Міністерство освіти і науки україни

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Математичний факультет

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Дисципліна «Автоматизоване тестування»

Лабораторна робота № 6

Виконав студент гр. 8.1214

Ніценко Вадим

Перевірив: Викладач

Лебідь Максим Володимирович

Запоріжжя

2024

**Завдання:**

1. Ознайомитись з переліком API для тестування https://gorest.co.in/
2. Виконати тестування основних методів вибраного API.
3. Створити github репозиторій з тестами.

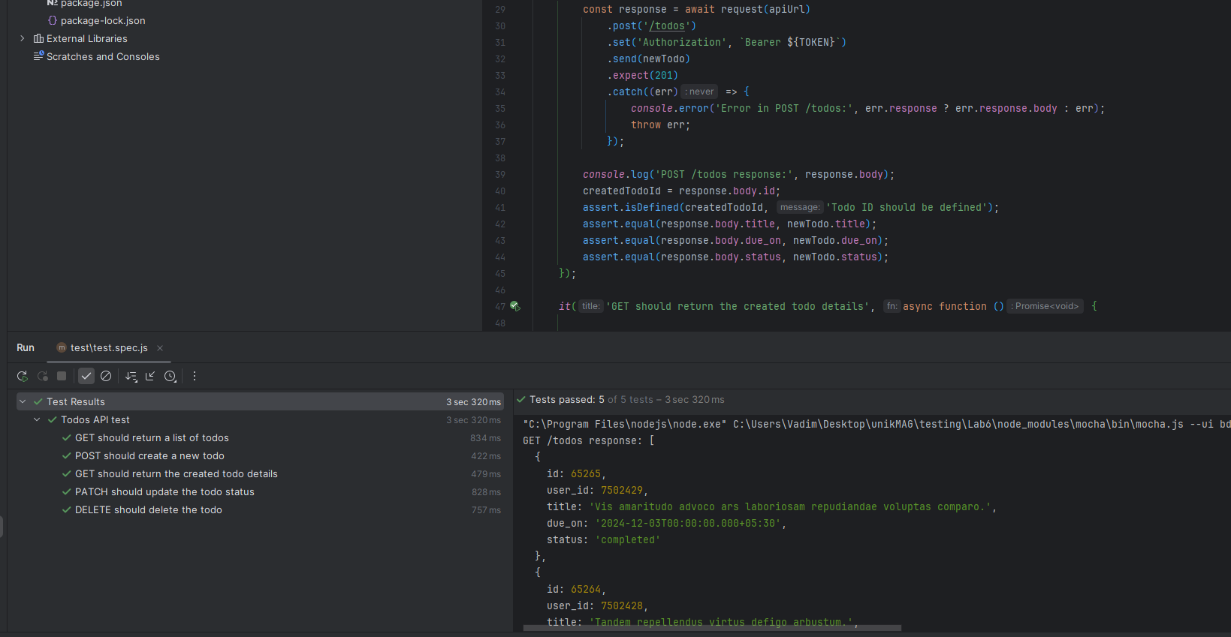
**Хід роботи**

**Репозиторій:** <https://github.com/Shonistro/testing/tree/main/Lab6>

Для тестування вибрав API todo

Код

const request = require('supertest');  
const { assert } = require('chai');  
  
*describe*('Todos API test', function () {  
 this.timeout(10000);  
 const apiUrl = 'https://gorest.co.in/public/v2';  
 const TOKEN = '8363484f5ce4a7931f0e89e9232f6efa0738a3dbe76cef3106d1d0b5f4e7b4a7';  
 let createdTodoId;  
  
 const newTodo = {  
 title: "Sample TODO title",  
 due\_on: "2024-12-03T00:00:00.000+05:30",  
 status: "pending",  
 user\_id: 7502429  
 };  
  
 *it*('GET should return a list of todos', async () => {  
 const response = await request(apiUrl)  
 .get('/todos')  
 .set('Authorization', `Bearer ${TOKEN}`)  
 .expect(200);  
  
 *console*.log('GET /todos response:', response.body);  
 assert.isArray(response.body, "Response should be an array");  
 assert.isNotEmpty(response.body, "Response array should not be empty");  
 });  
  
 *it*('POST should create a new todo', async () => {  
 const response = await request(apiUrl)  
 .post('/todos')  
 .set('Authorization', `Bearer ${TOKEN}`)  
 .send(newTodo)  
 .expect(201)  
 .catch((err) => {  
 *console*.error('Error in POST /todos:', err.response ? err.response.body : err);  
 throw err;  
 });  
  
 *console*.log('POST /todos response:', response.body);  
 createdTodoId = response.body.id;  
 assert.isDefined(createdTodoId, 'Todo ID should be defined');  
 assert.equal(response.body.title, newTodo.title);  
 assert.equal(response.body.due\_on, newTodo.due\_on);  
 assert.equal(response.body.status, newTodo.status);  
 });  
  
 *it*('GET should return the created todo details', async function () {  
  
 const response = await request(apiUrl)  
 .get(`/todos/${createdTodoId}`)  
 .set('Authorization', `Bearer ${TOKEN}`)  
 .expect(200);  
  
 *console*.log(`GET /todos/${createdTodoId} response:`, response.body);  
 assert.equal(response.body.id, createdTodoId);  
 assert.equal(response.body.title, newTodo.title);  
 assert.equal(response.body.due\_on, newTodo.due\_on);  
 assert.equal(response.body.status, newTodo.status);  
 });  
  
 *it*('PATCH should update the todo status', async function () {  
 if (!createdTodoId) this.skip();  
  
 const updatedStatus = 'completed';  
 const response = await request(apiUrl)  
 .patch(`/todos/${createdTodoId}`)  
 .set('Authorization', `Bearer ${TOKEN}`)  
 .send({ status: updatedStatus })  
 .expect(200);  
  
 *console*.log(`PATCH /todos/${createdTodoId} response:`, response.body);  
 assert.equal(response.body.status, updatedStatus);  
 });  
  
 *it*('DELETE should delete the todo', async function () {  
  
 const response = await request(apiUrl)  
 .delete(`/todos/${createdTodoId}`)  
 .set('Authorization', `Bearer ${TOKEN}`)  
 .expect(204);  
  
 *console*.log(`DELETE /todos/${createdTodoId} response:`, response.body);  
 });  
});

Результат

Контрольні запитання

1.Види тестування:

* Модульне тестування: тестування окремих модулів чи функцій програми, зазвичай ізольовано від решти коду.
* Інтеграційне тестування: тестування взаємодії між різними модулями чи компонентами програми.
* Системне тестування: тестування всього програмного комплексу як єдиної системи для перевірки відповідності вимогам.
* Приймальне тестування: фінальна перевірка програми перед її впровадженням, щоб переконатися, що вона відповідає вимогам користувача.
* Функціональне тестування: перевірка функцій програми відповідно до специфікації.
* Регресійне тестування: перевірка того, що зміни в коді не призвели до появи нових дефектів.
* Навантажувальне тестування: тестування продуктивності програми під високим навантаженням.
* Стресове тестування: перевірка програми на працездатність в умовах позаштатних ситуацій чи значного навантаження.

2.Особливості модульного та приймального тестування:

Модульне тестування:

* Зосереджене на тестуванні окремих функцій чи модулів програми в ізоляції від інших компонентів.
* Проводиться розробниками або автоматизованими засобами під час написання коду.
* Дає змогу виявити та виправити помилки на ранніх етапах розробки.

Приймальне тестування:

* Орієнтоване на перевірку відповідності програми вимогам кінцевих користувачів.
* Виконується на останньому етапі розробки, іноді безпосередньо замовником.
* Дозволяє впевнитися, що продукт готовий до використання в реальних умовах.

Призначення бібліотек Mocha та Chai:

Mocha

* Це фреймворк для тестування на JavaScript, який надає середовище для організації та виконання тестів.
* Дозволяє виконувати асинхронні тести, забезпечує функціонал для написання тестових наборів і структурування тестів.
* Підтримує різні стилі асерцій та інтеграцію з іншими бібліотеками для асерцій, як-от Chai.

Chai:

* Бібліотека для асерцій у JavaScript, яка використовується разом із фреймворком тестування.
* Забезпечує зручний синтаксис для перевірки значень і очікувань.
* Полегшує написання зрозумілих і простих для читання тестів за допомогою різних стилів асерцій.