Software Testing Plan

March 28th 2017

Five Pixels

Sponsor:
Joy Knudsen

Faculty Mentor:
Dr. Mohamed Elwakil

Team Members:
Brandon Garling
Xiangzhi Cao
Matthew Nielsen
Mohammad Alsobhi
Clarissa Calderon

Rev 1.0

1. Introduction 2

1.1. Purpose 2

1.2. Project Overview 2

2. Unit Testing 3

2.1. Overview 3

2.2. Frontend 4

2.2.1. BackendCommunicatorService 4

2.3. Backend 5

2.3.1. Serializer 6

3. Integration Testing 7

3.1. Overview 7

3.2. Frontend 7

3.2.1. ConfigService 8

3.3. Backend 9

3.3.1. Database Integration 9

3.3.2. Notifications Integration 9

4. Usability Testing 10

4.1. Overview 10

4.2. End to End Testing 10

4.3. Use Case 11

5. Conclusion 11

1. Introduction

1.1. Purpose

This test plan describes the testing approach and overall framework that will drive the testing of Groupwise, the group matching application for FlagFriends. This document introduces:

- Unit Testing: Testing small independent pieces of the application in order to ensure overall functionality
- Integration Testing: Testing the integration between all the small independent pieces of the application
- Usability Testing: Testing the usability of the application as a whole, interacting with the application and ensuring that all functionality works as intended

1.2. Project Overview

5 Pixels is creating a web application called Groupwise for a local Flagstaff non-profit organization named FlagFriends. With this program, we will bring the public the opportunity to take part in the experience of hosting foreign students with the intentions of having the students and hosts both learn about the experiences of each other. FlagFriends is currently running a program where incoming international students and Flagstaff hosts can connect and share experiences as well as orient the student with campus and the surrounding area. Currently, FlagFriends is running at minimum efficiency due to its completely manual process. Groupwise will change that by allowing students and hosts to create an account on the FlagFriends.org website and have the users create profiles and start the self-matching service that the webapp will provide. Students and hosts should be able to match themselves by viewing profiles and by chatting with each other. When they match themselves, the administrator will see the matches. He or she can still see how the program operates but she will not have to get involved with the matching process the way she used to. In the past, the administrator had to manually receive and manage paperwork to create matches with students and hosts, the web application for FlagFriends will allow him or her to monitor the program and will allow her to focus on other tasks such as marketing. The resulting applications for FlagFriends should make everyone's lives easier, students and hosts can pair up with whenever they want instead of trusting an administrator to create a good match, and the administrator will be saved days of work.

We will be testing the entire Groupwise application. The most important part of the service are accounts and the matching process. We have to make sure that anyone that opens our website can view the website in whichever browser they are using. We have to make sure and test that users are able to create accounts and use every function of the website that they are given. Different account types have different privileges, the admin account which will be given to the administrator(s) should be tested so that it can completely modify the service. Admin account holders should be able to modify questions students and hosts need to fill out to create their profile. Host and student accounts should be tested to make sure they can create a profile, view other profiles, and message other users. Hosts should be able to message students and students should be able to message hosts. The team has to make sure the matching process is running and working properly, the webapp doesn't create a match, the hosts and students match themselves after interacting with other users. Hosts and students should be able to look through profiles by using filters. The filter should be working to show specific profiles to users that ask for specific details about hosts or students. The upload image process in user accounts needs to be working along with the fields that allow users to enter information when answering their profile details.

The ultimate, unattainable goal of software testing is to provide a perfect product to its users. With software testing programmers can save hours of work that would have been needed had a user encountered an error. If a user finds an error that impedes what they are wanting to do when using a program, they can sometimes stop using the program; Errors or bugs in programs cause users to leave the service and possibly rate the program badly. Then that same user that left the service can spread information about the program being badly constructed. Software testing can make sure nothing goes wrong when users use the program they are using and in some cases software testing can even save lives, such as in programs that are used in space or any security program. The reason 5 Pixels is implementing tests is so that we make sure we deliver a quality service and so that users are able to properly use FlagFriends to create friendships that will last lifetimes.

2. Unit Testing

2.1. Overview

In order to unit test our software, we need to split our testing into two sections: frontend and backend unit tests. In our case, both our frontend and our backend use Javascript, we can leverage this so we only have to learn how to test with a few different testing libraries. To ensure that our software is well tested, we plan to unit test as many independent parts of the project as we can by carefully going through the project and identifying key components of our system that are prone to error and could be detrimental if they were to fail. We will be using Jasmine and the Karma test runner to run our testing to facilitate both our frontend and backend testing. Karma has many plugins which help in gathering relevant testing information such as code coverage that we can leverage to better assess how good our testing is.

2.2. Frontend

First focusing on the frontend; Using the Jasmine and Karma libraries we can test our code by importing our different components and testing independent functionality of each component.

Angular 2 can be split into a few different distinct building blocks: components, directives, pipes, services, and guards. A NPM package we leverage generates blueprints for each one of these distinct building blocks and creates sample test harnesses for each one which we can build our actual tests into. This makes it exceedingly simple to start testing our frontend Angular 2 code.

Quickly identifying some main pieces of our frontend application code, we can pick out some main services we would want to test thoroughly.

2.2.1. BackendCommunicatorService

Overall Functionality

- This service is extended in all other services that implement any kind of backend communication. It is responsible for extracting data from a payload response from the server, processing the data, and returning the result. This also handles creating and responding to errors appropriately.

<u>Function - *extractData(model: any, res: Response)*</u>

Extracts data from a response from the server.

- Boundary Values
  - Null, undefined, JSON without any keys, JSON with random keys
  - An HTTP response status of something other than 200
- Equivalence Partitions
  - JSON with an Error key and some value
  - JSON with an Error key and a Payload key with some values
  - JSON with a Payload key and some value
  - Null, undefined
  - Empty JSON
  - JSON with random keys

- Selected Test Cases
    - {“Error”: null}, HTTP: 200
    - {“Payload”: null}, HTTP: 200
    - {“Error”: [], “Payload”: null}, HTTP: 200
    - {“Error”: [“Some value”], “Payload”: {...}}, HTTP: 200
    - {“RandomKey”: {...}, “Payload”: {...}}, HTTP: 200
    - {“Error”: [], “Payload”: {...}}, HTTP: 200
    - {“Error”: [], “Payload”: {...}}, HTTP: 500

<u>Function - *parseError(res: Response)*</u>

Checks a response for errors and wraps them in a RestError container class if any exist, otherwise returns null.

- Boundary Values
    - Null, undefined, JSON without any keys, JSON with random keys
    - An HTTP response status of something other than 200
- Equivalence Partitions
    - JSON with an Error key and some value
    - JSON with an Error key and a Payload key with some values
    - JSON with a Payload key and some value
    - Null, undefined
    - Empty JSON
    - JSON with random keys
- Selected Test Cases
    - {“Error”: null}, HTTP: 200
    - {“Payload”: null}, HTTP: 200
    - {“Error”: [], “Payload”: null}, HTTP: 200
    - {“Error”: [“Some value”], “Payload”: {...}}, HTTP: 200
    - {“RandomKey”: {...}, “Payload”: {...}}, HTTP: 200
    - {“Error”: [], “Payload”: {...}}, HTTP: 200
    - {“Error”: [], “Payload”: {...}}, HTTP: 500

2.3. Backend

Next focusing on the backend; Because we broke our implementation out into different modules we can import our different modules and test the functionality of each function (where applicable) and ensure that they respond correctly to different inputs. We may also employ mocking to ensure that we are indeed only testing one independent piece of the application that doesn’t depend on another piece of the application. We can create separate test files for each one of our custom modules and create tests for each one, and finally create a master Karma config file that loads each test file and runs them.

Delving further into this we can focus our testing on the most important pieces of the backend and expand as time allows. Following the backend workflow, we can test the different pieces which each request from the frontend must follow, specifically each request follows a route to service workflow. Routes simply provide endpoints for the frontend to communicate with and pass all relevant parameters of a request to the correct service. In this case, we can focus our testing on the services layer of our workflow. In addition, because our backend service is almost entirely REST based we can focus on the custom functions we implemented to help facilitate this communication that are common to all services. We can also perform some simple testing on some of our other user made modules such as the mailer, encryption, logging, notifications, socket, and setup modules.

2.3.1. Serializer

Overall Functionality

- This module enables converting Data Transfer Objects (DTOs) to models and vice versa.

Function - *serialzeModel(model: SequelizeModel)*

Converts a Sequelize model to a simple JSON object, stripping out any keys which are not specified as serializable in the DTO map.

- Boundary Values
  - A model with no values
- Equivalence Partitions
  - Null, undefined
  - Something other than a Sequelize model
  - A Sequelize model
  - A Sequelize model that does not contain DTO mapping directions
- Selected Test Cases
  - A model with no values
  - Something other than a Sequelize model
  - A Sequelize model
  - A Sequelize model that does not contain DTO mapping directions

Function - *serializeModels(models: SequelizeModel [])*

Converts an array of Sequelize models to a simple JSON array containing JSON objects, stripping out any keys which are not specified as serializable in the DTO map.

- Boundary Values
  - An empty array
  - An array containing a model with no values
  - An array containing multiple different models

        - An array containing a single Sequelize model
    - Equivalence Partitions
        - Null, undefined
        - An array containing something other than a Sequelize model
        - An array of containing a multiple Sequelize models
    - Selected Test Cases
        - An empty array
        - An array containing a model with no values
        - An array containing multiple different models
        - An array containing a single Sequelize model
        - Null, undefined
        - An array containing something other than a Sequelize model
        - An array of containing a multiple Sequelize models

3. Integration Testing

3.1. Overview

Our major piece of integration testing will be testing the connectivity between the client and server and ensuring that communication is stable. Once again, we can leverage Jasmine and Karma test runner. This client server communication involves websockets as well as REST API calls and testing socket communications.

3.2. Frontend

In addition to testing the communication between client and server we can leverage another feature of the Angular 2 CLI package which is e2e (end-to-end) testing. This comes preconfigured for our project and allows us to use a package called Protractor to interact with the page and ensure that events are fired and routes are followed correctly.

Some tests we will conduct include:

- Testing going to pages which are guarded and ensuring that we are correctly redirected to the login page when appropriate
- Ensuring that each page can be rendered without any errors
- Ensuring that specific elements are present on rendered pages
- Ensuring that routing works for each of our different routes, including our page not found (404) route

- Ensuring that backend data is correctly loaded and displayed

3.2.1. ConfigService

Overall Functionality

- This service is responsible for retrieving and setting custom configuration fields on the server.

<u>Function - *getValue(key: string)*</u>

Retrieves a configuration value from the server given a specified configuration key.

- Boundary Values
    - Key: “”
    - Key: A key which does not exist on the server
    - Key: An exceedingly long key (>1024 characters)
    - Key: A key with spaces
- Equivalence Partitions
    - Key: Null, undefined
    - Key: A normal string
- Selected Test Cases
    - Key: “”
    - Key: A key with 1024 characters
    - Key: A key which does not exist on the server
    - Key: A key which does exist on the server

<u>Function - *setValue(key: string, value: any)*</u>

Sets a configuration value on the server given a specified configuration key and value.

- Boundary Values
    - Key: “”
    - Key: A key which does not exist on the server
    - Key: An exceedingly long key (>1024 characters)
    - Key: A key with spaces
- Equivalence Partitions
    - Key: Null, undefined
    - Key: A normal string
- Selected Test Cases
    - Key: “”
    - Key: A key with 1024 characters
    - Key: A key which does not exist on the server
    - Key: A key which does exist on the server

3.3. Backend

Our backend can also benefit from further integration testing between the different services and modules ensuring that data is passed correctly between modules, and errors are caught and handled correctly in all cases. In addition, we can further test our integration with external services such as our mail service and database directly.

Some testing we will conduct include:

3.3.1. Database Integration

Data Mapping

Test whether the data that is being sent back and forth from the UI to the database.

- Boundary Values
    - Data is misplaced
    - Data is not communicated
    - Data cannot be retrieved
    - Data is too big
- Selected Test Cases
    - Data is too big

ACID - *Atomicity, Consistency, Isolation, Durability*

Ensure the data follows the rules of ACID; pass or fail, state of the database is always valid, simultaneous requests to the database work as if done one at a time, and the system will endure events like power loss after a transaction is completed.

- Boundary Values
    - Data is stored with corrupted elements
    - Some reason to break consistency
    - Multiple requests to store/retrieve data to/from database
    - System reboot or crash for any reason
- Selected Test Cases
    - Multiple requests to store, retrieve data from database
    - System reboot or crash for any reason

3.3.2. Notifications Integration

Scheduled Tasks

Ensure that all scheduled tasks and notification checks occur on time and emit the correct events.

- Boundary Values
    - No notifications found
    - One notification found
- Selected Test Cases
    - No notifications found
    - One notification found
    - Many notifications found

4. Usability Testing

4.1. Overview

Usability testing is used to make sure users get the experience intended for them when using the service. For example, usability tests we will carry out will check if users can load the service in their browser. We will also test if the pages and text uses sizes that are well displayed on computer screens and phone screens. Users from different backgrounds will be using the software and everyone needs to be able to get the web application working as not everyone has technical knowledge to fix errors that may be encountered. Usability tests check if the software can run and be used.

4.2. End to End Testing

We will use Protractor for usability testing, which is an end-to-end test framework for Angular 2 applications, and it can run tests against our application running in a real browser, interacting with it as a user would.

In addition to using Protractor we will also be employing actual user testing by working with our sponsor and selecting a small group of testers for the application who we can work with and gather feedback for working and not working functions.

4.3. Use Case

We will release an alpha of the program and enlist people that are already registered with FlagFriends to try it out. At the same time, we will conduct a Google Forms survey and send it out to the users. We hope to recruit a large enough number of testers to give us a good idea of last minute UI/UX bugs and improvements.

We will also use this opportunity to test the web socket communication we have implemented for users to talk to each other through the application. This will help us improve a vital function of the application, as the biggest tool users will have to communicate with each other pre-match is our chat system.

This is the test we hope most carries out since getting a number of individuals to help who will actually be using the software when the software is released, will cause the frontend be restructured based on user feedback.

5. Conclusion

5 Pixels is working day and night to bring FlagFriends the software it deserves. It will unite participants of the program by making the process more efficient for individuals to make friends and help orient some NAU students who would have a harder time finding their way in the city. Groupwise will eliminate the need for the paper applications and the archaic filing system used by FlagFriends currently.

In our development of Groupwise, 5 Pixels will be using unit tests, integration tests, and usability tests to eliminate possible errors that could arise. 5 Pixels is using unit tests to test small pieces of the software. Integration testing tests the conjunction of the small pieces to ensure they work together. Usability testing is used to make sure different types of people can use the software. Testing is a very important part of software development as it guarantees the software will operate as it should. 5 Pixels will give the software to the administrator by May and with the use of testing, everything that the program needs to function will work. We are thankful for working on this project and we look forward to make sure it works properly.