Internal Testing Documentation

Note: commands such as npm install may need to be prefixed by sudo on some platforms.

How to run the back end tests

Our back-end tests are run through making API calls to the endpoints. We use the *Jest* testing framework and the *Supertest* library to run these tests. To run the tests yourself, please do the following steps:

1. Install MongoDB locally on your machine.
2. Create a directory called 'erukuDatabase'
3. Run mongod --dbpath=/path/to/erukuDatabase (this assumes you have mongod in your path)
4. Within the cloned repository, make a copy of .env.example and rename it to .env in the backend directory.
5. Within the .env file, specify the mongo URI. If the mongo port is 27017 (the default), then have MONGODB_URI=mongodb://localhost:27017/ in the .env file.
6. Within the .env file, replace <aws_access_key_id> with AWS access key id and <aws_secret_access_key> with AWS secret access key.
7. Run npm install within the backend directory. Ensure that the package.json contains *Jest* and *Supertest*.
8. Run npm run test.

*If some of the test cases are failing, that is probably because you have pre-existing info inside of the MongoDB database. Please ensure those are deleted before you run the tests.

To do this, run mongo and then run show dbs. It should list the databases. Assuming you just recently installed mongoDB, there should only be admin and local. If there are databases such as Events, then you need to drop them before you run the tests. Check this (https://docs.mongodb.com/manual/reference/method/db.dropDatabase/) to drop a database.

How to run the front end tests

Simply run npm install followed by npm test or jest --verbose --runInBand tests from the project root directory to run all tests written for the front end.

These tests have also been written using the *Jest* testing framework and require no additional dependencies. The option to use a test renderer library had been explored but due to the complexity and number of bugs with the libraries involved in the setup, this was not possible. Instead, the front-end tests verify the behaviour of the UI components by testing their underlying function implementations (refactored into separate files) with mock state and HTTP requests.

How to manually test push notifications

As testing push notifications is difficult to automate, we provide steps for manual testing below:

1. Run npm install, expo login, and npm start or expo start in the project directory.
2. Run npm install in the backend directory. In the .env file, include PUSH_NOTIFICATIONS_SCHEDULE=* * * * *. This will trigger the backend server to send the push notifications every minute. Run npm start in the backend directory.
3. Open the front-end mobile application on a physical mobile device, sign up or log in if not already authenticated, and ensure that the Receive Notifications toggle in the Settings page is in the active/enabled state to ensure that incoming notifications will be received.
4. Navigate to the Upcoming Events page in the application, create a future event and set the time of the event to an hour ahead from the current time.

How to run the stress tests

The backend server must be running on the default port in order to perform the stress tests. Python 3 also needs to be installed in order to run the tests. Simply run npm install -g artillery to install the artillery stress testing toolkit. Once that is completed run the following command python3 stress.py inside the eruku-mesi/stress directory. The command will run multiple stress tests and load tests for all of the endpoints in the backend. The stress test will output a file named report.txt. The report can then be reviewed and give details on how the backend performed. Status codes of 404 are not expected in the report. If a status code of 404 appears in the report then it indicates that an error occurred while stress testing. Status codes of 500 and 400 are expected if testing the backend when passing in faulty parameters. Since the backend logic does not allow duplicate inputs (for example two users sharing the same username) the stress test will return a 500 and 400 code for certain functions and is expected. This is expected for and '/add' route or any POST request. This is due to the fact that the stress test tool will create N instance of virtual users that all execute the same command with the same parameters. The stress tests still execute a heavy workload on the backend server.

Configuring the stress tests

The config.yaml configuration file can be modified to change the number of requests and duration of stress testing. Inside the configuration file, simply change the duration to the number of seconds you would like the stress test to take and change the arrivalRate to the number of virtual users that will be created per second to execute the test. For example, in the default configuration, the arrivalRate is set to 1, and the duration is set to 10. This means each yaml file will create 1 virtual user to execute the scenario defined in that yaml per for 10 seconds.

Acceptance tests

The above tests also serve as acceptance tests and are based on our functional requirements.
SRS - https://gitlab.cas.mcmaster.ca/tut/eruku-mesi/-/blob/master/doc/srs.md