New Leaf Term Project Report

Chapter 4: Framework Completion and Implementing Tests

Luke McGuire, Chris Taylor, Kasper Dugaw

College of Charleston

Fall 2020

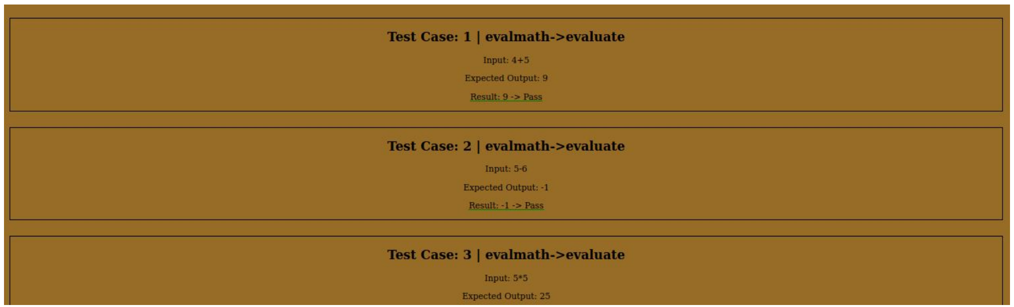
Introduction

During this chapter of our project we completed our testing framework, added functionality and styling to our generated reports, and also completed our 25 test cases.

Completing the Testing Framework

Since the last chapter we have almost entirely rewritten our runAllTests script and have made significant improvements. For one, our controller now performs quite a bit of error handling to handle cases including malformed test case definitions, missing drivers, drivers with errors or improper return values, and exceptions within the code being tested. Additionally, we have designed the controller such that only one test case is loaded at a time and that once the output is determined, it is written directly to the output file. We have also made substantial changes to output generation, which now includes a table which supports a sort option as well as scrollable components for items that are too large to fit within their space. The pass and fail colors are more clearly defined and we also have more fields describing the tests. A time stamp also appears in the upper left corner of the screen to show when the tests were run. The images below show the progress we have made between chapters 3 and 4 of our project.

Old:



Test Case: 1 | evalmath->evaluate

Input: 4+5

Expected Output: 9

Result: 9 -> Pass

Test Case: 2 | evalmath->evaluate

Input: 5-6

Expected Output: -1

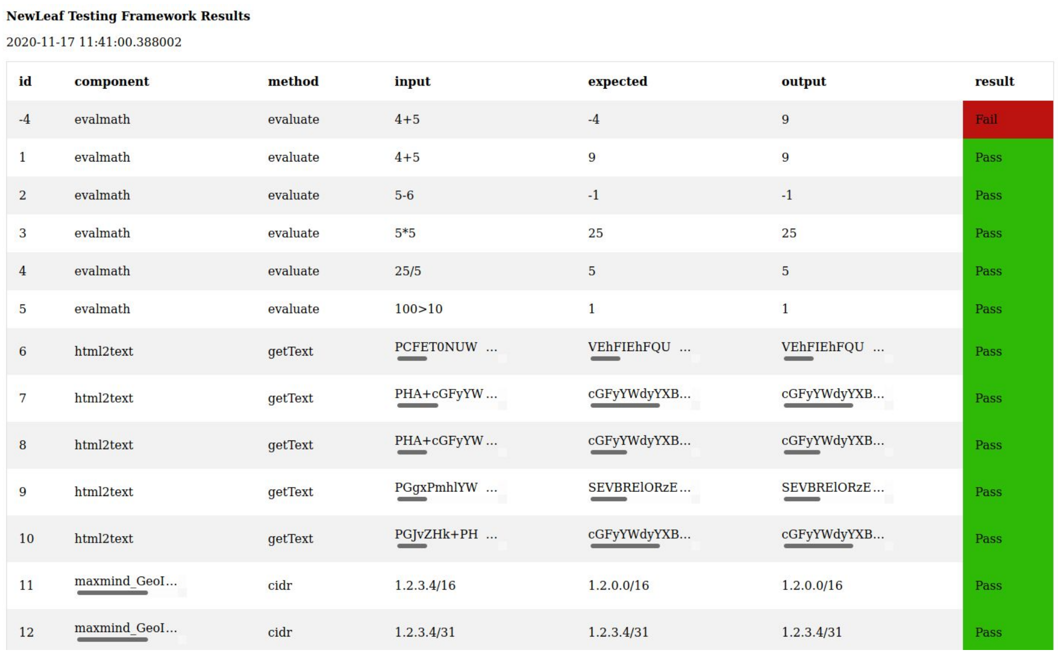
Result: -1 -> Pass

Test Case: 3 | evalmath->evaluate

Input: 5*5

Expected Output: 25

New:



NewLeaf Testing Framework Results

2020-11-17 11:41:00.388002

id	component	method	input	expected	output	result
-4	evalmath	evaluate	4+5	-4	9	Fail
1	evalmath	evaluate	4+5	9	9	Pass
2	evalmath	evaluate	5-6	-1	-1	Pass
3	evalmath	evaluate	5*5	25	25	Pass
4	evalmath	evaluate	25/5	5	5	Pass
5	evalmath	evaluate	100>10	1	1	Pass
6	html2text	getText	PCFET0NUW ...	VEhFIEhFQU ...	VEhFIEhFQU ...	Pass
7	html2text	getText	PHA+cGFyYW ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
8	html2text	getText	PHA+cGFyYW ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
9	html2text	getText	PGgxPmhlYW ...	SEVBRElORzE...	SEVBRElORzE...	Pass
10	html2text	getText	PGJvZHk+PH ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
11	maxmind_GeoI...	cidr	1.2.3.4/16	1.2.0.0/16	1.2.0.0/16	Pass
12	maxmind_GeoI...	cidr	1.2.3.4/31	1.2.3.4/31	1.2.3.4/31	Pass

Test Cases

At this point in time our team has completed 5/5 test drivers and 25/25 test cases. The drivers that we have implemented will be discussed in the following subsections.

evalMath_evaluate.driver.php

In the evaluate method inside of the EvalMath class, we are testing different mathematical operations that are inputted as a string. The method takes in a mathematical expression in the form of a string, parses it, calculates, and returns the answer the method comes up with. We then check to see if the returned value is the same as the expected value and either pass or fail the test.

evalMath_mod.driver.php

The modulo evalmath driver evaluates the module of the two numbers it is given. An example is if the user passes in a 7,7 the output should be 0 since 7 % 7 is 0. Modulo is calculated by getting the remainder of dividing the first input by the second. If the second number is larger than the first, the remainder will always be the first number since it is not divisible by the second number. It is impossible to take any number % 0 so when we tried that it returned an undefined value.

html2text_getText.driver.php

This driver performs tests on a method which parses text from html. For instance, the input "<p>An html paragraph</p>" would return the output "An html paragraph". In order to pass the input and output from the driver in a safer and more compressed manner the driver takes input and provides data as base64 strings. Our test cases include html with multiple tags, nested tags, and combinations of both.

htmlpurifier_UnitConverter_getSigFigs.driver.php

This driver performs tests on a method that computes the number of significant figures in a string which contains a decimal number. The method is part of a class which provides unit conversion implementations. The driver accepts a string as input, for example "01.23" and provides the integer count of significant digits, in this case 3, as output. Our test cases cover various scenarios including cases with and without leading zeros and cases with all zeros.

maxmind_Geolp2_Util_cidr.driver.php

This driver performs tests on a method that converts internet protocol addresses (IP) and their prefixes into Classless Inter-Domain Routing (CIDR) notation. This method is part of a class which provides IP address geolocation services. The driver accepts strings in the format "IP Address/Prefix" and passes the input to the method which returns a string in a similar format. For example, using the IP 192.168.86.42 with the prefix 16 would produce the output "192.168.0.0/16". The method also accepts IPv6 addresses. Our test cases for this method examine a variety of scenarios including IPv4 and IPv6 addresses and both maximal and minimal prefixes.