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\integration-demo>
```

Initialize the Project

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

```
PS E:\STM LAB\integration-demo> npm init -y
Wrote to E:\STM LAB\integration-demo\package.json:

{
  "name": "integration-demo",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "directories": {
    "test": "tests"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Install Required Dependencies

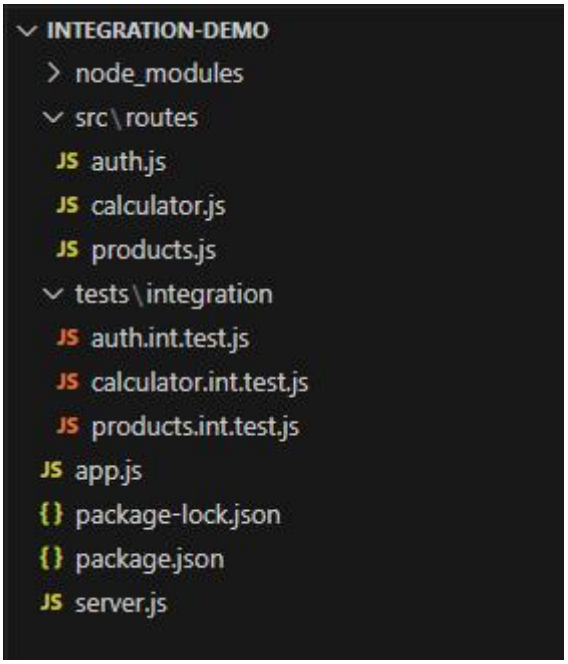
Install the runtime dependencies and testing libraries needed for the project.
Run the following commands one after another:

```
PS E:\STM LAB\integration-demo> npm install express

added 68 packages, and audited 69 packages in 6s

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

found 0 vulnerabilities
```

```
PS E:\STM LAB\integration-demo> 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-cache if you want a good an
d tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 313 packages, and audited 382 packages in 2m

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

found 0 vulnerabilities
```

Set Up the Folder Structure

Created the following directory structure inside project:



app.js

server.js

Product API

src/routes/products.js

tests/integration/products.int.test.js

Run test:

```
PS E:\STM LAB\integration-demo> npx jest tests/integration/products.int.test.js
 PASS  tests/integration/products.int.test.js
  Product API
    √ should return all products (54 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.078 s, estimated 3 s
Ran all test suites matching tests/integration/products.int.test.js.
PS E:\STM LAB\integration-demo>
```

Auth API

src/routes/auth.js

server.js | products.js | auth.js | calculator.js | auth.int.test.js

src > routes > auth.js > ...

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

router.post('/login', (req, res) => {
  const { user, pass } = req.body;
  if (user === 'admin' && pass === '1234')
    res.json({ message: 'Login success' });
  else res.status(401).json({ error: 'Invalid credentials' });
});

module.exports = router;
```

tests/integration/auth.int.test.js

server.js | products.js | auth.js | calculator.js | auth.int.test.js | calculator.int.test.js

tests > integration > auth.int.test.js > ...

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

describe('Auth API', () => {
  it('should login successfully with correct credentials', async () => {
    const res = await request(app).post('/auth/login').send({ user: 'admin', pass: '1234' });
    expect(res.statusCode).toBe(200);
  });

  it('should fail with wrong credentials', async () => {
    const res = await request(app).post('/auth/login').send({ user: 'admin', pass: 'wrong' });
    expect(res.statusCode).toBe(401);
  });
});
```

Run test:

```
PASS tests/integration/auth.int.test.js
  Auth API
    √ should login successfully with correct credentials (62 ms)
    √ should fail with wrong credentials (12 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.875 s, estimated 1 s
Ran all test suites matching tests/integration/auth.int.test.js.
PS E:\STM LAB\integration-demo>
```

Calculator API

src/routes/calculator.js

server.js | products.js | auth.js | calculator.js | auth.int.test

src > routes > calculator.js > ...

```
1  const express = require('express');
2  const router = express.Router();
3
4  router.get('/add', (req, res) => {
5    const a = Number(req.query.a);
6    const b = Number(req.query.b);
7    res.json({ sum: a + b });
8  });
9
10 module.exports = router;
```

tests/integration/calculator.int.test.js

server.js | products.js | auth.js | calculator.js | auth.int.

tests > integration > calculator.int.test.js > describe('Calculator API') callback

```
1  const request = require('supertest');
2  const app = require('../../app');
3
4  describe('Calculator API', () => {
5    it('should return correct sum', async () => {
6      const res = await request(app).get('/calc/add?a=3&b=2');
7      expect(res.body.sum).toBe(5);
8    });
```

Run test:

```
PS E:\STM LAB\integration-demo> npx jest tests/integration/calculator.int.test.js
 PASS  tests/integration/calculator.int.test.js
  Calculator API
    √ should return correct sum (49 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.252 s, estimated 2 s
Ran all test suites matching tests/integration/calculator.int.test.js.
```

Add this extra test case at the bottom:

```
  it("should fail intentionally for demo", async () => {
  const res = await request(app).get("/calc/add?a=2&b=2");
  // The correct answer is 4, but we expect 5 to make it fail
  expect(res.body.sum).toBe(5);
});
```

Run test again

```
PS E:\STM LAB\integration-demo> npx jest tests/integration/calculator.int.test.js
 FAIL  tests/integration/calculator.int.test.js
  Calculator API
    √ should return correct sum (48 ms)
    × should fail intentionally for demo (17 ms)

  ● Calculator API › should fail intentionally for demo

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

    Expected: 5
    Received: 4

      10 |   const res = await request(app).get("/calc/add?a=2&b=2");
      11 |   // The correct answer is 4, but we expect 5 to make it fail
    > 12 |   expect(res.body.sum).toBe(5);
         |                        ^
      13 | });
      14 | });
      15 |

      at Object.toBe (tests/integration/calculator.int.test.js:12:24)
```

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.