Хмельницький національний університет

Кафедра комп’ютерних наук

**Лабораторна робота** **№8**

з предмету:

«Основи програмної інженерії та тестування програмного забезпечення»

Виконав:

студент групи КН-22-2

Бекешко Ю.Р.

Перевірив:

Собко О. В.

Хмельницький-2024

**Завдання:**

Завдання 1. Розробити набір Unit Test за допомогою фреймворку MSTest згідно варіанту завдання. Варіант виконання завдання призначається викладачем індивідуально.

Завдання 2. Протестувати ПЗ за допомогою набору розроблених Unit Test.

Завдання 3. Згенерувати звіт з проведеного тестування.

Завдання 4. Оформити звіт з виконання лабораторної роботи, завантажити його та програмний проєкт в GitHub.

**Виконання завдання:**

Було розроблено набір Unit Test за допомогою фреймворку MSTest для програмного забезпечення «Оцінка енергетичної цінності продуктів за фотозображенням нейромережевими засобами».

Ці тести були виконані для перевірки коректності роботи окремих функціональних частин програми, таких як обчислення калорій, обробка некоректного вводу та розпізнавання продуктів:

* Тест TestCalculateCalories перевіряє правильність обчислення кількості калорій, білків, жирів і вуглеводів для продукту з вказаною вагою.
* Тест TestCalculateCaloriesForZeroGrams перевіряє, що обчислення кількості калорій, білків, жирів і вуглеводів правильно працює для продукту з нульовою вагою.
* Тест TestInvalidInput перевіряє, як програма реагує на некоректний ввід, у даному випадку, на спробу перетворення рядка з літерами на число.
* Тест TestProductNotFound перевіряє, як програма поводиться, коли продукт не знайдено в базі даних розпізнавача продуктів.

**Лістинг:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using MSAI4;

using System;

using System.Collections.Generic;

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

private Dictionary<string, Product> products;

private CalorieCalculator calorieCalculator;

private ProductRecognizer productRecognizer;

[TestInitialize]

public void TestInitialize()

{

products = new Dictionary<string, Product>

{

{ "Apple", new Product { Calories = 52, Protein = 0.3, Fat = 0.2, Carbs = 14 } },

{ "Banana", new Product { Calories = 96, Protein = 1.3, Fat = 0.3, Carbs = 27 } }

};

calorieCalculator = new CalorieCalculator();

productRecognizer = new ProductRecognizer();

}

[TestMethod]

public void TestCalculateCalories()

{

// Підготовка

var product = products["Apple"];

// Дія

var (calories, protein, fat, carbs) = calorieCalculator.Calculate(product, 150);

// Перевірка

Assert.AreEqual(78, calories, 0.01, "Неправильно обчислена кількість калорій");

Assert.AreEqual(0.45, protein, 0.01, "Неправильно обчислений вміст білків");

Assert.AreEqual(0.3, fat, 0.01, "Неправильно обчислений вміст жирів");

Assert.AreEqual(21, carbs, 0.01, "Неправильно обчислений вміст вуглеводів");

}

[TestMethod]

public void TestCalculateCaloriesForZeroGrams()

{

// Підготовка

var product = products["Apple"];

// Дія

var (calories, protein, fat, carbs) = calorieCalculator.Calculate(product, 0);

// Перевірка

Assert.AreEqual(0, calories, 0.01, "Кількість калорій повинна бути 0 для 0 грамів");

Assert.AreEqual(0, protein, 0.01, "Вміст білків повинен бути 0 для 0 грамів");

Assert.AreEqual(0, fat, 0.01, "Вміст жирів повинен бути 0 для 0 грамів");

Assert.AreEqual(0, carbs, 0.01, "Вміст вуглеводів повинен бути 0 для 0 грамів");

}

[TestMethod]

public void TestInvalidInput()

{

// Підготовка

string invalidInput = "abc";

// Дія

bool result = double.TryParse(invalidInput, out double grams);

// Перевірка

Assert.IsFalse(result, "Перетворення має зазнати невдачі для неправильного вводу");

}

[TestMethod]

public void TestProductNotFound()

{

// Підготовка

var recognizer = new ProductRecognizerStub();

// Дія

var label = recognizer.Recognize(new byte[0]);

// Перевірка

Assert.AreEqual("Unknown", label, "Продукт повинен бути невідомим, якщо не знайдено");

}

private class ProductRecognizerStub : ProductRecognizer

{

public override string Recognize(byte[] imageBytes)

{

return "Unknown";

}

}

}

}

Результат роботи тестувальних методів зображено на рисунку 1.1 Перевірку пройдено успішно.

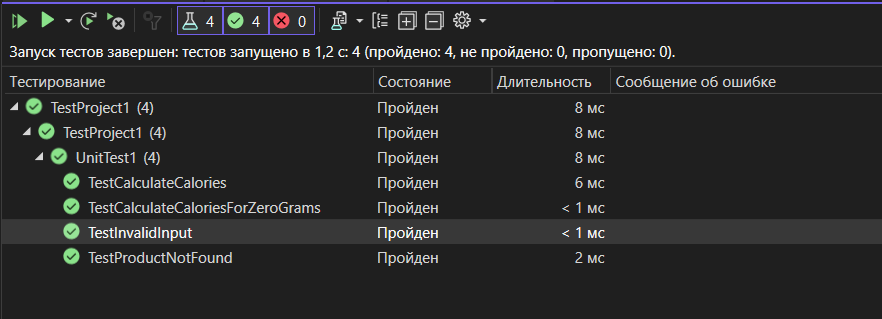


Рисунок 1.1 - Результат виконання юніт-тесту.

**Звіт з тестування Unit Test:**

Найменування тесту: TestCalculateCalories

Вхідні дані: Продукт "Apple" з вагою 150 грам.

Очікувані результати: Кількість калорій: 78, білки: 0.45, жири: 0.3, вуглеводи: 21.

Реальні результати: Кількість калорій: 78, білки: 0.45, жири: 0.3, вуглеводи: 21.

Статус тесту: Успішно

Найменування тесту: TestCalculateCaloriesForZeroGrams

Вхідні дані: Продукт "Apple" з вагою 0 грам.

Очікувані результати: Кількість калорій: 0, білки: 0, жири: 0, вуглеводи: 0.

Реальні результати: Кількість калорій: 0, білки: 0, жири: 0, вуглеводи: 0.

Статус тесту: Успішно

Найменування тесту: TestInvalidInput

Вхідні дані: Рядок "abc".

Очікувані результати: Перетворення рядка на число має зазнати невдачі.

Реальні результати: Перетворення рядка на число зазнало невдачі.

Статус тесту: Успішно

Найменування тесту: TestProductNotFound

Вхідні дані: Порожній масив байт.

Очікувані результати: Результат розпізнавання повинен бути "Unknown".

Реальні результати: Результат розпізнавання: "Unknown".

Статус тесту: Успішно

Після проведення тестування програмного продукту не було виявлено неправильно працюючих функцій. Це свідчить про те, що створена програма є відповідною до вимог специфікації та може бути використаною задля полегшення процесу підбору індивідуального раціону.

**Висновок:**

Під час лабараторної роботи №8 було зроблено тестування розробленого в рамках курсового проєкту застосунку, в додатку не було виявлено жодних проблем у його роботі. Це підтверджує, що додаток працює коректно і відповідає очікуванням.

**Контрольні запитання:**

1. Переваги автоматизації тестування:

- Зменшення часу, необхідного для виконання тестів.

- Підвищення ефективності та точності тестування.

- Повторне використання тестових сценаріїв.

- Підвищення покриття коду тестами.

- Автоматичне виявлення дефектів.

Недоліки автоматизації тестування:

- Високі витрати на розробку та підтримку тестів.

- Неможливість автоматизації всіх видів тестування.

- Потреба в постійній модифікації тестів з поновленням продукту.

- Важкість виявлення некоректностей у складних автоматизованих тестах.

2. Області застосування автоматизації тестування:

- Веб-додатки та веб-сервіси.

- Мобільні додатки.

- Вбудовані системи та IoT пристрої.

- API тестування.

- Тестування баз даних.

- Автоматизоване тестування інтеграції та взаємодії між компонентами.

3. Необхідні знання та навички для проведення автоматизованого тестування:

- Знання програмування та мови скриптування.

- Розуміння процесу розробки програмного забезпечення.

- Навички роботи з тестовими фреймворками та інструментами автоматизації.

- Уміння аналізувати та структурувати тести.

4. Інструменти для проведення автоматизованого тестування:

- Selenium WebDriver для веб-тестування.

- Appium для мобільного тестування.

- JUnit, NUnit, TestNG для модульного тестування.

- Postman, SOAPUI для API тестування.

- Jenkins, TeamCity для CI/CD.

5. Генератор тестів - це інструмент або програма, яка автоматично створює тести на основі специфікації, коду або інших вхідних даних.

6. Структура тестового набору включає в себе групу тестів, які організовані за спільною метою або характеристикою, таку як функціональність, модуль або інтеграція.

7. Додатки для автоматизованого прогону тестів включають Jenkins, Travis CI, TeamCity, CircleCI тощо.

8. Модульне тестування - це вид тестування програмного забезпечення, коли окремі модулі програми перевіряються окремо для визначення їхньої коректної роботи.

9. Основні принципи модульного тестування включають ізоляцію модулів, автоматизацію тестування, незалежність тестів та повторне використання тестових сценаріїв.

10. Переваги модульного тестування включають раннє виявлення помилок, полегшення рефакторингу та збереження документації та знань про код. Недоліки включають високі витрати на розробку та підтримку тестів.

11. Інструменти для модульного тестування включають JUnit для Java, NUnit для .NET, PHPUnit для PHP, Pytest для Python тощо.

12. Unit тест - це вид тестування, який перевіряє окремі функції, методи чи класи програми для визначення їхньої коректної роботи.

13. Структура unit тесту включає у себе три основні частини: підготовку (Arrange), виконання (Act) та перевірку (Assert).

14. Правила створення unit тестів включають ч іткість, однорідність, повноту тестування, виключення залежностей між тестами та незалежність від зовнішніх факторів.

15. Звіт з тестування Unit Test створюється для документування результатів виконання тестів. Він містить інформацію про найменування тестів, вхідні дані, очікувані результати та реальні результати тестування, а також статус кожного тесту (успішно чи неуспішно). Цей звіт допомагає у відстеженні результатів тестування, виявленні проблем та прийнятті рішень щодо подальшого розвитку програмного забезпечення.