Test Plan

# Purpose

*Obviously, the purpose is to test your software… But what does “testing” mean in your case? Ensuring data integrity is maintained across synchronized databases? Ensure that UI features function properly and with minimal delays in interactivity? Ensure the installer works? (What does “work” mean?) Are you testing all aspects of your software, only certain parts?*

Check that the UI functions properly

Check that the application does not crash unexpectedly

Ensure that updates occur in a timely manner

Ensure data validity across users

Ensure that data is manipulated as user expects

Test that features do what they are listed as doing

Test integration of features into application

# Assumptions

*Is there anything your software requires? Something it assumes is working, and that you aren’t responsible for? (Example: Our product relies on the Google Maps API for location data, and as such we don’t verify the accuracy of the information retrieved… we assume it’s correct).*

User has Python installed

All modules in use work correctly and are installed

# Testing procedure

*What steps are you going to take in order to verify <thing that you are testing>?*

* *Manual testing?*
* *Automated scripts or other processes?*
* *Unit tests?*
* *Something else?*

*Regardless of what you are going to do/use, how does that thing work? Explain it here.*

In order to test integration as well as the functionality of various parts of our program, unit testing will be utilized. Unit tests that manufacture both typical use cases and scenarios that the team predicts to be troublesome will be implemented in their own runnable .py files with dependencies on the components of the project that they are designed to test. A specific unit testing file could be called “DatabaseUnitTest.py”. An example unit test within it could be the “ConsecutiveUserNameChangesTest” which would return a true or false depending on whether the state of the database is what it ought to be.

In order to ensure that our UX is pleasant and that the various visual elements of the project manual testing is required. After each major feature implementation, team members will use the software in order to identify possible issues. Attempts will be made to both use the software normally and to produce abnormal user input through the application window. An example of a normal test would be whether the Sprints View opens when the Sprints View button is clicked. Abnormal use would include something like repeatedly closing and opening a window.

# Tools / Required Resources

*What is necessary to create or execute any of the tests? Are you using any outside resources? Anything to download or install? Do testers (which are often developers too) need to create an account on* [www.AwesomeTestingToolWeUse.com](http://www.AwesomeTestingToolWeUse.com)*?*

In order to test the software, all that is required is what is required to run the application. Unit testing will occur using the native Python UnitTest library. Testers will need to have the various source files used to make the project as the UnitTests have a dependence on the source code itself.

Manual testing should be able to occur so long as the testers are able to run the application itself.

# Verification

*How do you know the results of the tests? What sort of mechanism do you have to track this? Spreadsheets? Emailing of results? Are the tests quick, with simple screen output of Success/Failure as sufficient indicators?*

*Would someone other than the person who wrote the testing system be able to make any sense of it? Would a new team member be able to interpret results, or use the same tools with a minimal amount of fuss?*

Test results from unit testing will be output to a CSV file, the file will state what test failed in what unit test section. From there manual debugging will be necessary on the part of the developer. Manual testing will require testers to contact the development team with details regarding any problems, likely via Slack or through posting a new issue on Pivotal Tracker or the Github.

The test results are very simple to interpret and the unit tests themselves are extremely simple to implement so long as the author of the unit tests understands how to call the functions they are testing. Writing useful tests will require the author to have a knowledge of how the function should behave and some knowledge of internal mechanisms to know what problematic input would look like.