Product name: Don't Dine Alone
Team name: Granddaddy of All Teams
Date: 5/31/18

System and Unit Test Report

System Test Scenarios:

Stories:

A. User story 1 from sprint 1: As a user, I want a basic interactive UI / login screen, so that I know where to start.
B. User story 2 from sprint 1: As a user, I want to be able to login with my ucsc email address, so I only link with other UCSC students.

Scenario:

1. Start the Don't Dine Alone app
2. If you aren't currently registered click the register button (otherwise proceed to step e.)
    a. Enter a UCSC email address, password, and password confirmation then click register.
    b. User should not be able to register with a non-UCSC email address.
    c. User should not be able to register if passwords don't match.
    d. Otherwise, user should be brought back to the login screen with a message telling them to check their email to confirm registration.
    e. Confirm registration in the email, then in the app enter the email and password you just registered and click login.
    f. User should be brought to Welcome page.

Stories:

User story 1 from sprint 2: As a user, I want a welcome page so that the app can be more appealing to me.

Scenario:

1. (While on the Welcome page) User should see text showing all online users (if other users are currently online).
2. User should see buttons for Edit Profile, Log Out, Preferences, and Start Matching.
3. Edit Profile button should bring the user to the Edit Profile page.
4. Logout button should log the user out and bring them back to the login page.
5. Preferences button should open a pop-up for selecting preferences.
6. Start Matching should change color and bring the user to the next activity when a match is found.

Stories:

User story 2 from sprint 2: As a user, I want a profile so that I will be unique.

Scenario:

1. (From the Welcome page) Click the Edit Profile button.
2. Enter a Display Name, Gender, Spirit Animal, the press OK.
3. User should be brought back to the welcome page, they should now see “Welcome [their name]. Their profile information should now be located in the database.

Stories:

User story 3 from sprint 2: As a user, I want to manually select which dining hall I would like to eat at, so that I can choose anyplace I want to eat on campus.

User story 3 from sprint 2: As a user, I want to have preferences so that I can have more choice in who I group with.

Scenario:

1. (From the Lobby Page) Click the Preferences button.
2. Checkbox your preferred dining halls.
    a. If you press none, you will be asked again to make a selection.
3. User should then see that their preferences have been saved.

Stories:

User story 1 from sprint 4 (pushed back from story 2: sprint 2, and story 1: sprint 3): As a user, I want to be able to match with other users so I can meet new people.

User story 3 from sprint 3: As a user, I want to confirm my matches, so I can verify when I match with someone.

User story 4 from sprint 3: As a user, I want messaging, so I can contact the people I am dining with.

Scenario:

1. (From the welcome page) Click Match Preferences, a pop up will appear asking for preferred group size and preferred dining halls, by default all options should be selected.
2. Select your preferred option or leave all selected. If you are testing matching with a second phone make sure you have at least one group size / dining hall choice in common. Note: I you don’t have preferences in common you should not match with them.
3. Click start matching, as soon as you’ve matched with as many people that are in the group you will be transferred to the chat page.
4. Send a chat and verify the other user receives it.
5. Click the Leave button to exit the chat (reverse confirmation).

Stories:

User story 2 from sprint 4: As a user, I want to receive push notifications so I know when I have a match.

Scenario:

1. For matching notifications:
    a. (From the Welcome page) click Start Matching then exit the app.
    b. Have a user on another phone start matching.
    c. Once they match with you you should receive a push notification saying you've matched.
2. For message notifications:
    a. Once you are matched and in a chat exit the app.
    b. Have the user you are matched with enter a message in the chat.
    c. You should receive a push notification saying you've received a message, clicking it should bring you back to the chat.

Stories:

User story 3 from sprint 4: As a user, I want to be able to report via email, so I can report abuses.

Scenario:

1.User experience problems with other individuals or something that needs to brought to the developer team's attention:

a. In the Lobby, press button "Report Abuse", this will then launch your email application
b. If any error should error, a toast message will display and say "Error Loading Email Application please send your problem to support@dontdinealone.com", since there is a problem with launching, the user will have to manually email to us.

Unit Tests:

Testing is located in folders "src/androidTest" and "src/Test".
We currently do not have automated testing/continuous integration setup.

The "SUT" folder in "src/androidTest" was created so that the Entity (i.e. app user's data)
is not affected by testing. It currently doesn't completely work, but what it endeavored to do was:
- SaveEntityService: preserves and restores the user's Entity data
- EntityUnderTest: mocks an app user by creating a mock of FirebaseAuth and FirebaseUser
  along with Task that indicates success and failure.

We also create a pool of valid and invalid inputs, updating the tests as necessary when an
additional test cases are added.

The following were deemed to not require dynamic unit testing:
- Classes that only consist of getters and setters
- Classes that only hold references to the database.
- Classes that basically are wrappers around calls to components.
- Classes that basically are wrappers around calls to updating local and remote data.

Classes not requiring dynamic unit testing:
- Classes in the "data/entity" folder (excluding AuthUser which contains major logic).
- Classes in the "data/model" folder (excluding MatchPreferences which contains some logic).
- Classes in the "res" folder (because they just contain constants).
- Classes in the "util" folder (because it currently just provides base callbacks).

Classes with complete unit testing (i.e. with equivalence classes covered):
- PrimitiveArrayService

Classes that still need unit testing:
[data]:
- AuthUser:
    - Takes the FirebaseAuth and FirebaseUser information.
    - Then AuthUser initializes the rest of the Entity data both local and/or remotely,
      initializing to defaults.
    * We currently do not directly unit test this. We only indirectly confirm that
      the data is being updated correctly through the testing of the activities.

[model]:
- MatchPreferences:
    - has helper logic that is only partially tested.
- NotificationService
- OnlineService

- Tightly coupled with AuthUser, which is currently complicated and in need of testing.

[net]
- Connection
- Reader
- Writer

[main]
- all activities have partial testing except MatchedActivity which has none.

[Server]

Test Authors
- Tarun: null View checks for all activities except MatchedActivity
- Tyler: LobbyActivityTest
- Cody: EditProfileActivityTest, SUT
- Jean: SUT, PrimitiveArrayServiceTest, ViewServiceTest, paired Cody and Tyler
- Mackenzie: MatchPreferencesTest
- no one: RegisterActivityTest, MatchedActivity, MainActivity

Besides the simple PrimitiveArrayServiceTest, we were unable to have enough time to implement equivalence test coverage for most of the code. However, we setup the testing environment and debugged it such that almost full coverage would be a believable goal if we had another sprint iteration.