МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Автоматизоване тестування»

Лабораторна робота № 1

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

Перевірив: Лебідь Максим   
 Володимирович

Запоріжжя  
2023

**Завдання**

1. Ознайомитись з бібліотекою операцій над матрицями <https://github.com/zhufuge/Mtrx>
2. Виконати тестування основних функцій бібліотеки засобами бібліотек Mocha Chai.
3. Створити github репозиторій з тестами.

Бібліотека "mtrx" є JavaScript бібліотекою для роботи з матрицями та виконання операцій над ними в середовищі браузера або на стороні клієнта. Вона надає інструменти для створення, редагування, обчислення та операцій з матрицями, що може бути корисним в області веб-розробки і обробці даних.

Основні функції, які зазвичай доступні в бібліотеці "mtrx.js," включають:

* Створення матриць: Можливість створювати матриці заданого розміру і заповнювати їх значеннями.
* Операції над матрицями: Виконання різних математичних операцій, таких як додавання, віднімання, множення та ділення матриць.
* Транспонування: Зміна розмірів матриці шляхом обміну рядків і стовпців.
* Обчислення визначників: Знаходження визначника матриці.
* Інверсія матриць: Знаходження оберненої матриці.
* Розв'язання систем лінійних рівнянь: Знаходження розв'язків систем лінійних рівнянь, використовуючи матричні методи.
* Інші операції: Отримання власних значень і власних векторів, обчислення сліду матриці тощо.

**Код програми:**

const chai = require('chai');

const expect = chai.expect;

const Mtrx = require('labtest');

function subtract(matrixA, matrixB) {

  if (matrixA.length !== matrixB.length || matrixA[0].length !== matrixB[0].length) {

    throw new Error('Matrices have different sizes');

  }

  return matrixA.map((row, i) => row.map((el, j) => el - matrixB[i][j]));

}

describe('Mtrx', () => {

  describe('subtract', () => {

    it('should subtract two matrices', () => {

      const A = [

        [1, 2],

        [3, 4],

      ];

      const B = [

        [5, 6],

        [7, 8],

      ];

      const C = subtract(A, B);

      expect(C).to.deep.equal([

        [-4, -4],

        [-4, -4],

      ]);

    });

    it('should throw an error if matrices have different sizes', () => {

      const A = [

        [1, 2],

        [3, 4],

      ];

      const B = [

        [5, 6],

        [7, 8],

        [9, 10],

      ];

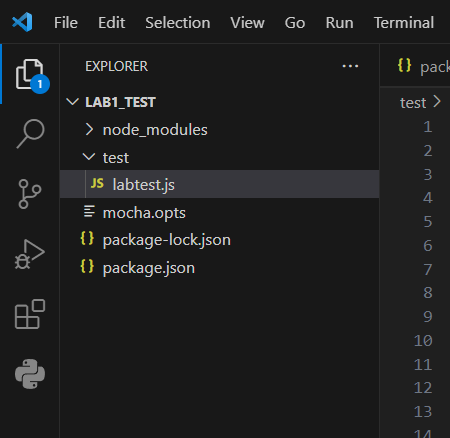
      expect(() => subtract(A, B)).to.throw(/Matrices have different sizes/);

    });

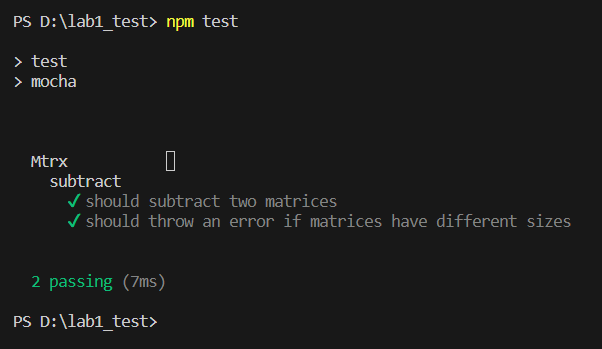
  });

});

**Структура файлів:**

****

**Результати роботи програми:**

****

**Посилання на репозиторій:**

<https://github.com/SevdaAbdullaeva/Mtrx-tests_lab1.git>

**Контрольні запитання:**

1. **Наведіть види тестування.**

Тестування - це процес оцінки програмного забезпечення для визначення його якості, правильності та здатності виконувати очікувані функції. Види тестування можуть включати в себе:

* Модульне тестування (Unit Testing): Використовується для тестування окремих модулів, функцій або класів програми. Модульні тести спрощують виявлення помилок в межах окремих компонентів.
* Інтеграційне тестування (Integration Testing): Оцінює, як різні компоненти програми взаємодіють один з одним. Тести визначають, чи працюють ці компоненти спільно правильно.
* Системне тестування (System Testing): Тестування великого функціонального блоку, включаючи всі компоненти програми. Мета - переконатися, що весь продукт працює відповідно до специфікацій.
* Приймальне тестування (Acceptance Testing): Тести, які визначають, чи відповідає програмне забезпечення вимогам замовника або користувача. Це може бути функціональним тестуванням, тестуванням сумісності, тощо.
* Функціональне тестування (Functional Testing): Оцінка функціональних можливостей програми для переконанняся, що вони відповідають специфікаціям.
* Навантажувальне тестування (Load Testing): Визначення та оцінка того, як програмне забезпечення працює при великому навантаженні, включаючи тестування продуктивності.
* Стрес-тестування (Stress Testing): Визначення меж можливостей програми шляхом високої інтенсивності навантаження або непередбачених обставин.
* Безпекове тестування (Security Testing): Оцінка програми для виявлення слабких місць, які можуть призвести до порушення безпеки даних або системи.
* Тестування регресії (Regression Testing): Перевірка програми на наявність нових помилок після внесення змін або оновлення.
* Тестування сумісності (Compatibility Testing): Перевірка, як програмне забезпечення працює на різних платформах, браузерах або пристроях.
* Тестування відмови (Failover Testing): Оцінка того, як система веде себе в разі відмови одного з її компонентів.
* Тестування відновлення (Recovery Testing): Визначення здатності системи відновити роботу після відмови.
* Тестування на забруднені дані (Data Testing): Перевірка програми на валідність та захищеність від шкідливих вхідних даних.
* Тестування користувацького інтерфейсу (User Interface Testing): Оцінка користувацького інтерфейсу на предмет зручності використання та відповідності дизайну.

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

Модульне тестування та приймальне тестування - це два різні види тестування, які мають свої особливості і призначення.

Модульне тестування:

* Об'єкт тестування: В модульному тестуванні великий програмний проект розбивається на окремі модулі, функції або класи, і кожен модуль тестується окремо.
* Обсяг тестування: Модульне тестування спрямоване на перевірку правильності роботи окремих частин програми, не враховуючи взаємодію між ними.
* Спосіб тестування: Для модульного тестування використовуються тестові дані, які зазвичай передаються у функцію або метод для перевірки правильності поведінки.
* Ціль тестування: Модульне тестування допомагає виявляти помилки та недоліки в конкретних функціях або модулях та виправляти їх раніше, коли вони не впливають на решту програми.
* Завдання тестувальника: Тестувальник повинен бути орієнтованим на деталі та здатним аналізувати внутрішню логіку коду.

Приймальне тестування:

* Об'єкт тестування: Приймальне тестування спрямоване на тестування програми в цілому, а не окремих її частин.
* Обсяг тестування: Приймальне тестування оцінює, наскільки програма відповідає вимогам замовника або користувача. Це включає в себе функціональність, продуктивність, надійність та інші аспекти.
* Спосіб тестування: Для приймального тестування використовуються тестові сценарії, які відтворюють типові ситуації, що виникають при використанні програми користувачем.
* Ціль тестування: Головною метою приймального тестування є переконатися, що програма відповідає очікуванням користувачів та вимогам до функціональності та продуктивності.
* Завдання тестувальника: Тестувальник повинен мати глибоке розуміння вимог користувача та можливостей програми в цілому. Він повинен оцінювати програму з точки зору кінцевого користувача.

Загальною метою обох видів тестування є забезпечення якості програмного забезпечення, але підходи та завдання відрізняються. Модульне тестування спрямоване на перевірку внутрішньої логіки та деталей коду, тоді як приймальне тестування оцінює, як програма виконується у реальних умовах та чи задовольняє вона потреби користувача.

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

Бібліотеки Mocha і Chai - це інструменти для тестування програмного забезпечення в середовищі JavaScript. Вони використовуються для написання і запуску тестів для JavaScript-коду та перевірки правильності його роботи. Кожна з цих бібліотек має свої призначення і взаємодоповнюється з іншими інструментами для забезпечення ефективного тестування.

Mocha:

* Призначення: Mocha - це фреймворк для тестування, який надає структуру та інструменти для написання та виконання тестів в середовищі Node.js та веб-браузерах. Він дозволяє створювати різні види тестів, такі як модульні, інтеграційні, функціональні, тощо.
* Основні особливості:
* Підтримка асинхронного тестування.
* Велика кількість плагінів та розширень для різних потреб.
* Легка налаштованість та конфігурування.
* Вивід результатів тестів у різних форматах.

Chai:

* Призначення: Chai - це бібліотека для створення тверджень (assertions) в тестових сценаріях. Вона спрощує написання тестів, оскільки дозволяє виразити очікування стосовно результатів функцій та перевірити, чи вони виконуються.
* Основні особливості:
* Підтримка різних стилів синтаксису для виразів очікувань (assertion styles).
* Підтримка ланцюгового виклику (chaining) для складних перевірок.
* Великий набір вбудованих та розширених перевірок.
* Розширені можливості налаштування та розширення функціональності.

Зазвичай Mocha та Chai використовуються разом: Mocha для створення тестових сценаріїв та запуску тестів, а Chai для створення тверджень, що перевіряють очікувані результати в тестах. Вони сприяють покращенню якості програмного забезпечення та полегшують процес розробки та тестування в JavaScript-проектах.