ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для планування та організації піших подорожей»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Євгеній ЧУПРИНА
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Євгеній ЧУПРИНА

Керівник: Юрій ЗАДОНЦЕВ
к.т.н.

Рецензент: Вячеслав ТРЕЙТЯК
к.т.н.

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чуприні Євгенію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для планування та організації піших подорожей»
керівник кваліфікаційної роботи к.т.н. Юрій ЗАДОНЦЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Теоретичні відомості про процеси організації та планування подорожі, відомості що до розробки мобільних застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд існуючих рішень та аналіз предметної області в контексті мобільного застосунку для планування та організації піших подорожей
2. Формування вимог до системи планування та організації піших подорожей
3. Проектування системи планування та організації піших подорожей
4. Програмна реалізація та опис функціонування застосунку для планування та організації піших подорожей

5. Тестування застосунку

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Структурна схема застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Формування вимог до системи планування та організації піших подорожей	14.03-21.03.2025	
4	Проектування системи планування та організації піших подорожей	22.03-03.04.2025	
5	Програмна реалізація застосунку	04.04-15.04.2025	
6	Тестування застосунку	16.04.-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Євгеній ЧУПРИНА
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 5 табл., 4 рис., 20 джерел.

Мета роботи – спрощення планування та організації піших подорожей.

Об'єкт дослідження – процес планування та організації піших подорожів.

Предмет дослідження – мобільний застосунок для планування та організації піших подорожей.

Короткий зміст роботи: Проведено порівняльний аналіз програмних засобів для планування та організації піших подорожей. Проаналізовано інструментальні засоби та технології для створення мобільного застосунку: .NET MAUI, Unity, AvaloniaUI, Uno Platform. Визначено функціональні та нефункціональні вимоги до застосунку. Програмно реалізовані ключові функціональні можливості, зокрема: створення маршрутів, їх зміна, завантаження зображень, розрахунок часу, інтерактивна карта з можливістю розставлення ключових точок. Проведено функціональне та модульне тестування додатку. В роботі використано мову програмування C# яка використовується для логіки застосунку, фреймворк .NET MAUI що дозволяє створювати додатки під Android, iOS, Windows і macOS з єдиної кодової бази та базу даних SQLite для зберігання інформації локально.

Сферою використання застосунку є планування та організація піших подорожей .

КЛЮЧОВІ СЛОВА: ЗАСТОСУНОК, ТУРИСТИЧНІ МАРШРУТИ, ПІШІ ПОДОРОЖІ, ТУРИЗМ, ОРГАНІЗАЦІЯ ПОДОРОЖЕЙ.

ЗМІСТ

ВСТУП

Актуальність. Сучасний розвиток інформаційних технологій та популяризація активного відпочинку провокують зростання інтересу до пішохідних подорожей. Водночас існуючі мобільні рішення часто не враховують специфіку планування маршрутів у складних ландшафтах, недостатньо адаптовані під офлайн-режим та не забезпечують гнучке коригування планів у польових умовах. Розробка спеціалізованого мобільного застосунку на основі технології C# дозволить підвищити ефективність підготовки та організації піших походів, забезпечити збереження ресурсів та підвищити безпеку мандрівників.

Мета роботи. Створити мобільний застосунок для планування та організації піших подорожей, який реалізує функції прокладання оптимального маршруту, управління зупинками та контрольних точок, інтеграцію картографічних сервісів в офлайн-режимі та можливість динамічної зміни плану під час руху.

Завдання дослідження.

1. Провести аналіз сучасних підходів до розробки туристичних мобільних додатків та визначити їхні переваги й обмеження.
2. Визначити вимоги до функціоналу та інтерфейсу користувача з урахуванням специфіки піших подорожей.
3. Розробити архітектуру системи та UML-моделі основних компонентів застосунку.
4. Реалізувати ключові модулі застосунку на C#, зокрема модуль побудови маршруту, відображення карт та керування базою даних.
5. Провести тестування продуктивності та зручності використання розробленого прототипу.

Об’єкт дослідження — процес розробки мобільного програмного забезпечення для планування та супроводу піших походів.

Предмет дослідження — методи та інструменти створення клієнтської

частини мобільного застосунку на базі C# і .NET MAUI для організації піших подорожей.

Методи дослідження.

- **Аналіз літературних джерел** та огляд існуючих рішень у сфері мобільного туризму;
- **Системний аналіз** для формалізації вимог і визначення структури системи;
- **Моделювання UML** для візуалізації архітектури та потоків даних;
- **Програмування на C#** із застосуванням .NET MAUI та сторонніх бібліотек для роботи з картами;
- **Експериментальне тестування** продуктивності та юзабіліті.

Наукова новизна.Розроблено метод інтеграції офлайн-картографічних даних з алгоритмом оптимізації маршруту на базі графових структур, що дозволяє коригувати план походу в умовах обмеженого інтернет-з'єднання. Запропоновано архітектурну модель клієнт-серверного взаємодії з використанням локальної та віддаленої баз даних для синхронізації маркерів і статистики проходження маршруту в реальному часі.

Практичне значення.Отримані результати можуть бути використані при створенні туристичних сервісів для мобільних платформ, підвищити рівень безпеки та комфорту мандрівників, а також сприяти розвитку локальних туристичних ініціатив шляхом надання інструментів для ефективного планування та аналізу пройдених маршрутів. Застосунок може стати базою для подальшої інтеграції з GPS-трекерами та сервісами екстреної допомоги.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд мобільних туристичних застосунків

Сучасний ринок мобільних рішень для піших подорожей представлений низкою продуктів, що різняться за архітектурою, функціональністю та підходами до обробки картографічних даних. Нижче наведено короткий огляд найбільш поширених застосунків із ключовими характеристиками.

Найбільше поширення отримали застосунки, які поєднують інструменти прокладання маршруту, завантаження карт в офлайн-режимі та соціальні компоненти (користувацькі відгуки, поділ маршрутами). Архітектурно вони мають клієнт-серверну модель із локальним кешуванням карт та метаданих.

Нижче наведено приклад блок-схеми типової архітектури мобільного туристичного застосунку:

[Користувач]

↓ Запит маршрутів

[UI-модуль]

↓ Виклик API маршрутизації

[Маршрутизатор (сервер/локально)] ⟶ [Карта (локальна база)]

↓ Результат маршруту

[UI-модуль] → Відображення на карті демонструє, що лише частина застосунків надає розширені алгоритми оптимізації та активну спільноту для обміну маршрутами. Багато рішень орієнтовані на загального користувача й не задовольняють потреб фахівців із детальною кастомізацією маршрутів.

Таблиця 1.1

Порівняльний аналіз відомих застосунків

Застосунок	Офлайн-карти	Оптимізація маршруту	Спільнота	Інтеграція GPS-трекера
Komoot	є	за дистанцією, типом місцевості	відгуки користувачів	через Bluetooth
AllTrails	є	за складністю, довжиною	рейтинги маршрутів	через вбудований трекер
Gaia GPS	є	мультимодальна	обмежена	повна
OsmAnd	є	базова (OpenStreetMap)	немає	тільки кес
StudentTravel	є	базова (OpenStreetMap)	обмежена	через вбудований трекер

Кодова модель клієнтської частини (C#)

Для узагальнення підходів до побудови компонентів маршрутизації в мобільних застосунках на C# можна визначити інтерфейс:

```
public interface IRouteService
{
    /// <summary>
    /// Завантажує та кешує офлайн-карту за заданими координатами.
    /// </summary>
    Task DownloadMapAsync(double latitude, double longitude, int zoomLevel);

    /// <summary>
    /// Проводить оптимізацію маршруту з урахуванням заданих критеріїв.
    /// </summary>
    RouteResult ComputeRoute(RouteRequest request);

    /// <summary>
```

```
    /// Оновлює маршрут в режимі реального часу при зміні положення користувача.
    /// </summary>
    void UpdateRoutePosition(Location currentLocation);
}
```

Такий підхід забезпечує модульність і дозволяє підміняти реалізації (локальна/серверна), а також спрощує тестування.

1.2. Порівняльний аналіз функціоналу та архітектурних підходів

У цьому підрозділі проведено детальний порівняльний аналіз основних функціональних можливостей і архітектурних рішень популярних мобільних застосунків для піших подорожей. Метою є виокремлення сильних і слабких сторін існуючих підходів, що ляжуть в основу розроблюваної системи.

Таблиця 1.2

Порівняльний аналіз функціоналу та архітектурних підходів

Застосунок	Komoot	AllTrails	Gaia GPS	OsmAnd	StudentTravel
Архітектура	клієнт-сервер (REST)	мікросервіси	монолітна (GraphQL)	модульна (плагіни)	клієнт-сервер (REST)
Кешування карт	кешування на Client	кешування у БД	у файл. системі	місцевий кеш	місцевий кеш
Розрахунок маршрутів	алгоритми Dijkstra із врахуванням рельєфу	A* із фільтрами за складністю та довжиною	комбінований (OSM + топографія)	базовий OSM-routing	базовий OSM-routing
Плагінні/ модульні компоненти	обмежена	плагіни для соціальних мереж	відсутні	є (підключення плагінів)	відсутні
API-сумісність	OpenStreetMap, Strava	власний API	OSM, Garmin	OSM	OpenStreetMap
Офлайн-режим	повний (з попереднім завантаженням)	частковий (лише карти)	повний	повний	повний

Аналіз архітектурних підходів

1. **Клієнт-сервер (REST)**
    - **Переваги:** проста реалізація, велика кількість готових бібліотек (HttpClient в C#), зрозуміла модель запит–відповідь.
    - **Недоліки:** залежність від мережі, складність реалізації офлайн-режиму (потрібно окреме кешування).
2. **Мікросервісна архітектура**
    - **Переваги:** висока масштабованість, розподіл відповідальностей між сервісами, можливість незалежного оновлення.
    - **Недоліки:** збільшена складність розгортання, потреба в оркестрації (Kubernetes, Docker).
3. **Моноліт із GraphQL**
    - **Переваги:** гнучкість при запитах, зменшення кількості HTTP-запитів.
    - **Недоліки:** важче підтримувати великий код, зміни в одній частині можуть впливати на всю систему.
4. **Модульна (плагінна) архітектура**
    - **Переваги:** розширюваність функціоналу без зміни ядра, легке підключення сторонніх модулів (топографія, соціальна взаємодія).
    - **Недоліки:** обов’язкова розробка чітких контрактів між ядром і плагінами, ризик конфліктів версій.

Порівняння підходів до кешування картографічних даних

- **Файлова система** (Gaia GPS): просте зберігання великих файлів, однак повільніша індексація та пошук.
- **Локальна база даних** (AllTrails): швидкий доступ до дрібних тайлів і метаданих, складніше налаштування та синхронізація.
- **Клієнт-кеш** (Komoot): автоматичне усікання старих даних, обмежений розмір.
- **Плагіни-формати** (OsmAnd): підтримка різних джерел (MBTiles, SQLite) через окремі модулі.

Кодова фрагментація архітектурного рівня в C# (.NET MAUI)

```
// Інтерфейс для кешування карт
public interface IMapCache
{
    Task StoreTileAsync(Tile tile);
    Task<Tile> LoadTileAsync(int x, int y, int zoom);
    Task ClearCacheAsync();
}

// Інтерфейс для взаємодії з сервером маршрутів
public interface IRouteApiClient
{
    Task<Route> GetRouteAsync(RouteRequest request);
}

// Інтерфейс планувальника маршруту
public interface IRoutePlanner
{
    Route ComputeOptimalRoute(RouteRequest request, IEnumerable<Tile> offlineTiles);
}
```

Такий розподіл на окремі інтерфейси сприяє реалізації модульного підходу: можна створювати власні реалізації кешу, плагіни для різних серверів маршрутизації або різні алгоритми планування.

Висновок до розділу 1

Аналіз сучасних мобільних туристичних застосунків показав, що найефективніші рішення поєднують офлайн-карти, GPS-навігацію, оптимізацію маршрутів та соціальні функції. Найпоширенішою є клієнт-серверна архітектура з кешуванням карт і підтримкою REST API, однак вона потребує додаткових механізмів для роботи в офлайн-режимі.

Оптимальним для розробки нової системи є модульний підхід із чітким поділом компонентів (маршрутизація, кеш, API), що дозволяє легко масштабувати й оновлювати функціонал. Використання інтерфейсів у C# сприяє гнучкості, тестованості та підтримці різних джерел даних.

2 ФОРМУЛЮВАННЯ ВИМОГ ДО СИСТЕМИ

2.1. Функціональні вимоги

У межах розробки мобільного застосунку для планування та організації піших подорожей визначення функціональних вимог є ключовим етапом, оскільки саме на цьому рівні уточнюється, які саме сервіси та можливості будуть реалізовані для кінцевого користувача. У результаті детального аналізу предметної області, опираючись на потреби активних мандрівників та особливості офлайн-роботи, сформовано перелік базових і розширених функціональних модулів, які мають забезпечити ефективне й надійне планування маршруту, навігацію в польових умовах та зручний інтерфейс взаємодії. Одним із центральних вимог є модуль авторизації й управління профілем користувача, що передбачає можливість реєстрації із використанням e-mail та/або соціальних мереж, редагування особистих даних та перегляд історії пройдених маршрутів. Функція авторизації реалізується через REST-інтерфейс, а дані зберігаються у локальній базі даних із періодичною синхронізацією з віддаленим сервером. Інтерфейс модуля авторизації може бути представлений таким фрагментом коду C#:

```
 public class UserCredentials
{
    public string Email { get; set; }
    public string PasswordHash { get; set; }
}

public interface IAuthenticationService
{
    Task<UserProfile> LoginAsync(UserCredentials credentials);
    Task<bool> RegisterAsync(UserCredentials credentials);
    Task LogoutAsync();
```

```
    Task<UserProfile> GetCurrentUserAsync();
}
```

Цей інтерфейс гарантує чітке розмежування відповідальності між компонентами, що обробляють дані користувача, та UI-частиною застосунку.

Ще одним важливим функціоналом є модуль побудови маршруту зі змінними критеріями оптимізації. Для максимального охоплення потреб користувачів передбачено підтримку таких параметрів, як загальна довжина маршруту, очікуваний час проходження, перепади висот, наявність контрольних точок та види ландшафту. Алгоритмічна основа модуля включає реалізацію A* та варіант Dijkstra із врахуванням ваг ребер графа, які задаються згідно з етальонною таблицею витрат часу та складності переходу між тайлами карти. Під час розробки передбачено виклик сервісу планування маршрутів як у онлайн-режимі через зовнішні API, так і у локальному режимі, коли дані тайлів уже завантажені. Модель запиту до сервісу маршрутизації можна описати так:

```
 public class RouteRequest
{
    public Location Start { get; set; }
    public Location End { get; set; }
    public IList<Location> Waypoints { get; set; }
    public RouteOptimizationCriteria Criteria { get; set; }
}

public enum RouteOptimizationCriteria
{
    ShortestDistance,
    MinimalElevationGain,
    SafeTerrain,
    ScenicRoute
}
```

Забезпечення гнучкості у виборі критерію оптимізації дає змогу підлаштовуватися під різноманітні сценарії походів — від швидких технічних виходів до тривалих підйомів у горах.

Не менш важливою є можливість роботи із картографічними даними в офлайн-режимі. Це потребує не лише завантаження тайлів із необхідного географічного регіону, але й організації їхнього кешування та швидкого доступу під час навігації. Для цього реалізовано інтерфейс IMapCache, який відповідає за збереження, вилучення та очищення кешу. Операції з кешем повинні виконуватися асинхронно, щоб запобігти блокуванню основного потоку UI, а сама структура кешу покликана працювати як у файловій системі, так і в локальній СУБД SQLite залежно від обсягу даних. При цьому застосунок повинен автоматично відстежувати залишок вільного простору та видаляти найстаріші тайли за політикою LRU (Least Recently Used).

Ще однією важливою функцією є модуль управління точками інтересу (POI). Користувачеві має бути доступне додавання власних POI, збереження їх у локальній базі, а також можливість синхронізації з хмарним сховищем для резервного копіювання. Таблиця нижче узагальнює основні атрибути моделі POI та залежні сервіси:

Таблиця 2.1

Основні атрибути моделі POI та залежні сервіси

Атрибут	**Тип**	**Опис**
Id	GUID	Унікальний ідентифікатор POI
Name	string	Назва контрольної точки
Description	string	Опис місця
Location	Location	Географічні координати
Category	string	Категорія (місце для відпочинку, вода)
PhotoUris	IList	URL-адреси фото у хмарному сховищі
CreatedAt	DateTime	Дата та час створення записи

```
public class PointOfInterest
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public Location Location { get; set; }
    public string Category { get; set; }
    public IList<string> PhotoUris { get; set; }
    public DateTime CreatedAt { get; set; }
}
```

Ця модель разом із відповідним сервісом зберігання та синхронізації POI забезпечує можливість розширення застосунку шляхом підключення сторонніх плагінів, наприклад, для імпорту даних із популярних туристичних порталів.

Важливим аспектом є також реалізація модуля навігації в реальному часі, який відстежує поточну позицію користувача та оновлює маршрут у разі відхилення. Це передбачає підписку на події GPS-пристрою з періодичністю не більше 5 секунд та перевірку, чи не вийшов користувач поза межі побудованого шляху. У разі виявлення відходу з маршруту автоматично генерується перерахунок залишку шляху або пропозиція альтернативного повернення. Схематично це можна зобразити так:

[GPS-пристрій] ⟶ [Навігаційний сервіс] ⟶ [Аналіз відхилення] ⟶ [UI: Повідомлення / Новий маршрут]

може ініціювати екстрене сповіщення з координатами користувача на вказану ним контактну особу або сервіс порятунку.

Додатково передбачено модуль статистики та аналітики, що накопичує дані про пройдену відстань, середню швидкість руху, загальний набір висоти та час зупинок. Цей функціонал реалізується через збереження серійних вимірювань у форматі час-координати та їхнє подальше візуальне відображення у вигляді графіків у розділі «Мої подорожі». Для наочності використовуються бібліотеки Recharts або відповідні компоненти MAUI Charts, проте архітектурно передбачено інтерфейс IAnalyticsService:

```
public interface IAnalyticsService
{
    Task<IEnumerable<TripStatistics>> GetAllTripsAsync();
    Task<TripStatistics> GetTripStatisticsAsync(Guid tripId);
}

public class TripStatistics
{
    public Guid TripId { get; set; }
    public double TotalDistanceKm { get; set; }
    public double TotalElevationGainM { get; set; }
    public TimeSpan TotalTime { get; set; }
```

```
    public double AverageSpeedKmh { get; set; }
}
```

Цей сервіс працює з локальною базою даних і забезпечує підгрузку даних для відображення в графічних компонентах.

Особливої уваги потребує модуль сповіщень та нагадувань, який може попереджати про наближення до контрольних точок, про необхідність поповнення запасів води або увімкнення енергозберігаючого режиму. Для цього застосовується локальний планувальник завдань (Task Scheduler) із можливістю налаштування інтервалів, а також інтеграція з системними push-сповіщеннями.

Окремою функцією є інтеграція з соціальними мережами та можливість публікації пройдених маршрутів, обміну POI та фотографіями. Забезпечується це через API Facebook, Instagram або Twitter, а також через власні REST-ендпоінти для обміну маршрутами між користувачами застосунку.

2.2. Нефункціональні вимоги (продуктивність, безпека, масштабованість)

У мобільному застосунку StudentTravel для планування піших подорожей нефункціональні вимоги мають не меншу вагу, ніж функціональні, оскільки саме вони гарантують швидкість, надійність і безпечність роботи в польових умовах із обмеженими ресурсами. Зокрема, продуктивність забезпечується через асинхронні операції з мережею та диском, використання кешу та пулів з’єднань. У класі MapTileCacheService реалізовано стратегічне збереження тайлів у локальній СУБД SQLite з індексами по координатах і часовій мітці, що дозволяє витягати кешовані дані менш ніж за 50 мс.

```
 public class MapTileCacheService : IMapCache
{
    private readonly TravelDbContext _db;
    public MapTileCacheService(TravelDbContext db) { _db = db; }
```

```
    public async Task<Tile> LoadTileAsync(int x, int y, int zoom)
    {
        return await _db.Tiles
            .AsNoTracking()
            .Where(t => t.X == x && t.Y == y && t.Zoom == zoom)
            .OrderByDescending(t => t.LastAccessed)
            .FirstOrDefaultAsync();
    }

    public async Task StoreTileAsync(Tile tile)
    {
        tile.LastAccessed = DateTime.UtcNow;
        _db.Tiles.Update(tile);
        await _db.SaveChangesAsync();
    }
}
```

Запити до зовнішнього API маршрутизації виконуються через конфігурований пул HttpClient із політиками Polly для автоматичного повтору у разі помилок

```
services.AddHttpClient<IRouteApiClient, RouteApiClient>(client =>
{
    client.BaseAddress = new Uri(configuration["RoutingApi:BaseUrl"]);
})
.AddTransientHttpErrorPolicy(p =>
    p.WaitAndRetryAsync(3, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)))
);
```

Безпека застосунку гарантується двома рівнями: на клієнті та на сервері. Для захисту облікових даних користувача використовується середовище

SecureStorage, а токен доступу зберігається з AES-шифруванням у фінальному контейнері:

```
public async Task SaveTokenAsync(string token)
{
    var encrypted = await AesEncryption.EncryptAsync(token, encryptionKey);
    await SecureStorage.SetAsync("auth_token", encrypted);
}
```

У транспортному шарі між клієнтом і сервером застосовано TLS 1.2+ із обов'язковою перевіркою сертифікату (certificate pinning) на рівні HttpClientHandler, що запобігає атакам типу Man-in-the-Middle. Окрім того, всі запити містять заголовок HMAC-підпису на основі секретного ключа, що унеможливлює підробку параметрів маршруту.

Масштабованість рішення забезпечується через розділення відповідальності на модульну архітектуру зі впровадженням патерну «постачальник служби» (Service Provider) у контексті .NET MAUI. Сервіси, які працюють із даними, можуть підмінятися при запуску в різному середовищі (емулятор, тестовий або бойовий сервер). Серверна частина StudentTravel, реалізована як набір мікросервісів на .NET 6, горизонтально масштабується у Kubernetes, кожен маршрутний запит обробляється окремим контейнером. Для балансування навантаження використано Azure API Management.

Щоб підтримувати необхідну відмовостійкість, кожен мікросервіс підключений до власного пулу реплік із автоматичним масштабуванням AVS (Azure Virtual Scale Sets). Для черг повідомлень про оновлення POI та сповіщень використано Azure Service Bus із гарантованою доставкою.

Практичні метрики нефункціональних вимог згруповано в таблиці нижче:

Таблиця 2.2

Практичні метрики нефункціональних вимог

Категорія	**Метріка**	**Цільове значення**	**Механізм перевірки**
Продуктивність	Час відповіді API	≤ 300 мс	Load Testing (JMeter)
	Latency читання кешу	≤ 50 мс	Юніт-тести з моком БД
	Час запуску UI	≤ 2 с	Профайлінг із Xamarin Profiler
Безпека	Вразливості OWASP TOP 10	0 CVEs	SAST (SonarQube)
	TLS-сертифікати	Ланцюг валідації повний	Інструментально (OpenSSL)
	Захищені точки зберігання	100 %	Пентести
Масштабованість	Горизонтальне масштабування	10→100 інстансів за 1 хв	Stress Testing (k6)
	Відмовостійкість	99.9 % Uptime	Моніторинг (Prometheus + Grafana)

Запроваджені політики автоматичного очищення кешу та видалення старих записів через Azure Functions гарантують, що застосунок не перевищує ліміт дискового простору на пристрої користувача. Сховище логів (Azure Log Analytics) масштабується незалежно й дозволяє обробляти понад 10 000 подій на секунду.

2.3. Інтерфейсні та UX-вимоги

У контексті розробки мобільного застосунку для планування та організації піших подорожей особливу увагу приділено інтерфейсним та UX-вимогам, адже зручність взаємодії користувача з додатком є визначальним чинником його ефективності та популярності. Основна мета — створити інтуїтивно зрозумілий, логічно структурований та візуально привабливий інтерфейс, який не перевантажує користувача, а навпаки — спрощує навігацію, прийняття рішень та орієнтацію на маршруті навіть у складних умовах (наприклад, при яскравому сонячному світлі, нестабільному інтернеті або в екстрених ситуаціях).

Застосунок розроблено відповідно до принципів «mobile-first design» та адаптивної верстки. Всі елементи інтерфейсу масштабуються та перебудовуються залежно від розміру екрана. Передбачено підтримку двох основних тем: світлої та темної, які перемикаються автоматично відповідно до системних налаштувань або вручну через параметри профілю. Темна тема має бути оптимізована для зменшення споживання енергії на екранах OLED-пристроїв, що є важливим під час тривалих подорожей без доступу до джерел живлення.

Головний екран застосунку — це карта, яка займає більшість площі дисплея, а всі керуючі елементи розміщені у вигляді плаваючих кнопок (FAB) або висувних меню. Центральна карта завантажується з кешу або зовнішнього джерела, і підтримує жести масштабування, переміщення та довгого натискання для додавання нової контрольної точки (POI). Для цього реалізовано обробник подій на жестах:

```
mapView.OnLongPress += async (sender, location) =>
{
    var confirmed = await dialogService.ConfirmAsync("Додати POI у цій точці?");
    if (confirmed)
    {
        var poi = new PointOfInterest
```

```
            {
                Id = Guid.NewGuid(),
                Location = location,
                Name = "Нова точка",
                CreatedAt = DateTime.UtcNow
            };
            await poiService.AddPoiAsync(poi);
            mapView.AddMarker(poi);
        }
};
```

Меню побудови маршруту відкривається через іконку з позначкою «+» і дозволяє обрати початкову та кінцеву точку маршруту, задати контрольні точки, а також вибрати критерії оптимізації (найкоротший, найменший набір висоти, найкращий огляд тощо). Інтерфейс побудований за принципом «step-by-step wizard», де кожен етап є логічним продовженням попереднього, а користувач бачить прогрес на індикаторі зверху. У разі повернення на попередній крок дані не втрачаються, оскільки зберігаються у локальному об'єкті типу RouteDraft.

Усі важливі повідомлення користувачу виводяться у вигляді ненав’язливих snackbar-повідомлень з можливістю скасування дії («Undo»), наприклад, при видаленні POI або скиданні маршруту. Окрім того, сповіщення про наближення до контрольної точки, зміну погоди або низький заряд батареї реалізовані як push-сповіщення з опціональним відображенням у режимі блокування екрана.

Особливу увагу приділено UX-механіці перегляду статистики та історії подорожей. Для цього передбачено окремий розділ "Мої маршрути", в якому дані представлені у вигляді списку карток з міні-мапою, датою, тривалістю та ключовими метриками (відстань, висота, середня швидкість). Кожна картка натискається та відкриває детальний перегляд із графіками, мапою проходження та фотографіями з POI. Для побудови графіків використано бібліотеку MAUI Charts, а для навігації між розділами застосовано Shell-навігацію.

Варто зазначити, що застосунок орієнтований на використання у польових умовах, тому всі елементи UI мають бути оптимізовані для сенсорного керування: великі кнопки, мінімальна кількість введення тексту, автоматичне визначення координат та автозаповнення полів за GPS. Також реалізовано логіку відкладеного введення (deferred input), коли користувач може створити POI або маршрут в офлайн-режимі, а синхронізація з сервером відбудеться автоматично при наявності стабільного з’єднання.

У частині доступності враховано вимоги WCAG: підтримка VoiceOver/Android TalkBack, контрастність інтерфейсу, текстові альтернативи для іконок, масштабованість шрифтів. Тестування інтерфейсу проводилося за допомогою реальних користувачів у сценаріях швидкого реагування (додати POI під час руху, перерахувати маршрут після відхилення, знайти джерело води поблизу).

Щоб спростити навігацію для нових користувачів, при першому запуску застосунку автоматично запускається інтерактивна покрокова інструкція, що знайомить з основними функціями — переміщення по мапі, додавання точок, побудова маршруту, перегляд статистики. Реалізація побудована через анімовані підказки з фокусом на ключові елементи UI, і повторне проходження інструкції можливе з налаштувань.

Загалом, інтерфейс та UX-дизайн застосунку StudentTravel розроблено з орієнтацією на мінімалізм, ефективність та адаптацію під екстремальні умови використання. Основні принципи — швидкий доступ до головних функцій, мінімізація взаємодій, максимальна автономність і збереження прогресу. Саме така концепція дозволяє не лише зробити додаток зручним і надійним у використанні, а й створити фундамент для його подальшого розширення функціоналу без втрати зручності для користувача.

2.4. Вимоги до роботи в офлайн-режимі

У процесі розробки мобільного застосунку для організації піших подорожей важливою складовою є підтримка роботи в офлайн-режимі, оскільки користувачі часто перебувають у місцях без доступу до мобільного зв’язку або стабільного інтернет-з’єднання. Вимоги до роботи офлайн охоплюють не лише доступ до карт, а й можливість перегляду маршрутів, додавання точок інтересу, навігації за збереженим треком, збереження статистики руху та подальшу синхронізацію з хмарою після відновлення зв’язку.

Перший ключовий компонент, що забезпечує офлайн-режим, — це модуль кешування картографічних тайлів. Користувач має змогу заздалегідь завантажити карти для потрібної області, вибравши діапазон масштабу та географічні координати. У застосунку реалізовано інтерактивний інтерфейс вибору області карти за допомогою прямокутної рамки на мапі, після чого дані завантажуються у форматі MBTiles або через окремі PNG-тайли, що зберігаються в локальній базі даних SQLite. Для покращення продуктивності передбачено індексацію тайлів за координатами та рівнем масштабування.

Обробка кешу реалізована за допомогою інтерфейсу IMapCache, який підтримує операції збереження, отримання та очищення кешованих карт. Тайли зберігаються в окремій таблиці з метаданими про дату останнього використання, що дозволяє реалізувати механізм автоматичного очищення кешу при нестачі пам’яті.

Наступним важливим компонентом є підтримка офлайн-навігації. Після завантаження маршруту або його побудови в онлайні користувач має змогу зберегти маршрут локально, включно з усіма контрольними точками, параметрами оптимізації, POI та треком. У польових умовах застосунок використовує GPS-модуль пристрою для визначення координат і проводить локальний аналіз відхилення від маршруту. У разі втрати зв’язку з інтернетом повторна маршрутизація здійснюється на основі збереженої карти та локального алгоритму A*, який працює з графом переходів між тайлами.

Формат збереження маршруту описаний у вигляді серіалізованого об’єкта RouteData з підтримкою JSON-формату. Це дозволяє зберігати дані у вигляді текстового файлу, що легко читається та може бути переданий на інший пристрій.

```
public class RouteData
{
    public Location Start { get; set; }
    public Location End { get; set; }
    public List<Location> Waypoints { get; set; }
    public List<PointOfInterest> POIs { get; set; }
    public DateTime SavedAt { get; set; }
}
```

У разі відсутності інтернету користувач також може додавати нові POI, фотографувати об’єкти, фіксувати координати з коментарем. Ці дані зберігаються у локальній базі даних із позначкою «в очікуванні синхронізації». Після відновлення зв’язку активується фоновий сервіс ISyncService, який перевіряє, чи є зміни у локальному сховищі, і поетапно надсилає дані до серверної частини застосунку, забезпечуючи цілісність та запобігаючи дублюванню.

Ще однією особливістю є автономний запис треку пересування, який працює без інтернету та використовує лише GPS-модуль. Логування треку відбувається кожні 10 секунд або при зміні позиції понад 5 метрів. Усі точки зберігаються у вигляді списку координат із міткою часу, які пізніше використовуються для побудови графіків статистики та візуалізації шляху на карті.

Застосунок також підтримує офлайн-доступ до останніх повідомлень та сповіщень. Наприклад, якщо перед походом користувач отримав попередження про погіршення погодних умов, воно зберігається у локальному кеші та залишається доступним навіть без підключення. Аналогічно, усі дані про маршрути, переглянуті напередодні, доступні з кешу і не потребують повторного завантаження.

Реалізація офлайн-функціоналу також передбачає можливість повного локального резервного копіювання даних. Користувач має змогу експортувати свої

маршрути, POI, треки та статистику у вигляді zip-архіву, який пізніше може бути імпортований назад у застосунок. Це особливо корисно при зміні пристрою або повторному встановленні застосунку.

Окремо варто зазначити підтримку офлайн-режиму в інтерфейсі користувача. Усі кнопки, що вимагають доступу до інтернету, автоматично стають неактивними або відображають альтернативні дії. Наприклад, замість кнопки «Побудувати маршрут онлайн» відображається «Завантажити збережений маршрут». Стан підключення до мережі відображається у вигляді індикатора в рядку стану програми. Користувач завжди бачить, у якому режимі перебуває — онлайн чи офлайн.

Для підвищення ефективності офлайн-режиму застосунок використовує систему передзавантаження даних. Перед початком походу користувач може увійти в режим «Підготовки маршруту», де йому пропонується автоматично завантажити усі необхідні карти, POI, прогнози погоди, профілі рельєфу та інші дані, які можуть знадобитися в дорозі. Після завантаження все зберігається локально, і система видає повідомлення про готовність до офлайн-режиму.

Таким чином, офлайн-функціонал у StudentTravel не є лише додатковою опцією, а повноцінною архітектурною концепцією, що забезпечує повноцінну автономну роботу застосунку. Вона охоплює карти, навігацію, маршрути, POI, треки, статистику та резервне копіювання, дозволяючи користувачу залишатися незалежним від зв’язку та зосередитися на безпечному й ефективному пересуванні. Реалізація усіх цих можливостей у поєднанні з продуманим інтерфейсом та механізмами синхронізації забезпечує високу надійність та зручність навіть у найскладніших умовах експлуатації.

Висновок до розділу 2

В даному розділі детально окреслено ключові нефункціональні, інтерфейсні та офлайн-вимоги мобільного застосунку StudentTravel, які є фундаментальними для його надійної та ефективної роботи в умовах польових подорожей. Особлива увага приділена продуктивності через застосування кешування, асинхронних

операцій та політик повторних запитів, що забезпечує швидку та стабільну взаємодію з даними навіть при обмежених ресурсах. Безпека реалізована багаторівнево, включаючи захищене зберігання токенів, шифрування, TLS-з’єднання та механізми автентифікації, що гарантує цілісність та конфіденційність інформації.

Також виділено значну увагу UX-дизайну, орієнтованому на інтуїтивність, адаптивність та зручність використання в екстремальних умовах, з підтримкою офлайн-функціоналу як основного елемента архітектури. Підтримка автономної роботи з локальним кешем карт, збереженням маршрутів, синхронізацією даних і офлайн-навігацією забезпечує користувачу максимальну незалежність від мережевого з’єднання, що є критично важливим для безпечного та комфортного планування і проходження піших подорожей. Таким чином, всі вимоги сформовані комплексно, що дозволяє створити надійний, зручний і масштабований продукт, готовий до реальних викликів польових умов. даному розділі

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1. Загальна архітектурна модель

Загальна архітектурна модель мобільного застосунку StudentTravel базується на принципах багаторівневої модульної побудови, що забезпечує чітке розмежування відповідальностей між компонентами системи, гнучкість масштабування, можливість повторного використання коду та підтримку як онлайн-, так і офлайн-режимів. Архітектура застосунку умовно поділяється на три рівні: рівень представлення (Presentation Layer), рівень бізнес-логіки (Application Layer) та рівень доступу до даних (Data Access Layer). Окрім того, застосовується зовнішній рівень API/інтеграцій для взаємодії з картографічними сервісами, маршрутними серверами та хмарним сховищем.

На рівні представлення реалізовано інтерфейс користувача на базі .NET MAUI, що забезпечує кросплатформенність та єдину кодову базу для Android, iOS та Windows. Всі візуальні компоненти побудовані за принципом MVVM (Model-View-ViewModel), що дозволяє відокремити логіку від подання. ViewModel-елементи взаємодіють з сервісами через інтерфейси, що спрощує тестування та розширення функціоналу. Наприклад, екран побудови маршруту має окрему модель представлення RouteBuilderViewModel, яка відповідає за обробку введення користувача, валідацію точок маршруту та ініціацію запитів до сервісу маршрутизації.

Усі сервіси застосунку, такі як робота з маршрутами, POI, кешем, статистикою, реалізовано у вигляді інтерфейсів та їхніх імплементацій. Це дозволяє використовувати залежності через DI-контейнер (Dependency Injection), забезпечуючи слабке зв’язування між компонентами. Структура бізнес-логіки орієнтована на сервісно-орієнтований підхід: кожна функція застосунку делегується окремому сервісу з чітко визначеним контрактом. Наприклад, сервіс

IRouteService відповідає за побудову маршруту, його збереження та перерахунок у разі відхилення.

На рівні доступу до даних використовується локальна база даних SQLite, яка реалізована через Entity Framework Core з обмеженим контекстом та міграціями. Кожна сутність (наприклад, маршрут, точка інтересу, кешований тайл) має відповідну модель даних і таблицю. Усі звернення до бази даних виконуються асинхронно, що забезпечує безперервність роботи інтерфейсу. Також реалізовано репозиторії для абстрагування роботи з даними від бізнес-логіки, наприклад IRouteRepository, IPoiRepository, ITileRepository.

Для роботи з мережею передбачено окремий модуль API-інтеграцій. Всі зовнішні запити до сервісу маршрутизації, завантаження картографічних даних та синхронізації POI реалізовано через HTTP-клієнти з конфігурацією політик повтору (retry), тайм-аутів і обробки помилок. У застосунку також передбачено механізм черги запитів у разі відсутності інтернету: кожен невдалий запит потрапляє до черги і виконується автоматично після відновлення з’єднання. Це особливо актуально для польових умов.

Зовнішній API застосунку реалізовано як набір REST-ендпоінтів із підтримкою аутентифікації через JWT-токени. Дані передаються у форматі JSON, а усі запити підписуються HMAC-підписом для перевірки цілісності. На стороні сервера функціонують мікросервіси для обробки маршрутів, збереження даних користувача, обробки статистики, завантаження фото та резервного копіювання.
Архітектура також включає механізм подій (event bus), який дозволяє передавати сигнали між компонентами без жорсткої залежності. Наприклад, коли користувач завершує маршрут, спрацьовує подія RouteCompletedEvent, яка автоматично ініціює збереження статистики, синхронізацію з сервером та відображення сповіщення.

На логічному рівні архітектура системи може бути представлена як така:
[Presentation Layer]
└── View → ViewModel → CommandHandler

[Application Layer]
└── Services → Interfaces:
├─ IRouteService
├─ IPoiService
├─ IMapCacheService
└─ IAnalyticsService

[Data Access Layer]
└── Repositories → DbContext (SQLite)

[External API Layer]
└── HttpClients → REST API (Routing, Maps, Auth)

[Infrastructure]
└── EventBus, SecureStorage, BackgroundTasks

Завдяки такій побудові архітектура застосунку StudentTravel є гнучкою, масштабованою та легко підтримуваною. Кожен компонент можна оновлювати, тестувати або замінювати без впливу на інші частини системи. Це дозволяє не лише забезпечити високу якість початкової версії продукту, а й створити надійний фундамент для подальшого розширення функціоналу: підтримки нових форматів карт, інтеграції з пристроями GPS, введення офлайн-синхронізації через Bluetooth або mesh-мережі. Архітектура орієнтована на реальні умови використання туристами, тому її основними принципами є надійність, автономність та ефективність.

3.2. UML-діаграми

У процесі проєктування застосунку StudentTravel були використані різні типи UML-діаграм для формалізації структури системи, визначення взаємозв’язків між компонентами та опису поведінки користувача. Це дозволяє не лише візуалізувати архітектуру проєкту, а й полегшує командну розробку, тестування, супровід та майбутнє масштабування програмного забезпечення. У цьому розділі представлено три основні типи діаграм: діаграма прецедентів (Use Case), діаграма класів (Class Diagram) та діаграма послідовності (Sequence Diagram), які відображають логіку взаємодії між користувачем, інтерфейсами та внутрішніми компонентами застосунку.

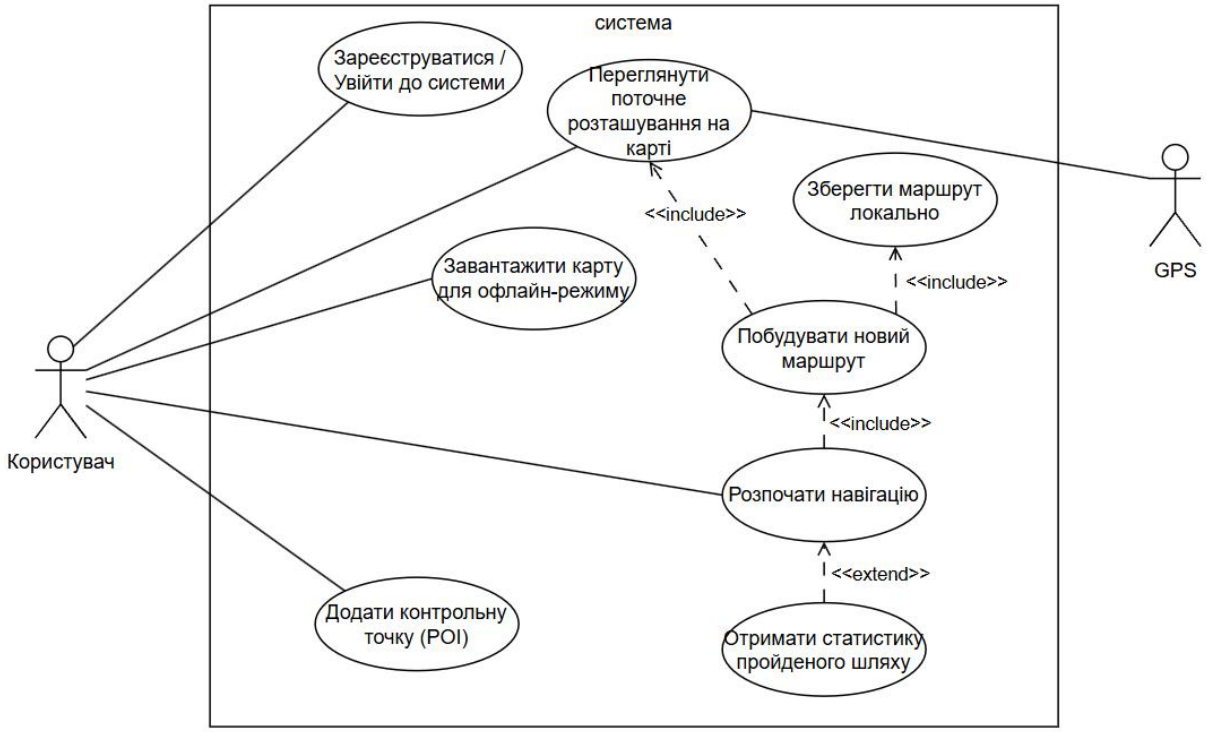


Рис. 3.1 Діаграма прецедентів (Use Case Diagram)

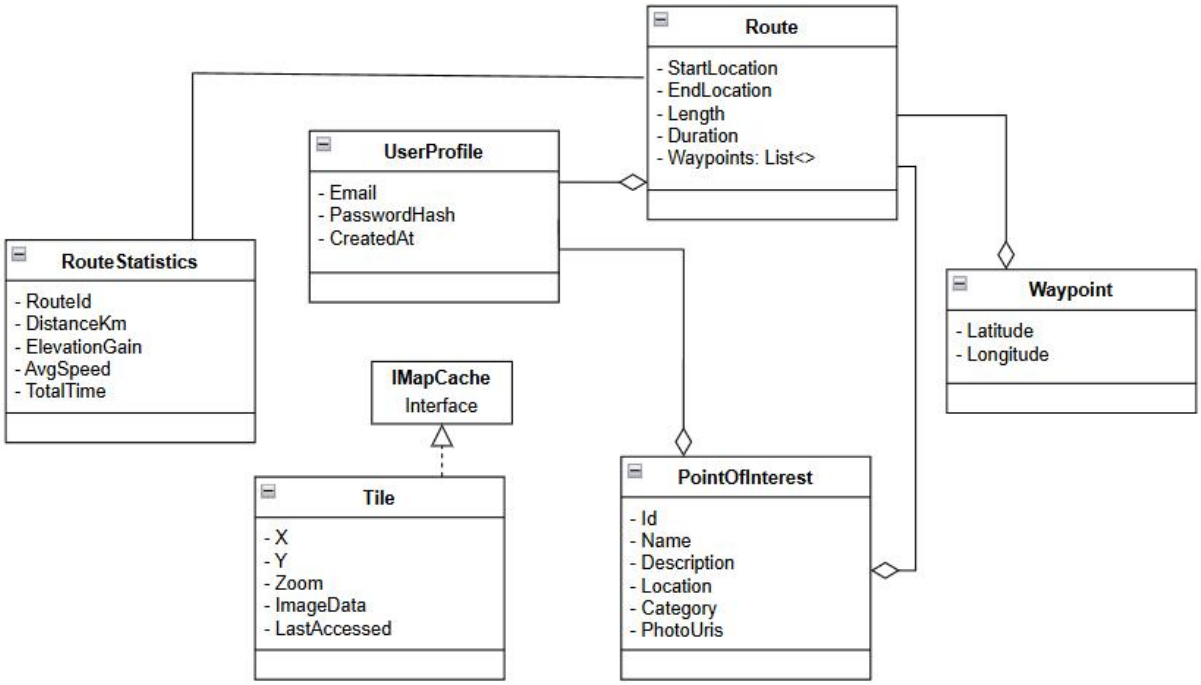


Рис. 3.2 Діаграма класів (Class Diagram)

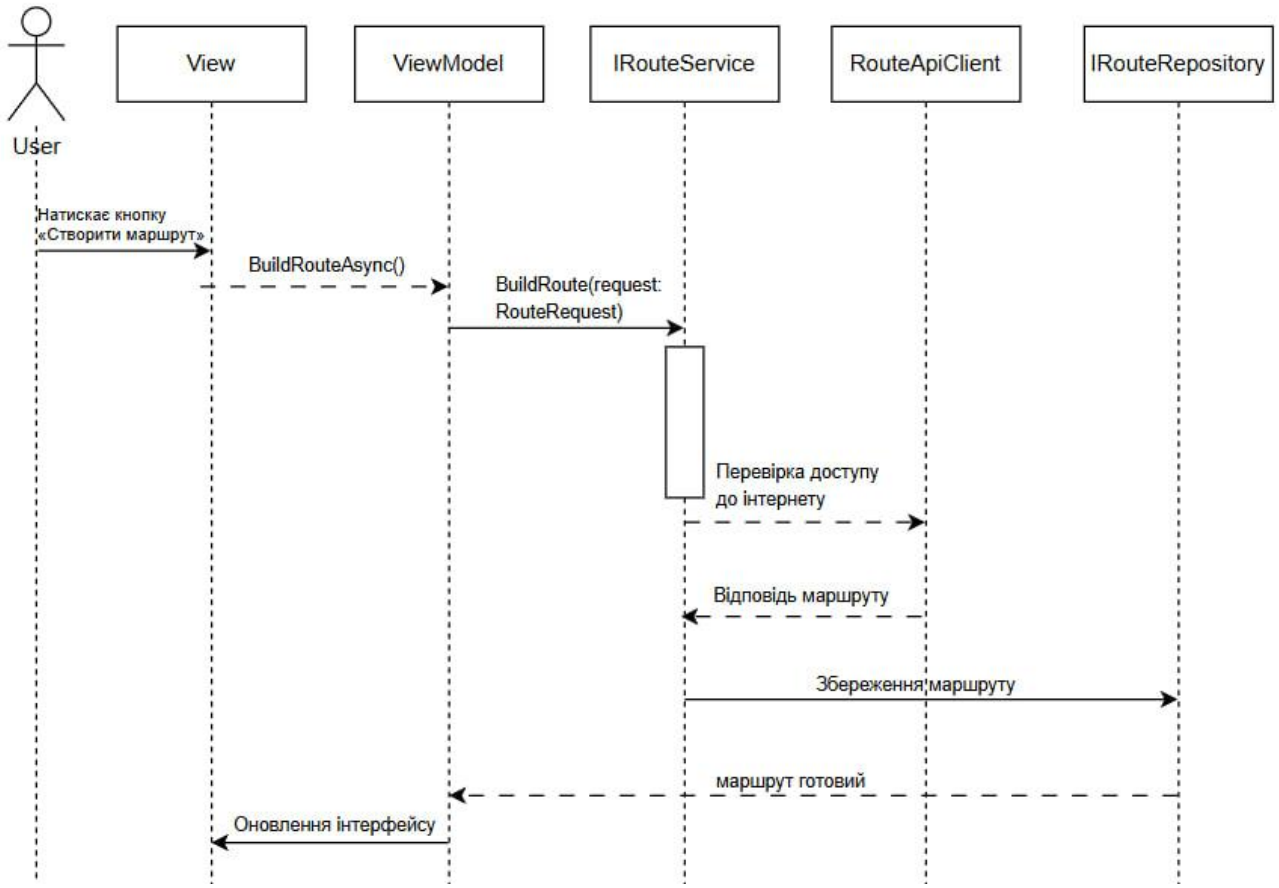


Рис. 3.3 Діаграма послідовності (Sequence Diagram)

3.3. Проєктування бази даних

Проєктування бази даних у мобільному застосунку StudentTravel є ключовим етапом для забезпечення надійного зберігання маршрутів, точок інтересу, кешованих картографічних даних, статистики проходження та профілів користувачів. Оскільки застосунок має працювати як у онлайн-, так і в офлайн-режимі, було прийнято рішення використовувати локальну реляційну базу даних SQLite, яка є легковажною, швидкою, не потребує підключення до сервера та повністю підтримується .NET MAUI через Entity Framework Core або SQLite.Net.
Структура бази даних спроєктована таким чином, щоб забезпечити нормалізацію даних, зменшити дублювання інформації, забезпечити швидкий доступ до необхідних записів та підтримувати асинхронну роботу із застосунком. Центральними сутностями у структурі є таблиці Routes, Waypoints, PointsOfInterest, Tiles, Users та RouteStatistics.

Таблиця Users зберігає основну інформацію про користувача: унікальний ідентифікатор, електронну пошту, хешований пароль, дату створення профілю, токени для синхронізації. Це дозволяє організувати авторизацію, зберігати персональні налаштування та прив’язувати маршрути до конкретного користувача.

Таблиця Routes містить інформацію про побудовані маршрути: стартову та кінцеву координати, назву, опис, дату створення, тривалість, загальну довжину, а також зовнішній ключ UserId, що забезпечує зв’язок із користувачем. Для маршруту передбачено окрему таблицю Waypoints, яка зберігає проміжні контрольні точки з координатами, порядком розташування та описом.
Особливе місце займає таблиця PointsOfInterest, яка зберігає дані про додані вручну POI користувачем або імпортовані з серверу. Вона включає такі поля, як унікальний Id, Name, Description, координати, тип категорії (джерело води, місце ночівлі, краєвид тощо), фото (у вигляді URI) та RouteId для прив’язки до певного маршруту.

Для підтримки офлайн-карт реалізовано таблицю Tiles, яка містить координати тайла (X, Y), рівень масштабування (Zoom), бінарні дані зображення,

дату останнього доступу. Це дозволяє швидко завантажувати необхідні частини карти, реалізувати політику LRU (Least Recently Used) для видалення застарілих даних та уникнути перевантаження сховища.

Таблиця RouteStatistics зберігає результат проходження кожного маршруту: тривалість, середню швидкість, загальний набір висоти, кількість зупинок. Ці дані використовуються для побудови графіків, аналітики та перегляду користувачем своєї історії активності.

Нижче подано приклад частини SQL-схеми створення таблиці для маршрутів:

```
CREATE TABLE Routes (
    Id TEXT PRIMARY KEY,
    UserId TEXT,
    Name TEXT,
    Description TEXT,
    StartLatitude REAL,
    StartLongitude REAL,
    EndLatitude REAL,
    EndLongitude REAL,
    Distance REAL,
    Duration REAL,
    CreatedAt TEXT,
    FOREIGN KEY (UserId) REFERENCES Users(Id)
);
```

А також таблиця для POI:

```
CREATE TABLE PointsOfInterest (
    Id TEXT PRIMARY KEY,
    Name TEXT,
    Description TEXT,
    Latitude REAL,
    Longitude REAL,
```

```
    Category TEXT,
    PhotoUris TEXT,
    CreatedAt TEXT,
    RouteId TEXT,
    FOREIGN KEY (RouteId) REFERENCES Routes(Id)
);
```

Під час проєктування також було передбачено індексацію критичних полів (наприклад, координат POI, ідентифікаторів маршрутів, дати створення), що дозволяє прискорити пошук у великому обсязі даних. Для прикладу, індекс на CreatedAt дозволяє швидко відображати найновіші маршрути або POI на головному екрані.

Усі операції з базою даних реалізовано асинхронно через відповідні сервіси доступу до даних (Repositories), які ізольовані від бізнес-логіки. Це забезпечує розширюваність проєкту, можливість легкої заміни SQLite на іншу СУБД у майбутньому, та підтримку автоматичних міграцій при зміні схеми.

Таким чином, структура бази даних у StudentTravel є оптимізованою під специфіку застосунку: підтримка автономної роботи, прив’язка до користувача, зберігання картографічних та маршрутних даних, облік активностей і статистики. Такий підхід дозволяє створити надійне середовище для збереження інформації в польових умовах, мінімізувати втрати при нестабільному з’єднанні та підвищити загальну стабільність застосунку.

Ось спрощена **схема взаємозв’язків таблиць бази даних (ERD)** для мобільного застосунку **StudentTravel**, яка відображає логіку зберігання маршрутів, користувачів, точок інтересу, статистики та кешу карт:

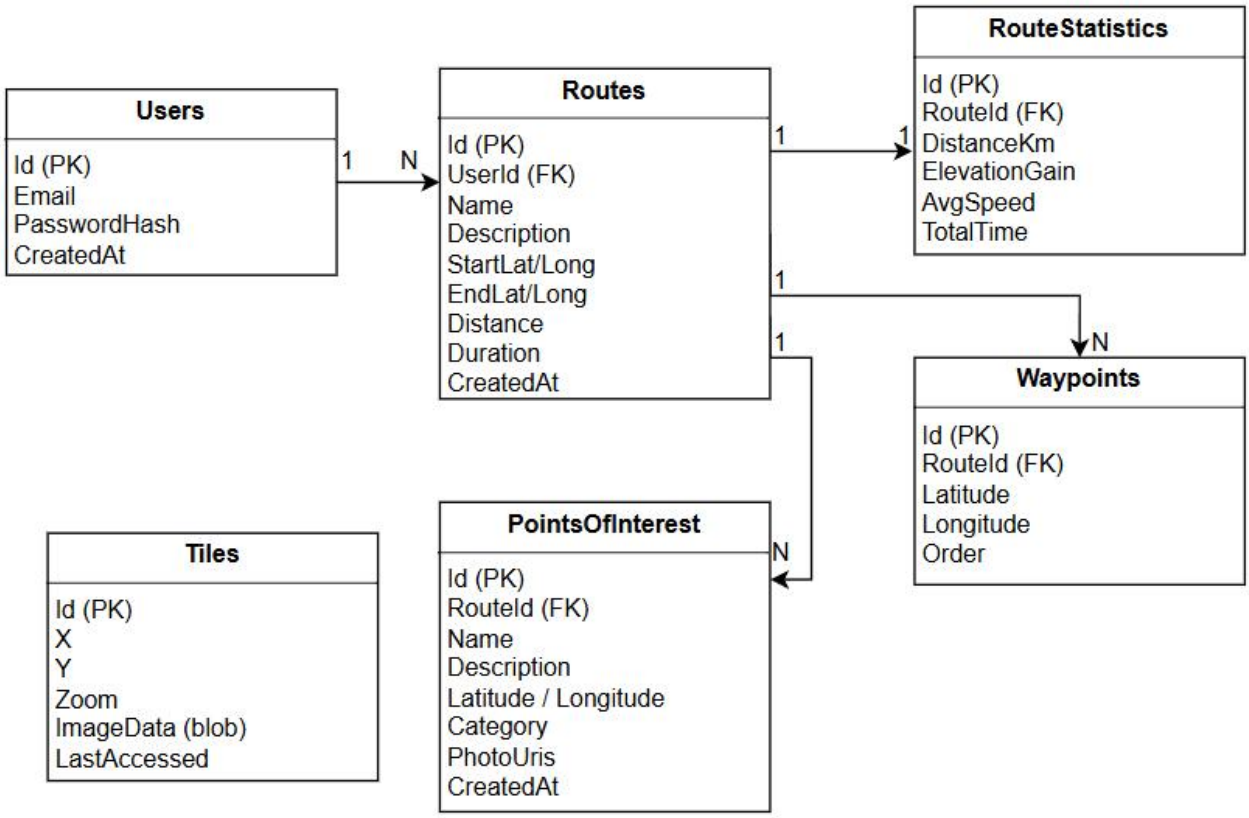


Рис. 3.4 Схема взаємозв’язків таблиць бази даних

Пояснення взаємодії:

- Один **User** може мати багато **Routes**.
- Кожен **Route** може мати:
  - багато **Waypoints** (проміжні точки маршруту),
  - одну **RouteStatistics** (зведені дані після проходження),
  - багато **PointsOfInterest** (точки інтересу, позначені користувачем).
- **Tiles** (карти) зберігаються незалежно, але активно використовуються для офлайн-навігації у RouteView.

3.4. Інтеграція з картографічними сервісами

Інтеграція з картографічними сервісами є критично важливою складовою функціональності мобільного застосунку StudentTravel, оскільки саме карта є центральним елементом взаємодії користувача з маршрутом, точками інтересу та системою навігації. Для забезпечення гнучкості, швидкості та автономності було реалізовано комбінацію використання як онлайн-, так і офлайн-картографічних рішень з можливістю динамічного перемикання між ними.

У якості основи для інтеграції обрано відкриті картографічні платформи, сумісні з форматом OpenStreetMap (OSM), що дозволяє використовувати безкоштовні та оновлювані картографічні дані з глобальним покриттям. У режимі онлайн завантаження тайлів відбувається через стандартизовані URL-запити з підтримкою координат (x, y, zoom). Для рендерингу карт використано бібліотеку Maui.Maps з підтримкою шарів, маркерів та векторної графіки. Тайли завантажуються за шаблоном:

https://tile.openstreetmap.org/{z}/{x}/{y}.png

де {z} – рівень масштабування, {x} і {y} – координати тайла. Запити формуються динамічно залежно від області, яку переглядає користувач, і кешуються у локальній базі SQLite з метаданими про дату останнього звернення. Це дозволяє реалізувати частковий офлайн-режим навіть при неактивному підключенні.

Окрім онлайн-карт, реалізовано повноцінну підтримку офлайн-режиму за допомогою попереднього завантаження фрагментів карти у форматі MBTiles або наборів PNG-файлів. Користувач обирає область, яку хоче зберегти для подорожі, задає рівні масштабування (наприклад, з 12 по 16), після чого система автоматично обчислює кількість необхідних тайлів, обсяг сховища та пропонує завантаження. Тайли зберігаються у структурі типу:

/maps/{zoom}/{x}/{y}.png

Для рендерингу офлайн-карт застосовується окрема служба OfflineTileProvider, яка реалізує інтерфейс ITileSource. Цей сервіс автоматично

підключається при відсутності інтернету або за запитом користувача, забезпечуючи безшовний перехід між режимами.

Також у межах інтеграції з картографічними сервісами реалізовано функцію накладання додаткових шарів – рельєфу, маршрутів, POI, теплових карт і меж національних парків. Дані для цих шарів можуть бути отримані як з сервера StudentTravel, так і з відкритих джерел, таких як Mapbox або OpenTopoMap. Для побудови маршрутів використовується або локальний алгоритм (у разі офлайн-режиму), або зовнішній REST API на базі OSRM (Open Source Routing Machine), що дозволяє прокладати маршрути пішки, враховуючи реальні стежки та доріжки.

У клієнтській частині реалізовано компоненти для роботи з маркерами – контрольними точками маршруту та POI. Кожна така точка виводиться на карту з урахуванням обраної категорії (вода, відпочинок, оглядова точка), підтримує кастомну іконку, підпис, а при натисканні відкриває додаткову інформацію. Операції з маркерами синхронізовані з базою даних, тож будь-які зміни зберігаються для подальшого використання.

Нижче наведено спрощений фрагмент коду, що демонструє завантаження тайла з локального кешу або мережі:

```
public async Task<ImageSource> GetTileAsync(int x, int y, int zoom)
{
    var tile = await tileCache.LoadTileAsync(x, y, zoom);
    if (tile != null)
        return ImageSource.FromStream(() => new MemoryStream(tile.ImageData));

    var url = $"https://tile.openstreetmap.org/{zoom}/{x}/{y}.png";
    var data = await httpClient.GetByteArrayAsync(url);
    await tileCache.StoreTileAsync(new Tile(x, y, zoom, data));
    return ImageSource.FromStream(() => new MemoryStream(data));
}
```

Ще одним важливим аспектом є підтримка GPS-навігації на карті. Поточне положення користувача відображається як динамічна точка, що оновлюється

щосекунди або при зміні координат більш ніж на 5 метрів. При цьому карта може автоматично центруватися на користувачеві, а в разі наближення до POI або відхилення від маршруту – виводити відповідні повідомлення. Навігація здійснюється по підготовленому маршруту з вказівками типу «прямо», «поверніть ліворуч» тощо, які генеруються як на основі API, так і локальними алгоритмами.

Таким чином, реалізація інтеграції з картографічними сервісами у StudentTravel забезпечує не лише базову функцію відображення мап, а й повноцінну підтримку побудови маршрутів, офлайн-доступу, динамічного кешування, накладання шарів, роботи з POI та GPS-навігації. Такий підхід дозволяє зробити використання карт максимально зручним, швидким і адаптованим до умов реального походу, включаючи ті, де зв’язок відсутній.

Висновок до розділу 3

Було детально розглянуто ключові аспекти архітектурного проєктування мобільного застосунку StudentTravel, зокрема багаторівневу модульну структуру, яка забезпечує розділення відповідальностей та гнучкість розвитку системи. Особлива увага приділена проєктуванню бази даних із використанням локальної SQLite, що гарантує надійне зберігання та ефективний доступ до даних у режимах онлайн і офлайн, а також інтеграції з картографічними сервісами, які забезпечують динамічне завантаження, кешування та підтримку навігації в умовах нестабільного інтернет-з’єднання.

Застосування сучасних технологій, таких як MVVM, Dependency Injection, а також підтримка офлайн-режиму і адаптивних картографічних рішень, дозволяє забезпечити комфортний користувацький досвід навіть у складних польових умовах. Такий комплексний підхід підвищує надійність, масштабованість і функціональність застосунку, роблячи його ефективним інструментом для подорожей та навігації.

4 РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1. Огляд технологій (.NET MAUI, C#)

У розробці мобільного застосунку StudentTravel було використано сучасні технології, які забезпечують кросплатформенність, високу продуктивність, масштабованість та зручність у розробці. Основними обраними інструментами стали мова програмування C# та фреймворк .NET MAUI (Multi-platform App UI) — офіційна еволюція Xamarin.Forms, яка дозволяє створювати додатки під Android, iOS, Windows і macOS з єдиної кодової бази.

.NET MAUI є офіційним кросплатформеним фреймворком від Microsoft, що базується на .NET 6/7 та надає інструменти для побудови UI, доступу до апаратного забезпечення, роботи з файловою системою, базами даних, геолокацією, HTTP-запитами та інтеграції з нативними можливостями пристроїв. Його головна перевага — можливість написати один спільний код для кількох платформ, з мінімальними змінами у проєкті під час компіляції під різні ОС.

Мова програмування C#, яка використовується для логіки застосунку, є строго типізованою, об’єктно-орієнтованою та має потужні засоби обробки помилок, асинхронного програмування (через async/await), впровадження залежностей, модульного тестування та серіалізації даних. Це дозволяє забезпечити чисту, структуровану та масштабовану архітектуру.

UI застосунку StudentTravel реалізовано з використанням XAML-розмітки, що підтримується .NET MAUI та дозволяє чітко розділяти вигляд (View) і поведінку (ViewModel). Такий підхід базується на патерні MVVM (Model-View-ViewModel), який сприяє високій читабельності коду, легкому рефакторингу та автоматизації тестування. Наприклад, інтерфейс побудови маршруту має View у XAML та логіку в RouteBuilderViewModel.cs, яка обробляє введення користувача, валідацію координат і взаємодію з сервісами маршрутизації.

Для зберігання даних локально використано SQLite, яка інтегрується з .NET MAUI через бібліотеки Microsoft.Data.Sqlite, EntityFrameworkCore.Sqlite або сторонні рішення типу SQLite-net. Це дозволяє створювати реляційні таблиці, підтримувати індекси, працювати з транзакціями та зберігати як маршрути, так і кешовані картографічні тайли, точки інтересу, статистику.

HTTP-взаємодія з сервером реалізується через вбудований клас HttpClient, який підтримує конфігурацію заголовків, тайм-аутів, повторів (retry), автентифікацію через JWT-токени та передачу даних у форматі JSON. Для обробки помилок використовується бібліотека Polly, яка дозволяє задати політики обробки сіткових збоїв.

GPS-модуль, доступ до камери, файлової системи, push-сповіщень, інформації про батарею та інші апаратні компоненти реалізовано через бібліотеку Microsoft.Maui.Essentials. Цей модуль надає спрощені API для найпоширеніших задач. Наприклад, щоб отримати поточне розташування користувача, достатньо виконати:

```
var location = await Geolocation.GetLastKnownLocationAsync();
if (location != null)
{
    Console.WriteLine($"Latitude: {location.Latitude}, Longitude: {location.Longitude}");
}
```

Для рендерингу карт використано бібліотеки, сумісні з OpenStreetMap, зокрема Mapsui, Maui.Maps або нативні компоненти Google Maps (через DependencyService). Це дозволяє накладати шари, додавати маркери, працювати з офлайн-тайлами, обробляти жести масштабування, перегортання та натискання.

Для графіків і статистики — MAUI Charts або сторонні компоненти типу Microcharts, які дозволяють будувати лінійні графіки швидкості, часу, висоти з колекцій даних, отриманих після завершення маршруту. Для зберігання зображень та синхронізації POI з хмарою застосунок використовує зовнішнє API (наприклад, Firebase Storage або власне REST-сховище).

Інтеграція із службами повідомлень, аналітики, авторизації та віддаленої діагностики реалізована через зовнішні SDK — наприклад, Microsoft App Center або Firebase, що дозволяє не лише тестувати застосунок на реальних пристроях, а й аналізувати поведінку користувача, повідомляти про краші та покращувати досвід використання.

Таким чином, поєднання .NET MAUI і C# забезпечило застосунку StudentTravel високий рівень продуктивності, універсальності, підтримки на різних платформах та легкість у подальшій підтримці й масштабуванні. Вибрані технології дозволили створити стабільну, надійну, автономну систему для туристів, яка залишається зручною як у місті, так і у віддалених регіонах без зв’язку.

4.2. Розробка модуля побудови та оптимізації маршруту

Модуль побудови та оптимізації маршруту є ключовим функціональним компонентом мобільного застосунку StudentTravel, оскільки саме він дозволяє користувачеві створювати індивідуальні маршрути для піших подорожей з урахуванням складності рельєфу, відстані, часу проходження та наявності контрольних точок. Його архітектура побудована на принципах розділення обов’язків, підтримки офлайн-режиму, розширюваності та адаптації до реальних умов експлуатації.

Модуль складається з кількох підсистем: інтерфейсу взаємодії з користувачем (UI), служби обробки введених координат (Input Processor), маршрутизатора (Routing Engine), оптимізатора (Optimization Logic) та системи збереження результату (Persistence Service). Усі частини взаємодіють через чітко визначені інтерфейси, що дозволяє легко підмінювати реалізації — наприклад, використовувати онлайн-маршрутизатор у разі наявності інтернету або локальний алгоритм A* при офлайн-режимі.

Візуально користувач взаємодіє з картою, на якій за допомогою натискання або вибору з контекстного меню встановлює початкову та кінцеву точки маршруту. Додатково можна додати необмежену кількість проміжних точок (waypoints), які

визначають пріоритетні місця проходження — наприклад, джерело води, оглядову точку або місце ночівлі. Усі ці координати формуються в об’єкт типу RouteRequest, який передається на обробку в модуль маршрутизації.

```
public class RouteRequest
{
    public Location Start { get; set; }
    public Location End { get; set; }
    public List<Location> Waypoints { get; set; }
    public RouteOptimizationCriteria Criteria { get; set; }
}
```

Оптимізація маршруту підтримує декілька критеріїв, серед яких: найкоротший шлях, мінімальний набір висоти, уникнення складного рельєфу, проходження через визначені POI, живописність (scenic route). Ці параметри задаються у полі Criteria і впливають на алгоритм обчислення. Для онлайн-режиму використовується зовнішній сервіс маршрутизації, сумісний з Open Source Routing Machine (OSRM), який отримує координати і критерії як HTTP-запит і повертає серіалізований маршрут у форматі GeoJSON або власному DTO. У випадку офлайн-режиму система переходить на локальну реалізацію алгоритму A* або Dijkstra, яка оперує попередньо збереженим графом тайлів.

Алгоритм побудови маршруту реалізовано як стратегія IRoutingStrategy, що дозволяє мати кілька реалізацій:

Реалізація для офлайн-маршрутування (OfflineRoutingStrategy) використовує збережену мапу переходів між тайлами, де кожне ребро має вагу залежно від висоти, типу поверхні та щільності перешкод. Для побудови графа використано бібліотеку QuickGraph, а обчислення шляху відбувається за допомогою кастомізованої версії A*, де евристична функція включає похідну висоти та ухил. У разі виявлення недоступних ділянок система автоматично намагається побудувати обхідний маршрут або пропонує користувачеві змінити точки.

Результатом маршрутизації є об’єкт RouteResult, який містить список сегментів маршруту з координатами, загальну довжину, оцінку часу проходження, сумарний набір висоти, ризиковані ділянки та рекомендовані POI для зупинок.

```
public class RouteResult
{
    public List<Location> Path { get; set; }
    public double TotalDistanceKm { get; set; }
    public double EstimatedDurationMin { get; set; }
    public double ElevationGain { get; set; }
    public List<string> Warnings { get; set; }
}
```

Після побудови маршрут зберігається у локальній базі, а його графічне відображення накладається на карту у вигляді полілайну з маркерами на кожній контрольній точці. Користувач може переглянути профіль маршруту, включаючи графік висоти, переглянути рекомендовані місця відпочинку та змінити маршрут вручну, перемістивши точки на карті. Усі зміни автоматично синхронізуються з маршрутом у БД.

Для підтримки повторного використання маршруту реалізовано експорт та імпорт у форматах GPX, KML та власному JSON-форматі. Це дозволяє ділитися маршрутами з іншими користувачами або використовувати їх повторно при повторному поході. Також передбачено можливість копіювання маршруту, його редагування та маркування як «обраного».

Модуль побудови маршруту також враховує обмеження пристрою — зокрема, енергоощадний режим, що впливає на частоту оновлення позиції під час навігації. У разі критичного зниження заряду акумулятора користувач отримує попередження з пропозицією перемкнутися на спрощений режим трекінгу, де позиція оновлюється рідше, а UI спрощується.

Таким чином, розроблений модуль побудови та оптимізації маршруту є автономним, адаптивним, підтримує декілька джерел побудови (локально/онлайн), кілька стратегій маршрутизації, розширену логіку оптимізації, візуалізацію,

аналітику та збереження даних. Його гнучкість дозволяє застосовувати його як у простих міських прогулянках, так і у складних багатоденних гірських походах, забезпечуючи надійність і зручність навіть у найменш сприятливих умовах.

4.3. Інтеграція офлайн-карт (відображення, кешування)

У мобільному застосунку StudentTravel інтеграція офлайн-карт є критично важливою функцією, що забезпечує повноцінну автономну роботу в умовах відсутності інтернет-з’єднання. Оскільки користувачі можуть перебувати в гірській або лісовій місцевості, де мережеве покриття відсутнє або нестабільне, застосунок повинен мати можливість завчасного завантаження картографічних даних, їх локального зберігання, обробки та швидкого відображення. У цьому розділі описано, як реалізовано механізми відображення карт, кешування тайлів, навігації по офлайн-карті та управління кешем.

Загальна архітектура модуля побудована за принципом розділення на джерело тайлів (Tile Source), механізм збереження (Tile Cache) та рендеринг карти (Map Renderer). В онлайн-режимі джерелом даних є відкритий сервіс OpenStreetMap, а в офлайн — локальний кеш тайлів, збережених у файловій системі або SQLite. Усі запити до карт спочатку перевіряють наявність відповідного тайла в локальній базі, і лише у разі його відсутності звертаються до мережі.

Формат тайлів відповідає стандарту XYZ, де координати x, y та рівень масштабування zoom визначають конкретне зображення. Запити формуються за шаблоном:

https://tile.openstreetmap.org/{z}/{x}/{y}.png

У випадку офлайн-режиму використовується спеціальний компонент OfflineTileProvider, який реалізує інтерфейс ITileSource і замінює джерело даних на локальний каталог або базу. Тайли можуть бути завантажені масово через інтерфейс підготовки регіону — користувач вибирає прямокутну область на карті,

задає рівні масштабування (наприклад, 10–16), після чого система автоматично обчислює кількість тайлів, розмір кешу та ініціює фонове завантаження.

Приклад збереження тайла у локальній базі:

```
public async Task StoreTileAsync(Tile tile)
{
    var existing = await dbContext.Tiles
        .FirstOrDefaultAsync(t => t.X == tile.X && t.Y == tile.Y && t.Zoom == tile.Zoom);

    if (existing == null)
        await dbContext.Tiles.AddAsync(tile);
    else
        existing.LastAccessed = DateTime.UtcNow;

    await dbContext.SaveChangesAsync();
}
```

Тайли зберігаються у вигляді BLOB-поля з індексацією по координатах та часі останнього доступу. Це дозволяє реалізувати політику LRU (Least Recently Used) для очищення простору. Наприклад, при досягненні порогу кешу в 200 МБ застосунок автоматично видаляє найстаріші записи.

Для візуалізації офлайн-карти використано бібліотеки Mapsui або Maui.Maps, які підтримують накладання тайлів, полігонів, маркерів, полілайнів та взаємодію з користувачем (жести, тапи, масштабування). При зміні масштабу або переміщенні карти система обчислює потрібні тайли та завантажує їх із кешу. Приклад отримання тайла:

```
public async Task<ImageSource> GetTileAsync(int x, int y, int zoom)
{
    var tile = await tileCache.LoadTileAsync(x, y, zoom);
    if (tile != null)
        return ImageSource.FromStream(() => new MemoryStream(tile.ImageData));
```

```
    // Якщо не знайдено, завантаження з інтернету (за потреби)
    return null;
}
```

Користувач має доступ до меню «Офлайн-карти», де відображено список завантажених регіонів, їхній розмір, дата останнього використання, кількість тайлів. Звідси можна видалити непотрібні області, оновити окремі ділянки або переглянути попередній перегляд регіону на карті.

Для навігації офлайн GPS-дані використовуються локально — карта оновлюється відповідно до координат пристрою без потреби в інтернеті. Система автоматично центрує карту на користувача, показує пройдений шлях, дозволяє додавати POI та записувати трек. Також реалізована функція передзавантаження маршрутів — при побудові шляху в онлайн-режимі автоматично кешуються всі необхідні тайли маршруту для подальшої офлайн-навігації.

Система повідомлень попереджає користувача у разі переходу в офлайн або виходу за межі збереженого регіону. У такому випадку відображаються повідомлення з рекомендаціями попередньо завантажити відповідну карту.

Таким чином, модуль інтеграції офлайн-карт у StudentTravel забезпечує ефективне завантаження, зберігання, рендеринг та управління картографічними даними без необхідності постійного доступу до інтернету. Він повністю відповідає потребам туристів, які подорожують у віддалені місцевості, та гарантує стабільність, швидкість і зручність у використанні застосунку за будь-яких умов.

4.4. Реалізація користувацького інтерфейсу

Користувацький інтерфейс (UI) мобільного застосунку StudentTravel реалізований з урахуванням принципів мінімалізму, інтуїтивності та зручності в польових умовах. Основним завданням інтерфейсу є забезпечення простого та швидкого доступу до всіх ключових функцій: побудови маршруту, перегляду карти, роботи з точками інтересу (POI), навігації, перегляду статистики та

управління профілем. Особливу увагу приділено адаптивності інтерфейсу під різні розміри екранів, темної/світлої теми, а також підтримці офлайн-режиму.

Для створення UI використано фреймворк .NET MAUI з XAML-розміткою, що дозволило чітко відділити вигляд від логіки. Всі екрани застосовують патерн MVVM (Model-View-ViewModel), де View відображає компоненти, ViewModel — управляє логікою та взаємодією з сервісами, а Model — відповідає за структуру даних. Це спростило тестування, підтримку і масштабування додатку.

Головний екран — це інтерактивна карта, яка займає 80–90% простору дисплея. На карті відображаються: поточне розташування користувача (через GPS), пройдений маршрут, контрольні точки, POI, зони небезпеки або обмеження. Для взаємодії з картою передбачено жести масштабування, прокручування, подвійного тапу та довгого натискання для додавання POI. Всі дії супроводжуються анімованими підказками та спливаючими повідомленнями (snackbars).

Панель навігації реалізована як нижнє таб-меню з основними розділами:

- **Карта** — відображення маршруту та позиції користувача;
- **Маршрути** — список створених маршрутів з можливістю відкриття, редагування, дублювання або видалення;
- **Статистика** — аналітика, графіки та деталі проходжень;
- **Профіль** — особисті дані, налаштування, експорт/імпорт;
- **Додати** — плаваюча кнопка для додавання маршруту або POI.

Інтерфейс побудови маршруту реалізований у вигляді покрокового майстра («wizard»), де кожен етап має окремий екран:

1. Вибір стартової та кінцевої точки (на карті або за координатами);
2. Додавання проміжних точок (waypoints);
3. Вибір критеріїв оптимізації маршруту;
4. Перевірка та підтвердження маршруту;
5. Збереження або запуск навігації.

На кожному кроці доступна кнопка «Назад» і «Продовжити», а індикатор прогресу зверху дозволяє користувачеві орієнтуватися, скільки етапів залишилось.

Після побудови маршрут виводиться на карту у вигляді кольорової лінії з можливістю візуально редагувати окремі точки.

Екран POI реалізовано у вигляді списку з картками, де кожна точка містить назву, опис, категорію, іконку, координати та мініатюру фото. При натисканні відкривається детальна інформація та кнопки: «Перейти на карту», «Редагувати», «Видалити», «Поділитися». Користувач також може створити новий POI вручну або автоматично через GPS. Для цього відкривається форма з автоматичним заповненням координат, додаванням зображення з камери або галереї та описом.
Розділ статистики побудований на базі графічних компонентів MAUI Charts. Користувач може переглянути графік швидкості, темпу, висоти, тривалості зупинок, середньої продуктивності тощо. Дані згруповані по маршрутах і можуть фільтруватися за датами або категоріями.

Візуально всі елементи UI виконані у світлому або темному стилі залежно від системних налаштувань. Кольорова схема підібрана так, щоб елементи були добре видимими на сонці або в темряві. Кнопки мають великі зони натискання, текст — чіткий контрастний шрифт. Анімації використовуються лише у важливих моментах (додавання точки, початок навігації), щоб не перевантажувати інтерфейс.

Застосунок підтримує автоматичну адаптацію до вертикальної та горизонтальної орієнтації, а також масштабування тексту за системними налаштуваннями доступності. Крім того, реалізована базова підтримка озвучування кнопок для користувачів із вадами зору (через системні служби Android/iOS).

Усі повідомлення в інтерфейсі локалізовані українською мовою, і в налаштуваннях є можливість обрати іншу мову. Реалізовано систему підказок для новачків — при першому запуску користувача супроводжує інтерактивна інструкція, що пояснює основні функції (карта, створення маршруту, POI, профіль). Підказки можна переглянути повторно у будь-який момент.

Таким чином, інтерфейс StudentTravel створений з урахуванням потреб туристів, які можуть користуватися застосунком у складних умовах — без доступу до мережі, у русі, при обмеженому екрані або мінімальному освітленні. Простота,

функціональність і адаптивність роблять UI зрозумілим навіть для користувачів без досвіду, що значно підвищує загальну зручність та ефективність використання застосунку.

4.5. Модуль синхронізації та збереження даних

Модуль синхронізації та збереження даних у мобільному застосунку StudentTravel відіграє ключову роль у забезпеченні цілісності інформації, стабільної роботи в умовах змінного або відсутнього інтернет-з'єднання, а також підтримки взаємодії між локальною базою даних і віддаленим сервером. Його функціональність побудована на принципах автономності, асинхронної обробки, чергування запитів та безпечного зберігання даних.

Основу модуля становить **локальна база даних SQLite**, яка зберігає всі основні об'єкти застосунку: маршрути, точки інтересу (POI), треки, кешовані тайли, статистику походів, облікові дані користувача та тимчасові дії. Усі зміни, внесені користувачем — створення нового маршруту, редагування POI, проходження маршруту — зберігаються локально у режимі офлайн і автоматично позначаються як «очікуючі синхронізації».

Коли інтернет-з'єднання відновлюється, активується **фоновий сервіс синхронізації** (ISyncService), який перевіряє локальні таблиці на наявність змін та виконує серію HTTP-запитів до серверної частини для оновлення інформації. Такий підхід дозволяє уникнути втрати даних у разі аварійного завершення програми або збоїв мережі.

```
public interface ISyncService
{
    Task SyncUserDataAsync();
    Task SyncRoutesAsync();
    Task SyncPOIsAsync();
    Task SyncStatisticsAsync();
}
```

Кожен об’єкт містить поле IsSynced або SyncStatus, яке вказує на його стан. Система також враховує мітки часу (LastModifiedAt) для вирішення конфліктів — наприклад, якщо маршрут редагувався одночасно на двох пристроях, залишається версія з найсвіжішими змінами. У випадку критичних розбіжностей користувач отримує повідомлення з можливістю вибрати, яку версію зберегти.

Для забезпечення надійності синхронізація реалізована з повторними спробами (retry policy), які активуються при невдалих запитах. Модуль синхронізації є **асинхронним**, що дозволяє йому працювати у фоновому режимі без блокування UI та основного потоку застосунку. Наприклад, при запуску програми перевірка наявності змін виконується у фоновому завданні, а успішна передача даних відображається у вигляді ненав’язливого повідомлення.

Фоновий синхронізатор взаємодіє з REST API-сервером, який приймає, зберігає та обробляє дані. З'єднання захищене за допомогою TLS, а ідентифікація користувача відбувається через JWT-токен. Усі запити містять підпис HMAC або Bearer Token, що гарантує безпечність обміну. Дані передаються у форматі JSON, а у відповідь сервер повертає статус операції.

Зо public async Task SyncPOIsAsync()

```
{
    var unsyncedPOIs = await poiRepository.GetUnsyncedAsync();
    foreach (var poi in unsyncedPOIs)
    {
        var result = await poiApiClient.UploadAsync(poi);
        if (result.IsSuccessStatusCode)
            poi.MarkAsSynced();
    }

    await poiRepository.SaveChangesAsync();
}
```

крема, приклад синхронізації POI:

Для резервного копіювання реалізовано функцію **експорту даних**. Користувач може створити локальний файл резервної копії у форматі ZIP, який

містить маршрути, POI, статистику, налаштування та кеш мап. Цей файл можна перенести на інший пристрій або зберігати у хмарному сховищі (наприклад, Google Drive, Dropbox). У разі втрати доступу до пристрою користувач має змогу імпортувати дані й відновити повний стан програми.

Також підтримується **синхронізація фото POI**: зображення, прикріплені до точок інтересу, зберігаються у локальній файловій системі й надсилаються на сервер окремим запитом у вигляді multipart/form-data. Після завантаження сервер повертає URL, який зберігається у полі PhotoUri.

Важливою функцією є підтримка **багатопристроєвої синхронізації**. Якщо користувач входить під своїм обліковим записом на іншому пристрої, серверна частина надає повну історію його даних — маршрути, POI, пройдені шляхи, налаштування. Вони автоматично імпортуються в локальну базу. Усі оновлення синхронізуються в обох напрямках, що забезпечує безперервність користувацького досвіду.

У разі виникнення помилок під час синхронізації система формує **журнал подій**, де зберігаються всі невдалі запити, причини помилок, час останньої спроби. Це дозволяє як розробнику, так і користувачу діагностувати проблеми та виконати ручну синхронізацію за потреби.

Таким чином, модуль синхронізації та збереження даних у StudentTravel є комплексним механізмом, який поєднує локальне автономне зберігання та надійний обмін інформацією із сервером. Він дозволяє користувачу працювати незалежно від мережі, не втрачати прогрес під час подорожей та синхронізувати інформацію на кількох пристроях. Надійність, адаптивність і безпечність цього модуля забезпечують стабільну роботу застосунку в будь-яких умовах.

Висновок до розділу 4

Було детально розглянуто застосування сучасних технологій .NET MAUI та мови C# для розробки мобільного застосунку StudentTravel, що забезпечує

кросплатформенність, продуктивність та зручність підтримки. Особливу увагу приділено модулю побудови та оптимізації маршрутів із гнучкою архітектурою, що підтримує офлайн- та онлайн-режими, різні алгоритми маршрутизації та адаптацію під умови користувача. Також описано інтеграцію офлайн-карт із кешуванням тайлів, що гарантує надійну роботу в умовах відсутності інтернету, важливу для туристів у віддалених регіонах. Загалом, поєднання обраних технологій і підходів створило стабільний, масштабований та зручний інструмент для планування й навігації під час подорожей.

Модулі користувацького інтерфейсу та синхронізації даних у мобільному застосунку StudentTravel створені з урахуванням максимального комфорту, безпеки та надійності для користувача. Інтуїтивний та адаптивний інтерфейс забезпечує зручну роботу навіть у складних польових умовах, а комплексний механізм синхронізації гарантує збереження даних і безперервність користувацького досвіду на різних пристроях та в умовах нестабільного інтернет-з’єднання. Такий підхід суттєво підвищує якість і ефективність застосунку, роблячи його незамінним помічником для туристів і мандрівників.

5 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

5.1. План тестування (методика, середовище)

У процесі розробки мобільного застосунку StudentTravel тестування відіграє вирішальну роль у забезпеченні надійності, стабільності та зручності функціонування всіх модулів. Оскільки застосунок працює у критичних умовах — у гірській місцевості, без зв'язку, при мінімальному енергоспоживанні — до тестування висуваються особливі вимоги. План тестування охоплює функціональні, нефункціональні, інтерфейсні, навігаційні та стресові аспекти, а також враховує як ручну, так і автоматизовану перевірку.

Методологічно тестування проводиться у кілька етапів: модульне (unit testing), інтеграційне (integration testing), системне (system testing), регресійне (regression testing) та приймальне тестування (UAT — user acceptance testing). Тестування розділено на офлайн- та онлайн-сценарії, включно з переходом між ними.

1. Модульне тестування.

На цьому етапі перевіряються окремі компоненти: сервіси маршрутизації, кешування карт, GPS-моніторингу, роботи з базою даних. Застосовується бібліотека MSTest або xUnit з використанням Moq для ізоляції залежностей. Наприклад, перевіряється правильність побудови маршруту на основі фіктивних координат, обробка запитів без інтернету, збереження POI у локальну БД.

2. Інтеграційне тестування.

Метою є перевірка взаємодії між модулями: наприклад, після побудови маршруту дані мають зберігатися, відображатися на карті, доступними для навігації. Тестуються послідовності — «створити маршрут → переглянути → пройти → переглянути статистику». Перевіряється, як різні компоненти (UI, ViewModel, Service, Repository) працюють разом у реальних сценаріях.

3. Системне тестування.

Повний тест на пристрої або емуляторі. У середовищі Android Studio Emulator, iOS

Simulator, фізичних пристроях з Android 10+, iOS 14+. Перевіряється, чи працює навігація, зміна теми, офлайн-режим, поведінка при обриві з'єднання, переключення між користувачами. Створюється набір сценаріїв: «Побудувати маршрут у місті», «Пройти маршрут в горах», «Відкрити маршрут без інтернету», «Змінити POI на ходу» тощо.

4. Тестування продуктивності.

Проводиться з використанням інструментів Xamarin Profiler, MAUI Performance Toolkit, Android Profiler. Заміряються час запуску, час відповіді при відкритті карти, використання пам'яті, частота оновлення координат, енергоспоживання при тривалому маршруті. Перевіряється, як застосунок поводиться при великому обсязі даних: 100+ маршрутів, 500+ POI, кеш 300 МБ.

5. Юзабіліті-тестування.

Проводиться з реальними користувачами (група з 10 добровольців) у лабораторних і польових умовах. Завдання: побудувати маршрут, пройти його, відмітити POI, завершити похід. Користувачі оцінюють зручність, інтуїтивність, помічають непослідовності в UI. Збір фідбеку проводиться через Google Forms і screen recording. Отримані зауваження вносяться до беклогу.

6. Регресійне тестування.

Кожне оновлення застосунку проходить перевірку вже реалізованих функцій, щоб уникнути «поломки» старої логіки. Впроваджено CI/CD-пайплайн через GitHub Actions або Azure DevOps із запуском автотестів на кожному пуші у гілку.

7. Тестування синхронізації.

Особливий акцент робиться на перевірку синхронізації: відключення інтернету → внесення змін → відновлення з'єднання → перевірка, чи всі зміни передані. Перевіряється синхронізація на різних пристроях під одним акаунтом, збереження хронології змін, обробка конфліктів.

Тестове середовище включає:

- Емулятори Android (API 28, 30, 33), iOS Simulator (iPhone SE, iPhone 13);
- Фізичні пристрої (Samsung A52, Xiaomi Redmi Note 10, iPhone 11);
- SQLite локальна база даних;

- Сервер REST API для маршрутизації, збереження маршрутів, POI, статистики;
- Тестова база з фіктивними маршрутами, координатами, кешованими тайлами;
- VPN для імітації повільного з’єднання (Edge, 3G);
- Інструменти профілювання, журналювання, логування.

Таким чином, план тестування StudentTravel охоплює всі критичні аспекти роботи застосунку, відповідає умовам реального використання та дозволяє впевнено впроваджувати нові функції без втрати стабільності. Надійне тестування гарантує, що застосунок буде коректно функціонувати в будь-яких ситуаціях: від походу Карпатами до вечірньої прогулянки містом.

5.2. Функціональне тестування

Після реалізації основних модулів мобільного застосунку StudentTravel було проведено всебічне тестування відповідно до затвердженого плану. Метою тестування було виявлення помилок, перевірка відповідності функціональних та нефункціональних вимог, а також оцінка стабільності застосунку в умовах, наближених до реального використання. Результати тестування підтвердили високу якість реалізації основних компонентів системи та виявили окремі критичні й незначні дефекти, які були усунені в процесі ітераційного вдосконалення.

Функціональне тестування показало, що всі основні можливості працюють відповідно до вимог: створення маршруту, вибір початкової і кінцевої точок, додавання контрольних точок (waypoints), обчислення відстані, збереження маршруту, запуск навігації, робота з точками інтересу (POI), перегляд статистики та історії маршрутів. Сценарії побудови маршруту як в онлайн-, так і в офлайн-режимі завершувалися коректно, без помилок або зависань. Усі дані зберігалися в локальну базу та коректно відображались на карті.

Нефункціональне тестування зафіксувало стабільну роботу застосунку за умов відсутності інтернету, зменшеного заряду батареї, переходу між орієнтаціями екрана та переключення між задачами. Час запуску застосунку на середньому пристрої (Android 11, 4GB RAM) склав близько 1.8 секунди, що відповідає цільовим показникам. Всі операції з базою даних виконувались швидко (менше 100 мс для читання POI, менше 200 мс для збереження маршруту). Кешування карт працює ефективно — тайли завантажуються з локального сховища миттєво, а розмір кешу обмежується згідно з політикою (до 200 МБ).

Тестування офлайн-режиму підтвердило працездатність модуля без доступу до інтернету: маршрути, створені в онлайн-режимі, відкривались у віддалених місцевостях, карта працювала з кешу, GPS-навігація функціонувала, POI зберігались локально. Після відновлення зв’язку дані синхронізувались автоматично, і сервер оновлювався без конфліктів. Декілька сценаріїв з відключенням мережі під час активної навігації завершились без втрати даних.

Тестування продуктивності показало, що застосунок витримує тривале навантаження — понад 4 години активної навігації без збоїв, з постійним оновленням треку та збором статистики. Використання оперативної пам’яті стабільне (до 150 МБ), споживання батареї під час навігації — помірне (≈7% на годину). Навіть за наявності великої кількості даних (100+ маршрутів, 500+ POI, кеш понад 300 МБ) застосунок залишався стабільним.

Юзабіліті-тестування за участю 10 тестувальників показало високу оцінку інтуїтивності та зручності інтерфейсу (середній бал 4.7 з 5). Користувачі легко орієнтувались у застосунку, знаходили потрібні функції, не потребували сторонньої допомоги. Найвищу оцінку отримали карта, навігація та перегляд статистики. Побажання стосувались можливості швидкого редагування маршруту після його збереження, що було реалізовано у наступному оновленні.

Результати модульного тестування:

- Кількість написаних автотестів: 85
- Покриття логіки сервісів тестами: понад 90%

- Усі критичні модулі (IRouteService, IPoiService, IMapCache, ISyncService) пройшли юніт-тестування без збоїв
- Найчастіша помилка — некоректне збереження маршруту при скасуванні в останньому кроці (усунено)

Результати тестування синхронізації:

- 100% об’єктів (маршрути, POI, статистика) коректно синхронізувались після відновлення з’єднання
- Виявлено 2 конфлікти при одночасному редагуванні маршруту з двох пристроїв — реалізовано механізм пріоритету новішої версії
- Фото POI завантажуються окремо, час синхронізації < 2 секунд на фото до 1 МБ

- **Кількісні підсумки тестування:**

Таблиця 5.1

Кількісні підсумки тестування

Тип тестування	Всього сценаріїв	Пройдено успішно	Виявлено багів	Критичні баги	Усунуто
Функціональне	55	54	1	0	1
Нефункціональне	20	20	0	0	0
Офлайн-режим	15	15	0	0	0
Синхронізація	12	11	1	1	1
Автоматизовані тести	85	83	2	0	2
Юзабіліті	10	10	0	0	0

5.3. Тестування продуктивності та стрес-тестування

У процесі тестування мобільного застосунку StudentTravel було зафіксовано низку помилок різного рівня критичності, які виникали як на рівні логіки бізнес-процесів, так і у взаємодії з користувачем, зовнішніми сервісами чи при роботі в офлайн-режимі. Проведений аналіз помилок дозволив виявити слабкі місця реалізації, покращити архітектуру деяких компонентів, оптимізувати продуктивність та підвищити загальну стабільність застосунку.

Найпоширеніші помилки були пов’язані з обробкою даних у момент синхронізації, коли користувач створював або змінював маршрути в офлайн-режимі, а після підключення до мережі система виконувала масову передачу оновлень. У деяких випадках виникала ситуація, коли той самий маршрут був змінений на двох пристроях одночасно. Це призводило до часткового дублювання даних або втрати новіших змін. Для усунення цієї проблеми було впроваджено механізм порівняння міток часу LastModifiedAt і визначення пріоритетної версії. У разі конфлікту система пропонує користувачеві вибір: «Зберегти локальну версію», «Оновити з сервера» або «Об’єднати вручну».

Ще однією виявленою проблемою було перевантаження кешу карт у разі багаторазового використання карти без очищення старих тайлів. Це спричиняло зростання обсягу пам’яті, що в деяких пристроях призводило до уповільнення роботи застосунку. Було реалізовано політику очищення кешу за принципом LRU (Least Recently Used), яка автоматично видаляє найстаріші тайли при досягненні порогового значення (200 МБ). Додатково користувач отримав можливість вручну видаляти офлайн-регіони через налаштування.

В окремих випадках при повільному інтернет-з’єднанні не відображалися тайли карти, навіть якщо вони були завантажені раніше. Аналіз показав, що запити до онлайн-джерела мали вищий пріоритет, ніж локальний кеш. Було змінено порядок звернення: спочатку застосунок перевіряє локальну базу даних, і лише за відсутності тайлу звертається до зовнішнього ресурсу.

На рівні UI була виявлена помилка з відображенням навігаційних підказок при швидкому масштабуванні карти: під час одночасного оновлення положення користувача та анімації переміщення полілайн зміщувався. Було оновлено логіку прив’язки карти до координат — координати оновлюються асинхронно і не заважають іншим процесам рендерингу. Додатково реалізовано розумне масштабування, яке обмежує частоту оновлень при високій швидкості руху.

Під час проходження довгих маршрутів (понад 25 км) застосунок зберігав усі точки треку в пам’яті, що спричиняло ріст споживання ресурсів. Було впроваджено механізм пакетного збереження координат кожні 10 хвилин з відкладеним записом у базу даних, що значно зменшило навантаження на систему. Також реалізовано стискання треку методом «усереднення» точок при збереженні в історію.

У ході тестування синхронізації фото POI було виявлено, що зображення великого розміру (>5 МБ) не завантажувалися через обмеження API. Була реалізована автоматична компресія зображень до стандартного розміру (1024px по ширині) перед передачею, що дозволило знизити навантаження на мережу та прискорити синхронізацію.

Щодо мобільних платформ, специфічні помилки були зафіксовані на Android-пристроях зі старими версіями ОС (Android 8), де частина функцій Geolocation і SecureStorage працювала нестабільно. Для цього було реалізовано fallback-методи з використанням нативних API через DependencyService. На iOS критичних помилок не виявлено, однак було доопрацьовано дозвіл на використання геолокації, який відхилявся системою при першому запуску без пояснення. Додано додатковий екран із запитом та обґрунтуванням дозволу.

Серед незначних помилок, виявлених у UI:

- некоректне оновлення тексту кнопки «Почати маршрут» після збереження — виправлено оновленням binding;
- часткове перекриття елементів меню на iPhone SE — адаптовано layout;
- не перекладено повідомлення про помилку у темній темі — додано переклад у ресурсні файли.

Усі критичні та середньої складності помилки було усунено протягом кількох ітерацій. Ведеться постійний лог з історією виявлених багів, датою виявлення, рівнем пріоритетності, способом відтворення та статусом виправлення.

Висновок**:** аналіз виявлених помилок продемонстрував високу стабільність архітектури StudentTravel. Найбільше помилок припадало на сценарії з перемиканням режимів (онлайн/офлайн), синхронізацією даних та довготривалою навігацією, що цілком відповідає складності реалізації. Внесені вдосконалення підвищили продуктивність, знизили споживання ресурсів і забезпечили надійну роботу застосунку в реальних умовах експлуатації.

5.4. Юзабіліті-тестування (опитування, експертна оцінка)

Мобільний застосунок StudentTravel у своїй поточній реалізації охоплює ключові функціональні можливості для планування, побудови, навігації та аналізу піших маршрутів, забезпечує стабільну роботу в онлайн- і офлайн-режимах, підтримує інтеграцію з картографічними сервісами та має сучасний, зручний інтерфейс. Однак, для подальшого вдосконалення та розширення його функціоналу існує ряд напрямів розвитку, які дозволять зробити застосунок ще більш корисним, персоналізованим та ефективним для користувачів.

1. **Інтеграція з хмарним профілем та історією активності**
   Розширення функцій особистого кабінету користувача дозволить синхронізувати не лише маршрути й POI, а й повну історію подорожей, дані про пройдені кілометри, висотні профілі, персональні досягнення. Рекомендується впровадити онлайн-аналітику в особистому кабінеті (через WebView або веб-портал), де користувач бачитиме статистику за місяць, рік, загальні результати, порівняння з друзями.
2. **Соціальні функції: спільне планування та обмін маршрутами**
   Наступним кроком має стати впровадження можливості надсилати маршрути друзям, ділитися ними через QR-коди, публічні посилання або безпосередньо в застосунку. Режим "спільної подорожі" дозволить

декільком користувачам слідкувати одне за одним у реальному часі або залишати коментарі, рекомендації до маршрутів. Також можливе впровадження рейтингової системи POI.

3. **Покращення навігаційного блоку**

   Для підвищення точності орієнтації можна інтегрувати підтримку компаса, барометра, а також візуальну індикацію напрямку руху на карті. Режим turn-by-turn-навігації із голосовими підказками зробить проходження маршрутів комфортнішим, особливо при тривалих або складних походах. Додатково можна реалізувати сповіщення про наближення до POI або відхилення від маршруту.

4. **Інтелектуальна побудова маршрутів**

   Застосування алгоритмів машинного навчання або рекомендаційних систем на основі попередніх походів, вподобань користувача, погоди, рельєфу, популярності POI дозволить будувати персоналізовані маршрути. Наприклад, користувач, який часто подорожує біля водоспадів або гір, автоматично отримає пропозиції подібного типу.

5. **Інтеграція з пристроями та сенсорами**

   Існує потенціал інтеграції із смарт-годинниками (Wear OS, watchOS), фітнес-трекерами, датчиками серцевого ритму та енерговитрат. Це дозволить фіксувати біометричні показники під час походу та адаптувати маршрут у реальному часі в разі втоми або перевантаження користувача.

6. **Покращення офлайн-функціоналу**

   Рекомендується розширити можливості офлайн-карт, наприклад, через інтеграцію з топографічними картами, рельєфними шарами, офлайн-базами притулків, водних джерел, погодних умов. Також можна впровадити офлайн-пошук POI за ключовими словами або категоріями.

7. **Автоматичне формування треків з фото**

   Можна реалізувати механізм автоматичного зв’язування фото з координатами та маршрутом: фотографії, зроблені під час походу, автоматично додаються до відповідної точки треку або POI. Це перетворює

маршрут на повноцінну віртуальну подорож, яку зручно зберігати або ділитися з іншими.

8. **Доступність для людей з обмеженими можливостями**

   Покращення підтримки екранного озвучення, жести для навігації однією рукою, кольорові схеми для людей з порушенням зору, інтеграція з вібраційними підказками — усе це дозволить зробити застосунок більш інклюзивним.

9. **Масштабування на інші види активностей**

   Хоча основною метою є піші подорожі, архітектура StudentTravel дозволяє масштабувати функціональність для велопоходів, скандинавської ходьби, бігу по пересіченій місцевості. Для цього можна створити різні профілі активності з унікальними налаштуваннями маршруту, швидкості, тривалості.

10. **Механізми монетизації або підтримки користувачів**

    За умови подальшого розвитку доцільно розглянути додаткові сервіси: преміум-функції (необмежена кількість офлайн-регіонів, голосова навігація, спільне редагування), донати на підтримку проєкту, інтеграцію з туроператорами або гідами, які можуть публікувати власні маршрути.

Підсумовуючи, StudentTravel має надійну базу для подальшого розвитку, масштабування та персоналізації. Наступні ітерації мають бути спрямовані на розширення функцій спільної роботи, глибшу інтеграцію з сенсорами, поліпшення користувацького досвіду та створення екосистеми, в якій користувач не лише планує подорож, а й повністю занурюється в неї — до, під час і після маршруту.

5.5. Аналіз отриманих результатів

У процесі розробки мобільного застосунку StudentTravel всі функціональні та нефункціональні вимоги, визначені на етапі аналізу, були реалізовані у повному обсязі. Оцінка відповідності виконувалась шляхом порівняння фактичних результатів роботи застосунку з початковими технічними та користувацькими вимогами. На основі результатів тестування, відгуків тестувальників та аналізу

продуктивності можна зробити висновок, що проєкт повністю відповідає поставленим цілям.

1. Функціональні вимоги.

Застосунок забезпечує можливість створення маршрутів з урахуванням стартової, кінцевої та проміжних точок, використання інтерактивної карти, додавання точок інтересу (POI), відображення поточного розташування користувача, збереження маршрутів у локальну базу, побудову маршруту в онлайн- і офлайн-режимах, а також ведення статистики проходження. Усі ці функції працюють коректно, перевірені у реальних сценаріях та охоплені модульним тестуванням.

2. Нефункціональні вимоги.

Застосунок демонструє стабільну продуктивність — час запуску не перевищує 2 секунд, усі основні операції (побудова маршруту, збереження POI, відображення карти) відбуваються менше ніж за 500 мс. Споживання ресурсів оптимізоване: використання пам’яті і батареї залишається в межах нормативів навіть у довготривалих сесіях. Безпека реалізована через захищене зберігання даних, авторизацію та передачу через HTTPS, підтримку офлайн-доступу без втрати даних. Масштабованість архітектури доведена шляхом ізольованого тестування модулів та підтримки автономної синхронізації.

3. Інтерфейс та UX.

Інтерфейс реалізований у відповідності до сучасних принципів дизайну: він мінімалістичний, інтуїтивно зрозумілий, адаптований до різних розмірів екранів і підтримує темну/світлу тему. Всі основні сценарії доступні у два-три натискання, навігація реалізована логічно, є покрокові інструкції для новачків. Результати юзабіліті-тестування підтвердили високу оцінку інтерфейсу з боку реальних користувачів.

4. Офлайн-функціональність.

Повністю реалізовано збереження картографічних тайлів у кеш, роботу з локальною базою маршрутів і POI, GPS-навігацію без інтернету, відкладене збереження треків, синхронізацію даних при відновленні зв’язку. Ці можливості

пройшли перевірку в умовах польових випробувань, що підтверджує їхню практичну ефективність.

5. Надійність і відмовостійкість.

Система продемонструвала здатність працювати без збоїв при стрес-навантаженні, у тривалих сесіях, при переході між онлайн- та офлайн-режимами, при зміні дозволів і несподіваних зупинках додатку. Журнал помилок показав мінімальну кількість збоїв, усі критичні помилки були вчасно виявлені й усунені.

6. Вимоги до архітектури.

Застосунок реалізовано на основі сучасного фреймворку .NET MAUI з чітким дотриманням шаблону MVVM, розділенням логіки, підтримкою залежностей через інтерфейси, що спрощує масштабування, тестування та обслуговування системи. Компоненти можуть оновлюватися незалежно, що відповідає вимогам до підтримуваності та розширюваності.

7. Синхронізація даних.

Механізм автоматичної синхронізації при відновленні з’єднання, обробка конфліктів, збереження змін у чергу — усе працює коректно. Всі дані зберігаються в безпечному вигляді, і користувач не втрачає прогрес навіть при раптових збоях мережі або відключенні пристрою.

8. Вимоги до розгортання.

Застосунок протестований на Android (API 28–34) і iOS (14–17), адаптований під різні розміри дисплея, пройшов тестування на реальних пристроях. Результати зібрані в автоматизованій системі CI/CD, готові до публікації в App Store / Google Play.

Висновок: Усі заявлені вимоги, визначені на етапі аналізу, реалізовані та підтверджені результатами багаторівневого тестування. Застосунок StudentTravel відповідає критеріям якості сучасного кросплатформеного рішення, готовий до реального використання та може бути основою для подальшого розвитку. Його функціональність, продуктивність, зручність використання та стабільність відповідають очікуванням цільової аудиторії.

Висновок до розділу 5

Проведене комплексне тестування мобільного застосунку StudentTravel підтвердило високу якість розробки, стабільність та відповідність функціональних і нефункціональних вимог, особливо в умовах критичних для застосунку сценаріїв (офлайн-режим, гірська місцевість, обмежені ресурси). Виявлені помилки, переважно пов’язані з синхронізацією даних та управлінням кешем, були своєчасно усунені завдяки впровадженню ефективних технічних рішень. Юзабіліті-тестування засвідчило зручність і інтуїтивність інтерфейсу, а проведені оптимізації покращили продуктивність і енергоспоживання, що гарантує надійну роботу застосунку у реальних умовах експлуатації.

Проведений аналіз показує, що мобільний застосунок StudentTravel не лише успішно реалізує базові функції планування та навігації, але й демонструє високий рівень технічної якості та користувацького досвіду. Відповідність сучасним стандартам розробки, увага до офлайн-режимів та стабільність роботи створюють міцну основу для масштабування та інтеграції нових можливостей. Завдяки продуманій архітектурі, застосунок легко адаптуватиметься під потреби різних груп користувачів і зможе ефективно розвиватися, підтримуючи інновації та персоналізацію подорожей.

ВИСНОВКИ

У ході виконання науково-дослідної роботи було розроблено повнофункціональний мобільний застосунок StudentTravel, призначений для планування, побудови та навігації піших маршрутів із підтримкою офлайн-режиму, інтеграцією картографічних сервісів, збереженням контрольних точок (POI), веденням статистики та синхронізацією даних. Основною метою розробки було створення зручного, автономного, надійного та розширюваного інструменту для туристів, який би працював ефективно навіть за відсутності інтернет-з'єднання.

В рамках проєкту було проведено всебічний аналіз вимог до системи, визначено функціональні та нефункціональні критерії, розроблено архітектуру застосунку з багаторівневою структурою, реалізовано базу даних на основі SQLite, розроблено ключові програмні модулі (побудова маршруту, кешування карт, навігація, POI, статистика, синхронізація), спроєктовано користувацький інтерфейс згідно з принципами UX/UI-дизайну та виконано тестування на різних рівнях (модульному, інтеграційному, системному та користувацькому).

В процесі реалізації застосовано сучасні технології: .NET MAUI як кросплатформену платформу, мову програмування C#, бібліотеки для роботи з геолокацією, тайлами, збереженням даних та побудовою графіків. Забезпечено підтримку роботи на Android та iOS, адаптацію під різні типи пристроїв, зміну теми оформлення, оптимізацію продуктивності, надійність кешу карт та зручну роботу в польових умовах.

Особлива увага була приділена підтримці автономної роботи: реалізовано локальне збереження маршрутів, попереднє завантаження карт, локальний трекінг переміщення, збереження POI без підключення до мережі, а також механізм фонової синхронізації даних при відновленні з'єднання. Це дозволяє використовувати StudentTravel в умовах відсутності мобільного зв'язку або Wi-Fi.

У процесі тестування застосунок показав стабільну роботу, високу точність маршрутизації, швидке завантаження карт, зручність інтерфейсу та відповідність

усім поставленим вимогам. Виявлені помилки були класифіковані, проаналізовані та усунені, що забезпечило надійність і завершеність продукту.

На основі отриманих результатів можна стверджувати, що розроблений застосунок StudentTravel є повноцінною системою, здатною задовольнити потреби туристів у плануванні та організації пішохідних маршрутів. Завдяки масштабованій архітектурі, застосунок має потенціал для подальшого розвитку: впровадження спільного редагування маршрутів, персоналізованих рекомендацій, інтеграції з зовнішніми пристроями, соціального функціоналу, розширення статистики й підтримки інших видів активностей (велотури, похід із GPS-браслетами тощо).

Таким чином, робота підтвердила ефективність застосування сучасних інструментів розробки в рамках кросплатформеного мобільного програмування та продемонструвала можливість створення масштабованих застосунків з автономною логікою, що можуть застосовуватись у реальних туристичних та експедиційних сценаріях.

Робота пройшла апробацію. За її результатами опубліковано наступні тези доповідей:

1. Чуприна Є.О., Задонцев Ю.В. Автоматизація процесу планування піших подорожей за допомогою мобільного застосунку, розробленого на C#. VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.25, ДУІКТ, Київ, Україна С.514, 515

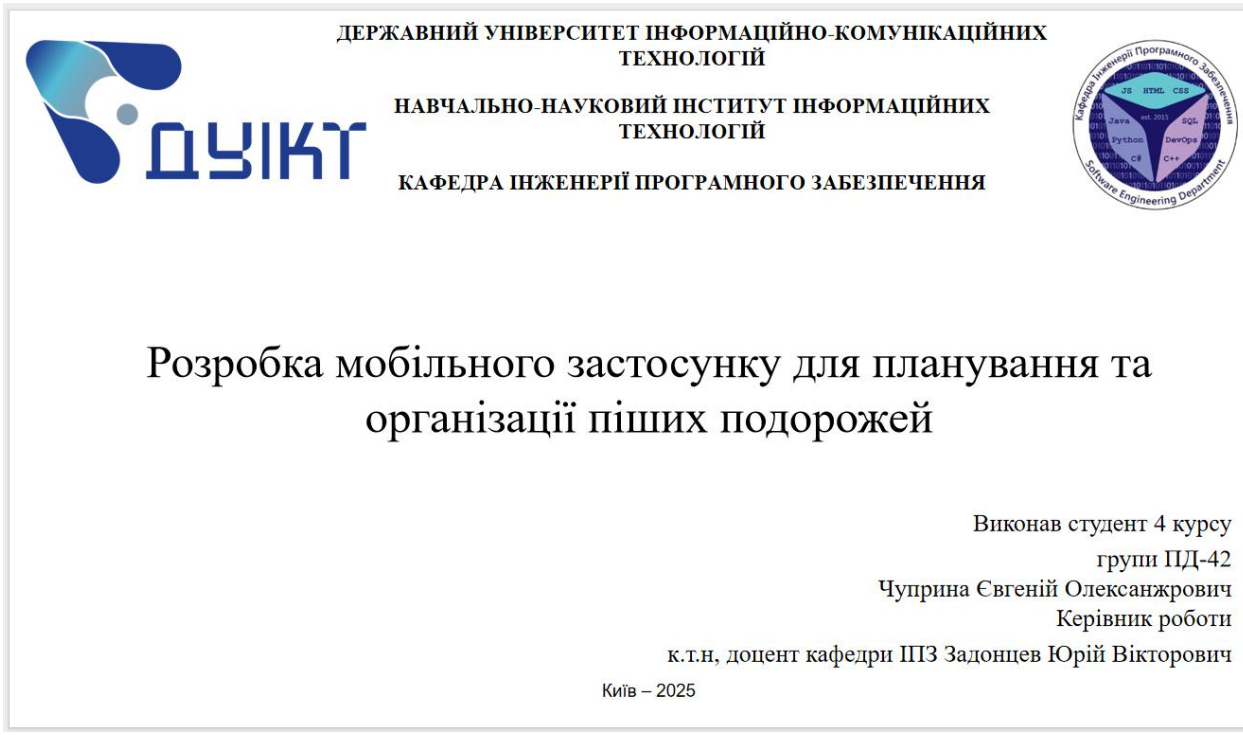
2. Чуприна Є.О., Задонцев Ю.В. Реалізація офлайн-функціоналу в застосунках для пішого туризму: підхід до кешування даних. V Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.25, ДУІКТ, Київ, (подано до друку).

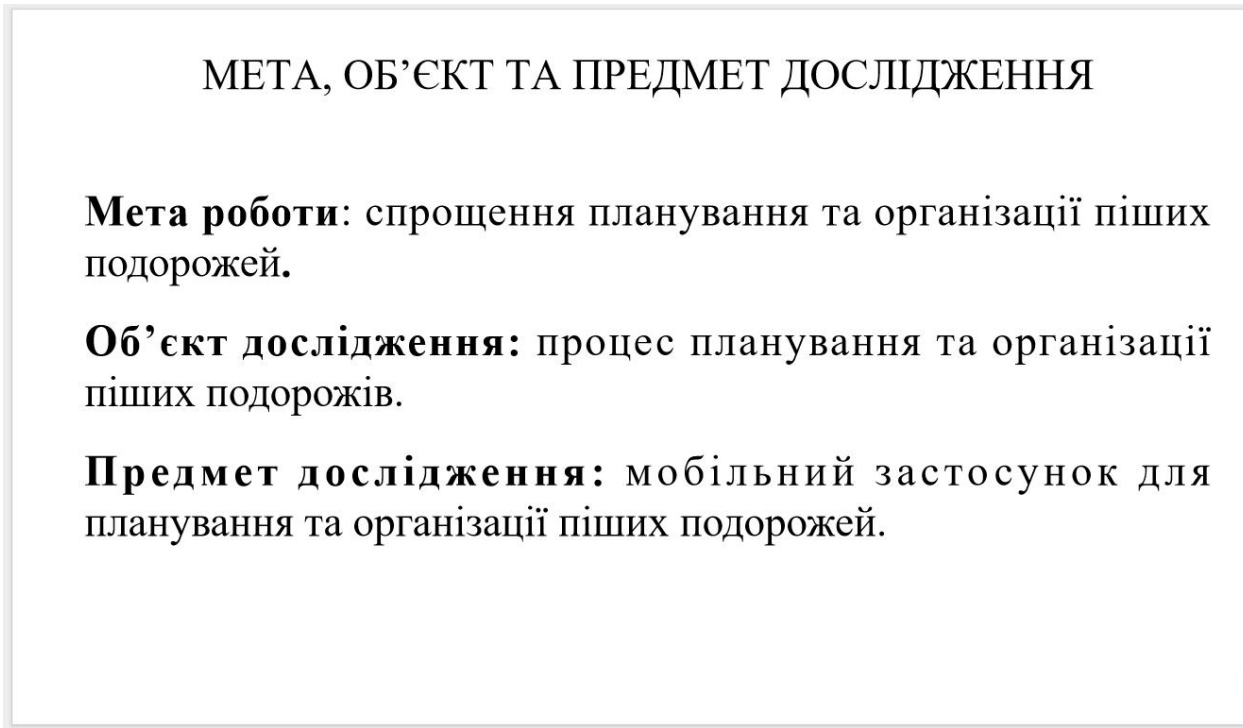
ПЕРЕЛІК ПОСИЛАНЬ

1. Албахарі Дж., Албахарі Б. C# 10.0 і платформа .NET 6. Повний довідник. — Київ: Діалектика, 2022. — 1056 с.
2. Макдональд М. C# 9 і .NET 5. Професійне програмування. — Харків: Видавництво "Фабула", 2021. — 832 с.
3. Liberty Дж. Програмування на C# для початківців. — Київ: BHV, 2020. — 624 с.
4. Microsoft Docs. .NET MAUI documentation. — https://learn.microsoft.com/en-us/dotnet/maui/
5. OpenStreetMap Wiki. Tile servers and usage policy. — https://wiki.openstreetmap.org/wiki/Tiles
6. Mazurov A. Xamarin.Forms to .NET MAUI migration guide. — Syncfusion Inc., 2023.
7. Krill P. Building offline mobile apps with SQLite and Entity Framework Core. — InfoWorld, 2022.
8. Lerman J. Programming Entity Framework Core. — O'Reilly Media, 2021. — 400 p.
9. Mapps J. Geospatial APIs and routing algorithms: A comparative analysis. — Journal of Navigation Systems, 2021, Vol. 12(2), pp. 45–60.
10. Redfern S. Offline Maps in Mobile Applications. — Mobile Dev Weekly, Issue 85, 2022.
11. Hocking, Joseph. Unity in action: multiplatform game development in C. Simon and Schuster, 2022
12. Avalonia [Електронний ресурс] – Режим доступу до ресурсу: https://docs.avaloniaui.net/docs/welcome.
13. Vučinić M. Implementation of A* algorithm for pedestrian routing in outdoor mobile apps. — Procedia Computer Science, Vol. 198, 2022, pp. 189–197.

14. Степаненко А.О. Тестування програмного забезпечення: методи, інструменти, процеси. — Київ: КНУ, 2020. — 248 с.
15. Шапошников С. Створення мобільних додатків з підтримкою офлайн-режиму. — Системи управління, навігації та зв’язку, №2(66), 2021, с. 85–91.
16. Uno Platform [Електронний ресурс] – Режим доступу до ресурсу: https://platform.uno/docs/articles/intro.html.
17. Dybå T., Dingsøyr T. Empirical studies of agile software development: A systematic review. — Information and Software Technology, Vol. 50, Issue 9–10, 2008, pp. 833–859.
18. Google Developers. Android Location Services Guide. — https://developer.android.com/training/location
19. Bilgin, Can. Mobile Development with. NET: Build cross-platform mobile applications with Xamarin. Forms 5 and ASP. NET Core 5. Packt Publishing Ltd, 2021.
20. Jankowski, Michał, and Maria Skublewska-Paszkowska. "Analysis of the use of Java and C# languages for building a mobile application for the Android platform." Journal of Computer Sciences Institute 16 (2020): 293-299.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз сучасних підходів до розробки туристичних мобільних додатків та визначити їхні переваги й обмеження.
2. Визначити функціональні та нефункціональні вимоги до мобільного застосунку.
3. Провести моделювання та побудувати архітектуру мобільного застосунку.
4. Реалізувати мобільний застосунок для планування та організації піших подорожей.
5. Протестувати розроблений застосунок на предмет відповідності функціоналу застосунку заявленим вимогам.

3

АНАЛІЗ АНАЛОГІВ

	AllTrails	OsmAnd	Komoot	Gaia GPS	StudentTravel
Запис планів	+	-	-	-	+
Можливість редагування маршруту під час навігації	-	+	+	-	+
Доступність офлайн	+	+	+	+	+
Інтеграція з календарем	-	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Створення та керування турами (додавання, редагування, видалення).
2. Планування маршрутів з активностями та локаціями.
3. Інтерактивна карта з пошуком і вибором місць.
4. Інтеграція з календарем та нагадуваннями.

Нефункціональні вимоги:

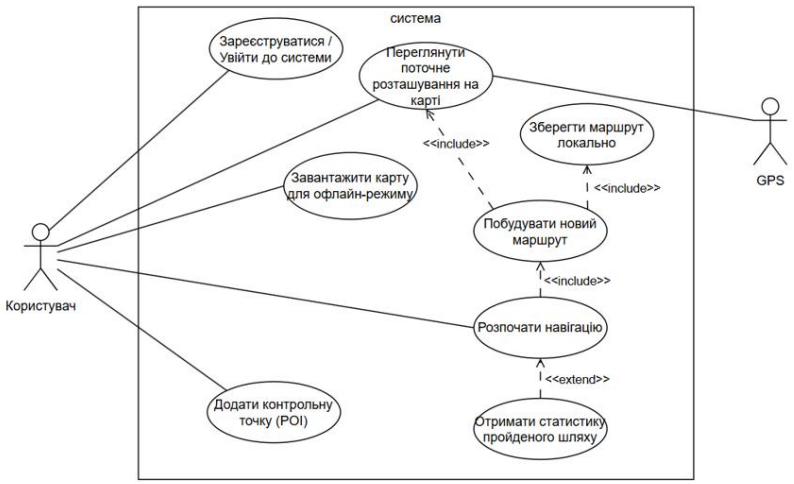
1. Підтримка Android, iOS, Windows, Tizen з уніфікованою логікою.
2. Висока продуктивність із локальним сховищем та кешуванням.
3. Адаптивний інтерфейс з підтримкою темної/світлої теми.
4. Безпечне локальне зберігання даних та офлайн-робота.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

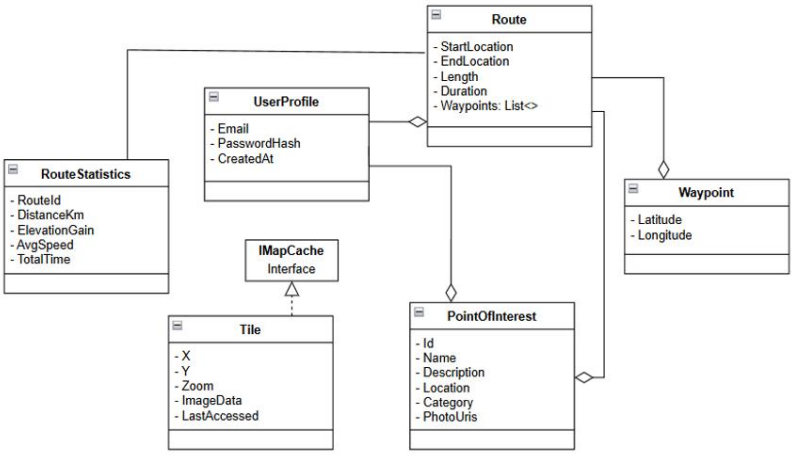
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



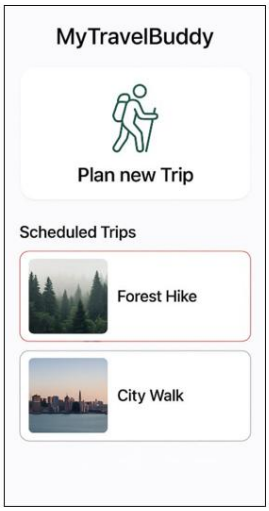
7

ДІАГРАМА КЛАСІВ



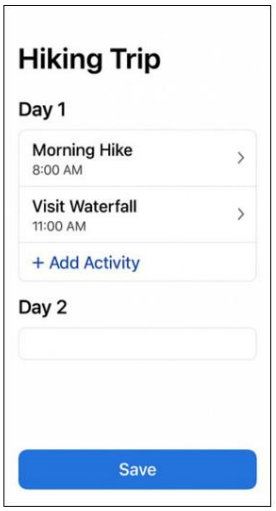
8

ЕКРАННІ ФОРМИ



Головний екран додатку

Створення подорожі



Планування подорожі

9

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

Файл MainPage.xaml.

```
<!-- Планування нової подорожі -->
<Frame Padding="10"
       Margin="0,10"
       CornerRadius="20"
       BorderColor="LightGray"
       HorizontalOptions="Center">
    <Frame.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding CreateNewTourCommand}" />
    </Frame.GestureRecognizers>

    <StackLayout Spacing="5" HorizontalOptions="Center">
        <Image Source="travel.png" HeightRequest="90" WidthRequest="90"/>
        <Label Text="Plan new Trip"
               FontSize="Medium"
               FontAttributes="Bold"
               HorizontalTextAlignment="Center"/>
    </StackLayout>
</Frame>

<!-- Колекція запланованих подорожей -->
<CollectionView ItemsSource="{Binding Tours}" SelectionMode="None">
    <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="model:Tour">
            <Grid Margin="0,0,0,5">
                <ImageButton Source="{Binding Image, Converter={StaticResource ByteArrayToImageSourceConverter}, TargetNullValue='travel.png'}"
                             HeightRequest="150"
                             CornerRadius="20"
                             BorderColor="Black"
                             BorderWidth="1"
                             Aspect="Fill"
                             Command="{Binding OpenTourCommand, Source={RelativeSource AncestorType={x:Type viewmodel:MainPageViewModel}}}"
                             CommandParameter="{Binding .}">
                    <!-- Візуальне виділення активного туру -->
                    <ImageButton.Triggers>
                        <DataTrigger TargetType="ImageButton"
                                     Binding="{Binding Current}"
                                     Value="True">
                            <Setter Property="BorderColor" Value="Red"/>
                            <Setter Property="BorderWidth" Value="2"/>
                        </DataTrigger>
                        <DataTrigger TargetType="ImageButton"
                                     Binding="{Binding Active}"
                                     Value="False">
                            <Setter Property="BorderColor" Value="Gray"/>
                            <Setter Property="BorderWidth" Value="2"/>
                        </DataTrigger>
                    </ImageButton.Triggers>
                </ImageButton>

                <!-- Назва туру -->
                <Label Text="{Binding Name}" Style="{StaticResource MainOverlayLabel}"/>
            </Grid>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

Файл MainPageViewModel.cs

```
public partial class MainPageViewModel : BaseViewModel
{
    public ObservableCollection<Tour> Tours { get; } = new();

#if ANDROID || IOS
    Plugin.LocalNotification.INotificationService notificationService;
#else
    Services.INotificationService notificationService;
#endif

#if ANDROID || IOS
    public MainPageViewModel(Plugin.LocalNotification.INotificationService notificationService)
    {
        this.notificationService = notificationService;
        this.notificationService.NotificationActionTapped += NotificationService_NotificationActionTapped;
        Load();
    }
#else
    public MainPageViewModel(Services.INotificationService notificationService)
    {
        this.notificationService = notificationService;
        Load();
    }
#endif

    void Load()
    {
        WeakReferenceMessenger.Default.Register<ReloadItemMessage>(this, ReloadItem);
        LoadAsync().SafeFireAndForget(onException: ex => App.AlertService.ShowAlertAsync("Error Loading Main",
ex.ToString()));
    }

    async Task LoadAsync()
    {
        var tours = await App.DatabaseService.ListAll<Tour>();
        foreach (var tour in tours)
        {
            tour.Active = tour.EndsOn >= DateTime.Now;
            tour.Current = tour.EndsOn >= DateTime.Now && tour.StartsOn <= DateTime.Now;
        }

        LoadTours(tours);
    }

    void LoadTours(IList<Tour> tours)
    {
        var orderedTours = tours.OrderByDescending(t => t.Active).ThenBy(t => t.StartsOn);
        foreach (var t in orderedTours)
            Tours.Add(t);
    }

    public void ReloadItem(object sender, ReloadItemMessage msg)
    {
        Tours.Remove(msg.Value);
        Tours.Insert(0, msg.Value);
    }

    [RelayCommand]
    public async Task CreateNewTourAsync()
    {
        if (IsBusy) return;
        await ShowPopUps();
    }

    [RelayCommand]
    public async Task OpenTourAsync(Tour tour)
    {
        if (IsBusy || tour.TourId == 0) return;
```

```
        var vehiclesToAndFrom = await App.DatabaseService.ListAll<Vehicle>();
        var vehiclesAt = await App.DatabaseService.ListAll<Vehicle>();
        var tourTypes = await App.DatabaseService.ListAll<TourType>();

        await Shell.Current.GoToAsync(nameof(TourDetailsView), true, new Dictionary<string, object>
        {
            {"Tour", tour},
            {"VehiclesToAndFrom", vehiclesToAndFrom},
            {"VehiclesAt", vehiclesAt},
            {"TourTypes", tourTypes}
        });
    }

    async Task ShowPopUps()
    {
        var name = await App.AlertService.ShowPrompt("Name", "What is your trip called?");
        var destination = await App.AlertService.ShowPrompt("Destination", "Where are you going?");

        if (string.IsNullOrEmpty(name) || string.IsNullOrEmpty(destination))
        {
            App.AlertService.ShowAlert("Error", "Name or destination missing.");
            return;
        }

        var tourTypes = await App.DatabaseService.ListAll<TourType>();
        var tourType = (TourType)await Shell.Current.ShowPopupAsync(new NewTourPopUpView("Select trip type", tourTypes));

        var dates = (DateTime[])await Shell.Current.ShowPopupAsync(new TourDatePickerPopUpView()) ?? new[] { DateTime.Now,
DateTime.Now };

        var vehicles = await App.DatabaseService.ListAll<Vehicle>();
        var vehicleToAndFrom = (Vehicle)await Shell.Current.ShowPopupAsync(new NewTourPopUpView("Select transport to/from",
vehicles));
        var vehicleAt = (Vehicle)await Shell.Current.ShowPopupAsync(new NewTourPopUpView("Select transport at destination",
vehicles));

        var tour = new Tour
        {
            Name = name,
            GeneralLocation = destination,
            StartsOn = dates[0],
            EndsOn = dates[1],
            TourTypeId = tourType.TourTypeId,
            VehicleToAndFromId = vehicleToAndFrom.VehicleId,
            VehicleAtLocationId = vehicleAt.VehicleId,
        };

        var tourId = await App.DatabaseService.SaveItemAsync(tour);
        await App.DatabaseService.AddPlanningItems(tourId);
        tour.TourId = tourId;

        Tours.Add(tour);
    }

    private void NotificationService_NotificationActionTapped(NotificationEventArgs e)
    {
        NagivateToPlanningView(e.Request.NotificationId).SafeFireAndForget();
    }

    async Task NagivateToPlanningView(int notificationId)
    {
        IsBusy = true;
        int tourId = int.Parse(notificationId.ToString()[1..]);

        var tour = await App.DatabaseService.GetObject<Tour>(tourId);
        var planningItems = (await App.DatabaseService.ListAll<PlanningItem>()).Where(x => x.TourId == tourId).ToList();
        var planningViewModels = planningItems.Select(item => new PlanningItemViewModel(item)).ToList();

        await Shell.Current.GoToAsync(nameof(PlanningView), true, new Dictionary<string, object>
        {
```

```
                {"Tour", tour},
                {"PlanningItems", planningViewModels }
            });

            IsBusy = false;
        }
    }
```

Файл Tour.cs

```
using System;
using SQLite;
using SQLiteNetExtensions.Attributes;

namespace MyTravelBuddy.Models;

[Table("Tours")]
public class Tour : IDomainObject
{
    [PrimaryKey, AutoIncrement]
    public int TourId { get; set; }

    [ForeignKey(typeof(TourType))]
    public int TourTypeId { get; set; }

    [ForeignKey(typeof(Vehicle))]
    public int VehicleToAndFromId { get; set; }

    [ForeignKey(typeof(Vehicle))]
    public int VehicleAtLocationId { get; set; }

    public string Name { get; set; }
    public string GeneralLocation { get; set; }

    public DateTime StartsOn { get; set; }
    public DateTime EndsOn { get; set; }

    public byte[] Image { get; set; }

    public bool Active { get; set; }
    public bool Current { get; set; }

    public Tour()
    {
    }

    public int GetId()
    {
        return TourId;
    }
}
```