System and Unit Test Report

Product: Sluggo-iOS

Team: Sluggo Team

Date: 05/30/2021

System Testing:

Sprint 1:

User story 2: As a user I want to be able to switch between different team contexts.

Scenario:

1. Start the Sluggo application, login with any user in multiple teams.
2. Navigate to the team select sidebar by swiping from the left on any top level view (Home, Tickets, Members, or Admin)
3. Select another team in the table that you want to switch to, the background will refresh.
4. Swipe out of the sidebar, and go to the homeview, notice that it corresponds to the other team.

Sprint 2:

User story 2: As a team member, I want to create a ticket to track a task over the course of the project lifetime.

Scenario:

1. Start the Sluggo application, login with a user.
2. Navigate to the ticket view through hitting the Ticket tab on the bottom left, then the plus button on the top right.
3. Create a ticket through filling out all fields.
    a. Set the title to: “Design the ROV”
    b. Set the description to: “Finish the ROV design by 10/20/21 to qualify to finals”
    c. Enable the due date and set it to 10/20/21
4. Hit save in the top right, which will close the modal and return you to the ticket view.
5. Select the ticket titled “Design the ROV”, you will see the contents you previously created.

Sprint 3:

User story 1: As a team member, I want to quickly glance at any tickets that are assigned to me.

Scenario:

1. Start Sluggo app, login
2. Navigate to ticket view
3. Create a new ticket with title <arbitrary>
4. Click on the ticket, and assign it to yourself
5. Navigate to the homepage, the assigned ticket should be there.

Sprint 4:

User story 1: As a team members I'd want to be able to invite users to the team

Scenario:

1. Open the Django admin dashboard, navigate to the user model page and add a user
    a. Username should be <arbitrary>
    b. Email should be <arbitrary@arbitrary.arbitrary>
    c. Password should be <p@ssw0rd12>
2. Add another user called <to be invited>, with email <to_be_invited@asdf.com>
3. Open the Django admin dashboard, navigate to the team model page and add a team
    a. Team name should be <arbitrary>
4. In the Django admin dashboard, navigate to the members model page and add an admin member
    a. User should be the user <arbitrary>
    b. Team should be the team <arbitrary>
5. Open the app logging in with <arbitrary>
6. Navigate to the admin tab, select invites, and select add
7. In the dialog, enter the email <to_be_invited@asdf.com>
8. Log out of <arbitrary>
9. Log into <to_be_invited>
10. In the team selection screen, navigate to invites, and accept the invite.

Unit Testing

Sluggo-iOS unit testing:

1. Model-Based Testing:

Since all models are just structs, the major test cases that can be conducted on them are just the CRUD functionalities. These tests are all built around the JsonLoader class, whose purpose is to convert to and from the JSON understood by the backend API and the Swift models. Each test takes in or compares against an example JSON string used for the validation process:

Member Tests:

These tests (linked here) are built around the Member struct, the model used to represent the users of a team on Sluggo. Within this test, we check different aspects of the structure: can it be deserialized from a JSON response with just the basic (required properties), can it still be deserialized once we add all the optional parameters, can it be properly serialized back, and can it be deserialized from a paginated list JSON response. All of these are tasks that are necessary with every API request Sluggo-iOS makes.

This test mechanism is re-utilized in the rest of the model based tests that occur on the application, all linked here.

2. Manager-Based testing:

Ticket Manager Test

The ticket manager test (linked here) creates an exampleUser and an exampleMember. With these, the test asserts that that list url and the filtered list url generation are correct.

3. Backend Tests

The API has tests which assert the correctness of all endpoints. While this isn't necessarily a unit test for each model, it does prove the efficacy of the interface. Most of these tests follow a pretty standard set of steps. The test function first creates necessary data in the test database. Django's APIClient executes the request. The state of the database and the results of the request are asserted for correctness.