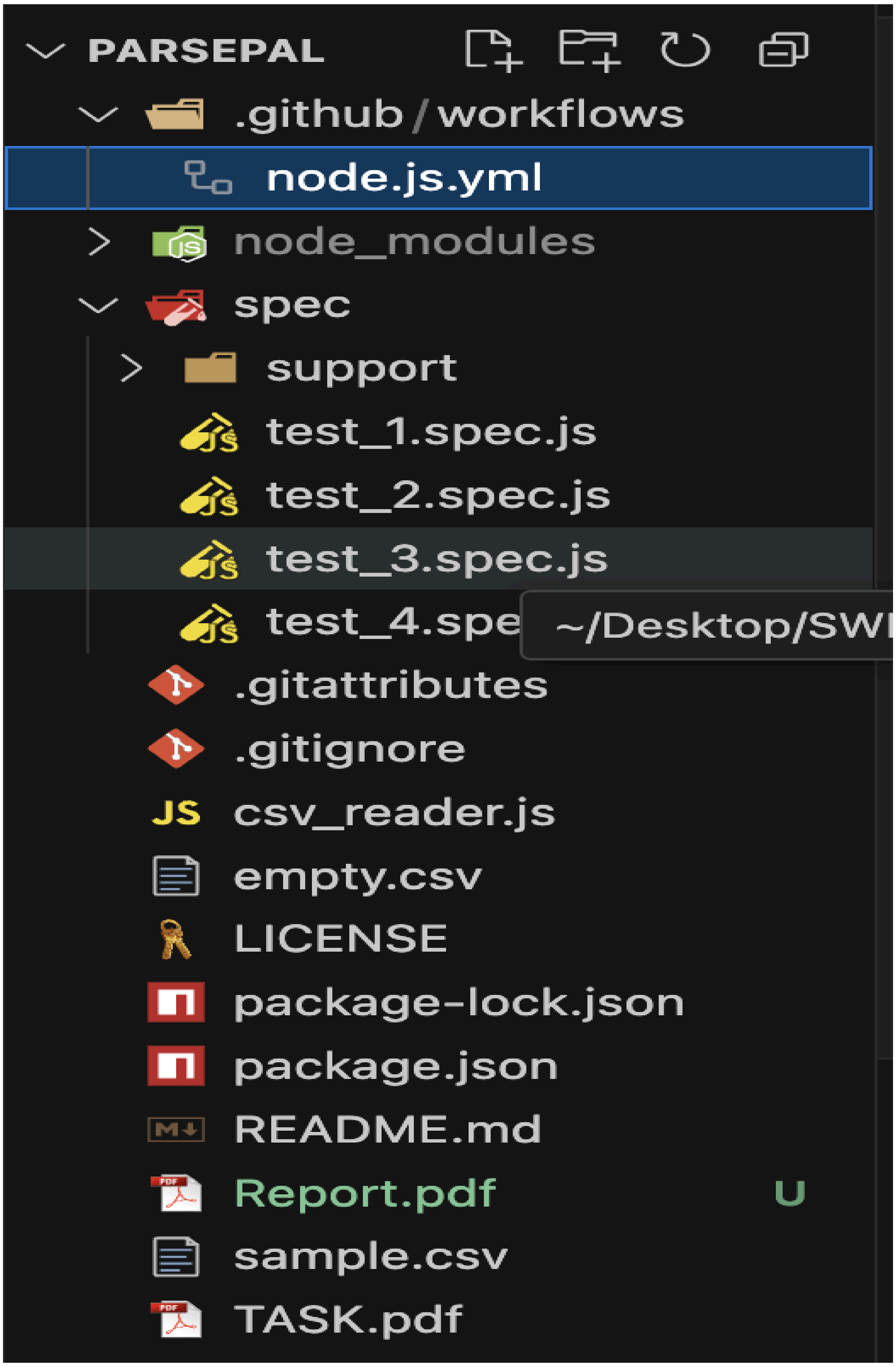
Name: Kelvin Ihezue **Net ID: ki120**

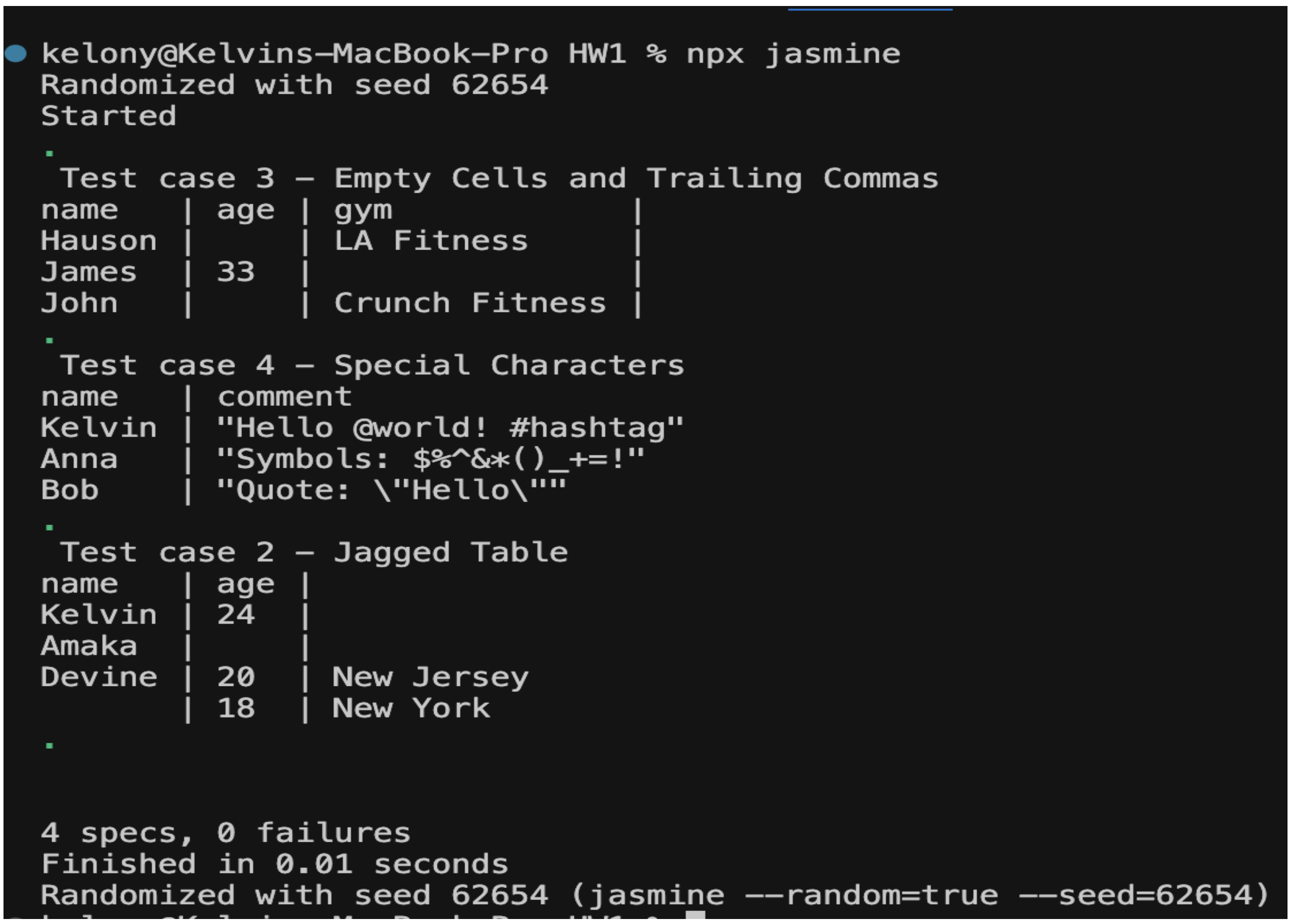
PROGRAM REPORT

File organization



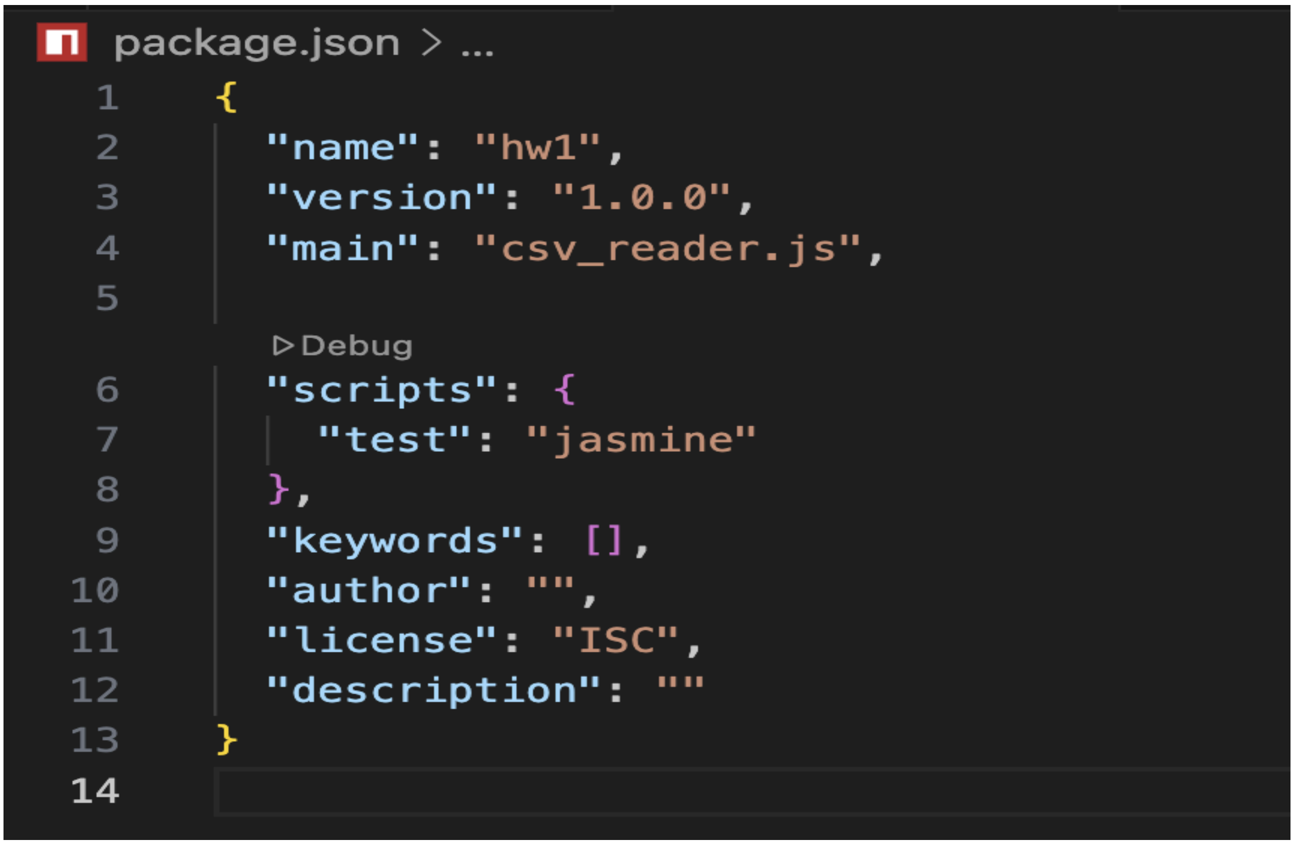
```
PARSEPAL
  .github / workflows
    node.js.yml
  node_modules
  spec
    support
    test_1.spec.js
    test_2.spec.js
    test_3.spec.js
    test_4.spe   ~/Desktop/SWI
  .gitattributes
  .gitignore
  csv_reader.js
  empty.csv
  LICENSE
  package-lock.json
  package.json
  README.md
  Report.pdf                         U
  sample.csv
  TASK.pdf
```

When I run `npx jasmine`, it displays the internal representation of the spreadsheet as it is processed against the different test cases I created:



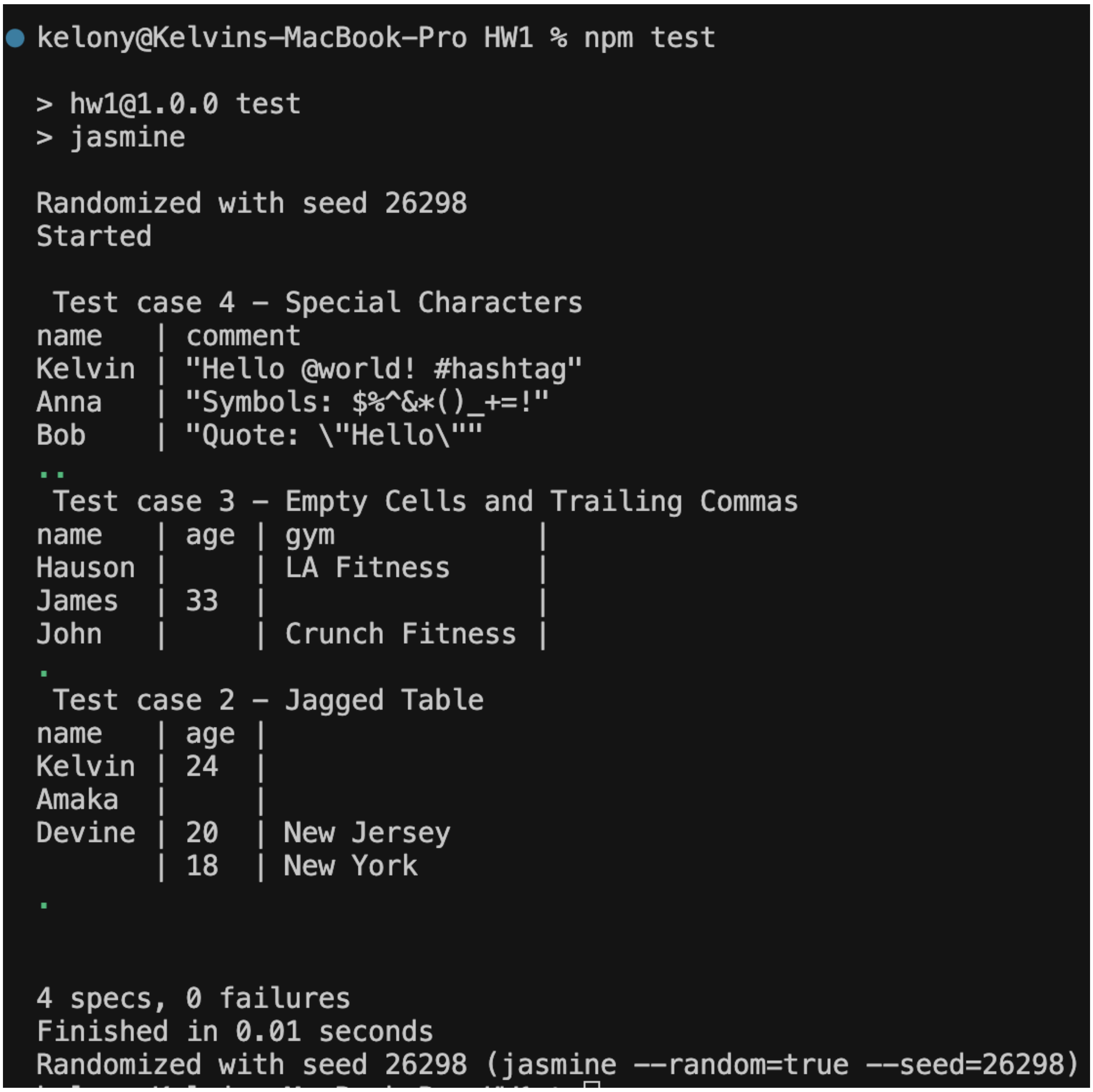
```
kelony@Kelvins-MacBook-Pro HW1 % npx jasmine
Randomized with seed 62654
Started
.
 Test case 3 - Empty Cells and Trailing Commas
name    | age | gym              |
Hauson  |     | LA Fitness       |
James   | 33  |                  |
John    |     | Crunch Fitness |
.
 Test case 4 - Special Characters
name    | comment
Kelvin | "Hello @world! #hashtag"
Anna    | "Symbols: $%^&*()_+=!"
Bob     | "Quote: \"Hello\""
.
 Test case 2 - Jagged Table
name    | age |
Kelvin | 24   |
Amaka   |      |
Devine | 20   | New Jersey
        | 18   | New York
.


4 specs, 0 failures
Finished in 0.01 seconds
Randomized with seed 62654 (jasmine --random=true --seed=62654)
```

Alternatively, I can run `npm test` to get the same output, since I added a `package.json` file with Jasmine defined under the `test` script.



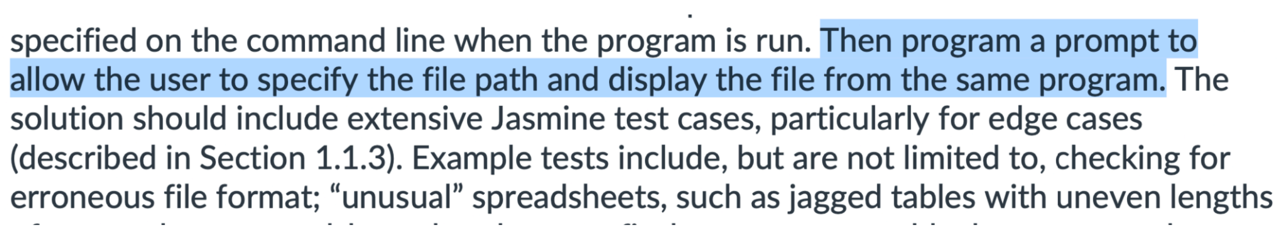
package.json > ...

```json
{
  "name": "hw1",
  "version": "1.0.0",
  "main": "csv_reader.js",

  ▷ Debug
  "scripts": {
    "test": "jasmine"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

```



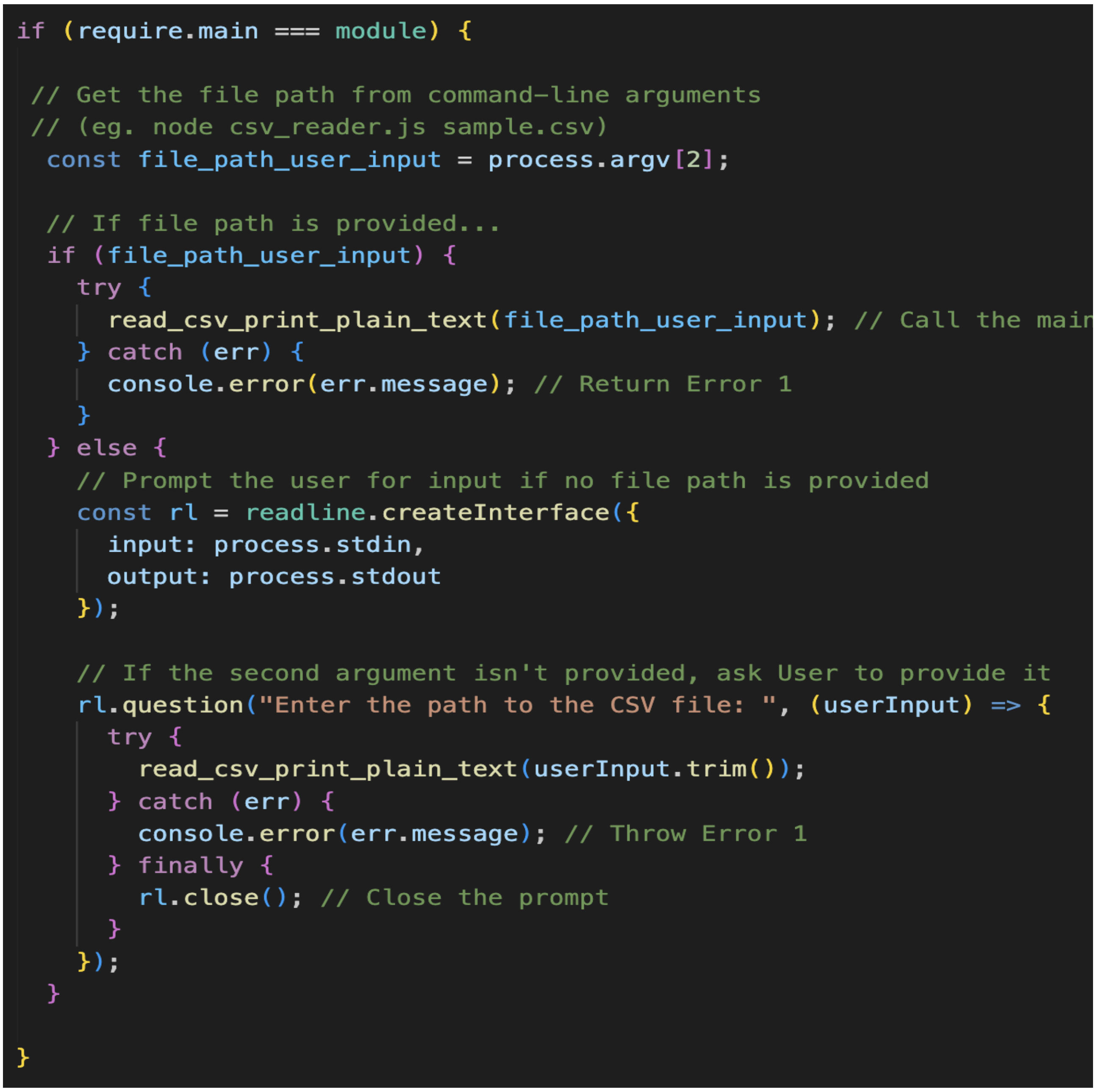
```
● kelony@Kelvins-MacBook-Pro HW1 % npm test

  > hw1@1.0.0 test
  > jasmine

  Randomized with seed 26298
  Started

   Test case 4 - Special Characters
  name   | comment
  Kelvin | "Hello @world! #hashtag"
  Anna   | "Symbols: $%^&*()_+=!"
  Bob    | "Quote: \"Hello\""
  ..
   Test case 3 - Empty Cells and Trailing Commas
  name   | age | gym            |
  Hauson |     | LA Fitness     |
  James  | 33  |                |
  John   |     | Crunch Fitness |
  .
   Test case 2 - Jagged Table
  name   | age |
  Kelvin | 24  |
  Amaka  |     |
  Devine | 20  | New Jersey
         | 18  | New York
  .


  4 specs, 0 failures
  Finished in 0.01 seconds
  Randomized with seed 26298 (jasmine --random=true --seed=26298)
```

Prompt implementation that allows the user to specify the file path:



specified on the command line when the program is run. Then program a prompt to allow the user to specify the file path and display the file from the same program. The solution should include extensive Jasmine test cases, particularly for edge cases (described in Section 1.1.3). Example tests include, but are not limited to, checking for erroneous file format; "unusual" spreadsheets, such as jagged tables with uneven lengths

Code implementation on csv_reader.js



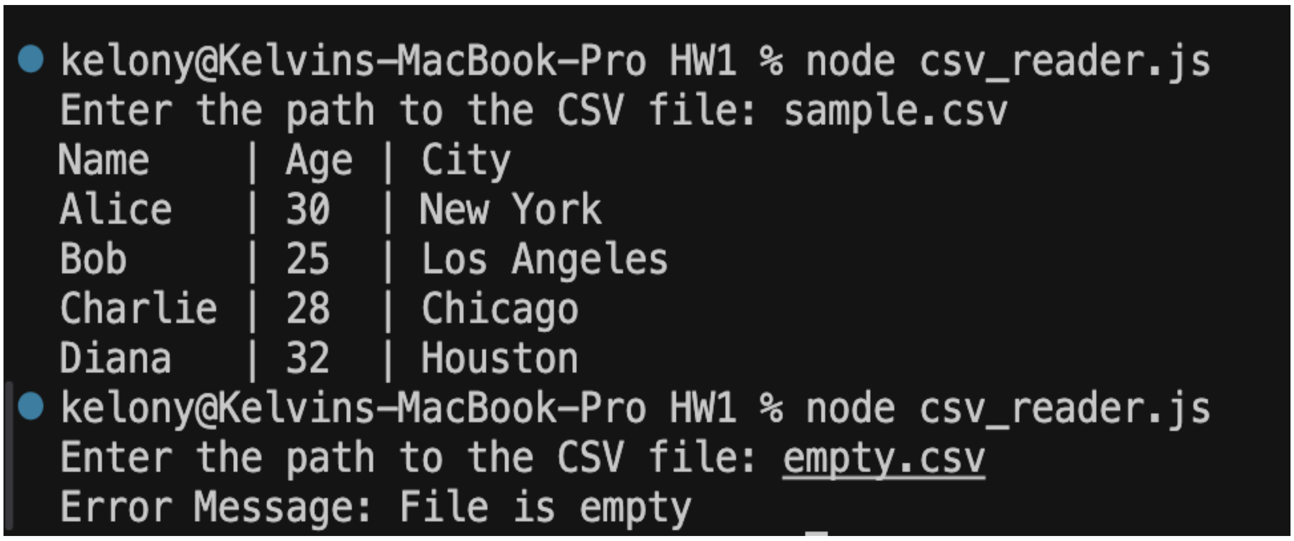
```
if (require.main === module) {

 // Get the file path from command-line arguments
 // (eg. node csv_reader.js sample.csv)
  const file_path_user_input = process.argv[2];

  // If file path is provided...
  if (file_path_user_input) {
    try {
      read_csv_print_plain_text(file_path_user_input); // Call the mai
    } catch (err) {
      console.error(err.message); // Return Error 1
    }
  } else {
    // Prompt the user for input if no file path is provided
    const rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout
    });

    // If the second argument isn't provided, ask User to provide it
    rl.question("Enter the path to the CSV file: ", (userInput) => {
      try {
        read_csv_print_plain_text(userInput.trim());
      } catch (err) {
        console.error(err.message); // Throw Error 1
      } finally {
        rl.close(); // Close the prompt
      }
    });
  }

}
```

Which leads to my final implementation:

I adopted an edge case handling in my main program (`csv_reader.js`) that prompts the user to provide a specific file path. The implementation includes:

- Checking if the file exists or is empty → if it's empty, an appropriate error message is thrown.

- Reading and displaying the full contents of the CSV file → to verify correct and expected output.



```
● kelony@Kelvins-MacBook-Pro HW1 % node csv_reader.js
  Enter the path to the CSV file: sample.csv
  Name    | Age | City
  Alice   | 30  | New York
  Bob     | 25  | Los Angeles
  Charlie | 28  | Chicago
  Diana   | 32  | Houston
● kelony@Kelvins-MacBook-Pro HW1 % node csv_reader.js
  Enter the path to the CSV file: empty.csv
  Error Message: File is empty
```