LAB 2

Test-Driven Development (TDD) & Behavior-Driven Development (BDD)

Objective

The aim of this lab was to learn and apply the concepts of Test-Driven Development (TDD) and Behavior-Driven Development (BDD) using Node.js.

Tools Used

- **Node.js** — JavaScript runtime for development
- **Mocha + Chai** — for TDD (unit testing)
- **Jasmine** — for BDD (behavior testing)
- **VS Code / Terminal** — for development and running tests

What We Learned

Test-Driven Development (TDD)

- Write a failing test first
- Write the minimal code to pass the test
- Refactor the code while keeping tests passing

Behavior-Driven Development (BDD)

- Focuses on describing the system's behavior in natural language
- Tests scenarios from the user's perspective using tools like Jasmine

Steps Followed

A. TDD using Mocha & Chai

1. Initialize Node project and install packages:

```
npm init -y
npm install mocha chai --save-dev
```

2. calculator.js (implementation file):

```
function add(a, b) {
 return a + b;
}
module.exports = { add };
```

3. test/calculator.test.js (test file):

```
const { expect } = require('chai');
const { add } = require('../calculator');

describe('Calculator - TDD', () => {
 it('should return 7 for 4 + 3', () => {
  expect(add(4, 3)).to.equal(7);
 });
});
```

4. Run the test:

```
npx mocha
```

B. BDD using Jasmine

1. Install Jasmine and initialize:

```
npm install --save-dev jasmine
npx jasmine init
```

2. calculator.js (same as before):

```
function add(a, b) {
 return a + b;
}
module.exports = add;
```

3. spec/calculatorSpec.js:

```
const add = require('../calculator');

describe('Calculator - BDD', () => {
 it('adds 4 and 3 to return 7', () => {
  expect(add(4, 3)).toBe(7);
 });
});
```

4. Run Jasmine tests:

```
npx jasmine
```

Conclusion

This lab clarified the importance of writing tests early. TDD helped us build code step by step with confidence, while BDD emphasized creating tests that reflect how users expect the system to behave.