МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Кафедра інформаційних технологій

**Лабораторна робота № 6**

З дисципліни: Інструментальні засоби розробки програмного забезпечення

Виконав:

Студент групи ІСТм-24

Йовенко Дмитро

Київ-2025

**Мета:** забезпечення якості мобільного застосунку "Трекер корисних звичок" шляхом створення юніт-тестів із використанням бібліотеки xUnit.net.

**Хід роботи**

У цій роботі реалізовано тести для перевірки функціоналу додавання, отримання та видалення звичок у класі DatabaseService, а також базові перевірки моделі Habit.

Для початку роботи з бібліотекою xUnit, її було завантажено за допомогою пакетного менеджера Nuget. Також б

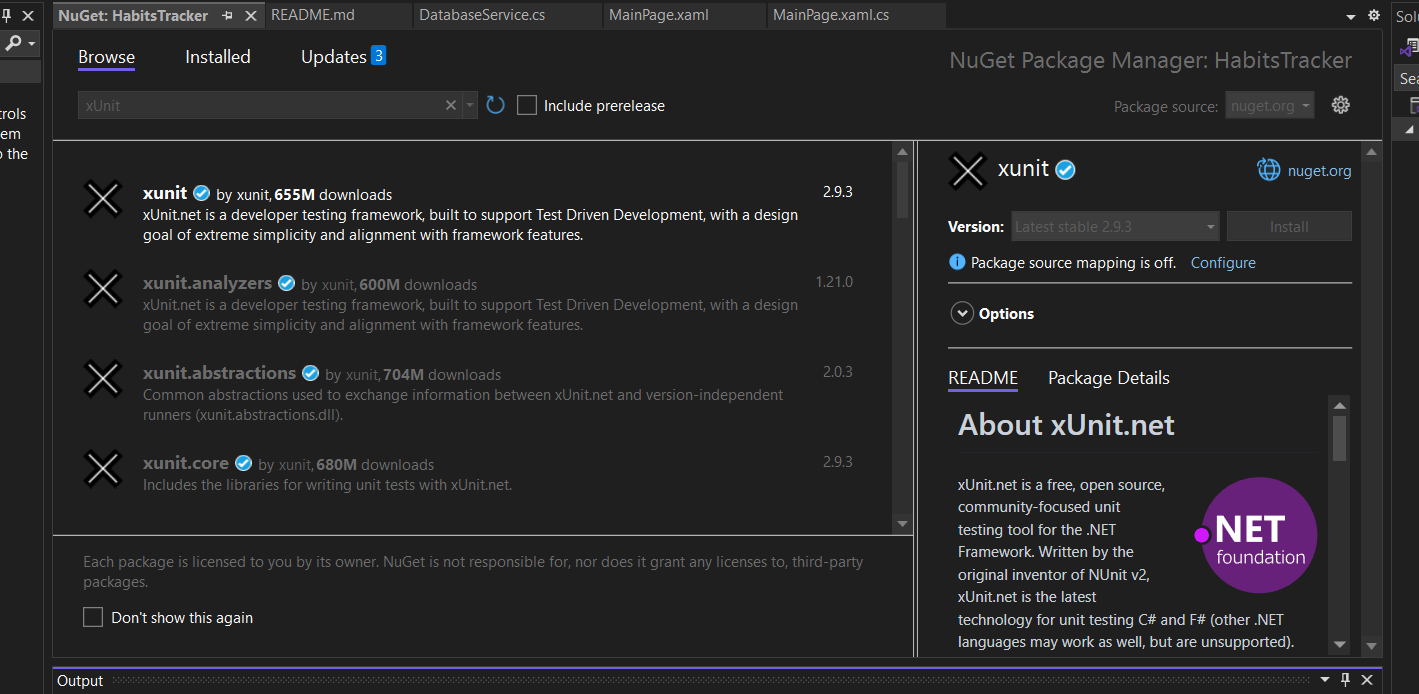


Рисунок 6.1 – Встановлення xUnit

Для створення тестових сценаціїв було створено гілку feature/tests та новий проект HabitsTracker.Test, яки у подальшому було приєдано до одного рішення.

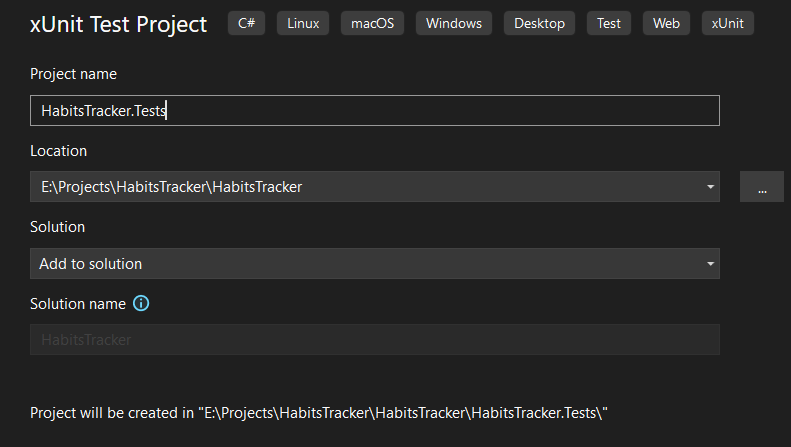


Рисунок 6.2 – Створення проетку для тестування

Важливою була конфігурація нового проетку. Оскільки, тестовий проект HabitsTracker.Test має цільову платформу net8.0, а основний проект HabitsTracker має цільові платформи net8.0-android і net8.0-windows10.0.19041.0, які не сумісні із тестовим проєктом, оскільки net8.0 не може безпосередньо посилатися на платформозалежні цілі. Тому щоб уникнути створення окремої бібліотеки і спростити інтеграцію тестів, у файл HabitsTracker.csproj внесено зміни які зображено на рисунку 6.3.

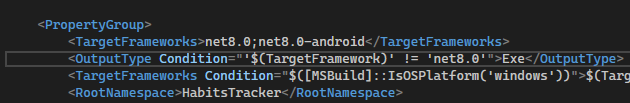


Рисунок 6.3 – Зміни конфігурації основного проекту

Ці зміни дозволили компілювати HabitsTracker як бібліотеку для net8.0, що сумісна з тестовим проєктом, зберігаючи функціональність MAUI для net8.0-android і net8.0-windows. Тестовий проєкт HabitsTracker.Test налаштовано з залежностями:

* xUnit для написання тестів.
* xUnit.runner.visualstudio для інтеграції з Visual Studio.
* Microsoft.NET.Test.Sdk для виконання тестів.
* coverlet.msbuild для аналізу покриття коду.
* SQLite-net-pcl для роботи з базою даних у тестах.

**Створення тестів для DatabaseService**

Клас DatabaseService відповідає за взаємодію з базою даних SQLite, зокрема за додавання, отримання та видалення звичок. Для тестування створено клас DatabaseServiceTests у HabitsTracker.Test, який містить шість тестів, що покривають ключові сценарії.

**Структура DatabaseServiceTests**

Тести використовують тимчасову базу даних (test\_habits.db3), яка створюється в Path.GetTempPath() і очищається перед кожним тестом для забезпечення ізоляції. Клас реалізує інтерфейс IDisposable для закриття з’єднання SQLite після кожного тесту, уникаючи блокування файлу бази даних. Розроблений тестувальний клас зображно на рисунку 6.4.

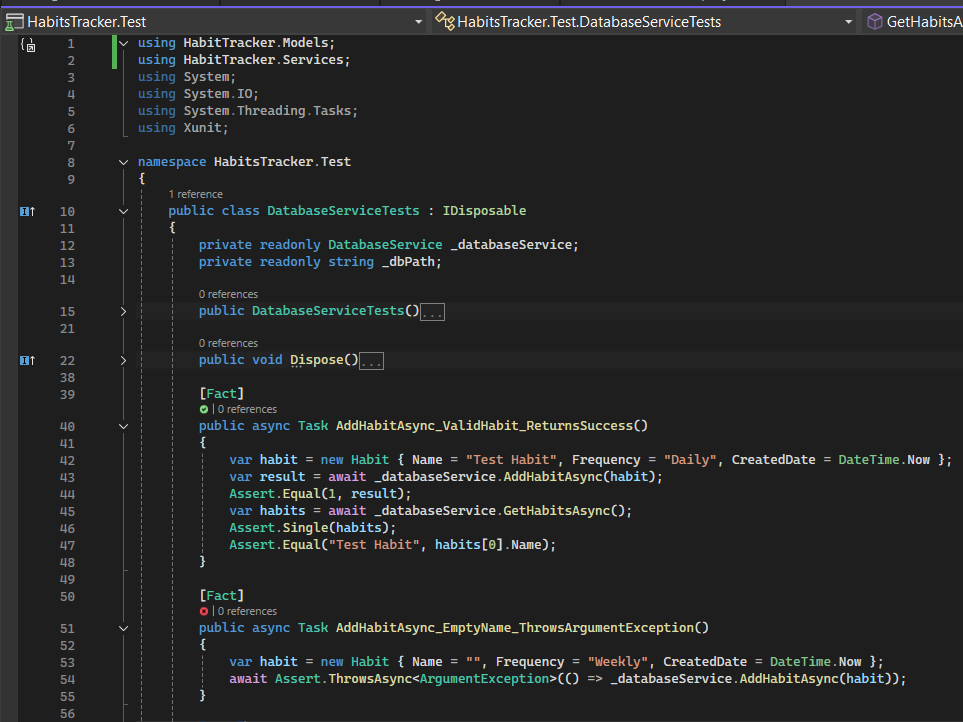


Рисунок 6.4 – Клас HabitTracker.Test

**Опис тестів**

1. **AddHabitAsync\_ValidHabit\_ReturnsSuccess**:
   * **Мета**: Перевірити, що додавання звички з валідними даними повертає успіх і коректно зберігає звичку в базі.
   * **Дії**: Створюється об’єкт Habit із назвою "Test Habit", частотою "Daily" і поточною датою. Викликається AddHabitAsync, перевіряється результат (1), а потім отримується список звичок (GetHabitsAsync) для перевірки, що додано одну звичку з правильною назвою.
   * **Очікуваний результат**: Результат вставки дорівнює 1, у базі одна звичка з назвою "Test Habit".
2. **AddHabitAsync\_EmptyName\_ThrowsArgumentException**:
   * **Мета**: Перевірити, що спроба додати звичку з порожньою назвою викликає виняток ArgumentException.
   * **Дії**: Створюється об’єкт Habit із порожньою назвою. Викликається AddHabitAsync у блоці Assert.ThrowsAsync.
   * **Очікуваний результат**: Виняток ArgumentException із повідомленням "Habit name cannot be empty".
3. **GetHabitsAsync\_EmptyDatabase\_ReturnsEmptyList**:
   * **Мета**: Перевірити, що виклик GetHabitsAsync на порожній базі повертає порожній список.
   * **Дії**: Викликається GetHabitsAsync після очищення бази в конструкторі.
   * **Очікуваний результат**: Список звичок порожній (Assert.Empty).
4. **GetHabitsAsync\_WithHabits\_ReturnsCorrectList**:
   * **Мета**: Перевірити, що GetHabitsAsync повертає коректний список звичок після їх додавання.
   * **Дії**: Додаються дві звички ("Habit 1" і "Habit 2") через AddHabitAsync. Викликається GetHabitsAsync.
   * **Очікуваний результат**: Список містить дві звички з правильними назвами.
5. **DeleteHabitAsync\_ExistingHabit\_RemovesHabit**:
   * **Мета**: Перевірити, що видалення існуючої звички видаляє її з бази.
   * **Дії**: Додається звичка, отримується список звичок, видаляється перша звичка через DeleteHabitAsync. Перевіряється результат і стан бази.
   * **Очікуваний результат**: Результат видалення дорівнює 1, база порожня.
6. **DeleteHabitAsync\_NonExistingHabit\_ReturnsZero**:
   * **Мета**: Перевірити, що спроба видалити неіснуючу звичку повертає 0.
   * **Дії**: Створюється об’єкт Habit із неіснуючим Id (999). Викликається DeleteHabitAsync.
   * **Очікуваний результат**: Результат дорівнює 0.

В результаті першого проходження тестів, маємо результат який зображно на рисунку 6.5.

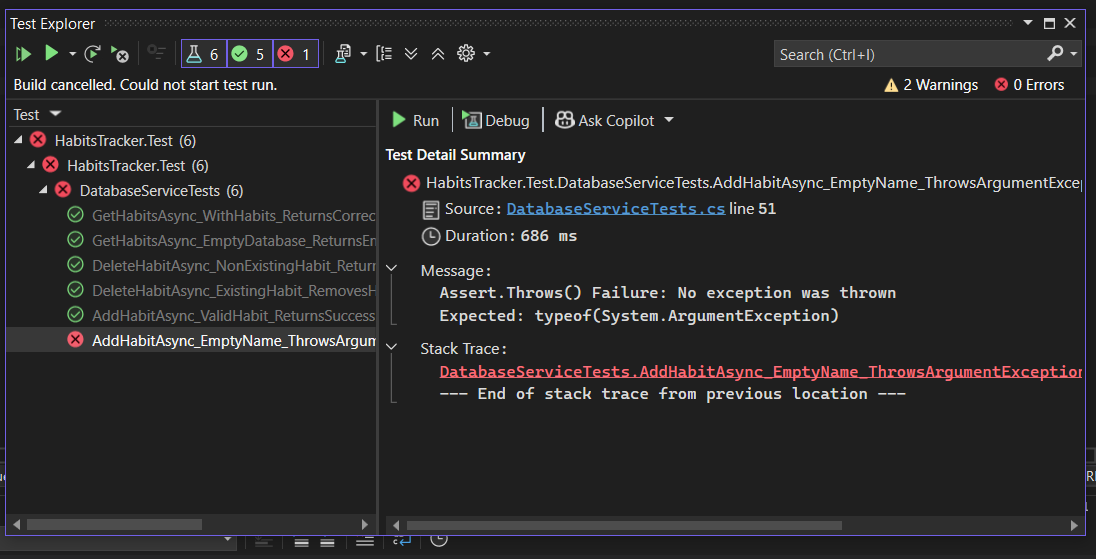
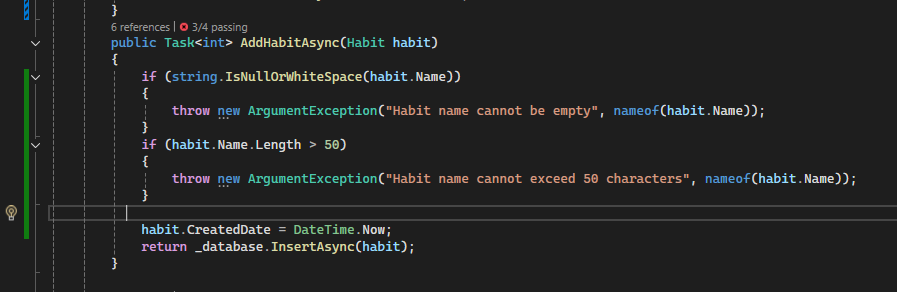


Рисунок 6.5 – Перше unit-тестування

Як видно з результатів, код пройшов 5 тестів з 6. Не пройдено тест AddHabitAsync\_EmptyName\_ThrowsArgumentException, а отже програма не повертає помилку при спробі додати до бази даних звичку без назви. Тому цей момент було виправлено у логіці Database.Service(рис.6.6). Та після змін, успічно пройдено всі тести(рис.6.7).



Рисуно 6.6 – Виправлення у Database.Service

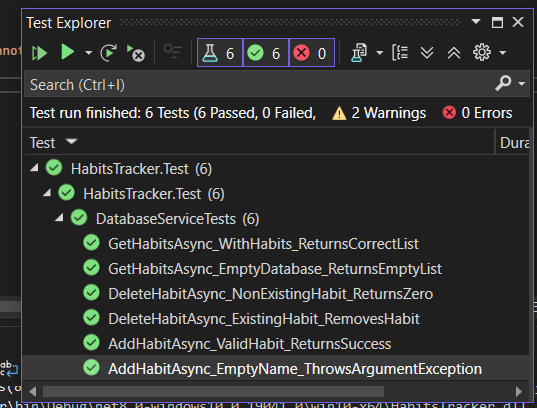


Рисунок 6.7 – Успішне проходження тестів

Наступним кроком був коміт змін у новій гілці, та створення пул реквесту на git hub. Підчас спроби злиття було виявлено конфлікт який зображно на рисунку 6.8.

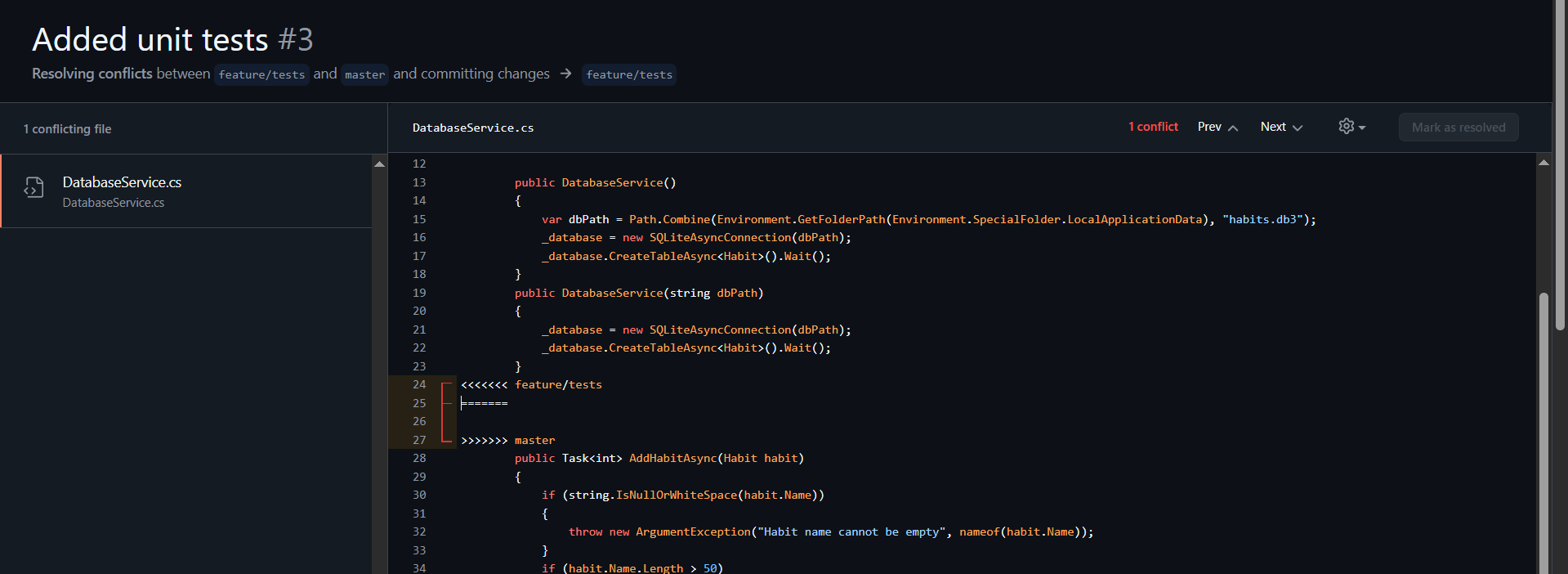


Рисунок 6.8 – Конфлік гілок master, feature/tests

Конфлікт виник через лишню порожній рядок у гілкці master, тому його було швико вирішено та проведено merge(рис.6.9).

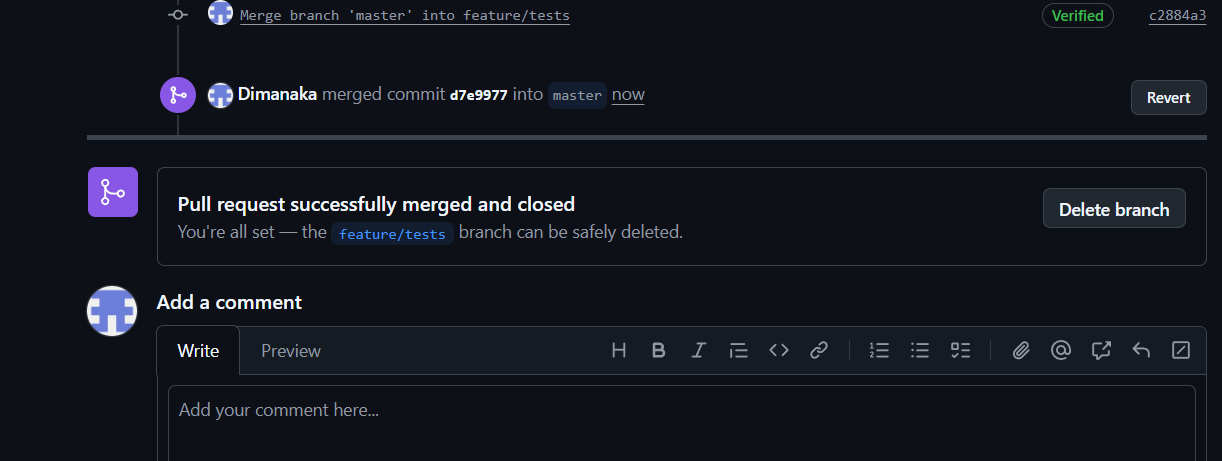


Рисунок 6.9 ­­– Злиття гілок

Посилання на репозиторій: <https://github.com/Dimanaka/GoodHabitsTracker>

**Висновок**

У результаті лабораторної роботи №6 успішно реалізовано юніт-тести для застосунку "Трекер корисних звичок" із використанням xUnit.net. Створено шість тестів для класу DatabaseService, які покрили ключові функції (додавання, отримання, видалення звичок). Виявлено та виправлено проблему з несумісністю платформ. Вивчення xUnit.net і написання тестів поглибило розуміння автоматизованого тестування. Інтеграція з GitHub забезпечила документацію та відтворюваність. Виправлення помилок підкреслило важливість ізоляції тестів і правильного управління ресурсами бази даних. Набуті навички будуть корисними для забезпечення якості в майбутніх проєктах