and update the open date time, sender name, sender user ID, and storage path fields
6. **Downloading from database (Firebase)** -- Tested by Kyle Won
    a. With no capsule in the database, click "refresh" on capsules page
        i. Expected behavior: pop-up message indicating no new capsules available
    b. With a capsule in the database whose open date/time is before or equal to the current date/time, click "refresh" on capsules page
        i. Expected behavior: fetch capsule from database and show it on capsules screen
    c. With a capsule in the database whose open date/time is after the current date/time, click "refresh" on capsules page
        i. Expected behavior: pop-up message indicating no new capsules available
    d. After receiving a new capsule from the database, delete the capsule from the capsules page before opening it and press "refresh" again
        i. Expected behavior: fetch new capsule from database again and show it on capsules screen
    e. After receiving a new capsule from the database and opening it, delete the capsule from the capsules page and press "refresh" again
        i. Expected behavior: capsule has been deleted from database after opening, so show pop-up message indicating no new capsules available

Testing approach:

- Used manual testing approach
- Components were tested after they were developed
- Developer determined some equivalence classes which represented all significant inputs
- Developer acted as control point—feeding the CUT predetermined input for each test scenario

- Developer acted as observation point—using assertions, print statements, and visual inspection to determine whether the test output matched the expected output
- Unit test passed if developer observed that test output matched expected output for all test scenarios
- Upon unit test failure, developer modified the code and ran the test again

Manual testing implications:

- Test coverage was not as comprehensive as an automated test framework
- All tests were not easily repeatable or reusable since test code was removed after tests passed
- Lightweight testing allowed developers to focus on integral feature development
- Shortened time to reach the Minimum Viable Product at the cost of defect density