Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для організації та управління груповими подорожами. Серверна частина
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-3

Дар'я ТОПЧІЙ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доцент кафедри ПІ Дмитро КОЛЕСНИКОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Топчій Дар'ї Дмитривні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для організації та управління груповими подорожами. Серверна частина

Затверджена наказом по університету від 19.05. 2025р. № 397 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10.06.2025

3. Вихідні дані до роботи Розробити серверну частину програмної системи для організації групових подорожей з можливістю обміну повідомленнями, відстеження біометричних, формування статистики, генерації та реагування на надзвичайні ситуації. Реалізацію виконати мовою програмування C# із використанням платформи ASP.NET Core 8, бази даних PostgreSQL, бібліотеки Entity Framework Core.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2025	*виконано*
2	Створення специфікації ПЗ	10.04.2025	*виконано*
3	Проектування ПЗ	12.04.2025	*виконано*
4	Розробка ПЗ	24.04.2025	*виконано*
5	Тестування ПЗ	21.05.2025	*виконано*
6	Оформлення пояснювальної записки	22.05.2025	*виконано*
7	Підготовка презентації та доповіді	25.06.2025	*виконано*
8	Попередній захист	08.06.2025	*виконано*
9	Нормоконтроль, рецензування	08.06.2025	*виконано*
10	Здача роботи у електронний архів	10.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	10.06.2025	*виконано*

Дата видачі завдання «8» «квітня» 2025р.

Здобувач _____ Дар’я ТОПЧІЙ
(підпис)

Керівник роботи ____ доцент кафедри ПІ Дмитро КОЛЕСНИКОВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи бакалавра: 132 стор., 5 розділів, 21 рис., 18 джерел.

БІОМЕТРИЧНІ ДАНІ, БЕЗПЕКА, МОБІЛЬНИЙ ЗАСТОСУНОК, МОНІТОРИНГ, РЕАЛЬНИЙ ЧАС, СЕРВЕРНА ЧАСТИНА, СИСТЕМИ ГРУПОВИХ ПОДОРОЖЕЙ, ТРЕКІНГОВІ ДАНІ, ШИФРУВАННЯ, API, ASP.NET, GPS-ТРЕКІНГ, POSTGRESQL.

Об’єкт розробки – серверна частина програмної системи для організації групових подорожей, що включає моніторинг місцезнаходження та фізичного стану учасників у реальному часі. Мобільний клієнт взаємодіє з сервером через REST API.

Мета роботи – проектування серверної частини для обробки даних учасників подорожей, включаючи координати та біометричні показники, їх зберігання і формування статистики.

Методи реалізації включають використання ASP.NET Core 8 для серверної частини, PostgreSQL для зберігання даних, Entity Framework Core для взаємодії з базою, а також JWT-токени для авторизації та AES-шифрування для захисту даних.

В результаті роботи розроблено серверну частину програмного запезпечення, що здійснює обробку та зберігання трекінгових даних і біометрії учасників, а також надає API для клієнтського застосунку.

ABSTRACT

Explanatory note to bachelor’s qualification work: 132 pages, 5 chapters, 21 figures, 18 sources.

API, ASP.NET, BIOMETRIC DATA, ENCRYPTION, GPS TRACKING, MOBILE APPLICATION, MONITORING, POSTGRESQL, REAL-TIME, SERVER PART, SYSTEMS OF GROUP TRAVELS, TRACKING DATA.

The object of the development is the server part of the software system for organizing group trips, which includes real-time monitoring of participants' location and physical condition. The mobile client interacts with the server via REST API.

The aim of the work is design and implementation of the server part for processing data of participants in group trips, including coordinates and biometric indicators, their storage, and statistics generation.

Methods of implementation include the use of ASP.NET Core 8 for the server part, PostgreSQL for data storage, Entity Framework Core for database interaction, as well as JWT tokens for authentication and AES encryption for data protection.

As a result of the work, the server part of the software was developed, which handles the processing and storage of tracking data and biometric information of participants and provides the API for the mobile application.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface, інтерфейс прикладного програмування

AES – Advanced Encryption Standard, стандарт симетричного блочного шифрування

GPS – Global Positioning System, глобальна система позиціонування

HTTPS – HyperText Transfer Protocol Secure, протокол захищеної передачі гіпертексту

IP – Internet Protocol, протокол міжмережевої взаємодії

JWT – JSON Web Token, формат передачі токенів авторизації

MA – Moving Average, рухоме середнє

ORM – Object-Relational Mapping, об'єктно-реляційне відображення

REST – Representational State Transfer, архітектурний стиль для побудови вебсервісів

SOLID – п’ять принципів об’єктно-орієнтованого проєктування (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion)

UI – User Interface, користувацький інтерфейс

UML – Unified Modeling Language, уніфікована мова моделювання

ВСТУП

У сучасному інформаційному середовищі зростає потреба у програмних системах, які дозволяють ефективно організовувати спільну діяльність. Вони забезпечують безпеку, взаємозв’язок і моніторинг учасників у режимі реального часу. Однією з актуальних сфер застосування таких систем є організація групових подорожей – пішохідних походів, велосипедних мандрівок, спортивних тренувань тощо. У подібних випадках важливими є координація учасників, оперативний обмін повідомленнями та можливість швидкого реагування у критичних ситуаціях. Розробка спеціалізованого програмного забезпечення, що дозволяє автоматизувати процес керування такими подорожами, є актуальним завданням прикладної інформатики.

Метою бакалаврської роботи є створення серверної частини програмної системи, яка забезпечує організацію групових подорожей з функціями створення та управління маршрутами, збору GPS-даних та вимірювання частоти серцебиття, обміну повідомленнями між учасниками та виклику екстреної допомоги. Основними вимогами до системи є безпечність, масштабованість, надійність, підтримка обробки даних у режимі, наближеному до реального часу, та дотримання принципів чистого коду й модульної архітектури.

Реалізацію серверної частини здійснено на базі сучасного фреймворку ASP.NET Core 8.0. Для зберігання даних застосовано реляційну СКБД PostgreSQL із використанням ORM-бібліотеки Entity Framework Core. З метою забезпечення безпеки реалізовано токенну автентифікацію (JWT) та симетричне шифрування персональних даних за допомогою алгоритму AES. Для розподілу навантаження та ефективної обробки великої кількості записів у таблиці трекінгу впроваджено партиціювання за часовою ознакою. Передбачено централізоване логування дій користувачів, а також документування API засобами Swagger.

Серверна частина побудована за принципами класичної трирівневої архітектури, що включає рівні представлення (контролери), бізнес-логіки (сервіси) та доступу до даних (репозиторії). У проєкті застосовано об’єкти передавання даних (DTO), впровадження залежностей та механізми валідації запитів. Усі компоненти реалізовані з урахуванням принципів DRY та SOLID. Під час розробки використовувалась система контролю версій Git.

У ході виконання роботи закріплено знання з архітектури програмного забезпечення, нормалізації баз даних, розробки API, безпечної обробки даних, а також моделювання з використанням UML. Реалізовано діаграми варіантів використання, розгортання, класів та структури бази даних.

Об’єктом дослідження є процес організації групових подорожей із моніторингом стану учасників і маршруту руху. Предметом дослідження є архітектура серверної частини програмної системи, що забезпечує реалізацію зазначених функцій із дотриманням вимог безпеки, модульності та масштабованості.

У результаті виконання бакалаврської роботи було спроєктовано та реалізовано серверну частину програмної системи, яка забезпечує безпечну та організовану підтримку групових подорожей у режимі, наближеному до реального часу. Система відповідає поставленим вимогам і демонструє можливість інтеграції технологій збору фізіологічних та геолокаційних даних у єдиний архітектурний розв’язок. Отримані результати підтверджують актуальність обраної теми та доцільність впровадження подібних систем у практичні сценарії, пов’язані з колективною активністю, безпекою й реагуванням у надзвичайних ситуаціях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

У сучасному світі активний відпочинок і групові подорожі стрімко набирають популярності. Згідно з дослідженням [1], глобальний ринок застосунків для планування групових поїздок оцінюється в понад $250 млн з очікуваним темпом зростання понад 15% на рік, що свідчить про високу динаміку розвитку галузі. Основними драйверами такого зростання є цифровізація туризму, популяризація активного стилю життя серед молоді та потреба у простих інструментах для координації груп.

На ринку виділяються три ключові гравці – Troupe, Wanderlog та Komoot.

Troupe – це мобільний застосунок та вебдодаток, створений компанією JetBlue Travel Products для організації подорожей у невеликих групах. Сервіс насамперед орієнтований на невеликі групи друзів або родичів, де важливо враховувати думку кожного [2]. Основна логіка роботи сервісу побудована навколо групового прийняття рішень. Кожен учасник групи може пропонувати локації, варіанти житла або дати поїздки. Після цього інші учасники голосують, використовуючи систему пріоритетів із трьома рівнями: перше, друге і третє місце. Результати голосування доступні в окремому розділі, що спрощує остаточний вибір.

Ілюстрація стартового екрану, що наведена на рисунку 1.1, містить просту інструкцію щодо початку роботи. На екрані вказано, що потрібно увійти або зареєструватися, після чого стануть доступні основні функції планування.

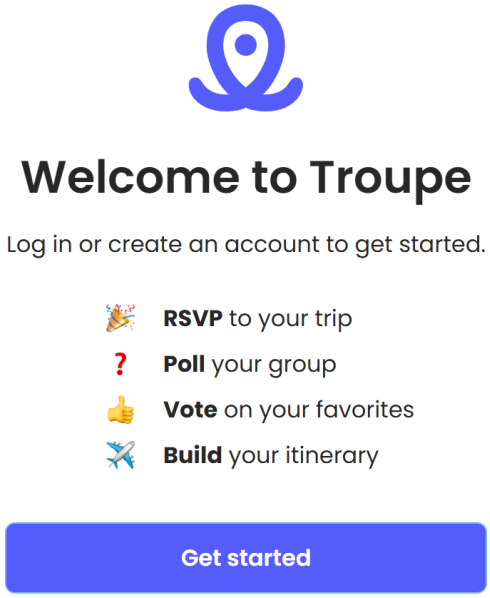

Рисунок 1.1 – Стартовий екран застосунку Troupe (рисунок виконано самостійно)

Інтерфейс програми поділений на вкладки: напрямки, проживання та нотатки. У розділі напрямків відображається список запропонованих локацій із фотографіями, назвами міст, посиланням на Google Maps і кнопками для коментарів і перегляду голосування. Цей інтерфейс показано на рисунках 1.2 і 1.3.

Рисунок 1.2 – Інтерфейс голосування за напрямок подорожі в застосунку Troupe (рисунок виконано самостійно)

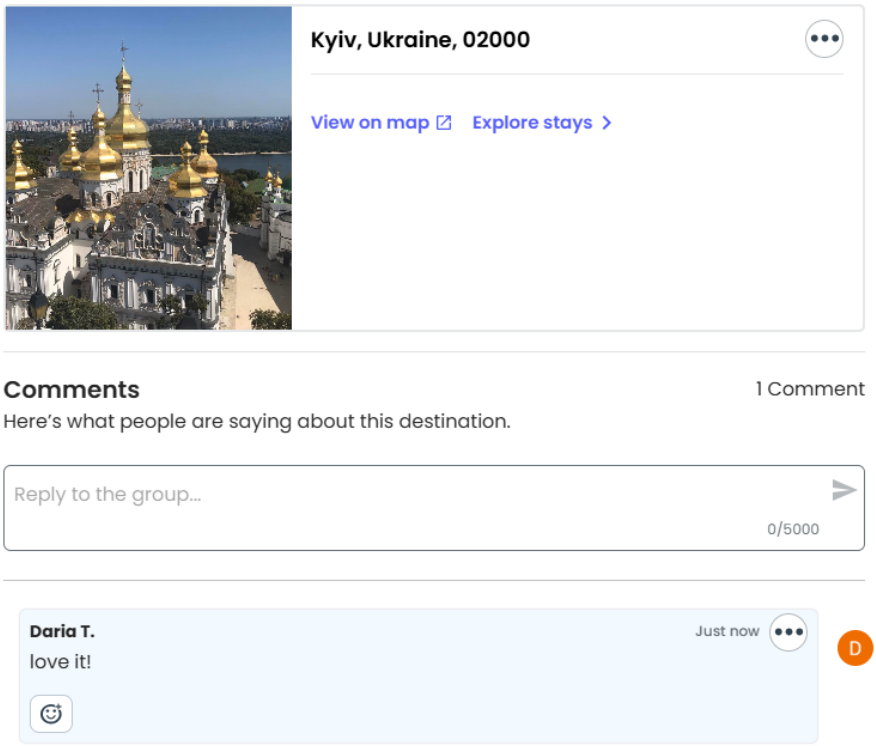

Рисунок 1.3 – Секція коментарів до напрямку в застосунку Troupe (рисунок виконано самостійно)

Користувач може натиснути на блок коментарів і перейти до списку повідомлень від інших учасників. Кожне повідомлення містить ім'я, дату й текст. Наприклад, у коментарі один користувач пропонує зустрітися в Мехіко та скуштувати місцеву кухню, інший підтверджує участь. Це дозволяє фіксувати думки групи щодо кожної конкретної пропозиції.

У розділі проживання реалізовано аналогічну модель вибору варіантів житла. У вкладці нотаток зберігаються ключові деталі подорожі, включаючи ідеї для активностей або важливі нагадування. Основні функції застосунку наведено на рисунку 1.4. Там показано, що користувач після входу в обліковий запис має змогу приєднатися до подорожі, створити опитування, проголосувати за улюблені варіанти та сформувати маршрут.

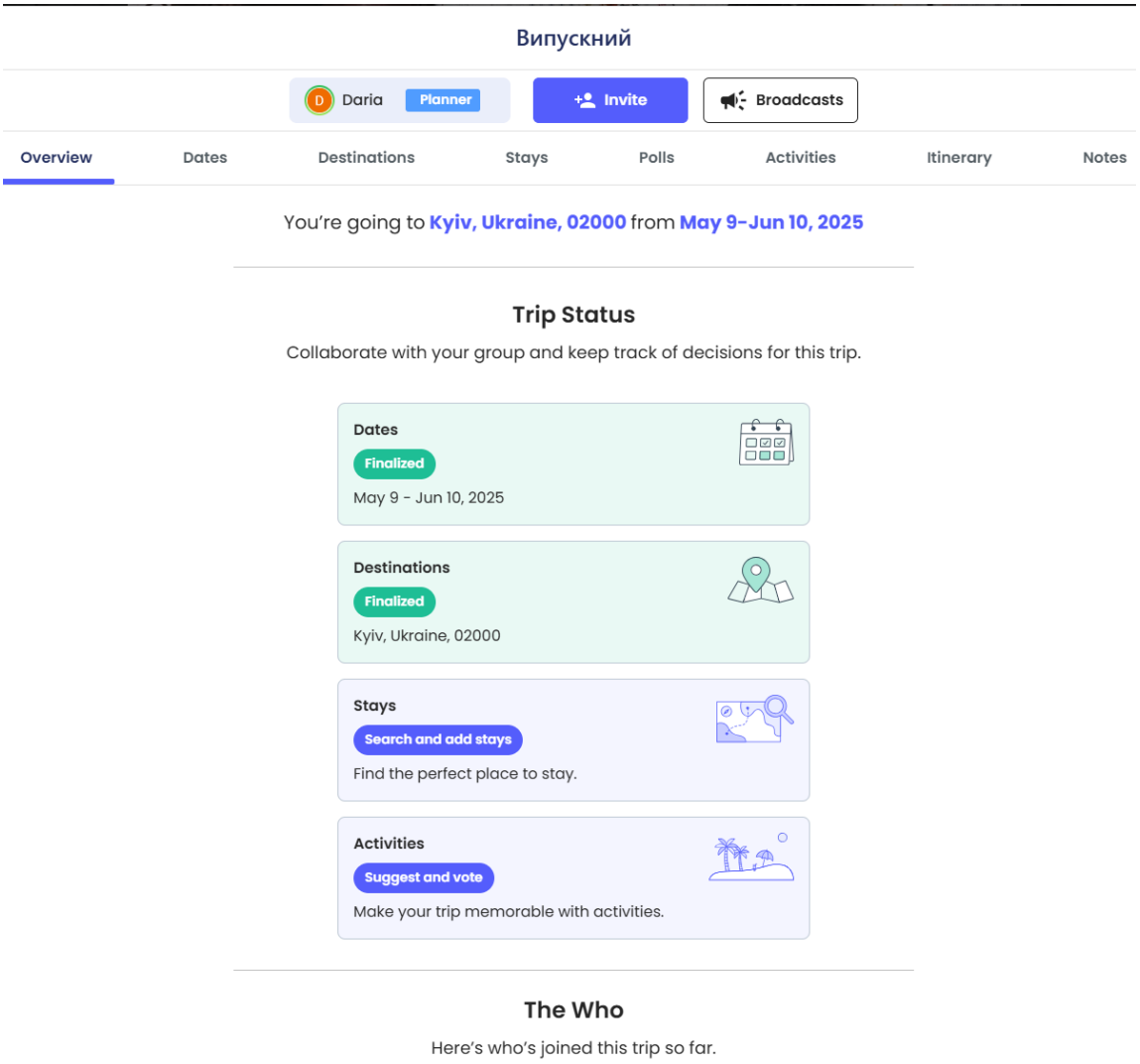

Рисунок 1.4 – Головна навігаційна панель застосунку Troupe (рисунок виконано самостійно)

Серед позитивних сторін варто відзначити зручність голосування, можливість коментування кожного варіанту окремо, просту навігацію, інтуїтивну структуру та відсутність перевантаженості інтерфейсу. Усі дії фіксуються в одному середовищі, що дозволяє використовувати Troupe як основну платформу для координації групових поїздок на етапі планування.

Серед недоліків можна виділити відсутність функцій супроводу подорожі в реальному часі. У застосунку не реалізовано збір GPS-координат або фізіологічних даних користувачів. Також відсутня можливість фіксації подій під час подорожі або виклику допомоги. Програма не має історії

подорожей або інструментів для створення особистого кабінету з архівом активностей. Уся взаємодія обмежується попереднім етапом підготовки до подорожі.

Wanderlog – це один із найпотужніших сучасних інструментів для планування подорожей, що орієнтований як на індивідуальних користувачів, так і на малі туристичні групи [3]. Його ключові можливості охоплюють створення детального маршруту, імпорт бронювань, категоризацію зупинок, офлайн-доступ до даних, розподіл витрат і роботу з нотатками. Візуальний інтерфейс програми зручний для редагування, а збереження структури маршруту виконується автоматично під час змін з будь-якого пристрою.

На рисунку 1.5 наведено стартовий екран, де користувач обирає наявну поїздку або створює нову.

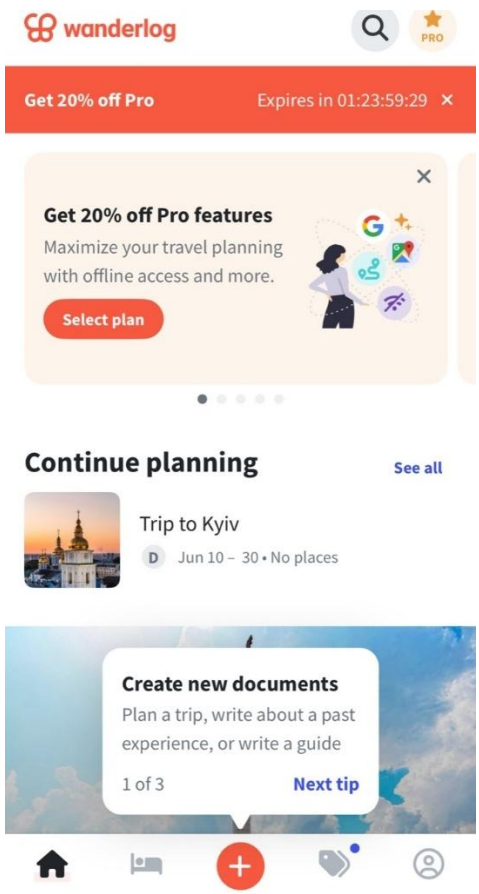

Рисунок 1.5 – Головний екран застосунку Wanderlog (рисунок виконано самостійно)

Розділ Explore, зображений на рисунку 1.6, дозволяє шукати популярні об’єкти в місті, фільтрувати їх за категоріями, а також отримати доступ до офлайн-гайдів.

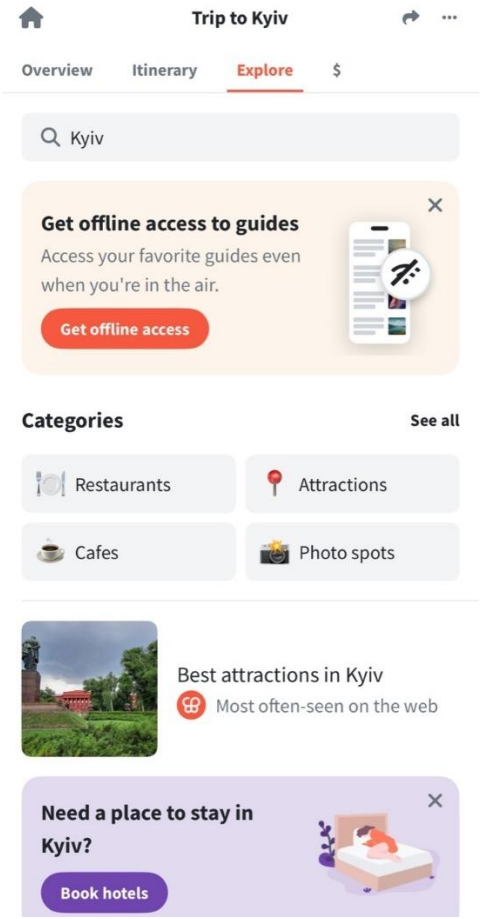

Рисунок 1.6 – Розділ Explore для подорожі в місті Київ (рисунок виконано самостійно)

Рисунок 1.7 демонструє структуру планування подорожі від додавання бронювань до прикріплення нотаток, включаючи розділи Overview, Itinerary і вкладення документів.

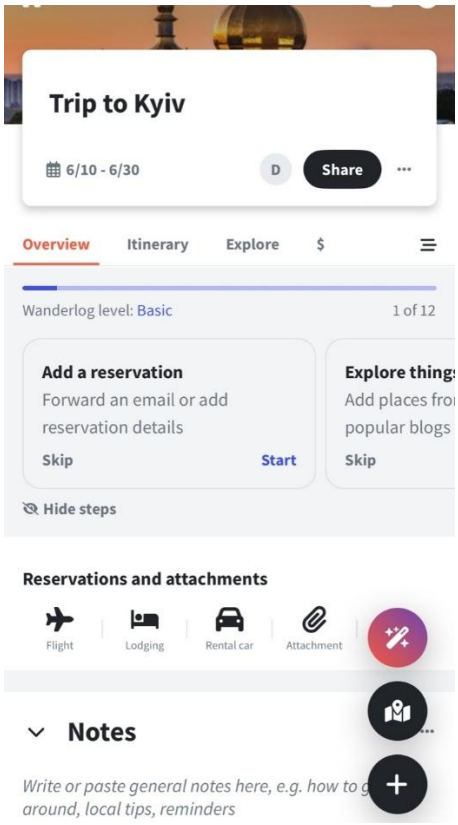

Рисунок 1.7 – Редактор маршруту в застосунку Wanderlog (рисунок виконано самостійно)

На рисунку 1.8 показано перелік функцій, доступних у базовій та Pro-версіях, серед яких присутні автоматичне сканування пошти, експорт у Google Maps та необмежена кількість вкладень.

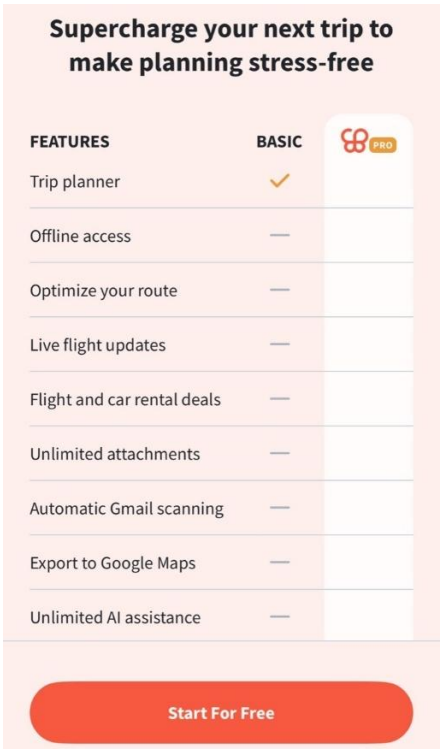

Рисунок 1.8 – Перелік функцій базового і Pro-плану в Wanderlog (рисунок виконано самостійно)

Попри потужний набір інструментів, Wanderlog має обмеження, які критично впливають на застосування в умовах реального пересування групи. Серед недоліків варто виділити відсутність моніторингу фізіологічного стану учасників, зокрема таких параметрів, як серцебиття або рівень активності. Не реалізовано збір GPS-даних у реальному часі з кожного пристрою групи, що унеможливлює точне відстеження переміщення або втрату зв’язку з окремим учасником. Також у застосунку відсутні механізми виклику допомоги або інші протоколи екстреного реагування.

Таким чином, попри універсальність і широкий функціонал у сфері планування, Wanderlog не покриває потреби, пов’язані з безпекою, контролем стану та координацією в реальному часі. Це створює передумови для розробки окремої системи, яка фокусується не лише на маршрутах, а й на моніторингу фізичного стану учасників, зборі геоданих, комунікації в критичних ситуаціях і зберіганні повної статистики для аналізу подорожі після її завершення.

Komoot – це цифрова платформа для планування маршрутів, створена компанією komoot GmbH, що базується в Німеччині [4]. Основне призначення застосунку є побудовою індивідуальних маршрутів для активного відпочинку, включаючи піші походи, велосипедні поїздки, біг, альпінізм та гірські подорожі. Сервіс працює на основі векторних топографічних карт з високою деталізацією та підтримує побудову треків з урахуванням типу активності, складності рельєфу, покриття доріг та доступності локацій.

Однією з ключових особливостей Komoot є система персоналізованих рекомендацій, які базуються на поведінковій моделі користувача, вибраній активності та попередніх маршрутах. Кожен побудований маршрут автоматично доповнюється інформацією про набір висоти, тип покриття, точну довжину, часову оцінку та профіль складності. Усі дані синхронізуються з мобільним застосунком, доступні офлайн і можуть бути експортовані у форматах GPX для сторонніх пристроїв.

На рисунку 1.9 зображено головний екран застосунку Komoot з вибором активності. Користувач може вибрати тип маршруту: пішохідний похід, біг, гірський велоспорт, шосейний велосипед або прогулянка. Вибір напряму відбувається вручну або за допомогою GPS-геопозиції.

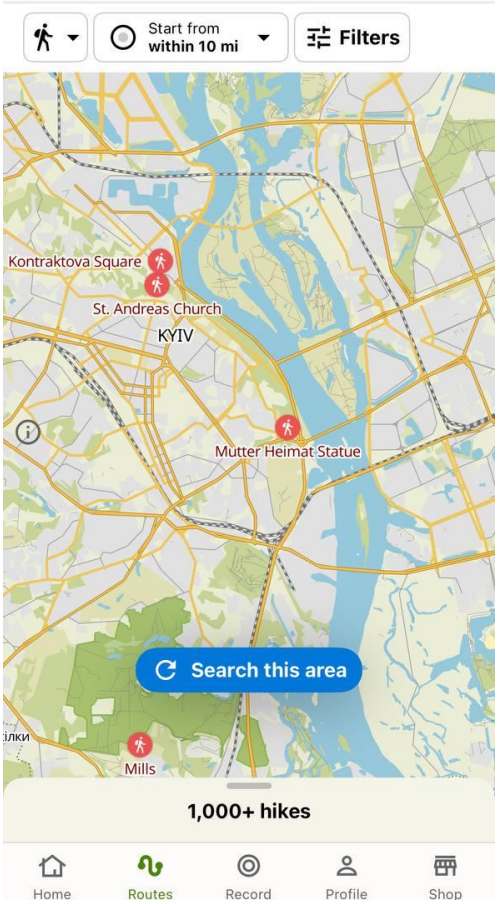

Рисунок 1.9 – Головний екран застосунку Komoot із вибором активності та побудовою маршруту (рисунок виконано самостійно)

Після генерації маршруту сервіс автоматично формує детальний план з усіма точками зупинок. Це показано на рисунку 1.10. Кожна точка містить інформацію про вид покриття, очікуваний час проходження та висотний профіль. Також доступна опція завантаження маршруту в телефон для офлайн-навігації.

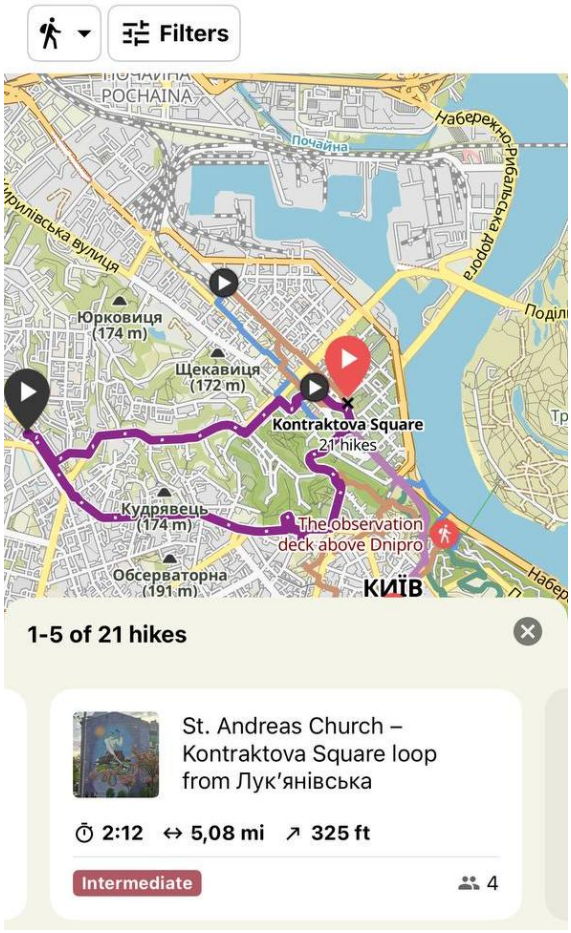

Рисунок 1.10 – Побудований маршрут у застосунку Komoot із висотним профілем і покриттям (рисунок виконано самостійно)

Застосунок активно підтримує українську територію як повноцінний маршрутний регіон. Картографічне покриття України повністю доступне, а маршрути відображаються з однаковою точністю, як і в країнах ЄС. У списках регіонів для завантаження офлайн-карт присутній прапор України, який добре візуалізований і легко помітний серед інших країн. Це ілюструє включення України до основної інфраструктури підтримки Komoot та свідчить про її визнання як пріоритетної території. Цей інтерфейс показано на рисунку 1.11.

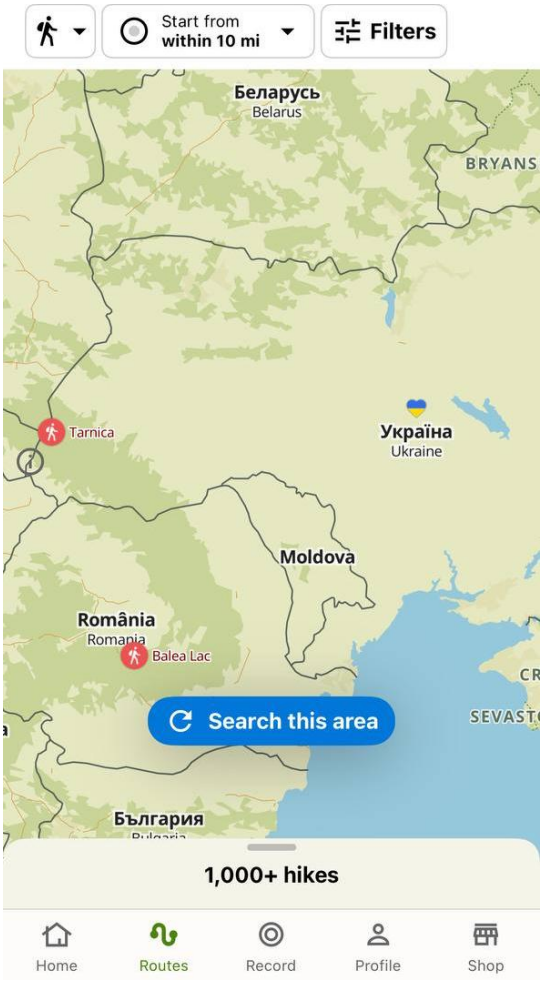

Рисунок 1.11 – Вибір регіону для офлайн-карт. Україна представлена з повноцінним прапором (рисунок виконано самостійно)

Сервіс також має вбудовану соціальну платформу, де користувачі можуть публікувати власні маршрути, залишати відгуки, додавати фотографії й отримувати реакції. Komoot дозволяє підписуватись на профілі інших користувачів та створювати спільноти за географічним принципом або за типом активності. Приклади таких спільнот подано на рисунку 1.12.

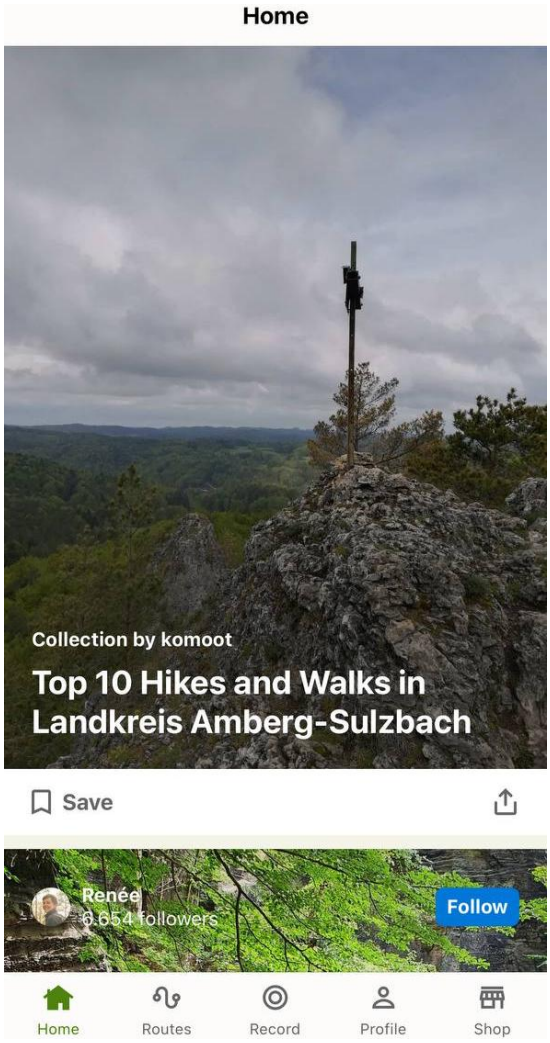

Рисунок 1.12 – Інтерфейс соціальної активності в Komoot. Перелік маршрутів від користувачів та локальні рекомендації (рисунок виконано самостійно)

Серед позитивних сторін Komoot слід відзначити точну побудову маршрутів, підтримку офлайн-навігації, повноцінне використання картографії для України, широкий вибір типів активностей, синхронізацію з годинниками та трекерами, а також наявність локальних рекомендацій. Застосунок добре масштабований і не перевантажений зайвим інтерфейсом.

Основними недоліками є відсутність системи обміну повідомленнями між учасниками подорожі, неможливість збору й синхронізації GPS-даних у режимі реального часу з декількох пристроїв, а також відсутність моніторингу фізіологічного стану користувачів. У застосунку не реалізовано механізмів виклику допомоги або фіксації критичних подій.

Попри потужну функціональність, проаналізовані сервіси мають суттєві обмеження:

а) вони не реалізують моніторинг фізичного стану учасників, зокрема даних про серцебиття;

б) відсутні механізми екстреного реагування;

в) немає централізованого збору GPS-даних у реальному часі від кожного учасника.

Аналіз предметної галузі показав, що наявні програмні рішення не охоплюють повного спектру потреб, пов’язаних з організацією безпечних групових подорожей у реальному часі. Це створює передумови для розробки нової програмної системи, яка буде орієнтована на моніторинг фізичного стану користувачів, збирання GPS-даних, підтримку спілкування та екстреного реагування. Запропоноване в межах цієї кваліфікаційної роботи програмне забезпечення вирішує вказані проблеми та доповнює наявні рішення для підвищення безпеки та ефективності групових мандрівок.

1.2 Цільова аудиторія програмної системи

Запропонована програмна система орієнтована на користувачів, які беруть участь у групових подорожах, мають потребу в координації дій у реальному часі та прагнуть забезпечити безпеку кожного учасника. До основних груп цільової аудиторії належать:

а) туристичні гіди, волонтери або інструктори, які супроводжують групи із підвищеним рівнем ризику (діти, літні люди, особи з хронічними захворюваннями);

б) учасники туристичних груп, які подорожують у складній місцевості або в умовах обмеженого зв’язку, та потребує можливості швидко передати своє місцезнаходження або надіслати сигнал тривоги;

в) організатори активних мандрівок, походів та спортивних заходів, які відповідають за склад групи, контроль маршруту та оперативне реагування у випадку надзвичайної ситуації;

г) тренери та супроводжуючі особи, що проводять виїзні заняття на відкритому повітрі, у гірській місцевості чи за містом, де важливо моніторити фізичний стан учасників.

Кожен тип користувача матиме доступ до окремого набору функціональностей залежно від ролі в подорожі (організатор або учасник), що дозволить адаптувати систему до різних сценаріїв використання.

1.3 Постановка задачі

Метою цієї кваліфікаційної роботи є розробка серверної частини програмної системи, що забезпечує організацію групових подорожей, трекінг місцезнаходження та фізичного стану учасників у реальному часі, а також обробку й збереження відповідних даних з дотриманням вимог до продуктивності та інформаційної безпеки. Клієнтська частина системи, що відповідає за взаємодію з користувачем, розробляється окремою командою в межах спільного проєкту.

Для реалізації цієї мети система має реалізовувати такі основні функції:

– реєстрація та автентифікація користувачів із розмежуванням прав доступу;створення, перегляд та управління подорожами;

– приєднання користувачів до подорожі;

– прийом даних про координати та серцебиття в реальному часі;

– реагування на екстрені ситуації (виклик допомоги);

– формування статистики за результатами подорожей;

– забезпечення конфіденційності та цілісності переданих і збережених даних.

Серверна частина програмної системи включатиме декілька взаємопов’язаних компонентів. Базу даних необхідно розробити з урахуванням високої частоти надходження телеметричних даних, вона має забезпечувати зберігання інформації про користувачів, групи подорожей, повідомлення, трекінгові записи та виклики допомоги. Для оптимізації продуктивності слід реалізувати партиціювання трекінгових даних за датою. Веб-API має надавати інтерфейси для взаємодії з клієнтською частиною, обслуговуючи операції створення подорожей, обробки GPS-даних, викликів тривоги та формування статистики. Механізми безпеки мають відповідати сучасним вимогам: авторизація на основі JWT [5], шифрування координат і біометричних даних за алгоритмом AES, обов’язкове використання HTTPS і обмеження доступу до функціональності відповідно до ролей користувачів. Модуль моніторингу має приймати координати та дані серцебиття з інтервалом у 20 секунд і зберігати їх для подальшого аналізу. Для забезпечення контролю за діями користувачів необхідно реалізувати логування активності з фіксацією подій у відповідній таблиці.

Задля реалізації основного функціоналу необхідно вирішити такі задачі:

– спроєктувати структуру бази даних з урахуванням великої кількості телеметричних записів та логічних зв’язків між сутностями;

– побудувати механізм зберігання інформації про подорожі, користувачів, повідомлення, трекінг і сигнали тривоги з підтримкою масштабованості та продуктивності;

– реалізувати логіку взаємодії з клієнтським застосунком, включаючи реєстрацію, авторизацію, створення подорожей, обмін повідомленнями та трекінг у реальному часі;

– забезпечити обробку координат і біометричних даних з урахуванням затримки та періодичності передачі;

– реалізувати систему захисту даних, що включає розмежування доступу, шифрування та контроль автентичності користувачів;

– передбачити логування дій користувачів у межах системи для аудиту й аналітики;

– забезпечити наявність документації до API та можливість її перегляду безпосередньо під час роботи із системою;

– перевірити роботу API у типовому сценарії взаємодії з клієнтом, включаючи позитивні та негативні сценарії.

Для реалізації серверної частини використовуватиметься сучасний технологічний стек. Основна платформа розробки – ASP.NET Core 8 [6], яка забезпечує високу продуктивність і масштабованість. Як систему керування базами даних обрано PostgreSQL [7], з огляду на її стабільність, підтримку партиціювання та сумісність із .NET-середовищем. Для взаємодії з базою даних застосовуватиметься ORM-фреймворк Entity Framework Core, який дозволяє зручно працювати з моделями даних і автоматизує більшість рутинних операцій. Розробка здійснюватиметься в середовищі JetBrains Rider. API тестуватиметься за допомогою інструменту Postman, що забезпечує зручну перевірку ендпоінтів і моделювання сценаріїв запитів. Для автоматизованої генерації та перегляду документації до API використовуватиметься Swagger UI, який дозволить швидко переглядати структуру запитів, доступні маршрути та відповіді сервера.

2 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Концепція

Серверна частина системи підтримує логіку функціонування, при якій учасники подорожі взаємодіють із клієнтом, який у свою чергу надсилає запити до API. Після реєстрації та авторизації користувач обирає подорож або створює нову. Учасники подають запит на приєднання, після чого організатор затверджує або відхиляє їхню участь. Після початку подорожі кожен учасник автоматично надсилає дані про своє місцезнаходження та серцебиття з фіксованою періодичністю.

На стороні сервера ці дані зберігаються, шифруються та обробляються. У разі натискання кнопки тривоги система створює екстрене повідомлення з координатами, яке фіксується в базі й передається організатору для оперативного реагування. Після завершення маршруту користувач має можливість переглянути аналітику: середню швидкість, пульс, пройдену відстань, карту переміщення. Усі дії користувачів логуються.

Адміністратор системи має доступ до перегляду інформації про користувачів, маршрути, журнали активності, сигнали тривоги, а також до загального технічного стану системи через інтерфейс керування.

2.2 Функціональні вимоги

Система повинна забезпечувати реєстрацію користувачів із перевіркою правильності введених даних та контролем на унікальність облікового запису. Авторизація реалізується за допомогою JWT-токенів, що забезпечує безпечне управління сесіями користувачів.

Необхідно підтримувати розмежування прав доступу залежно від ролі. Передбачено дві ролі: користувач та адміністратор. Кожна роль має власний набір дозволених дій і доступних інтерфейсів.

Користувач має змогу створювати подорожі, вказуючи ключові параметри: назву, опис, тривалість, маршрут, він автоматично стає організатором. Інші користувачі можуть подавати заявки на участь, які організатор може підтвердити або відхилити. Таким чином формується склад групи. У запланований час організатор має можливість перевести подорож до активного стану.

Під час подорожі учасник у межах подорожі періодично надсилає на сервер дані про своє місцезнаходження (GPS) і частоту серцебиття із інтервалом у 20 секунд. Ці дані повинні зберігатися у форматі, придатному для подальшої обробки та аналізу.

Після завершення подорожі система має надавати доступ до статистики: карта маршруту, середня швидкість, середній пульс, загальна дистанція, витрачений час.

Слід реалізувати механізм виклику допомоги: користувач повинен мати змогу надіслати SOS-сигнал, що міститиме координати поточного розташування. Така подія має бути зафіксована на сервері й доступна організатору.

Передбачається реалізація внутрішнього чату. Учасники групи повинні мати можливість обмінюватися текстовими повідомленнями в межах однієї подорожі. Повідомлення мають передаватися в реальному часі.

Також має бути реалізований модуль логування, що фіксуватиме ключові події: у систему, створення подорожі, запити на участь, виклики допомоги тощо. Це дозволить відстежувати дії користувачів та виявляти збої.

Уся функціональність має бути реалізована через REST API з відповідною документацією, доступною для ознайомлення через Swagger UI.

2.3 Нефункціональні вимоги

Нефункціональні вимоги стосуються безпеки, продуктивності, масштабованості, надійності та супроводжуваності.

Усі запити повинні оброблятися лише через захищене з'єднання HTTPS. Для автентифікації використовується JWT-токен, що дозволяє ефективно перевіряти доступ до ресурсів. Дані про фізіологічний стан зберігаються в зашифрованому вигляді за допомогою алгоритму AES [8].

Система повинна бути здатна обробляти велику кількість однотипних запитів. З огляду на те, що кожен учасник надсилає телеметричні дані кожні 20 секунд, сервер має відповідати з мінімальною затримкою.

Архітектура повинна підтримувати горизонтальне масштабування для збільшення кількості підключень у реальному часі. У структурі бази даних передбачається партиціювання таблиці з трекінговими даними за датою, що знижує навантаження на операції запису та читання.

Система повинна бути стійкою до короткострокових збоїв зв'язку. Сервер має підтримувати обробки відкладених даних без втрати інформації.

Архітектура серверної частини має бути модульною. Весь код повинен бути тестований і готовий до розширення. Документація до API має залишатися актуальною впродовж усього життєвого циклу продукту.

2.4 Обмеження та припущення

Під час проєктування і реалізації системи мають бути враховані такі обмеження та припущення, які впливають на архітектуру, вибір технологій і функціональні можливості:

– сервер обробляє трекінгові дані з інтервалом у 20 секунд, що знижує навантаження, але обмежує точність моніторингу при високій частоті надсилання;

– шифрування координат і біометричних даних здійснюється за алгоритмом AES, що потребує додаткових обчислювальних ресурсів;

– партиціювання таблиці трекінгових даних виконується лише за датою, без сегментації за користувачами;

– партиціювання таблиці «Дані трекінгу» реалізовано за часовою ознакою (дата), без урахування ідентифікатора користувача.

– усі запити обробляються лише через HTTPS, що унеможливлює роботу в середовищах без SSL;

– чат між учасниками реалізовано лише у вигляді текстових повідомлень, без передачі мультимедійних даних.

Припущення:

а) клієнтський застосунок надсилає запити у форматі, що повністю відповідає описаній структурі REST API;

б) збір згоди на обробку персональних даних виконується виключно клієнтом до початку взаємодії з сервером;

в) трекінгові пристрої стабільно передають дані у форматі, сумісному з API, та не вимагають додаткового серверного перетворення або адаптації;

г) повторна передача втрачених даних (у випадку збою зв’язку) реалізується клієнтом. сервер приймає лише надіслані запити, без повторного опитування пристроїв;

д) авторизовані запити вважаються достовірними. перевірка фізичної справності пристрою або точності GPS-координат не виконується;

е) серверна частина відповідає лише за прийом, валідацію, шифрування, зберігання та видачу даних. інтерфейси, оповіщення й візуалізація реалізуються окремо на стороні клієнта.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз та UML-моделювання предметної області

Проєктування архітектури програмного забезпечення здійснювалося на основі об’єктно-орієнтованої парадигми з використанням уніфікованої мови моделювання UML (Unified Modeling Language). UML дозволяє створити формалізоване графічне подання як статичних, так і динамічних аспектів системи. Це забезпечує можливість чіткого структурування логіки проєкту, візуалізації взаємодії компонентів, а також підвищує зрозумілість моделі для сторонніх розробників.

У процесі проєктування було розроблено кілька діаграм, що відображають ключові функціональні сценарії системи, взаємозв’язки між класами, послідовність обміну повідомленнями, а також логіку дій користувача в рамках одного з базових сценаріїв.

Діаграма прецедентів відображає функціональні можливості системи з точки зору користувачів. Вона дозволяє встановити, які ролі взаємодіють із системою, до яких дій мають доступ і за яких умов ці дії виконуються. Такий підхід допомагає структурувати функціональність системи відповідно до вимог безпеки, розмежування прав і логіки поведінки інтерфейсу.

Система підтримує два типи користувачів: звичайного користувача та адміністратора. Усі функції системи розподілено між ними відповідно до призначення та прав доступу.

Доступ користувача до функціоналу забезпечується після проходження автентифікації та авторизації. Реєстрація передбачає валідацію введеної адреси електронної пошти. Після входу до системи користувач отримує доступ до трьох основних блоків: керування профілем, керування подорожами та участь у подорожах.

Функціональність керування профілем включає оновлення персональних даних, зміну пароля та можливість видалити обліковий запис. Блок керування подорожами містить функції створення, редагування, видалення подорожей, перегляду доступних груп, подання заявок на участь, розгляду запитів інших учасників (якщо користувач є організатором), а також старт і завершення подорожі.

Блок участі в подорожі активується для користувача, який є учасником однієї з груп. Він містить можливість надсилання повідомлень, передачі даних про місцезнаходження та серцебиття, надсилання SOS-сигналу, виходу з групи, перегляду статистики подорожі та читання повідомлень. Надсилання телеметричних даних і SOS доступне лише під час активної подорожі.

Адміністратор взаємодіє із системою через окремий блок функцій, пов’язаних із моніторингом і керуванням. Цей блок включає перегляд журналів дій користувачів, перегляд системних подій, блокування облікових записів та перевірку поточного стану системи. Адміністратор не має доступу до функціоналу користувача і не бере участі в подорожах.

На рисунку 3.1 зображено повну діаграму прецедентів, яка деталізує функціональну модель серверної частини системи відповідно до розмежування ролей та умов доступу.

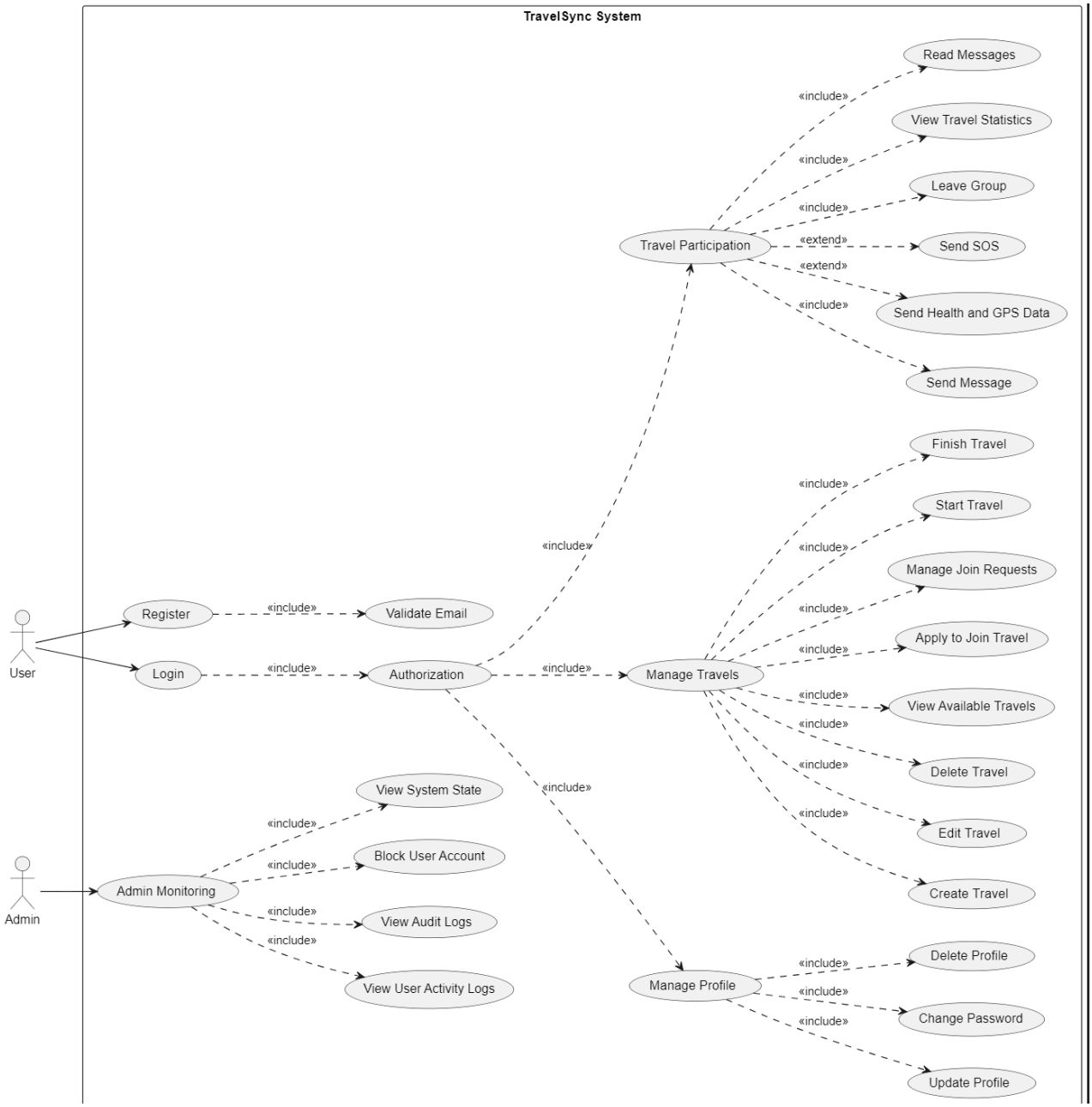

Рисунок 3.1 – Діаграма прецедентів для системи організації групових подорожей (рисунок виконано самостійно)

Діаграма послідовності демонструє взаємодію між компонентами серверної частини системи під час виконання одного з ключових сценаріїв. Зокрема, на рисунку 3.2 зображено процес, у якому користувач послідовно проходить автентифікацію, створює нову подорож, підтверджує заявки інших користувачів, запускає подорож і бере в ній участь.

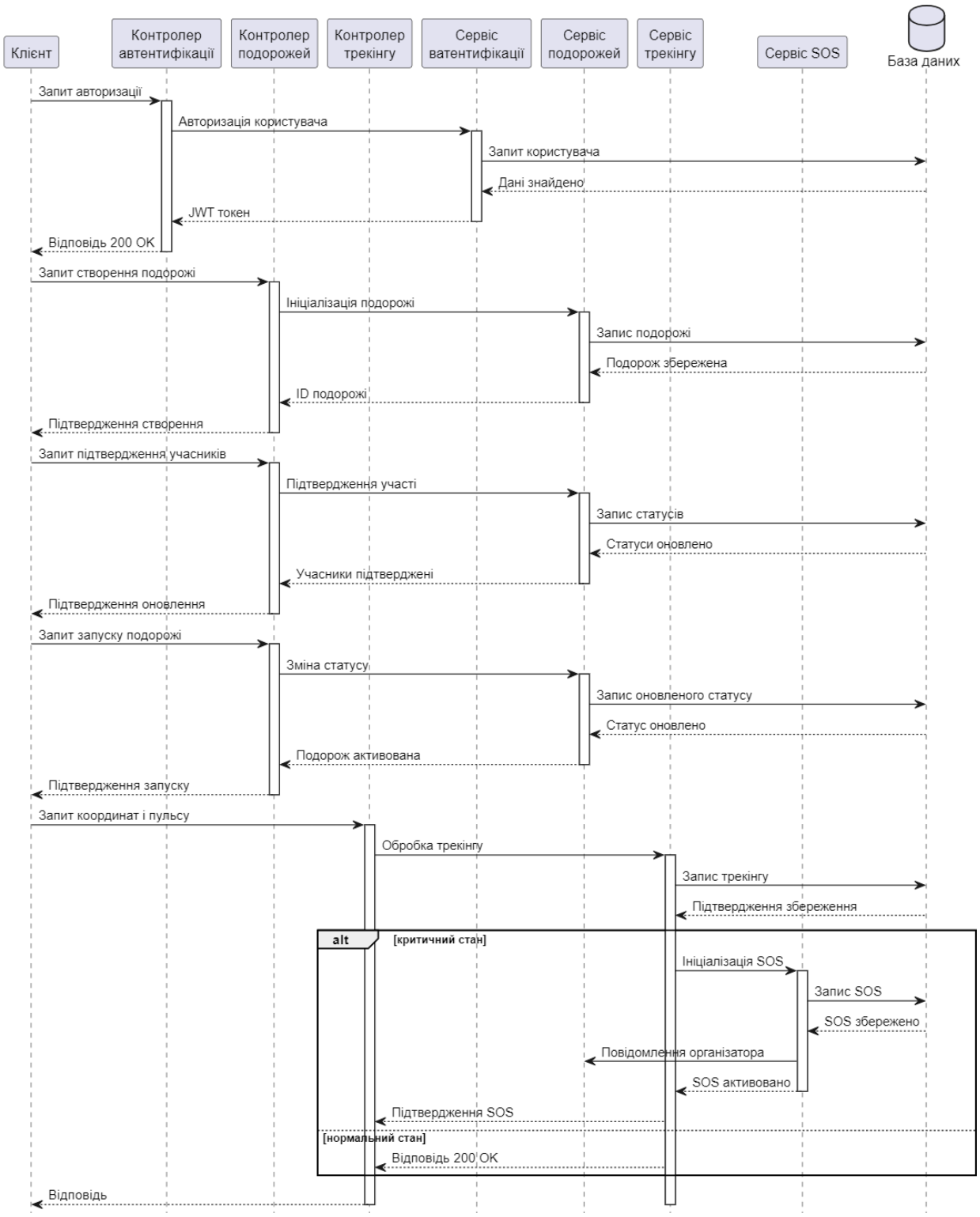

Рисунок 3.2 – Діаграма послідовності сценарію створення та запуску подорожі (рисунок виконано самостійно)

У межах активної подорожі клієнтська частина періодично надсилає трекінгові дані (GPS-координати та частоту серцебиття). У разі виявлення критичних значень пульсу або координат активується механізм надсилання SOS-сигналу.

Учасниками діаграми є клієнтська частина, контролери та сервіси, що відповідають за обробку запитів, логіку виконання сценаріїв і взаємодію з базою даних. Кожна дія ініціюється запитом до відповідного контролера, після чого виконується виклик сервісного рівня, який у разі потреби звертається до бази даних. Усі відповіді повертаються тим самим шляхом, через який було ініційовано запит. У частині обробки трекінгових даних передбачено умовне розгалуження: якщо отримані параметри не перевищують встановлені пороги, сервіс повертає стандартну відповідь; у разі виявлення критичних значень запускається окремий сценарій обробки тривоги із збереженням події SOS у базі та сповіщенням організатора подорожі.

Діаграма демонструє чіткий розподіл обов’язків між компонентами, дотримання принципу односторонньої відповідальності й відповідність архітектурі REST API. Взаємодія структурована у вигляді синхронних повідомлень, активність об’єктів відображено через вертикальні прямокутники, що відповідають часу виконання операцій.

Діаграма активності наочно демонструє поведінку системи в контексті циклічного трекінгу під час подорожі. Вона починається з авторизації користувача та перевірки його участі в активній подорожі. Далі запускається цикл, який з періодичністю 20 секунд виконує перевірку наявності даних від датчика, надсилає їх на сервер і фіксує результат. У разі надходження сигналу SOS виконання гілкується та створюється запис у таблиці екстрених викликів, який доступний для організатора.

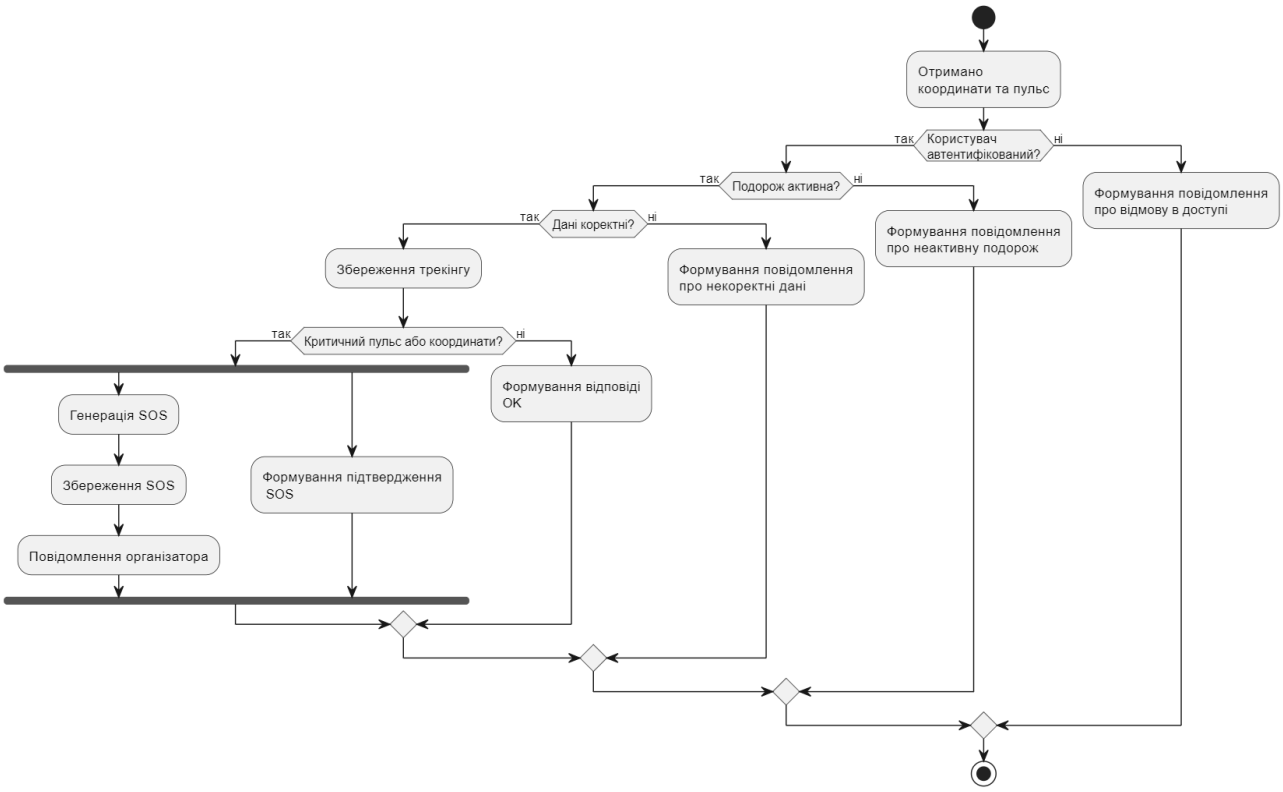

Рисунок 3.3 – Діаграма активності обробки трекінгових даних і SOS-сценарію (рисунок виконано самостійно)

На рисунку 3.3 зображено послідовність дій серверної частини системи під час обробки телеметричних даних користувача. Враховано перевірку автентифікації, стану подорожі, коректності даних, а також альтернативну гілку для критичних значень із активацією SOS-сценарію.

3.2 Проєктування архітектури програмного забезпечення

Архітектура програмної системи побудована за моделлю клієнт–сервер, що дозволяє розподілити відповідальність між клієнтським застосунком, сервером і базою даних. У межах цієї роботи розглядається серверна частина системи, яка відповідає за обробку запитів, облік даних і взаємодію з базою. Такий підхід забезпечує централізовану обробку інформації, підвищену безпеку, модульність і масштабованість.

Серверна частина побудована на основі трирівневої архітектури : рівень представлення, рівень бізнес-логіки та рівень доступу до даних [9]. Цей підхід реалізується згідно з принципами об’єктно-орієнтованого програмування [10], що лягли в основу проєктування системи і дає змогу ізолювати логіку застосунку від технічної реалізації зберігання даних і взаємодії з API.

Рівень представлення (Presentation Layer) реалізовано за допомогою контролерів ASP.NET Core. Контролери обробляють вхідні HTTP-запити, виконують валідацію даних і викликають відповідні сервіси. Для документування та тестування API використовується Swagger UI та Postman.

Рівень бізнес-логіки (Business Logic Layer) складається з сервісів, що реалізують основні сценарії: реєстрація та авторизація користувачів, створення подорожей, обробка телеметричних даних, надсилання SOS-сигналів, генерація статистики. Усі сервіси реалізуються через інтерфейси, що забезпечує слабке зв’язування та спрощує тестування.

Рівень доступу до даних (Data Access Layer) відповідає за зчитування й запис інформації до бази даних PostgreSQL. Для доступу до таблиць застосовується ORM-фреймворк Entity Framework Core. Застосовано шаблон репозиторію, що дозволяє централізовано реалізувати CRUD-операції для кожної сутності.

Для керування залежностями використовується механізм впровадження залежностей (Dependency Injection), вбудований у платформу ASP.NET Core. Це дає змогу централізовано конфігурувати компоненти та знижує рівень зв’язності між модулями.

Основні доменні моделі, інтерфейси та об’єкти передавання даних зберігаються в окремому Core-проєкті. Це дозволяє підтримувати чітку структуру проєкту, уникати циклічних залежностей і повторного дублювання структур даних.

Для перевірки вхідних запитів застосовуються middleware-компоненти валідації з окремими класами-валідаторами, що дозволяє зменшити

навантаження на контролери. Аутентифікація реалізується за допомогою JWT-токенів, а контроль доступу – згідно з роллю користувача.

Обробка чутливих даних, а саме координат та пульсу, здійснюється із використанням симетричного шифрування AES. Шифрування реалізовано на окремому сервісному рівні до моменту запису в базу.

Для підвищення продуктивності впроваджено партиціювання таблиці з трекінговими даними за датою. Це спрощує агрегацію, архівацію та доступ до великих обсягів телеметрії.

Зовнішні залежності керуються через менеджер пакетів NuGet. Контроль версій здійснюється через систему Git, яка забезпечує можливість командної розробки, відстеження змін та контроль за гілками.

Запропонована архітектура відповідає принципам модульності та підтримуваності. Вона забезпечує можливість поступового розвитку системи без порушення її цілісності.

Взаємозв’язки між компонентами представлено на діаграмі розгортання (рисунок 3.4), яка демонструє фізичну структуру системи: запити від клієнтського пристрою, сервер застосунку, сервер бази даних і окремі модулі авторизації, шифрування та логування.

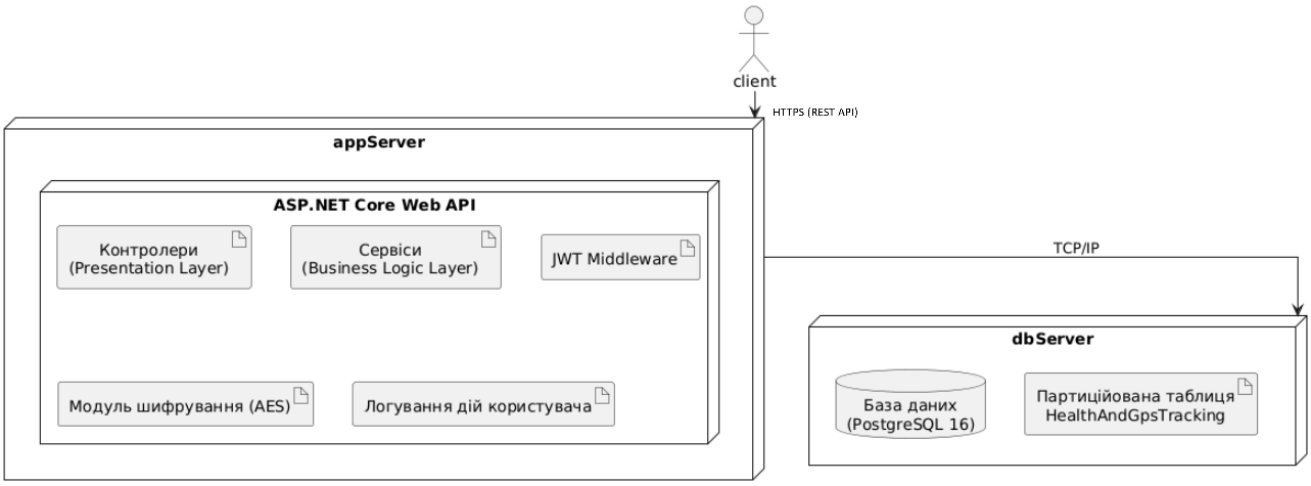

Рисунок 3.4 – Діаграма розгортання (рисунок виконано самостійно)

3.3 Проєктування бази даних

Проєктування бази даних програмної системи охоплює кілька взаємопов’язаних етапів: концептуальне, логічне та фізичне моделювання. На концептуальному рівні формується уявлення про об’єкти предметної області, їх основні характеристики та взаємозв’язки. Логічне моделювання дозволяє описати структуру бази даних у вигляді сутностей, атрибутів і зв’язків між ними, незалежно від обраної системи керування базами даних. Фізичне моделювання реалізує логічну модель у вигляді таблиць, типів даних, ключів та індексів у конкретному середовищі.

На етапі концептуального моделювання було сформовано загальну схему взаємодії ключових об’єктів системи. До основних понять, які мають бути відображені в базі даних, належать: користувач, групова подорож, повідомлення, трекінгові дані, сигнал тривоги, журнал дій, а також точки маршруту, які утворюють просторову структуру подорожі. На рисунку 3.5 наведено концептуальну схему, яка демонструє взаємозв’язки між об’єктами предметної області. Ця схема є основою для логічного проєктування.

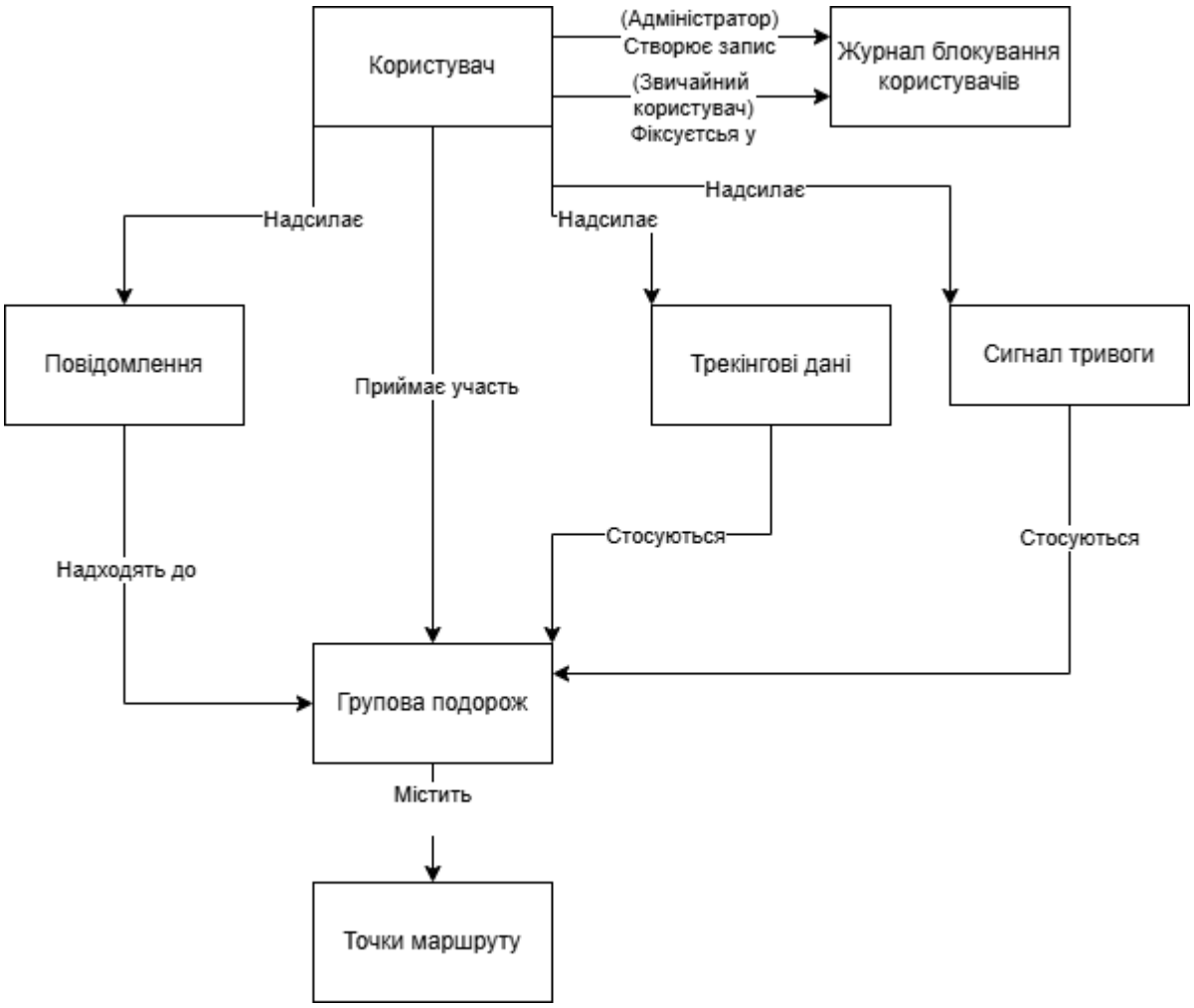

Рисунок 3.5 – Схема об'єктів предметної області системи (рисунок виконано самостійно)

У результаті аналізу функціональності системи було виділено такі сутності: «Користувач», «Групова подорож», «Участь», «Повідомлення», «Дані трекінгу», «Сигнал тривоги», «Журнал блокування користувачів» та «Точка маршруту подорожі». Вони формують ядро доменної моделі та забезпечують реалізацію основних сценаріїв взаємодії з даними.

Зв’язки між цими сутностями мають такий характер:

– один користувач може мати кілька записів у журналі блокувань як об'єкт блокування – зв’язок один до багатьох;

– один користувач може мати кілька записів у журналі блокувань як адміністратор – зв’язок один до багатьох;

– один користувач може створити багато подорожей – зв’язок один до багатьох;

– один користувач може брати участь у багатьох подорожах, і навпаки – багато до багатьох.

Для реалізації цього зв’язку, а також для збереження пов’язаних із участю даних (трекінгові дані, повідомлення, сигнали тривоги), у моделі введено окрему проміжну сутність «Участь». Вона дозволяє зберігати індивідуальні стани взаємодії користувача з кожною подорожжю.

Таким чином, встановлені наступні зв’язки:

– одна участь пов’язана з багатьма трекінговими записами;

– одна участь пов’язана з багатьма повідомленнями;

– одна участь може містити один або кілька сигналів тривоги – один до багатьох;

– кожна подорож містить набір точок маршруту, впорядкованих за послідовністю проходження – один до багатьох.

На основі цих взаємозв’язків сформовано логічну модель бази даних, яка включає наступні сутності:

Сутність «Користувач» має такі атрибути:

– унікальний ідентифікатор користувача;

– повне ім’я;

– електронна пошта для авторизації та сповіщень;

– хеш пароля;

– роль у системі (учасник або адміністратор);

– дата реєстрації;

– прапорець активності облікового запису.

Сутність «Групова подорож» має такі атрибути:

– унікальний ідентифікатор групової подорожі;

– назва подорожі;

– опис маршруту;

– дата початку;

– дата завершення;

– регіон або геолокація;

– максимальна кількість учасників.

Сутність «Участь» має такі атрибути:

– унікальний ідентифікатор участі;

– ідентифікатор користувача;

– чи є користувач організатором;

– ідентифікатор групової подорожі;

– статус заявки (очікує, підтверджено, відхилено);

– дата приєднання до групи.

Сутність «Дані трекінгу» має такі атрибути:

– унікальний ідентифікатор трекінгового запису;

– ідентифікатор участі;

– широта;

– довгота;

– частота серцебиття;

– час надходження даних;

– позначка аномального значення.

Сутність «Повідомлення» має такі атрибути:

– унікальний ідентифікатор повідомлення;

– ідентифікатор участі;

– вміст повідомлення;

– дата й час надсилання.

Сутність «Сигнал тривоги» має такі атрибути:

– унікальний ідентифікатор сигналу тривоги;

– ідентифікатор участі;

– координати сигналу (широта та довгота);

– статус обробки;

– дата створення.

Сутність «Журнал блокування користувачів» має такі атрибути:

– унікальний ідентифікатор запису;

– ідентифікатор користувача, якого було заблоковано;

– ідентифікатор адміністратора, який здійснив блокування;

– причина блокування;

– дата та час блокування;

– дата розблокування (необов’язкова);

– статус активності блокування (діє/скасовано).

Сутність «Точка маршруту подорожі» має такі атрибути:

– унікальний ідентифікатор точки;

– ідентифікатор групової подорожі, до якої належить ця точка;

– широта;

– довгота;

– назва точки («старт», «оглядова вежа»);

– опис;

– порядковий номер у маршруті.

Усі атрибути відповідають своїй логічній ролі та мають атомарну структуру, що виключає зберігання складних або вкладених типів даних.

Модель відповідає вимогам третьої нормальної форми (3НФ), що підтверджується такими умовами:

– усі атрибути є атомарними (1НФ);

– жоден неключовий атрибут не залежить від частини складеного ключа (2НФ);

– немає транзитивних залежностей між неключовими атрибутами (3НФ).

Такий підхід дозволяє уникнути дублювання, забезпечити логічну цілісність даних і спростити супровід системи [11].

Структура наведених сутностей і їх взаємозв’язків відображена на схемі бази даних, зображеній на рисунку 3.6.

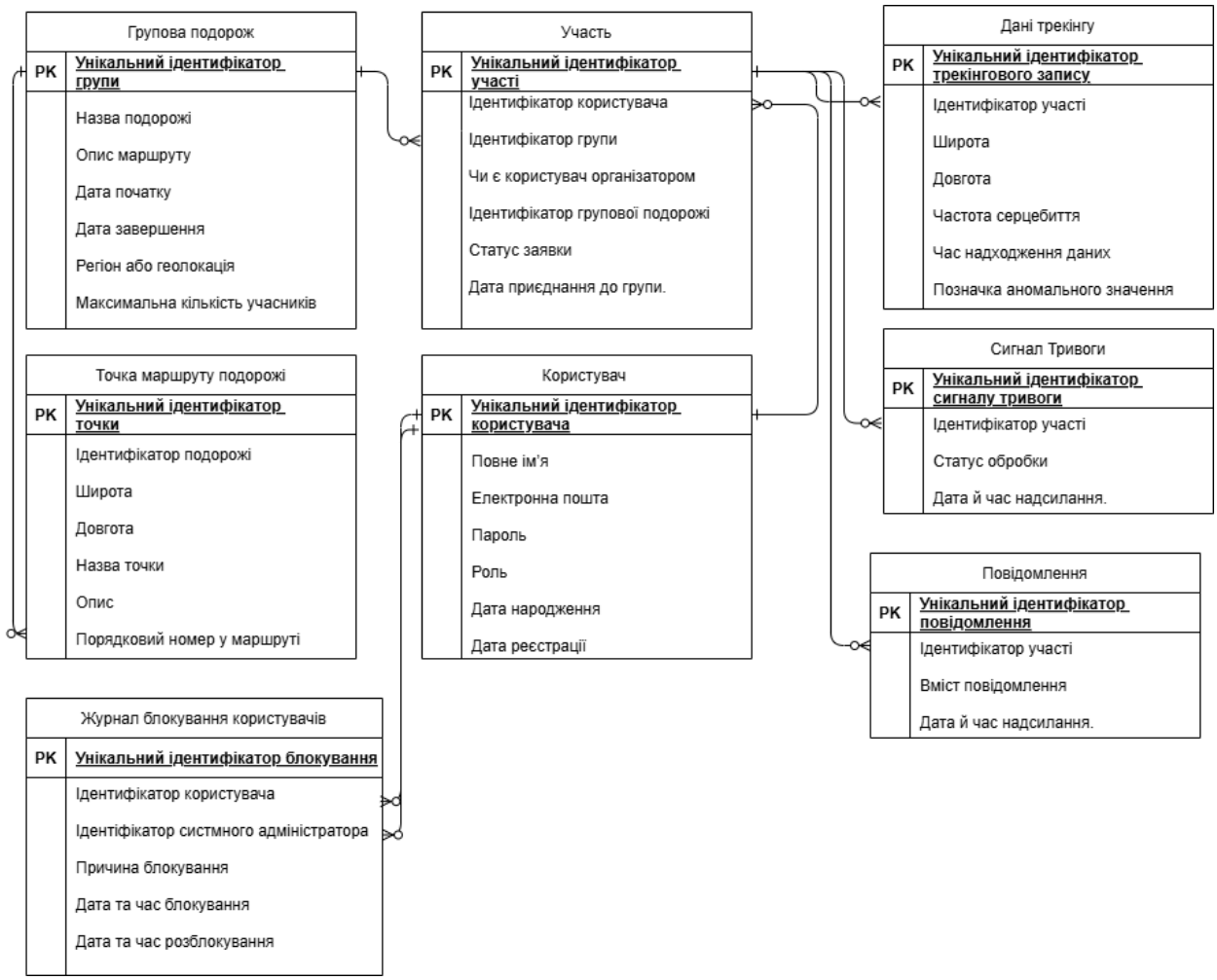

Рисунок 3.6 – Логічна модель бази даних (рисунок виконано самостійно)

На основі цієї логічної моделі реалізується фізична структура таблиць у базі даних PostgreSQL.

3.4 Алгоритми обробки даних у серверній частині

У межах реалізації серверної частини системи виникла потреба в обробці координат, біометричних показників і формуванні персоналізованих рекомендацій. Для кожної з цих задач були підібрані оптимальні алгоритми, що враховують вимоги до точності, ресурсоємності, швидкодії в режимі

реального часу та особливостей зберігання. Нижче детально описано три ключові алгоритми, які використовуються в різних модулях системи.

3.4.1 Алгоритм Гаверсина для обчислення відстані між GPS-точками

Система отримує геолокаційні координати кожного користувача кожні 20 секунд. Ці дані надходять через клієнтський застосунок у сервіс трекінгу, обробляються на рівні бізнес-логіки й зберігаються в таблицю «Дані трекінгу». Для визначення пройденої відстані, виявлення виходу за межі маршруту та побудови треку потрібне точне обчислення відстані між двома точками, заданими в координатах широти й довготи.

Було розглянуто кілька методів. Евклідова відстань була відхилена через обмеження на площинну геометрію, бо вона не враховує кривизну Землі. Формула сферичного косинуса забезпечує кращу точність, але має числові нестабільності на малих відстанях. Алгоритм Вінсенті є точним, але надто складний для використання у реальному часі при високому навантаженні.

Зважаючи на баланс між точністю, стабільністю й продуктивністю, у системі використовується формула Гаверсина, яка забезпечує мінімальну відстань на сфері:

$$a = sin^2(\Delta\varphi/2) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot sin^2(\Delta\lambda/2) \quad (3.4.1)$$

$$c = 2 \cdot arctan2(\sqrt{a}, \sqrt{(1 - a)}) \quad (3.4.2)$$

$$d = R \cdot c \quad (3.4.3)$$

де φ – широта, λ – довгота, R – радіус Землі (приблизно 6371 км), d – шукане значення відстані.

Теоретична похибка формули Гаверсина становить менше 0,5% для відстаней до 100 км, що є допустимим для аматорських задач моніторингу. Для

коротких ділянок (до 10 м) відхилення не перевищує 1 метрам. У порівнянні з формулою сферичного косинуса або евклідовою метрикою, формула Гаверсина демонструє стабільніші результати на коротких відстанях, що критично для задач із високою частотою збору координат. Хоча алгоритм Вінсенті забезпечує вищу точність, його обчислювальна складність робить його менш придатним для використання в режимі реального часу [13].

Застосування аналогічних підходів для синхронізації та аналізу просторових даних описано в працях з автоматизації моніторингу [14].

Реалізація відбувається на рівні бізнес-логіки в сервісі трекінгу: під час надходження нової точки координати порівнюються з попередньою для того самого користувача й подорожі, після чого обчислена відстань додається до поля загальної довжини маршруту. Це значення зберігається в базі та використовується для формування статистики, побудови маршруту та оцінки швидкості пересування.

3.4.2 Алгоритм рухомого середнього для згладжування біометричних даних

Біометричні дані, зокрема частота серцебиття, надходять із клієнтського застосунку на сервер із тією ж частотою, що й координати. У зв’язку з технічними особливостями пристроїв зчитування (контакт, рух, перешкоди) у даних можливі короткочасні стрибки, які не є клінічно значущими. Для фільтрації таких артефактів згладжування реалізовано на сервері перед збереженням у таблицю «Дані трекінгу». Було розглянуто кілька варіантів.

Фільтр Калмана відкинуто через складність реалізації та необхідність побудови математичної моделі динаміки пульсу. У разі використання без попереднього навчання він може давати нестабільні або некоректні оцінки.

Експоненціальне згладжування також розглядалося, але потребує ручного підбору коефіцієнта згладжування (альфа), що залежить від

фізіологічних особливостей користувача. У системі, де користувачі є анонімними до моменту збору, параметр не можна адаптувати індивідуально.

Найбільш стабільним і простим виявився метод ковзного середнього [15] з фіксованим вікном останніх п’яти значень.

Його суть полягає в тому, що кожне нове значення усереднюється з кількома попередніми. Завдяки цьому окремі випадкові відхилення втрачають вагу, а загальний результат краще відображає реальний стан користувача. Формула виглядає так:

$$MA_{\mathrm{t}} = \frac{1}{n} \cdot \sum_{i=t-n+1}^{i=t} x_{\mathrm{i}} \qquad (3.4.4)$$

де x_i – значення пульсу в момент часу i, MA_t – згладжене значення на час t, n – ширина ковзного вікна.

У системі використовується фіксоване вікно з п’яти останніх вимірів.

Реалізація здійснюється в окремому сервісі обробки телеметрії. Для кожного користувача зберігається буфер останніх 5 значень у пам’яті. Після обчислення згладженого значення воно зберігається в базу як основне й використовується для побудови графіків, статистики та аномалій. У разі підозри на аномальний пульс система також звертається до незгладженого значення, яке паралельно записується для медичного журналу.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Архітектурні рішення

Серверну частину програмної системи реалізовано за принципами трирівневої архітектури, що включає шар прикладного інтерфейсу (API), шар бізнес-логіки (BL) та шар доступу до даних (Data Access). Крім того, у структурі проєкту виділено окремий доменний рівень (Domain), який містить типи, перерахування, об’єкти передавання даних, профілі для AutoMapper і контракти (інтерфейси), а також інфраструктурний рівень (Infrastructure), що відповідає за обробку компонентних залежностей: логування, шифрування, кешування та надсилання електронної пошти.

Між шарами передаються об’єкти, призначені для транспортування даних, для яких реалізовано окремі засоби валідації. Перевірка вхідних запитів здійснюється через бібліотеку FluentValidation. Кожне правило супроводжується інформативним повідомленням про помилку, яке повертається користувачу у разі некоректного введення.

Обробка винятків реалізована централізовано за допомогою проміжного програмного компонента (middleware) – обробника винятків (ExceptionHandlingMiddleware). Цей компонент перехоплює всі винятки, класифікує їх за типами та формує уніфіковану відповідь у форматі JSON. Такий підхід дозволяє уникнути дублювання логіки обробки помилок у кожному контролері та забезпечує узгоджений формат повідомлень для клієнтської частини.

У бізнес-логіці використовується узагальнений тип результату дії (Result<T>), який дозволяє описати як успішне виконання з даними, так і неуспішне, з повідомленням про помилку й відповідним кодом. Завдяки цьому обробка помилок не відбувається через кидання винятків, а передбачає контрольоване повернення уніфікованої відповіді з усією необхідною інформацією.

Відповідність між доменними моделями та об’єктами передавання даних забезпечується за допомогою AutoMapper, з налаштуванням спеціалізованих профілів. Профіль користувача (UserProfile) виконує перетворення об’єкта моделі користувача у відповідний об’єкт для API-відповіді з урахуванням вкладених колекцій.

Усі сервіси створювалися з урахуванням принципів SOLID і DRY. Контролери залишаються мінімальними за розміром і виконують лише функцію обробки HTTP-запитів, передаючи логіку в сервіси.

4.2 Безпека та шифрування

У межах системи реалізовано механізм автентифікації. Авторизація користувача можлива за електронною адресою або ім’ям користувача. Перевірка облікових даних виконується у відповідному сервісі автентифікації (AuthService), де введений пароль порівнюється з хешованим значенням у базі даних. Для хешування паролів використовується алгоритм Bcrypt, реалізований у сервісі хешування паролів (BcryptHashingService). У разі успішної автентифікації генерується маркер доступу — JSON Web Token. Його створює сервіс роботи з токенами (JwtService), задаючи термін дії (12 годин) та включаючи у payload такі claims: ідентифікатор користувача, email, ім’я користувача, призначення токена (purpose) та ознаку адміністратора системи (IsSystemAdmin).

Після реєстрації нового користувача система надсилає лист для підтвердження електронної пошти. У листі міститься спеціальне посилання, що містить токен підтвердження. Відправлення здійснюється за допомогою сервісу надсилання пошти (EmailService), який використовує бібліотеку MailKit та протокол SMTP із TLS-з’єднанням. Тіло листа форматується з використанням HTML.

Нижче наведено фрагмент коду встановлення з’єднання та надсилання листа:

```
      await client.ConnectAsync(_settings.SmtpServer,
_settings.SmtpPort, SecureSocketOptions.StartTls);
      await client.AuthenticateAsync(_settings.SmtpUser,
_settings.SmtpPass);
      await client.SendAsync(emailMessage);
```

Активаційний лист надсилається від імені системи на адресу, вказану під час реєстрації (див. рис. 4.1). Логування процесу надсилання виконується через ILoggingService.

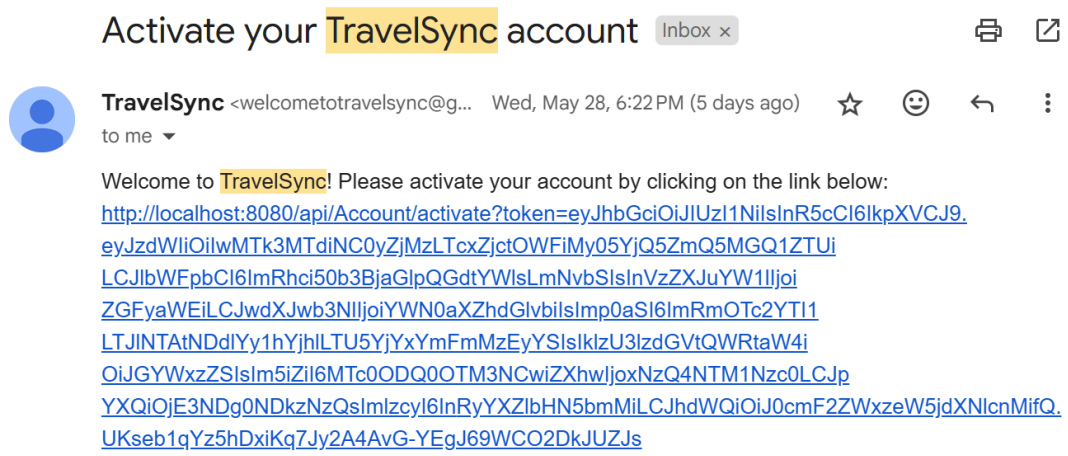

Рисунок 4.1 – Лист для підтвердження електронної пошти (рисунок виконаний самостійно)

Модуль керування подорожами забезпечує повний цикл управління подорожами. Реалізована логіка створення нових маршрутів з використанням фабрики, яка дозволяє стандартизовано формувати складні об'єкти подорожей. Кожна подорож має один із статусів: запланована, активна, завершена або скасована. Статус визначає доступність функціоналу (активація доступна лише для подорожей зі статусом Planned). Модуль підтримує фільтрацію та сортування за назвою, статусом, датами та пагінацію для роботи зі значною

кількістю записів. Автоматично додається адміністратор, що створив подорож, який потім керує заявками інших учасників.

Участь у подорожі реалізовано через окрему сутність, яка зберігає зв'язок між користувачем і подорожжю. Користувач може подати заявку на участь у подорожі. Участь має один із трьох статусів: Pending (очікує підтвердження), Accepted (підтверджено), Rejected (відхилено). Організатор або інший адміністратор групи має право приймати або відхиляти заявки.

Сутність участі містить атрибути фактичного приєднання до подорожі та роль адміністратора групи. Ці параметри використовуються для перевірки прав доступу в інших модулях (зокрема трекінгу, повідомленнях та SOS-викликах). Підходи до розподілу прав і оперативного контролю за станом користувачів детально описані у проєкті «Ментор» [16].

Перед виходом із подорожі перевіряється, чи користувач не є єдиним адміністратором активної подорожі. Якщо так – вихід заборонено. Також реалізовано перевірку на максимальну кількість учасників (MaxParticipants), після якої нові заявки автоматично відхиляються.

4.3 Обробка та зберігання трекінгових даних

Модуль трекінгу відповідає за обробку координат і частоти серцебиття, які надходять від смарт-годинника кожні 15–20 секунд. Кожен запис включає GPS-координати, частоту серцевих скорочень і мітку часу.

Після перевірки дані проходять шифрування. Для захисту конфіденційної інформації використовується алгоритм симетричного шифрування AES. Координати та пульс шифруються окремо, ключ і вектор ініціалізації зберігаються в системі конфігурації. Нижче наведена реалізація шифрування що відбувається у відповідному сервісі:

```
    public string Encrypt(string plainText, string? secretKey = null)
{
```

```
    var keyBytes = GetAesKey(secretKey);
    using var aes = Aes.Create();
    aes.Key = keyBytes;
    aes.GenerateIV();

    using var encryptor = aes.CreateEncryptor();
    var plainBytes = Encoding.UTF8.GetBytes(plainText);
    var cipherBytes = encryptor.TransformFinalBlock(plainBytes, 0,
plainBytes.Length);

    var result = new byte[IvSize + cipherBytes.Length];
    Buffer.BlockCopy(aes.IV, 0, result, 0, IvSize);
    Buffer.BlockCopy(cipherBytes, 0, result, IvSize,
cipherBytes.Length);

    return Convert.ToBase64String(result);
```

Для обчислення дистанції між точками використовується формула Гаверсина. Вона враховує сферичну форму Землі, що забезпечує високу точність навіть при обробці коротких відрізків маршруту. Обчислення виконується на рівні сервісу:

```
     public double CalculateDistanceKm(double lat1, double lon1,
double lat2, double lon2)
{
    const double R = 6371; // радіус Землі в км
    var dLat = DegreesToRadians(lat2 - lat1);
    var dLon = DegreesToRadians(lon2 - lon1);
    var a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
            Math.Cos(DegreesToRadians(lat1)) *
Math.Cos(DegreesToRadians(lat2)) *
            Math.Sin(dLon / 2) * Math.Sin(dLon / 2);
    var c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
    return R * c;
}
```

Значення частоти серцебиття згладжується методом ковзного середнього. Кожен користувач має буфер останніх п’яти значень, який оновлюється при кожному новому записі. Середнє обчислюється динамічно, що дозволяє зменшити вплив короткочасних стрибків або помилок пристрою. Сервіс згладження серцебиття наведено нижче.

```
     public class PulseSmoothingBufferService :  IPulseSmoothingBuffer
{
    private const int WindowSize = 5;
    private readonly Dictionary<Guid, Queue<int>> _buffers = new();
```

```
    public int AddAndSmooth(Guid userId, int newValue)
    {
        if (!_buffers.ContainsKey(userId))
            _buffers[userId] = new Queue<int>();

        var queue = _buffers[userId];

        if (queue.Count == WindowSize)
            queue.Dequeue();

        queue.Enqueue(newValue);

        return (int)Math.Round(queue.Average());
    }
}
```

Дані зберігаються навіть при відсутності зміни координат. Це дозволяє виявляти втрату зв’язку або збої в передачі, що критично важливо для безпеки користувачів.

Модуль екстрених викликів забезпечує дві незалежні логіки: надсилання сигналу «SOS» користувачем через інтерфейс та автоматичне формування виклику на основі критичних біометричних показників, зокрема частоти серцевих скорочень. Обидва сценарії реалізовано у відповідному сервісі обробки екстрених сигналів (EmergencyCallService), який взаємодіє з репозиторієм екстрених викликів (EmergencyCall) і виконує перевірку участі користувача.

У випадку ручного виклику користувач самостійно обирає тип інциденту (наприклад, травма, втрата орієнтації, загроза життю чи здоров’ю). Перед реєстрацією виклику система перевіряє, що участь у подорожі має статус «Підтверджено» (Accepted). Після цього дані зберігаються у таблиці екстрених викликів (EmergencyCalls) зі статусом «Активний» (Active).

Автоматичне надсилання сигналу здійснюється у випадках, коли зафіксовано п’ять поспіль критичних значень частоти серцевих скорочень менше 35 або більше 190 ударів на хвилину. Для оцінки частоти серцевих скорочень були враховані актуальні рекомендації Всесвітньої організації охорони здоров’я та провідних кардіологічних асоціацій [18]. Ці значення

беруться з кешу Redis, де зберігається буфер останніх точок трекінгу. У разі недостатньої кількості точок надсилання не виконується. Перевірка критичності виконується за допомогою умови, яка аналізує останні п’ять значень з кешу:

```
var isCritical = pointsResult.Data.All(p =>
    p.SmoothedHeartRate < 35 || p.SmoothedHeartRate > 190);
```

У задачах аналізу ситуаційної обізнаності важливим є не лише збір, а й оцінка достовірності кожного типу інформації [17]. Щоб уникнути надмірної генерації SOS-сигналів, реалізовано механізм повторної затримки (cooldown), через який повторне автоматичне надсилання можливе лише через п’ять хвилин після попереднього. Також перевіряється, чи вже існує активний сигнал від цього учасника.

Кожна автоматична спроба створення SOS-сигналу виконується з до двох повторних спроб у разі помилки з логуванням кожного етапу. Тип сигналу в цьому випадку фіксується як «Автоматичний SOS за пульсом» (AutomaticPulseSOS).

Після створення сигнал одразу стає доступним усім учасникам відповідної подорожі. Для цього використовується механізм надсилання повідомлень у реальному часі на основі SignalR, який також застосовується в модулі обміну повідомленнями (див. підрозділ 4.5).

4.4 Система кешування

Для зменшення навантаження на базу даних і забезпечення швидкого доступу до актуальних даних у системі реалізовано кешування з використанням Redis. Кеш застосовується переважно для зберігання останніх трекінгових точок кожного учасника подорожі, а також результатів допоміжних обчислень.

Під час кожного надсилання трекінгового запису нова точка (координати, згладжене значення пульсу, мітка часу) додається до кешу. Для кожного користувача зберігається п’ять останніх точок у форматі об’єкта «Остання точка трекінгу» (LastTrackingPointDto). Якщо таких записів уже п’ять, найстаріший видаляється, а новий додається в кінець буфера. Це забезпечує обмежений обсяг пам’яті та стабільну швидкодію кешу.

Перед виконанням виявлення перевищення пульсу для автоматичного SOS-виклику дані витягуються з Redis. У разі відсутності кешу або недостатньої кількості записів система виконує звернення до основної бази даних. Після цього кеш оновлюється на основі отриманої актуальної інформації.

Кожен запис у Redis має обмежений термін зберігання (time-to-live), який у поточній реалізації становить 5 хвилин. Це дає змогу зберігати лише актуальні пульсові та географічні дані без потреби в ручному очищенні.

Фрагмент виклику додавання запису в кеш:

```
await _db.StringSetAsync(key, json, TimeSpan.FromMinutes(5));
```

Після завершення часу життя ключа кешу запис автоматично видаляється з Redis-сервера. У системі кешування відіграє ключову роль у виконанні автоматичних перевірок у трекінговому модулі. Воно необхідне для виявлення критичних станів частоти серцевих скорочень, розрахунку відстані між точками без прямого звернення до бази даних, а також для попередніх аналітичних обчислень статистики.

Модуль кешування реалізовано у вигляді ізольованого сервісу, який не залежить від конкретної реалізації Redis.

Централізоване кешування дає змогу зменшити кількість звернень до основної бази даних, знизити навантаження на систему при високій частоті запитів (один запис кожні 15–20 секунд від кожного учасника) та підвищити швидкість реакції на зміну стану користувача.

Після завершення подорожі кожному учаснику надається можливість переглянути підсумкову статистику, сформовану на основі збережених трекінгових даних. Така статистика включає загальну пройдену відстань, середню швидкість руху, середню частоту серцевих скорочень і маршрут переміщення.

Відстань розраховується як сума усіх значень, обчислених за формулою Гаверсина під час надходження кожної точки. Ці значення зберігаються в полі «Відстань від попередньої точки, км» (DistanceFromPreviousKm), а їх загальна сума зберігається в полі «Загальна відстань, км» (TotalDistanceKm), яке належить до сутності участі в подорожі.

Середня швидкість визначається шляхом ділення загальної дистанції на сумарний час активного трекінгу. Якщо в системі зафіксовано втрату сигналу або наявні великі проміжки між точками, ці інтервали не враховуються під час обчислення.

Показник середньої частоти серцевих скорочень формується на основі попередньо згладжених і зашифрованих значень, що зберігаються разом із кожною точкою. Під час обробки ці значення розшифровуються і агрегуються у вигляді середнього арифметичного. У випадку недостатньої кількості даних у системі передбачено механізм із маркуванням результату як «недостатньо інформації».

Генерація статистики виконується окремим сервісом у шарі бізнес-логіки. Підсумкові дані передаються через інтерфейс прикладного програмування у вигляді спеціального об’єкта передачі даних.

Для забезпечення точності та узгодженості статистичних показників система також використовує контроль узгодженості часу надходження даних. Усі GPS- і біометричні значення маркуються часовими мітками з точністю до секунди, що дозволяє виключити повтори або зміщення у хронології. У випадку виявлення аномалій, таких як дублікати або перевищення максимально допустимого інтервалу між точками, відповідні записи

позначаються як невалідні й не враховуються під час обчислення фінальних метрик.

4.5 Модуль обміну повідомленнями

Модуль обміну повідомленнями реалізовано з використанням бібліотеки SignalR (SignalR), яка забезпечує двосторонній обмін даними в режимі реального часу поверх протоколу WebSocket. Це дозволяє надсилати повідомлення одразу після їх створення, без потреби періодичного опитування сервера, що знижує навантаження на систему та підвищує швидкість взаємодії.

Серверна частина модуля складається з трьох основних компонентів: класу зв’язку з клієнтами (UniversalHub), сервісу обробки повідомлень (MessageService) та інтерфейсу для надсилання сповіщень (IRealTimeNotifier). При підключенні до хабу клієнт передає ідентифікатор групи (travelGroupId), після чого підключення додається до відповідної групи з'єднання. Обробка підключень і відключень виконується в методах OnConnectedAsync та OnDisconnectedAsync відповідно.

Повідомлення створюються у разі, якщо користувач має підтверджену участь у подорожі й не залишив її. Після збереження у базу повідомлення трансформується у формат об’єкта передачі даних та надсилається до групи учасників за допомогою сервісу SignalRNotifier, який викликає метод:

```
await hubContext.Clients.Group(groupKey)
    .SendAsync(methodName, payload);
```

Цей підхід забезпечує ізоляцію логіки надсилання сповіщень у межах універсального сервісу, який можна повторно використовувати в інших модулях. Зокрема, SignalR застосовується також у модулі екстрених викликів (див. підрозділ 4.3), де використовується той самий механізм для оперативного розсилання сигналів SOS усім учасникам подорожі.

4.6 Безпека системи

Безпека системи реалізована як багаторівнева модель, яка охоплює автентифікацію, шифрування, валідацію доступу та централізовану обробку помилок. Основну увагу приділено захисту персональних і біометричних даних, а також перевірці прав користувача при кожній взаємодії з прикладним інтерфейсом.

Для автентифікації використовується маркер доступу у форматі JSON Web Token. Він створюється на сервері з обмеженим терміном дії і містить у собі набір атрибутів: ідентифікатор користувача, електронну пошту, ім’я користувача, призначення токена та ознаку адміністратора системи.

Паролі користувачів зберігаються у вигляді хешів, отриманих за допомогою алгоритму Bcrypt, який забезпечує високу ентропію та індивідуальну сіль для кожного запису. Сервіс хешування реалізує окремо функції створення хешу та його перевірки.

Для шифрування чутливих даних, зокрема координат і пульсу, використовується симетричний алгоритм AES. Фрагмент реалізації наведено нижче:

```
    public string Encrypt(string plainText, string? secretKey = null)
    {
        var keyBytes = GetAesKey(secretKey);
        using var aes = Aes.Create();
        aes.Key = keyBytes;
        aes.GenerateIV();

        using var encryptor = aes.CreateEncryptor();
        var plainBytes = Encoding.UTF8.GetBytes(plainText);
        var cipherBytes = encryptor.TransformFinalBlock(plainBytes,
0, plainBytes.Length);

        var result = new byte[16 + cipherBytes.Length];
        Buffer.BlockCopy(aes.IV, 0, result, 0, 16);
        Buffer.BlockCopy(cipherBytes, 0, result, 16,
cipherBytes.Length);

        return Convert.ToBase64String(result);
```

```
}
```

Ключ шифрування зберігається у конфігураційному файлі, а перед використанням додатково хешується за алгоритмом SHA-256. Шифрування реалізується у вигляді окремого сервісу.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Під час розробки серверної частини програмної системи TravelSync, призначеної для організації групових подорожей, особлива увага приділялася забезпеченню правильності реалізації бізнес-логіки та стійкості системи до граничних ситуацій. У зв’язку з цим було обрано поєднання модульного тестування (unit testing) та інтеграційної перевірки через Swagger UI. Такий підхід дозволив перевірити окремі сервіси в ізоляції, а також емпірично підтвердити працездатність усієї системи шляхом взаємодії з REST API.

Модульне тестування забезпечує перевірку окремих методів і сервісів у суворо контрольованому середовищі без підключення до реальної бази даних. Це дозволяє виявити логічні помилки ще до інтеграції в загальну систему. Для реалізації тестів було використано фреймворк NUnit у поєднанні з бібліотекою NSubstitute, що надала змогу створювати підставні реалізації залежностей. Для емулювання асинхронних запитів LINQ застосовувалась бібліотека MockQueryable.NSubstitute.

Інтеграційне тестування здійснювалось за допомогою Swagger UI, автоматично згенерованого з OpenAPI-специфікації. Такий підхід дав можливість безпосередньо надсилати HTTP-запити до ендпоінтів сервера, перевіряти формати вхідних та вихідних даних, коди відповідей і поведінку системи в умовах відсутності авторизації, недійсних параметрів або порушень бізнес-правил. На рис. 5.1 відображений фрагмент тестувального інтерфейсу.

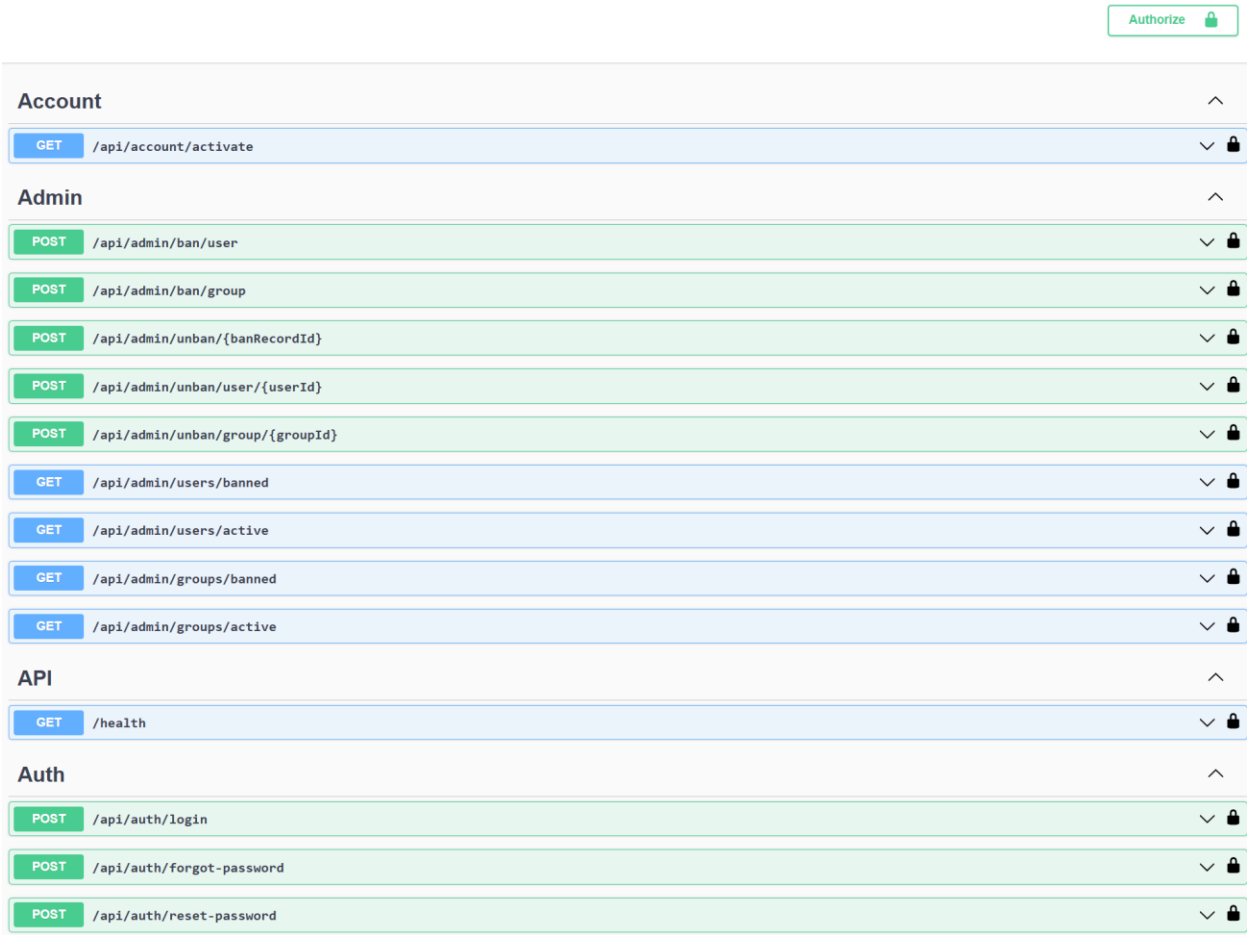

Рисунок 5.1 – Виконання інтеграційних тестів через Swagger UI (рисунок виконано самостійно)

У процесі модульного тестування було реалізовано 26 тестів, які охоплюють основні функціональні сервіси: TravelService, ParticipationService, TrackingService, UserService та EmergencyCallService. Кожен із них протестовано як у типових позитивних сценаріях, так і в умовах обмежень, помилок або порушення бізнес-логіки.

У сервісі подорожей перевірено створення маршруту, оновлення даних подорожі, зміну статусу, фільтрацію за назвою, датою та вподобаннями користувача, а також сортування за результатами кластеризації. Окремо протестовано обробку спроби оновити або видалити неіснуючу або вже активну подорож, а також ситуації з недостатніми правами.

Сервіс участі було протестовано на предмет подання заявки, підтвердження та відхилення участі, виходу з групи, підтвердження фактичного приєднання, а також видалення участі до початку подорожі. Значну увагу приділено сценаріям, пов’язаним із роллю адміністратора, перевіркою права на дію, перевищенням ліміту учасників та спробами повторної заявки.

У трекінговому модулі перевірено надсилання координат та пульсу, обробку буфера останніх значень, розрахунок дистанції, оновлення загальної пройденої відстані, шифрування даних за допомогою AES, збереження до Redis та створення автоматичних SOS-сигналів при виявленні аномального серцебиття. Тестування включає моделювання критичних станів для перевірки стійкості до помилок.

Додаткове навантажувальне тестування модуля трекінгу проводилось з метою визначення оптимального інтервалу передачі даних, що впливає на точність моніторингу та стабільність системи. Тестування виконувалося за допомогою утиліти k6 [12], яка дозволяє моделювати контрольоване навантаження на REST API. Було змодельовано сценарій із 1000 одночасних користувачів, які надсилали координати, пульс та мітку часу з інтервалом 10, 20 та 30 секунд.

У процесі тестування фіксувалися ключові метрики: відсоток успішно оброблених запитів, частка відмов та середній час відповіді сервера. Отримані результати наведено в таблиці 5.1.

Таблиця 5.1 – результати тестування працездатності модуля трекінгу за різного рівня навантаження:

Інтервал передачі, с	Успішні запити, %	Відмови, %	Середній час відповіді, мс
10	91,3	8,7	290
20	98,9	1,1	260

30	99,2	0,8	230

Як видно з таблиці, інтервал у 10 секунд призводить до підвищеного навантаження на сервер і збільшення часу відповіді, тоді як 30 секунд зменшують навантаження, але затримують реакцію на фізіологічні зміни. Тому інтервал у 20 секунд було визнано найкращим компромісом між оперативністю реагування та стабільністю системи в реальному часі.

У сервісі користувача протестовано створення нового користувача з перевіркою унікальності email та username, отримання профілю за ідентифікатором, валідацію claims з JWT-токена, а також обробку випадків конфліктів під час реєстрації. Окремо перевірено обробку авторизації та хешування паролів.

У модулі екстрених викликів перевірено створення SOS-сигналу вручну, перегляд активних викликів у межах подорожі та оновлення їх статусу. Тестування охоплює як позитивні сценарії, так і спроби доступу без участі в подорожі або з неактуальним статусом.

Інтеграційне тестування проводилось безпосередньо в процесі розробки через Swagger UI. Кожен реалізований ендпоінт проходив перевірку на коректну обробку авторизованих і неавторизованих запитів, відповідність запитів специфікації, а також на обробку граничних значень і невалідних параметрів. Наприклад, при спробі приєднатися до подорожі з перевищеним лімітом учасників система повертає помилку з поясненням, а при зміні статусу без адміністративних прав — повідомлення про відсутність дозволу.

Усі тести виконувалися локально в середовищі розробки без помилок. Після запуску unit-тестів було підтверджено проходження 26 сценаріїв, що охоплюють основні бізнес-вимоги. Середній час виконання одного тесту не перевищував 100 мілісекунд, що свідчить про ефективність архітектурних рішень і правильну побудову шарів логіки.

На рисунку 5.2 наведено результати виконання модульних тестів, які підтверджують коректність функціоналу основних сервісів системи.

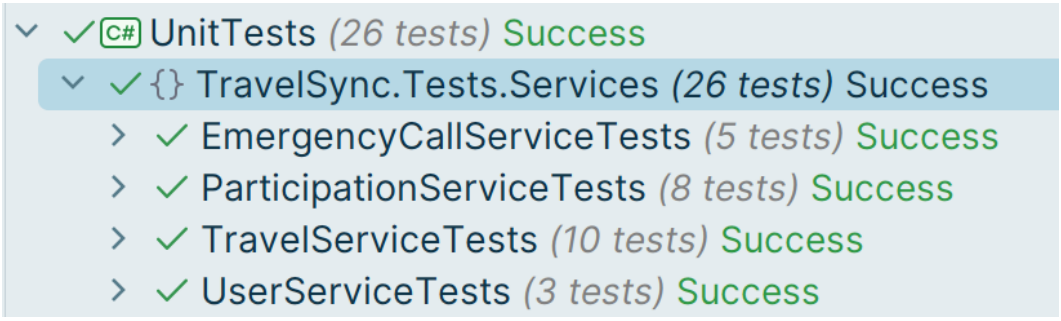

Рисунок 5.2 – Результати виконання модульних тестів у середовищі Rider (рисунок виконано самостійно)

Результати тестування підтверджують, що реалізоване програмне забезпечення відповідає функціональним вимогам, демонструє стійкість до помилок і забезпечує коректну роботу в типових і виняткових сценаріях.

ВИСНОВКИ

У результаті виконання атестаційної роботи було спроєктовано та реалізовано серверну частину програмної системи для організації групових подорожей. Розроблена система забезпечує реєстрацію користувачів, підтвердження електронної пошти, авторизацію з використанням JWT, управління подорожами, обробку заявок на участь, трекінг місцезнаходження та біометричних показників у режимі, наближеному до реального часу, формування статистичних звітів після завершення маршруту, обмін повідомленнями між учасниками, а також надсилання екстрених сигналів у разі критичних станів.

У межах реалізації було виконано моделювання предметної області, побудовано логічну структуру бази даних, налаштовано ключові зв’язки між сутностями, реалізовано перевірки бізнес-правил і обмеження відповідно до ролей користувачів. Система враховує обмеження на кількість учасників у групі, розділяє повноваження між організаторами та звичайними користувачами, перевіряє права доступу в кожному модулі та забезпечує недопущення критичних конфліктів, наприклад, виходу останнього адміністратора з активної подорожі.

Трекінговий модуль приймає координати та пульс із фіксованим інтервалом надходження. Для захисту даних використано симетричне шифрування за алгоритмом AES, а для згладжування пульсових значень — метод ковзного середнього. Всі трекінгові точки зберігаються незалежно від змін, що дозволяє фіксувати втрату зв’язку або підозрілі стани. Алгоритм Гаверсина використовується для розрахунку пройденої відстані між точками, а критичні значення пульсу автоматично активують створення SOS-сигналу з розсилкою повідомлень усім учасникам подорожі через механізм SignalR.

Обмін повідомленнями реалізовано на основі WebSocket-з’єднання з використанням бібліотеки SignalR. Кожен учасник приєднується до групи за

ідентифікатором подорожі, а повідомлення доставляються в реальному часі через серверний хаб, що обробляє підключення та від’єднання. Це дозволяє організувати постійний канал зв’язку без навантаження на сервер через опитування.

Серверну частину реалізовано з використанням технологій ASP.NET Core, PostgreSQL, Entity Framework Core, Redis і SignalR, що забезпечує масштабованість, продуктивність та відповідність сучасним вимогам до безпеки. Архітектура рішення включає чітке розділення на шари прикладного інтерфейсу, бізнес-логіки, доступу до даних та інфраструктури, з окремим доменним рівнем. Реалізовано централізовану обробку виключень, шифрування конфіденційних даних, кешування останніх трекінгових точок, відправлення електронної пошти, логування критичних дій і валідацію запитів на основі FluentValidation. Усі API протестовано за допомогою Postman, проведено перевірку коректності сценаріїв, включаючи обробку помилок та граничні умови.

Розроблена серверна частина є завершеним функціональним ядром системи та готова до масштабування, інтеграції з мобільним клієнтом і розгортання в умовах продуктивного середовища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Group Travel Planning Apps Decoded: Comprehensive Analysis [Електронний ресурс]. – Режим доступу: https://www.marketreportanalytics.com/reports/group-travel-planning-apps-75331 (дата звернення: 10.04.2025). – Загол. з екрану.

2. Troupe – Group Travel Planning App by JetBlue [Електронний ресурс]. – Режим доступу: https://troupe.com (дата звернення: 12.04.2025). – Загол. з екрану.

3. Wanderlog – Trip Planner with Itineraries & Maps [Електронний ресурс]. – Режим доступу: https://wanderlog.com (дата звернення: 12.04.2025). – Загол. з екрану.

4. Komoot – Route Planner and Outdoor Navigation [Електронний ресурс]. – Режим доступу: https://www.komoot.com (дата звернення: 12.04.2025). – Загол. з екрану.

5. Дудар З.В., Кобзєв В.Г., Семенець В.В. Історія, стан та перспективи розвитку загальнонаціональних систем електронної ідентифікації користувачів // Міжнародна науково-практична конференція «Застосування інформаційних технологій». – 2023. – Режим доступу: http://kinf.nangu.edu.ua/since_files/Doc/tezMNPC_2023.pdf (дата звернення: 10.06.2025). – Загол. з екрану.

6. Microsoft. ASP.NET Core 8 Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core (дата звернення: 17.04.2025). – Загол. з екрану.

7. Мазурова О.О., Широкопетлєва М.С. Навчальний посібник з дисципліни «Бази даних» (Ч.1) для студентів напрямку 050103 – «Програмна інженерія» [Електронний ресурс]. – Харків: ХНУРЕ, 2012. – 176 с. – Режим

доступу: http://catalogue.nure.ua/download=150083 (дата звернення: 10.06.2025). – Загол. з екрану.

8. Advanced Encryption Standard (AES) [Електронний ресурс] / National Institute of Standards and Technology (NIST). – FIPS PUB 197. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (дата звернення: 17.04.2025). – Загол. з екрану.

9. Літвін С.Г. Метод онтологічного опису при побудові сервіс-орієнтованих систем розподіленного навчання // Сучасний стан наукових досліджень та технологій в промисловості. – 2024. №1 (27). – С. 39–53. – Режим доступу: https://www.itssi-journal.com/index.php/ittsi/article/view/458 (дата звернення: 10.06.2025). – Загол. з екрану.

10. Методичні вказівки до лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування» для студентів першого (бакалаврського) рівня вищої освіти усіх форм навчання спеціальності 121 «Інженерія програмного забезпечення», освітньо-професійна програма «Програмна інженерія» [Електронний ресурс] / Упоряд.: Ю.Ю. Черепанова, В.М. Бондарєв, М.С. Широкопетлєва, А.О. Функендорф. – Харків: ХНУРЕ, 2020. – 28 с. – Режим доступу: https://catalogue.nure.ua/download=238143 (дата звернення: 10.06.2025). – Загол. з екрану.

11. Груздо І.В., Галуза О.А. Методичні вказівки до практичних занять з дисципліни «Інформаційні системи в аналізі даних». – Харків: ХНУРЕ, 2024. – 68 с. – Загол. з екрану.

12. Sinnott R. W. Virtues of the Haversine. – Sky and Telescope, Vol. 68, No. 2, 1984, p. 159. – [Електронний ресурс]. – Режим доступу: https://www.movable-type.co.uk/scripts/latlong.html (дата звернення: 30.04.2025). – Загол. з екрану.

13. Карні, Ч. Ф. Ф. Алгоритми для геодезичних розрахунків. // Journal of Geodesy. – 2013. – Т. 87(1). – С. 43–55. – [Електронний ресурс]. – https://doi.org/10.1007/s00190-012-0578-z.

14. Majka M. Moving Averages in Time Series Analysis [Електронний ресурс] / Marcin Majka. – 2024. – Режим доступу: https://www.researchgate.net/profile/Marcin-Majka-2/publication/384080857_Moving_Averages_in_Time_Series_Analysis/links/66e9d34ea438c86fdcd41de7/Moving-Averages-in-Time-Series-Analysis.pdf (дата звернення: 01.05.2025). – Загол. з екрану.

15. World Health Organization. *Reference card for WHO Emergency Unit Form – General* [Електронний ресурс]. – 2021. – Режим доступу: https://cdn.who.int/media/docs/default-source/documents/emergency-care/reference-card-for-who-emergency-unit-form-general.pdf (дата звернення: 23.06.2025). – Загол. з екрану.

16. Бондарєв В.М. Ментор – система оперативного контроля знаний // Научно-практическая конференция «Информационные технологии в образовании». – 2005. – Вып. 4.

17. Усачов В.О., Дудар З.В. Методи кваліметричного оцінювання інформації в середовищах з відкритими даними // Збірник наукових праць «Сучасні проблеми телекомунікацій». – 2023. – №4 (48). – С. 57–66. – Режим доступу: https://journals.nupp.edu.ua/spt/article/view/3125 (дата звернення: 10.06.2025). – Загол. з екрану.

18. Посилання на GitHub репозиторій з вихідним кодом, відео та специфікацією. URL: https://github.com/NureTopchiiDaria/2025_B_PI_PZPI-21-3_Topchii_D_D (дата звернення: 08.06.2025).