TESTING

Front End Testing

Overview:

The roomies application utilizes Go as the front and back end frameworks for development. The testing environment will follow principally unit testing standards. However, front end testing proves to be a difficult task for such testing. However, the front end framework Fyne, offers robust solutions to be able to accurately and efficiently create unit test cases that test the front end functionality of the code.

Test To Be Performed:

- Overall functionality of Components.
    - This test will have separate directories for each of the main views on the application, primarily, the homepage, grocery list page, and widgets page. It will select each of the buttons and fields individually. For text boxes and other information fields, the testing framework will fill the field and test that the field is the same for the test.
- Paging Functionality
    - This test will page through each of the different views that the user is able to select. By paging through each of the views, the tester will place unit tests to ensure that the time to page between views is not excessive and that the necessary components on each page are loaded into the user view port.
- Full Stack Integration Testing
    - These tests will test the entire pipeline of multiple activities that the user can perform including sending and receiving information from the server. This test will send a simple query through the Go interface and expect a response from the server given the query is successful.

Frequency of Tests:

- Overall Functionality of Components
    - Given that the front end usage of buttons and functionalities will be changing rapidly and frequently, it is expected that after each bug fix that each developer run the test harness to ensure functionality of their specific component. It is also expected that each developer write integrated tests for each component. This will ensure modularity of the testing environment and complete testing
- Paging Functionality
    - Paging functionality will often remain the same between each of the views (Messaging, Widgets, Grocery list) and will likely not need to be modified or adjusted frequently by the tester. However, given the usage if unit testing and the ease of implementation, it is still expected that the test harness for paging functionality be run at the completion of each task. Because of the dependence on separate variables within the application, the components must be tested for each page and ensure that the paging functionality remains consistent for each view.
- Full Stack Integration Testing
    - The full integration testing is the most vital and frequently disrupted test in the harness. Because of this reason, the full stack integration tests should be completed frequently and consistently. Along with testing individual components as previously discussed. The team should instantiate end to end tests before

committing code to the Git codebase. This will ensure highly functional and basic code. By testing responses from the back to front end and ensuring that the response returned from the server is what is expected, the full stack integration testing allows for complete and inclusive tests that will be an overall tell of functionality for the application.

Measure of Success per test:

- Overall Functionality of Components
    - For the front end overall functionality of components to be in an accepted state, each of the components in the platform must be accepting and returning the correct input and output from the standard unit tests. This requires 100% passing for each component.
- Paging Functionality
    - Similar to the overall functionality of the individual components test, the paging functionality also must pass all tests with 100%. This test assumes that each of the components between the pages retains functionality.
        - Furthermore, these tests need to ensure isolation meaning that each of the tests run to completion and that one of the tests do not affected any of the others.
        - Tests for paging functionality must also ensure durability. This entails that the modification of one of the elements on the other page will not have a direct impact or change to the current page.
        - Lastly the tests must also be consistency. Consistency means that between each of the pages, when one component is changed, it is also changed within the other paged views.
    - Tests for paging functionality will be determined successful if the above ACID properties are utilized https://en.wikipedia.org/wiki/ACID
- Full Stack Integration Testing
    - Full integration tests like the previous tests before must also incorporate ACID properties but are more focused on the overall functionality of the application
    - These tests focus on the results and interactions between the front and back end of the tech stack. To be determined successful the tester needs to be able to reach an endpoint and receive a result that is consistent with the type of query that the user input.

Example Code

```
package main

import (
	"testing"
)

func TestGreeting(t *testing.T) {
}

out, in := makeUI()
```

```
        if out.Text != "Hello world!" {
                t.Error("Incorrect initial greeting")
        }

test.Type(in, "Andy")
        if out.Text != "Hello Andy!" {
                t.Error("Incorrect user greeting")
        }
```

Back End Testing

Overview:

The roomies application utilizes Go as the front and back end frameworks for development. The testing environment will follow industry unit testing standards. Tests will run periodically as part of a pipeline to ensure updates are pushed and merged without breaking changes.

Test To Be Performed:

- Endpoint testing
    - Each endpoint will contain a fair amount of logic for its desired function. Unit tests will be implemented covering these core pieces of the backend architecture.
- DAO functionality
    - The DAOs will interact with our database and guarantee the integrity of the database as well as the operations that we need to perform on them. This will be carried out using unit tests.
- Router testing
    - With the router we have a variety of pathways and data being passed through to the endpoints. Unit tests will be implemented to ensure that is happening properly

Frequency of Tests:

- We are planning to implement a make command and a pipeline for these tests to be run at least when we create pull requests and are working to merge new changes into master. This will allow us to ensure no breaking changes are introduced.

Measure of Success per test:

- Endpoint testing
    - These tests will be successful if the logic operates correctly and gives us a valid result..
- DAO functionality
    - These tests will be successful if we can properly interact with the database and perform the needed operations on them.
- Router testing
    - These tests will be successful if the logic operates correctly and calls the correct handlers.