

Experiment 04 - Version Control and Testing Lab

December 22, 2022

Aim

Write Unit test cases and carry-out Functional Testing


Theory

- **Unit Testing** : It is a way of testing a unit - the smallest piece of code that can be logically isolated in a system.
- **Functional Testing** : Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered. The unit testing is a ***subset*** of the functional testing and performing multiple such test simulations the basis of this practical.

Results

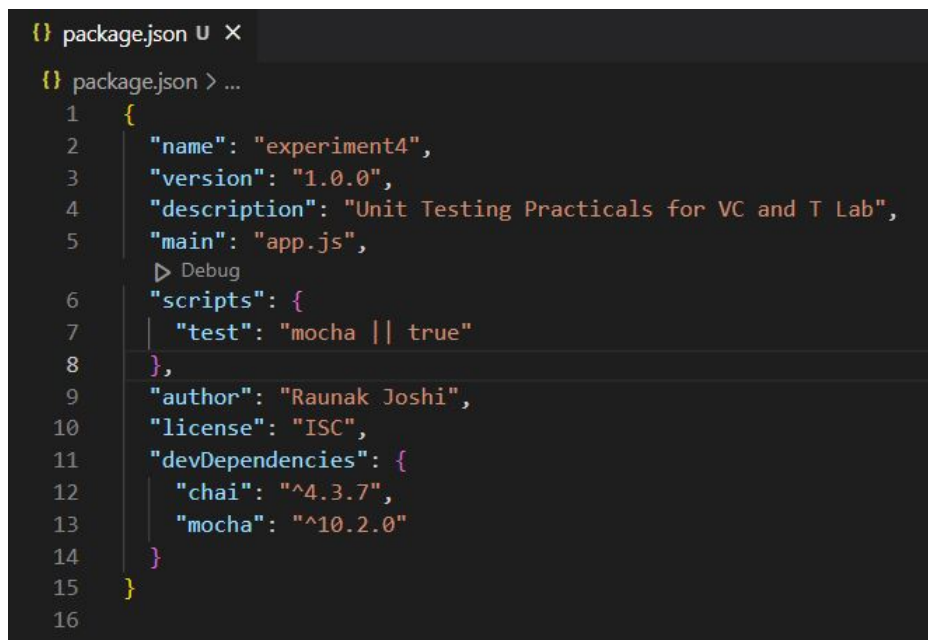
A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is selected and has a white underline. Below the tabs, the terminal shows the command prompt 'PS C:\Users\Admin\Desktop\lab1SS\experiment4>' followed by the command 'npm init' in yellow text. A white cursor is positioned at the end of the command.

Figure 1: The program starts with installation of node and using the command to initiate the project



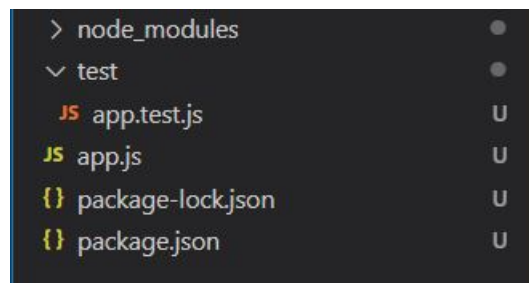
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Admin\Desktop\lab1SS\experiment4> npm install mocha chai --save-dev
```

Figure 2: The command in the image is used to install the mocha and chai for unit testing



```
{ } package.json U X
{ } package.json > ...
1  {
2    "name": "experiment4",
3    "version": "1.0.0",
4    "description": "Unit Testing Practicals for VC and T Lab",
5    "main": "app.js",
6    "scripts": {
7      "test": "mocha || true"
8    },
9    "author": "Raunak Joshi",
10   "license": "ISC",
11   "devDependencies": {
12     "chai": "^4.3.7",
13     "mocha": "^10.2.0"
14   }
15 }
16
```

Figure 3: After installation the package file



```
> node_modules
v test
JS app.test.js U
JS app.js U
{ } package-lock.json U
{ } package.json U
```

Figure 4: The folder and file structure

```
{ } package.json U JS app.js U X
JS app.js > [?] <unknown> > isPrime
1  module.exports = {
2  ...
3    isPrime : function (num) {
4      if (num <= 1) {
5        return false;
6      }
7      if (num % 2 == 0 && num > 2) {
8        return false;
9      }
10
11     const s = Math.sqrt(num)
12
13     for(let i = 3; i <= s; i += 2) {
14       if(num % i === 0) {
15         return false;
16       }
17     }
18     return true;
19   }
20 }
```

Figure 5: Basic Prime Number program used for testing

```
{ } package.json U JS app.js U X
JS app.js > [?] <unknown> > isPrime
1  module.exports = {
2  ...
3    isPrime : function (num) {
4      // if (num <= 1) {
5      //   return false;
6      // }
7
8      if (num % 2 == 0 && num > 2) {
9        return false;
10      }
11
12     const s = Math.sqrt(num)
13
14     for(let i = 3; i <= s; i += 2) {
15       if(num % i === 0) {
16         return false;
17       }
18     }
19     return true;
20 }
```

Figure 6: Commenting the base case of prime number being less than 1 to test it

```
{ } package.json U JS app.js U JS app.test.js U X
test > JS app.test.js > describe('isPrimeFunction') callback
1  const assert = require('chai').assert;
2  const isPrime = require('../app').isPrime;
3  const number = 2
4
5  describe('isPrimeFunction', () => {
6    it('Should ensure the base condition that number lies above 1', () => {
7      let result = isPrime(number);
8      assert.equal(result, number > 1);
9    })
10 })
```

Figure 7: Unit Test program for ensuring the number is above 1

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Admin\Desktop\lab1SS\experiment4> npm run test
> experiment4@1.0.0 test C:\Users\Admin\Desktop\lab1SS\experiment4
> mocha || true

isPrimeFunction
  ✓ Should ensure the base condition that number lies above 1

1 passing (3ms)
PS C:\Users\Admin\Desktop\lab1SS\experiment4> █
```

Figure 8: Successful execution of the test

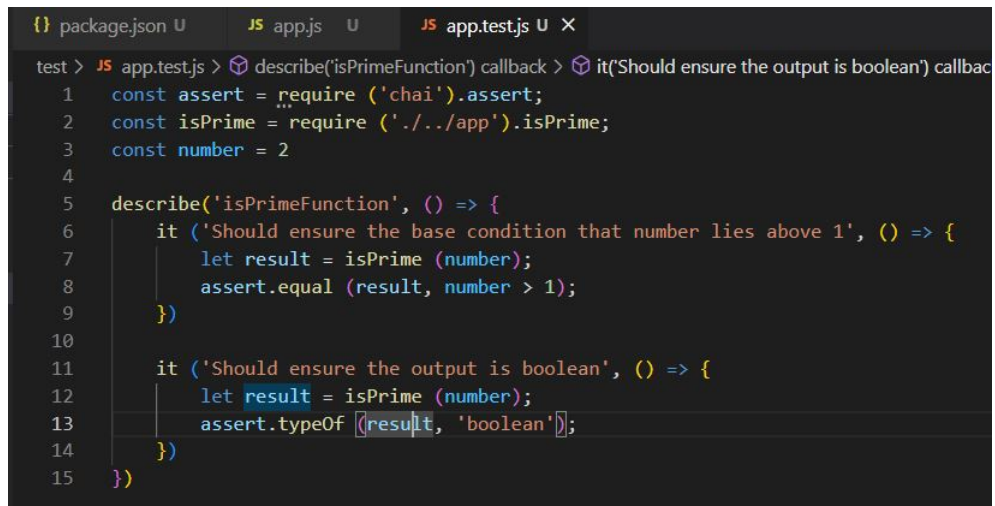
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Admin\Desktop\lab1SS\experiment4> npm run test
> experiment4@1.0.0 test C:\Users\Admin\Desktop\lab1SS\experiment4
> mocha || true

isPrimeFunction
  1) Should ensure the base condition that number lies above 1

0 passing (5ms)
1 failing

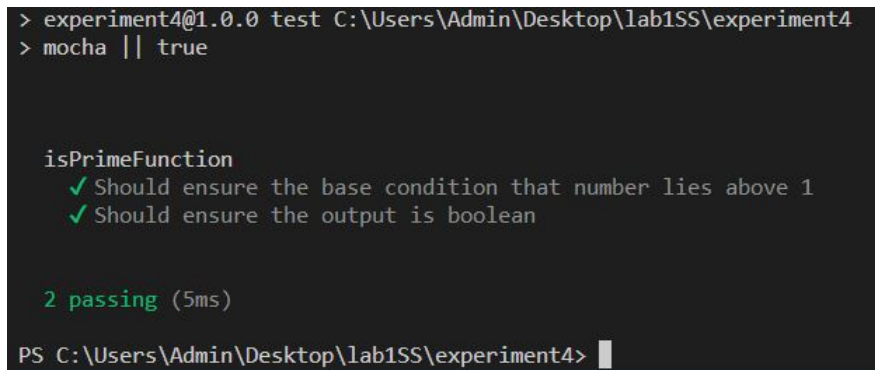
1) isPrimeFunction
  Should ensure the base condition that number lies above 1:
    AssertionError: expected true to equal false
      + expected - actual
```

Figure 9: Failing the test case for checking the output



```
test > JS app.test.js > describe('isPrimeFunction') callback > it('Should ensure the output is boolean') callback
1  const assert = require ('chai').assert;
2  const isPrime = require ('../app').isPrime;
3  const number = 2
4
5  describe('isPrimeFunction', () => {
6      it ('Should ensure the base condition that number lies above 1', () => {
7          let result = isPrime (number);
8          assert.equal (result, number > 1);
9      })
10
11     it ('Should ensure the output is boolean', () => {
12         let result = isPrime (number);
13         assert.typeOf (result, 'boolean');
14     })
15 })
```

Figure 10: Writing one more test case for ensuring the output is always boolean



```
> experiment4@1.0.0 test C:\Users\Admin\Desktop\lab1SS\experiment4
> mocha || true

isPrimeFunction
  ✓ Should ensure the base condition that number lies above 1
  ✓ Should ensure the output is boolean

2 passing (5ms)

PS C:\Users\Admin\Desktop\lab1SS\experiment4>
```

Figure 11: Successful execution of the boolean test case

Conclusion

In this experiment, deep implementation of the unit test is performed successfully. Stacking the unit cases is also done and an appropriate execution of the functional testing is performed.