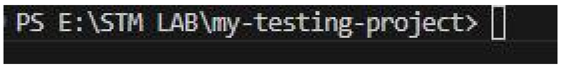
Integration Testing Using Jest with supertest

Jest is a JavaScript testing framework developed by Meta (Facebook) used for writing unit and integration tests.
Supertest is a library that works with Jest (or other testing frameworks) to test HTTP endpoints in Node.js applications by simulating real API requests and responses.
When we use Jest and Supertest together it allows developers to test both the functionality and behavior of backend APIs. Jest handles the testing environment, assertions, and mocking, while Supertest sends HTTP requests to your Express.js or Node.js server to verify routes, status codes, and responses ensuring your API behaves as expected.

Open the Project Directory:

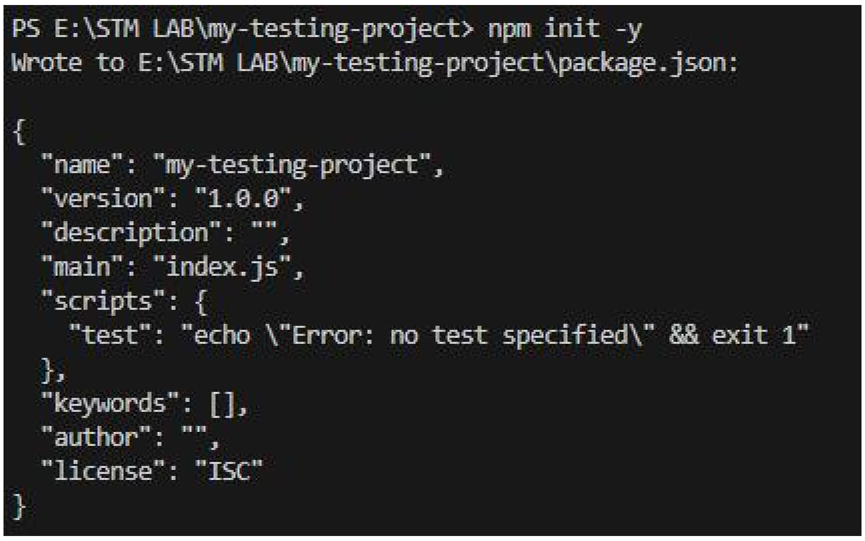
Navigate to the project folder where we will do the testing .



```
PS E:\STM LAB\my-testing-project> 
```

Initialize the Project

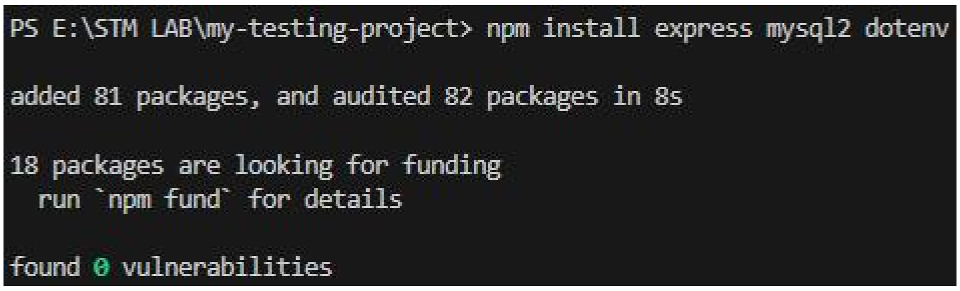
Set up a new Node.js project by running the initialization command inside the project folder:



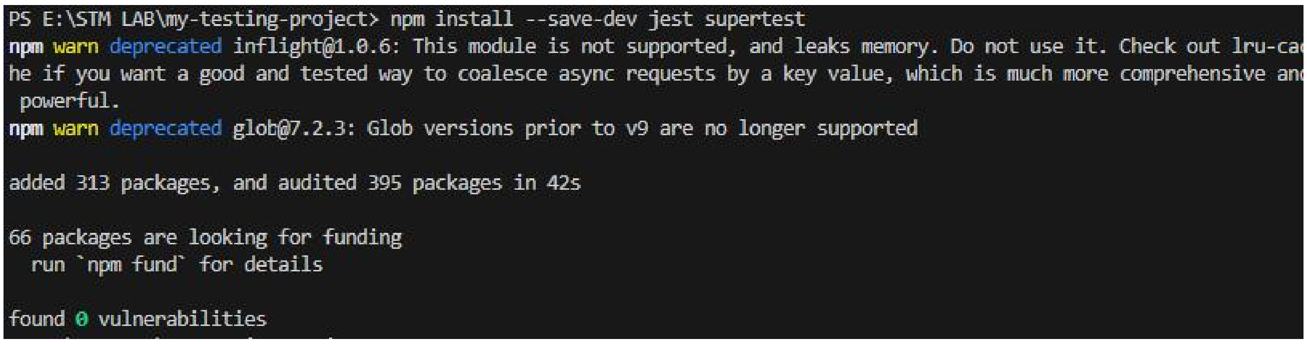
```
PS E:\STM LAB\my-testing-project> npm init -y
Wrote to E:\STM LAB\my-testing-project\package.json:

{
  "name": "my-testing-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Install Required Dependencies

Install the runtime dependencies and testing libraries needed for the project.
Run the following commands one after another:



```
PS E:\STM LAB\my-testing-project> npm install express mysql2 dotenv

added 81 packages, and audited 82 packages in 8s

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

found 0 vulnerabilities
```



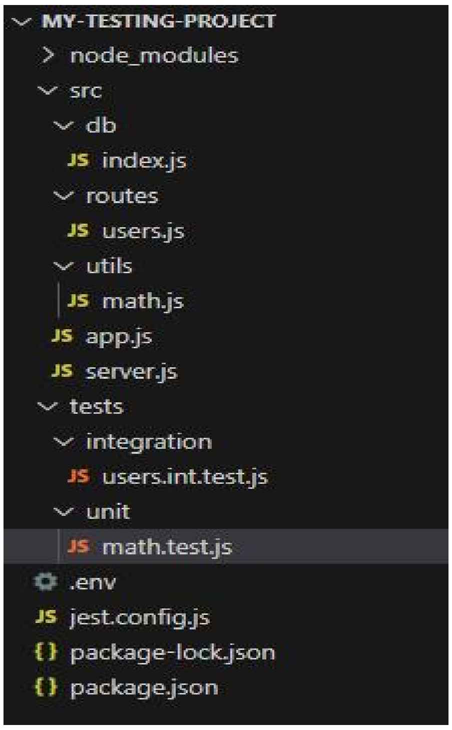
```
PS E:\STM LAB\my-testing-project> npm install --save-dev jest supertest
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-ca
he if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive an
 powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 313 packages, and audited 395 packages in 42s

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

found 0 vulnerabilities
```

npm install express mysql2 dotenv → Installs the main (runtime) dependencies:

npm install --save-dev jest supertest → Installs the testing tools as development dependencies

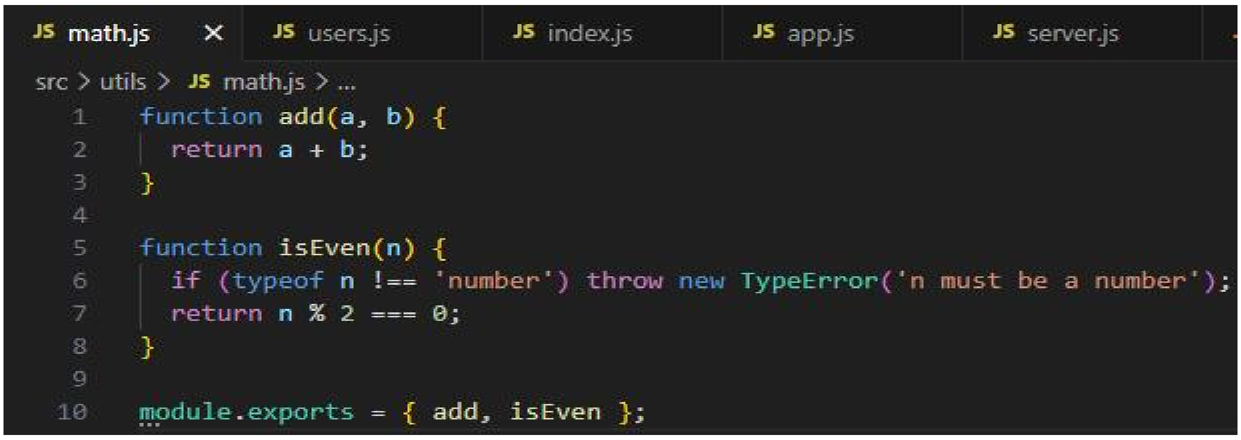
Set Up the Folder Structure

Created the following directory structure inside project:

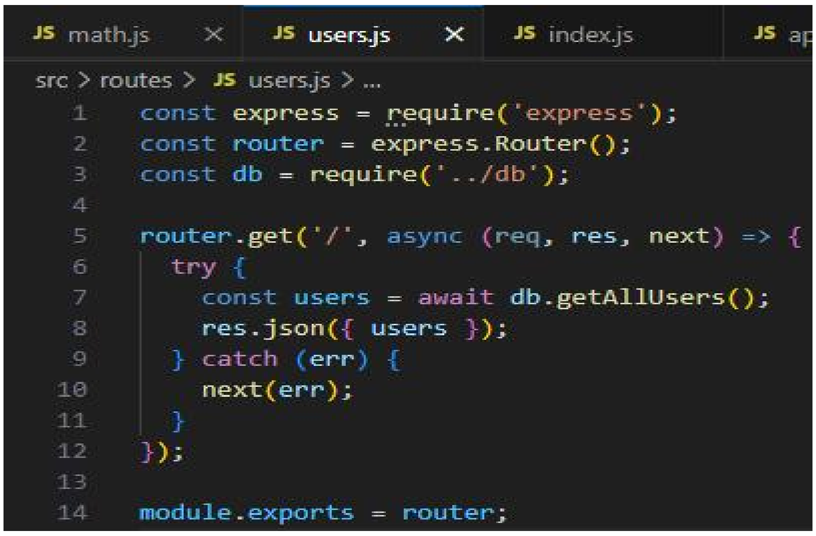


```
MY-TESTING-PROJECT
  > node_modules
  ∨ src
    ∨ db
      JS index.js
    ∨ routes
      JS users.js
    ∨ utils
      JS math.js
    JS app.js
    JS server.js
  ∨ tests
    ∨ integration
      JS users.int.test.js
    ∨ unit
      JS math.test.js
  .env
  JS jest.config.js
  {} package-lock.json
  {} package.json
```

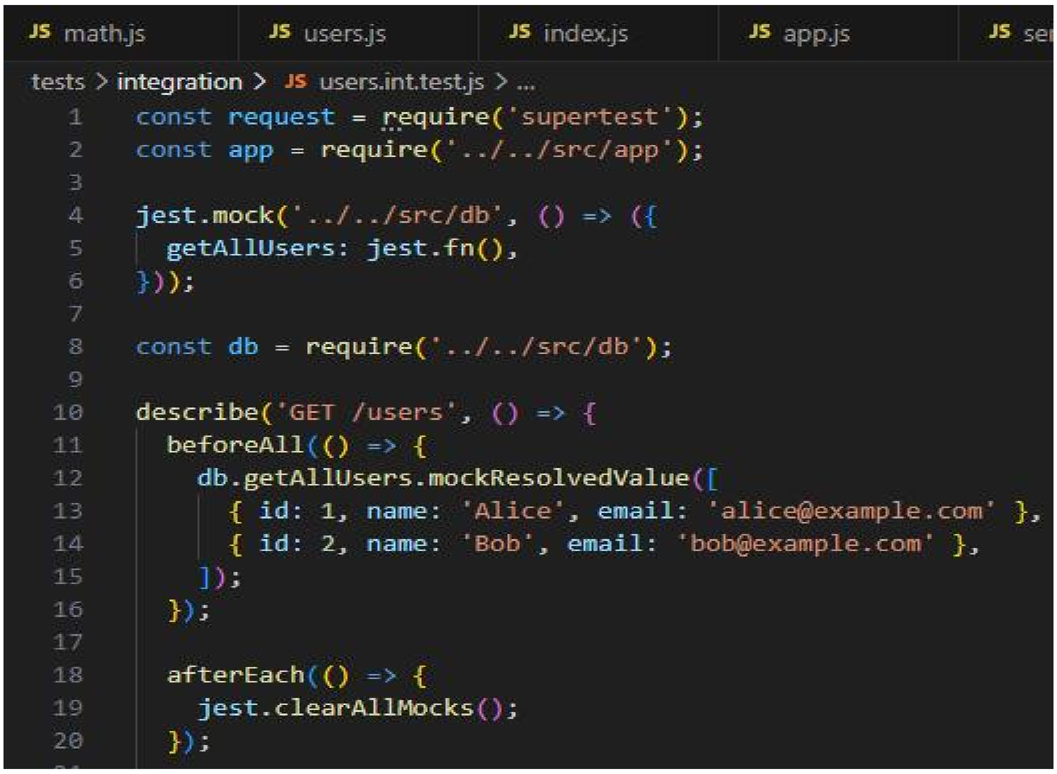
Write code for src\utils\math.js and src\routes\user.js and tests\unit\math.test.js and tests\integration\users.int.test.js .



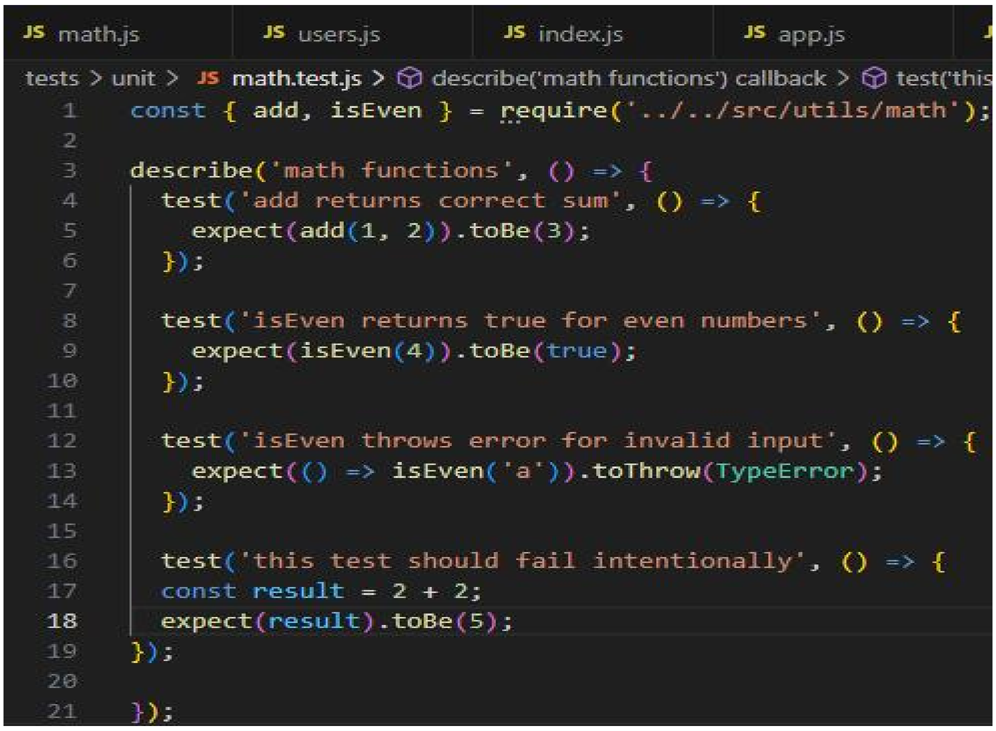
```javascript
function add(a, b) {
  return a + b;
}

function isEven(n) {
  if (typeof n !== 'number') throw new TypeError('n must be a number');
  return n % 2 === 0;
}

module.exports = { add, isEven };
```



```javascript
const express = require('express');
const router = express.Router();
const db = require('../db');

router.get('/', async (req, res, next) => {
  try {
    const users = await db.getAllUsers();
    res.json({ users });
  } catch (err) {
    next(err);
  }
});

module.exports = router;
```



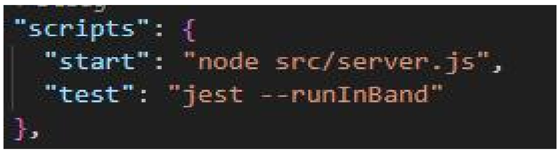
```javascript
const request = require('supertest');
const app = require('../../src/app');

jest.mock('../../src/db', () => ({
  getAllUsers: jest.fn(),
}));

const db = require('../../src/db');

describe('GET /users', () => {
  beforeAll(() => {
    db.getAllUsers.mockResolvedValue([
      { id: 1, name: 'Alice', email: 'alice@example.com' },
      { id: 2, name: 'Bob', email: 'bob@example.com' },
    ]);
  });

  afterEach(() => {
    jest.clearAllMocks();
  });
```


JS math.js
JS users.js
JS index.js
JS app.js
tests > unit > JS math.test.js > describe('math functions') callback > test('this

```javascript
const { add, isEven } = require('../../src/utils/math');

describe('math functions', () => {
  test('add returns correct sum', () => {
    expect(add(1, 2)).toBe(3);
  });

  test('isEven returns true for even numbers', () => {
    expect(isEven(4)).toBe(true);
  });

  test('isEven throws error for invalid input', () => {
    expect(() => isEven('a')).toThrow(TypeError);
  });

  test('this test should fail intentionally', () => {
  const result = 2 + 2;
  expect(result).toBe(5);
});

});
```

Configure Scripts in package.json

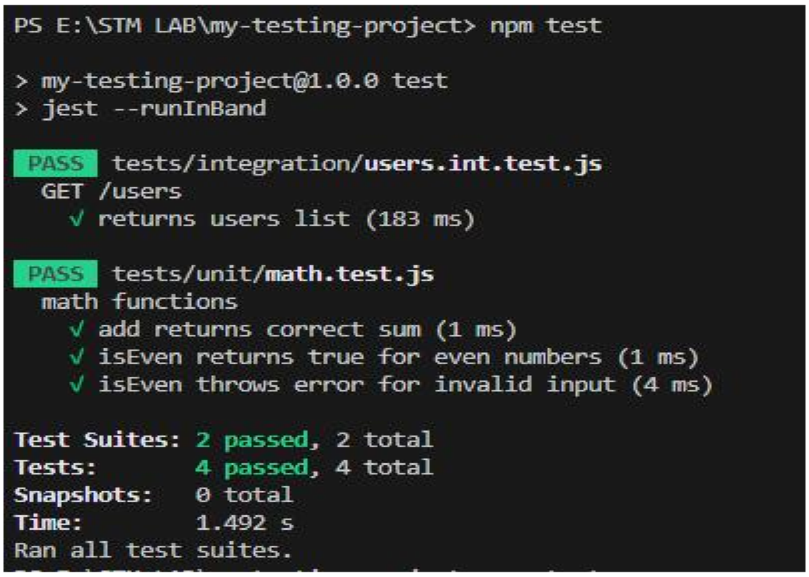
Open package.json and replace the default scripts section with:



```json
"scripts": {
  "start": "node src/server.js",
  "test": "jest --runInBand"
},
```

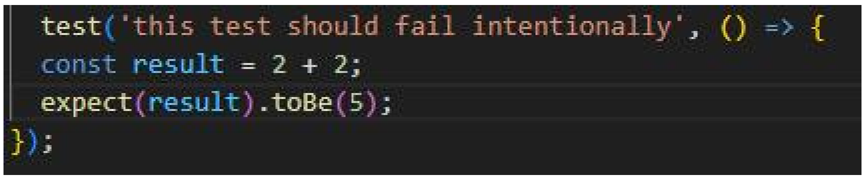
Step 7: Run Unit and Integration Tests

Execute the test command in the terminal:



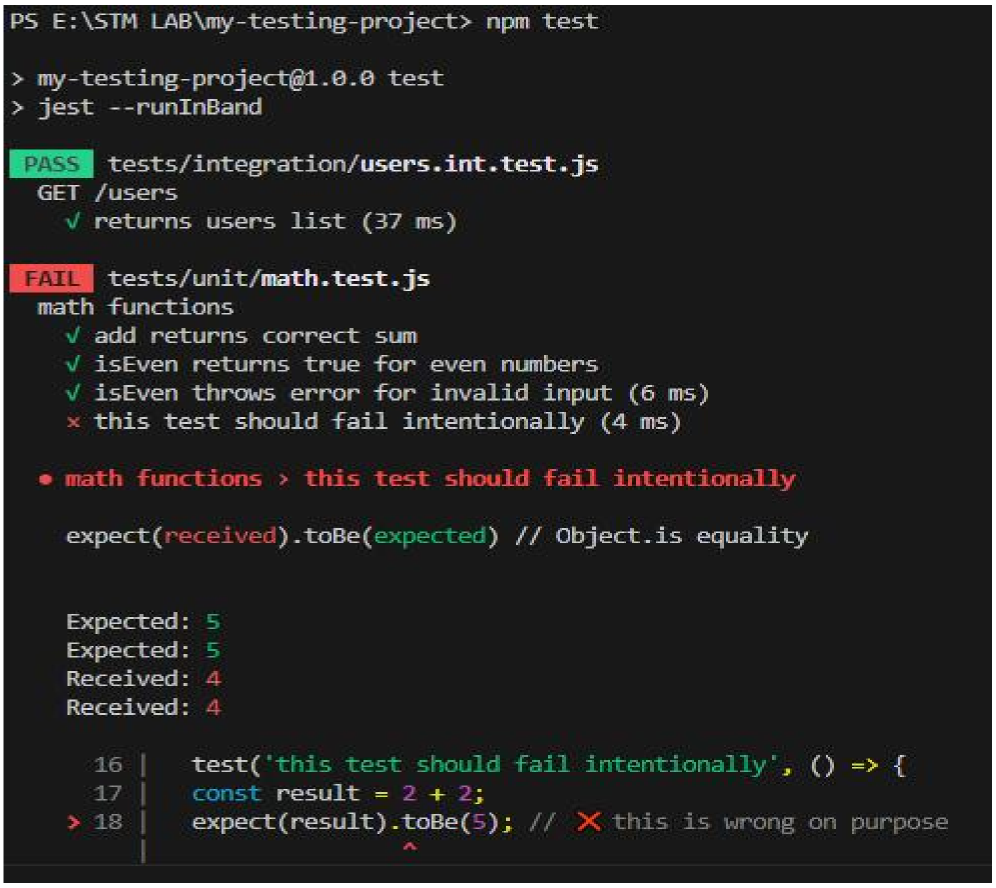
```
PS E:\STM LAB\my-testing-project> npm test

> my-testing-project@1.0.0 test
> jest --runInBand

 PASS  tests/integration/users.int.test.js
  GET /users
    √ returns users list (183 ms)

 PASS  tests/unit/math.test.js
  math functions
    √ add returns correct sum (1 ms)
    √ isEven returns true for even numbers (1 ms)
    √ isEven throws error for invalid input (4 ms)

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.492 s
Ran all test suites.
```

To demonstrate a failing test add this line to math.test.js:



```
  test('this test should fail intentionally', () => {
  const result = 2 + 2;
  expect(result).toBe(5);
});
```

Run again:

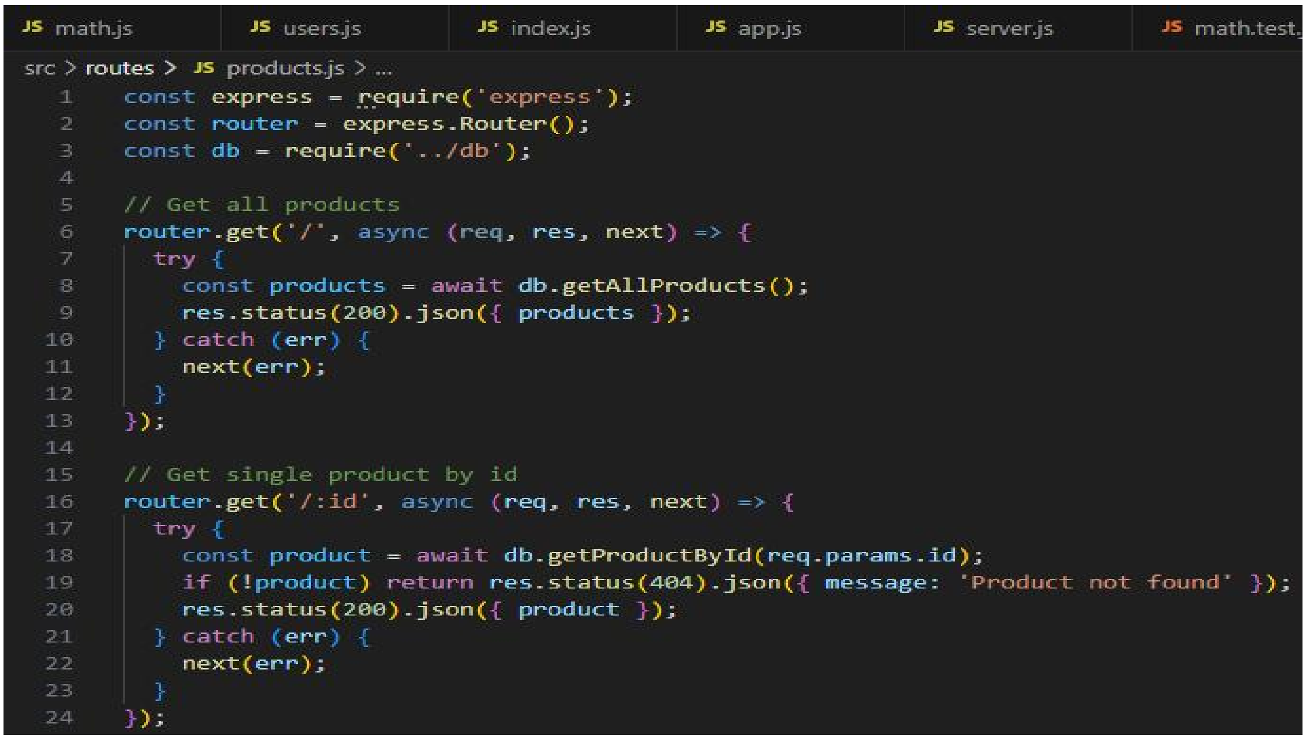


```
PS E:\STM LAB\my-testing-project> npm test

> my-testing-project@1.0.0 test
> jest --runInBand

 PASS  tests/integration/users.int.test.js
  GET /users
    √ returns users list (37 ms)

 FAIL  tests/unit/math.test.js
  math functions
    √ add returns correct sum
    √ isEven returns true for even numbers
    √ isEven throws error for invalid input (6 ms)
    × this test should fail intentionally (4 ms)

  ● math functions › this test should fail intentionally

    expect(received).toBe(expected) // Object.is equality


    Expected: 5
    Expected: 5
    Received: 4
    Received: 4

      16 |     test('this test should fail intentionally', () => {
      17 |     const result = 2 + 2;
    > 18 |     expect(result).toBe(5); // ❌ this is wrong on purpose
         |                    ^
```

Product API using Jest and Supertest

In this we will test that the API correctly returns the list of all available products.

Write code under src\routes\products.js

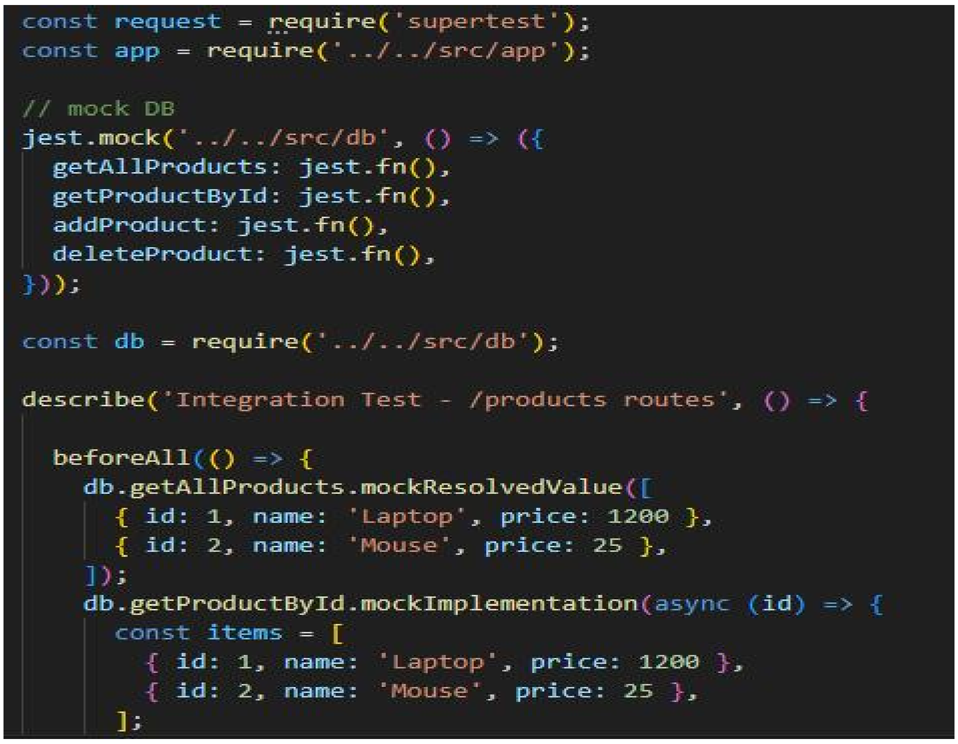


JS math.js   JS users.js   JS index.js   JS app.js   JS server.js   JS math.test.

src > routes > JS products.js > ...

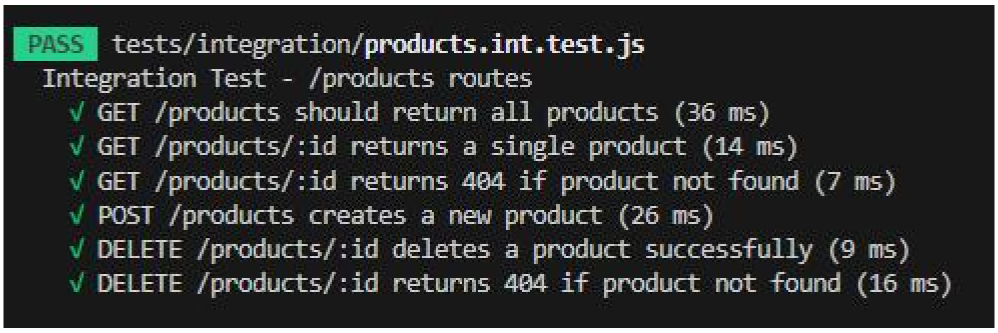
```
const express = require('express');
const router = express.Router();
const db = require('../db');

// Get all products
router.get('/', async (req, res, next) => {
  try {
    const products = await db.getAllProducts();
    res.status(200).json({ products });
  } catch (err) {
    next(err);
  }
});

// Get single product by id
router.get('/:id', async (req, res, next) => {
  try {
    const product = await db.getProductById(req.params.id);
    if (!product) return res.status(404).json({ message: 'Product not found' });
    res.status(200).json({ product });
  } catch (err) {
    next(err);
  }
});
```

Write code under tests\integration\products.int.test.js



```
const request = require('supertest');
const app = require('../../src/app');

// mock DB
jest.mock('../../src/db', () => ({
  getAllProducts: jest.fn(),
  getProductById: jest.fn(),
  addProduct: jest.fn(),
  deleteProduct: jest.fn(),
}));

const db = require('../../src/db');

describe('Integration Test - /products routes', () => {

  beforeAll(() => {
    db.getAllProducts.mockResolvedValue([
      { id: 1, name: 'Laptop', price: 1200 },
      { id: 2, name: 'Mouse', price: 25 },
    ]);
    db.getProductById.mockImplementation(async (id) => {
      const items = [
        { id: 1, name: 'Laptop', price: 1200 },
        { id: 2, name: 'Mouse', price: 25 },
      ];
```

Run test

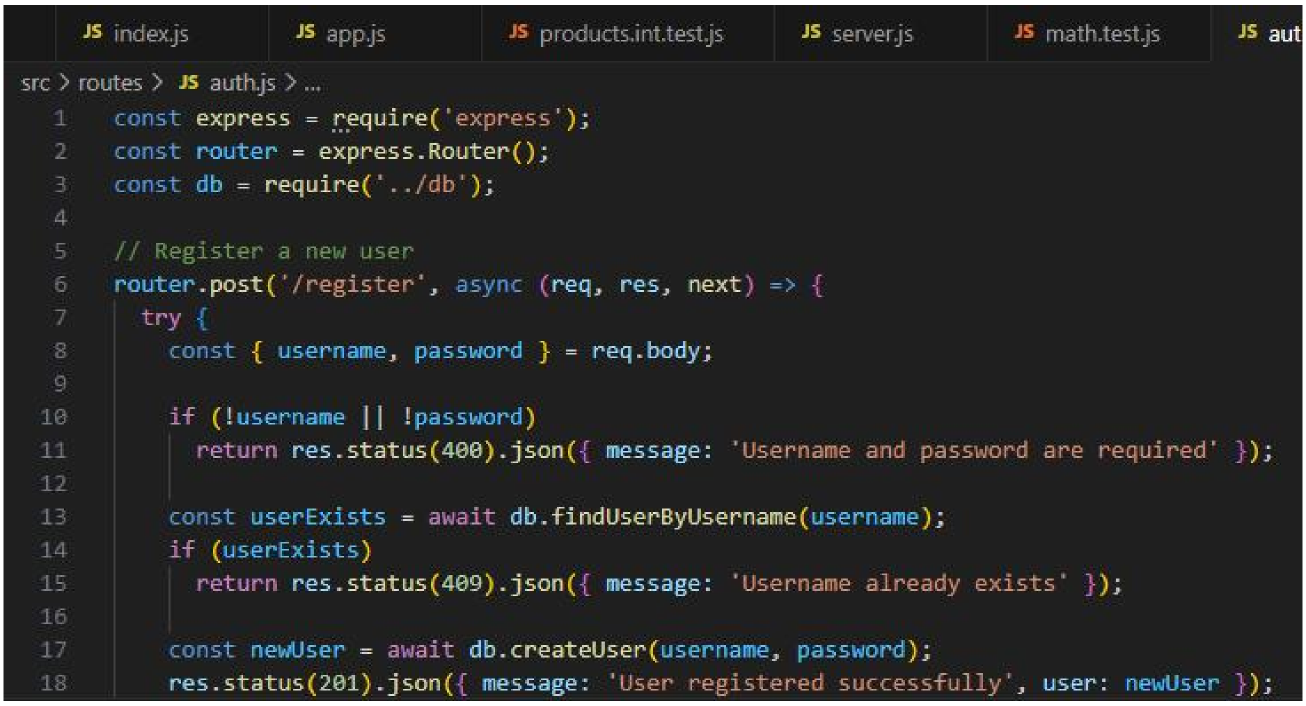


```
PASS  tests/integration/products.int.test.js
  Integration Test - /products routes
    √ GET /products should return all products (36 ms)
    √ GET /products/:id returns a single product (14 ms)
    √ GET /products/:id returns 404 if product not found (7 ms)
    √ POST /products creates a new product (26 ms)
    √ DELETE /products/:id deletes a product successfully (9 ms)
    √ DELETE /products/:id returns 404 if product not found (16 ms)
```

Integration Testing for User Authentication API

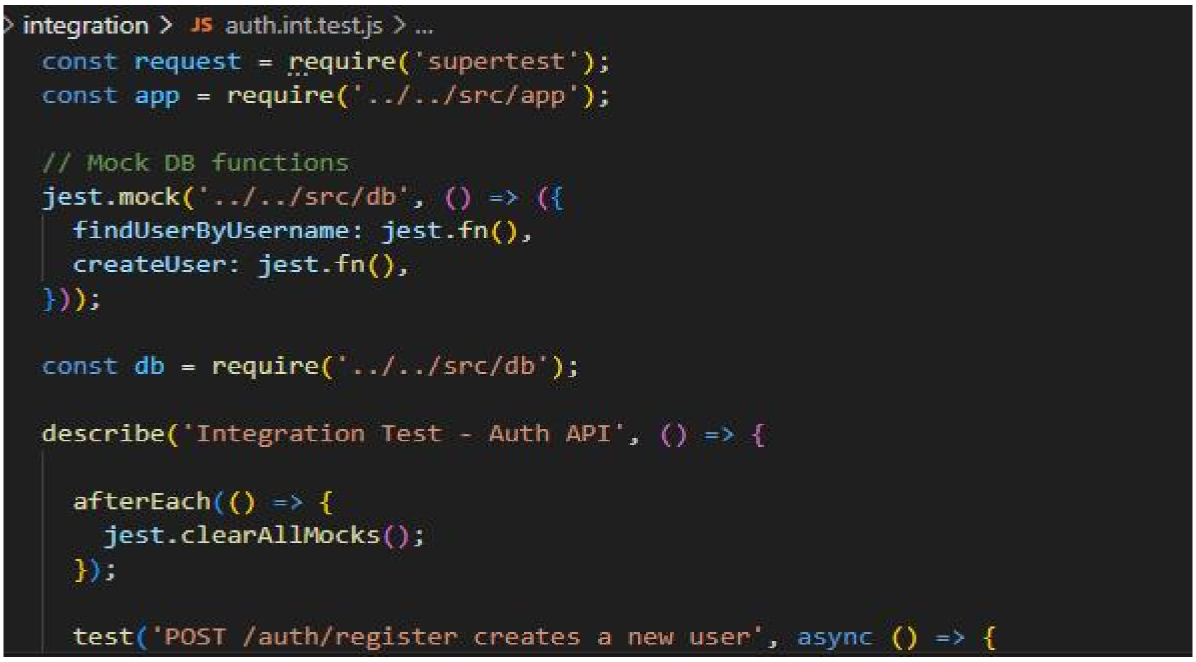
In this we will test that the authentication API responds correctly to valid and invalid inputs, returning proper status codes

Write code under src/routes/auth.js



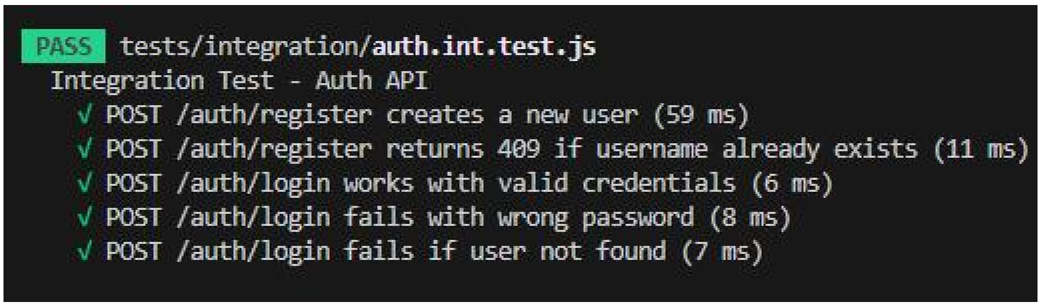
```
index.js   app.js   products.int.test.js   server.js   math.test.js   aut
src > routes > auth.js > ...
const express = require('express');
const router = express.Router();
const db = require('../db');

// Register a new user
router.post('/register', async (req, res, next) => {
  try {
    const { username, password } = req.body;

    if (!username || !password)
      return res.status(400).json({ message: 'Username and password are required' });

    const userExists = await db.findUserByUsername(username);
    if (userExists)
      return res.status(409).json({ message: 'Username already exists' });

    const newUser = await db.createUser(username, password);
    res.status(201).json({ message: 'User registered successfully', user: newUser });
```

Write code under tests/integration/auth.int.test.js



```
integration > auth.int.test.js > ...
const request = require('supertest');
const app = require('../../src/app');

// Mock DB functions
jest.mock('../../src/db', () => ({
  findUserByUsername: jest.fn(),
  createUser: jest.fn(),
}));

const db = require('../../src/db');

describe('Integration Test - Auth API', () => {

  afterEach(() => {
    jest.clearAllMocks();
  });

  test('POST /auth/register creates a new user', async () => {
```

Run the test



```
PASS tests/integration/auth.int.test.js
  Integration Test - Auth API
    √ POST /auth/register creates a new user (59 ms)
    √ POST /auth/register returns 409 if username already exists (11 ms)
    √ POST /auth/login works with valid credentials (6 ms)
    √ POST /auth/login fails with wrong password (8 ms)
    √ POST /auth/login fails if user not found (7 ms)
```

Advantages of Using Supertest with Jest

1. Complete Testing Setup

- Together, they allow you to test:

    I. Unit tests (with Jest)
    II. Integration tests (with Supertest)

- Can test both backend logic and API endpoints in the same environment.

2. Simple and Readable Syntax

- Jest + Supertest supports async/await and chained assertions, making tests clean.

3. Realistic API Testing

- Simulates **real client behavior** by sending actual HTTP requests.
- Helps verify that routing, middleware, and controllers work correctly together.

4. Powerful Assertions and Matchers

- Jest's built-in expect() provides flexible matchers for verifying responses:

    I. Status codes
    II. JSON structure
    III. Headers
    IV. Error messages

5. Automatic Test Discovery and Running

- Jest automatically finds test files (like *.test.js or *.spec.js) - no manual setup needed.

6. Mocking Support (Jest)

- We can use Jest's mocking features to fake external dependencies (like databases or APIs) during tests.

7. Code Coverage Reports

- Jest can generate code coverage even for integration tests with Supertest:

```
npx jest --coverage
```

- Helps us track how much of our backend code is tested.

8. Integrated Error Reporting

- Jest provides detailed error messages and colored output.
- Makes it easy to see which endpoint or assertion failed.

9. Works Perfectly with CI/CD

- Jest + Supertest tests can be easily integrated into continuous integration tools like GitHub Actions, Jenkins, or Travis CI.

Disadvantages of Using Supertest with Jest

1. Slower than Unit Tests

- Integration tests using Supertest are slower because:
- They simulate real HTTP requests.
- They may connect to a real (or test) database.

2. Complex Database Setup

- When using MySQL, we need:

  I. A **separate test database**
  II. Scripts to **seed or clean data** before each test

- Without proper isolation, tests can interfere with each other.

3. Harder Debugging

- If a test fails, it can be unclear whether the problem is in:

  I. The Express route
  II. Middleware
  III. Database query
  IV. The test setup itself

4. Limited Frontend Testing

- Jest + Supertest only tests backend APIs.
  We will still need tools like **React Testing Library** or **Cypress** for frontend UI tests.

5. Requires Additional Configuration for Large Projects

- For small apps, setup is easy.
- But in larger apps (many routes, DBs, auth), we'll need extra setup like:

  I. Mocking JWT authentication

II. Handling sessions/cookies
III. Connecting and disconnecting from databases before/after tests

6. Not Meant for End-to-End (E2E) Browser Tests

- Supertest can’t test how frontend and backend work together in a browser - it’s API-only.