Milestone 5- Testing Plan

Group: Sophia AbiEzzi, Tiffany Phan, Jay Hayward, Andy Stieber, Jackson Boynton, Shelby Keupp

Project Title: bitw1se

User-Acceptance Test Case Plans:

*note: Our outputs will always return the decimal value of the completed operations, but we also present the binary representations on our website.

Feature	Test Case	User Activity
Testing Decimal Cases	*Decimal test cases* We're testing to see if our functions correctly compute an answer using the corresponding operations on the user's decimal input. 2|3= 3 4&10=0 5^9=12 ~-1=0 3<<4=48 5>>1=2 16>>>3=2	The user will type the test expression into the textbox and hit enter. If the correct decimal answer is displayed on the screen with leading zeros to the nearest byte, then the test has passed. For the decimal cases, we tested if our program returned the correct result.
Testing Hexadecimal Cases	*Hexadecimal test cases* We're testing to see if our functions correctly compute an answer using the corresponding operations on the user's hexadecimal input. 0x15|0x3=23 0x4&0x17=4 0x5^0x18=29 ~0x16= -23 0x3<<0x4=48 0x5>>0x1=2 0x16>>>0x3=2	The user will type in hexadecimal expressions to test if the corresponding binary operators produce the correct output. After the user finishes typing, the correct output should be shown and the test should pass. For the hexadecimal cases, we tested if our program returned the correct result and if the test case is has only one numerical user input.

Testing Binary Cases	*Binary test cases* We're testing to see if our functions correctly compute an answer using the corresponding operations on the user's decimal input. 0b11|0b1010=11 0b1010&0b1111=10 0b1101^0b11=14 ~0b1100=-13 0b1001<<0b11=72 0b1001>>0b11=1	The user will type in binary expressions to test if the corresponding binary operators produce the correct output. After the user finishes typing, the correct output should be shown and the test should pass. For the binary test cases, we checked if our functions returned the correct solution and if it detected if the user input was in binary.

Automated Tests:

https://mochajs.org

We used Mocha, which is a JavaScript testing framework, to test the member functions of our class, such as: if we're getting the correct output result of our bitwise operations, if the user input is valid or not, if the correct bitwise operator is grabbed from our list of binary operators, etc.

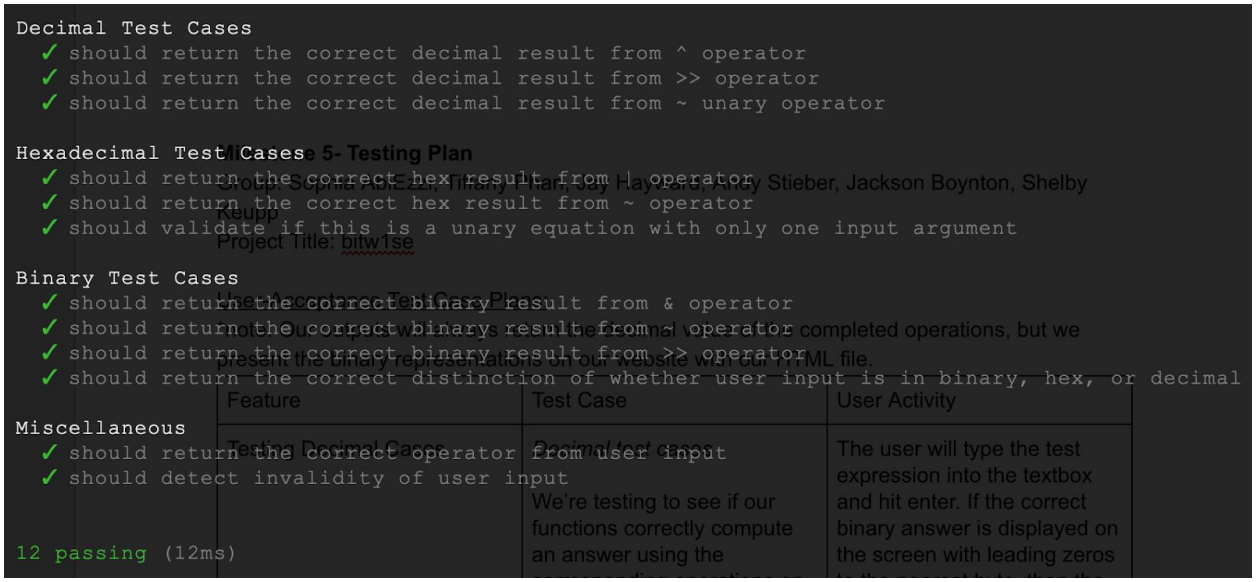
To run the automated test cases with Mocha, we made a "test" directory to run mocha in and contains all of our test cases in a JavaScript file. We installed Mocha through npm. To run, all that is needed to be typed in the terminal is "mocha" and it will run all of the test cases.

The main file that we're testing is our "parse.js" file, which is a file that takes in user input, and recognizes if we're dealing with a hexadecimal, decimal, or binary expression, and then returns the correct solution. Because we used a class, we needed to export all of the class functions to our test.js file for testing using "module.exports.EquationObj = EquationObj" at the bottom of the parse.js file. We were then able to create instances of this class in our test.js file.

The screenshot below is of our test.js file with all of our test cases decimal, binary, and hexadecimal cases.

```
var assert = require('assert');
let app = require('../public/parse_unit_test.js');


describe('Decimal Test Cases', function() {

  // Decimal Test Cases
  it('should return the correct decimal result from ^ operator', function() {
    var equationObj = new app.EquationObj("5^9");
        assert.equal(equationObj.getResult(), 12);
    });
  it('should return the correct decimal result from >> operator', function() {
    var equationObj = new app.EquationObj("5>>1");
        assert.equal(equationObj.getResult(), 2);
    });
  it('should return the correct decimal result from ~ unary operator', function() {
    var equationObj = new app.EquationObj("~~1");
        assert.equal(equationObj.getResult(), 0);
    });
});

describe('Hexadecimal Test Cases', function() {
  // Hexadecimal Test Cases
  it('should return the correct hex result from | operator', function() {
    var equationObj = new app.EquationObj("0x15|0x3");
        assert.equal(equationObj.getResult(), 23);
    });
  it('should return the correct hex result from ~ operator', function() {
    var equationObj = new app.EquationObj("~0x16");
        assert.equal(equationObj.findUnarySolution(), -23);
    });
  it('should validate if this is a unary equation with only one input argument', function() {
    var equationObj = new app.EquationObj("~0x16");
        assert.equal(equationObj.isUnaryEquation(), true);
    });
});

describe('Binary Test Cases', function() {
  // Binary Test Cases
  it('should return the correct binary result from & operator', function() {
    var equationObj = new app.EquationObj("b1101&b11");
        assert.equal(equationObj.getResult(), 9);
    });
  it('should return the correct binary result from ~ operator', function() {
    var equationObj = new app.EquationObj("~b0101");
        assert.equal(equationObj.getResult(), -102);
    });
  it('should return the correct binary result from >> operator', function() {
    var equationObj = new app.EquationObj("b1001>>b11");
        assert.equal(equationObj.findBinarySolution(), 0);
    });
  it('should return the correct distinction of whether user input is in binary, hex, or decimal', function() {
        assert.equal(app.EquationObj.getBase("0b1001"), 'b');
    });
});
```

```
  describe('Miscellaneous', function() {
    it('should return the correct operator from user input', function() {
        assert.equal(app.EquationObj.getOperator("|", ['&&', '||', '<<', '>>>', '^', '&', '|', '>>']), '|');
      });
    it('should detect invalidity of user input', function() {
        var equationObj = new app.EquationObj("like a hundred bees");
        assert.equal(equationObj.isValid(), false);
      });
  });
```



The screenshot above is of all of our test cases running and passing all of the tests once we ran “mocha” in our terminal.