Master Test Plan

Introduction

This is the test plan for the Innisfree Web Portal. Our testing will encompass all functionality of the final version of our website. This will include tests for individual controllers and models, as well as tests to make sure that the views are displaying as intended. We will have automated tests that provide 100% code coverage. In addition, there will be qualitative tests of the views to make sure that they work as intended, and the layouts are in accord with the requirements.

Assumptions

1. All required resources, i.e. servers for testing, etc. will be available during the process
2. This MTP is assuming that the current version of the website is the site to be tested. This document may need to be expanded depending on forthcoming changes to the site.

Parts to be Tested & What Tests to Write

I. Requirements Testing

The tests will each contribute to testing one of the given requirements. The unit tests will test all code that contributes to meeting these requirements. However, the requirements will also be tested manually, to ensure that the code works together properly and as intended. The requirements can be found under the innisfree github wiki here.

II. Unit Testing

For each of the components in the MVC, every part of the code will be covered in full.

- Every method will be tested by a specific unit test
- The unit test will also test all interactions between different components of the MVC.
- Most components have actions that can only be performed by an authorized user. Separate unit tests will be created to make sure that
    1. Unauthorized users cannot access said functionality
    2. Authorized users can access said functionality
- Unit tests will also be created to make sure that related method calls (e.g. from before_action and before_filter) will activate if the coded condition is true, and act as expected.
- We have assigned members of the team to create unit tests on different sections of the site to complete the coverage percentage requirements:
- Domenic: Users Controller
- Xavier: Residents Controller
- Robert: Appointments Controller
- Will: Cars Controller
- Zoe: Doctors Helper, Doctors Model, Houses Model
- Bethany: Users Model

Models

- appointment.rb
- apt_type.rb
- car.rb
- doctor.rb
- house.rb
- post.rb
- recurring_reminder.rb
- report.rb
- reservation.rb
- resident.rb
- user.rb

Views

- appointments
- cars
- devise
- doctors
- houses
- layouts
- newusers
- notification_mailer (emailed reminders)
- reports
- residents
- settings
- users

Controllers

- appointments_controller.rb
- cars_controller.rb
- doctors_controller.rb
- houses_controller.rb
- reports_controller.rb
- residents_controller.rb
- settings_controller.rb
- users_controller.rb

III. Usability Testing

Usability testing will be done by the customer themselves. We will continue to test as the final features are added are added per the TDD methodology. The customer will continue to test the site and provide feedback based on their usability needs. We will also find testers who have no experience with the site, and give them a set of tasks to complete, which will be based on the core requirements of the site, and observe them and collect feedback in the process. This will ensure that the site flow is usable and easy to understand even for someone with no experience with the system.

IV. Security Testing

Security testing will be largely carried out through the unit tests, which will make sure that the user must be properly authenticated to access each part of the website. We have multiple levels of authentication, including admin, staff, volunteer, and workstation head. Each of these levels will have permissions specifying which parts of the site they can access, which the unit tests will ensure is possible. As part of our requirements testing, will will also manually ensure that unauthorized users cannot access parts of the site or perform actions that they are restricted from.

Malformed URLs

Security testing will include making sure that users without access to a specific page or feature cannot navigate to that page as the appropriate links will be hidden to them. Trying to manually navigate there through URLs will return users to the main page and display an error, as with trying to navigate to any non-existent part of the website.

V. Installation Instructions Test

To test the installation instructions, we will follow them to deploy our application to BlueHost. Once deployed, we will run all of the rake tests to ensure it is correctly deployed. Once that's done, we'll try to access the URL, just to make sure all the webserver loaded correctly and all the resources loaded correctly.

VI. Compatibility Testing

We will execute the requirements test on the most recent version of the major web browsers (Firefox, Chrome, IE, and Safari). If — and only if — the requirements test passes on all of those browsers, then the compability test will be considered to have passed.

Who Will Write Tests?

The entire development team will contribute to writing the tests. During our testing phases, we will split up the test writing based on components of the MVC, and each person will be responsible for complete code coverage and test writing for that section. For example, one person might write all the unit and requirements tests for the residents model, controller, and view, while another person is in charge of the tests for appointments.

Cases to Test

In addition to the cases specified by the requirements and unit tests, we will also have additional tests for corner cases, malformed input, and scheduling. We will have to ensure that required form elements must be filled, and creating duplicate entries will result in an error. Form input that is too long or contains illegal characters will display an appropriate error, and redirect the user to the original form with instructions on what they did incorrectly. On submitting a form, we will have to test that the data was manipulated correctly, which will require using fixtures and inspecting the databases. As a corollary to this, there will also be manual tests to make sure that manipulating the data is reflected correctly in the views. These tests will need to be performed for every MVC component.

Indications of Testing

Each test will test the requirements at a fundamental level, as each test will test a M/V/C component that builds up to a specified requirement. Each unit test will specify in a comment which requirement the test contributes to. The requirements to be listed as part of these tests can be found in the “Requirements Testing” section.