Testing Plan

This testing plan suggests what should be done during the development of the system. It is based on the recent book by Mauricio Aniche - Effective Software Testing (2021).

Overview

This document goes over how the team will conduct unit tests, integration tests, and system tests over the course of the development of the Decagon Condo Management System. It will also outline the tools used for each type of test.

Unit Testing

Unit tests test small units of code, isolated from the rest. Its goal is to test only the basic functionality of a function or class. Unit tests test the behavior of the code itself and by themselves are not capable of validating that the requirements and specifications of the system set by the stakeholders are being respected.

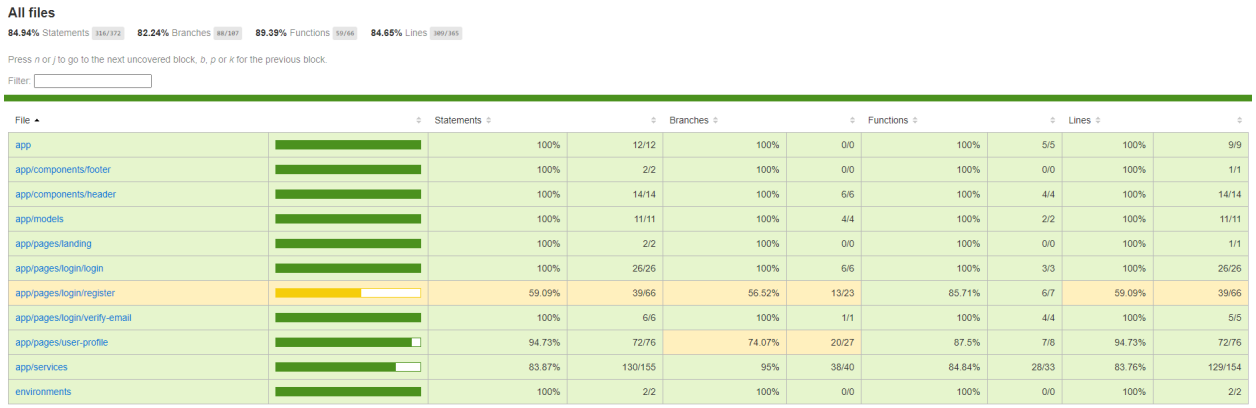
The Decagon Condo Management System is being developed with the Angular framework using TypeScript and a Firebase backend. The Jasmine testing framework is installed by Angular for testing JavaScript and TypeScript code. Each Angular component is generated with a test file where all the unit tests for that component will be written. The “ng test” command from the Angular CLI will run all the Jasmine tests using the Karma test runner which will show which tests pass or fail and the error when a test fails. Adding the “--code-coverage” parameter to the command will report how many many statements, branches, functions, and lines are covered with a percent for each of them. We want to have at least 80% coverage for statements. Adding the “--no-watch” parameter will generate a html file outlining which statements from each class is covered by a test and which are not which will make achieving this test coverage easier.

Each developer will write and run unit tests for the components that they work on. Unit tests are also configured to run on GitHub automatically after every push to a branch and every pull request. The test run is configured to fail if the statement coverage is less that 80%, enforcing this requirement.

Coverage example

```
=============================== Coverage summary ===============================
Statements   : 84.94% ( 316/372 )
Branches     : 82.24% ( 88/107 )
Functions    : 89.39% ( 59/66 )
Lines        : 84.65% ( 309/365 )
================================================================================
```

Visualization of coverage



All files

84.94% Statements 316/372 **82.24%** Branches 88/107 **89.39%** Functions 59/66 **84.65%** Lines 309/365

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
app		100%	12/12	100%	0/0	100%	5/5	100%	9/9
app/components/footer		100%	2/2	100%	0/0	100%	0/0	100%	1/1
app/components/header		100%	14/14	100%	6/6	100%	4/4	100%	14/14
app/models		100%	11/11	100%	4/4	100%	2/2	100%	11/11
app/pages/landing		100%	2/2	100%	0/0	100%	0/0	100%	1/1
app/pages/login/login		100%	26/26	100%	6/6	100%	3/3	100%	26/26
app/pages/login/register		59.09%	39/66	56.52%	13/23	85.71%	6/7	59.09%	39/66
app/pages/login/verify-email		100%	6/6	100%	1/1	100%	4/4	100%	5/5
app/pages/user-profile		94.73%	72/76	74.07%	20/27	87.5%	7/8	94.73%	72/76
app/services		83.87%	130/155	95%	38/40	84.84%	28/33	83.76%	129/154
environments		100%	2/2	100%	0/0	100%	0/0	100%	2/2

Integration Testing

The goal of integration tests is to test multiple related components to ensure they work well together. We are using cypress end to end testing to test multiple related components.

For the Decagon Condo Management System, we can create integration tests with cypress to test multiple components on the same page, and switch between pages required to perform an action. We can assert that the components cooperate the way they are intended to.

These tests will ensure that different systems work together properly. Mainly, the team wants to assure that the front-end system (Angular and TypeScript) and back-end system (Firebase services) work together well. Cypress provides a way to intercept API calls easily which makes writing tests simple.

The cypress tests are configured to run using the command “npm run cypress:run” or “npm run cypress:open” which will open the tests in a browser which will visualize the process. Cypress is also configured to run on GitHub for every push to a branch and pull request.

System Testing

The goal of system testing is to test the system as a whole. It is meant to test all the requirements from start to finish. We will also do manual tests where we test the system requirements manually. We do these tests throughout the implementation and during the review process to ensure that the system is performing well.

Cypress offers the team end-to-end testing which is exactly what we need to perform for system testing. It is convenient since these tests are automated and interact with the software application on its own. Furthermore, since these tests are automated, they can be introduced into the cd/ci pipeline, which enables regression. Regression testing ensures that requirements tested in the past, still work to this day and in the future. All in all, this ensures that system requirements are being tested fast and efficiently.