Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ НАПАРНИКА ДЛЯ СПІЛЬНИХ ЗАНЯТЬ СПОРТОМ»

Виконав: студент 4 курсу, групи 2ІСТ-21б
спеціальності 126 – «Інформаційні
системи та технології»
_____ Дмитро КОЗУБ
Керівник: доц. каф. САІТ
_____Євгеній КРИЖАНОВСЬКИЙ
«08» ____06____ 2025 р.
Рецензент: к.т.н., ас. каф. КН
_____Ілона БОГАЧ
«08» ____06____ 2025 р.

Допущено до захисту

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«06» ____06____ 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма – Прикладні інформаційні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"28" 05 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Козубу Дмитру Вікторовичу

1. Тема роботи: «Інформаційна система пошуку напарника для спільних занять спортом»
керівник роботи: Крижановський Є.М., доц. каф. САІТ
затверджені наказом ВНТУ від «20» 05 2025 року № 97
2. Термін подання студентом роботи 31.05.25 р.
3. Зміст текстової частини:
1) Аналіз предметної області;
2) Вибір оптимальної ІТ-інфраструктури;
3) Проєктування інформаційної системи;
4) Розроблення інформаційної системи пошуку напарника для спільних занять спортом.
4. Перелік ілюстративного матеріалу:
1) Діаграма кооперації пошуку партнера;
2) Діаграма класів;
3) Загальна діаграма станів системи;
4) Діаграма пошуку партнера;
5) Інтерфейс інформаційної системи.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 28.05.25 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	01.04	15.04	Вик
2	Вибір оптимальної IT-інфраструктури	15.04	30.04	Вик
3	Проєктування інформаційної системи	01.05	10.05	Вик
4	Розроблення інформаційної системи пошуку напарника для спільних занять спортом	10.05	20.05	Вик
5	Оформлення матеріалів до захисту БКР	20.05	30.05	Вик

Студент Дмитро КОЗУБ

Керівник роботи Євгеній КРИЖАНОВСЬКИЙ

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 84 сторінок формату А4, на яких є 40 рисунків, список використаних джерел містить 27 найменувань.

Метою дипломної роботи є розробка інформаційної системи для пошуку спортивного напарника, яка спрощує організацію спільних тренувань шляхом фільтрації за вподобаннями, місцем та активністю користувача.

У межах проєкту створено мобільний додаток для платформи Android з використанням мови програмування Kotlin. Реалізовано інтуїтивно зрозумілий інтерфейс на базі компонентів Material Design, локальне зберігання даних за допомогою бібліотеки Room, функцію надсилання запрошень, пошук за тегами, а також інтеграцію з картографічною службою для відображення спортивних локацій. Розроблена інформаційна система забезпечує автономну роботу без постійного інтернет-з'єднання та адаптована під потреби україномовної аудиторії.

Ключові слова: інформаційна система, мобільний додаток, Android, Kotlin, Room, спортивна активність, пошук партнера.

ABSTRACT

The bachelor's thesis consists of 84 A4 pages, which contain 40 figures, the list of sources used contains 27 titles.

The aim of the work is to develop an information system for finding a sports partner, facilitating the organization of joint training sessions through filtering by preferences, location, and activity.

As part of the project, a mobile application for the Android platform was developed using the Kotlin programming language. The app features an intuitive user interface based on Material Design components, local data storage via the Room library, invitation functionality, tag-based search, and integration with a mapping service to display sports locations. The developed information system operates autonomously without requiring a constant internet connection and is tailored to the needs of Ukrainian-speaking users.

Key words: information system, mobile application, Android, Kotlin, Room, sports activity, partner searc

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі дедалі більше людей прагне підтримувати активний спосіб життя, займаючись спортом не лише для підтримки та розвитку своєї фізичної форми, але й для соціальної взаємодії. Водночас виникають труднощі пов’язані із пошуком напарника для спільних спортивних активностей. На це впливає немало факторів, зокрема спортивні вподобання, графік, локація, рівень підготовки та інше. Інформаційна система, яка здатна спростити та покращити пошук напарника для спільних занять спортом, могла б стати корисним інструментом для тих, хто бажає займатися спортом в компанії однодумців.

Мета і задачі дослідження. Метою роботи є розроблення інформаційної системи у вигляді мобільного застосунку, що дозволить здійснювати пошук напарника для спільних занять спортом з урахуванням спортивних інтересів, геолокації та можливості надсилання запрошень. Оптимальність IT-інфраструктури та технологій системи визначаються такими критеріями, як підтримка локального збереження даних, зручність інтеграції, наявність документації, спільнота підтримки та відповідність специфіці Android-розробки.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область.
- дослідити та визначити необхідний функціонал інформаційної системи.
- розробити інформаційну систему відповідно поставленим критеріям та вимогам.

Об’єкт дослідження є процес розроблення інформаційної системи для пошуку напарника для спільних занять спортом.

Предмет дослідження є методи та програмні засоби створення інформаційних систем, що забезпечують фільтрацію, пошук, надсилання запрошень на спільні заняття спортом та перегляд отриманих запрошень.

Публікації. За результатами виконання даної роботи опубліковано тези доповідей на тему: «Інформаційна система пошуку напарника для спільних

занять спортом» на LIV Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2025) [1].

1 **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

1.1 Проблематика пошуку напарника

На сьогоднішній день фізична активність зросла як ніколи в повсякденному житті людини. В епоху цифрових технологій, коли всі необхідні сервіси знаходяться буквально під рукою, потрібно не забувати про підтримку своєї фізичної форми. Регулярні заняття спортом допомагають не лише покращити фізичний стан організму, а й психологічне самопочуття. Також фізичні активності знижають рівень стресу та стимулюють людей позитивніше дивитися на навколишній світ. Та проти цього всього існує вагома причина, яка заважає людям систематично займатись спортом чи іншими фізичним активностями. Цією причиною є недостача соціальної взаємодії та мотивації, насамперед відсутність напарника для спільних тренувань [2].

Багато людей мають бажання тренуватись, однак через щільний графік, переїзди, зміну місця проживання або специфічні вподобання щодо виду спорту вони не можуть знайти людину, з якою б могли займатись разом. Залишаючись наодинці, частина користувачів втрачає інтерес до занять або відмовляється від них після кількох невдалих спроб. Соціальна складова у спорті відіграє ключову роль: тренування в компанії сприяють більшій відповідальності, підвищенню інтересу, формуванню позитивного змагального духу та досягненню кращих результатів [2].

Традиційно пошук партнера для занять відбувається за допомогою особистих знайомств, через спортзали або рекомендації друзів. Однак такий підхід має суттєві обмеження: немає гарантії, що в колі знайомих буде людина з подібними спортивними цілями; труднощі у погодженні графіків та місць проведення тренувань; відсутність прозорого механізму пошуку й фільтрації потенційних партнерів.

З появою цифрових технологій з’явилась можливість вирішити цю проблему. Проте аналіз існуючих інструментів показує, що більшість мобільних

додатків зі спортивної тематики зосереджуються на трекінгу фізичної активності (Strava, Google Fit, Samsung Health), складанні індивідуальних програм тренувань (Nike Training Club, Fitbod) або веденні статистики тренувань [3].

Функціонал пошуку напарників у цих додатках зазвичай або повністю відсутній, або має обмежені можливості — наприклад, відображення лише друзів або учасників однієї спільноти без фільтрів за інтересами, місцем проживання чи графіком тренувань.

Користувачі, які бажають знайти напарника, нерідко вдаються до альтернативних способів: публікацій у соціальних мережах, пошуку по чатах месенджерів або групах у Instagram чи Viber. Такі методи не завжди є безпечними або ефективними, оскільки складно перевірити достовірність інформації про людину, відсутня можливість швидкого сортування або фільтрації за критеріями, користувачам доводиться самостійно координувати процес [4].

Таким чином, існує чітко виражена потреба в інформаційній системі у вигляді мобільного додатку, який би дозволяв користувачам швидко та ефективно знаходити собі напарників для спільних занять спортом. Така система стала б незамінним інструментом в досягнені спортивних цілей багатьох людей, які ведуть активний спосіб життя.

1.2 Аналіз ринку мобільних додатків

Ринок мобільних застосунків, що сприяють спортивній активності, є доволі насиченим. Серед популярних платформ варто відзначити Strava, Nike Training Club, Adidas Running та інші (рис. 1.1-2). Однак більшість таких додатків орієнтовані передусім на фіксацію спортивних результатів, аналіз тренувань або індивідуальну активність. Функціональність пошуку партнера часто реалізована або частково, або взагалі відсутня [5].

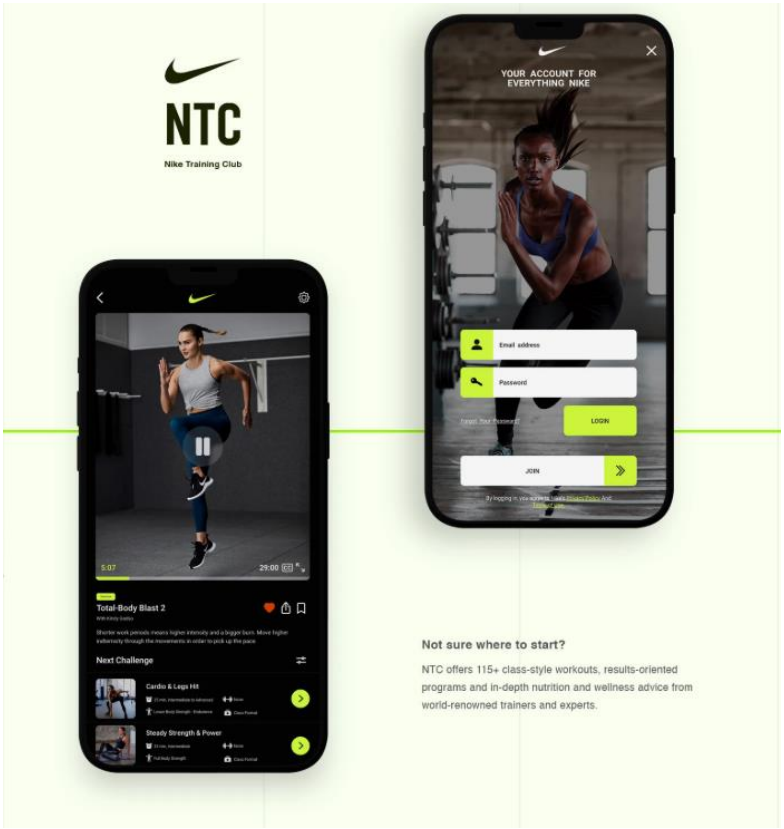

Рисунок 1.1 – Додаток Nike Training Club.

Рисунок 1.2 – Додаток Adidas Running.

Світовий ринок мобільних застосунків у сфері здоров’я та фітнесу демонструє стабільне зростання. Згідно з аналітичним звітом Verified Market Research, у 2023 році його обсяг становив приблизно 9,85 мільярда доларів США, а прогнозується зростання до 42,43 мільярда доларів США до 2031 року із середньорічним темпом приросту (CAGR) близько 19,7%. Такий попит зумовлений не лише зростаючою популярністю фітнесу, але й розвитком технологій трекінгу активності, носимих пристроїв і соціалізованих платформ для здорового способу життя [6]. Ринок очікує на рішення, які поєднують не лише тренувальний функціонал, а й елементи взаємодії між користувачами — зокрема функцію пошуку напарника.

На українському ринку аналогічні сервіси не мають широкого розповсюдження. Часто користувачі вдаються до пошуку тренувальних партнерів через соціальні мережі, форуми або локальні чати, що не є зручним або безпечним способом для формування довготривалих спортивних зв’язків.

Таким чином, створення мобільного додатку з інтуїтивно-зручним інтерфейсом, фільтрами пошуку, інтеграцією з картою та базою даних потенційних напарників, є доцільним і перспективним з погляду подальшого розвитку ринку.

1.3 Огляд існуючих інформаційних систем пошуку спортивних напарників

Аналіз аналогічних інформаційних систем є важливою частиною етапу проектування, оскільки дозволяє визначити, які функціональні можливості вже реалізовані, які підходи виявились успішними, а які мають недоліки. Це дає змогу врахувати досвід розробників та користувачів існуючих сервісів, уникнути повторення помилок, а також адаптувати найкращі практики для власного проєкту.

Однією з платформ, яка частково реалізує ідею пошуку спортивних партнерів, є Playseek. Це мобільний застосунок, що надає користувачам змогу знаходити партнерів для спільних спортивних активностей, таких як теніс,

футбол, біг, фітнес тощо. На рисунку 1.3 зображено вигляд подій всередині додатку [7].

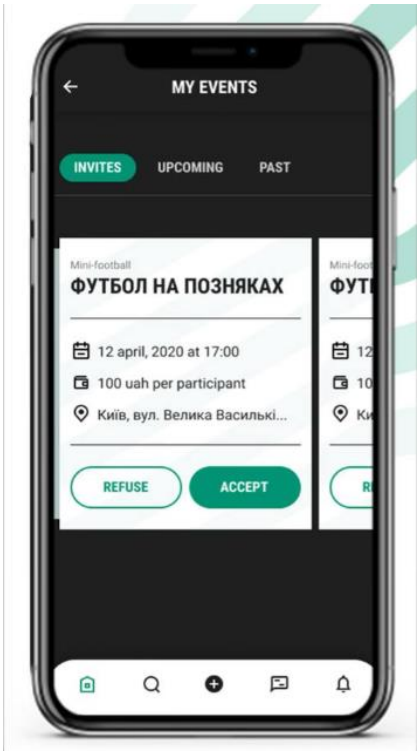

Рисунок 1.3 – Огляд подій в Playseek.

Playseek дозволяє створювати публічні події, запрошувати учасників, переглядати профілі інших користувачів та фільтрувати їх за видом спорту, рівнем підготовки та розташуванням (рис 1.4-1.5).

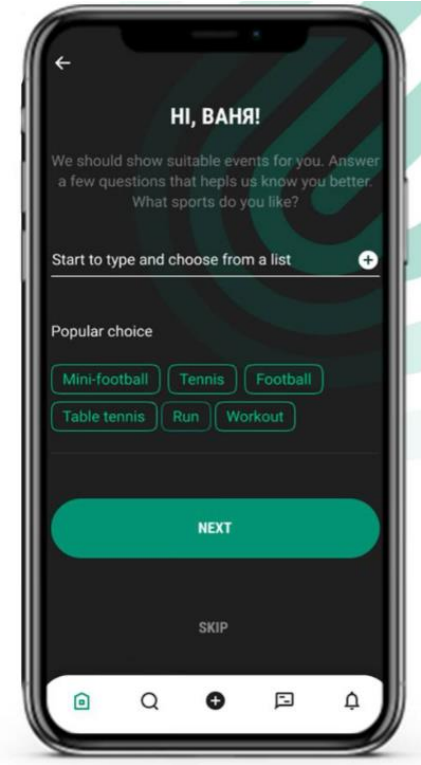

Рисунок 1.4 – Пошук подій за ключовими словами в Playseek.

Рисунок 1.5 – Відображення потенційних учасників спортивних подій на карті в Playseek.

Загалом додаток Playseek є зручним способом знайти спортивну подію та доєднатися до неї. Даний додаток має зручний пошук за ключовими словами, підтримки геолокації та можливість переглядати відомості про спортивні події (час, місце, кількість учасників).

Веб-сервіс FindySport орієнтований на організацію спільних тренувань серед користувачів з однаковими інтересами. Сервіс підтримує створення груп за видами спорту, пошук тренувань поблизу, а також надання відгуків після участі в події (рис 1.6-1.8) [8].

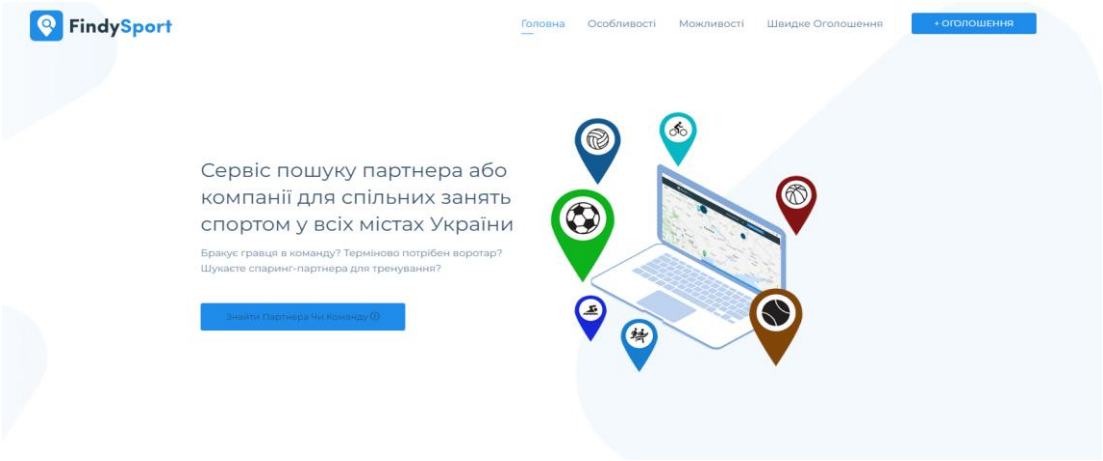

Рисунок 1.6 – Головна сторінка FindySport.

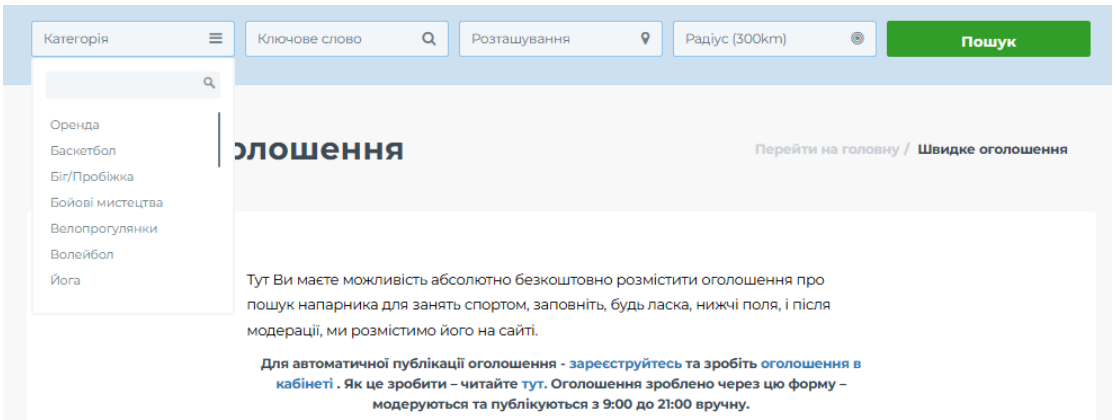

Рисунок 1.7 – Інтерфейс пошуку груп для спільних тренувань в FindySport.

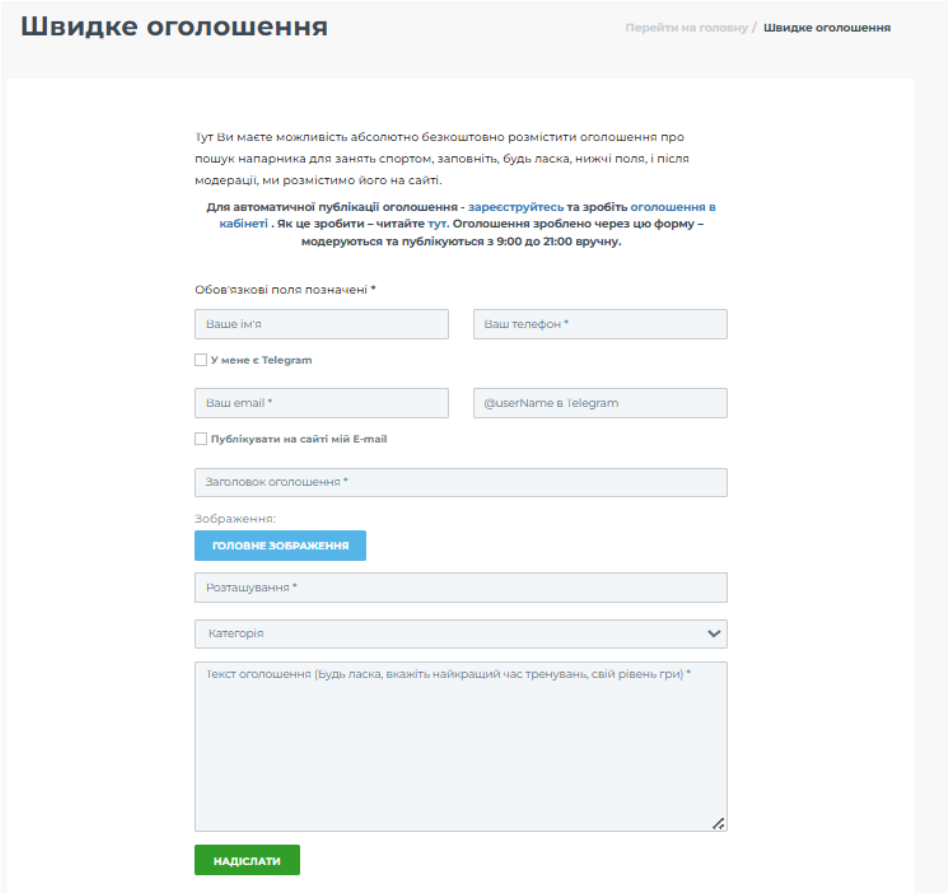

Рисунок 1.8 – Вікно запрошення для спільного тренування в FindySport.

Веб-сервіс FindySport є хорошим інструментом для людей, які прагнуть знайти групу з відповідними вподобаннями (баскетбол, велосипедні прогулянки, бойові мистецтва). Публікація власного оголошення з вказанням особистих вподобань щодо занять спортом допомагає краще підібрати напарника для спіільних занять спортом.

Ще одним прикладом подібного застосунку є Fitior. Основна його мета — підбір фітнес-тренера або партнера для тренувань за допомогою спеціального пошуку. Сервіс містить функціонал фільтрації за видом спорту, цілями, доступним часом (рис 1.9-1.12) [9].

Рисунок 1.9 – Вигляд сторінки з видами спорту в Fitior.

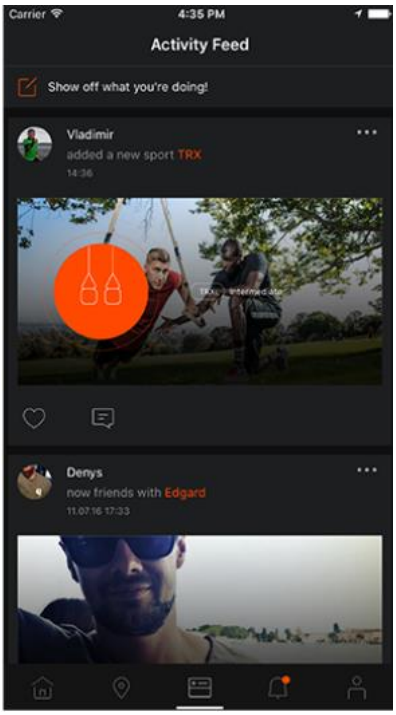

Рисунок 1.10 – Стрічка активностей в Fitior.

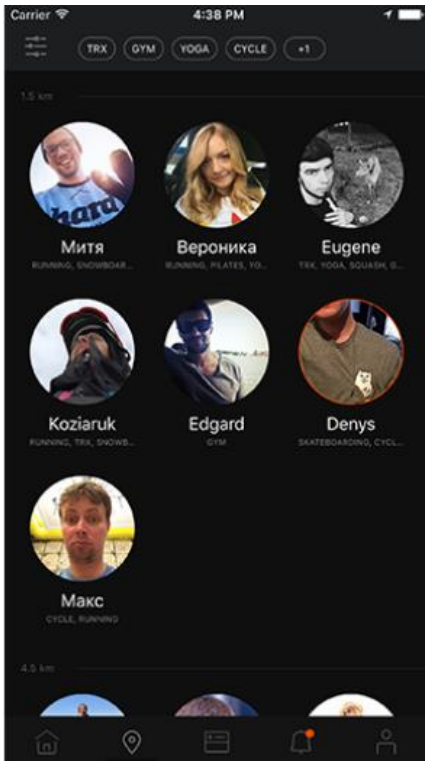

Рисунок 1.11 – Результат пошуку можливих напарників в Fitior.

Застосунок має простий інтерфейс, зручну систему пошуку напарника для стрічку активностей. Це непоганий спосіб знайти напарника для спільних занять спортом.

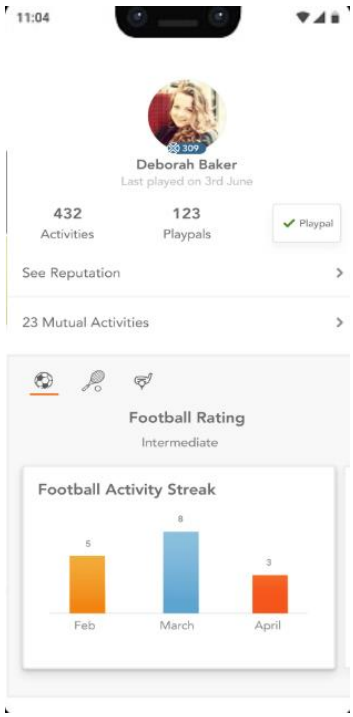

Рисунок 1.12 – Список спортивних подій в Playo.

Playo — це мобільний застосунок, що поєднує функціональність пошуку спортивних напарників з можливістю бронювання спортивних майданчиків. Сервіс орієнтований на активних користувачів, які займаються командними або індивідуальними видами спорту, і має на меті створити спільноту спортсменів у межах окремих міст або районів (рис 1.13) [10].

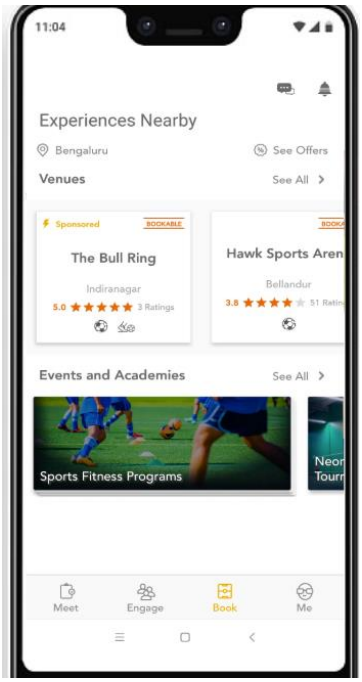

Рисунок 1.13 – Список спортивних подій в Playo.

Додаток Playo дозволяє здійснювати пошук партнерів для гри у футбол, теніс, бадмінтон, настільний теніс, баскетбол тощо; створювати івенти, спортивні зустрічі із відкритим або приватним доступом; фільтрувати користувачів за видом спорту, рівнем гри та доступністю по часу. Цей додаток гарний інструмент для організації спільних занять спортом.

Проаналізувавши всі існуючі аналоги, можна виділити спільні риси розглянутих систем:

– використання фільтрів за видом спорту та геолокацією;
– наявність профілю користувача з описом вподобань;

– орієнтація на великі міста або обмежене покриття.

1.4 Висновки

У цьому розділі виконано аналіз предметної області, а саме: досліджено і виявлено проблематику пошуку напарника, проаналізовано попит ринку на мобільні додатки, які спрямовані на просування спортивної тематики, розглянуто аналоги. В результаті виявлено, що всі існуючі аналоги лише частково охоплюють потреби спортивної взаємодії, проте не вирішують їх комплексно. Це створює передумови для створення нової інформаційної системи з урахуванням локальної специфіки, розширеної системи фільтрів, підтримкою візуалізації спортивних об’єктів на карті, можливістю прямого запрошення користувача до спільного заняття та збереженням інформації в локальній базі даних.

2 ВИБІР ОПТИМАЛЬНОЇ ІТ-ІНФРАСТРУКТУРИ

2.1 Формування системних вимог

Оптимальна інфраструктура для розробки інформаційної системи залежить від вимог до цієї системи. На основі проведеного аналізу можна визначити мінімально життєздатний продукт (MVP). Цей підхід представляє собою одну з технік розробки, де новий продукт створюється з достатнім набором функціональних особливостей для задоволення потреб перших користувачів [11].

Таким чином, до основних вимог інформаційної системи пошуку напарника для спільних занять спортом відносяться:

- функціонал пошуку партнерів за фільтрами (вид спорту, район, час, рівень, тип активності);
- система тегів/хештегів для уточнення інтересів користувача;
- візуалізація спортивних локацій на карті з маркерами;
- вікно з інформацією про користувача при натисканні на маркер;
- можливість надіслати запрошення або відхилити його;

Згідно з MVP методологією, остаточний набір функцій проектується і розробляється тільки після врахування відгуків перших користувачів продукту [11]. Потенційно, це б могли бути вимоги як:

- система сповіщень про отримання запрошення або повідомлення;
- чат між користувачами після підтвердження запрошення;
- підтримка рейтингів або відгуків щодо попередніх тренувань;
- синхронізація з календарем або Google Fit;
- інтеграція з зовнішніми API для пошуку спортивних об’єктів;

Сформовані вимоги відповідають головним потребам клієнта та надають корисну інформацію, що може вплинути на ухвалення подальших рішень для результативного розвитку програмного застосунку.

Для реалізації інформаційної системи пошуку напарника для спільних занять спортом доцільним є використання мобільної архітектури, яка забезпечує доступність, персоналізовану взаємодію та високу зручність для кінцевого користувача. В умовах сучасного способу життя саме мобільні пристрої є основним засобом комунікації, організації особистого часу та використання цифрових сервісів. Тому реалізація системи у форматі мобільного застосунку дозволяє найкраще задовольнити потреби цільової аудиторії [12].

Однією з ключових переваг такої архітектури є постійний доступ до системи незалежно від місцезнаходження користувача. Враховуючи, що більшість спортивних подій чи зустрічей відбувається поза межами стаціонарного робочого місця, саме мобільна версія системи є найбільш логічним та ефективним рішенням. Крім того, користувачі можуть оперативно отримувати повідомлення про запрошення, переглядати локації на карті, швидко фільтрувати потенційних напарників — і все це без потреби у використанні браузера чи комп’ютера.

Мобільна архітектура дозволяє реалізувати інтуїтивно зрозумілий інтерфейс, адаптований під малий екран смартфона, з оптимізованими елементами навігації. Адаптивність інтерфейсу та підтримка елементів візуалізації (наприклад, інтерактивних мап) підвищують рівень взаємодії та позитивно впливають на загальне враження від роботи з додатком [12].

З технічного погляду, мобільна архітектура також дозволяє реалізувати підключення до зовнішніх сервісів — картографічних платформ, геолокаційних API, сховищ даних. Це відкриває можливості для побудови гнучкої структури, яку можна масштабувати залежно від кількості користувачів, обсягу інформації та подальших функціональних розширень.

Крім того, реалізація у вигляді мобільного застосунку дозволяє врахувати параметри конфіденційності та безпеки, такі як зберігання даних локально, захист особистої інформації, обмеження доступу до профілів та контроль над запитами між користувачами. Це особливо важливо в умовах роботи з персональними профілями, історією активності та геоданими.

Отже, саме мобільна архітектура є найбільш доцільною у контексті розробки інформаційної системи для пошуку напарників для занять спортом, оскільки враховує як поведінкові особливості цільової аудиторії, так і технічні вимоги до сучасного застосунку.

2.2 Вибір програмного забезпечення.

Для успішної розробки інформаційної системи необхідно правильно визначити елементи програмного забезпечення. Середовище розробки (IDE), мова програмування, спосіб збереження та відображення даних, інтеграція додаткового функціоналу (наприклад Google Maps) та інші технології обираються в залежності від вимог, які визначені для розробки інформаційної системи. Так як вимоги було визначено, тепер необхідно вибрати необхідні елементи та технології, за допомогою яких в подальшому буде реалізований функціонал інформаційної системи у вигляді мобільного додатку.

Серед ключових технологій, які використовуються при створенні мобільних додатків, можна виділити:

- UI-фреймворки та бібліотеки: забезпечують реалізацію зручного та адаптивного інтерфейсу (наприклад, Material Components, Jetpack Compose, XML-макети, Flutter UI, SwiftUI).
- Інструменти для навігації: дозволяють організувати перемикання між екранами (Navigation Components, StackNavigator).
- Системи зберігання даних: як локальні (Room, SQLite, SharedPreferences), так і хмарні (Firebase, Cloud Firestore, AWS Amplify).
- Картографічні сервіси: Google Maps SDK, Mapbox, OpenStreetMap для візуалізації локацій та роботи з геолокаційними даними.
- API та інтеграції: REST API, JSON, Retrofit, OkHttp для обміну даними з сервером або іншими сервісами.
- Інструменти керування станом: ViewModel, LiveData, BLoC, Redux.

- Контроль версій: Git — для відстеження змін коду, командної роботи та збереження історії розробки.
- Середовища тестування: Firebase Test Lab, Espresso, JUnit — для перевірки функціоналу.

Інструменти та технології, які вибираються для конкретного проекту, залежать від його вимог, а також від досвіду та переваг розробників. Full-stack розробники можуть працювати над всіма аспектами проекту, а саме: frontend (інтерфейс користувача) та backend (серверна частина). Це забезпечує ефективність та економію часу, однак, вони не завжди мають глибокі знання в окремих областях. Через головоломку з декількома технологіями, full-stack розробники можуть іноді не мати достатнього часу для оновлення своїх навичок у всіх областях.

IDE (Integrated Development Environment) – це інтегроване середовище розробки, яке представляє собою комплексний програмний простір, що поєднує інструменти, необхідні для розробки програмного забезпечення. До таких інструментів як правило входять текстовий редактор для написання коду, інструменти для побудови та відлагодження програм, а також інші корисні функції, такі як система контролю версій. Різні середовища розробки (IDE) призначені для використання з різними мовами програмування та стеками технологій. Ось декілька популярних IDE для веб-розробки:

– Visual Studio Code: IDE від Microsoft, яке підтримує безліч мов програмування, включаючи JavaScript, TypeScript, Python, C#, PHP і багато інших. VS Code має велику кількість розширень, що надають додаткові можливості, такі як підтримка фреймворків та бібліотек, інтеграція з Git, підсвічування синтаксису, автодоповнення коду тощо [13].
– Android Studio: офіційне середовище розробки мобільних додатків для Android-платформи. Підтримує візуальне компонування інтерфейсу, роботу з базами даних, емуляцію пристроїв, вбудований Git та зневадження [14].

– Xcode: IDE для розробки застосунків під iOS, macOS та інші продукти Apple. Має інтеграцію з SwiftUI, візуальне редагування, розширене тестування [15].
– WebStorm: IDE від JetBrains, яке спеціально розроблено для JavaScript розробки, включаючи Node.js та фронтенд JavaScript фреймворки і бібліотеки, такі як React, Angular, Vue.js тощо [16].
– PyCharm: IDE від JetBrains, яке розроблено спеціально для Python, включаючи Django та Flask, і підтримує веб-розробку на Python [17].

Кожне середовище має свої сильні та слабкі сторони, тому вибір IDE залежить від потреб та вподобань конкретного розробника.

Android Studio є одним з найзатребуваніших інтегрованих середовищ розробки (IDE), до того ж воно чудово підходить для створення мобільних додатків [14].

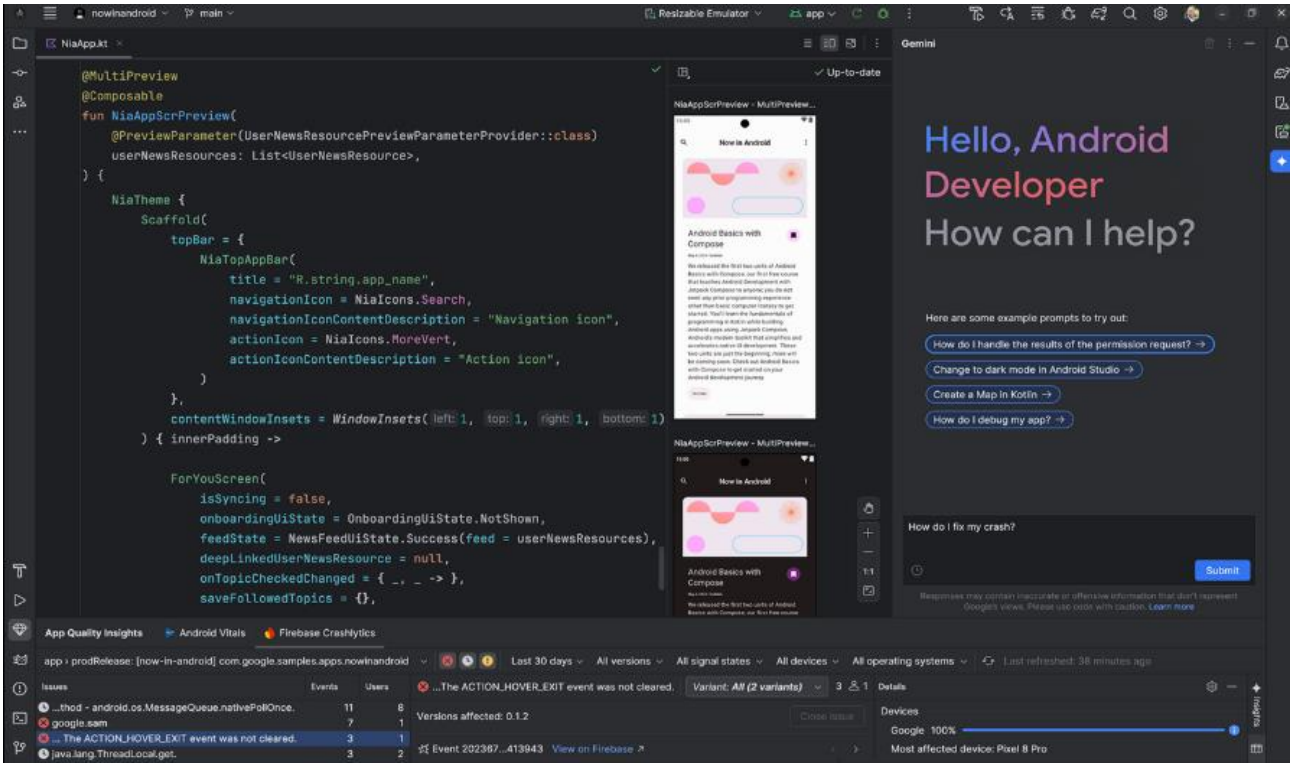

Рисунок 2.1 – Огляд IDE Android Studio.

Нижче описані причини вибору саме його як IDE для розробки інформаційної системи пошуку напарника для спільних занять спортом:

- Офіційна підтримка: Android Studio підтримується компанією Google, що гарантує стабільність, регулярні оновлення та сумісність з останніми версіями Android.
- Вбудований емулятор: дозволяє перевіряти роботу застосунку без необхідності фізичного пристрою.
- Зручна робота з UI: можливість створення макетів у режимі Drag & Drop або редагування XML-коду з попереднім переглядом.
- Інтеграція з базами даних: підтримка бібліотек Room, LiveData та ViewModel із глибокою інтеграцією в Android архітектуру.
- Велика спільнота розробників: велика кількість готових рішень, документації та прикладів.
- Вбудована підтримка контролю версій: можливість працювати з Git безпосередньо з середовища розробки.
- Плагіни та розширення: підтримка різноманітних розширень для автодоповнення, рефакторингу, аналізу коду та оптимізації продуктивності.

Android Studio також дозволяє проводити профілювання продуктивності, налагодження додатку в режимі реального часу, а також використання інструментів аналізу пам’яті та енергоспоживання. Це особливо важливо при розробці мобільних застосунків, які мають працювати стабільно на різних пристроях і версіях ОС [14].

Основною мовою реалізації застосунку обрано Kotlin. Це сучасна мова програмування, офіційно рекомендована Google для розробки застосунків під Android. Kotlin поєднує в собі зручний синтаксис, гнучкість та високу безпеку, що робить її придатною як для початкової розробки, так і для масштабування проєктів у майбутньому [18, 26].

Серед переваг мови програмування Kotlin можна виділити:

- Компактність і виразність коду. Синтаксис Kotlin дозволяє уникнути зайвого дублювання, а також зменшує обсяг коду без втрати читабельності. Це пришвидшує розробку і зменшує кількість помилок,

особливо в частині обробки запитів, фільтрації та взаємодії з базою даних.

- Сумісність із Java та Android SDK. Це дозволяє безперешкодно використовувати бібліотеки, необхідні для інтеграції з такими компонентами, як Google Maps або Room, без додаткових обмежень.
- Безпечна робота з null-значеннями. Kotlin запроваджує суворе розділення між типами, що допускають “null”, та тими, що не допускають. Це зменшує вірогідність виникнення критичних помилок під час роботи програми
- Підтримка корутин. Для реалізації операцій, пов’язаних із базою даних, фільтрацією користувачів та завантаженням мапи, застосовано асинхронну обробку даних. Корутини Kotlin дозволяють виконувати ці задачі ефективно, не блокуючи основний потік.
- Інтеграція з архітектурними компонентами. Kotlin ідеально поєднується з ViewModel, LiveData, а також з підходом MVVM, який використано в структурі проєкту.

Тому використання Kotlin буде хорошим рішенням для розробки інформаційної системи. Ця мова програмування має підтримку в Android Studio, котра надає купу корисних інструментів та плагінів з підтримкою Kotlin, що може знадобитися в майбутній розробці інформаційної системи.

Оскільки застосунок є автономним і має зберігати дані локально на пристрої, важливою складовою стане впровадження ефективного рішення для бази даних. У проєкті планується використати SQLite у поєднанні з бібліотекою Room, що дозволить реалізувати повноцінну реляційну структуру без необхідності у зовнішньому сервері [19, 27].

Переваги такого підходу:

1. Локальне зберігання профілів та запитів. Усі дані — облікові записи користувачів, вподобання (теги), історія запрошень, координати спортивних об’єктів — зберігаються локально, що забезпечує автономність

та високу швидкість доступу без необхідності постійного з’єднання з мережею.

2. Бібліотека Room як надбудова над SQLite. Room спрощує написання SQL-запитів завдяки використанню анотацій та автоматично генерує код для взаємодії з базою. Це забезпечує:
   – типобезпеку SQL-запитів під час компіляції;
   – чітку структуру: Entity (таблиці), DAO (запити), ViewModel (зв’язок із UI);
   – гнучкість і розширюваність архітектури.
3. Оптимальна структура бази даних. У межах проєкту буде реалізовано такі основні таблиці:
   – Users – для зберігання особистих даних користувача (ім’я, контакти, вид спорту, рівень);
   – Tags – окремо винесені категорії спортивних уподобань;
   – Invitations – запити на спільні тренування між користувачами;
   – Locations – координати спортивних об'єктів (зали, парки, майданчики);
   – (опціонально) Preferences – для збереження параметрів пошуку та фільтра.
4. Масштабованість. Room дозволяє ефективно працювати навіть із великим обсягом даних завдяки підтримці індексів, зовнішніх ключів і фільтрації. У перспективі база може бути частково або повністю винесена у хмару (наприклад, Firebase), зберігаючи при цьому поточну архітектуру застосунку.
5. Безпека та контроль над даними. Завдяки локальному зберіганню забезпечено більший контроль над конфіденційною інформацією користувачів. У разі потреби легко реалізується шифрування бази, обмеження доступу до таблиць або механізм резервного копіювання.

Однією з ключових функцій мобільного застосунку буде відображення спортивних локацій на інтерактивній мапі. Це дозволить користувачам бачити найближчі спортивні зали, майданчики, парки та отримувати інформацію про те,

які користувачі також зацікавлені в тренуваннях у тих самих локаціях. Для реалізації цієї функції буде використано Google Maps SDK, який надає гнучкі засоби для інтеграції карт у мобільні застосунки на Android-платформі [20].

Серед основних можливостей, які надає Google Maps API і які буде використано в розробці мобільного додатку:

- відображення мапи з точним масштабуванням та навігацією;
- встановлення маркерів (Markers) у точках, що відповідають спортивним локаціям;
- виведення інформаційних вікон (InfoWindow) з короткими відомостями про місце;
- виявлення взаємозв'язку між місцем та користувачами, у яких збігаються спортивні хештеги або уподобання;
- відкриття діалогового вікна з користувачами, що займаються спортом на цій локації, з можливістю перегляду профілю та надсилання запрошення.

Застосування Google Maps дозволяє реалізувати візуальну взаємодію, яка є більш зрозумілою та зручною для користувача в порівнянні з текстовими списками. Такий підхід сприяє кращому орієнтуванню в реальному середовищі, підвищує рівень залученості та дозволяє швидше приймати рішення про вибір партнера для тренування [20].

Google Maps SDK легко інтегрується із застосунком, створеним у відповідному середовищі розробки, та має гарну сумісність з архітектурними компонентами, які будуть використані в додатку. Також API підтримує взаємодію з GPS-модулем пристрою, що дозволяє точно визначати поточну позицію користувача для персоналізованого пошуку.

Таким чином визначені ключові елементи програмного забезпечення, які дозволять виконати поставлені умови до даної інформаційної системи.

2.3 Вибір апаратного забезпечення

При майбутній розробці інформаційної системи вона являтиме собою локальне клієнтське рішення, яке встановлюється безпосередньо на пристрій користувача. Уся логіка взаємодії, зберігання та обробки даних буде реалізована на стороні клієнта без необхідності постійного підключення до віддалених серверів або хмарних сервісів. Такий підхід забезпечує високу автономність роботи, стабільність доступу до основного функціоналу та збереження конфіденційності персональних даних.

Основні модулі системи (пошук, перегляд профілів, надсилання запрошень, управління вподобаннями, взаємодія з картою) виконуються безпосередньо на пристрої. Але не інтеграція з картографічним сервісом, для якої потрібне інтернет-з'єднання, оскільки Google Maps завантажує мапи з віддалених серверів [20].

Застосунок призначений для роботи на смартфонах і планшетах під управлінням операційної системи Android, починаючи з версії 7.0 (Nougat) і вище. Завдяки використанню лише стандартних компонентів Android SDK, застосунок не вимагає високопродуктивного обладнання і може бути успішно встановлений на пристрої середнього класу.

Підтримувані пристрої:

- смартфони з Android OS;
- планшети з Android OS;
- емулятори Android Virtual Device (AVD) — для тестування на етапі розробки.

На етапі розробці дана інформаційна система працюватиме повністю автоматично. Можливе горизонтальне масштабування із підключенням хмарних платформ. Це надає можливість додавати нові функції, такі як збереження даних у хмарі, синхронізація профілів між пристроями, push-сповіщення, обмін повідомленнями та інші. Для цього можна буде використати одне із

найзручніших рішень для інтеграції з мобільними застосунками: Firebase або Google Cloud Platform.

Firebase чудово підходить для мобільних пристроїв, бо інтегрується з Android Studio. Ця хмарна платформа не потребує складного конфігурування серверної частини та надає гнучке масштабування залежно від кількості користувачів в системі. Серед інструментів цієї платформи можна виділити Firebase Authentication (модуль авторизації з підтримкою Google), Cloud Firestore (хмарна NoSQL-база, яка дозволяє зберігати й синхронізувати дані в режимі реального часу) і Firebase Hosting (для надсилання push-сповіщень між користувачами) [21].

Google Cloud Platform допомагає у створенні окремих мікросервісів для обробки запитів, обміну повідомленнями, управління даними або аналітики. Серед його переваг можна виділити створення API через Cloud Functions або App Engine, автоматичне масштабування та балансування навантаження, аналіз великої кількості подій через BigQuery [22].

В залежності від потреб під час подальшої розробки інформаційної системи можна обрати одну з двох хмарних платформ.

Щодо безпеки та конфіденційності, то в базовій локальній версії інформаційної системи конфіденційність гарантується завдяки відсутності сторонніх серверних з’єднань – усі дані зберігаються на пристрої користувача. У разі масштабування основними принципами забезпечення безпеки будуть обмежений доступ до файлів додатка в межах системи Android, використання протоколів HTTPS та SSL/TLS при з’єднані з хмарними сервісами та правила доступу до Firestore, що передбачають гнучке налаштування прав доступу до даних, зокрема, для окремих користувачів [23-24].

2.4 Висновки

В цьому розділі було сформовано системні вимоги, розглянуто інформаційні технології для розробки мобільних додатків. Тож для розробки

обрано мову програмування Kotlin, Android Studio як IDE. Для обробки і збереження даних було обрано SQLite у поєднанні з бібліотекою Room. Саме така IT-інфраструктура забезпечить коректну роботу інформаційної системи.

3 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Ключові компоненти архітектури

Архітектура мобільного застосунку, що реалізує пошук спортивних напарників, побудована відповідно до принципів сучасної клієнтської архітектури та структурована на основі моделі MVVM (Model–View–ViewModel), яка є рекомендованим підходом при створенні Android-додатків [25].

Обрана архітектура забезпечує чітке розмежування обов'язків між компонентами, що значно підвищує масштабованість, тестованість та зручність підтримки системи. Увесь функціонал реалізовано на клієнтському боці — додаток встановлюється на пристрій користувача, і всі основні операції (обробка даних, відображення, логіка пошуку та фільтрації, збереження інформації) виконуються локально, без використання віддаленого сервера.

Система умовно поділена на три рівні:

1. Рівень представлення (View)

Цей рівень реалізований за допомогою Activity та Fragment-компонентів у поєднанні з XML-макетами. Його основне завдання — забезпечення взаємодії користувача з додатком: введення параметрів пошуку, перегляд результатів, перегляд профілю потенційного напарника, використання фільтрів тощо.

Компоненти рівня View не містять бізнес-логіки, натомість реагують на дії користувача та відображають зміни, отримані з ViewModel.

2. Рівень бізнес-логіки (ViewModel)

Цей рівень слугує посередником між інтерфейсом користувача та рівнем даних. Він обробляє введення з View, ініціює взаємодію з базою даних через DAO, трансформує дані для відображення та керує станами екранів.

ViewModel зберігає дані навіть у випадку зміни конфігурації (наприклад, повороту екрана), що покращує UX. Для обміну даними використовуються

LiveData, MutableLiveData або StateFlow, які дозволяють реактивно оновлювати інтерфейс.

3. Рівень доступу до даних (Model + Repository)

На цьому рівні зосереджена логіка зберігання, отримання та обробки інформації. Дані зберігаються у локальній реляційній базі SQLite, з якою взаємодія відбувається через бібліотеку Room. Room надає функціональність ORM-функціональність (Object-Relational Mapping), що дає змогу працювати з SQL через анотації та автоматично генерує код запитів [25].

Кожна таблиця в базі має відповідну сутність (Entity), DAO-інтерфейс і клас репозиторію, що забезпечує розмежування обов'язків і дає змогу централізовано обробляти всі запити до БД. Розмежування між ViewModel та DAO реалізоване саме через Repository, який виступає як єдина точка доступу до даних і дозволяє підключати інші джерела в майбутньому (наприклад, REST API).

У структурі бази даних виділено кілька основних сутностей:

- UserEntity – містить інформацію про користувача (ім'я, опис, вік, стать, рівень фізичної підготовки, унікальний ID, активні теги);
- TagEntity – перелік спортивних вподобань користувачів, пов'язаний через зовнішній ключ;
- InvitationEntity – зберігає історію запрошень, включаючи відправника, одержувача, статус (активне/скасоване/прийняте);
- LocationEntity – координати доступних локацій (зали, майданчики, парки), які згодом використовуються у фільтрації за місцем проведення тренувань.

Для взаємозв'язків між таблицями використовуються зовнішні ключі, що забезпечує логічну цілісність даних. Індекси в таблицях дозволяють оптимізувати вибірку при фільтрації (наприклад, за хештегами або місцезнаходженням).

На рисунку 3.1 зображено структуру таблиць бази даних інформаційної системи пошуку напарника для спільних занять спортом.

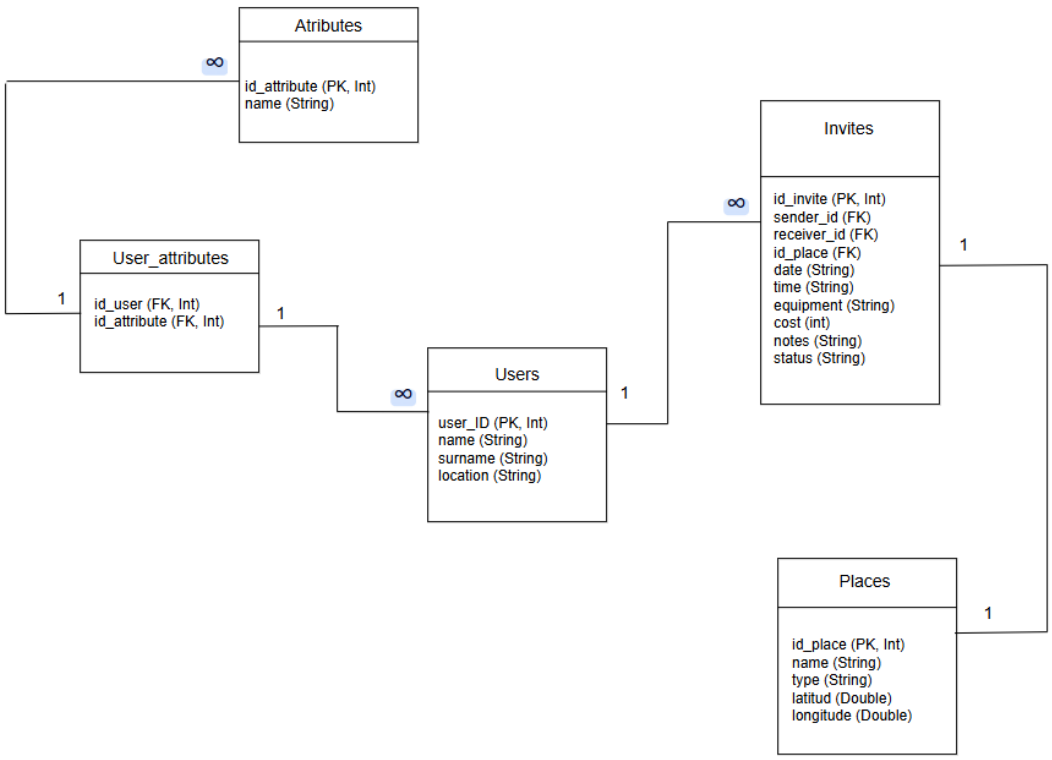

Рисунок 3.1 – Структура таблиць бази даних.

Дана база даних містить 5 таблиць: користувачі (Users), атрибути (Attributes), атрибути користувачів (User_attributes), запрошення (Invites) та локація (Places).

Короткий опис до кожної таблиці:

1. Users – таблиця, яка зберігає дані про користувача
2. Attributes – таблиця, яка зберігає дані про види спорту та теги
3. User_attributes – таблиця, яка є зв‘язковою для попередніх двох таблиць. Вона зберігає дані про певного користувача та його види спорту і теги.
4. Invites – таблиця, яка зберігає дані запрошень
5. Places – таблиця, яка зберігає дані про спортивні місця (парки, спортивні зали і майданчики)

Саме така структура бази даних забезпечує просту, але надійну роботу даної інформаційної системи.

3.2 UML-діаграми

Для кращого розуміння архітектури додатку для пошуку партнера для спільних занять спортом та взаємодії між її компонентами, було побудовано низку UML-діаграм. Вони дозволяють наочно представити структуру додатку, логіки пошуку партнера та надсилання запрошення для спільного тренування.

На рисунку 3.2 представлено діаграму випадків використання, яка ілюструє взаємодію основних акторів із системою: користувач і адміністратор.

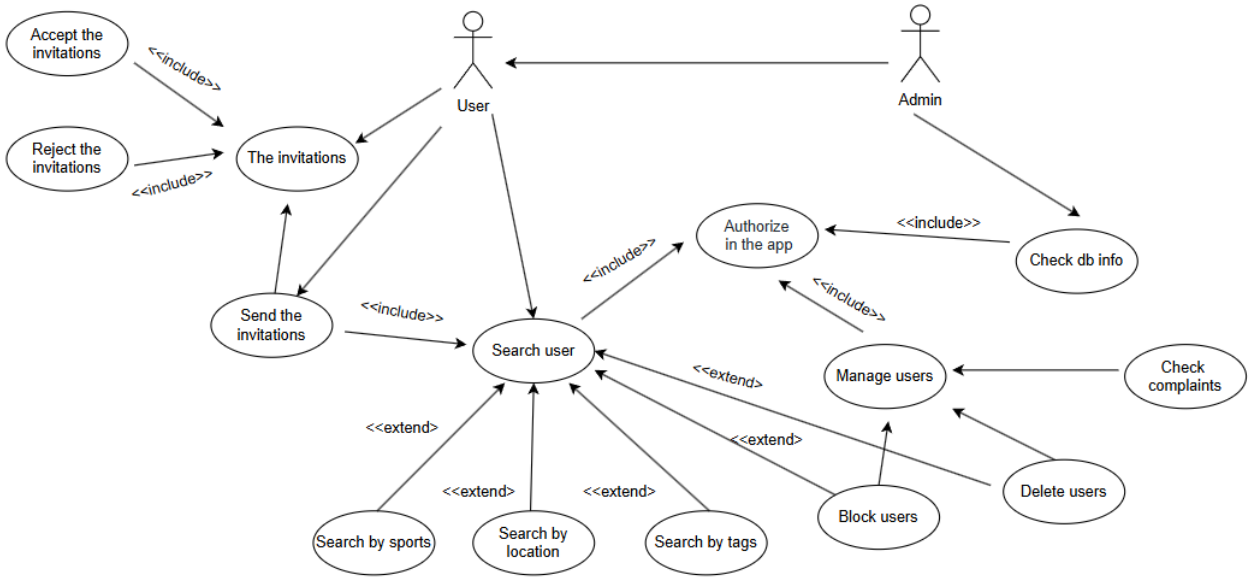

Рисунок 3.2 – Діаграма випадків використання.

Згідно з діаграмою, кожен користувач має доступ до визначеного набору функцій:

Адміністратор:

- Перевіряти інформацію в базі даних
- Перевіряти лист скарг, який фіксує повідомлення про порушення користувачами правил спільноти додатку
- Блокувати користувачів, які отримали багато скарг
- Видаляти користувачі

Користувач:

- Використовувати внутрішньо-пошукову систему додатку
- Надсилати іншим користувачам запрошення на проведення спільних занять спортом
- Переглядати профілі інших користувачів

Дана діаграма дозволяє побачити, як саме розмежовується функціонал у системі залежно від ролі користувача, що забезпечує гнучкість та безпеку в її роботі.

На рисунку 3.3 представлено діаграму послідовності, що демонструє процес пошуку користувачем підходящих партнерів для спільного заняття спортом.

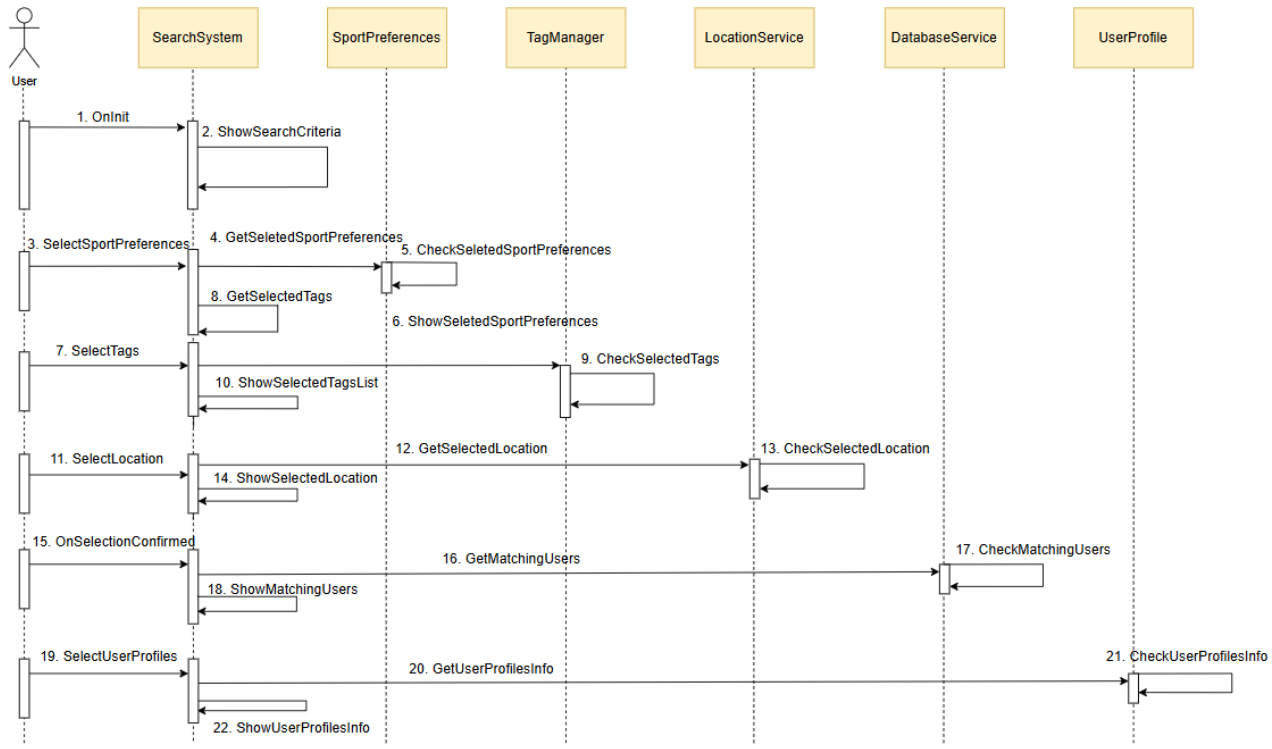

Рисунок 3.3 – Діаграма послідовності: «пошук користувачем підходящих партнерів для спільного заняття спортом».

Учасники процесу (об’єкти):

- Користувач – ініціює пошук партнера.

- SearchSystem – виконує пошукову функцію, відправляє та отримує параметри пошуку.
- SportPreferences – надсилає запит та повертає інформацію по параметру “Види спорту”.
- TagManager – надсилає запит та повертає інформацію по параметру “Теги”.
- LocationService – надсилає запит та повертає інформацію по параметру “Місцезнаходження”.
- DatabaseServise – містить інформацію про інших користувачів, повертає дані по користувачах, дані яких співпадають із запитами пошуку.
- UserProfile – повертає інформацію по запиту по даних профілю користувачів.

Послідовність дій:

1. Користувач ініціює пошук (заходить у пошуковий розділ додатку).
2. Користувач відкриває параметри пошуку.
3. Пошукова система відображає параметри пошуку.
4. Користувач обирає параметр “Види спорту”.
5. Пошукова система звертається до відповідного модуля і той повертає підтвердження вибору.
6. Користувач обирає параметр “ Теги ”.
7. Пошукова система звертається до відповідного модуля і той повертає підтвердження вибору.
8. Користувач обирає параметр “ Місцезнаходження”.
9. Пошукова система звертається до відповідного модуля і той повертає підтвердження вибору.
10. Після вибору всіх параметрів пошуку користувач відправляє запит на відображення результатів пошуку.
11. Пошукова система відправляє запит до бази даних.
12. База даних повертає результат у вигляді переліку профілів користувачів, дані яких співпали з даними запиту.

13. Користувач натискає на будь-який знайдених профіль.

14. Пошукова система звертається до відповідного модуля і той повертає інформацію про користувача, за яким був здійснений запит.

Ця діаграма дозволяє простежити усі кроки роботи пошукової системи — від моменту початку пошукового процесу до його кінця. Завдяки цьому можна побачити логіку та послідовність виконання пошукової системи додатку. Також така схема передбачає швидке виявлення та усунення помилок при роботі пошукової системи.

На рисунку 3.4 показано процес блокування користувача, на якого прийшло багато скарг, адміністратором.

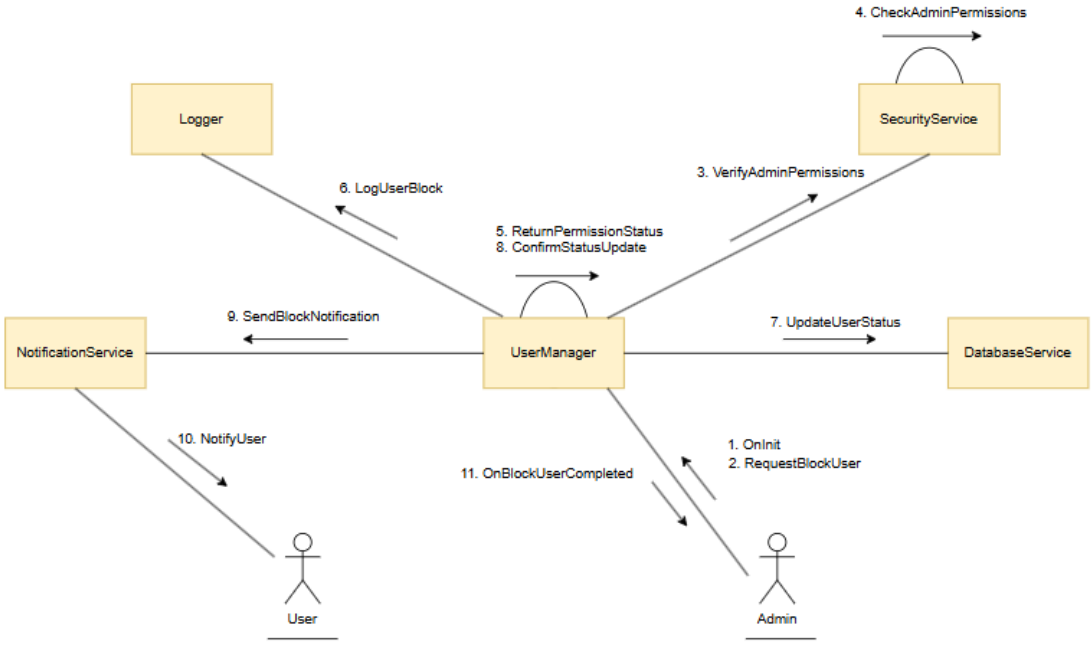

Рисунок 3.4 – Діаграма кооперації: “Блокування користувача адміністратором”.

На цій діаграмі зображено процес надання дозволу адміністратором системі здійснити блокування користувача. Після успішного блокування, заблокований користувач ортимує повідомлення про причини блокування.

Основні учасники процесу:

- Admin – адміністратор, який ініціює блокування користувача.
- User – користувач, який в процесі буде заблокований і оповіщений про це.

- UserManager – модуль, який відповідає за керування дій, які здійснюються над акаунтами користувачів.
- SecurityService – модуль, який відповідає за надання дозволу на дії, пов’язані із зміною/видаленням інформації користувачів.
- DatabaseServise – модуль, який відповідає за збереження даних і їх повернення на запити.
- Logger – допоміжний модуль, який відповідає за керування дій, які здійснюються над акаунтами користувачів.
- NonificationService – модуль, якій відповідає за оповіщення користувачів.

Послідовність взаємодії:

1. Адміністратор ініціює процес блокування користувача.
2. Адміністратор надсилає запит на блокування користувача в систему.
3. Йде перевірка права дозволу на блокування користувача адміністратором.
4. Після успішної перевірки йде запит на саме блокування користувача всередині логера.
5. Оновлюється статус користувача (на “заблокований”) всередині бази даних.
6. Завершення процесу блокування всередині системи.
7. Відправляється запит на модуль оповіщення для відправлення повідомлення користувачу про блокування його акаунту.
8. Повідомлення про блокування надсилається користувачу, якого заблокували.

Ця діаграма ілюструє, наскільки структуровано реалізовано процес блокування користувача в системі. Завдяки послідовній перевірці усіх запитів на блокування, система забезпечує цілісність, безпеку та зручність цього процесу всередині даної системи.

На рисунку 3.5 зображено діаграму станів, яка представляє процес надсилання і оброблення запрошення всередині системи додатку.

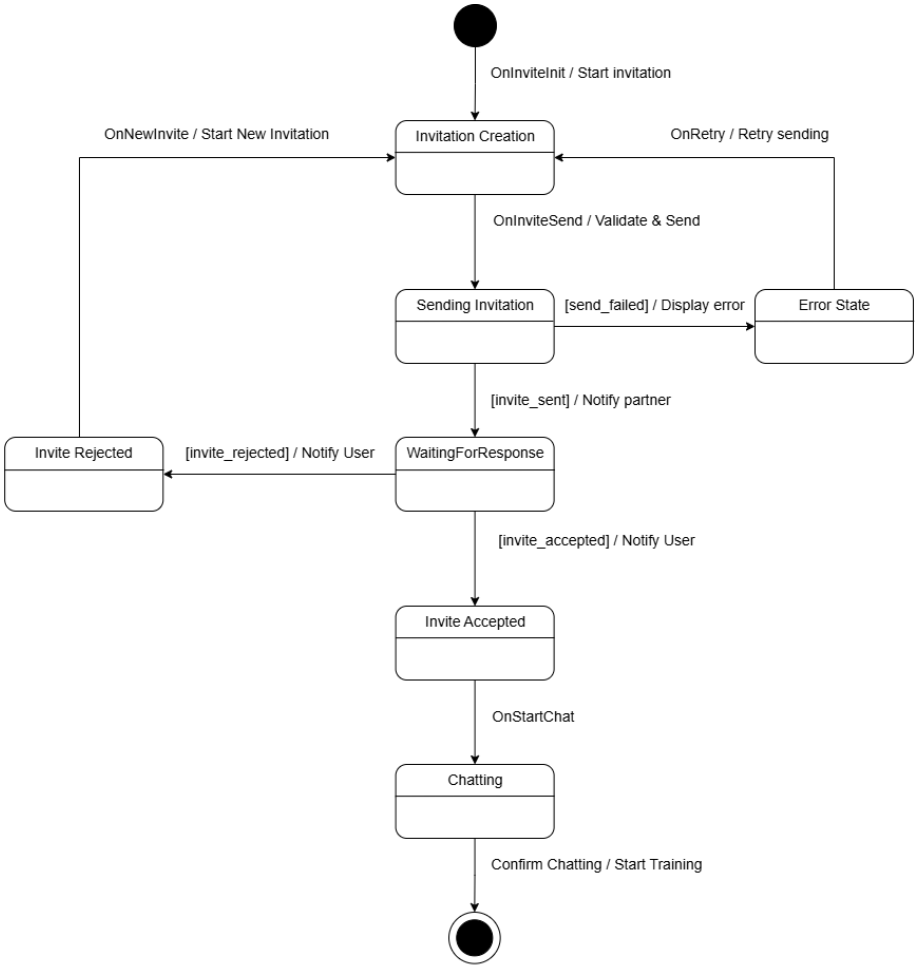

Рисунок 3.5 – Діаграма станів для процесу надсилання і обробки запрошень (Invitation Process Diagram).

На цій діаграмі відображено всі стани системи під час виконання надсилання запрошення на спільне заняття спортом користувачем.

Початок роботи: ініціалізація запрошення, початок запрошення (OnInviteInit / Start invitation).

Послідовність роботи: стан створення запрошення (Invitation Creation) – стан відправлення запрошення (Invitation):

- При не успішному відправлені стан (Error State), дія повторне створення і відпрвлення запрошення (OnRetry / Retry sending).
- При успішному відправлені стан (WaitingForResponse)

Перебуваючи в стані (WaitingForResponse) очікуємо один з двох результатів запрошення:

- При не успішному (invite_rejected) стан (Invite Rejected), дія початок створення нового запрошення (Start New Invitation), повернення в стан створення запрошення (Invitation Creation).
- При успішному (invite_accepyed), дія сповіщення користувача про прийняте запрошення (Notify User), перезід в стан прийняття запрошення (Invite Accepted).

З стану (Invite Accepted)), дія початок спілкування (OnStartChat) – перехід в стан спілкування (Chatting) – кінець роботи, дія (Confirm Chatting / Start Training).

Ця діаграма наочно демонструє всі стани системи під час виконання надсилання запрошення на спільне заняття спортом користувачем. Завдяки ній зрозуміло логіку роботи станів системи під час виконання надсилання запрошення.

3.3 Висновки

В цьому розділі було описано вибір архітектури інформаційної системи пошуку напарника для спільних занять спортом. Розглянуто основні UML-діаграми взаємодії компонентів системи. В результаті стала зрозуміла логіка архітектури інформаційної системи. Завдяки UML-діаграмам видно як саме працюватимуть основні функції інформаційної системи (пошук напарника, відправка запрошення, адміністрування інформаційної системи та інші).

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ НАПАРНИКА ДЛЯ СПІЛЬНИХ ЗАНЯТЬ СПОРТОМ

4.1 Розробка функції пошуку напарника

Основним функціоналом даної системи є система пошуку напарника, надсилання і отримання запрошень на спільні тренування. Спершу було розроблене вікно пошуку і перегляду можливих напарників. Результати пошуку відображаються у вигляді клікабельних карток, натиснувши на які можна переглянути відомості про можливого напарника. Можливі напарники представлені у вигляді списку з елементами (item) (рис.4.1).

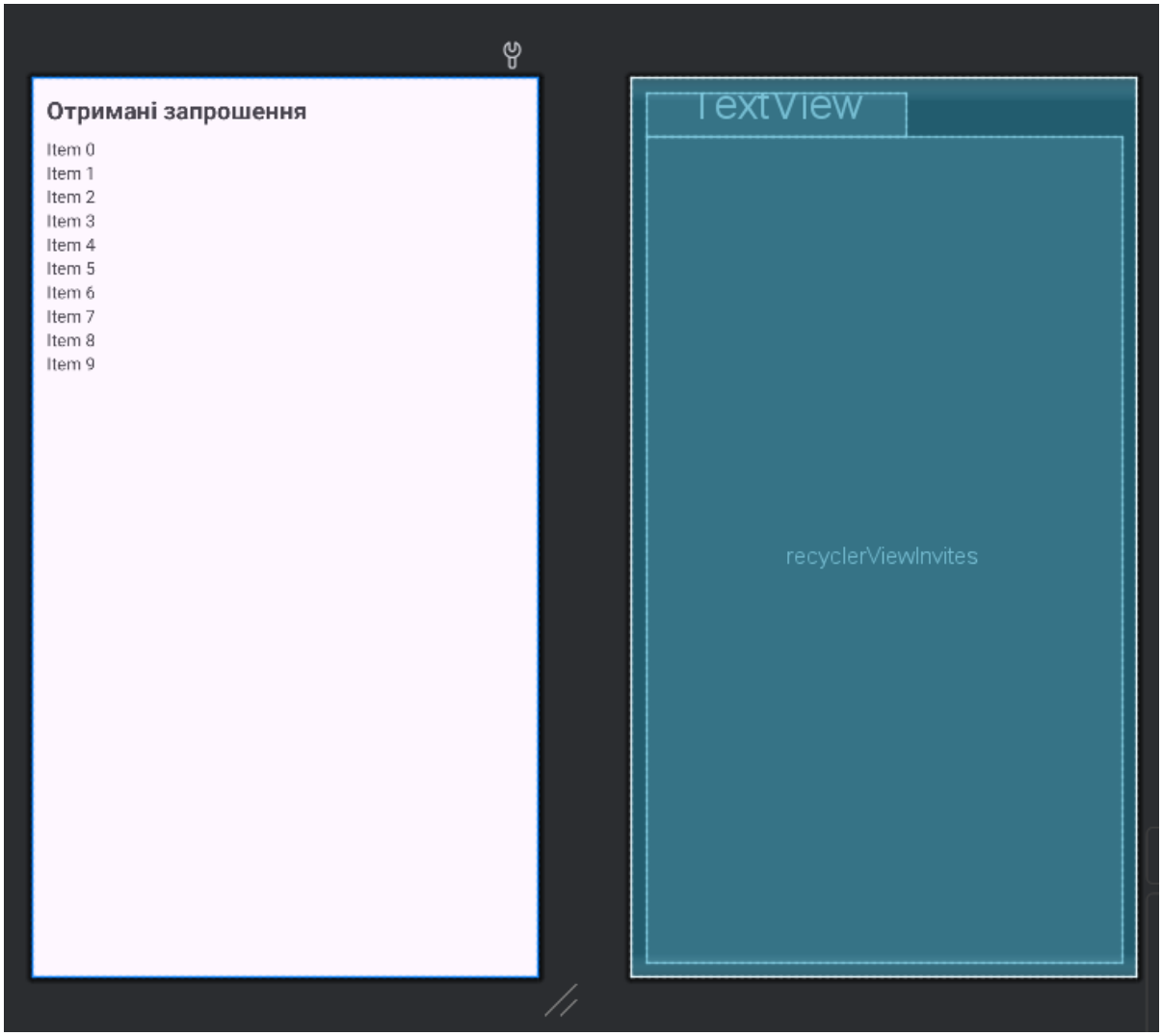

Рисунок 4.1 – Макет списку напарників.

Сама структура списку з елементами є зручним та ефективним способом перегляду результатів пошуку. Також список можна скролити донизу, що покращує візуальне сприйняття. На рисунку 4.2 зображено вигляд сторінки при запуску програми.

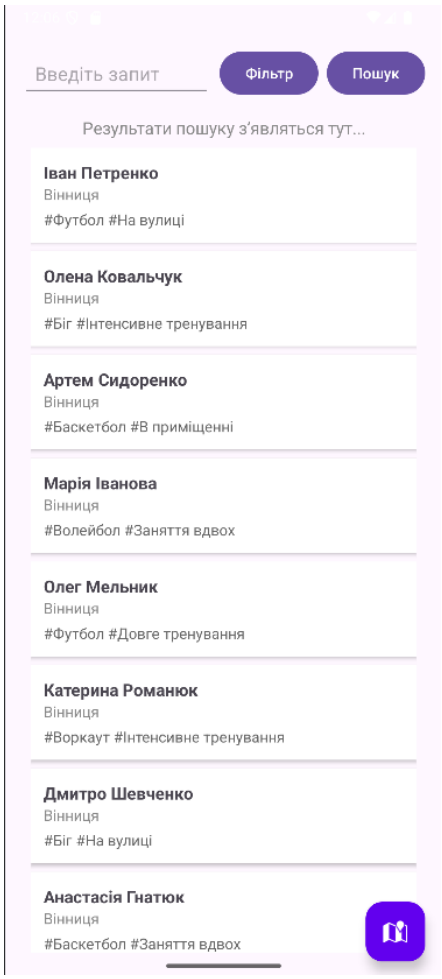

Рисунок 4.2 – Вигляд вікна пошуку напарників.

У верхній частині екрану знаходяться елементи пошукової системи. Користувач додатку може здійснити пошук за:

- ключовими словами;
- фільтрами (види спорту, теги)
- картою (іконка в нижній частині вікна)

Код для обробки введених ключових слів зображено на рисунку 4.3.

```
val editTextSearch = findViewById<EditText>(R.id.editTextSearch)
findViewById<Button>(R.id.buttonSearch).setOnClickListener {
    val query = editTextSearch.text.toString()
    if (query.isNotEmpty()) {
        searchViewModel.searchUsers(query)
    } else {
        Toast.makeText( context: this,  text: "Введіть запит для пошуку", Toast.LENGTH_SHORT).show()
    }
}
```

Рисунок 4.3 – Код для обробки введених ключових слів.

Логіка пошукової системи за ключовими словами побудована навколо поля текстового вводу “editTextSearch”. Користувач вводить в це поле ключове слово (наприклад вид спорту) і натискає кнопку “пошук”. Після цього дані передаються в клас SearchViewModel (рис. 4.4).

```
class SearchViewModel(private val dao: UserDao) : ViewModel() {

    private val _results = MutableLiveData<List<UserWithAttributes>>()
    val results: LiveData<List<UserWithAttributes>> = _results

    fun searchUsers(query: String) {
        viewModelScope.launch {
            _results.value = dao.searchUsersWithAttributes(query)
        }
    }

    fun filterUsers(attributeIds: List<Int>) {
        viewModelScope.launch {
            _results.value = dao.filterUsersByAttributes(attributeIds)
        }
    }

    suspend fun getAllAttributes(): List<AttributeEntity> {
        return dao.getAllAttributes()
    }

}
```

Рисунок 4.4 – Код класу SearchViewModel.

Після натискання на кнопку “Пошук” запит обробляється класом “SearchViewModel”, де відбувається зчитування ключових слів. Далі надсилається SQL-запит який відповідає за пошук збігів в базі даних (рис. 4.5).

```
    @Transaction
    @Query("""
    SELECT DISTINCT users.* FROM users
    LEFT JOIN user_attributes ON users.id_user = user_attributes.id_user
    LEFT JOIN attributes ON user_attributes.id_attribute = attributes.id_attribute
    WHERE users.name LIKE '%' || :search || '%'
       OR users.surname LIKE '%' || :search || '%'
       OR attributes.name LIKE '%' || :search || '%'
""")
    suspend fun searchUsersWithAttributes(search: String): List<UserWithAttributes>
```

Рисунок 4.5 – Код SQL-запиту.

Після зчитування введених ключових слів йде запит на базу даних через SQL-запит, який шукає в середині бази даних збіги за ключовими словами. Результатом є відображення списку із знайденими результатами (рис. 4.6).

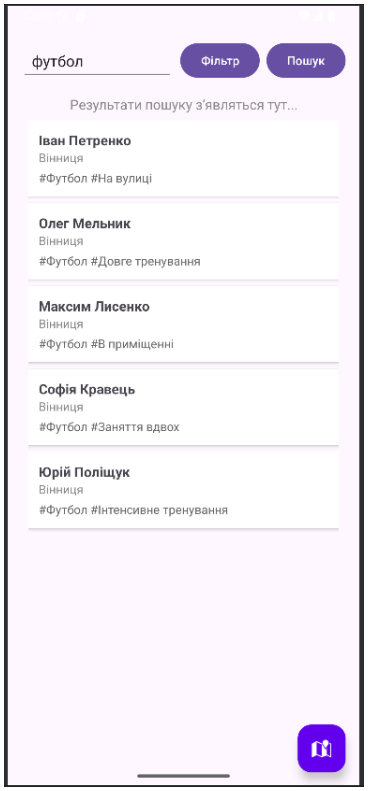

Рисунок 4.6 – Вікно пошуку за ключовим словом “футбол”

Як ми бачимо запит виконався коректно і нам відобразились всі напарники, які мають в своєму описі ключове слово “футбол”.

Логіка пошуку за тегами подібна. Користувач натискає на кнопку “Фільтр”. Відображається спливаюче вікно із запропонованими фільтрами у вигляді чекбоксів: види спорту і теги. Після вибору потрібних фільтрів, користувач натискає на кнопку “Застосувати”. Обрані фільтри зберігаються в методі “selectedFilterNames”. Назви обраних атрибутів переводяться у їх ID, які потім передаються у ViewModel. Після цього йде звернення до бази даних через функцію “filterUsers” у вигляді SQL-запиту. Після знаходження всіх збігів в базі даних – результати пошуку відображаються у вигляді списку. На рисунках 4.7-4.11 відображено всю вищеописану логіку.

```
findViewById<Button>(R.id.buttonFilter).setOnClickListener {
    showFilterDialog()
}
```

Рисунок 4.7 – Код кнопки “Фільтр”.

Як можна побачити на рисунку 4.7 зображено код кнопки “Фільтр”. Саме ця кнопка відповідає за пошук напарників за обраними критеріями. При натисканні на неї відображається спливаюче вікно із запропонованими фільтрами у вигляді чекбоксів: види спорту і теги.

```
btnApply.setOnClickListener {
    val newlySelected = attributeCheckBoxes
        .mapNotNull { (id, name) ->
            val checkBox = dialogView.findViewById<CheckBox>(id)
            if (checkBox?.isChecked == true) name else null
        }
```

Рисунок 4.8 – Код збереження обраних фільтрів.

На рисунку 4.8 зображено код збереження обраних фільтрів. Після вибору потрібних фільтрів, користувач натискає на кнопку “Застосувати”. Обрані фільтри зберігаються в методі “selectedFilterNames”.

```
    @Query("""
    SELECT * FROM users
    WHERE id_user IN (
        SELECT id_user FROM user_attributes
        WHERE id_attribute IN (
            SELECT id_attribute FROM attributes
            WHERE name IN (:tags)
        )
    )
""")
    suspend fun getUsersByTags(tags: List<String>): List<UserEntity>
```

Рисунок 4.9 – Код SQL-запиту.

На рисунку 4.9 зображено код SQL-запиту. Після збереження фільтрів йде звернення до бази даних через функцію “filterUsers” у вигляді SQL-запиту. Після знаходження всіх збігів в базі даних – результати пошуку відображаються у вигляді списку. На рисунках 4.10-4.11 зображено інтерфейс функції фільтрації.

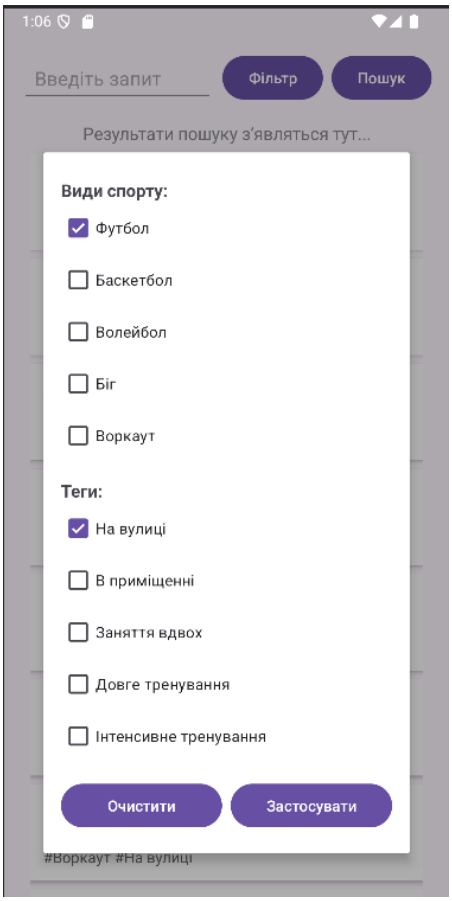

Рисунок 4.10 – Вікно з фільтрами.

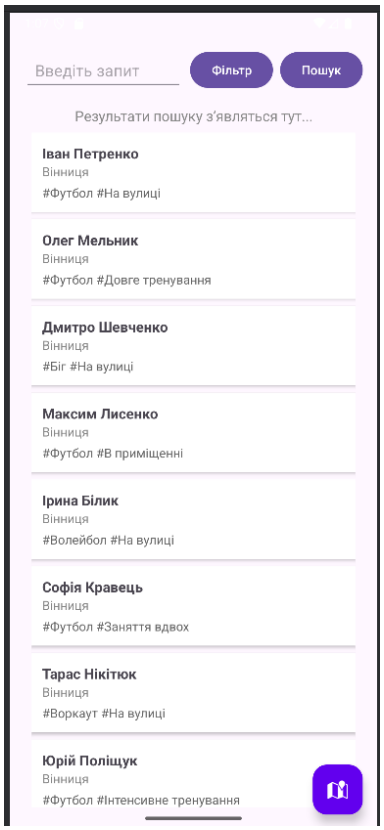

Рисунок 4.11 – Результат пошуку за тегами.

Отже функція пошуку напарника за тегами працює коректно. Додатковою можливістю, яка спрощує пошук в даному додатку є збереження обраних тегів. Тобто коли після першого пошуку користувач забажає змінити пару тегів, то йому не доведеться повністю починати спочатку обирати теги: раніше обрані чекбокси будуть відображатися до тих пір, поки користувач не зітре їх натиснувши на кнопку “Очистити”.

Пошук за картою відбувається через нанесені зарання маркери на карті, які відображають популярні місця проведення спортивних активностей (зали, спортивні майданчики, парки) у місті. На рисунках 4.12-4.13 візуалізовано логіку пошуку напарника за картою.

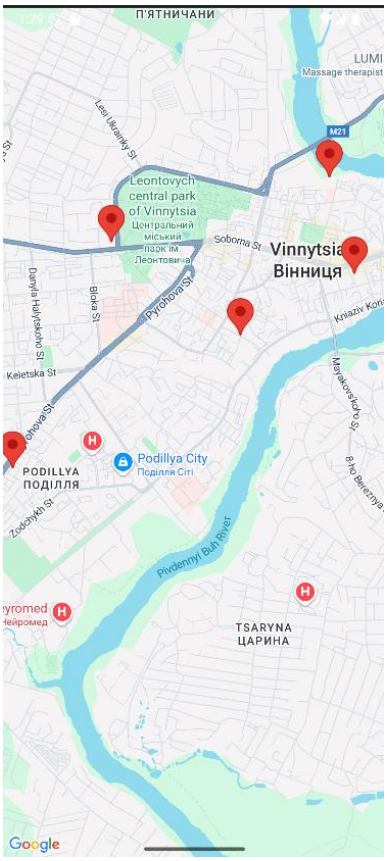

Рисунок 4.12 – Вигляд карти з маркерами.

На рисунку 4.12 зображено інтерфейс вигляду карти з маркерами. Логіка такого пошуку наступна: користувач натискає на плаваючу кнопку з картою. Його

перенаправляє на окрему сторінку з картою, де він може натиснути на один із запропоновати маркерів. Після натискання на маркер йому відображається список із можливими напарниками, чиї теги підходять для даного місця. Наприклад: якщо у певного користувача стоять теги “Футбол” і “На вулиці”, то його картка автоматично записується до відповідного місця – парк або футбольне поле (рис. 4.13).

Деякі теги можуть підходити одразу для декількох місць. Та це не заважає пошуку напарника, а навпаки додає різноманіття у виборі місця майбутнього тренування.

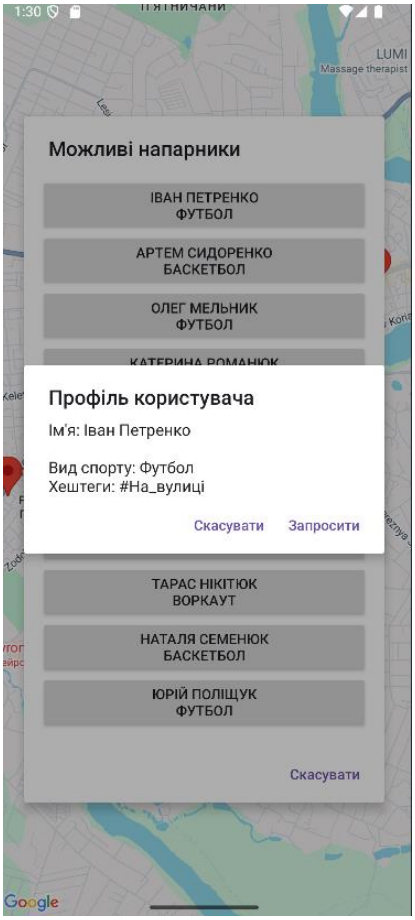

Рисунок 4.13 – Відображення списку напарників.

Отже розглянуто функціонал пошуку напарника даної інформаційної системи у вигляді мобільного додатку. Всі елементи пошуку працюють коректно.

4.2 Розроблення функції надсилання запрошень.

Після успішного виконання пошуку напарника слідує процес надсилання запрошення на спільне тренування. Для відправки запрошення на кожній картці напарника є кнопка “Запросити”. Користувач натискає на цю кнопку і йому відображається спливаюче вікно із полями запрошення, які треба заповнити. Після заповнення і надсилання запис запрошення зберігається в середині бази даних (рис. 4.14-4.18).

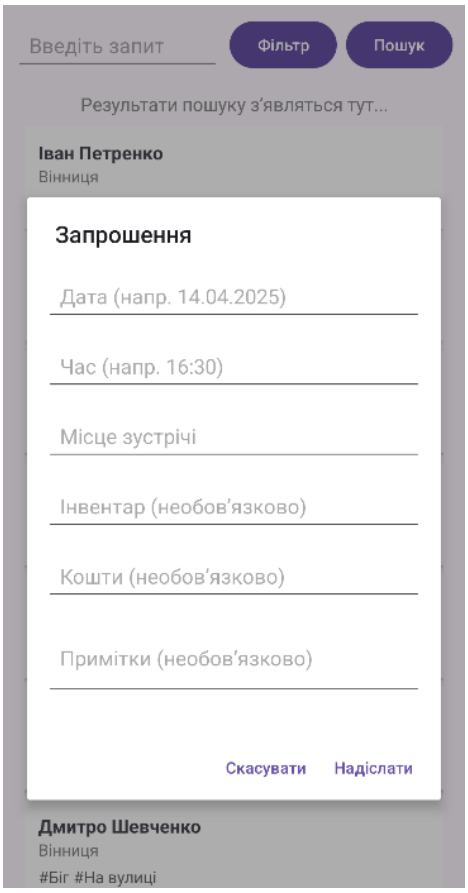

Рисунок 4.14 – Вікно запрошення.

На рисунку 4.14 зображено вікно запрошення. На даному вікні є 3 обов’язкових поля (дата, час, місце зустрічі) та 3 не обов’язкових (інвентар, кошти, примітки). Після заповнення полів, користувач натискає на кнопку “Надіслати”.

```
val senderId = 1
val receiverId = user.user.id_user

val invite = InviteEntity(
    sender_id = senderId,
    receiver_id = receiverId,
    date = date,
    time = time,
    location = location,
    equipment = editEquipment.text.toString().takeIf { it.isNotBlank() },
    cost = editCost.text.toString().takeIf { it.isNotBlank() },
    notes = editNotes.text.toString().takeIf { it.isNotBlank() },
    status = "pending"
)
```

Рисунок 4.15 – Код формування об‘єкту “InviteEntity”.

На рисунку 4.15 зображено код формування об‘єкту “InviteEntity”. Логіка надсилання запрошень: після введення інформації у поля об’єкту “InviteEntity” йде збереження цих даних всередині бази даних за допомогою методу “sendInvite”.

```
data class InviteEntity(
    @PrimaryKey(autoGenerate = true) val id_invite: Int = 0,
    val sender_id: Int,
    val receiver_id: Int,
    val date: String,
    val time: String,
    val location: String,
    val equipment: String? = null,
    val cost: String? = null,
    val notes: String? = null,
    val status: String = "pending" // "accepted", "rejected"
)
```

Рисунок 4.16 – Код таблиці “InviteEntity”.

На рисунку 4.16 зображено код таблиці “InviteEntity”. Дані зберігаються всередині таблички бази даних. Також йде збереження відправника (sender_id) і отримувача (id_user). Вигляд заповненого запрошення і його збереження в базі даних зображено далі на рисунках 4.17-4.18.

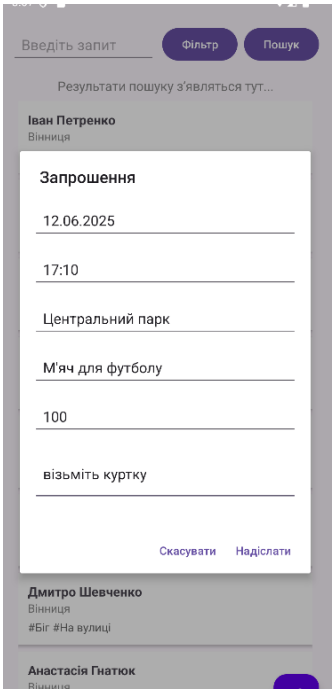

Рисунок 4.17 – Вигляд заповнених полів запрошення.

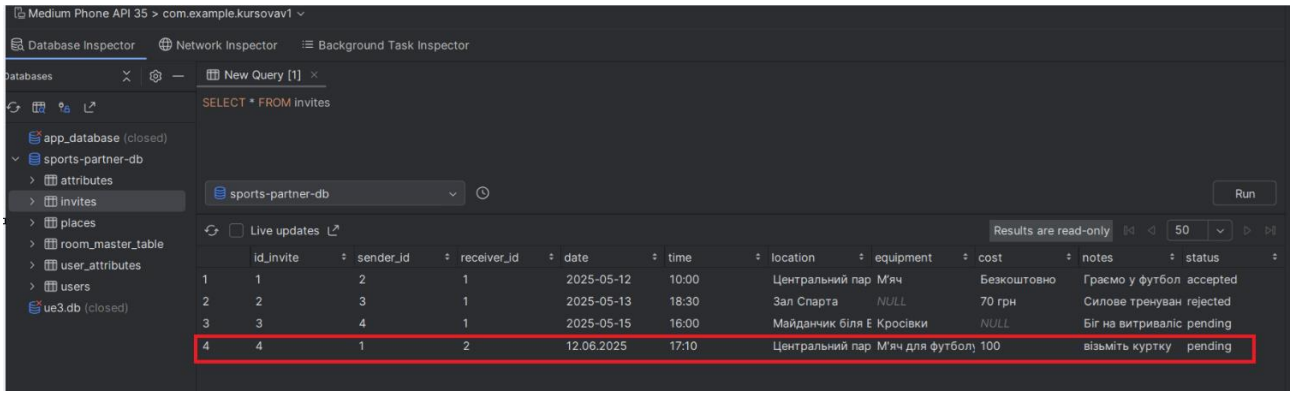

Рисунок 4.18 – Запис надісланого запрошення в бд.

Функція надсилання запрошення є дуже важливою частиною даного мобільного додатку. З цієї причини було проведене додаткове unit-тестування роботи надсилання запрошення (рис. 4.19-4.21).

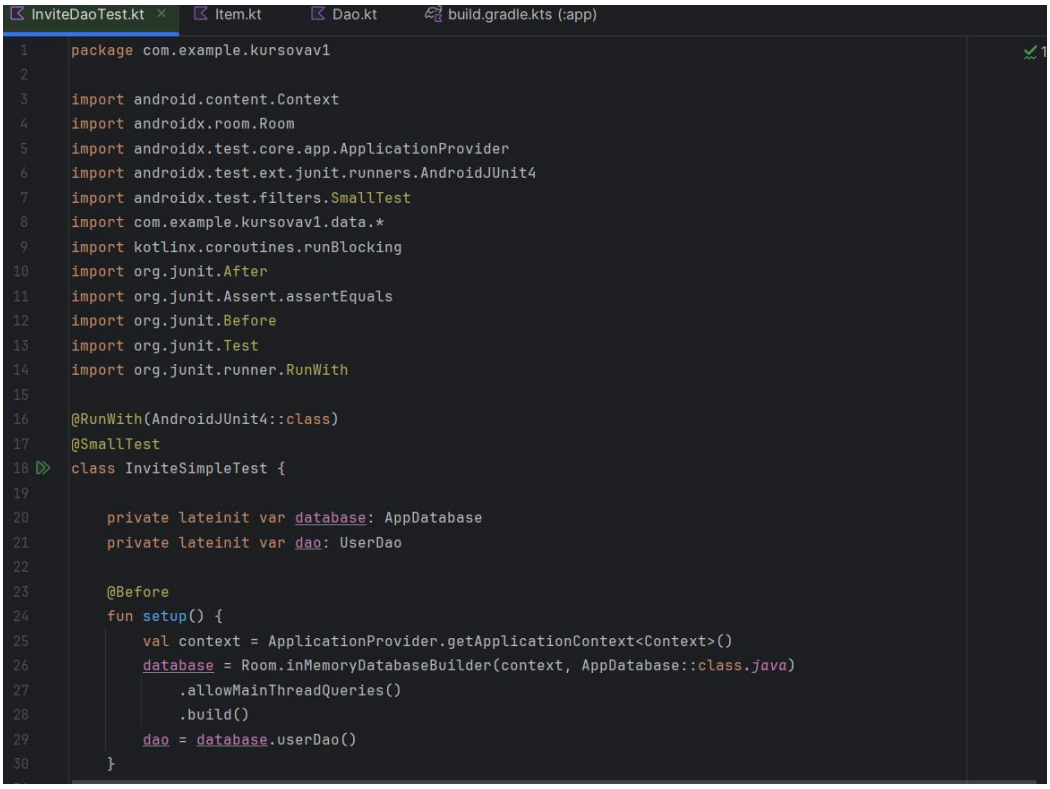

Рисунок 4.19 – Код unit-тестування.

На рисунку 4.19 зображено код unit-тестування. Завдяки ньому буде перевірено роботу функції надсилання запрошення.

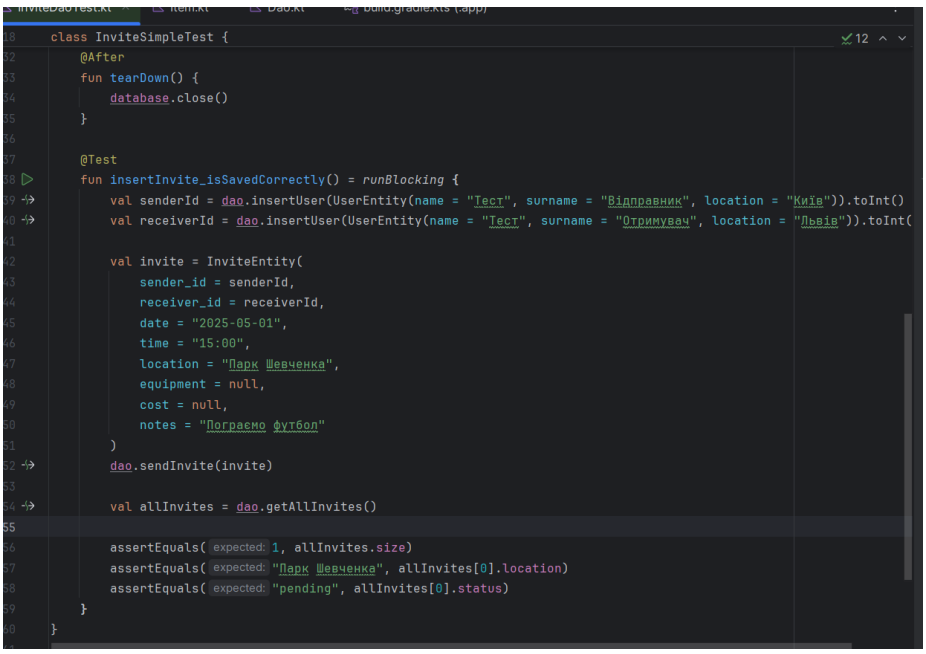

Рисунок 4.20 – Код unit-тестування.

На рисунку 4.20 зображено продовження коду unit-тестування. Завдяки ньому буде перевірено роботу функції надсилання запрошення.

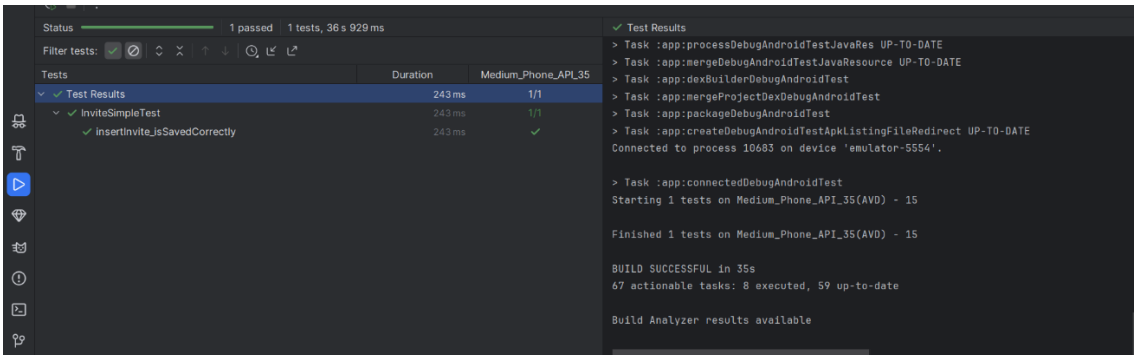

Рисунок 4.21 – Результат unit-тестування.

На рисунку 4.21 зображено результат unit-тестування, який є позитивним. Отже, за результатами модульного і unit-тестування можна стверджувати, що функція надсилання запрошень і їх запису в базі даних працює коректно. Дані зберігаються без помилок.

4.3 Розроблення функції перегляду отриманих запрошень.

Після реалізації функцій пошуку напарника і надсилання запрошень для спільного тренування залишилось реалізувати останню, але не менш важливу функцію – перегляд отриманих запрошень.

На початковому вікні в правому верхньому кутку міститься плаваюча кнопка із отриманими запрошеннями. Після натискання на неї користувач переходить до окремого вікна, де відображаються отримані запрошення у вигляді списку карток. На кожній картці запрошення відображається її статус: прийняте, відхилене або очікується. Натискаючи на карточку користувач переглядає відомості, які містяться в запрошені. Також він може переглянути відомості про відправника запрошення. Після перегляду він може обрати чи прийняти запрошення, чи відхилити його. За це відповідають кнопки “Відхилити” і “Прийняти”. Після вибору в базу даних записується оновлений статус

конкретного запрошення. На рисунках 4.22-4.23 зображено інтерфейс перегляду і дії над запрошеннями.

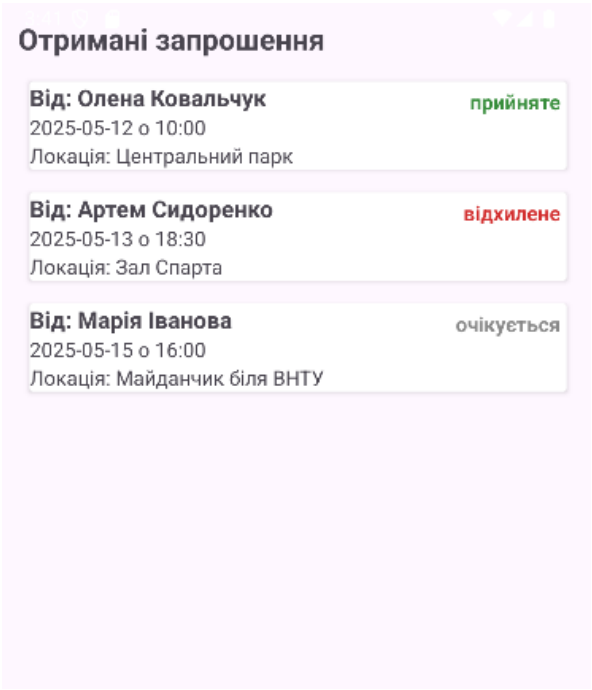

Рисунок 4.22 – Вигляд вікна отриманих запрошень із статусами.

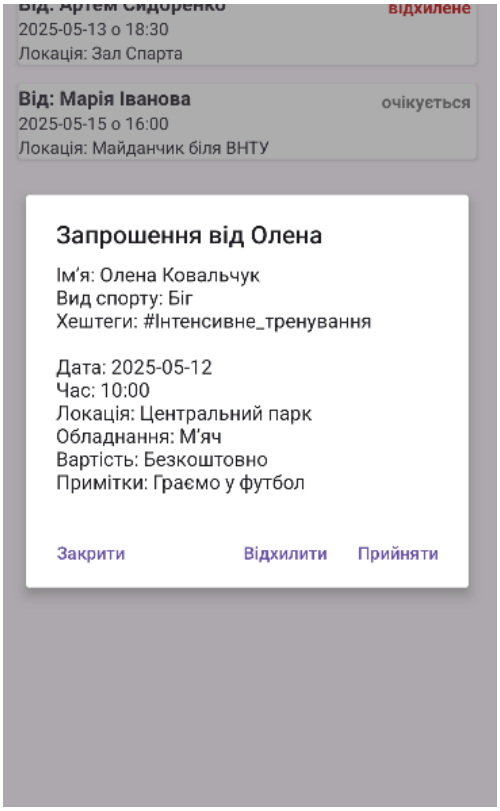

Рисунок 4.23 – Перегляд вмісту запрошення.

Отже функція перегляду запрошень і взаємодії з ними працює коректно. Всі статуси оновлюються і записуються в базу даних. Після зміни статусу картки залишаються у вікні отриманих запрошень. Це було зроблено для кращого розуміння роботи зміни статусу запрошень.

Загальну структуру мобільного додатку можна переглянути на рисунках 4.24-4.25.

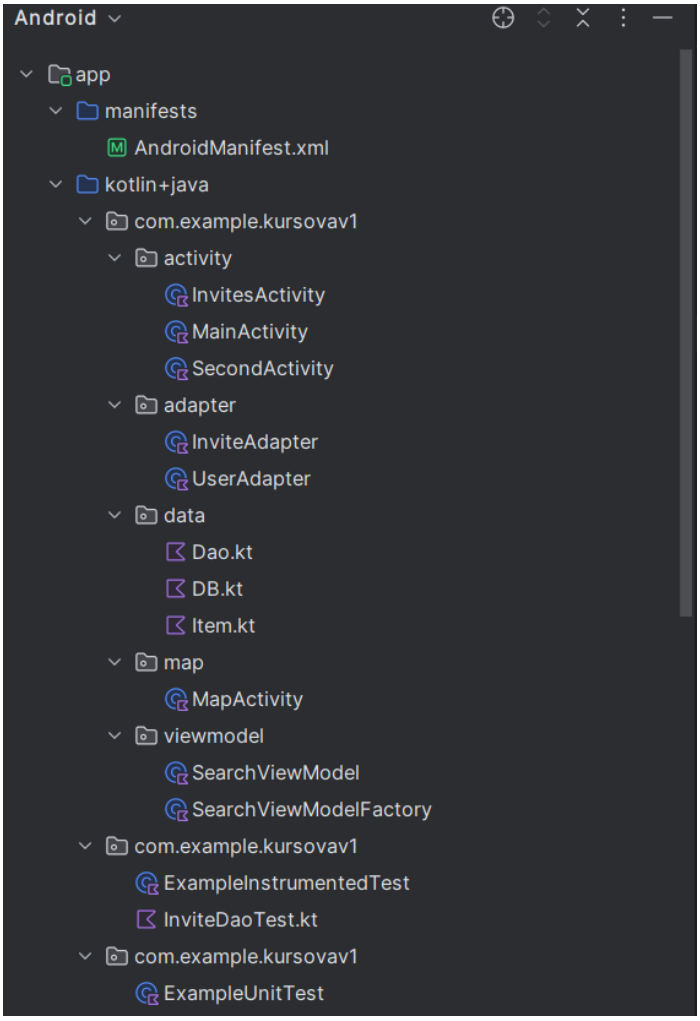

Рисунок 4.24 – Головні активності мобільного додатку.

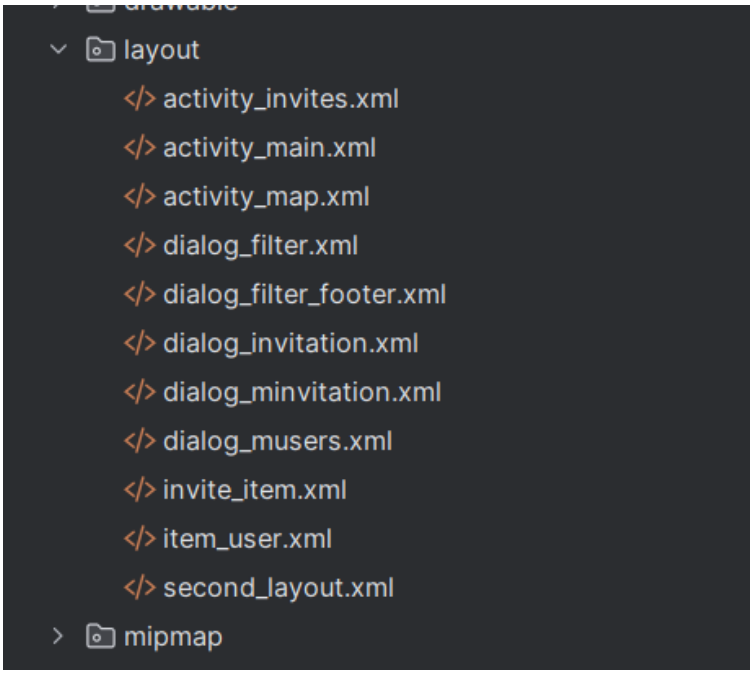

Рисунок 4.25 – Шари мобільного додатку.

4.4 Висновки

В даному розділі розглянуто програмну реалізацію інформаційної системи пошуку напарника для спільних занять спортом у вигляді мобільного додатку. В результаті розроблено основні функції додатку: пошуку напарника, надсилання і перегляду отриманих запрошень, взаємодія із отриманими запрошеннями. Завдяки розробленим функціям дана інформаційна система працює швидко та коректно: всі функції інформаційної системи були успішно протестовані модульно в емуляторі Android Studio. Всі записи коректно зберігаються в базу даних.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи була розроблена інформаційна система пошуку напарника для спільних занять спортом у вигляді мобільного застосунку. Проаналізовано предметну область, досліджено та визначено необхідний функціонал інформаційної системи. Система реалізує функціонал пошуку за хештегами, типами активності та спортивними локаціями, що підвищує ефективність взаємодії та знижує часові витрати користувача.

На відміну від багатьох існуючих рішень, завдяки поєднанню SQLite та бібліотеки Room розроблена інформаційна система забезпечує повноцінну локальну роботу без постійного підключення до Інтернету та має простий інтерфейс, адаптований під українського користувача.

Використання мови Kotlin у поєднанні з Jetpack-компонентами дозволило досягти високої стабільності, масштабованості та розширюваності додатку.

Вибір IT-інфраструктури та технологій здійснено на основі порівняння кількох варіантів за такими критеріями, як підтримка локального збереження даних, зручність інтеграції, наявність документації, спільнота підтримки та відповідність специфіці Android-розробки."

Основним обмеженням створеної системи є відсутність серверної частини. Всі операції виконуються локально на пристрої користувача. Це з одного боку забезпечує приватність даних, а з іншого обмежує можливості синхронізації, масштабування та взаємодії між користувачами в реальному часі.

Розроблена інформаційна система відповідає всім поставленим критеріям, вимогам і задачам дослідження.

Опубліковано тези доповідей на тему: «Інформаційна система пошуку напарника для спільних занять спортом» на LIV Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2025).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крижановський Є. М. Інформаційна система пошуку партнера для спільних занять спортом [Електронний ресурс] / Є. М. Крижановський, Козуб Д. В. // Тези доповідей Всеукраїнаська науково-технічна конференція, Вінниця, 20 травня 2025 р. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23794/19656 (дата звернення: 15.05.2025).

2. Гелета, Д. Д., and О. Г. Красилич. "Важливість спорту та фізичного виховання при малорухливій роботі." URL: https://openarchive.nure.ua/server/api/core/bitstreams/da4bd1b4-d99c-44aa-adaa-da82c0d362e9/content (дата звернення: 15.05.2025).

3. Галатюк, Михайло. "Розвиток спортивної культури майбутніх фахівців фізичного виховання у контексті процесів діджиталізації спорту." 2022 р. URL:

http://www.ir.dspu.edu.ua/jspui/bitstream/123456789/2405/1/351-%D0%A2%D0%B5%D0%BA%D1%81%D1%82%20%D1%81%D1%82%D0%B0%D1%82%D1%82%D1%96-880-1-10-20220831.pdf (дата звернення: 16.05.2025).

4. Дємєнтьєв, Євгеній, Володимир Шамоня, and Олена Семеніхіна. "Підготовка ІТ-фахівців до створення побільних додатків: огляд актуальних досліджень." Освіта. Інноватика. Практика 13.1 (2025): ст.7-14. URL: https://oip-journal.org/index.php/oip/article/view/494 (дата звернення: 16.05.2025).

5. Basto, P. S., & Ferreira, P. (2025). Mobile applications, physical activity, and health promotion. BMC Health Services Research, 25, Article 359. URL: https://bmchealthservres.biomedcentral.com/articles/10.1186/s12913-025-12489-z (дата звернення: 16.05.2025).

6. Verified Market Research. Health and Fitness App Market Size and Forecast. 2024 p. URL: https://www.verifiedmarketresearch.com/product/health-and-fitness-app-market/ (дата звернення: 16.05.2025).

7. Playseek. Social sports app. 2025 p. URL: https://www.playseek.me/ (дата звернення: 17.05.2025).

8. FindySport. Find sport partners near you. 2025 р. URL: https://findysport.com/ (дата звернення: 17.05.2025).

9. Fitior. Your sports social network. 2025 р. URL: https://www.fitior.com/ (дата звернення: 17.05.2025).

10. Playo. Find Your Sports Buddy & Book Venues. 2021 р. URL: https://play.google.com/store/apps/details?id=com.techjini.playo (дата звернення: 17.05.2025).

11. Minimum Viable Product (MVP): What it is and how to build one. URL: https://slickplan.com/blog/minimum-viable-product (дата звернення: 17.05.2025).

12. Capstera. Mobile Architecture: Designing for the Future. URL: https://www.capstera.com/mobile-architecture/ (дата звернення: 18.05.2025).

13. Visual Studio Code. Офіційна документація. 2025 р. URL: https://code.visualstudio.com/docs (дата звернення: 20.05.2025).

14. Android Developers. Офіційна документація. 2025 р. URL: https://developer.android.com/ (дата звернення: 20.05.2025).

15. Apple Developer. Xcode Documentation. 2025 р. URL: https://developer.apple.com/xcode/ (дата звернення: 20.05.2025).

16. JetBrains WebStorm. Офіційна документація. 2025 р. URL: https://www.jetbrains.com/webstorm/documentation/ (дата звернення: 20.05.2025).

17. JetBrains PyCharm. Офіційна документація. 2025 р. URL: https://www.jetbrains.com/pycharm/documentation/ (дата звернення: 20.05.2025).

18. Guśpiel M. Mobile App Development Using Kotlin Multiplatform. 2025 р. URL:

https://www.theseus.fi/handle/10024/852201 (дата звернення: 21.05.2025).

19. Vogella. Android SQLite and Room Tutorial. 2025 р. URL: https://www.vogella.com/tutorials/AndroidSQLite/article.html (дата звернення: 21.05.2025).

20. Google Maps Platform Documentation. 2025 р. URL: https://developers.google.com/maps/documentation (дата звернення: 21.05.2025).

21. Firebase. Офіційна документація від Google. 2025 р. URL: https://firebase.google.com/docs (дата звернення: 21.05.2025).

22. Google Cloud. Офіційна документація. 2025 р. URL: https://cloud.google.com/docs (дата звернення: 21.05.2025).

23. Firebase Security Rules. Офіційна документація. 2025 р. URL: https://firebase.google.com/docs/rules (дата звернення: 21.05.2025).

24. Android Developers. Data and file access. 2025. URL: https://developer.android.com/privacy-and-security/data (дата звернення: 21.05.2025).

25. Android Developers. Guide to app architecture (MVVM). 2025. URL: https://developer.android.com/topic/architecture (дата звернення: 21.05.2025).

26. Phillips, B., Stewart, C., Hardy, K. (2021). Android Programming: The Big Nerd Ranch Guide (4th Edition). 2021 p. URL: https://dokumen.pub/android-programming-the-big-nerd-ranch-guide-5nbsped-0137645546-9780137645541-0137645635-9780137645633-9780137645794.html (дата звернення: 22.05.2025).

27. Nagy, M., Somogyi, G. (2021). Mastering Kotlin: Learn Advanced Kotlin Programming Techniques (2nd Edition). 2021 p. URL: https://www.storytel.com/tv/books/mastering-kotlin-learn-advanced-kotlin-programming-techniques-to-build-apps-for-android-ios-and-the-web-learn-advanced-kotlin-programming-techniques-to-build-apps-for-android-ios-and-the-web-940810 (дата звернення: 22.05.2025).

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
“____”_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на бакалаврську кваліфікаційну роботу
ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ ПАРТНЕРА ДЛЯ СПІЛЬНИХ ЗАНЯТЬ СПОРТОМ
08-34.БКР.009.00.000 ТЗ

Керівник: доц.
_____ Євгеній КРИЖАНОВСЬКИЙ
«__» _____ 2025 р.
Розробив студент гр. 2ІСТ-21б
_____ Дмитро КОЗУБ
«__» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.
Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2025р., та індивідуальне завдання на БКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.
2. Джерела розробки:
1) Android Developers. Build UI with Layout Editor.URL: https://developer.android.com/studio/write/layout-editor
2) Android Developers. Kotlin for Android. URL: https://developer.android.com/kotlin
3) SQLite Documentation. URL: https://www.sqlite.org/docs.html

3.Мета і призначення роботи.

Метою роботи є розроблення інформаційної системи у вигляді мобільного застосунку, що дозволить здійснювати пошук напарника для спільних занять спортом з урахуванням спортивних інтересів, геолокації та можливості надсилання запрошень.
4. Методи дослідження:
Аналіз існуючих мобільних додатків для пошуку спортивних напарників, дослідження користувацьких потреб, розроблення UML-діаграм для моделювання взаємодій системи, створення прототипів інтерфейсу мобільного застосунку, реалізація функціоналу на основі архітектури MVVM, локальне зберігання даних за допомогою бібліотеки Room, інтеграція з картографічною службою Google Maps, функціональне тестування системи на реальних та віртуальних пристроях.
5. Етапи роботи і терміни їх виконання:
a) Аналіз предметної області _____ – _____
b) Вибір оптимальної інфраструктури _____ – _____
c) Проектування інформаційної системи _____ – _____
d) Розроблення інформаційної системи пошуку напарника для спільних занять спортом. _____ – _____
e) Оформлення матеріалів до захисту БКР _____ – _____
7. Очікувані результати та порядок реалізації
Отримання інформаційної системи у вигляді мобільного застосунку, що дозволить здійснювати пошук напарника для спільних занять спортом з урахуванням спортивних інтересів, геолокації та можливості надсилання запрошень.
8. Вимоги до розробленої документації
Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні

системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист	«___» _____ 2025 р.
Початок розробки	«___» _____ 2025 р.
Граничні терміни виконання БКР	«___» _____ 2025 р.

Розробив студент групи 2ІСТ-21б _____ Дмитро КОЗУБ

Додаток Б

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Інформаційна система пошуку напарника для спільних занять спортом»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,91 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

□ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____
(підпис)

Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ _____
(підпис)

Особа, відповідальна за перевірку _____ Сергій ЖУКОВ
(підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ
(підпис)

Здобувач _____ Дмитро КОЗУБ
(підпис)

Додаток В

(довідниковий)

Фрагмент лістингу програми

Код файлу додатку MainActivity.kt:

```
package com.example.kursovav1.activity

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import com.example.kursovav1.R
import com.example.kursovav1.SecondActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val button = findViewById<Button>(R.id.btn1)
        button.setOnClickListener {
            val intent = Intent(this, SecondActivity::class.java)
            startActivity(intent)
        }
    }
}
```

Код файлу додатку SecondActivity.kt:

```
package com.example.kursovav1

import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
```

```
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.lifecycleScope
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import androidx.room.Room
import com.example.kursovav1.adapter.UserAdapter
import com.example.kursovav1.data.*
import kotlinx.coroutines.launch
import android.widget.CheckBox
import android.widget.LinearLayout


class SecondActivity : AppCompatActivity() {

    private lateinit var searchViewModel: SearchViewModel
    private lateinit var userAdapter: UserAdapter
    private val selectedFilterNames = mutableSetOf<String>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.second_layout)

        val db = Room.databaseBuilder(
            applicationContext,
            AppDatabase::class.java,
            "sports-partner-db"
        )
            .fallbackToDestructiveMigration()
            .build()
        val dao = db.userDao()

        val factory = SearchViewModelFactory(dao)
        searchViewModel = ViewModelProvider(this, factory)[SearchViewModel::class.java]

        // Адаптер + RecyclerView
        userAdapter = UserAdapter()
        val recyclerView = findViewById<RecyclerView>(R.id.recyclerViewUsers)
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = userAdapter
```

```
    userAdapter.setOnItemClickListener(object : UserAdapter.OnItemClickListener {
        override fun onItemClick(user: UserWithAttributes) {
            showUserDialog(user)
        }
    })

    val editTextSearch = findViewById<EditText>(R.id.editTextSearch)
    findViewById<Button>(R.id.buttonSearch).setOnClickListener {
        val query = editTextSearch.text.toString()
        if (query.isNotEmpty()) {
            searchViewModel.searchUsers(query)
        } else {
            Toast.makeText(this, "Введіть запит для пошуку", Toast.LENGTH_SHORT).show()
        }
    }

    findViewById<Button>(R.id.buttonFilter).setOnClickListener {
        showFilterDialog()
    }

    searchViewModel.results.observe(this) { users ->
        userAdapter.setUsers(users)
    }

    addFakeUsers(dao)
}

private fun addFakeUsers(dao: UserDao) {
    lifecycleScope.launch {
        val existing = dao.searchUsersWithAttributes("") // отримаємо всіх користувачів
        if (existing.size >= 15) {
            searchViewModel.searchUsers("")
            return@launch
        }

        val attrIds = mutableMapOf<String, Int>()
        val attributeNames = listOf(
            "Футбол", "Баскетбол", "Волейбол", "Біг", "Воркаут",
            "На вулиці", "В приміщенні", "Заняття вдвох", "Довге тренування", "Інтенсивне тренування"
```

```
        )

        for (name in attributeNames) {
            val id = dao.insertAttribute(AttributeEntity(name = name)).toInt()
            attrIds[name] = id
        }
        val fakeUsers = listOf(
            Triple("Іван", "Петренко", listOf("Футбол", "На вулиці")),
            Triple("Олена", "Ковальчук", listOf("Біг", "Інтенсивне тренування")),
            Triple("Артем", "Сидоренко", listOf("Баскетбол", "В приміщенні")),
            Triple("Марія", "Іванова", listOf("Волейбол", "Заняття вдвох")),
            Triple("Олег", "Мельник", listOf("Футбол", "Довге тренування")),
            Triple("Катерина", "Романюк", listOf("Воркаут", "Інтенсивне тренування")),
            Triple("Дмитро", "Шевченко", listOf("Біг", "На вулиці")),
            Triple("Анастасія", "Гнатюк", listOf("Баскетбол", "Заняття вдвох")),
            Triple("Максим", "Лисенко", listOf("Футбол", "В приміщенні")),
            Triple("Ірина", "Білик", listOf("Волейбол", "На вулиці")),
            Triple("Андрій", "Козак", listOf("Біг", "Інтенсивне тренування")),
            Triple("Софія", "Кравець", listOf("Футбол", "Заняття вдвох")),
            Triple("Тарас", "Нікітюк", listOf("Воркаут", "На вулиці")),
            Triple("Наталя", "Семенюк", listOf("Баскетбол", "Довге тренування")),
            Triple("Юрій", "Поліщук", listOf("Футбол", "Інтенсивне тренування"))
        )

        for ((name, surname, userAttrs) in fakeUsers) {
            val userId = dao.insertUser(UserEntity(name = name, surname = surname, location = "Львів")).toInt()
            for (attr in userAttrs) {
                attrIds[attr]?.let { dao.insertUserAttributeCrossRef(UserAttributeCrossRef(userId, it)) }
            }
        }

        searchViewModel.searchUsers("")
    }
}

private fun showUserDialog(user: UserWithAttributes) {
    val attributes = user.attributes.map { it.name }

    val sport = attributes.firstOrNull() ?: "Невідомо"
    val tags = attributes
```

```
            .drop(1) // усі крім першого
            .joinToString(" ") { "#${it.replace(" ", "_")}" }

        val message = buildString {
            append("Ім'я: ${user.user.name} ${user.user.surname}\n")
            append("Місто: ${user.user.location}\n\n")
            append("Вид спорту: $sport\n")
            if (tags.isNotEmpty()) {
                append("Хештеги: $tags")
            }
        }

        AlertDialog.Builder(this)
            .setTitle("Профіль користувача")
            .setMessage(message)
            .setPositiveButton("Запросити") { dialog, _ ->
                showInvitationDialog(user)
                dialog.dismiss()
            }
            .setNegativeButton("Скасувати") { dialog, _ ->
                dialog.dismiss()
            }
            .create()
            .show()
    }

    private fun showFilterDialog() {
        val dialogView = layoutInflater.inflate(R.layout.dialog_filter, null)

        val attributeCheckBoxes = listOf(
            R.id.checkFootball to "Футбол",
            R.id.checkBasketball to "Баскетбол",
            R.id.checkVolleyball to "Волейбол",
            R.id.checkRunning to "Біг",
            R.id.checkWorkout to "Воркаут",
            R.id.checkOutdoor to "На вулиці",
            R.id.checkIndoor to "В приміщенні",
            R.id.checkPair to "Заняття вдвох",
            R.id.checkLong to "Довге тренування",
            R.id.checkIntense to "Інтенсивне тренування"
```

```
)

val dialog = AlertDialog.Builder(this)
    .setView(dialogView)
    .create()

dialog.show()
dialog.window?.setLayout(
    (resources.displayMetrics.widthPixels * 0.9).toInt(),
    LinearLayout.LayoutParams.WRAP_CONTENT
)

attributeCheckBoxes.forEach { (id, name) ->
    val checkBox = dialogView.findViewById<CheckBox>(id)
    checkBox?.isChecked = selectedFilterNames.contains(name)
}

val btnApply = dialogView.findViewById<Button>(R.id.buttonApply)
val btnClear = dialogView.findViewById<Button>(R.id.buttonClear)

btnApply.setOnClickListener {
    val newlySelected = attributeCheckBoxes
        .mapNotNull { (id, name) ->
            val checkBox = dialogView.findViewById<CheckBox>(id)
            if (checkBox?.isChecked == true) name else null
        }

    selectedFilterNames.clear()
    selectedFilterNames.addAll(newlySelected)

    lifecycleScope.launch {
        val allFromDb = searchViewModel.getAllAttributes()
        val selectedIds = allFromDb
            .filter { selectedFilterNames.contains(it.name) }
            .map { it.id_attribute }

        if (selectedIds.isNotEmpty()) {
            searchViewModel.filterUsers(selectedIds)
        } else {
            Toast.makeText(this@SecondActivity, "Атрибути не обрані", Toast.LENGTH_SHORT).show()
```

```
            }
        }

        dialog.dismiss()
    }

    btnClear.setOnClickListener {
        selectedFilterNames.clear()
        searchViewModel.searchUsers("")
        dialog.dismiss()
    }
}

private fun showInvitationDialog(user: UserWithAttributes) {
    val dialogView = layoutInflater.inflate(R.layout.dialog_invitation, null)

    val editDate = dialogView.findViewById<EditText>(R.id.editTextDate)
    val editTime = dialogView.findViewById<EditText>(R.id.editTextTime)
    val editLocation = dialogView.findViewById<EditText>(R.id.editTextLocation)
    val editEquipment = dialogView.findViewById<EditText>(R.id.editTextEquipment)
    val editCost = dialogView.findViewById<EditText>(R.id.editTextCost)
    val editNotes = dialogView.findViewById<EditText>(R.id.editTextNotes)

    AlertDialog.Builder(this)
        .setTitle("Запрошення")
        .setView(dialogView)
        .setPositiveButton("Надіслати") { dialog, _ ->
            val date = editDate.text.toString()
            val time = editTime.text.toString()
            val location = editLocation.text.toString()

            if (date.isBlank() || time.isBlank() || location.isBlank()) {
                Toast.makeText(this, "Заповніть всі обов'язкові поля!", Toast.LENGTH_SHORT).show()
                return@setPositiveButton
            }

            val senderId = 1
            val receiverId = user.user.id_user

            val invite = InviteEntity(
```

```
                sender_id = senderId,
                receiver_id = receiverId,
                date = date,
                time = time,
                location = location,
                equipment = editEquipment.text.toString().takeIf { it.isNotBlank() },
                cost = editCost.text.toString().takeIf { it.isNotBlank() },
                notes = editNotes.text.toString().takeIf { it.isNotBlank() },
                status = "pending"
            )

            // Запис у БД
            lifecycleScope.launch {
                val db = Room.databaseBuilder(
                    applicationContext,
                    AppDatabase::class.java,
                    "sports-partner-db"
                ).build()

                db.userDao().sendInvite(invite)

                Toast.makeText(this@SecondActivity, "Запрошення надіслано!", Toast.LENGTH_SHORT).show()
            }

            dialog.dismiss()
        }
        .setNegativeButton("Скасувати") { dialog, _ ->
            dialog.dismiss()
        }
        .create()
        .show()
    }
}
```

Код файлу додатку UserAdapter.kt:

```
package com.example.kursovav1.adapter

import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.kursovav1.R
import com.example.kursovav1.data.UserWithAttributes

class UserAdapter : RecyclerView.Adapter<UserAdapter.UserViewHolder>() {

    private var userList: List<UserWithAttributes> = emptyList()
    interface OnItemClickListener {
        fun onItemClick(user: UserWithAttributes)
    }

    private var listener: OnItemClickListener? = null

    fun setOnItemClickListener(clickListener: OnItemClickListener) {
        listener = clickListener
    }

    fun setUsers(users: List<UserWithAttributes>) {
        userList = users
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_user, parent, false)
        return UserViewHolder(view)
    }

    override fun getItemCount(): Int = userList.size

    override fun onBindViewHolder(holder: UserViewHolder, position: Int) {
        val item = userList[position]
        holder.bind(item)
    }

    inner class UserViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val nameText = itemView.findViewById<TextView>(R.id.textViewName)
```

```
        private val locationText = itemView.findViewById<TextView>(R.id.textViewLocation)
        private val tagsText = itemView.findViewById<TextView>(R.id.textViewTags)

        fun bind(user: UserWithAttributes) {
            nameText.text = "${user.user.name} ${user.user.surname}"
            locationText.text = user.user.location
            tagsText.text = user.attributes.joinToString(" ") { "#${it.name}" }

            itemView.setOnClickListener {
                listener?.onItemClick(user)
            }
        }
    }
}
```

Код файлу додатку Item.kt:

```
package com.example.kursovav1.data

import androidx.room.Embedded
import androidx.room.ForeignKey
import androidx.room.Entity
import androidx.room.Junction
import androidx.room.PrimaryKey
import androidx.room.Relation


@Entity(tableName = "users")
data class UserEntity(
    @PrimaryKey(autoGenerate = true) val id_user: Int = 0,
    val name: String,
    val surname: String,
    val location: String
)

@Entity(tableName = "attributes")
data class AttributeEntity(
    @PrimaryKey(autoGenerate = true) val id_attribute: Int = 0,
    val name: String
```

```
)

@Entity(
    tableName = "user_attributes",
    primaryKeys = ["id_user", "id_attribute"],
    foreignKeys = [
        ForeignKey(entity = UserEntity::class,
            parentColumns = ["id_user"],
            childColumns = ["id_user"],
            onDelete = ForeignKey.CASCADE),
        ForeignKey(entity = AttributeEntity::class,
            parentColumns = ["id_attribute"],
            childColumns = ["id_attribute"],
            onDelete = ForeignKey.CASCADE)
    ]
)
data class UserAttributeCrossRef(
    val id_user: Int,
    val id_attribute: Int
)

data class UserWithAttributes(
    @Embedded val user: UserEntity,
    @Relation(
        parentColumn = "id_user",
        entityColumn = "id_attribute",
        associateBy = Junction(
            value = UserAttributeCrossRef::class,
            parentColumn = "id_user",
            entityColumn = "id_attribute"
        )
    )
    val attributes: List<AttributeEntity>
)

@Entity(
    tableName = "invites",
    foreignKeys = [
        ForeignKey(
            entity = UserEntity::class,
```

```
            parentColumns = ["id_user"],
            childColumns = ["sender_id"],
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = UserEntity::class,
            parentColumns = ["id_user"],
            childColumns = ["receiver_id"],
            onDelete = ForeignKey.CASCADE
        )
    ]
)
data class InviteEntity(
    @PrimaryKey(autoGenerate = true) val id_invite: Int = 0,
    val sender_id: Int,
    val receiver_id: Int,
    val date: String,
    val time: String,
    val location: String,
    val equipment: String? = null,
    val cost: String? = null,
    val notes: String? = null,
    val status: String = "pending" // "accepted", "rejected"
)
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ НАПАРНИКА ДЛЯ СПІЛЬНИХ ЗАНЯТЬ СПОРТОМ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

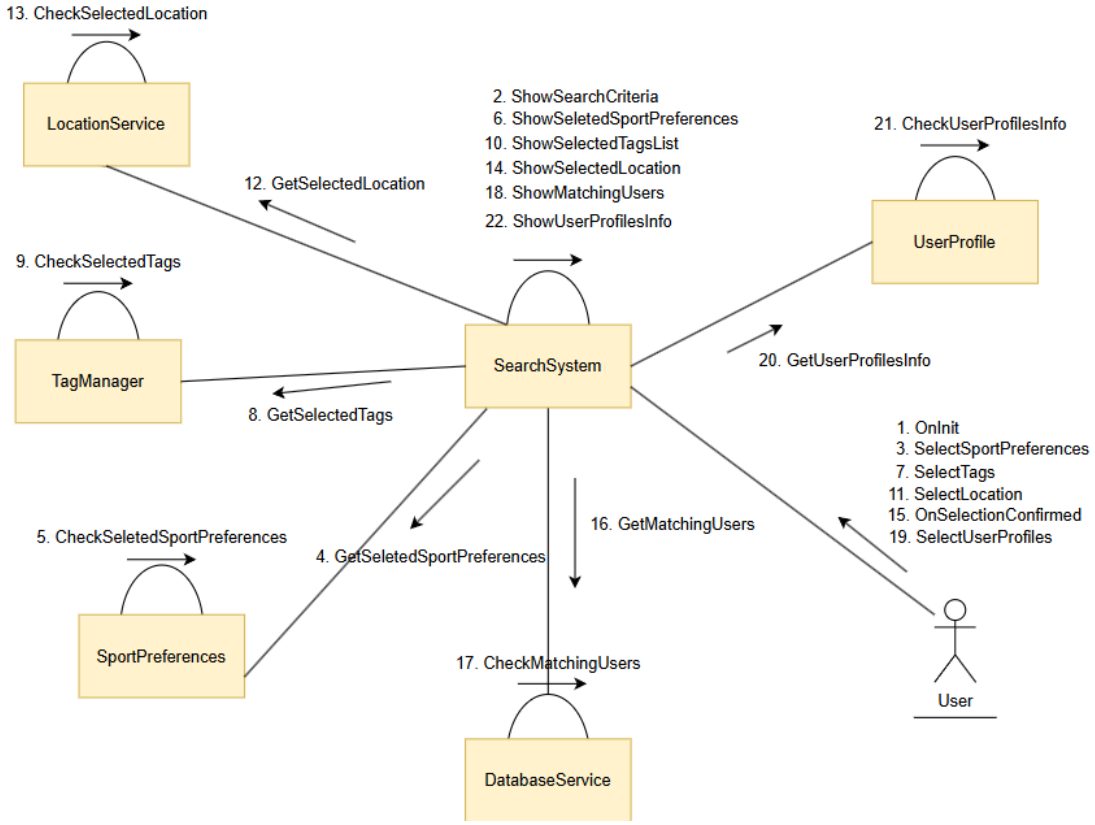

Рисунок Г.1 – Діаграма кооперації пошуку партнера.

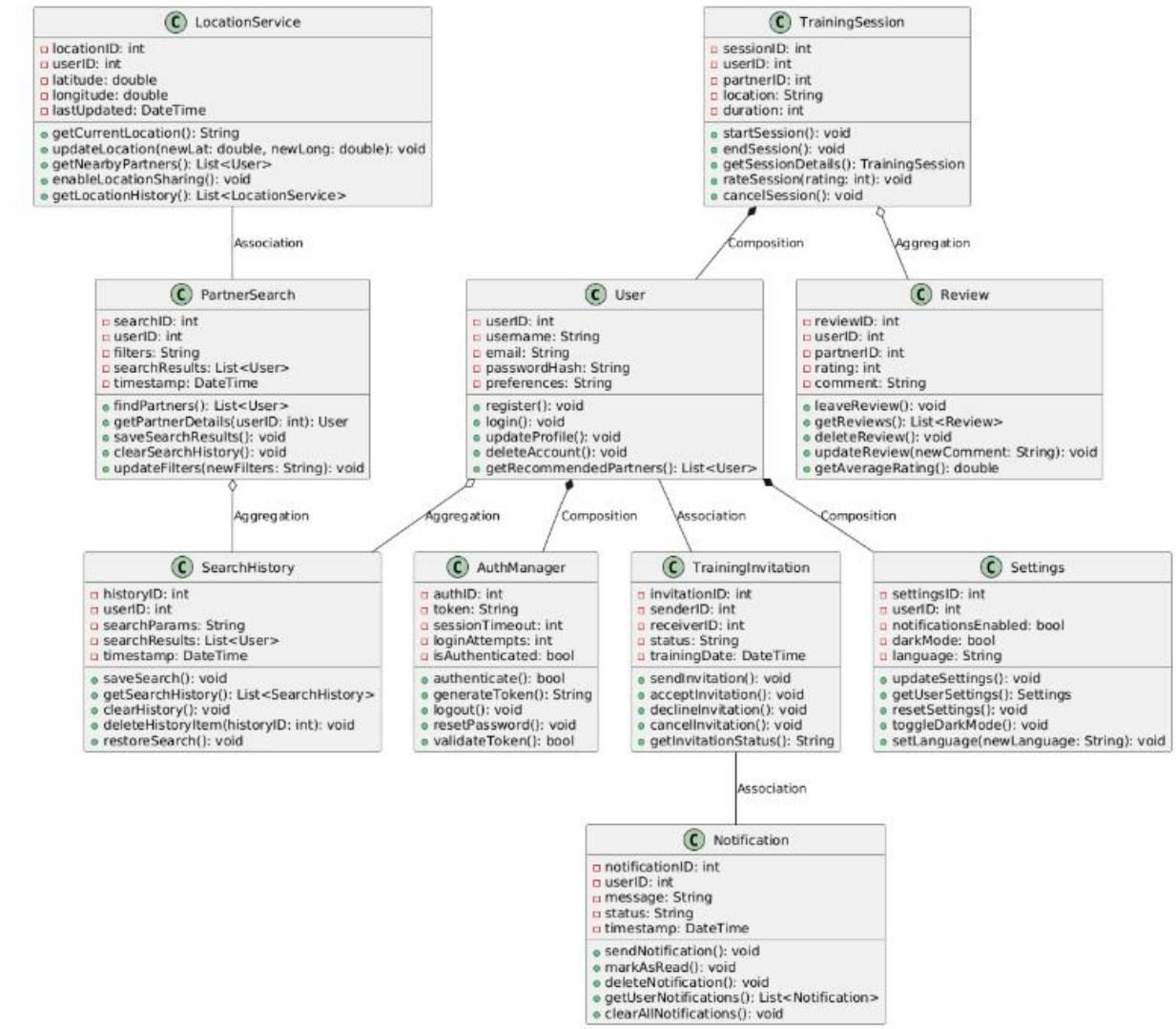

Рисунок Г.2 – Діаграма класів.

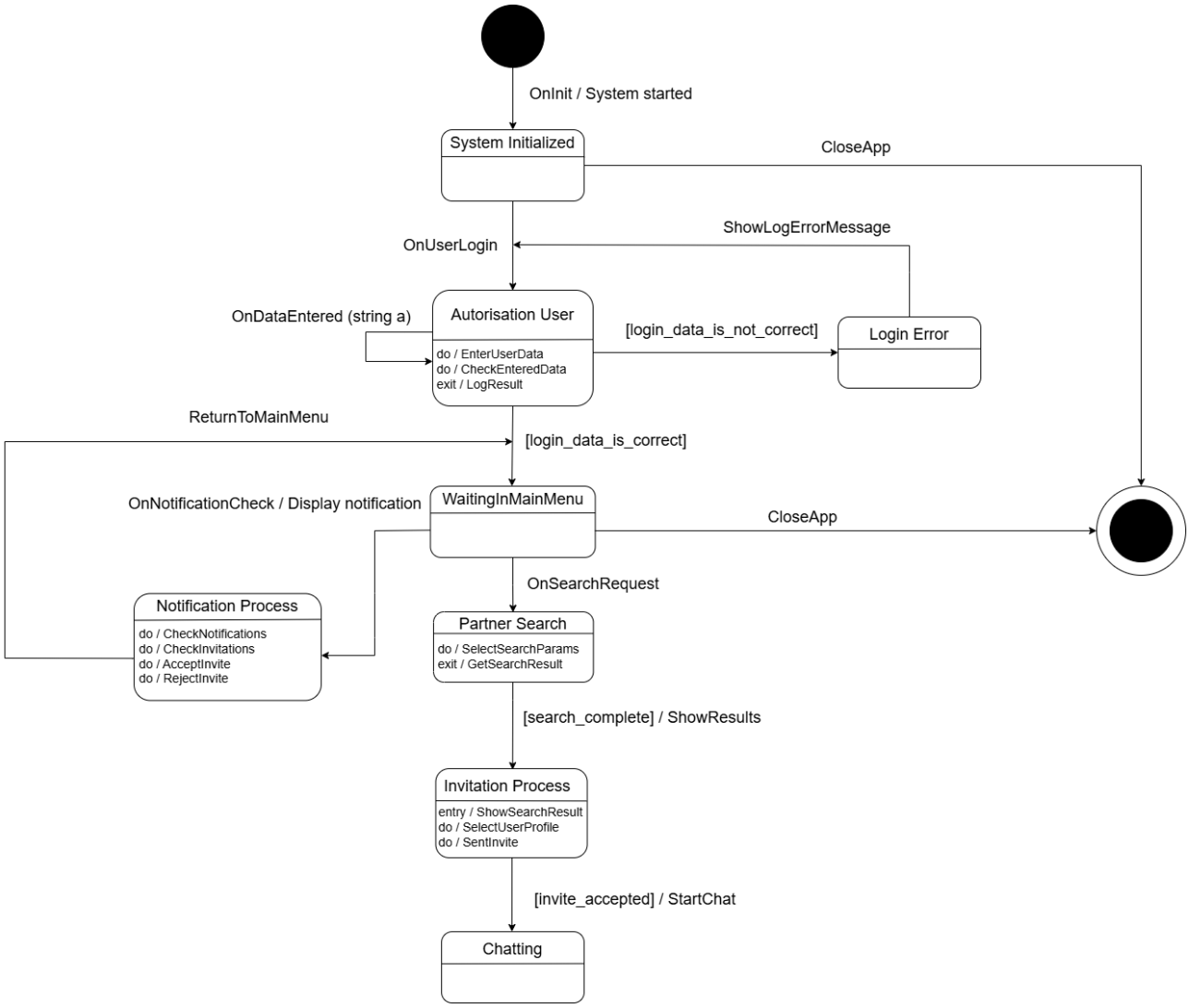

Рисунок Г.3 – Загальна діаграма станів системи.

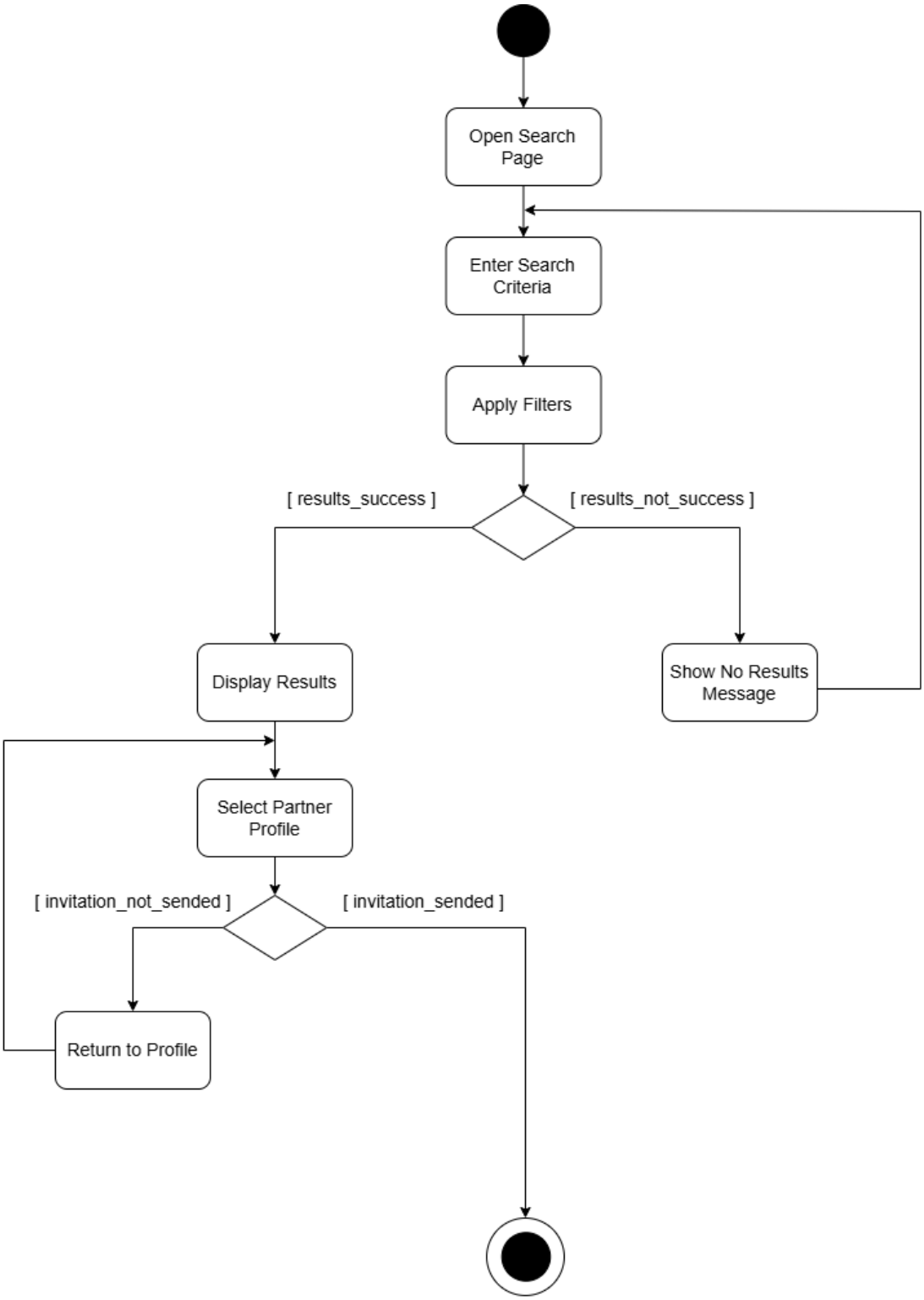

Рисунок Г.4 – Діаграма пошуку партнера.

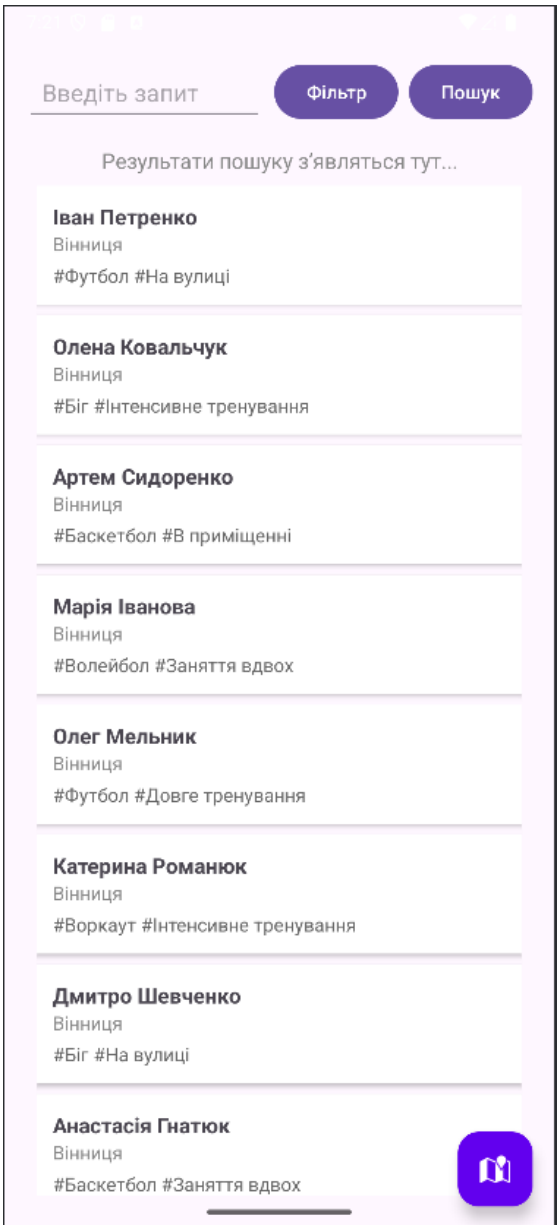

Рисунок Г.5 – Інтерфейс інформаційної системи.