1. Introduction

The purpose of this project was to build an administrative web platform for the Rivanna Trails Foundation (RTF), a non-profit located in Charlottesville tasked with administering the day-to-day upkeep of the Rivanna Trails. The web platform was to make the tasks of RTF easier by providing a consolidated software system that would manage all of the information RTF needed on a day-to-day basis. This document serves as the Master Test Plan of the RTF project as the main functionality has been implemented and the focus is shifting towards testing.

1.1. Scope

The goal of the testing effort is to ensure that all of the requirements of the software system have been adequately implemented, that the system is usable by the intended audience, that there are no security flaws, and that the installation of the system is clear. We will assume when planning out our tests (specifically our usability and installation tests) that our audience has a non-technical background, which increases the scope of what we need to test to ensure that the system is usable to an audience other than the developers.

Exclusions: parts of of the system that rely directly on the Django framework

Inclusions: every piece of functionality that we wrote

Limits: things we assume that already work, like Django and all of the packages that we have used

1.2. References

There are two reference documents: the requirements.md and install.md documents in the Github repository. These documents are important to understanding the system framework and requirements, along with the installation instructions that we have created for RTF.

1.3. System overview and key features

Refer to requirements document for details.

1.5. Test overview

1.5.1 Organization

The unit testing process begins early in the development cycle and is performed continuously through the end of development. Other tests (requirements, usability, installation, and security testing) are performed after the major development phase is over. Any issues brought up during testing of the code are brought to whoever is responsible for the failing piece of code. Any issues with requirements or usability, or whose resolution is unclear, are brought to the customer for resolution.

1.5.2 Master test schedule

Unit testing takes place continuously from iteration 3 to 14 to ensure continuous feedback to the development process. The other types of testing will primarily take place during iterations 12 to 14.

1.5.5 Responsibilities

In general, individuals will write tests for functionality that they are responsible for (this applies to unit tests). The requirements tests will be done by everyone, because the requirements will be clarified as a group with the customer to ensure that everyone understands the entire project. The security tests will also be performed by everyone in the group. The usability tests will be performed by Trisha and Elizabeth, who have been involved with developing the user interface, along with external people who are not involved in the project to ensure that our system is straightforward to use.

1.5.6 Tools, techniques, methods, and metrics

Determine which tests to write: We are trying to ensure that the functionality we are writing is performing in the way that we want it to. This is the main pillar that will define how we are writing our tests for the different portions of the system.

Hardware: Ubuntu Server with a connection to the Internet

Software: Django framework, MySQL, Google Maps JS API, Shapely,

Test Tools: Django Unit Testing, QUnit (JS), Blanket (JS), PhantomJS to create JS coverage report

Test Techniques: We are writing tests that demonstrate the desired functionality as well as in-depth tests that involve understanding the specific implementation (black-box and white-box tests). Certain behaviors will require mocking functionality, such as HTML 5 Location and certain Window behaviors in JS.

Test Environment: Unit tests to be done in a sandbox with CircleCI. Other tests to be done directly on the course and/or deployment server.

2. Details of the Master Test Plan

2.1. Test processes including definition of test levels

Task	**Generate requirements tests**
Methods	Gather requirements from the customer and ensure that the requirements are clear and feasible. After implementing the requirements, ensure that the implemented requirements match the actual requirements given by the customer.
Inputs	Requirements List
Outputs	Validated overall features
Schedule	This begins 2 iterations after development begins, and ends when development ends
Resources	Requirements List
Risk(s) and assumptions	*Risk*: Our idea of a requirement is different from the customer’s idea of the requirement

	Assumption: We meet with the customer regularly, so this should be mitigated
Roles and responsibilities	All members in the team will be responsible for clarifying and implementing the requirements as the features are developed

Task	**Generate Unit tests**
Methods	For each feature implemented based on the requirements, we want to write a unit test to make sure that the implemented feature behaves the way we expect it to
Inputs	Source Code
Outputs	Validated features
Schedule	This begins 2 iterations after development begins, and ends 2 iterations after development ends
Resources	Requirements List, Source Code
Risk(s) and assumptions	*Risk*: We do not think about the entire scope of how the functions should behave *Assumption*: With all of us writing tests, we will be able to cover a good comprehensive portion of it
Roles and responsibilities	Generally, those who write a specific piece of the functionality will be the person who writes the unit tests

Task	**Generate security tests**
Methods	Understand the security risks and ensure that our system is robust enough to mitigate those risks
Inputs	Software System

Outputs	Validated security features
Schedule	This begins 2 iterations before the end of development and continues on to 2 iterations after the end of development.
Resources	Customer, hardware
Risk(s) and assumptions	*Risk*: We do not consider all of the potential security flaws of the system *Assumption*: We will be able to think about this in collaboration with the customer to understand the specific security holes he is concerned about
Roles and responsibilities	Each member will be responsible for testing out this segment, both in planning and executing

Task	**Generate usability tests**
Methods	Understand how a normal user would use the system and ensure that the user experience is up to par with that
Inputs	Technical users, nontechnical users, and the customer
Outputs	Validated user experience
Schedule	This begins after the first semester of development is finished and ends when all of the development is finished
Resources	Personnel to test the usability of the system, the software
Risk(s) and assumptions	*Risk*: We do not think about all of the different users that could be using the system and leave out a key user group that may be affected by the usability *Assumption*: Between all of our peers, we would come up with a solid group of people to test the system

Roles and responsibilities	We all will be responsible for testing out this segment, both in planning and executing

Task	**Generate installation tests**
Methods	Understand how the intended person in charge of installing our system will be able to understand the installation process. Develop a clear document that will aid in installing the system.
Inputs	Installation instructions
Outputs	A correctly installed system
Schedule	This begins at the beginning of the 2nd semester after the required features are developed
Resources	Customer, sandboxed container, software system
Risk(s) and assumptions	*Risk*: We do not understand how our customer would interpret our installation instructions, or misjudge our customer's knowledge of the software and how it needs to be installed *Assumption*: We will be able to mitigate this by meeting with the customer and discussing the installation plan
Roles and responsibilities	Only one or two members of the team will be needed to develop this plan and ensure that the customer understands it