

SolarSize - Code Testing Plan

SSE Capstone – University of Regina

SolarSize

Winter 2022

Tristan Brown-Hannibal

Karlee Fidek

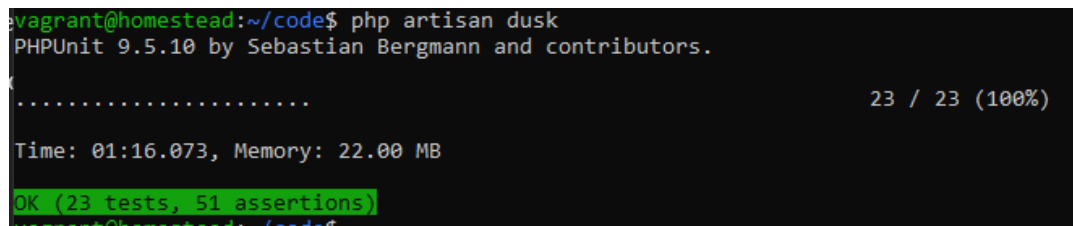
Kaden Goski

Executive Summary

This is a code testing plan for our capstone project, SolarSize. SolarSize is a web-based application that utilizes building consumption metrics and solar intensity data to calculate and size solar panel installations based on ROI. The goal of the application is to suggest ideal solar installations that maximize ROI while minimizing power costs. This is done by taking in various inputs and utilizing NASA solar irradiation data to calculate estimated power production values.

Our code testing took place by designing the tests following our defined methodology in this document. We used an automated script program, Laravel Dusk, to write tests and covered good input values, erroneous input values, edge case inputs, and whole calculations with verification of results. Dusk allowed us to run tests virtually in Google Chrome; taking screenshots of progress in tests and when they had failures. This proved useful when things went awry or just to confirm the tests were working as they were supposed to.

Our final test suite and results included 23 tests with 51 assertions. The tests ran successfully and our project worked as intended. An example of the successful runs can be seen in Figure 1 along with an example test.



```
vagrant@homestead:~/code$ php artisan dusk
PHPUnit 9.5.10 by Sebastian Bergmann and contributors.

.....
23 / 23 (100%)

Time: 01:16.073, Memory: 22.00 MB

OK (23 tests, 51 assertions)
```

Figure 1. Final Test Results – Successful Run



```
public function testLongi360Panel()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
        ->keys('#vs1__combobox > div.vs__selected-options > input','Test','{ENTER}')
        ->keys('#grant','00')
        ->keys('#interest',{backspace},{backspace},'5')
        ->assertValueIsNot('#latitude','0')
        ->assertValueIsNot('#longitude','0')
        ->attach("#fileUpload > div > input", '/home/vagrant/code/tests/Browser/monthdata_cleaned.csv')
        ->screenshot('panelTest1')
        ->click('#mainForm > div.submit-container > button.submit-button')
        ->screenshot('panelTest1.1')
        ->waitForText('Return Statistics',30)
        ->scrollIntoView('#app > div.extra > div:nth-child(2) > div:nth-child(1) > div:nth-child(1) > div > header > h3')
        ->screenshot('panelTest1.2')
        ->assertSee('317796');
    });
    $browser->driver->manage()->deleteAllCookies();
};
```

Figure 2. Solar Panel Calculation Test

Table of Contents

Executive Summary	i
Introduction.....	1
Purpose.....	1
Project	1
Stakeholders	1
Code Testing Strategy	1
Code Testing Objectives	1
Test Assumptions.....	1
Test Principles.....	2
Scope of Tests.....	2
Execution and Management Process	2
Test Management Tool.....	2
Test Design	2
Test Execution and Failure Analysis.....	3
Test Results.....	3
Results	3
Issues	4

List of Figures

<i>Figure 1. Final Test Results – Successful Run</i>	<i>3</i>
<i>Figure 2. Solar Panel Calculation Test</i>	<i>3</i>

Introduction

Purpose

This code testing plan describes the testing approach, tools, and environment used to test our capstone project, SolarSize – A solar estimation tool. This document will cover:

- i. Code Testing Strategy: Objective and methodology of the tests, assumptions and process outline to setup tests, principles, and scope of the tests.
- ii. Execution and Management Process: Describes the tools and environment used to run the tests, how each test was design in line with the code testing strategy, and test execution along with failure analysis and bug fixing implementation.
- iii. Test Results: The overall results of the tests that were implemented and tested, along with any issues that were faced along the way.

Project

SolarSize is a web-based application that utilizes building consumption metrics and solar intensity data to calculate and size solar panel installations based on ROI. The goal of the application is to suggest ideal solar installations that maximize ROI while minimizing power costs. It does this by calculating how much a given solar panel will output; taking into account numerous variables such as panel tilt, solar irradiation data from NASA, and more.

Stakeholders

The stakeholders of this project are; us (the SolarSize project team), our mentors Dr. Timothy Maciag and Dr. Kin-Choong Yow, and GreenWave Innovations, the project business partner and ultimately the end users.

Code Testing Strategy

Code Testing Objectives

The objective of the code testing is to validate and verify the functionality of the SolarSize application according to our desired design.

The code testing will execute and assert test scripts, output any failures and screenshot results along the way. This will be achieved by using a test suite that utilizes Chrome to simulate usage of the application to get desired results.

Test Assumptions

- Data and desired results will be calculated beforehand and used to tailor the application in testing.
- Testing will be automated and ran via scripts.

- This testing will be similar to user testing / UAT; but using automated scripts to ensure desired results.
- Any test failures will result in a failure of the whole script and flagging of the failed test.
- The test scripts will look at the output and inputs, not the inner workings of the program, a black box approach.

Test Principles

- Testing will focus on meeting and ensuring baseline goals of the project. Core functionality over niche uses.
- Tests will be created using a consistent baseline procedure.
- Testing environment will run locally on the server.
- Testing will be repeatable at any moment when triggered.
- Testing will have success and failure results (save screenshots on complex and failed tests).

Scope of Tests

The scope of the tests will include: input validation and associated error messages, desired calculations and results from sequences of inputs, observing graphs for proper values, and any other edge case tests.

Execution and Management Process

Test Management Tool

The management tool for the scripts was chosen by looking at meeting the requirements of the project and considering the existing framework. The existing framework consisted of PHP Laravel and an Apache HTTP server. This meant that an ideal test management tool would work with both of these natively. We also preferred that tools did not require extra drivers such as Selenium to avoid co-dependencies.

Taking these requirements into mind, a tool created by Laravel, Laravel Dusk was selected. It worked natively with our existing framework and provided automated testing of the web application using Chrome or Firefox.

It offers basic tests such as observing that an element is set to a value or that an error message appears when an invalid input is selected. It also offers enhanced functionality such as sequence testing with which we can test that our model is calculating panel output or ROI correctly by observing the summary page after inputting on the inputs page. Another key feature is screenshots of the test environment when it runs or when it fails. This allows for visual confirmation of tests and helps to confirm what went wrong.

Test Design

The design of our tests focused on two main areas; inputs and calculations. Testing the inputs was as simple as determining edge cases, valid cases and invalid, and creating tests for these. These confirmed that the application did not accept invalid inputs and outputted error messages or alternatively accepted the valid inputs.

Testing the calculations was more complex, but done with the same core principles in mind. We created calculation tests by determining the edge case combination inputs that should result in different recommendations in ROI, panel choice, how many panels, etc. This was capped at one of each; panel recommendation, a minimum, median, and maximum number of panels, negative ROI or close to 0 over lifespan, and a good ROI (under 15 years to payback).

Test Execution and Failure Analysis

Test execution is currently set to manually run with the command “php artisan dusk” in the terminal of the server. This is to be done every time we push to our GitHub. It would be more ideal to have an automated pipeline (CI/CD) to run these tests, however this is a whole extra layer that just was not feasible in the scope of our project.

In the case of a failure, the tests will return with red text and the results will be saved to a failure file. This is then analysed by looking at the file and recreating the steps with a debugger in Chrome. The goal of this is to diagnose and fix the issue and then rerun the tests, repeating this process until it passes.

Test Results

Results

In the end, in the last week of March, we had 23 tests with 51 assertions. We deemed this number sufficient as we covered all the base and edge cases of our program with inputs and calculations.

While executing our tests throughout our development process, we would get intermittent failures. They showed when our pushes broke or updated a calculation. This proved very useful as it could be used to confirm or expose changes that were done. Without this feedback, we would have had many more intermittent issues and undiagnosed bugs.

```
vagrant@homestead:~/code$ php artisan dusk
PHPUnit 9.5.10 by Sebastian Bergmann and contributors.

.....                                     23 / 23 (100%)

Time: 01:16.073, Memory: 22.00 MB

OK (23 tests, 51 assertions)
vagrant@homestead:~/code$
```

Figure 1. Final Test Results – Successful Run

```
public function testLongi360Panel()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
        ->keys('#vs1_combobox > div.vs__selected-options > input','Test',{ENTER})
        ->keys('#grant','00')
        ->keys('#interest',{backspace},{backspace},'5')
        ->assertValueIsNot('#latitude','0')
        ->assertValueIsNot('#longitude','0')
        ->attach("#fileUpload > div > input", '/home/vagrant/code/tests/Browser/monthdata_cleaned.csv')
        ->screenshot('panelTest1')
        ->click('#mainForm > div.submit-container > button.submit-button')
        ->screenshot('panelTest1.1')
        ->waitForText('Return Statistics',30)
        ->scrollIntoView('#app > div.extra > div:nth-child(2) > div:nth-child(1) > div:nth-child(1) > div > header > h3')
        ->screenshot('panelTest1.2')
        ->assertSee('317796');
    });
    $browser->driver->manage()->deleteAllCookies();
});
```

Figure 2. Solar Panel Calculation Test

Issues

Overall, Laravel Dusk proved to be a powerful tool that allowed us to write PHP unit tests that controlled our application, entering inputs, and asserting them and overall calculations. However, it came with a learning curve and writing these tests took some trial and error. Functionality such as clearing inputs would not work, so we had to workaroud it by using key inputs such as backspace to clear inputs in-between tests of the same type.