Ashlyn Martin

avm102

AVMartin14

Discussion of each function and associated tests

I have created 5 functions which are:

- The **bmi** function takes in the users inputted height in inches and feet along with their weight and calculates their body mass index. I used the formula from the link under the notes and resources tab in the instructions to calculate the body mass index. This function also returns the users calculated body mass index rounded to the nearest decimal place.
- The other 4 functions are functions I created to calculate and determine each category the user's body mass index falls into.
    - The **underweight** function returns true if the user's body mass index is below 18.5, otherwise it returns false.
    - The **normalweight** function returns true if the user's body mass index falls within 18.5 to 24.9, otherwise it returns false.
    - The **overweight** function returns true if the user's body mass index falls within 25 to 29.9, otherwise it returns false.
    - The **obese** function returns true if the user's body mass index is higher than or equal to 30, otherwise it returns false.

I have constructed 3 test cases for the bmi function and 2 test cases for each of the four category functions, thus I have 11 test cases in all.

- Test cases for the **bmi** function:

1. *test('BMI for 6ft and 150lbs is 20.8', ( ) => {*

    expect(bmi(6, 0, 150)).toBe('20.8');

    });

- This test case proves that the user can enter their height in inches as 0 and it still outputs the correct body mass index.

2. *test('BMI for 4ft 13in and 115lbs is 22.3', ( ) => {*

   expect(bmi(4, 13, 115)).toBe('22.3');

   });

3. *test('BMI for 5ft 1in and 115lbs is 22.3', ( ) => {*

   expect(bmi(5, 1, 115)).toBe('22.3');

   });

- I chose to construct these two test cases to prove that if the user enters inches that are greater than 12 inches (aka 1 foot) then the body mass index can still be properly calculated. In this case, 4ft and 13 inches is equivalent to 5ft and 1 inch.
- These test cases also prove that the body mass index is always returned with just one decimal place.

- Test cases for the **underweight** function:

1. *test('U – BMI(17) is less than 18.5 should return true', ( ) => {*

   expect(underweight(17)).toBe(true);

   });

- This test case shows that the correct inputted body mass index value falls within the correct range with it being smaller than 18.5 and will return true.

2. *test('U – BMI(19) is more than 18.5 should return false, ( ) => {*

   expect(underweight(19)).toBe(false);

   });

- This test case shows that the incorrect inputted body mass index value that does not fall within the correct range in that it is bigger than 18.5 and will return false.

- Test cases for the **normalweight** function:

1. *test('N – BMI(22) is between 18.5 and 24.9 should return true', ( ) => {*

   expect(normalweight(22)).toBe(true);

   });

   - This test case shows that the correct inputted body mass index value falls within the correct range with it being between 18.5 and 24.9 and will return true.

2. *test('N – BMI(18.4) is not between 18.5 and 24.9 should return false', ( ) => {*

   expect(normalweight(18.4)).toBe(false);

   });

   - This test case shows that the incorrect inputted body mass index value does not fall within the correct range in that it is between 18.5 and 24.9 and will return false.

- Test cases for the **overweight** function:

1. *test('O – BMI(27) is between 25 and 29.9 should return true', ( ) => {*

   expect(overweight(27)).toBe(true);

   });

   - This test case shows that the correct inputted body mass index value falls within the correct range with it being between 25 and 29.9 and will return true.

2. *test('O – BMI(30) is not between 25 and 29.9 should return false', ( ) => {*

   expect(overweight(30)).toBe(false);

   });

- This test case shows that the incorrect inputted body mass index value does not fall within the correct range in that it is between 25 and 29.9 and will return false.

- Test cases for the **obese** function:

1. *test('Ob – BMI(35) is greater than or equal to 30 should return true', ( ) => {*

   expect(obese(35)).toBe(true);

   });

   - This test case shows that the correct inputted body mass index value falls within the correct range with it being bigger than or equivalent to 30 and will return true.

2. *test('Ob – BMI(29.9) is less than 30 should return false', ( ) => {*

   expect(obese(29.9)).toBe(false);

   });

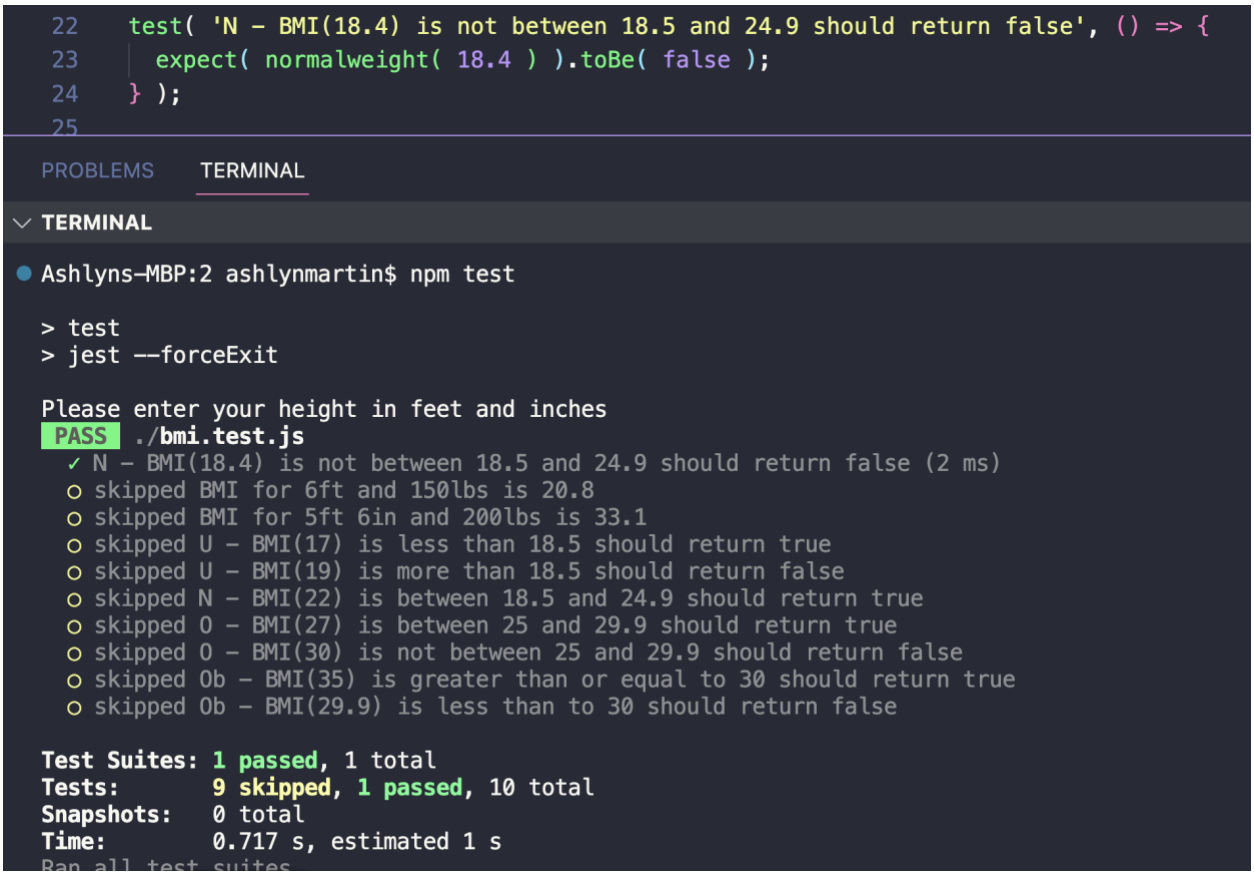
   - This test case shows that the incorrect inputted body mass index value does not fall within the correct range with it being smaller than 30 and will return false.

Boundary testing technique used to choose test cases

I believe that I am using the Weak N x 1 boundary technique. My test cases prove that I am not using the EPC boundary technique because as shown below when I use a boundary shift of 0.1 on the lower boundary of the normal weight category, my function is able to catch that the shift is in fact false as it should be. If I was using an EPC approach, then my test case would return true because it would then be unable to catch this slight boundary shift. I am not using the 1 x 1 boundary testing technique because with 1 x 1 one boundary value is chosen for each input parameter whereas my test cases are inputted with a single boundary value at a time, thus inferring the Weak N x 1 approach.

Inducing boundary shift by 0.1 at the lower boundary of “Normal Weight”

I induced a boundary shift by 0.1 at the lower boundary of the normal weight category by passing in the body mass index value of 18.4 as seen below in the screenshot.



Discussion of test case catching this boundary shift problem

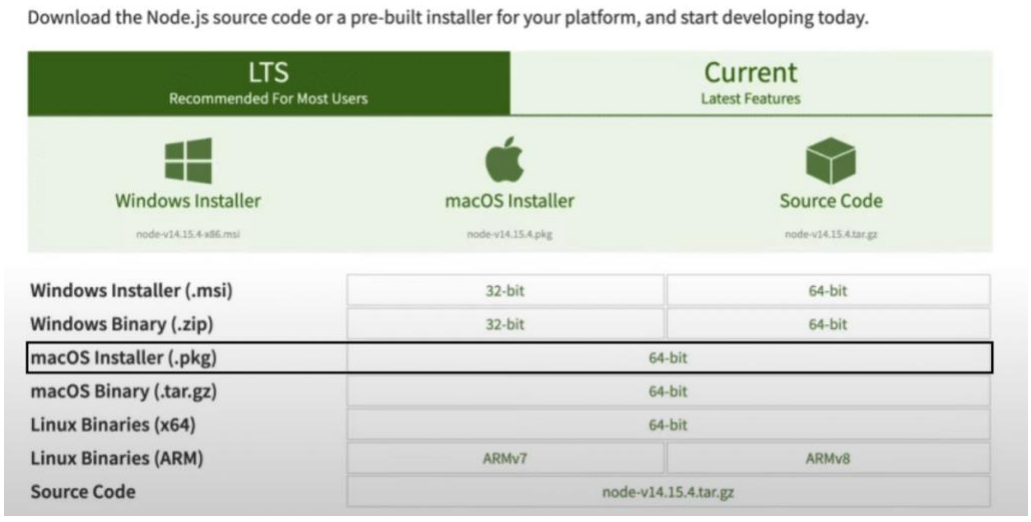
Thankfully, my test case did catch this boundary shift problem and returned false because in my if statement I specifically created 18.5 to be a catch integer. Since I am rounding the body mass index to one decimal place, my normalweight function can in fact catch that the body mass index of 18.4 to be false. Also, by creating functions for each of the categories it is easier to catch the boundary shifts because the functions are just returning true or false to the main function which actually displays the category name based off of the categories functions.

Detailed setup and execution instructions

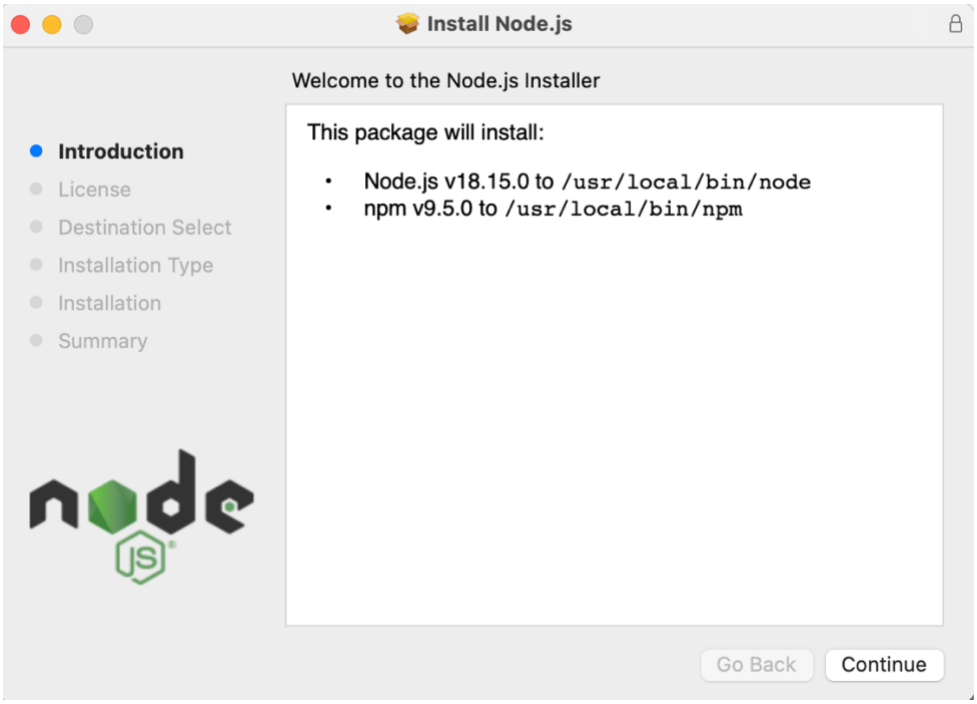
I used Visual Studio Code with a Mac OS to construct and execute my application and test cases, thus I will explain how I did it and then I will include the instructions of how to presumably do it on Windows from the Internet. I used JavaScript as my code and Jest as my framework.

To download the files I used for this assignment, navigate to my GitHub using this link: https://github.com/AVMartin14/STQA-Assignmentt-2.git.

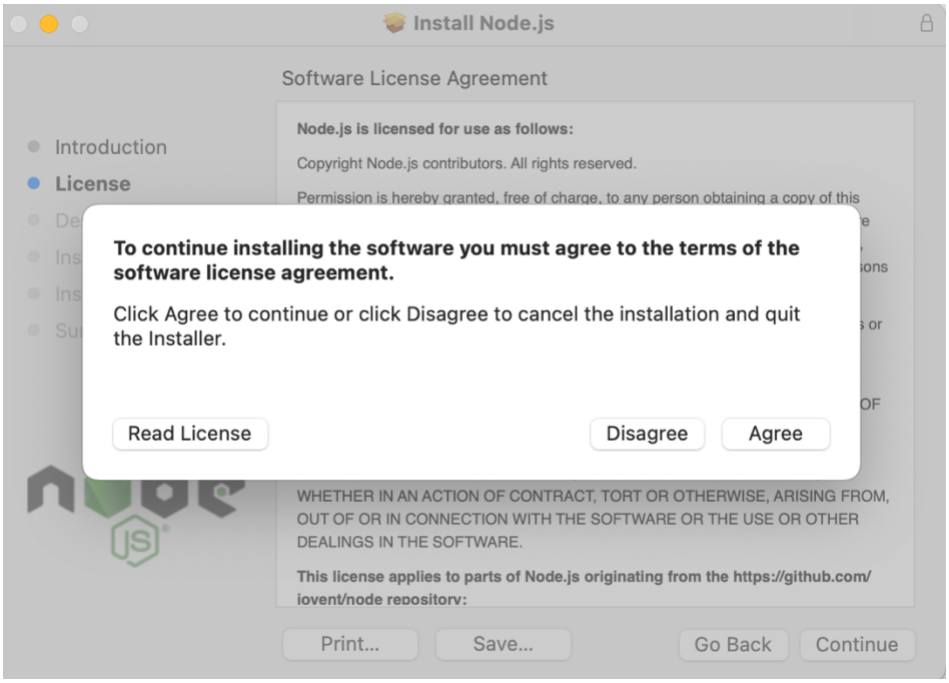
- MacOS:
    - After downloading my files from GitHub to be able to execute the bmi.js file in the command line, node must be installed on your computer. I used this website(https://kinsta.com/blog/how-to-install-node-js/) as guidance in this process; however, I am listing the steps below:
        1. I went to this website(https://nodejs.org/en/download) and downloaded the macOSInstaller (.pkg) 64-bit.



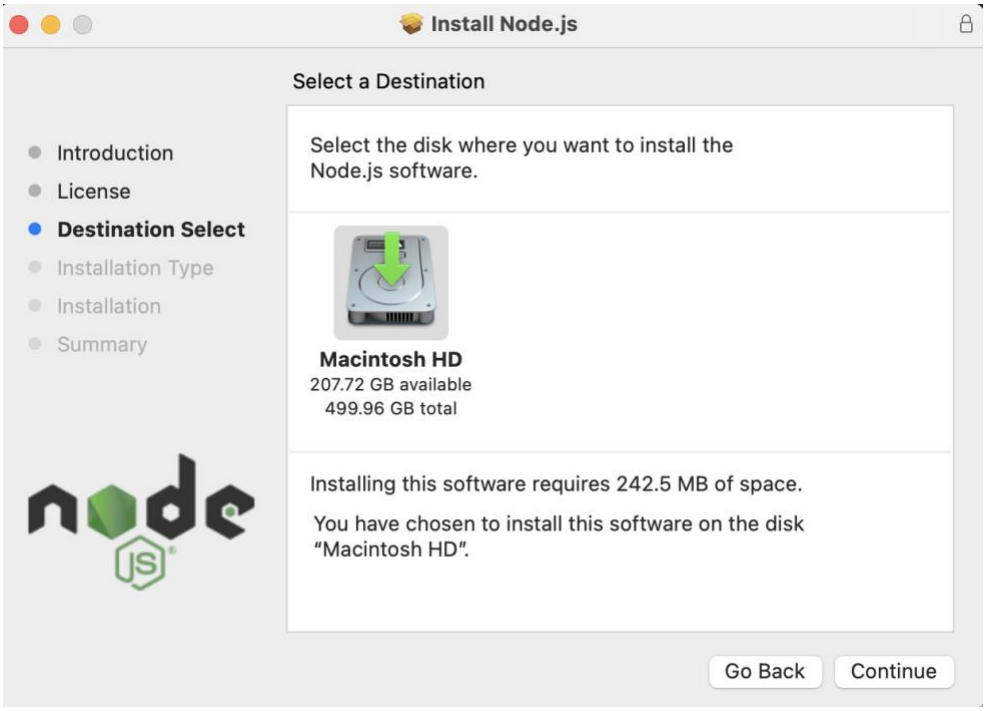
        2. After the installer package was downloaded to my laptop, I went to my Downloads and clicked on the package. The installation window popped up and I began the process of installing it by clicking Continue.



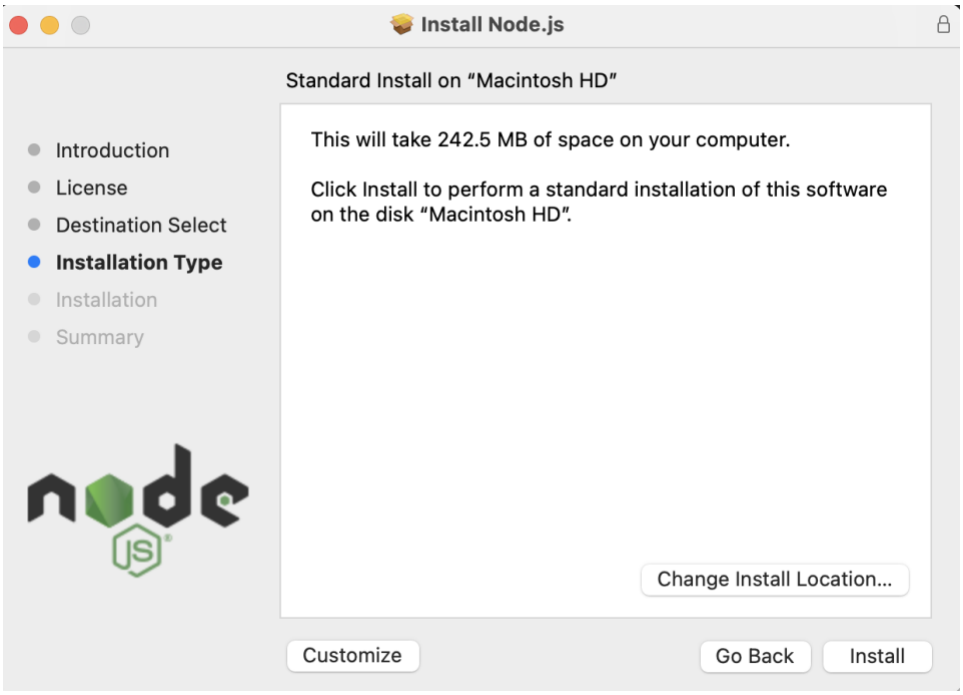
3. I hit Continue on the License process and it pops up an alert to where you have to hit Agree and then it will move to the next step.



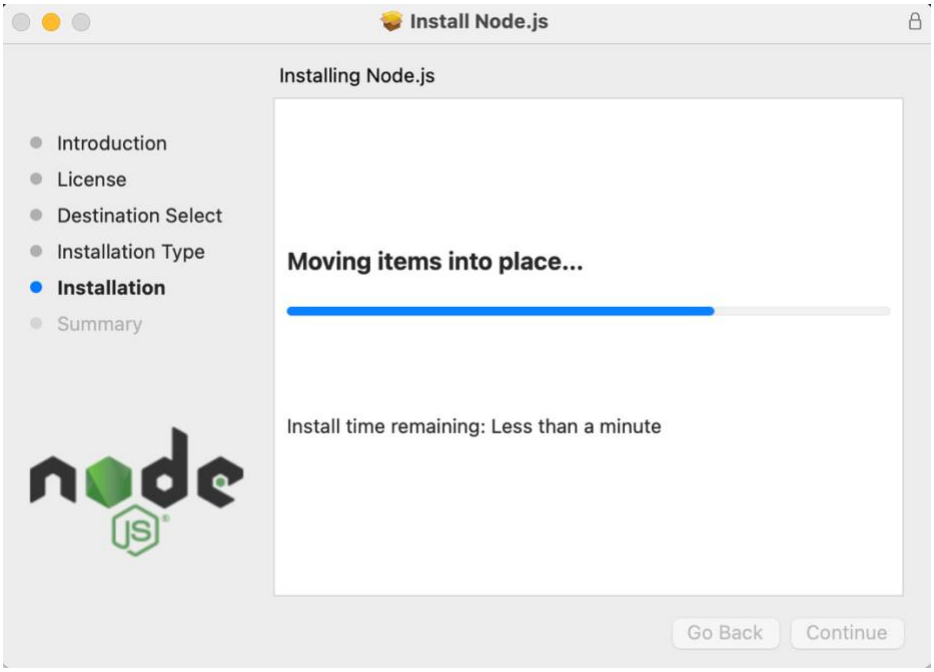
4. I just hit Continue on the next step for the Destination Select.



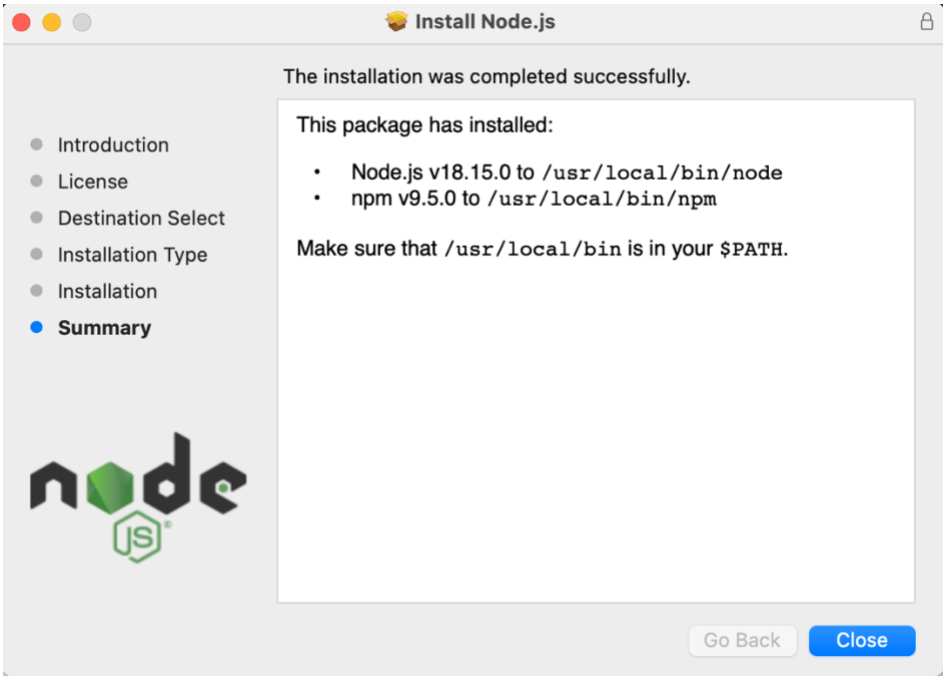
5. I hit Install on the Installation Type page.



6. A popup came up to use my password to verify the install and began installing it.



7. Once installed, the summary page popped up and I hit Close.



8. To verify that node had been installed I went to the terminal and inputted "Node --version" and "npm --version".

```
Ashlyns-MacBook-Pro:~ ashlynmartin$ Node --version
v18.15.0
Ashlyns-MacBook-Pro:~ ashlynmartin$ npm --version
9.5.0
```

9. Now you will be able to run the bmi.js file through the terminal (which I did in Visual Studio Code) by inputting “node bmi.js” and input your height in feet and inches along with your height to receive your BMI and category.

```
Ashlyns-MBP:2 ashlynmartin$ node bmi.js
Please enter your height in feet and inches
   feet: 5
   inches: 4
Please enter your weight in pounds: 130
BMI: 22.9
Category: Normal weight
Ashlyns-MBP:2 ashlynmartin$
```

- To run the test cases, you must install Jest. You can do this by following these steps:

1. In the terminal, input “npm install –save-dev jest” and the output should end up like the image below.

```
Ashlyns-MBP:2 ashlynmartin$ npm install --save-dev jest

added 1 package, and audited 281 packages in 2s

31 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

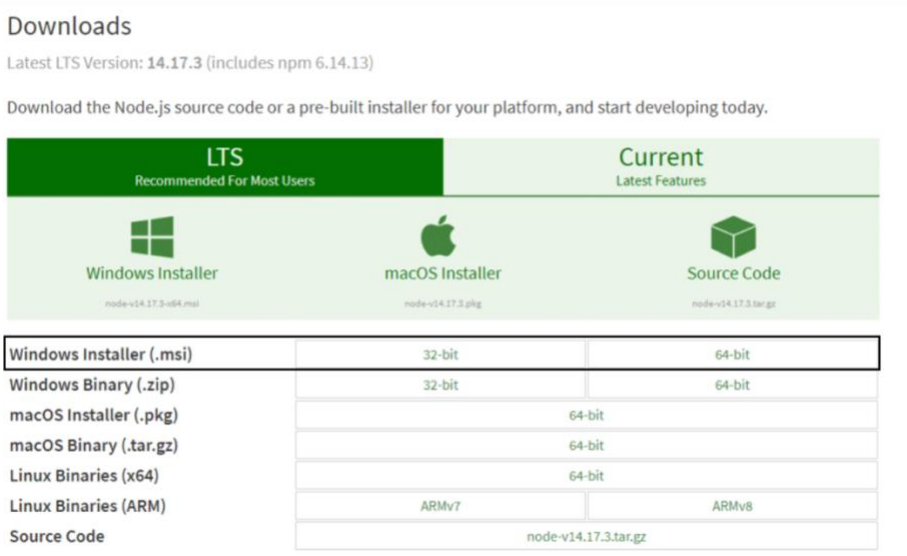
2. You can now run the bmi.test.js file by inputting “npm test bmi.test.js” into the terminal and get the following output.

```
Ashlyns-MBP:2 ashlynmartin$ npm test bmi.test.js

> test
> jest --forceExit bmi.test.js

Please enter your height in feet and inches
 PASS  ./bmi.test.js
  ✓ BMI for 6ft and 150lbs is 20.8 (3 ms)
  ✓ BMI for 4ft 13in and 115lbs is 22.3
  ✓ BMI for 5ft 1in and 115lbs is 22.3
  ✓ U - BMI(17) is less than 18.5 should return true
  ✓ U - BMI(19) is more than 18.5 should return false (1 ms)
  ✓ N - BMI(22) is between 18.5 and 24.9 should return true (1 ms)
  ✓ N - BMI(18.4) is not between 18.5 and 24.9 should return false (1 ms)
  ✓ O - BMI(27) is between 25 and 29.9 should return true (3 ms)
  ✓ O - BMI(30) is not between 25 and 29.9 should return false
  ✓ Ob - BMI(35) is greater than or equal to 30 should return true
  ✓ Ob - BMI(29.9) is less than 30 should return false

Test Suites: 1 passed, 1 total
Tests:       11 passed, 11 total
Snapshots:   0 total
Time:        0.629 s, estimated 1 s
Ran all test suites matching /bmi.test.js/i.
Force exiting Jest: Have you considered using `--detectOpenHandles` to detect asy
nc operations that kept running after all tests finished?
```

3. When I ran the test file for the first time, Visual Studio Code automatically added a package-lock.json and package.json file to my workspace. I left the package-lock.json alone; however, I had to change the package.json to look like the image below because of the command line being able to accept user input jest did not want to end the testing, so I had to make it exit after the test case had ran.

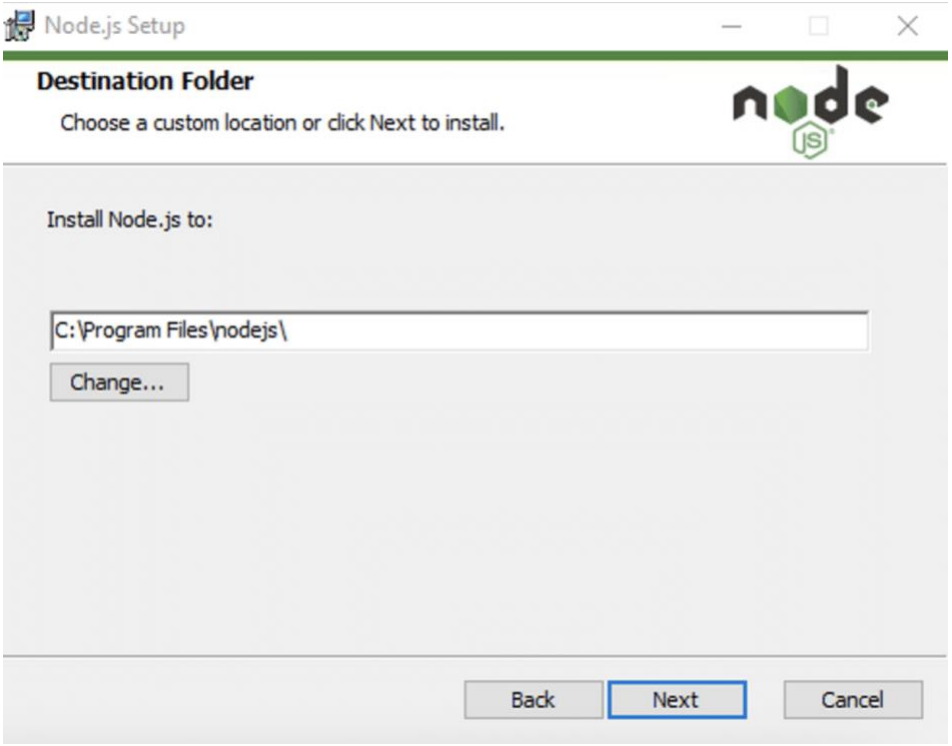
```
{} package.json > ...
 1  {
 2    "main": "bmi.js",
        ▷ Debug
 3    "scripts": {
 4      "test": "jest --forceExit"
 5    },
 6    "dependencies": {
 7      "readline": "^1.3.0"
 8    },
 9    "devDependencies": {
10      "jest": "^29.5.0"
11    }
12  }
```

- Windows:
    - After downloading the files from my GitHub like shown above, you will have to make sure node and jest are installed on your windows.
    - According to the website I provided above, these are the steps you will have to follow to install node.
        1. Download the Windows Installer (.msi). If you are using a 64-bit operating system, download the 64-bit version. If you are using a 32-bit operating system, download the 32-bit version.

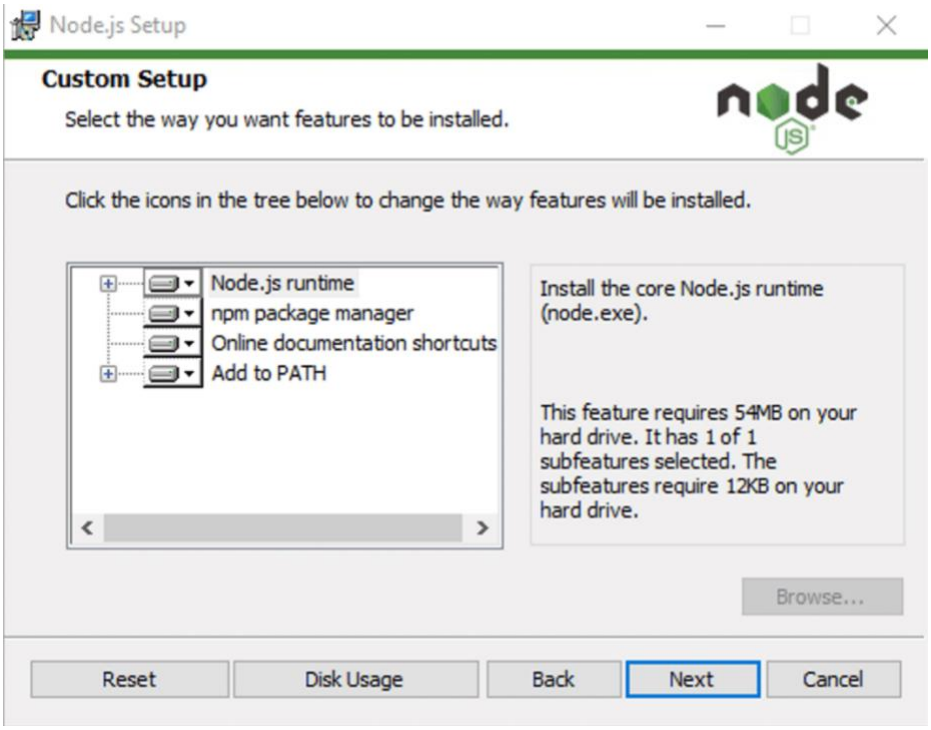


        2. Double-click on the installer file and run it. The installer will ask you to accept the Node.js license agreement. To move forward, check the “I accept” box and click Next:

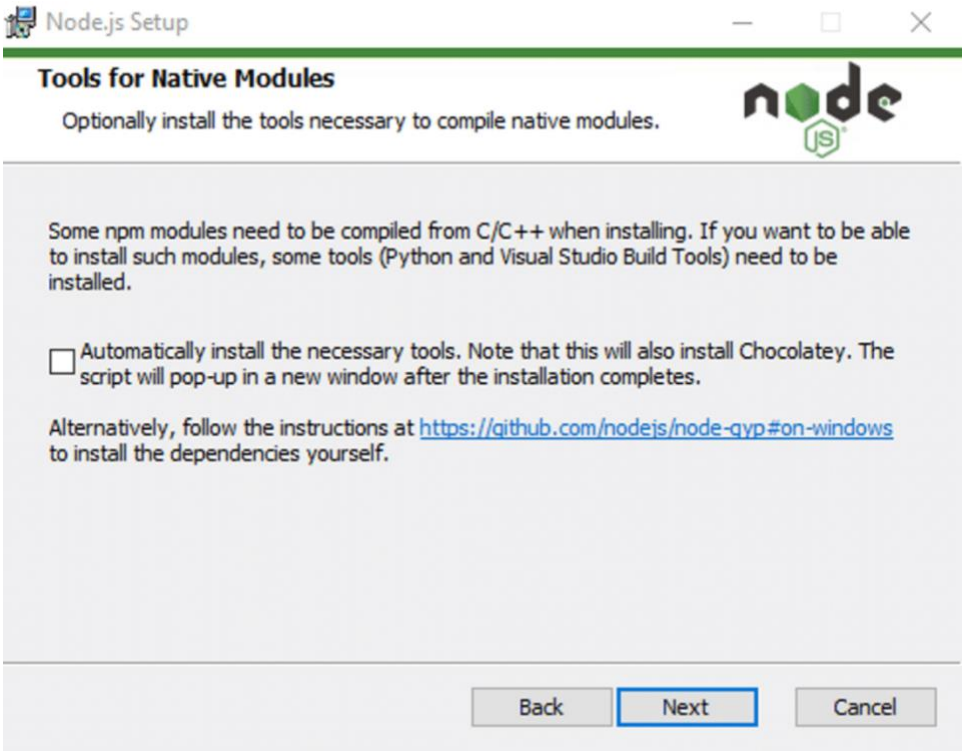
3. Then, select the destination where you want to install Node.js. If you don't want to change the directory, go with the Windows default location and click the Next button again.



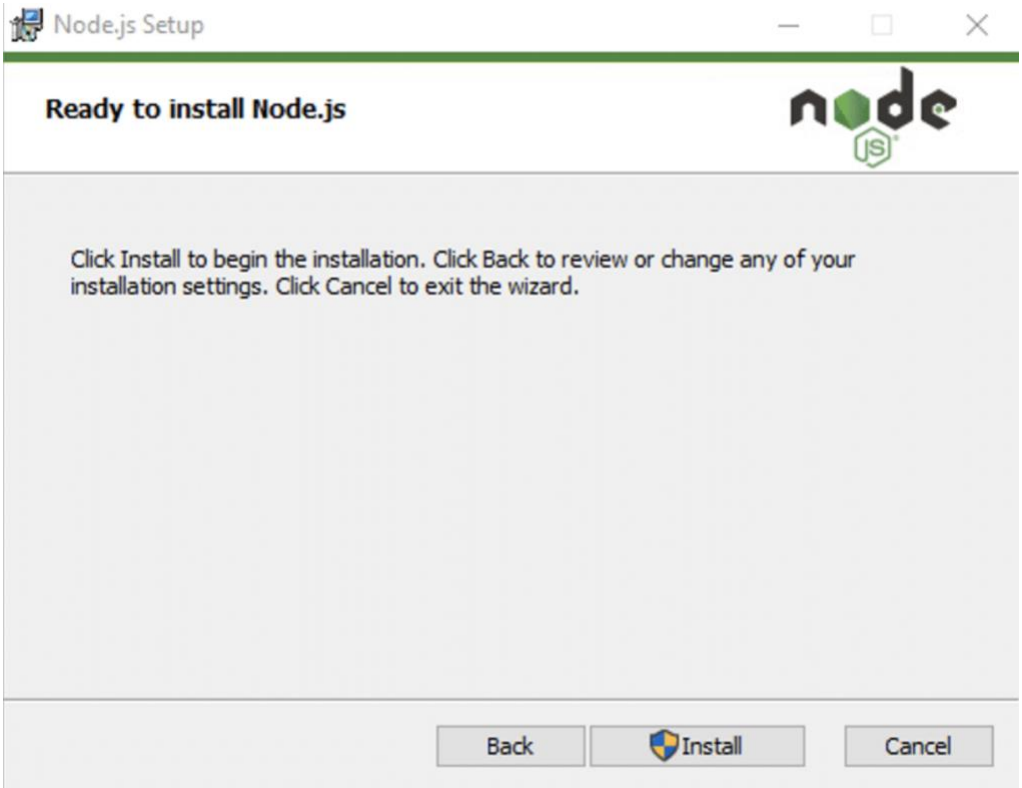
4. The next screen will show you custom setup options. If you want a standard installation with the Node.js default features, click the Next button. Otherwise, you can select your specific elements from the icons in the tree before clicking Next:



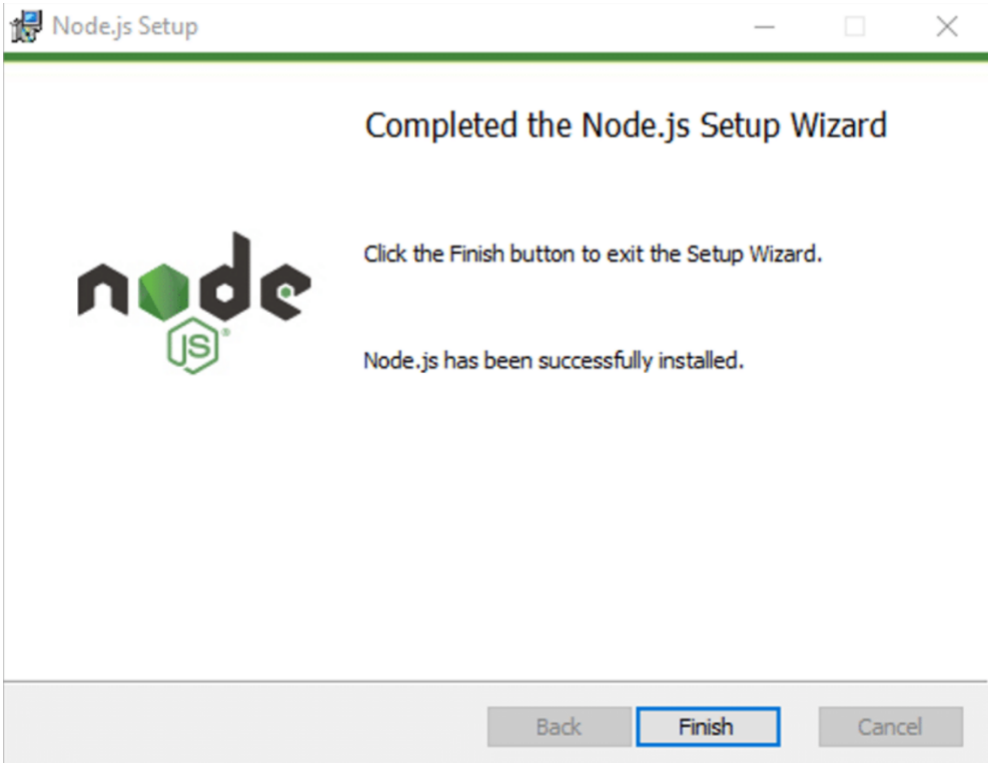
5. Node.js offers you options to install tools for native modules. If you're interested in these, click the checkbox to mark your preferences, or click Next to move forward with the default:



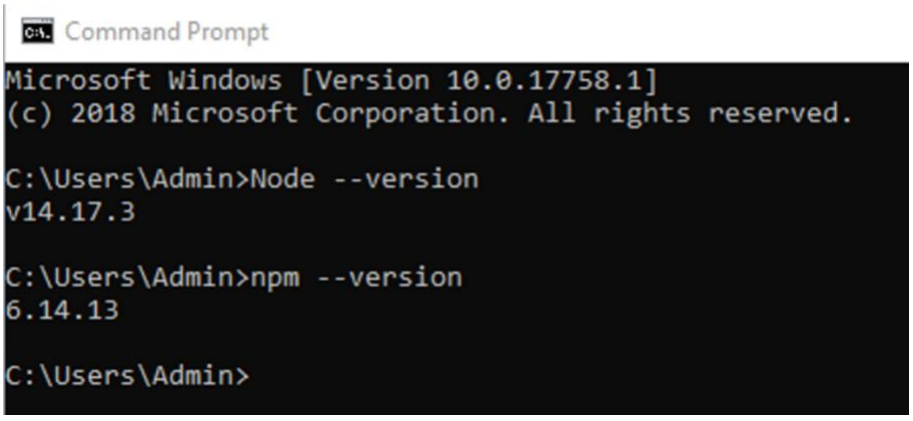
6. Click the install button to begin the installation process:



7. The system will complete the installation within a few seconds or minutes and show you a success message. Click on the Finish button to close the Node.js installer:



8. To verify the installation just like MacOS, you can input “Node --version” and “npm --version” into the command prompt:

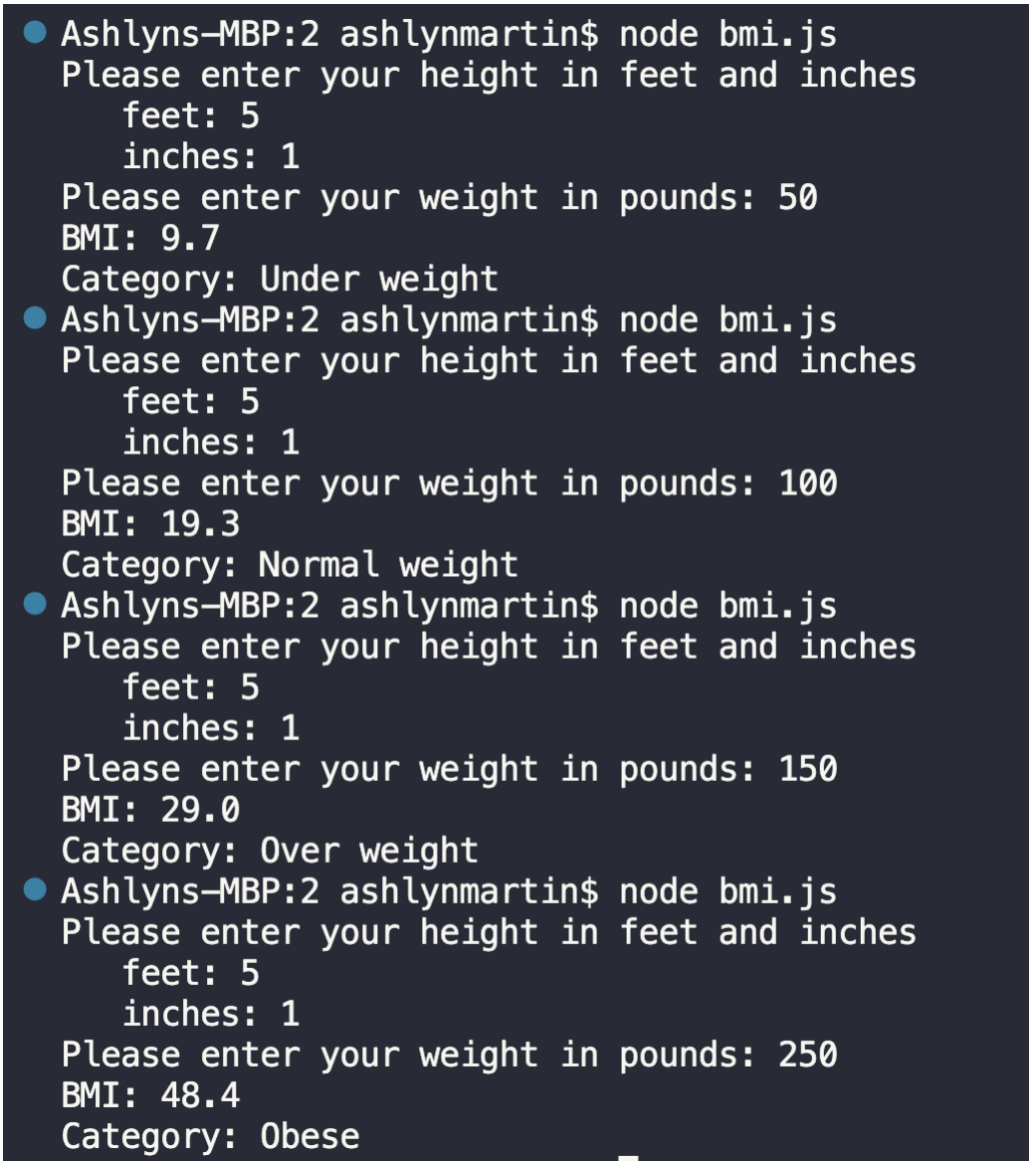


9. To execute the bmi.js file, do the same thing as I listed for MacOS and input “node bmi.js” into the command prompt.

- You install and execute Jest just like I stated to do with MacOS by inputting “npm install –save-dev jest” into the command prompt.

Application returning the correct output for all four BMI categories

The following two screenshots below prove that when the user inputs values into the command line interface, the correct body mass index will be displayed along with the correct category and it does in fact cover all four of the category functions. By running the test cases, I have also proven that it does in fact output correctly all four of the categories.



```
 PASS  ./bmi.test.js
   ✓ BMI for 6ft and 150lbs is 20.8 (2 ms)
   ✓ BMI for 5ft 6in and 200lbs is 33.1
   ✓ U - BMI(17) is less than 18.5 should return true (1 ms)
   ✓ U - BMI(19) is more than 18.5 should return false
   ✓ N - BMI(22) is between 18.5 and 24.9 should return true (1 ms)
   ✓ N - BMI(18.4) is not between 18.5 and 24.9 should return false
   ✓ O - BMI(27) is between 25 and 29.9 should return true (1 ms)
   ✓ O - BMI(30) is not between 25 and 29.9 should return false
   ✓ Ob - BMI(35) is greater than or equal to 30 should return true
   ✓ Ob - BMI(29.9) is less than to 30 should return false

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.663 s, estimated 1 s
Ran all test suites.
```