**Application to test:** Paint Calculator

**Steps to launch:**

* open command prompt
* go to folder where app is downloaded
* 'python run.py'
* open <http://127.0.0.1:5000/> in browser

**Modules/Pages to test:**

* Home page where user enters number of rooms to paint
* Dimensions page where user enters length, width and height of each room from previous step
* Results page which displays the 'amount of feet to paint' and 'gallons required' for each room. Also shows total gallons required for all rooms

**Main scenarios to cover (in scope of testing):**

1. Verification of # of rooms consistent throughout the experience:
   1. In dimensions page, need to check whether we show data input fields for number of rooms equal to what was entered by user in home page
   2. In results page, need to check whether we show results for number of rooms equal to what was entered by user in home page
2. Verification of formulas:
   1. verify if the calculation for 'feet to paint' is correct as per formula mentioned in footer
   2. verify if the calculation for 'gallons required' is correct as per formula mentioned in footer, including rounding up to higher integer
   3. verify if the calculation for total gallons required is correct
3. Verification of navigation between pages
   1. home > dimensions > results > home
4. Negative tests for invalid data entry
   1. validating that user input fields in all pages accept only valid data (no negative, non-numeric, empty or 0 for "# of rooms' or dimensions of rooms)
   2. validating that proper error messages are shown when this condition is not met and user cannot proceed to next step till corrected

**Detailed test cases:** Refer ‘Detailed test cases.xlsx’ attached



**Out of scope for testing:**

* testing with a very large number for rooms to paint : since application becomes unresponsive/slow when a large number is given, skipping this test

**Assumptions:**

* The formula given in the page footer to calculate feet & paint are assumed to be the expected & correct ones (though it is better to round up the ‘total gallons required’, rather than rounding up gallons for each room)
* As per the formula, we round up the gallons required for each room. Say, 1st room needs 2.4 gallons and 2nd room needs 1.2 gallons, then we round up each as 3 & 2 and the total gallons=5.

Instead of this, if we added the actual gallons per room (3.6) and then rounded it up, then total gallons=4 which is what users might prefer since it is cost-effective

But we will go ahead with the given formula, assuming that each room has a different paint color, type so cannot be clubbed together and rounded up

**Fixes made to the application code:**

* As per the formula in footer, surface area to paint is ((Length \* 2) + (Width \* 2)) \* Height. But this was not coded properly. Corrected the formula in ‘calculate\_feet’ method
* As per the formula in footer, 1 gallon of paint needed for painting 400 ft. But in the code, it was 350. Corrected it in ‘calculate\_gallons\_required’ method

**Automation scripts:**

* Developed using Java, Selenium Webdriver, TestNG, Maven
* Page factory model was used
* A single flow/run will cover all scenarios end-to-end across all pages
* Data provider used which would provide different inputs for each run

**System setup needed to run the test scripts:**

* Java 8
* Maven
* Chrome browser

**To change or add more inputs to test runs:**

* Go to src/test/java/com.ddtest.Paint/TestFactory.java
* Add more rows in the roomsDataProvider return object array (currently has 2 rows, 1st with 2 rooms and 2nd with 4 rooms. Make sure to enter as many room dimensions also as # of rooms)

**How to trigger test scripts:**

* Go to app root folder in command prompt
* Run mvn clean test

**How to view test report:**

* Reports are created in target/surefire-reports/index.html
* Logs are available at c:\log\PaintCalculator.log

The sections listed below are usually part of test plan, but they are not applicable or not needed for the paint calculator testing.

<References> Link to requirements, use cases, acceptance criteria etc. - typically provide links to where these are stored, like stories in jira, UX wires, API specifications, data flow models, walking skeleton for interactions between systems etc. Anything that is an input to the expected behavior of an application.

<Dependencies> - none, except that application should be running (if there are other applications/services that must be ready for our application to work, we define them here. All these dependent applications must be ready too for starting to test. If not, look into options of using stubs for testing)

<Environment to test> - NA (when there are multiple environments available, we define which one would be used to test the current features and all list other service endpoints if application interacts with external services)

<Entry criteria> - NA (generally we define that unit tests must be passed for at least basic positive flows work before code is given for testing)

<Exit criteria> - NA (generally we define that there be no outstanding issues in priority flows)

<Test Schedule> - planned start and end dates for testing to meet project deadlines. This can be split by multiple phases of testing, like

* test planning
* create, review & finalize test cases
* test data & environment setup
* test execution (could be multiple iterations, see below)

<Test cycles> - how many rounds of testing is planned and in which environment/on what branches and timelines for each.

* 1st round: feature testing done in QA environment with feature branch
* 2nd round: integration testing done in QA env with feature/integration branch
* 3rd round: end-to-end & regression testing done in staging environment with release branch
* Production verification done in prod environment once deployed (master branch)

<Resource information> - Name, role & contact details of each person working on the project across teams