МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Автоматизоване тестування»

Лабораторна робота № 6

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

Перевірив: Лебідь Максим   
 Володимирович

Запоріжжя  
2023

**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.

API (Application Programming Interface) для тестування - це інтерфейс, який надає можливість взаємодії з певним програмним продуктом чи сервісом, зазвичай за допомогою HTTP-запитів. Для тестування API ви можете використовувати різні методи HTTP, такі як GET, POST, PUT, DELETE, для взаємодії з ресурсами чи об'єктами на сервері.

API для тестування може бути створено розробниками чи командами, які надають зручний спосіб взаємодії з функціональністю їх програмного продукту. Наприклад, соціальні мережі, платіжні системи, веб-сервіси та інші інтернет-ресурси надають API для взаємодії з їхніми службами.

Під час тестування API ви можете перевіряти, чи працюють різні запити та відповіді, чи повертаються очікувані дані, і чи коректно обробляються різні сценарії взаємодії з сервером. Тестування API дозволяє забезпечити стабільність та правильність роботи програмного продукту на рівні взаємодії з іншими системами.

**Код програми:**

const axios = require('axios');

// Функція для виконання GET-запиту

async function getUsers() {

  try {

    const response = await axios.get('https://gorest.co.in/public-api/users');

    console.log('GET Users Response:', response.data);

  } catch (error) {

    console.error('Error:', error.message);

  }

}

// Функція для виконання POST-запиту

async function createUser() {

  try {

    const userData = {

      name: 'John Doe',

      email: 'john.doe@example.com',

      gender: 'male',

    };

    const response = await axios.post('https://gorest.co.in/public-api/users', userData);

    console.log('POST User Response:', response.data);

  } catch (error) {

    console.error('Error:', error.message);

  }

}

// Функція для виконання PUT-запиту

async function updateUser(userId) {

  try {

    const updatedData = {

      name: 'Updated Name',

    };

    const response = await axios.put(`https://gorest.co.in/public-api/users/${userId}`, updatedData);

    console.log('PUT User Response:', response.data);

  } catch (error) {

    console.error('Error:', error.message);

  }

}

// Функція для виконання DELETE-запиту

async function deleteUser(userId) {

  try {

    const response = await axios.delete(`https://gorest.co.in/public-api/users/${userId}`);

    console.log('DELETE User Response:', response.data);

  } catch (error) {

    console.error('Error:', error.message);

  }

}

// Виклик функцій для тестування

getUsers();

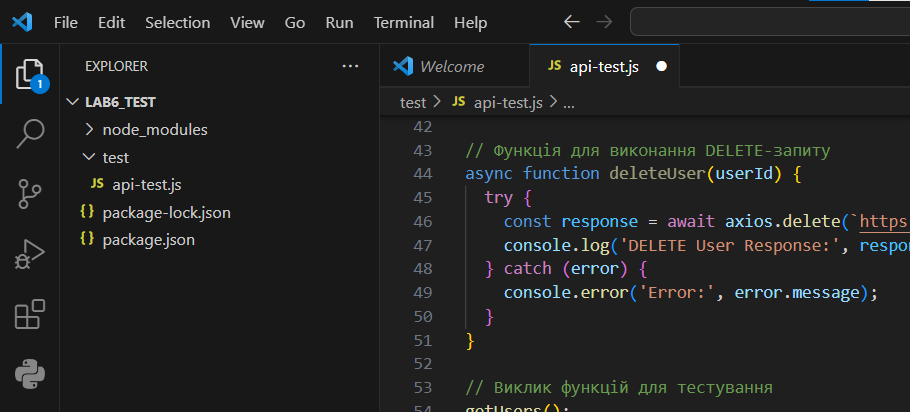
// Також можна використовувати дані, які були отримані відповідь у попередньому запиті, для подальших тестів

// createUser();

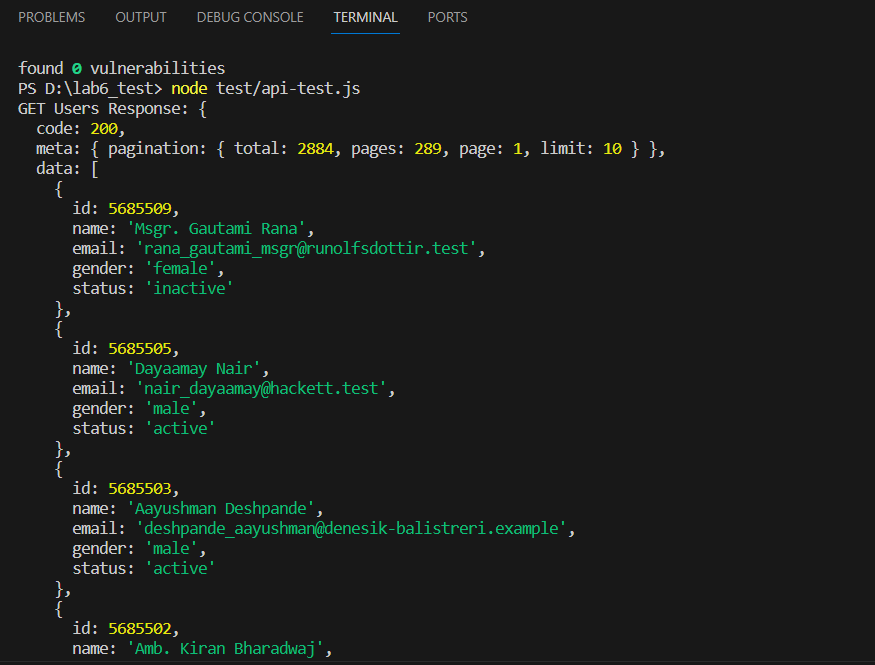
// updateUser(userId);

// deleteUser(userId);

**Структура файлів:**

****

**Результати роботи програми:**

****

**Посилання на репозиторій:**

<https://github.com/SevdaAbdullaeva/API-test_lab6.git>

**Контрольні запитання**

1. **Наведіть види тестування.**

Тестування - це процес перевірки програмного продукту на відповідність вимогам і виявлення помилок. Існує багато різних видів тестування, які можуть бути застосовані в різних ситуаціях. Ось деякі з найпоширеніших видів тестування:

* Функціональне тестування (Functional Testing): Перевірка, чи виконує програма очікувані функції і вимоги. Включає тестування інтерфейсу, функціональність і взаємодію між компонентами програми.
* Негативне тестування (Negative Testing): Спроби використовувати програму в неправильний або некоректний спосіб для перевірки її стійкості та відповідності обробці помилок.
* Тестування користувацького інтерфейсу (UI Testing): Перевірка користувацького інтерфейсу, включаючи відображення, реакцію на взаємодію користувача та специфікації дизайну.
* Тестування продуктивності (Performance Testing): Оцінка продуктивності програми, включаючи тестування навантаження, швидкості відгуку, масштабованості та інших характеристик продуктивності.
* Тестування безпеки (Security Testing): Перевірка на вразливості та виявлення помилок в забезпеченні безпеки програми. Включає тестування на вразливості, тестування на проникнення та інші види тестування безпеки.
* Автоматизоване тестування (Automated Testing): Використання автоматизованих інструментів та скриптів для виконання тестів автоматично, замість ручного тестування.
* Тестування на мобільних пристроях (Mobile Testing): Тестування програм для мобільних пристроїв, включаючи тести для iOS та Android платформ.
* Тестування веб-додатків (Web Application Testing): Тестування веб-сайтів та веб-додатків, включаючи перевірку сумісності з різними браузерами та платформами.
* Тестування відновлення (Recovery Testing): Відновлення системи після збою або відновлення даних з резервних копій.
* Тестування вмісту (Content Testing): Перевірка відповідності вмісту програми вимогам та якості.
* Тестування на відмову (Failure Testing): Тестування стійкості програми до надмірного навантаження та відмови окремих компонентів.
* Тестування сумісності (Compatibility Testing): Перевірка сумісності програми з різними операційними системами, браузерами та апаратними засобами.
* Тестування масштабованості (Scalability Testing): Визначення здатності програми масштабуватися під велике навантаження або зростання даних.
* Тестування відновлення даних (Data Recovery Testing): Перевірка здатності відновити втрачені дані після відмови.
* Тестування резервних копій (Backup Testing): Перевірка роботи процедур створення та відновлення резервних копій даних.

1. **Особливості модульного та приймального тестування.**

Модульне тестування і приймальне тестування - це два різні види тестування програмного продукту, які використовуються для різних цілей і на різних етапах розробки та тестування. Воті їх основні особливості:

**Модульне тестування (Unit Testing):**

* Ціль: Модульне тестування спрямоване на перевірку окремих компонентів (модулів) програми, таких як функції, класи, методи, або інші великі частини коду, які можна тестувати окремо.
* Ізоляція: Під час модульного тестування інші компоненти програми, крім того, який тестується, можуть бути імітовані або підроблені, щоб забезпечити ізоляцію.
* Автоматизація: Модульні тести часто автоматизовані, і вони виконуються швидко після кожного змінного коду.
* Мета: Головною метою модульного тестування є виявлення помилок і дефектів на ранніх етапах розробки і впевненість в правильному функціонуванні окремих компонентів.

**Приймальне тестування (Acceptance Testing):**

* Ціль: Приймальне тестування спрямоване на перевірку того, чи відповідає готовий програмний продукт вимогам та очікуванням клієнта або користувачів. Воно визначає, чи програма готова до впровадження або випуску.
* Ізоляція: Приймальне тестування зазвичай виконується на весь програмний продукт, як цілісну систему, і не включає імітацію або підроблення частин програми.
* Вручні тести: Приймальне тестування може включати в себе ручні тести, автоматизовані тести або комбінацію обох.
* Мета: Головною метою приймального тестування є перевірка програмного продукту на відповідність бізнес-вимогам та очікуванням користувачів і підтвердження його готовності до впровадження.

**Масштаби тестування:**

* Модульне тестування зазвичай виконується на низькому рівні - на рівні окремих компонентів або класів.
* Приймальне тестування зазвичай виконується на вищому рівні - на рівні цілого програмного продукту або системи.

**Час виконання:**

* Модульне тестування виконується під час розробки і тестування окремих компонентів, і це процес, який часто автоматизується та виконується під час кожної ітерації.
* Приймальне тестування зазвичай виконується після завершення розробки всього програмного продукту або його значних частин.

1. **Призначення бібліотек Mocha та Chai.**

Mocha і Chai - це популярні бібліотеки для тестування на платформі Node.js, які використовуються для створення та виконання тестів в JavaScript. Кожна з цих бібліотек виконує свою функцію і має свої особливості:

**Mocha:**

* Призначення: Mocha - це фреймворк для написання, організації та виконання тестів в JavaScript. Він дозволяє створювати тести з різними видами викликів, такими як синхронні, асинхронні, або чергові тести.
* Особливості:
* Підтримка BDD (Behavior-Driven Development) та TDD (Test-Driven Development) структуризації тестів.
* Зручний засіб для створення хуків перед та після виконання тестів.
* Можливість використання різних звітних форматів для результатів тестування.

**Chai:**

* Призначення: Chai - це бібліотека для засобів, які використовуються для створення асерцій (тверджень) в тестах. Вона дозволяє вам визначати, які обгрунтування ви хочете перевірити в тестах та яким чином.
* Особливості:
* Підтримка різних стилів асерцій, таких як BDD (Behavior-Driven Development) та TDD (Test-Driven Development).
* Зручний синтаксис для порівняння значень та перевірки умов.
* Можливість встановлення різних плагінів та розширень для розширення можливостей асерцій.

В разі комбінації Mocha і Chai Ви можете створювати зрозумілі, легкі для читання та детально розроблені тести. Mocha використовується для організації та виконання тестів, включаючи управління хуками, а Chai надає засоби для створення асерцій для перевірки тверджень у ваших тестах.