Unit Testing Tool : Mocha

Author: Akila Nipo

Overview

Mocha is a widely-used JavaScript test framework that runs on Node.js and in the browser. It allows developers to write and execute tests for their applications efficiently, offering a flexible and extensible environment that supports various testing styles and methodologies.

Key Features of Mocha

- Organized Test Structure
- Asynchronous Testing
- Customizable Assertions
- Browser Compatibility
- Reporting Options
- Lifecycle Hooks

➞ For more, explore Mocha's features here

Mocha Vs Other Unit Testing Tools

📌 Mocha vs. Jest

-

Feature	Mocha	Jest
Flexibility & Customization	Modular design allows using various custom libraries (Chai, Sinon) for assertions and mocking, providing greater flexibility.	More opinionated, with built-in tools for assertions, mocking, etc.
Integration with Other Tools	Easily integrates with a wide range of third-party tools and libraries, making it ideal for diverse setups.	Less flexible; optimized for its own ecosystem and tools.
Large Projects (Test Coverage)	Greater configurability for handling large and complex projects, allowing tailored test setups.	Simpler, but might lack granular control for large-scale test setups.
Parallelism & Process Control	Fine-grained control over test execution order and environment setup, making it suitable for complex workflows.	Runs tests in parallel by default; less flexible for customized test execution.
Legacy Projects & Ecosystem	Better suited for legacy projects with older or non-standard codebases, ensuring compatibility.	More modern and suited for newer projects, but less adaptable to legacy setups.
Browser Testing	Easily supports testing in real browser environments, allowing for comprehensive integration tests.	Typically focused on Node.js, with browser simulation via JSDOM.
Reporters & Output Customization	Offers a large variety of customizable reporters for test results, providing detailed feedback.	Fewer built-in options for customizing test output.
Non-Node.js Environments	Works well in both Node.js and browser-based environments, making it versatile for various applications.	Optimized for Node.js environments, less flexible for non-Node.js contexts.

📌Mocha vs. Cypress

Feature	Mocha	Cypress
Focus	Primarily for unit and integration testing, offering flexibility for various test types.	Focused on end-to-end testing for web applications.
Browser Testing	Can run tests in both Node.js and browsers, offering more flexibility.	Runs tests directly in real browsers, simulating user interactions.
Test Execution Speed	Dependent on test runner and browser setup.	Faster for end-to-end tests as it directly interacts with browsers.
Debugging	Relies on external tools for debugging (e.g., DevTools).	Built-in time travel and snapshots for easy debugging.
Automatic Waiting	Requires manual wait and timeout handling.	Automatically waits for DOM elements to load, reducing test flakiness.
Real-Time Reloading	Not available natively; requires configuration.	Built-in real-time reloads when saving files, making development faster.
Assertion Libraries	More flexible, supports custom assertion libraries like Chai or Should.	Has its own built-in assertions, but supports custom libraries as well.
CI/CD Integration	Integrates with various CI tools, offering greater flexibility through plugins.	Seamless integration with popular CI/CD tools like Jenkins, CircleCI, etc.

📌Mocha vs. Jasmine

Feature	Mocha	Jasmine
Flexibility	More flexible, allowing different assertion libraries (Chai, Should, etc.).	More opinionated with built-in assertions and spies.
Setup	Simpler to set up and configure, especially for custom setups.	Has more built-in features, which can add complexity to setup.
Test Doubles (Mocks/Spies)	Relies on external libraries like Sinon for mocks and spies.	Provides built-in support for mocks and spies.

💼 Setting Up Mocha : A Quick Guide

1. Prerequisites

- Ensure Node.js is installed

2. Install Mocha

- **Local Installation**
  ```
  npm install mocha
  ```
- **Global Installation**
  ```
  npm install -g mocha
  ```
- **Install as a development dependency**
  ```
  npm install --save-dev mocha
  ```

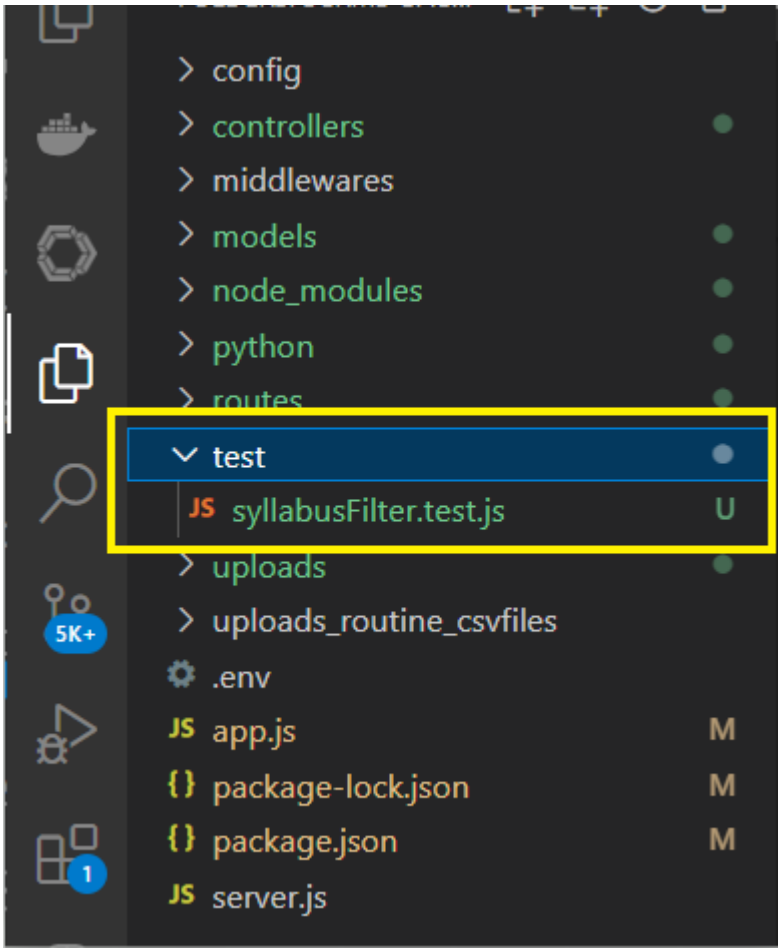
3. Add a Test Script in `package.json`

Open your `package.json` file and add the following under `"scripts"`:

```
"scripts": {
  "test": "mocha"
}
```

4. Create a Test Directory and Files

I've created a folder named **`test/`** in the root of my project.
Inside the `test/` folder, I have created a test file (e.g., `syllabusFilter.test.js`).



5. Write a Test for Sample Test Case

In this step,I've written test cases for the `fetchCourseData` function from my `syllabusFilterController.js`. Here's how you to set it up:

◆ 5.i) `fetchCourseData` method:

```
const fetchCourseData = (departmentName, sessionName, examYear, courseName, callback) => {

    /**
     * @constant {string} queryDept - SQL query to select department ID based on department name.
     */
    const queryDept = 'SELECT dept_id FROM department WHERE Dept_Name = ?;';

    /**
     * @constant {string} querySession - SQL query to select session ID based on department ID and session name.
     */
    const querySession = 'SELECT session_id FROM session WHERE dept_id = ? AND Session_name = ?;';

    /**
     * @constant {string} queryExamYear - SQL query to select exam year ID based on session ID and exam year.
     */
    const queryExamYear = 'SELECT exam_year_id FROM examyear WHERE session_id = ? AND Exam_year = ?;';

    /**
     * @constant {string} queryCourse - SQL query to select course data based on exam year ID and course title.
     */
    const queryCourse = `
        SELECT
            c.course_id,
            c.Course_code,
            c.Course_code,
            c.course_title,
            c.course_type,
            c.contact_hour,
            c.rationale
        FROM course c
        WHERE c.exam_year_id = ? AND c.course_title = ?;
    `;
```

◆ 5.ii) Import necessary libraries in `test/syllabusFilterController.js`

```
const assert = require('assert');
const sinon = require('sinon');
const { fetchCourseData } = require('../controllers/syllabusFilterController');
const pool = require('../config/db'); // Mock the database pool
```

◆ **5.iii) should return an error if department not found: This test ensures that the `fetchCourseData` function correctly returns an error with the message 'Department not found' when an invalid department name is provided.**

```
/**
 * Tests the fetchCourseData function.
 */
describe('fetchCourseData', () => {
    afterEach(() => {
        // Restore the original methods after each test
        sinon.restore();
    });

    /**
     * Tests the scenario where the department is not found.
     * Should return an error with message 'Department not found'.
     *
     * @returns {void}
     */
    it('should return an error if department not found', (done) => {
        // Mock database queries with empty results for department
        sinon.stub(pool, 'query').callsFake((query, params, callback) => {
            callback(null, []); // No department found
        });

        fetchCourseData('InvalidDept', '2019-2020', '2020', 'Software Engineering', (err, data) => {
            assert(err instanceof Error);
            assert.strictEqual(err.message, 'Department not found');
            assert.strictEqual(data, null);
            done();
        });
    });
```

◆ **5.iv) should return an error if session not found: This test verifies that the `fetchCourseData` function returns the appropriate error message 'Session not found' when a valid department is given but an invalid session is requested.**

```
/**
 * Tests the scenario where the session is not found.
 * Should return an error with message 'Session not found'.
 *
 * @returns {void}
 */
it('should return an error if session not found', (done) => {
    const deptStub = sinon.stub(pool, 'query');

    // Mock department query
    deptStub.withArgs(sinon.match.string, sinon.match.array).onFirstCall().callsFake((query, params, 
        callback(null, [{ dept_id: 1 }]); // Department found
    });

    // Mock session query (not found)
    deptStub.withArgs(sinon.match.string, sinon.match.array).onSecondCall().callsFake((query, params,
        callback(null, []); // No session found
    });

    fetchCourseData('Department of Computer Science and Engineering', 'InvalidSession', '2020',
                    'Software Engineering',
        (err, data) => {
        assert(err instanceof Error);
        assert.strictEqual(err.message, 'Session not found');
        assert.strictEqual(data, null);
        done();
    });
});
```

◆ 5.v) should return an error if the exam year does not exist: This test checks that the `fetchCourseData` function responds with the error message 'Exam year not found' when valid department and session parameters are supplied, but the specified exam year does not exist.

```
 * Should return an error with message 'Exam year not found'.
 * @returns {void}
 */
it('should return an error if the exam year does not exist', (done) => {
    const deptStub = sinon.stub(pool, 'query');

    // Mock department query
    deptStub.withArgs(sinon.match.string, sinon.match.array).onFirstCall().callsFake((query, params,
        callback(null, [{ dept_id: 1 }]); // Department found
    });

    // Mock session query
    deptStub.withArgs(sinon.match.string, sinon.match.array).onSecondCall().callsFake((query, params,
        callback(null, [{ session_id: 2 }]); // Session found
    });

    // Mock exam year query (not found)
    deptStub.withArgs(sinon.match.string, sinon.match.array).onThirdCall().callsFake((query, params,
        callback(null, []); // No exam year found
    });

    fetchCourseData('Department of Computer Science and Engineering', '2019-2020', 'InvalidYear',
                    'Software Engineering',
        (err, data) => {
        assert(err instanceof Error);
        assert.strictEqual(err.message, 'Exam year not found');
        assert.strictEqual(data, null);
        done();
    });
```

6. Run Tests

Execute the following command to run your tests:

```
npx mocha .\syllabusFilter.test.js
```

```
PS C:\SmartRoutine\scrms-backend> cd .\test\
PS C:\SmartRoutine\scrms-backend\test> npx mocha .\syllabusFilter.test.js
```

7. Test Output

```
PS C:\SmartRoutine\scrms-backend\test> npx mocha .\syllabusFilter.test.js


  Connected to MySQL database
    fetchCourseData
      ✓ should return an error if department not found
      ✓ should return an error if session not found
      ✓ should return an error if the exam year does not exist


    3 passing (22ms)
```

Assertions

Click Here

References:Mocha