

# **SolarSize - Code Testing Plan**

SSE Capstone – University of Regina

SolarSize

Winter 2022

Tristan Brown-Hannibal

Karlee Fidek

Kaden Goski

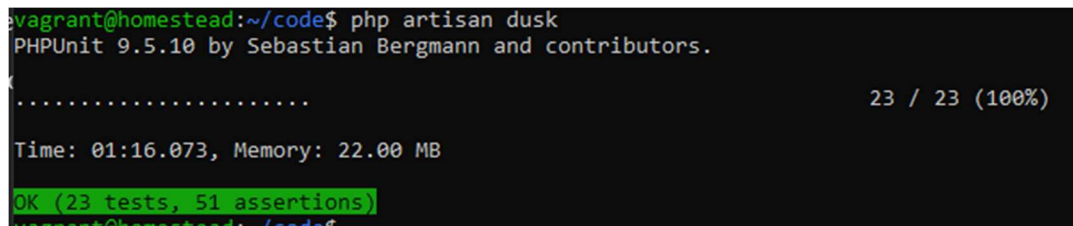


## Executive Summary

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

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.