НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____**Ігор БОЛБОТ**	_____**Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____2026 р.	«___» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: « Інформаційна система для підтримки соціальної взаємодії користувачів на основі геолокаційних сервісів »
Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи к.е.н, ст. викладач	_____ (підпис)	**Дмитро НІКОЛАЄНКО**
Виконав	_____ (підпис)	**Захар КОНЄВЄГА**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент_____Белла ГОЛУБ

(підпис)

«____» _____2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
_____КОНЄВЄЗІ ЗАХАРУ СЕРГІЙОВЧУ_____

(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи
«____Інформаційна система для підтримки соціальної взаємодії користувачів на основі геолокаційних сервісів_____»
затверджена наказом від «06» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «22» _травня_2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи:
Метод радіусної геопросторової фільтрації за формулою Вінсента, метод багатокритеріальної фільтрації з AND-композицією, Метод просторової кластеризації маркерів, Метод рольового розмежування доступу_____
Перелік питань, що підлягають дослідженню:
Архітектурні підходи до побудови мобільних застосунків, Методи реалізації геопросторового пошуку, Механізми забезпечення безпеки даних, Методи кластеризації просторових даних_____
Перелік графічних документів (за необхідності).
_____
Дата видачі завдання «_____»_____2026 р.

Керівник бакалаврської кваліфікаційної роботи _ _ **Дмитро НІКОЛАЄНКО**
(підпис)

Завдання взяв до виконання _ _ **Захар КОНЄВЄГА**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4
ВСТУП 5
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Постановка завдання 7
1.2 Моделювання предметної області 11
2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 18
2.1 Логічна модель даних 18
2.2 Вибір системи управління інформаційною базою 23
2.3 Створення інформаційної бази 24
2.4 Організація доступу до інформаційної бази 29
3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 31
3.1 Організаційна структура програмного забезпечення 31
3.2 Вибір інструментарію для створення ППЗ 34
3.3 Алгоритмізація та програмування програмних модулів 37
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 50
4.1 Тестування системи 50
4.2 Вимоги до апаратного та програмного забезпечення 55
4.3 Склад інсталяційного пакету 56
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61
Додаток А 63
Додаток Б 88

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface - інтерфейс програмування застосунків
APK	Android Package Kit - формат інсталяційного пакету Android
CRUD	Create, Read, Update, Delete - базові операції з даними
GPS	Global Positioning System - глобальна система позиціонування
HTTP	HyperText Transfer Protocol - протокол передачі гіпертексту
HTTPS	HyperText Transfer Protocol Secure - захищений протокол передачі даних
JWT	JSON Web Token - стандарт відкритих токенів доступу
JSON	JavaScript Object Notation - текстовий формат обміну даними
OAuth	Open Authorization - протокол авторизації
PKCE	Proof Key for Code Exchange - розширення безпеки OAuth 2.0
PostGIS	геопросторове розширення для PostgreSQL
REST	Representational State Transfer - архітектурний стиль API
RLS	Row-Level Security - безпека на рівні рядків
RPC	Remote Procedure Call - виклик віддалених процедур
SDK	Software Development Kit - комплект засобів розробки
СКБД	Система керування базами даних
SQL	Structured Query Language - мова структурованих запитів
TLS	Transport Layer Security - протокол захисту транспортного рівня
UI	User Interface - інтерфейс користувача
UUID	Universally Unique Identifier - універсальний унікальний ідентифікатор
UML	Unified Modeling Language - уніфікована мова моделювання
WGS-84	World Geodetic System 1984 - геодезична система координат
XML	eXtensible Markup Language - розширювана мова розмітки

ВСТУП

Актуальність. Зі збільшенням доступності цифрових технологій серед побутових споживачів зменшується частка міжособистісного спілкування у сімейному, дружньому та більш широкому соціальному середовищі. Соціальні мережі, попри свою назву, все частіше виступають інструментом пасивного споживання контенту, а не реального спілкування. Внаслідок цього у користувачів формується технологічний стрес, знижується комунікабельність та зростає відчуття відчуженості від суспільства. Наявні мобільні застосунки або орієнтовані на вузькі закриті спільноти, або не надають інструментів для організації соціальних активностей у реальному світі поблизу місцезнаходження користувача. Саме порушення цього питання спонукало до розробки інформаційної системи, метою якої є збільшення міжособистісного спілкування задля зменшення тривоги, стресу та соціальної ізоляції.

Мета. Метою розробки є проєктування та реалізація мобільного застосунку для платформи Android, який забезпечує підтримку соціальної взаємодії користувачів через створення, пошук та участь у геолокаційних активностях. Система також передбачає модерацію контенту уповноваженими менеджерами, зокрема схвалення або відхилення активностей та блокування облікових записів користувачів, що порушують правила платформи. Актуальність розробки підтверджується відсутністю відкритих платформ, які б дозволяли будь-якому користувачу організувати або приєднатися до активності в реальному часі на основі свого географічного положення, без необхідності належати до закритої групи чи спільноти.

Методи та технології. При розробці інформаційної системи використано такі методи та технології. Клієнтська частина реалізована мовою програмування Kotlin для платформи Android з використанням Android Views та ViewBinding для побудови інтерфейсу користувача. Для відображення активностей на карті та роботи з геолокацією застосовано Google Maps SDK з бібліотекою Maps Utils

для кластеризації маркерів. Автентифікація користувачів реалізована за допомогою сервісу Auth0 на основі JWT-токенів, що забезпечує безпечну передачу даних між клієнтом та сервером. Мережева взаємодія здійснюється через бібліотеку Retrofit у поєднанні з OkHttp. Серверна частина побудована на основі PostgREST - інструменту, що автоматично формує REST API на основі схеми бази даних PostgreSQL. Для реалізації фонових push-сповіщень використано WorkManager. Завантаження та відображення зображень виконується бібліотекою Coil. Навігація між екранами реалізована за допомогою Jetpack Navigation Component.

Структура записки. Записка складається з переліку умовних позначень, вступу, основної частини, висновків та списку використаних джерел. Основна частина містить чотири розділи. У першому розділі проведено системний аналіз предметної області: сформовано постановку завдання, виконано огляд існуючих рішень та їх порівняльний аналіз, розроблено моделі предметної області. У другому розділі описано інформаційне забезпечення системи: побудовано логічну модель даних, обґрунтовано вибір системи управління базою даних та описано її створення. У третьому розділі висвітлено питання прикладного програмного забезпечення: організаційну структуру застосунку, вибір інструментарію розробки та алгоритмізацію ключових програмних модулів. У четвертому розділі наведено рекомендації щодо впровадження та експлуатації системи, а саме: результати тестування, вимоги до апаратного та програмного забезпечення, склад інсталяційного пакету.

31 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

31.1 Постановка завдання

Сучасний ритм життя та надмірне занурення у цифровий простір призводять до зменшення реального міжособистісного спілкування. Існуючі соціальні платформи здебільшого орієнтовані на утримання користувача всередині застосунку, тоді як запропонована система має протилежну мету - спонукати користувача вийти у реальний світ та взаємодіяти з іншими людьми безпосередньо.

Розроблювана інформаційна система призначена для організації та пошуку соціальних активностей на основі геолокації користувача. Система має забезпечувати зберігання та обробку такої інформації:

- дані облікових записів користувачів (ідентифікатор, електронна пошта, ім'я, дата народження, аватар, опис профілю);
- дані активностей (назва, опис, дата та час проведення, географічні координати, назва місця, максимальна кількість учасників, мінімальний вік, пароль доступу, статус модерації);
- дані про участь користувачів в активностях;
- дані про організаторів активностей;
- повідомлення чату в межах окремої активності;
- теги активностей для категоризації та фільтрації;
- сповіщення для користувачів;
- відгуки та оцінки організаторів;
- архів активностей користувача;
- записи модерації з рішеннями щодо активностей.

Система має автоматизувати такі операції:

- реєстрацію та автентифікацію користувачів через зовнішній сервіс Auth0;
- створення, редагування та видалення активностей організатором;

- відображення активностей на інтерактивній карті з урахуванням радіусу пошуку та поточного місцезнаходження користувача;
- блокування користувачів менеджером із автоматичним примусовим виходом заблокованого з системи;
- фільтрацію активностей за тегами та пошуковим запитом;
- приєднання до активності з перевіркою пароля та вікового обмеження;
- обмін повідомленнями між учасниками активності в режимі реального часу;
- надсилання push-сповіщень про нові події у фоновому режимі;
- модерацію активностей уповноваженими користувачами з роллю менеджера;
- виставлення відгуків та оцінок організаторам після завершення активності.

За характером використання інформації розроблювана система відноситься до інформаційно-довідкових систем з елементами соціальної мережі, оскільки вона забезпечує як пошук та перегляд даних, так і комунікацію між користувачами. За масштабом система є локально орієнтованою - основний сценарій використання передбачає пошук активностей у межах заданого радіусу від місцезнаходження користувача, проте архітектура не обмежує географічне охоплення. За підтримуваними стандартами та технологіями комунікації система використовує протокол REST для взаємодії між клієнтом і сервером, стандарт OAuth 2.0 для автентифікації та формат JWT для передачі даних авторизації. За ступенем автоматизації система є автоматизованою: більшість операцій виконується без участі адміністратора, за винятком процесу модерації активностей.

До функціональних вимог системи належать:

- можливість реєстрації та входу через Auth0;
- відображення активностей на карті з кластеризацією маркерів;
- створення активності з визначенням місця на карті, дати, опису та додаткових параметрів;
- приєднання до активності та вихід з неї;

- перегляд профілів учасників;
- можливість блокування та розблокування облікового запису користувача менеджером системи;
- чат всередині активності;
- система сповіщень з фоновою перевіркою нових подій;
- панель модерації для менеджерів;
- можливість залишати відгуки після завершення активності.

До нефункціональних вимог належать:

- безпека: передача токена авторизації у кожному запиті, захист даних на рівні рядків бази даних (Row-Level Security);
- продуктивність: відповідь сервера не більше 3 секунд за стандартних умов мережі;
- сумісність: підтримка Android API рівня 26 та вище (Android 8.0+);
- масштабованість: архітектура системи дозволяє нарощувати функціональність без зміни базової структури.

Огляд інформаційних джерел та існуючих рішень. На ринку мобільних застосунків існує ряд рішень, дотичних до тематики організації соціальних активностей. Нижче наведено огляд найбільш поширених із них з аналізом їх переваг та недоліків відносно задачі, що вирішується у даній роботі.

Meetup (meetup.com) - одна з найвідоміших платформ для організації групових зустрічей та заходів за інтересами. Застосунок дозволяє створювати тематичні групи, організовувати регулярні зустрічі та знаходити людей зі схожими інтересами.

Переваги: розвинена система тегів та категорій, велика активна спільнота, підтримка регулярних подій, наявність мобільного застосунку.

Недоліки: платна модель для організаторів, орієнтованість на заздалегідь сформовані групи, відсутність інтеграції з картою для спонтанного пошуку активностей поблизу, складний процес вступу до нових груп для нових користувачів. Застосунок більше підходить для планових заходів, ніж для стихійних активностей.

Facebook Events - вбудований інструмент соціальної мережі Facebook для створення та поширення заходів. Дозволяє запрошувати друзів, публікувати події публічно та отримувати нагадування.

Переваги: інтеграція з великою існуючою соціальною мережею, простота створення події, широке охоплення аудиторії.

Недоліки: прив'язаність до екосистеми Facebook, відсутність геолокаційного пошуку на карті, події здебільшого поширюються серед знайомих, а не серед незнайомців поблизу. Платформа не вирішує задачу знайомства з новими людьми у реальному світі, оскільки орієнтована на вже існуючі соціальні зв'язки.

Eventbrite - платформа для організації та продажу квитків на заходи різного масштабу: від невеликих воркшопів до великих концертів.

Переваги: зручний інструментарій для організаторів, підтримка платних заходів, широкі можливості для просування подій.

Недоліки: орієнтованість переважно на комерційні заходи, відсутність функції спонтанного приєднання до активності поблизу, відсутність чату між учасниками, надмірна складність для побутового користувача, який хоче просто знайти компанію для прогулянки або гри у футбол.

Порівняльний висновок. Жодне з розглянутих рішень не забезпечує в повній мірі можливість спонтанного пошуку та організації неформальних активностей у реальному часі на основі геолокації користувача. Розроблювана система заповнює цю нішу, надаючи відкритий інструмент без прив'язки до існуючих соціальних зв'язків, з інтерактивною картою, чатом та системою модерації контенту.

Для наочного порівняння у табл. 1.1 систематизовано функціональні можливості розглянутих платформ за критеріями, що є визначальними для розроблюваної системи.

Таблиця 1.1

Порівняльний аналіз існуючих рішень

Критерій	Meetup	Facebook Events	Eventbrite	Actimapy
Пошук на карті поблизу	обмежено	–	–	+
Спонтанні (без групи) активності	–	–	–	+
Захист паролем / вік	–	–	–	+
Чат учасників	–	–	–	+
Модерація контенту	–	–	+	+
Безкоштовно для організатора	– (платно)	+	– (платно)	+
Незнайомі поблизу як аудиторія	–	–	–	+
Push-сповіщення (WorkManager)	+	+	+	+
Відкритий вихідний код	–	–	–	+

Таким чином, розроблювана система є єдиною серед розглянутих, що одночасно підтримує геолокаційний пошук без прив'язки до груп, захист активностей паролем та віковим обмеженням, вбудований чат учасників та безкоштовну участь для всіх ролей.

31.2 Моделювання предметної області

Для опису предметної області розроблено комплекс моделей, що відображають як структуру даних системи, так і взаємодію між її учасниками.

Учасники системи (актори):

Система передбачає три типи акторів:

- Неавтентифікований користувач - має доступ лише до екрану входу та реєстрації через Auth0;
- Автентифікований користувач - може переглядати карту активностей, створювати власні активності, приєднуватись до чужих, користуватись чатом, редагувати профіль та отримувати сповіщення;
- Менеджер (actimapy_police) - уповноважений користувач з розширеною роллю, який має доступ до панелі модерації для схвалення або відхилення нових активностей.

Use Case діаграма. Діаграма варіантів використання (рисунок 1) відображає взаємодію між трьома типами акторів та системою. Відношення між акторами та варіантами використання визначають межі доступу для кожної ролі.

Неавтентифікований користувач взаємодіє лише з одним варіантом використання:

UC01 «Реєстрація / вхід через Auth0» - єдиний сценарій, доступний без попередньої автентифікації. Включає відкриття веб-форми Auth0, введення облікових даних та перенаправлення до основного екрану після успішної автентифікації.

Автентифікований користувач має доступ до п'ятнадцяти варіантів використання:

UC01 «Реєстрація / вхід» - доступний також для повторного входу після завершення або примусового завершення сесії.

UC02 «Перегляд активностей на карті» - перегляд затверджених активностей у вигляді маркерів на інтерактивній Google Map. Пов'язаний відношенням «extend» з UC03.

UC03 «Фільтрація за тегами» - розширення UC02, що активується при виборі тега фільтрації. Обмежує відображення активностей за обраною категорією без повторного звернення до сервера.

UC04 «Створення активності» - заповнення форми з назвою, описом, датою, розташуванням на карті, максимальною кількістю учасників, мінімальним віком та опціональним паролем. Створена активність отримує статус pending до рішення менеджера.

UC05 «Редагування активності» - зміна параметрів власної активності організатором. Доступне лише до початку активності.

UC06 «Видалення активності» - видалення власної активності організатором з каскадним видаленням усіх пов'язаних записів (повідомлення, учасники, теги).

UC07 «Приєднання до активності» - основний сценарій участі. Включає два підпорядкованих варіанти: UC08 та UC09.

UC08 «Перевірка пароля активності» - підключений до UC07 відношенням «include»: виконується при кожній спробі приєднатися через тиху перевірку порожнім рядком.

UC09 «Перевірка вікового обмеження» - підключений до UC07 відношенням «include»: обчислює вік користувача на клієнті та порівнює з полем min_age активності.

UC10 «Вихід з активності» - видалення запису участі з таблиці activity_participant.

UC11 «Надсилання повідомлень у чат» - надсилання текстових повідомлень у чат активності; доступне лише учасникам та організатору.

UC12 «Редагування профілю» - оновлення імені, опису, дати народження та аватара у форматі Base64.

UC13 «Перегляд профілю учасника» - перегляд публічного профілю іншого користувача з його активностями та середньою оцінкою.

UC14 «Залишення відгуку про організатора» - виставлення числової оцінки та коментаря після участі в активності. Унікальне обмеження у БД гарантує один відгук від учасника на активність.

UC15 «Отримання push-сповіщень» - фонове отримання сповіщень через NotificationWorker без прямої взаємодії з UI.

Менеджер (actimapy_police) успадковує всі варіанти використання автентифікованого користувача (відношення узагальнення на діаграмі) та додатково має доступ до чотирьох адміністративних сценаріїв:

UC16 «Перегляд активностей на модерацію» - список активностей зі статусом pending у ManagerFragment.

UC17 «Затвердження активності» - зміна статусу на approved; активність стає видимою на карті для всіх користувачів.

UC18 «Відхилення активності» - зміна статусу на rejected із зазначенням причини у полі validation_note таблиці activity_validation.

UC19 «Блокування / розблокування користувача» - встановлення або скидання прапора is_blocked в таблиці app_user; заблокований користувач отримує примусовий логаут при наступному відкритті застосунку.

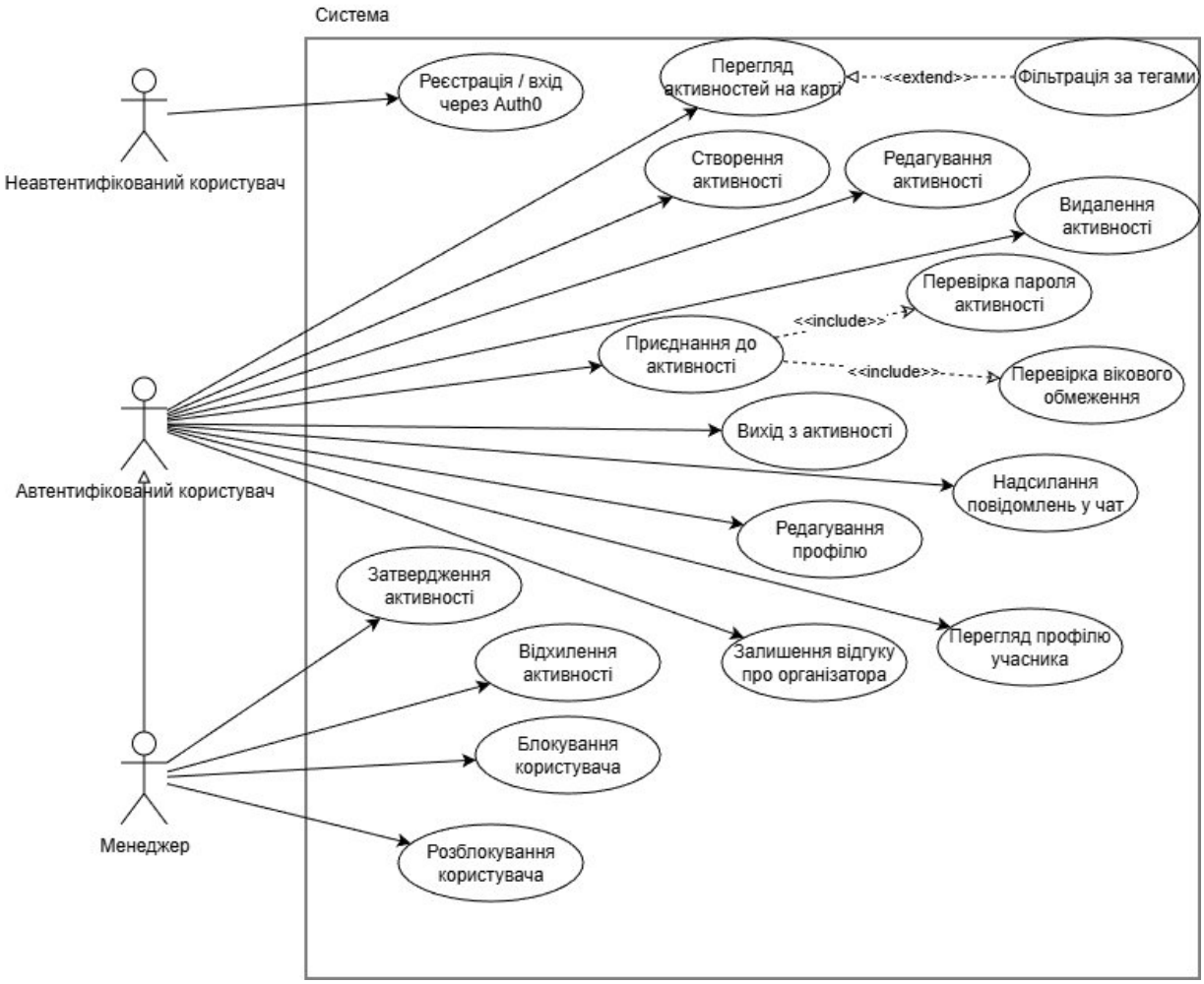


Рисунок 1 Діаграма Use Case

Діаграма послідовності (рисунок 2) відображає взаємодію між чотирма учасниками: Користувач, ActivityDetailFragment (клієнтська частина UI), PostgREST API та PostgreSQL. Діаграма містить два альтернативних блоки (alt), що описують розгалуження сценарію залежно від налаштувань активності.

Основна послідовність розгортається таким чином:

1. Користувач натискає кнопку «Приєднатися» у ActivityDetailFragment.
2. ActivityDetailFragment надсилає GET /app_user?user_id=eq.{id} до PostgREST для отримання дати народження.
3. PostgREST передає запит до PostgreSQL та повертає профіль користувача.

Перший альтернативний блок (вікова перевірка): якщо min_age > 0, ActivityDetailFragment самостійно обчислює вік через функцію calculateAge() без додаткового мережевого запиту. Якщо вік менший за вимогу - сценарій

завершується відображенням повідомлення «Недостатній вік» без подальших серверних викликів.

4. ActivityDetailFragment надсилає POST /rpc/check_activity_password з порожнім рядком як паролем (тиха перевірка).
5. PostgREST викликає збережену функцію check_activity_password у PostgreSQL, яка повертає Boolean.

Другий альтернативний блок (перевірка пароля): якщо функція повертає false (активність захищена), відображається AlertDialog із полем введення пароля. Після підтвердження виконується повторний POST з введеним значенням. Якщо пароль хибний - сценарій завершується повідомленням про помилку. Якщо вірний (або активність відкрита з першої спроби):

6. ActivityDetailFragment надсилає POST /activity_participant з тілом {user_id, activity_id}.
7. PostgREST перевіряє RLS-політику activity_participant_insert та виконує INSERT у PostgreSQL.
8. PostgreSQL повертає HTTP 201 Created.
9. ActivityDetailFragment оновлює UI: кнопка «Приєднатися» замінюється на «Залишити», відображається чат активності.

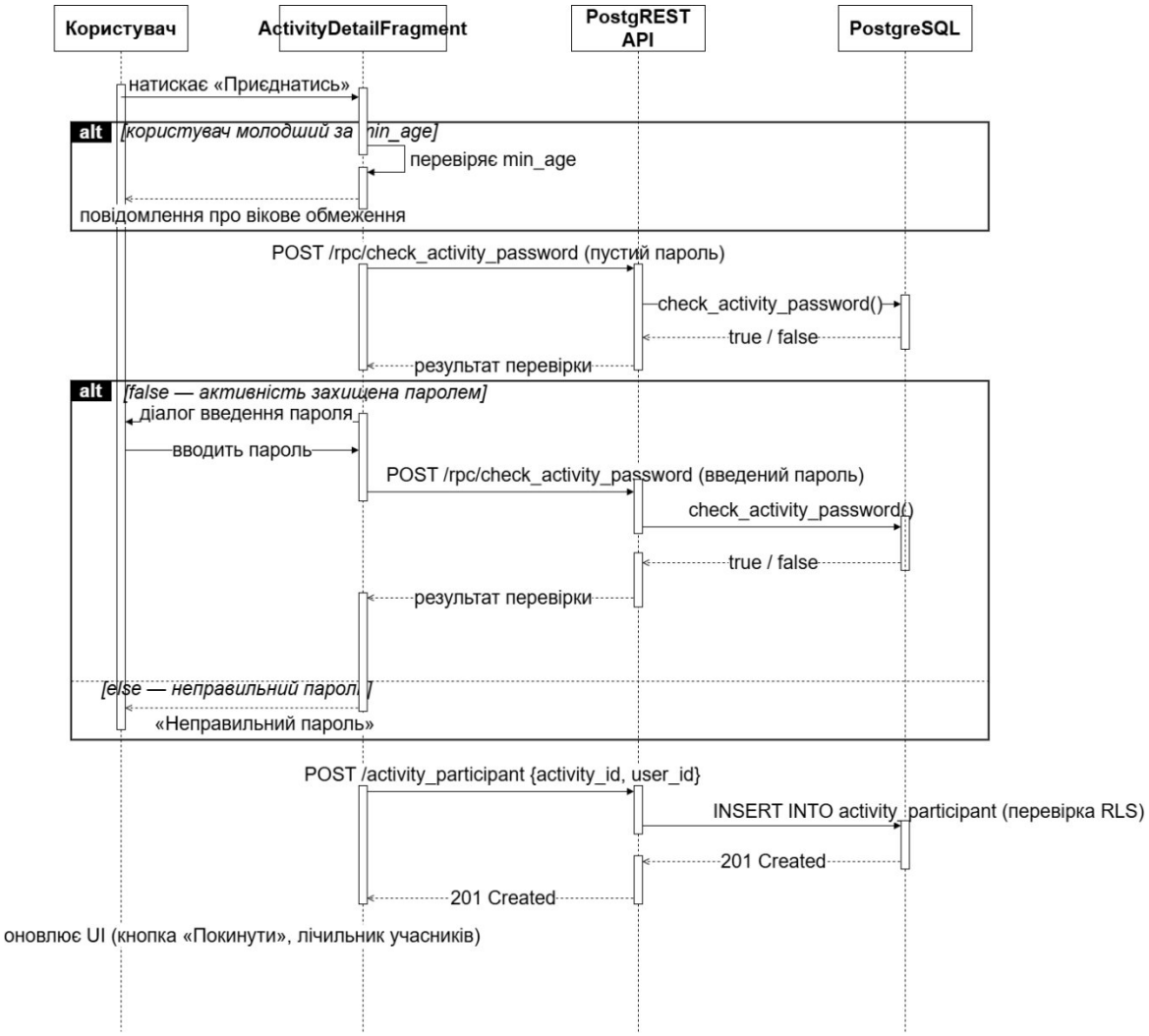
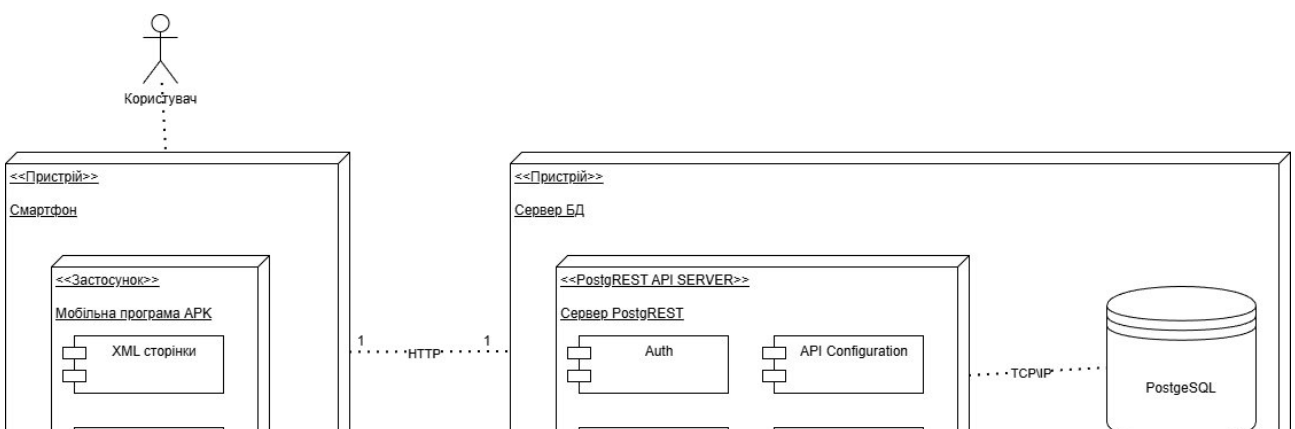


Рисунок 2 Діаграма послідовності

Діаграма компонентів. Система складається з двох основних компонентів: мобільного застосунку (клієнт) та серверної частини. Клієнтська частина включає модулі автентифікації, відображення карти, управління активностями, чату, профілю користувача та фонового сервісу сповіщень. Серверна частина представлена PostgREST, що виступає шаром між клієнтом та базою даних PostgreSQL, і автоматично генерує REST API на основі структури таблиць та RLS-політик.



32 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

32.1 Логічна модель даних

Інформаційна база системи побудована на основі реляційної моделі та складається з п'ятнадцяти таблиць і одного представлення, між якими визначені відповідні зв'язки із забезпеченням каскадних операцій видалення там, де це необхідно.

Центральною сутністю системи є app_user - профіль користувача. Вона містить унікальний ідентифікатор (user_id), отриманий від Auth0, електронну пошту (user_email), ім'я (user_name), необов'язковий опис профілю (user_description), дату народження (user_birthdate), посилання на аватар (avatar_url), дату реєстрації (created_at) та прапор блокування (is_blocked).

Сутність activity зберігає дані про активності: унікальний ідентифікатор (activity_id), назву (activity_name), опис (activity_description), дату та час проведення (activity_date), географічні координати у форматі PostGIS (location), назву місця (location_name), максимальну кількість учасників (max_participants), мінімальний вік для участі (min_age), пароль доступу (activity_password), статус модерації (status: pending, approved, rejected) та дату створення.

Зв'язок між активністю та її організатором реалізований через сутність host, що містить user_id організатора та activity_id. Участь у активностях відображається через сутність activity_participant з полями user_id, activity_id та часом приєднання joined_at. Первинним ключем обох таблиць є складений ключ із двох полів.

Сутність police є довідником уповноважених менеджерів системи. Вона містить єдине поле - user_id із зовнішнім ключем до app_user. Наявність запису у цій таблиці підтверджує повноваження користувача виконувати дії з модерації. Рішення менеджерів щодо активностей зберігаються у сутності activity_validation: activity_id, police_user_id, час прийняття рішення

(validated_at) та необов'язкова причина відхилення (validation_note). Первинний ключ - activity_id, що означає одне рішення на активність.

Сутність warning фіксує попередження, видані менеджерами користувачам: warning_id, user_id порушника, police_user_id менеджера, текст попередження (warning_text) та дату.

Повідомлення чату зберігаються у сутності message: message_id, activity_id, user_id відправника, текст повідомлення (message_text) та час відправлення. При видаленні активності всі повідомлення видаляються каскадно.

Сповіщення для користувачів зберігаються у сутності notification: notification_id, user_i отримувача, заголовок (title), текст (body), прапор прочитання (is_read) та дата створення.

Відгуки учасників про організаторів реалізовані через сутність review: review_id, reviewer_id, host_id, activity_id, числова оцінка (rating) та необов'язковий коментар. Унікальне обмеження (reviewer_id, activity_id) гарантує один відгук від учасника на активність. Окремо існує сутність review_activity для оцінювання самої активності: user_id, activity_id, rating та comment. Унікальне обмеження аналогічне - один відгук від користувача на активність.

Архів збережених активностей реалізований через сутність archived_activity: user_id, activity_id та час збереження saved_at.

Категоризація реалізована через дві пов'язані сутності: tag - довідник тегів із полями tag_id та tag_name (унікальне ім'я), та activity_tag - зв'язок активності з тегами. Додатково існує сутність user_tag, що дозволяє прив'язувати теги інтересів безпосередньо до профілю користувача.

Для відображення активностей на карті створено представлення activity_map_view, яке об'єднує поля таблиці activity з іменем організатора та числовими координатами, отриманими з геопросторового поля через функції PostGIS. Представлення повертає лише активності зі статусом approved.

У табл. 2.1 наведено зведений перелік усіх сутностей інформаційної бази із зазначенням їх призначення та типу первинного ключа.

Таблиця 2.1

Сутності інформаційної бази

№	Назва сутності	Призначення	Первинний ключ
1	app_user	Профіль користувача	user_id (UUID, Auth0)
2	Activity	Дані про активність	activity_id (UUID)
3	Host	Зв'язок організатора з активністю	(user_id, activity_id)
4	activity_participant	Участь користувача в активності	(user_id, activity_id)
5	police	Довідник менеджерів системи	user_id
6	activity_validation	Рішення менеджера щодо активності	activity_id
7	warning	Попередження, видані менеджером	warning_id (UUID)
8	message	Повідомлення чату активності	message_id (UUID)
9	notification	Сповіщення для користувача	notification_id (UUID)
10	review	Відгук учасника про організатора	review_id (UUID)
11	review_activity	Відгук користувача про активність	(user_id, activity_id)
12	archived_activity	Збережені активності користувача	(user_id, activity_id)
13	tag	Довідник тегів категорій	tag_id (UUID)
14	activity_tag	Прив'язка тегів до активності	(activity_id, tag_id)
15	user_tag	Прив'язка тегів інтересів до профілю	(user_id, tag_id)
16	activity_map_view	Представлення для відображення на карті	– (view)

Таблиця 2.1 (закінчення)

Між сутностями визначено такі типи зв'язків. Зв'язок один-до-багатьох (1:N) реалізовано між app_user та activity_participant, message, notification, review, archived_activity - один користувач може мати необмежену кількість пов'язаних записів у кожній із цих таблиць. Аналогічний зв'язок існує між activity та message, activity_participant, activity_tag. Зв'язок багато-до-багатьох (M:N) між app_user і activity реалізований через дві проміжні таблиці: host (для

організаторів) та activity_participant (для учасників). Теги пов'язані з активностями та профілями через activity_tag і user_tag відповідно.

Цілісність посилань забезпечена зовнішніми ключами з каскадним видаленням (ON DELETE CASCADE) у таблицях message, activity_participant, host, activity_tag, archived_activity, activity_validation - при видаленні активності або користувача всі пов'язані записи видаляються автоматично. Для таблиць review та notification застосовано ON DELETE SET NULL, щоб зберегти аналітичні дані після видалення профілю.

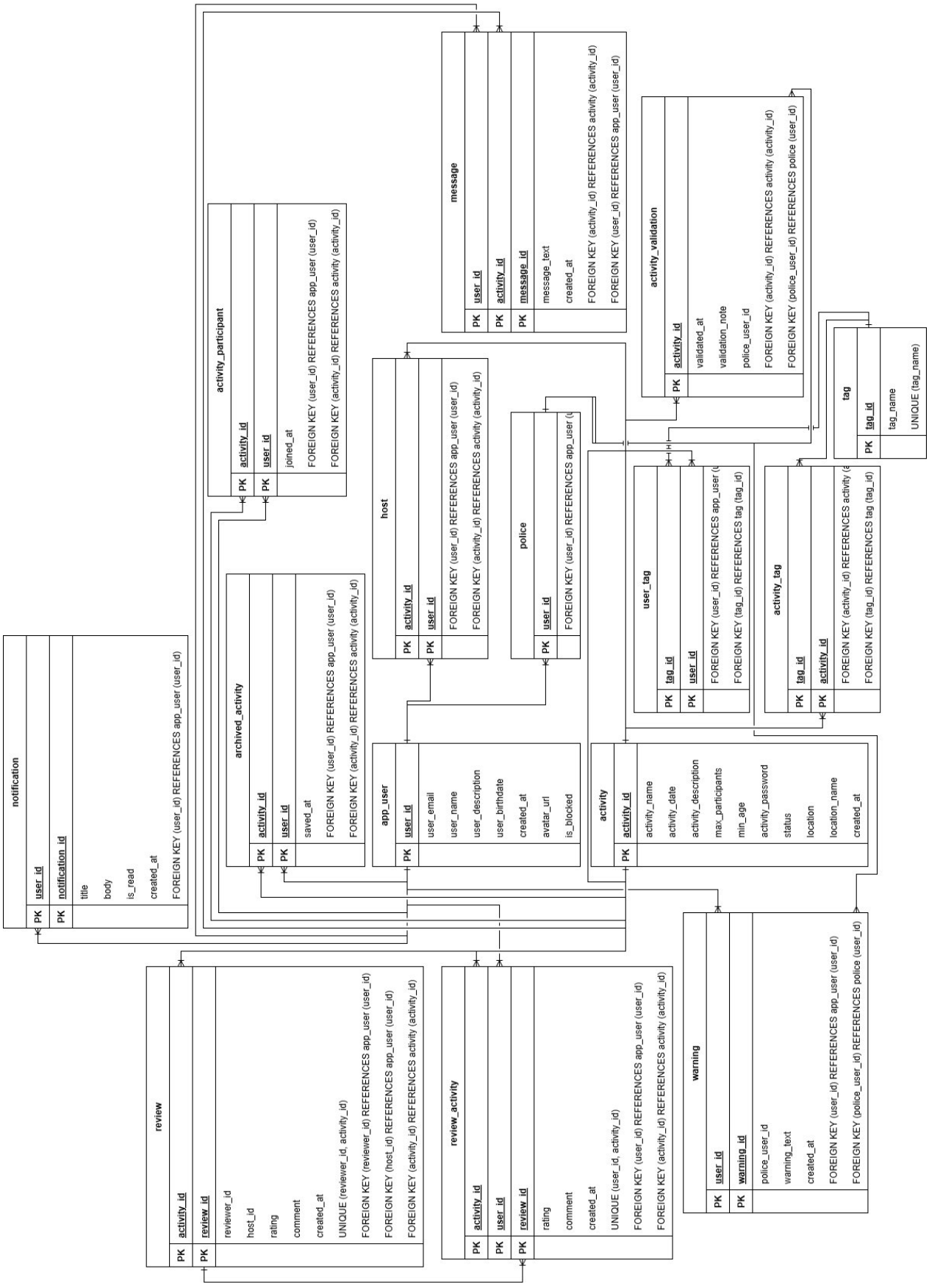


Рисунок 3 ER діаграма

32.2 Вибір системи управління інформаційною базою

Для зберігання та управління даними системи обрано реляційну СКБД PostgreSQL версії 16. Вибір обґрунтовано такими факторами.

По-перше, PostgreSQL підтримує розширення PostGIS, яке додає до стандартного SQL підтримку геопросторових типів даних та функцій. Зокрема, тип geometry(Point, 4326) дозволяє зберігати координати активностей, а функція ST_DWithin - ефективно фільтрувати активності в заданому радіусі від точки без завантаження всіх записів на клієнт.

По-друге, PostgreSQL підтримує механізм Row-Level Security (RLS) - політики безпеки на рівні рядків таблиці. Це дозволяє визначити правила доступу до даних безпосередньо у БД, а не лише на рівні застосунку. Наприклад, RLS-політика гарантує, що користувач може редагувати лише власний профіль, а видалити активність може лише її організатор, незалежно від того, як сформований HTTP-запит.

По-третє, PostgreSQL є відкритим програмним забезпеченням з активною спільнотою, добре документованим API та широкою підтримкою з боку сумісних інструментів, зокрема PostgREST, який використовується у серверній частині даної системи.

Альтернативою міг бути MySQL або SQLite. Проте MySQL не має повноцінної підтримки RLS, а SQLite не підходить для серверних застосунків з одночасним доступом кількох користувачів. MongoDB як нереляційна СКБД не відповідає структурі даних системи, де між сутностями визначені чіткі зовнішні ключі та обмеження цілісності.

У табл. 2.2 наведено порівняльний аналіз розглянутих СКБД за ключовими критеріями вибору.

Таблиця 2.2

Порівняльна характеристика СКБД

Критерій	PostgreSQL 16	MySQL 8	SQLite 3	MongoDB 7
Підтримка RLS	+	–	–	–
Геопросторові типи (PostGIS)	+	частково	–	+ (GeoJSON)
Зовнішні ключі та CASCADE	+	+	+	–
ACID-транзакції	+	+	+	частково
Збережені функції (PL/pgSQL)	+	+	–	–
Сумісність з PostgREST	+	–	–	–
Ліцензія	відкрита	GPL/комерційна	відкрита	SSPL
Підтримка кількох з'єднань	+	+	обмежено	+

Окремо слід зазначити, що PostgreSQL є єдиною СКБД з наведених, яка підтримується інструментом PostgREST - API-шлюзом, що використовується у серверній частині системи. Ця сумісність є визначальним чинником вибору, оскільки PostgREST генерує REST-ендпоінти безпосередньо з таблиць і функцій PostgreSQL, усуваючи необхідність написання окремого серверного застосунку.

32.3 Створення інформаційної бази

Інформаційна база розгорнута на локальному сервері під управлінням PostgreSQL. Усі таблиці створено з явним визначенням первинних та зовнішніх ключів, що забезпечує цілісність даних на рівні СКБД.

Для кожної таблиці активовано механізм Row-Level Security. Визначено набір ролей: actimapy_user - звичайний автентифікований користувач, actimapy_police - менеджер з розширеними правами модерації. Роль призначається користувачу через claim role у JWT-токені, виданому Auth0, і зчитується PostgREST автоматично при кожному запиті.

Доступ до геопросторових функцій забезпечено підключенням розширення PostGIS:

Фрагмент SQL-кода для підключенням розширення PostGIS

CREATE EXTENSION IF NOT EXISTS postgis;

Для відображення активностей на карті створено представлення activity_map_view, що об'єднує поля таблиці activity з іменем організатора та координатами у числовому форматі:

Фрагмент SQL-кода для створення представлення activity_map_view

```
CREATE VIEW activity_map_view AS
SELECT
   a.activity_id   AS id,
   a.activity_name AS name,
   a.activity_date AS date,
   a.activity_description AS description,
   a.location_name,
   a.max_participants,
   ST_Y(a.location::geometry) AS lat,
   ST_X(a.location::geometry) AS lon,
   u.user_name AS host_name
FROM activity a
LEFT JOIN host h ON h.activity_id = a.activity_id
LEFT JOIN app_user u ON u.user_id = h.user_id
WHERE a.status = 'approved';
```

Для перевірки пароля при приєднанні до захищеної активності реалізовано RPC-функцію:

Фрагмент SQL-кода для реалізації RPC-функції

```
CREATE OR REPLACE FUNCTION check_activity_password(act_id UUID, entered TEXT)
RETURNS BOOLEAN AS $$
   SELECT activity_password IS NULL OR activity_password = entered
   FROM activity WHERE activity_id = act_id;
$$ LANGUAGE sql STABLE SECURITY DEFINER;
```

Резервне копіювання бази даних здійснюється засобами стандартної утиліти pg_dump. Для відновлення після збою використовується відповідна утиліта pg_restore. Оскільки система є однорівневою (без реплікації), механізм реплікації даних не передбачений у поточній версії, однак архітектура не виключає його додавання у майбутніх версіях.

Для автоматичного створення профілю після першого входу через Auth0 реалізовано RPC-функцію upsert_profile. Вона приймає email та ім'я користувача і вставляє запис до app_user, якщо він ще не існує, або ігнорує виклик в іншому випадку:

Фрагмент SQL-кода для реалізації RPC-функції

```
CREATE OR REPLACE FUNCTION upsert_profile(p_email TEXT, p_name
TEXT)
RETURNS VOID AS $$
BEGIN
  INSERT INTO app_user (user_id, user_email, user_name)
  VALUES (auth.uid(), p_email, p_name)
  ON CONFLICT (user_id) DO NOTHING;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

Функція оголошена з атрибутом SECURITY DEFINER, що дозволяє їй виконуватись з правами власника функції, а не з правами поточної ролі. Це необхідно, оскільки RLS-політика таблиці app_user за замовчуванням забороняє INSERT від імені звичайного користувача до чужих рядків.

Для геопросторового пошуку активностей реалізовано функцію nearby_activities, що приймає координати точки та радіус у метрах і повертає відфільтровані записи з activity_map_view:

Фрагмент SQL-кода для створення функції nearby_activities

```
CREATE OR REPLACE FUNCTION nearby_activities(
  lat DOUBLE PRECISION,
  lon DOUBLE PRECISION,
```

```
    radius_m DOUBLE PRECISION
)
RETURNS SETOF activity_map_view AS $$
    SELECT * FROM activity_map_view
    WHERE ST_DWithin(
        (SELECT location FROM activity
         WHERE activity_id = activity_map_view.id),
        ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geography,
        radius_m
    );
$$ LANGUAGE sql STABLE SECURITY DEFINER;
```

Функція використовує ST_DWithin з типом geography, що забезпечує коректне обчислення відстані в метрах на поверхні еліпсоїда WGS-84, а не в умовних одиницях площини. Це критично важливо для геолокаційних задач, де похибка плоского обчислення може становити десятки метрів уже на відстані одного кілометра.

RLS-політики визначено для кожної таблиці окремо. Для прикладу, таблиця activity має такі основні політики:

Фрагмент SQL-кода для опису політик

```
-- Переглядати можуть усі автентифіковані
CREATE POLICY activity_select ON activity FOR SELECT
    TO actimapy_user, actimapy_police
    USING (true);
-- Створювати може лише автентифікований користувач
CREATE POLICY activity_insert ON activity FOR INSERT
    TO actimapy_user
    WITH CHECK (true);
-- Редагувати може лише організатор або менеджер
CREATE POLICY activity_update ON activity FOR UPDATE
    TO actimapy_user, actimapy_police
```

```
    USING (
      EXISTS (SELECT 1 FROM host WHERE activity_id = activity.activity_id
            AND user_id = auth.uid())
       OR current_role = 'actimapy_police'
    );
-- Видаляти може лише організатор
CREATE POLICY activity_delete ON activity FOR DELETE
    TO actimapy_user
    USING (
      EXISTS (SELECT 1 FROM host WHERE activity_id = activity.activity_id
            AND user_id = auth.uid())
    );
```

Аналогічний підхід застосовано до таблиць app_user (лише власник може редагувати власний профіль, поліція - виконувати блокування), message (надсилати може лише учасник або організатор), review (залишати відгук може лише учасник, який відвідав активність).

Для підвищення продуктивності запитів створено індекси на полях, що найчастіше використовуються у фільтрах:

Фрагмент SQL-кода для створення індексацій

```
CREATE INDEX idx_activity_status      ON activity (status);
CREATE INDEX idx_host_activity        ON host (activity_id);
CREATE INDEX idx_participant_activity ON activity_participant
(activity_id);
CREATE INDEX idx_message_activity     ON message (activity_id);
CREATE INDEX idx_notification_user    ON notification (user_id, is_read);
CREATE INDEX idx_activity_location    ON activity USING GIST (location);
```

Просторовий індекс GIST на полі location є обов'язковим для ефективного виконання запитів із ST_DWithin - без нього PostgreSQL виконував би послідовне сканування всієї таблиці активностей при кожному геопросторовому фільтрі.

32.4 Організація доступу до інформаційної бази

Для організації доступу клієнтського застосунку до даних використано інструмент PostgREST версії 12. PostgREST - це автономний HTTP-сервер, що автоматично перетворює схему PostgreSQL на повноцінний REST API без написання жодного рядка серверного коду. Кожна таблиця, представлення або збережена функція бази даних стає ендпоінтом, доступним через стандартні HTTP-методи: GET, POST, PATCH, DELETE.

Архітектура взаємодії компонентів побудована таким чином: мобільний застосунок надсилає HTTP-запити до PostgREST, який виступає єдиною точкою входу до бази даних. PostgREST підключається до PostgreSQL від імені спеціального облікового запису-посередника (authenticator), перевіряє JWT-токен запиту та перемикає роль підключення на ту, що вказана у claim role всередині токена. Це дозволяє PostgreSQL застосовувати відповідні RLS-політики для кожного запиту.

Верифікація JWT-токенів виконується самим PostgREST на основі налаштованого секрету або публічного ключа. У даній системі Auth0 видає токени за алгоритмом RS256, тому PostgREST налаштований на верифікацію за публічним ключем відповідного Auth0 tenant'a. Структура claim'ів у токені виглядає наступним чином:

Фрагмент коду для опису у форматі JSON

```
{
  "sub": "auth0|64a1b2c3d4e5f6a7b8c9d0e1",
  "role": "actimapy_user",
  "email": "user@example.com",
  "exp": 1748000000
}
```

де поле sub відповідає user_id у таблиці app_user, а role визначає рівень доступу до даних через RLS-політики.

Фільтрація даних у PostgREST реалізується через query-параметри у форматі PostgREST-операторів. Наприклад, запит GET /activity?status=eq.approved поверне лише затверджені активності, а GET /notification?user_id=eq.{id}&is_read=eq.false - лише непрочитані сповіщення конкретного користувача. Embedded resource запити дозволяють отримувати пов'язані дані в одному запиті: GET /message?select=*,app_user(user_name) повертає повідомлення разом з іменем відправника, використовуючи зовнішній ключ між таблицями. У табл. 2.3 наведено перелік основних ендпоінтів API, що використовуються мобільним застосунком.

Таблиця 2.3

Основні ендпоінти REST API

Метод	Ендпоінт	Призначення
GET	/app_user?user_id=eq.{id}	Отримати профіль користувача
PATCH	/app_user?user_id=eq.{id}	Оновити профіль або заблокувати
GET	/activity?status=eq.approved	Список затверджених активностей
POST	/activity	Створити нову активність
PATCH	/activity?activity_id=eq.{id}	Оновити активність або статус
DELETE	/activity?activity_id=eq.{id}	Видалити активність
POST	/activity_participant	Приєднатися до активності
DELETE	/activity_participant?...	Покинути активність
GET	/message?activity_id=eq.{id}	Отримати повідомлення чату
POST	/message	Надіслати повідомлення
GET	/notification?user_id=eq.{id}	Отримати сповіщення
PATCH	/notification?user_id=eq.{id}	Позначити сповіщення як прочитані
POST	/rpc/upsert_profile	Створити/перевірити профіль після входу
POST	/rpc/check_activity_password	Перевірити пароль активності
GET	/rpc/nearby_activities	Активності в заданому радіусі
GET	/activity_map_view	Активності для відображення на карті

33 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

33.1 Організаційна структура програмного забезпечення

Мобільний застосунок організований відповідно до принципів розділення відповідальності та побудований за пакетною структурою, де кожен пакет відповідає за окремий аспект функціонування системи.

Кореневий пакет com.actimapyPack.actimapy містить компоненти верхнього рівня: точку входу застосунку (ActimapyApp), дві Activity (MainActivity, MainScreen) та всі фрагменти інтерфейсу користувача. Вкладений пакет data розділений на два підпакети: model - моделі даних, та network - засоби мережевої взаємодії.

Пакет data.model містить дата-класи Kotlin, що відображають структури даних, отримані від PostgREST у форматі JSON. Класи розподілені за тематичними файлами: ActivityModels.kt - моделі активностей, UserModels.kt - профілі користувачів та запити оновлення, MessageModels.kt - повідомлення чату, NotificationModels.kt - сповіщення. Серіалізація та десеріалізація виконується бібліотекою Gson через анотації @SerializedName.

Пакет data.network містить три компоненти: ActimapyApi - інтерфейс Retrofit з описом усіх HTTP-ендпоінтів, RetrofitHelper - синглтон, що будує та зберігає єдиний екземпляр OkHttpClient і Retrofit, TokenStore - синглтон для зберігання токена та метаданих сесії в оперативній пам'яті під час роботи застосунку.

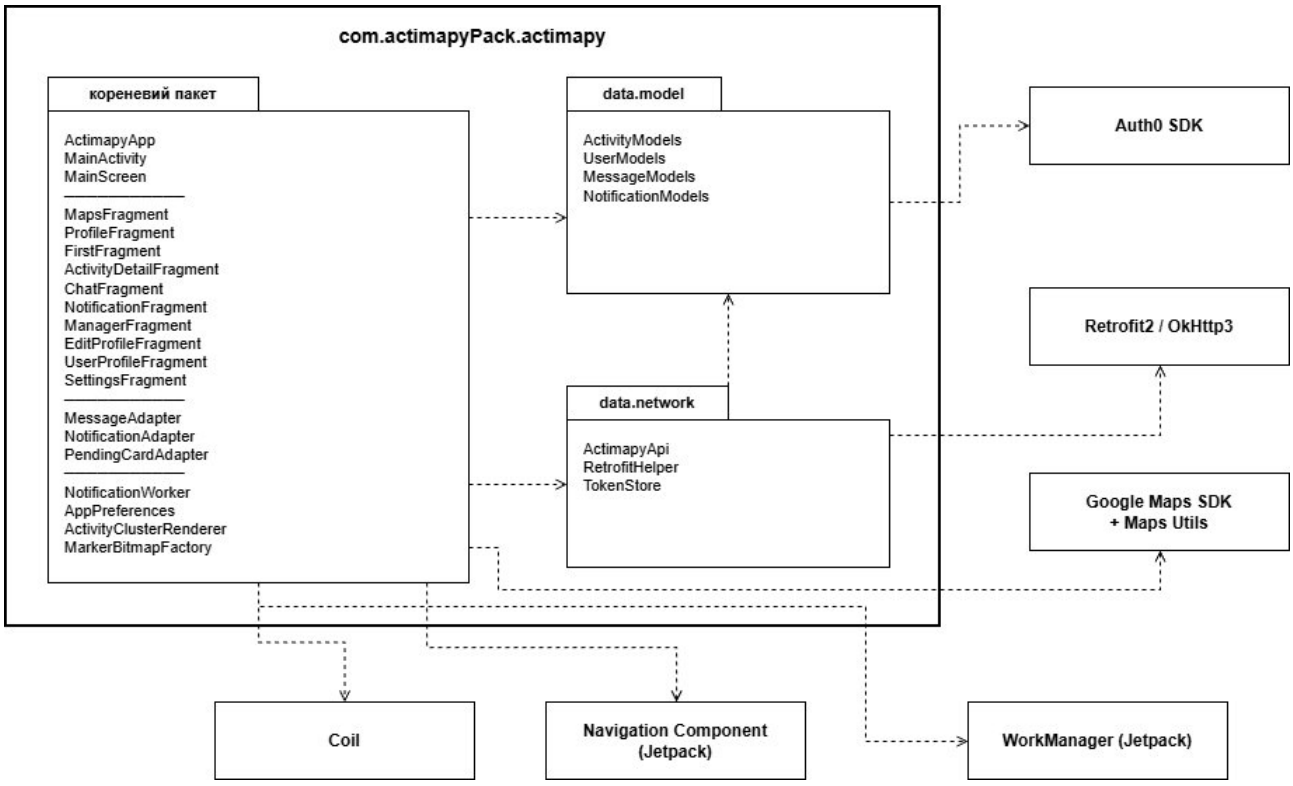


Рисунок 4 Діаграма пакетів

Фрагменти інтерфейсу користувача розміщені у кореневому пакеті та реалізують окремі екрани застосунку. Кожен фрагмент взаємодіє із сервером безпосередньо через RetrofitHelper.api, використовуючи корутини Kotlin у межах lifecycleScope.

Адаптери RecyclerView

(MessageAdapter, NotificationAdapter, PendingCardAdapter) розміщені поруч із відповідними фрагментами у кореневому пакеті.

У табл. 3.1 наведено повний перелік компонентів застосунку із зазначенням їх типу та функціонального призначення.

Таблиця 3.1

Компоненти мобільного застосунку Actimapy

Компонент	Тип	Призначення
ActimapyApp	Application	Точка входу; реєструє WorkManager-задачу та ініціалізує callback onUnauthorized у RetrofitHelper

Таблиця 3.1 (продовження)

MainActivity	Activity	Екран входу; ініціює Auth0 WebAuth flow
MainScreen	Activity	Основний контейнер; містить BottomNavigationView та NavHostFragment
MapsFragment	Fragment	Відображення активностей на Google Map з кластеризацією через ClusterManager
ProfileFragment	Fragment	Власний профіль; відображає бейдж непрочитаних сповіщень; перевіряє is_blocked у onResume
FirstFragment	Fragment	Список активностей (RecyclerView) з рядком пошуку та фільтром тегів
ActivityDetailFragment	Fragment	Деталі активності; реалізує алгоритм приєднання з перевіркою вікового обмеження та пароля
ChatFragment	Fragment	Чат активності; RecyclerView повідомлень та форма надсилання
NotificationFragment	Fragment	Список сповіщень із позначенням прочитаних; скидає лічильник Worker
ManagerFragment	Fragment	Панель модерації для ролі actimapy_police; список активностей зі статусом pending
EditProfileFragment	Fragment	Форма редагування профілю (ім'я, опис, дата народження, аватар у Base64)
UserProfileFragment	Fragment	Профіль іншого користувача; показує кнопку блокування виключно для ролі police
SettingsFragment	Fragment	Налаштування (радіус пошуку, теги фільтрації) зі збереженням в AppPreferences

Таблиця 3.1 (продовження)

MessageAdapter	RecyclerView.Adapter	Рендеринг повідомлень чату; розрізняє вхідні та вихідні бульбашки
NotificationAdapter	RecyclerView.Adapter	Рендеринг сповіщень із форматуванням відносної дати
PendingCardAdapter	RecyclerView.Adapter	Картки pending-активностей із кнопками «Затвердити» / «Відхилити»
NotificationWorker	CoroutineWorker	Фонова перевірка нових сповіщень кожні 15 хвилин через ізольований Retrofit

AppPreferences	SharedPreferences wrapper	Зберігання токена, userId та лічильника сповіщень між сесіями
ActivityClusterRenderer	DefaultClusterRenderer	Побудова Bitmap-маркерів через Canvas; стабільний колір акцентної смуги за хешем activity_id
MarkerBitmapFactory	Object	Генерація Bitmap-зображень одиничних маркерів та кластерних кіл

Таблиця 3.1 (закінчення)

33.2 Вибір інструментарію для створення ППЗ

Мова програмування та середовище розробки. Клієнтська частина реалізована мовою Kotlin у середовищі Android Studio. Kotlin є офіційно рекомендованою мовою для розробки Android-застосунків від Google починаючи з 2019 року. Порівняно з Java, Kotlin забезпечує лаконічніший синтаксис, вбудовану підтримку null-безпеки та нативну підтримку корутин для асинхронного програмування, що суттєво спрощує роботу з мережевими запитами без блокування головного потоку.

Мережева взаємодія. Для виконання HTTP-запитів до PostgREST використовується бібліотека Retrofit 2 у поєднанні з OkHttp. Retrofit дозволяє описати всі API-ендпоінти декларативно через Kotlin-інтерфейс з анотаціями (@GET, @POST, @PATCH, @DELETE, @Query, @Body), що значно зменшує обсяг шаблонного коду порівняно з ручним формуванням HTTP-запитів. OkHttp використовується як HTTP-клієнт нижнього рівня та надає механізм інтерцепторів для автоматичного додавання заголовку авторизації та обробки відповіді HTTP 401.

У табл. 3.2 наведено порівняльну характеристику основних Android-бібліотек для мережевої взаємодії за критеріями, що є визначальними для даної системи.

Таблиця 3.2

Порівняння бібліотек мережевої взаємодії

Критерій	Retrofit 2 + OkHttp	Volley	Ktor Client
Тайп-сейф API через інтерфейс	+	–	+
Нативна підтримка корутин	+	–	+
Механізм інтерцепторів	+ (OkHttp)	обмежено	+
Автоматична десеріалізація (Gson)	+	+/–	+
Обробка помилок через Response<T>	+	–	+
Рівень підтримки Google	висока	обмежена	висока
Інтеграція з OkHttp	нативна	–	–
Поширеність у Android-проєктах	дуже висока	середня	зростаюча

Обраний стек Retrofit 2 + OkHttp є галузевим стандартом для Android-застосунків, що взаємодіють із REST API. Ключова перевага перед альтернативами - механізм OkHttp-інтерцепторів, який дозволяє централізовано ін'єктувати токен авторизації та обробляти HTTP 401 без дублювання коду у кожному фрагменті.

Автентифікація. Реєстрація та вхід користувачів реалізовані через сервіс Auth0 з використанням офіційного Android SDK. Auth0 забезпечує повний цикл автентифікації за протоколом OAuth 2.0 з відкриттям браузера для введення облікових даних. Результатом успішного входу є пара токенів: accessToken у форматі JWT для авторизації запитів та idToken з даними профілю користувача. Вибір Auth0 замість власної реалізації авторизації дозволяє зосередити зусилля на розробці основного функціоналу та делегувати питання безпеки зовнішньому перевіреному сервісу.

Геолокація та карти. Відображення активностей на карті реалізовано за допомогою Google Maps SDK для Android. Для об'єднання близько розташованих маркерів у кластери застосовано бібліотеку Maps Utils з кастомним рендерером ActivityClusterRenderer, що відображає одиничні активності у вигляді інформаційних карток, а кластери - у вигляді кіл з кількістю елементів.

Фонові задачі. Для фонової перевірки нових сповіщень використано WorkManager - бібліотеку Jetpack для планування гарантованих фонових задач. WorkManager автоматично відновлює заплановані задачі після перезавантаження пристрою та враховує обмеження на виконання (наявність

мережі). Мінімальний інтервал між запусками відповідно до обмежень Android становить 15 хвилин.

Завантаження зображень. Для завантаження та кешування зображень аватарів використано бібліотеку Coil, розроблену з нативною підтримкою корутин Kotlin. Аватари, збережені у форматі base64 data-URI, декодуються вручну через BitmapFactory без участі Coil.

У табл. 3.3 наведено порівняння популярних Android-бібліотек для завантаження зображень.

Таблиця 3.3

Порівняння бібліотек завантаження зображень

Критерій	Coil	Glide	Picasso
Написано на Kotlin	+	–(Java)	– (Java)
Нативна підтримка корутин	+	–	–
Розмір бібліотеки	малий	середній	малий
Кешування в пам'яті та на диску	+	+	+
Підтримка GIF	+	+	–
Lifecycle-aware завантаження	+	+	–
Активна підтримка	+	+	обмежена

Навігація. Перехід між екранами реалізований через Jetpack Navigation Component із єдиним графом навігації. Це забезпечує централізоване управління стеком переходів та спрощує передачу аргументів між фрагментами через Bundle.

Серверна частина. Замість розробки власного бекенду використано PostgREST - інструмент, що автоматично генерує повноцінний REST API на основі схеми бази даних PostgreSQL. Це дозволило зосередити зусилля на розробці клієнтської частини та логіки бази даних, уникнувши написання шаблонного серверного коду. Захист даних реалізований безпосередньо у PostgreSQL через механізм Row-Level Security.

33.3 Алгоритмізація та програмування програмних модулів

Модуль автентифікації та управління сесією. Після успішного входу через Auth0 застосунок зберігає accessToken у двох місцях: у TokenStore (оперативна пам'ять, для поточної сесії) та в AppPreferences (SharedPreferences, для фонового NotificationWorker). JWT-токен автоматично додається до кожного HTTP-запиту через OkHttp-інтерцептор у RetrofitHelper:

Фрагмент Kotlon-кода для додавання JWT-токена

```
val token = TokenStore.accessToken
val request = if (token != null) {
    chain.request().newBuilder()
        .header("Authorization", "Bearer $token")
        .build()
} else chain.request()
```

Після успішного входу через Auth0 ідентифікатор користувача (userId) та роль отримуються з idToken. Оскільки верифікація підпису JWT виконується на стороні PostgREST за алгоритмом RS256, клієнт лише декодує Base64-payload без перевірки підпису. Це прийнятно, оскільки сам сервер відхилить підроблений токен:

Фрагмент Kotlon-кода для декодування JWT-токену

```
private fun decodeJwtPayload(token: String): JSONObject {
    val payload = token.split(".")[1]
    val decoded = Base64.decode(
        payload, Base64.URL_SAFE or Base64.NO_PADDING
    )
    return JSONObject(String(decoded))
}
```

Отримані значення sub (userId), role та email записуються до TokenStore та дублюються у AppPreferences для використання фоновим NotificationWorker. Такий підхід є компромісом між зручністю (Worker не потребує власного Auth0-

flow) та безпекою (токен зберігається у SharedPreferences з режимом MODE_PRIVATE, доступним лише для даного застосунку).

Якщо сервер повертає HTTP 401, інтерцептор очищає TokenStore та викликає зареєстрований callback onUnauthorized на головному потоці, що ініціює перехід на екран логіну без участі користувача. Поля TokenStore позначені анотацією @Volatile, оскільки пишуться на головному потоці, але читаються з потоку OkHttp.

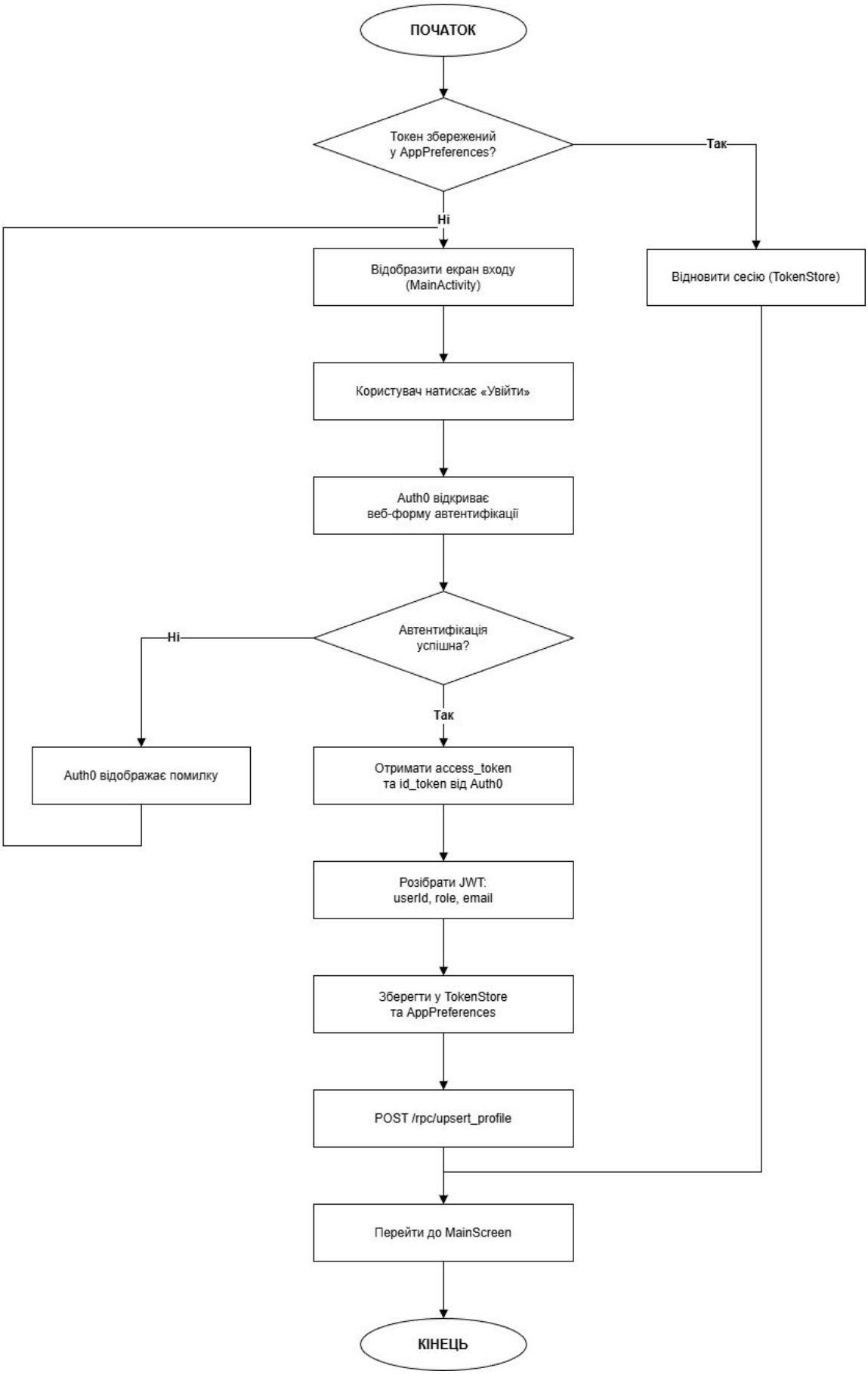


Рисунок 5 Блок-схема AUTH

Модуль приєднання до активність. Алгоритм приєднання є найбільш розгалуженим у системі та включає послідовну перевірку двох умов. Спочатку перевіряється вікове обмеження: якщо min_age > 0, застосунок завантажує дату народження з профілю та обчислює повний вік. Функція calculateAge() повертає -1 у разі помилки парсингу, що трактується як невідомий вік і блокує доступ (fail-safe підхід). Далі перевіряється пароль: застосунок тихо надсилає порожній пароль до RPC-функції check_activity_password. Якщо активність відкрита - функція повертає true і приєднання відбувається без діалогу. Якщо захищена - відображається діалог введення пароля.

Фрагмент Kotlon-кода для перевірки віку користувача

```
private fun calculateAge(birthdate: String): Int = try {
    val p = birthdate.split("-")
    require(p.size == 3)
    val today = Calendar.getInstance()
    var age = today.get(Calendar.YEAR) - p[0].toInt()
    val m = today.get(Calendar.MONTH) + 1
    val d = today.get(Calendar.DAY_OF_MONTH)
    if (m < p[1].toInt() || (m == p[1].toInt() && d < p[2].toInt())) age--
    age
} catch (_: Exception) { -1 }
```

Тиха перевірка пароля виконується одразу після перевірки вікового обмеження: застосунок надсилає запит з порожнім рядком як паролем. Якщо RPC-функція check_activity_password повертає true - активність відкрита, і приєднання відбувається без діалогу. Якщо false - відображається AlertDialog з полем введення пароля, після підтвердження якого виконується повторний запит:

Фрагмент Kotlon-кода для під’єднання до активності

```
private suspend fun attemptJoin(activityId: String) {
    // Тиха перевірка: чи активність відкрита?
    val silentCheck = api.checkActivityPassword(
```

```
            CheckPasswordRequest(activityId, "")
        )
        if (silentCheck.body() == true) {
            doJoin(activityId)
        } else {
            showPasswordDialog { enteredPassword->
                lifecycleScope.launch {
                    val result = api.checkActivityPassword(
                        CheckPasswordRequest(activityId, enteredPassword)
                    )
                    if (result.body() == true) doJoin(activityId)
                    else showError(getString(R.string.wrong_password))
                }
            }
        }
    }
```

Функція doJoin() виконує POST до /activity_participant та оновлює стан UI через перезавантаження деталей активності.

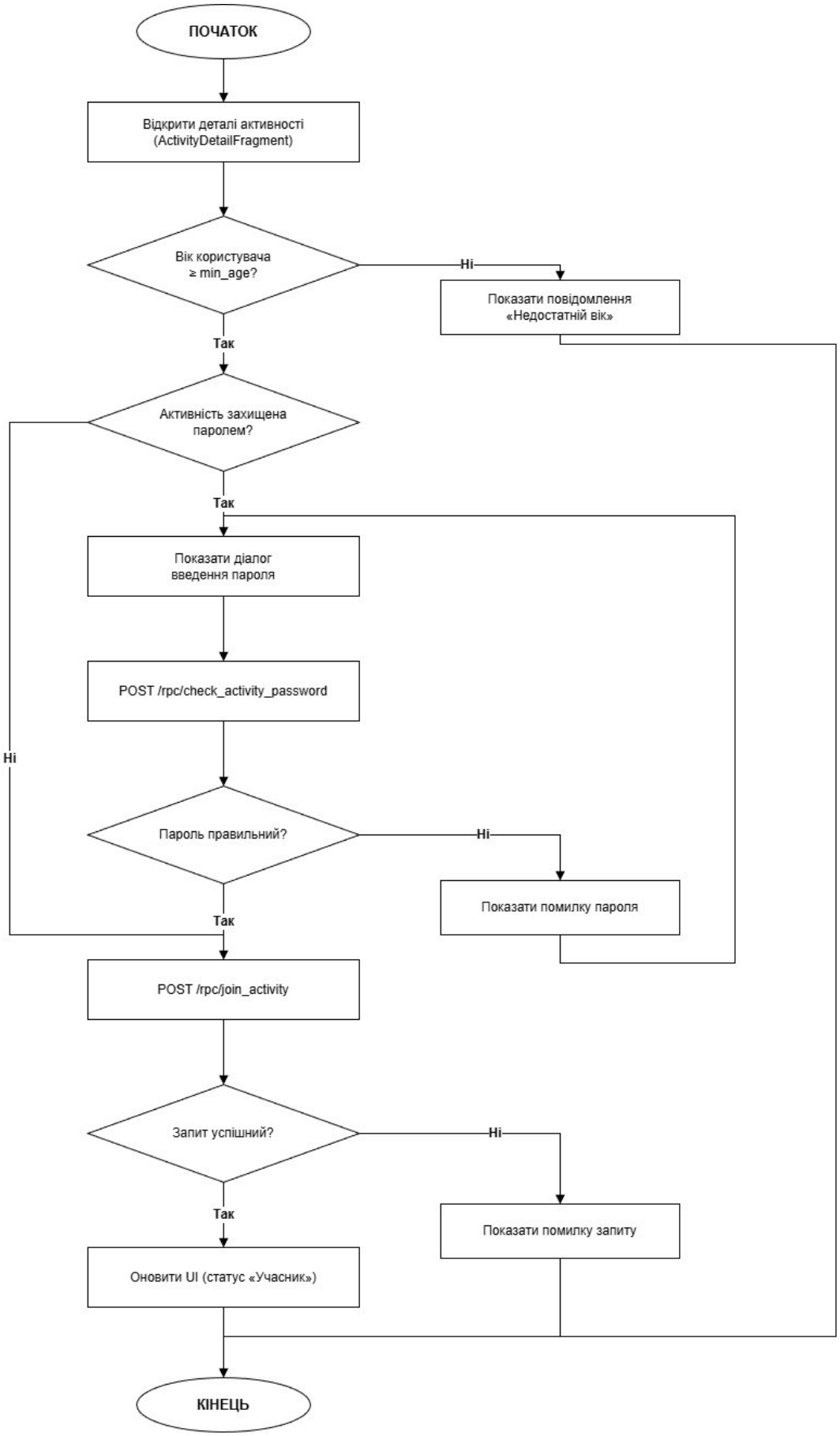


Рисунок 6 Блок-схема приєднання до активності

Модуль відображення карти з кластеризацією. Маркери активностей на карті кластеризуються бібліотекою Maps Utils через ClusterManager. Кастомний рендерер ActivityClusterRenderer будує Bitmap-зображення маркерів програмно через Canvas без використання XML-макетів. Для одиничної активності рендериться картка з назвою, датою та іменем організатора, колір акцентної смуги якої визначається стабільно за хешем ідентифікатора активності:

Фрагмент Kotlon-кода для автоматичного надання кольорів активностям

```
val color = PALETTE[((item.id.hashCode() % PALETTE.size) + PALETTE.size) % PALETTE.size]
```

Формула ((hash % size) + size) % size використовується замість Math.abs(hash % size) для уникнення переповнення у разі Int.MIN_VALUE, абсолютне значення якого в JVM дорівнює самому Int.MIN_VALUE.

Кастомний рендерер ActivityClusterRenderer перевизначає два методи базового класу DefaultClusterRenderer. Метод onBeforeClusterItemRendered() викликається для кожного одиничного маркера та делегує побудову зображення до MarkerBitmapFactory.makeItemBitmap(), який малює через Canvas картку з назвою активності, датою та іменем організатора. Метод onBeforeClusterRendered() аналогічно будує кругле зображення кластера з кількістю вкладених елементів:

Фрагмент Kotlon-кода для кластеризації активностей на мапі

```
override fun onBeforeClusterItemRendered(
    item: ActivityMapItem,
    options: MarkerOptions
) {
    val bmp = MarkerBitmapFactory.makeItemBitmap(context, item)
    options.icon(BitmapDescriptorFactory.fromBitmap(bmp))
        .anchor(0.5f, 1f)
}

override fun onBeforeClusterRendered(
```

```
        cluster: Cluster<ActivityMapItem>,
        options: MarkerOptions
    ) {
        val bmp = MarkerBitmapFactory.makeClusterBitmap(
            context, cluster.size
        )
        options.icon(BitmapDescriptorFactory.fromBitmap(bmp))
            .anchor(0.5f, 0.5f)
    }
```

Перед кожним оновленням списку маркерів після зміни фільтра або отримання нових даних викликається clusterManager.clearItems(), що запобігає появі дублікатів на карті. Після додавання нових елементів через addItems() викликається cluster() для перерахунку кластерів з урахуванням поточного масштабу карти.

Завантаження маркерів відбувається через геопросторову RPC-функцію nearby_activities, якщо геолокація доступна, або через представлення activity_map_view для всіх активностей. Після завантаження застосовується клієнтська фільтрація за пошуковим запитом та тегом.

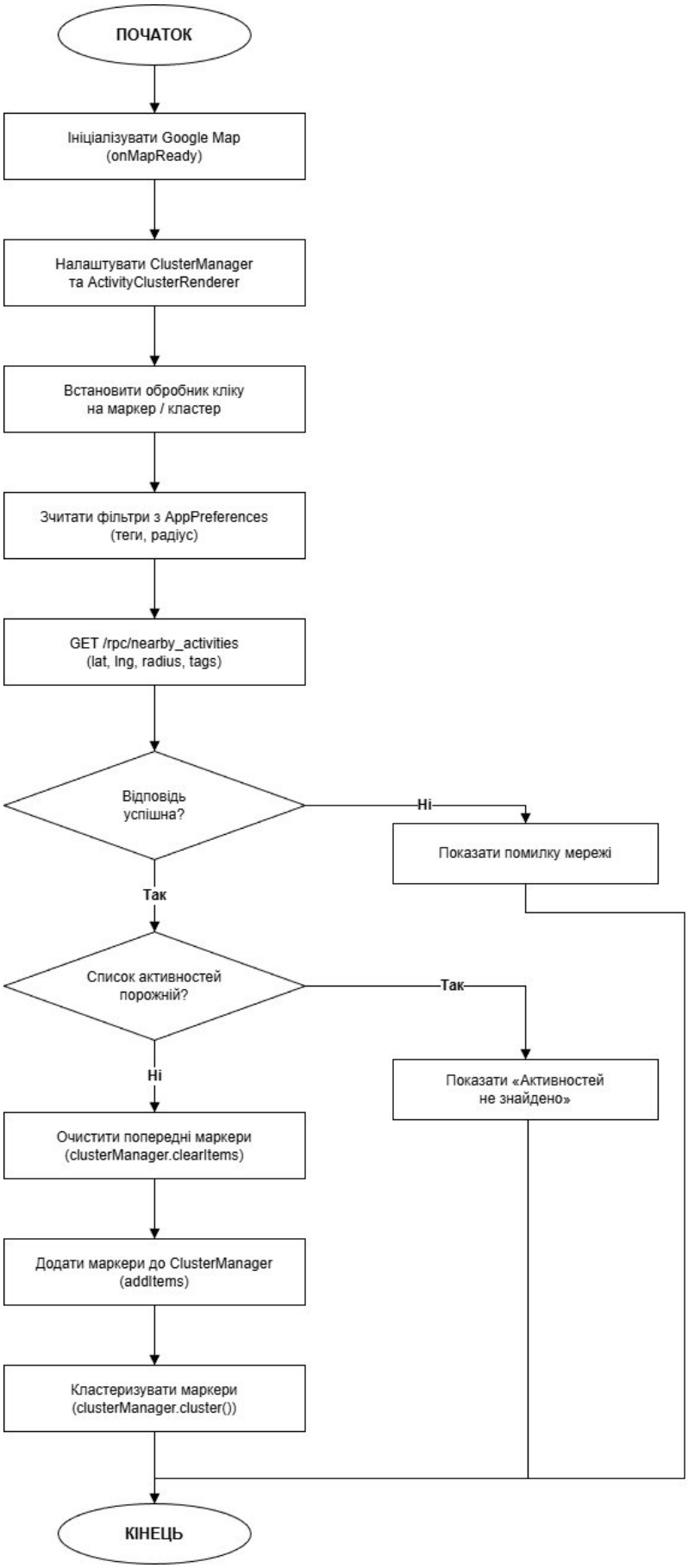


Рисунок 7 Блок-схема завантаження карти

Модуль фонових сповіщень. NotificationWorker запускається WorkManager кожні 15 хвилин за наявності мережевого з'єднання. Для запобігання конфлікту зі спільним синглтоном RetrofitHelper Worker будує власний ізольований екземпляр OkHttpClient та Retrofit. Алгоритм роботи: отримати токен та userId із AppPreferences , підрахувати кількість непрочитаних сповіщень , порівняти з останнім збереженим значенням , якщо нових більше, показати системне push-сповіщення та оновити збережений лічильник. При відкритті екрану сповіщень лічильник скидається до нуля, щоб Worker не надсилав повторні push для вже прочитаних повідомлень.

Для уникнення конфлікту зі спільним синглтоном RetrofitHelper Worker будує власний ізольований HTTP-клієнт безпосередньо у методі doWork():

Фрагмент Kotlon-кода для побудови HTTP-клієнту

```
override suspend fun doWork(): Result {
    val prefs = AppPreferences(applicationContext)
    val token  = prefs.accessToken  ?: return Result.failure()
    val userId = prefs.userId       ?: return Result.failure()
    val client = OkHttpClient.Builder()
        .addInterceptor { chain ->
            chain.proceed(
                chain.request().newBuilder()
                    .header("Authorization", "Bearer $token")
                    .build()
            )
        }.build()
    val api = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
        .create(ActimapyApi::class.java)
```

```
        val notifications = api.getNotifications(
            "eq.$userId"
        ).body() ?: return Result.retry()
        val unreadCount = notifications.count { !it.isRead }
        val storedCount = prefs.lastNotificationCount
        if (unreadCount > storedCount) {
            showPushNotification(unreadCount - storedCount)
            prefs.lastNotificationCount = unreadCount
        }
        return Result.success()
    }
```

Порівняння з lastNotificationCount, збереженим в AppPreferences, дозволяє показувати push лише для нових сповіщень, що з'явились з моменту останнього запуску Worker. При відкритті NotificationFragment лічильник скидається до нуля, щоб уникнути повторних push для вже переглянутих сповіщень.

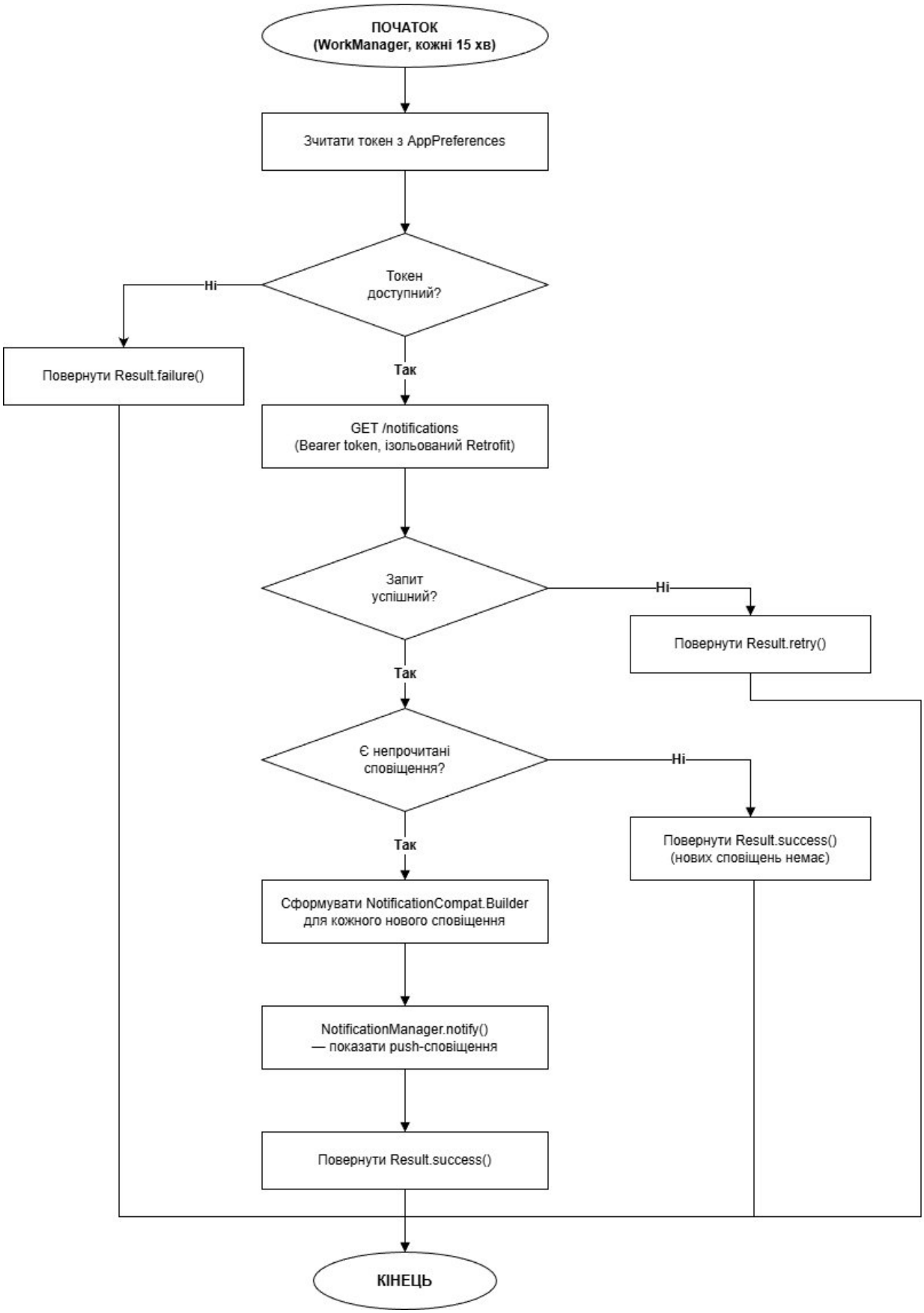


Рисунок 8 Блок-схема NotificationWorker

Модуль блокування користувачів. Менеджер системи може заблокувати будь-якого користувача з його профілю через пункт меню у тулбарі, видимий лише для ролі actimapy_police. Блокування виконується через PATCH-запит до таблиці app_user зі зміною поля is_blocked на true. При наступному відкритті вкладки "Профіль" заблокований користувач отримує примусовий логаут: застосунок зчитує is_blocked == true з відповіді сервера, очищає токени та перенаправляє на екран входу.

Видимість пункту меню "Заблокувати" контролюється через перевірку ролі поточного користувача зі значення TokenStore.userRole. Оскільки роль встановлюється одноразово після входу і зберігається протягом сесії, перевірка виконується синхронно без додаткового мережевого запиту:

Фрагмент Kotlon-кода для зміни блокування\розблокування користувача

```
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
    if (TokenStore.userRole == "actimapy_police") {
        inflater.inflate(R.menu.menu_user_profile, menu)
        val blockItem = menu.findItem(R.id.action_block)
        blockItem.title = if (isBlocked)
            getString(R.string.unblock_user)
        else
            getString(R.string.block_user)
    }
}
```

Після блокування сервер встановлює is_blocked = true для цільового користувача. Сам заблокований не отримує миттєвого сповіщення - примусовий логаут відбувається при наступному зверненні до вкладки «Профіль», де onResume() викликає loadProfile(), зчитує is_blocked == true з відповіді та ініціює очищення TokenStore і перехід до MainActivity. Такий підхід не вимагає WebSocket або механізму push і достатній для практичного сценарію, оскільки між моментом блокування та виявленням цього факту застосунком проходить не більше одного сеансу активного використання.

34 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

34.1 Тестування системи

Тестування системи Actimapy проводилося у формі функціонального тестування на реальному пристрої. Такий підхід обрано з огляду на специфіку мобільного застосунку, де коректність роботи залежить від взаємодії між компонентами UI, мережевими запитами та дозволами операційної системи. Автоматизоване unit-тестування бізнес-логіки не застосовувалося - весь сценарний контроль реалізовано вручну за заздалегідь визначеними тест-кейсами.

Тестування охоплювало такі функціональні блоки: автентифікація, управління активностями, геолокація та фільтрація, чат, сповіщення, профіль та блокування користувача. Кожен тест-кейс містить: ідентифікатор, назву, передумови, кроки виконання та очікуваний результат.

Таблиця 4.1

Тест-кейси функціонального тестування

ID	Назва	Передумови	Кроки	Очікуваний результат
TC –01	Вхід через Auth0	Застосунок встановлено, інтернет наявний	1. Відкрити додаток. 2.Натиснути "Увійти". 3. Ввести коректний email та пароль у веб-формі Auth0	Відбувається перенаправлення до MainScreen. TokenStore містить access token та userId

Таблиця 4.1 (продовження)

ТС –02	Вхід з хибними даними	Застосунок встановлено	1. Відкрити додаток. 2. Натиснути "Увійти". 3. Ввести некоректний пароль	Auth0 відображає повідомлення про помилку. Перенаправлення до MainActivity не відбувається
ТС –03	Автологаут при 401	Користувач увійшов, токен прострочено	1. Дочекатися закінчення терміну дії токена. 2. Виконати будь-яку мережеву дію	RetrofitHelper-interceptor отримує HTTP 401, викликає onUnauthorized - користувача перекинуто на екран входу
ТС –04	Створення активності	Користувач авторизований, роль user	1. Перейти до форми створення. 2. Заповнити назву, опис, час, координати. 3. Натиснути "Створити"	Активність зберігається зі статусом pending. У списку "Мої активності" з'являється новий запис
ТС –05	Відхилення активності без обов'язкових полів	Форма відкрита	1. Залишити поле "Назва" порожнім. 2. Натиснути "Створити"	Форма відображає повідомлення про помилку валідації. Запит на сервер не надсилається

Таблиця 4.1 (продовження)

ТС –06	Вступ до активності з паролем	Активність має пароль, користувач не є учасником	1. Відкрити деталі активності. 2. Натиснути "Приєднатись". 3. Ввести коректний пароль у діалозі	Виклик /rpc/check_activity_password повертає true. Користувача записано до activity_participant
ТС –07	Вступ до активності з хибним паролем	Активність має пароль	1. Відкрити деталі. 2. Натиснути "Приєднатись". 3. Ввести хибний пароль	Сервер повертає false. Діалог відображає повідомлення "Невірний пароль". Запис до учасників не виконується
ТС –08	Вікове обмеження при вступі	Активність має мінімальний вік 21 рік, користувач молодший	1. Відкрити деталі. 2. Натиснути "Приєднатись"	Клієнт розраховує вік з user_birthdate. Відображається повідомлення про вікове обмеження. Кнопка "Підтвердити" недоступна
ТС –09	Відображення маркерів на карті	Є затверджені активності, GPS доступний	1. Перейти на вкладку карти. 2. Дозволити доступ до геопозиції	На карті відображаються маркери активностей. Кластери об'єднують близькі точки

Таблиця 4.1 (продовження)

ТС –10	Фільтрація за тегом	На карті є активності з різними тегами	1. Обрати тег з випадаючого списку	Відображаються лише активності з обраним тегом. Решта маркерів приховується
ТС –11	Надсилання повідомлення в чат	Користувач є учасником або хостом	1. Відкрити чат. 2. Ввести текст. 3. Натиснути "Надіслати"	Повідомлення з'являється у списку. POST на /message повертає HTTP 201
ТС –12	Перевищення ліміту повідомлен	Чат відкрито	1. Ввести рядок довжиною понад 1000 символів.	Кнопка "Надіслати" залишається неактивною або

	ня		2. Спробувати надіслати	відображається лічильник з попередженням
ТС –13	Отримання push-сповіщення	Notification Worker запущено, з'явилося нове сповіщення в БД	1. Дочекатися наступного циклу Worker (≈15 хв)	Системне push-сповіщення відображається у notification drawer. Бейдж на вкладці профілю оновлюється

Таблиця 4.1 (продовження)

ТС –14	Блокування користувача	Виконуючий - роль police, переглядається профіль іншого user	1. Відкрити профіль користувача. 2. Натиснути ":" , "Заблокувати"	PATCH на /app_user?user_id=eq.{id} з is_blocked: true. Мітка меню змінюється на "Розблокувати"
ТС –15	Автологаут заблокованого користувача	Користувача заблоковано під час сесії	1. Залишити застосунок відкритим. 2. Повернутися до вкладки профілю	onResume() викликає loadProfile(). is_blocked == true , TokenStore очищено, навігація до LoginActivity
ТС –16	Затвердження активності менеджером	Роль police, є активність зі статусом pending	1. Відкрити ManagerFragment. 2. Обрати активність. 3. Натиснути "Затвердити"	PATCH змінює status на approved. Активність зникає зі списку pending. Хост отримує сповіщення
ТС –17	Відхилення активності з причиною	Роль police	1. Обрати активність. 2. Натиснути "Відхилити". 3. Ввести причину	status змінюється на rejected. Запис в activity_validation зберігається. Хост отримує сповіщення

ТС –18	Редагування профілю з аватаром	Користувач авторизований	1. Перейти до редагування профілю. 2. Обрати фото з галереї. 3. Зберегти	Зображення конвертується до Base64. PATCH на /app_user зберігає avatar_url. Аватар відображається в ProfileFragment

Таблиця 4.1 (закінчення)

Усі тест-кейси пройдено успішно на пристрої з Android 11 (API 30). Виявлені в процесі тестування дефекти були усунені до фінальної версії - зокрема, виправлено переповнення hashCode у функції обчислення кольору аватара та додано перевірку на відсутність прив'язки у listener-колбеках Coil після зміни стану фрагмента.

34.2 Вимоги до апаратного та програмного забезпечення

Система Actimapy складається з двох основних компонентів: мобільного клієнта на Android та серверної частини, що включає базу даних PostgreSQL і шлюз PostgREST. Розгортання кожного компонента висуває окремі вимоги до середовища.

Вимоги до клієнтської частини. Мобільний застосунок розроблено для платформи Android. Мінімальна підтримувана версія - Android 8.0 (API level 26), що охоплює понад 95% активних пристроїв станом на момент розробки. Рекомендованою є версія Android 10 (API level 29) та вище.

Апаратні вимоги до пристрою:

оперативна пам'ять - не менше 2 ГБ;

вільний простір у пам'яті пристрою - не менше 100 МБ;

модуль GPS або A-GPS для визначення геопозиції;

підключення до мережі Інтернет (Wi-Fi або мобільні дані).

Програмні вимоги та дозволи:

ACCESS_FINE_LOCATION - точна геолокація для відображення позиції на карті та пошуку найближчих активностей;

ACCESS_COARSE_LOCATION - груба геолокація як резервний варіант;

INTERNET - мережеві запити до PostgREST API;

POST_NOTIFICATIONS (Android 13+) - відображення push-сповіщень через WorkManager.

На пристрої повинні бути встановлені сервіси Google Play Services (Google Maps SDK та Google Sign-In інфраструктура використовуються опосередковано).

Вимоги до серверної частини

Серверна частина може бути розгорнута на будь-якому Linux-сервері або у хмарному середовищі. Рекомендованою є наступна конфігурація:

процесор: 2+ ядра, x86-64;

оперативна пам'ять: 4 ГБ і більше;

дисковий простір: 20 ГБ і більше (з урахуванням зростання даних);

операційна система: Ubuntu 22.04 LTS або Debian 12.

Програмне забезпечення серверу:

PostgreSQL 16 з розширенням PostGIS 3.4 - реляційна СУБД з підтримкою геопросторових запитів;

PostgREST 12 - REST API шлюз, що автоматично генерує ендпоінти на основі схеми БД;

Auth0 (SaaS) - зовнішній провайдер автентифікації; локального розгортання не потребує, достатньо зареєстрованого tenant'a.

Між клієнтом та PostgREST здійснюється HTTP-комунікація. Для продуктивного середовища рекомендовано додатково налаштувати Nginx як reverse proxy з підтримкою HTTPS (TLS 1.3), що забезпечить шифрування трафіку та можливість балансування навантаження.

34.3 Склад інсталяційного пакету

Дистрибутив системи Actimapy складається з кількох компонентів, що розгортаються незалежно.

Клієнтська частина. Мобільний застосунок постачається у вигляді файлу actimapy-release.apk - підписаного релізного APK-пакету. Встановлення виконується безпосередньо на пристрій (sideloading) або через публікацію у

Google Play Store. Для публікації через Play Store необхідно сформувати App Bundle (.aab) засобами Android Studio (Build , Generate Signed Bundle).

До складу APK входять:

скомпільований байт-код застосунку (DEX-файли);

ресурси (макети XML, зображення, рядки локалізації);

бібліотеки третіх сторін, статично вбудовані через Gradle (Retrofit, OkHttp, Gson, Coil, Google Maps SDK, Auth0, WorkManager).

Зовнішніх залежностей під час виконання не потребує.

Серверна частина. До складу серверного дистрибутиву входять:

Скрипти міграції бази даних - набір SQL-файлів, що послідовно виконуються і формують повну схему БД:

01_extensions.sql - підключення розширень uuid-ossp та postgis;

02_tables.sql - створення всіх 15 таблиць та представлення activity_map_view;

03_rls.sql - налаштування Row-Level Security: ролі actimapy_user і actimapy_police, політики доступу для кожної таблиці;

04_functions.sql - збережені функції upsert_profile, nearby_activities, check_activity_password;

05_seed.sql (опціонально) - початкові дані: перелік тегів.

Конфігураційний файл PostgREST (postgrest.conf) - містить параметри підключення до БД, налаштування JWT-секрету для верифікації Auth0 токенів та адресу прослуховування:

Фрагмент файлу конфігурації для PostgREST

db-uri = "postgres://authenticator:password@localhost:5432/actimapy"

db-schema = "public"

db-anon-role = "web_anon"

jwt-secret = "<Auth0 RS256 public key або HS256 secret>"

server-port = 3000

Скрипт запуску (start.sh) - автоматизує послідовність: запуск PostgreSQL , виконання міграцій , запуск PostgREST.

Порядок розгортання

Розгортання серверної частини виконується у такій послідовності:

Встановити PostgreSQL 16 та PostGIS: apt install postgresql-16 postgresql-16-postgis-3.

Створити базу даних та користувача: createdb actimapy, createuser authenticator.

Послідовно виконати SQL-скрипти міграції: psql -d actimapy -f 01_extensions.sql і далі за нумерацією.

Налаштувати postgrest.conf відповідно до середовища.

Запустити PostgREST: postgrest postgrest.conf.

Налаштувати Auth0 tenant: зареєструвати Native Application, отримати domain та clientId, вказати їх у res/raw/auth0_config.json мобільного застосунку перед збіркою APK.

Встановити APK на пристрій та перевірити підключення до сервера.

Після успішного проходження всіх кроків система готова до експлуатації. Подальше обслуговування зводиться до стандартного адміністрування PostgreSQL (резервне копіювання через pg_dump, моніторинг навантаження) та оновлення APK при виході нових версій застосунку.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено інформаційну систему для підтримки соціальної взаємодії користувачів на основі геолокаційних сервісів - мобільний застосунок Actimapy для платформи Android.

У ході виконання роботи досягнуто такі результати.

У першому розділі проведено системний аналіз предметної області. Сформовано постановку задачі, визначено функціональні та нефункціональні вимоги до системи. Виконано огляд та порівняльний аналіз існуючих рішень - Meetup, Facebook Events та Eventbrite - і встановлено, що жодне з них повною мірою не вирішує задачу спонтанного пошуку та організації неформальних активностей у реальному часі на основі геолокації без прив'язки до існуючих соціальних зв'язків. Розроблено комплекс UML-моделей: діаграму варіантів використання з трьома акторами та дев'ятнадцятьма сценаріями, діаграму послідовності для ключового сценарію приєднання до активності та діаграму пакетів мобільного застосунку.

У другому розділі сформовано інформаційне забезпечення системи. Розроблено реляційну модель даних, що включає 15 таблиць та одне представлення, між якими визначені зовнішні ключі з каскадними операціями видалення. Обґрунтовано вибір PostgreSQL 16 з розширенням PostGIS як СКБД - єдиного варіанту, що одночасно підтримує геопросторові запити (ST_DWithin з типом geography), механізм Row-Level Security та сумісний з PostgREST. Реалізовано схему бази даних з RLS-політиками для п'яти ролей доступу, збереженими RPC-функціями (upsert_profile, nearby_activities, check_activity_password) та шістьма оптимізаційними індексами, серед яких просторовий індекс GIST. Організовано доступ до бази даних через PostgREST із описом шістнадцяти REST-ендпоінтів.

У третьому розділі реалізовано прикладне програмне забезпечення. Описано пакетну структуру застосунку з розподілом на три пакети: кореневий, data.model та data.network. Обґрунтовано вибір восьми ключових інструментів розробки: Kotlin, Retrofit/OkHttp, Google Maps SDK з Maps Utils, Auth0, WorkManager, Coil та Navigation Component. Алгоритмізовано та реалізовано п'ять ключових модулів: автентифікації та управління сесією з автоматичним оновленням токена, приєднання до активності з послідовною перевіркою вікового обмеження та пароля, відображення карти з динамічною кластеризацією та стабільним кольором маркерів, фонових сповіщень на основі WorkManager та блокування користувачів менеджером. Для кожного модуля розроблено блок-схему алгоритму.

У четвертому розділі наведено рекомендації щодо впровадження та експлуатації системи. Розроблено 18 тест-кейсів функціонального тестування, які охоплюють автентифікацію, управління активностями, геолокацію, чат, сповіщення, модерацію та блокування. Усі тест-кейси пройдено успішно на реальному пристрої з Android 11. Визначено вимоги до апаратного та програмного забезпечення клієнтської (Android 8.0+, 2 ГБ RAM, GPS) та серверної (Ubuntu 22.04, PostgreSQL 16, PostgREST 12) частин. Описано склад інсталяційного пакету та семикроковий порядок розгортання системи.

Практична цінність розробки полягає у реалізації відкритої платформи для організації соціальної активності з геолокаційними функціями, яка не потребує членства у закритих групах та орієнтована на спонтанні активності в реальному світі. Архітектурне рішення на основі PostgREST дозволило суттєво скоротити обсяг серверного коду - захист даних повністю реалізовано на рівні СКБД через RLS, а REST API генерується автоматично зі схеми бази даних.

До перспектив подальшого розвитку системи належать: впровадження реального чату на основі WebSocket замість HTTP-поллінгу, додавання алгоритму рекомендацій активностей на основі тегів інтересів та геолокаційної історії користувача, реалізація верифікації облікових записів, розширення аналітичної панелі для менеджерів та публікація застосунку у Google Play Store.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гудімова А. Х. ПОВЕДІНКОВІ ПАТЕРНИ КОРИСТУВАЧІВ СОЦІАЛЬНИХ МЕРЕЖ ЯК УМОВА ЇХ ПСИХОЛОГІЧНОГО БЛАГОПОЛУЧЧЯ : дис. … д-ра психол. наук : 053.05 / ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І. І. МЕЧНИКОВА МІНІСТЕРСТВО НАУКИ І ОСВІТИ УКРАЇНИ. Одеса, 2021. 248 с.

2. Android Developers. Android Documentation [Електронний ресурс]. - Режим доступу: https://developer.android.com/docs (дата звернення: 12.10.2025).

3. Kotlin Documentation. Kotlin Language Reference [Електронний ресурс]. - Режим доступу: https://kotlinlang.org/docs/home.html (дата звернення: 12.10.2025).

4. PostgreSQL Global Development Group. PostgreSQL 16 Documentation [Електронний ресурс]. - Режим доступу: https://www.postgresql.org/docs/16/ (дата звернення: 05.11.2025).

5. PostGIS Development Group. PostGIS 3.4 Manual [Електронний ресурс]. - Режим доступу: https://postgis.net/docs/manual-3.4/ (дата звернення: 05.11.2025).

6. PostgREST Authors. PostgREST Documentation v12 [Електронний ресурс]. - Режим доступу: https://postgrest.org/en/v12/ (дата звернення: 05.11.2025).

7. Auth0 Inc. Auth0 Documentation [Електронний ресурс]. - Режим доступу: https://auth0.com/docs (дата звернення: 05.11.2025).

8. Square Open Source. Retrofit 2. Type-safe HTTP client for Android and Java [Електронний ресурс]. - Режим доступу: https://square.github.io/retrofit/ (дата звернення: 10.01.2026).

9. Square Open Source. OkHttp: An Efficient HTTP Client [Електронний ресурс]. - Режим доступу: https://square.github.io/okhttp/ (дата звернення: 10.01.2026).

10.Google LLC. Maps SDK for Android [Електронний ресурс]. - Режим доступу: https://developers.google.com/maps/documentation/android-sdk (дата звернення: 08.02.2026).

11.Coil Contributors. Coil: Image Loading for Android backed by Kotlin Coroutines [Електронний ресурс]. - Режим доступу: https://coil-kt.github.io/coil/ (дата звернення: 25.03.2026).

12.Jones M. B. JSON Web Token (JWT) : RFC 7519 [Електронний ресурс] / M. B. Jones, J. Bradley, N. Sakimura. - Internet Engineering Task Force, 2015. - Режим доступу: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 25.03.2026).

13.Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : дис. ... д-ра / R. T. Fielding. - University of California, Irvine, 2000. - 162 с.

14.PostgreSQL Global Development Group. Row Security Policies [Електронний ресурс]. - Режим доступу: https://www.postgresql.org/docs/16/ddl-rowsecurity.html (дата звернення: 17.04.2026).

15.Pillai S. Kotlin Programming Cookbook / S. Pillai. - Birmingham : Packt Publishing, 2018. - 402 p.

16.Google LLC. Understand the Activity Lifecycle [Електронний ресурс]. - Режим доступу: https://developer.android.com/guide/components/activities/activity-lifecycle (дата звернення: 12.10.2025).

17.Auth0 Inc. Auth0 Android SDK [Електронний ресурс]. - Режим доступу: https://github.com/auth0/Auth0.Android (дата звернення: 05.03.2026).

18.Google LLC. ClusterManager - Maps Utils [Електронний ресурс]. - Режим доступу: https://googlemaps.github.io/android-maps-utils/ (дата звернення: 01.03.2026).

19.Хорстманн К. С. Java. Бібліотека професіонала. Т. 1: Основи / К. С. Хорстманн. - 11-те вид. - М. : Вільямс, 2019. - 864 с.

Додаток А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

Модуль зберігання сесії - TokenStore.kt

```
package com.actimapyPack.actimapy.data.network

// @Volatile — поля пишуться на main-thread, читаються з OkHttp-потоку
object TokenStore {
    @Volatile var accessToken: String? = null
    @Volatile var userId: String? = null
    @Volatile var role: String? = null  // "actimapy_user" | "actimapy_police"

    fun isLoggedIn() = accessToken != null
    fun isPolice()   = role == "actimapy_police"

    fun clear() {
        accessToken = null
        userId      = null
        role        = null
    }
}
```

HTTP-клієнт - RetrofitHelper.kt

```
package com.actimapyPack.actimapy.data.network

import android.os.Handler
import android.os.Looper
import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

// Єдиний Retrofit + OkHttpClient для PostgREST.
// onUnauthorized викликається на main-thread при HTTP 401.
object RetrofitHelper {

    const val BASE_URL = "http://192.168.0.105:3000/"

    private val mainHandler = Handler(Looper.getMainLooper())

    var onUnauthorized: (() -> Unit)? = null

    val api: ActimapyApi by lazy {
        val client = OkHttpClient.Builder()
            .addInterceptor { chain ->
                val token = TokenStore.accessToken
                val request = if (token != null) {
                    chain.request().newBuilder()
                        .header("Authorization", "Bearer $token")
                        .build()
                } else {
                    chain.request()
                }
```

```
            val response = chain.proceed(request)

            if (response.code == 401 && token != null) {
                TokenStore.clear()
                mainHandler.post { onUnauthorized?.invoke() }
            }

            response
        }
        .build()

    Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
        .create(ActimapyApi::class.java)
}
}
```

REST-інтерфейс PostgREST - ActimapyApi.kt

```
import com.actimapyPack.actimapy.data.model.*
import retrofit2.Response
import retrofit2.http.*

interface ActimapyApi {

    // Профіль
    @POST("rpc/upsert_profile")
    suspend fun upsertProfile(@Body request: UpsertProfileRequest):
Response<Unit>

    @GET("app_user")
    suspend fun getMyProfile(@Query("user_id") userIdEq: String):
Response<List<AppUser>>

    @PATCH("app_user")
    suspend fun updateProfile(
        @Query("user_id") userIdEq: String,
        @Body update: UpdateProfileRequest
    ): Response<Unit>

    // Активності
    @POST("activity")
    suspend fun createActivity(@Body activity: NewActivityRequest):
Response<Unit>

    @GET("activity")
    suspend fun getApprovedActivities(): Response<List<ActivityItem>>
```

```
    @PATCH("activity")
    suspend fun updateActivity(
        @Query("activity_id") activityIdEq: String,
        @Body update: NewActivityRequest
    ): Response<Unit>

    @DELETE("activity")
    suspend fun deleteActivity(@Query("activity_id") activityIdEq: String):
Response<Unit>

    // Геофільтр
    @GET("rpc/nearby_activities")
    suspend fun getNearbyActivities(
        @Query("lat") lat: Double,
        @Query("lon") lon: Double,
        @Query("radius_m") radiusM: Double
    ): Response<List<NearbyActivityItem>>

    // Хост
    @POST("host")
    suspend fun registerAsHost(@Body entry: HostEntry): Response<Unit>

    @GET("host")
    suspend fun getMyHostedActivities(
        @Query("user_id") userIdEq: String,
        @Query("select")  select: String = "activity_id,activity(*)"
    ): Response<List<HostWithActivity>>

    // Учасники
```

```
@GET("activity_participant")
suspend fun getMyParticipations(
    @Query("user_id") userIdEq: String,
    @Query("select")  select: String = "activity_id,activity(*)"
): Response<List<ParticipantWithActivity>>

@POST("activity_participant")
suspend fun joinActivity(@Body request: JoinActivityRequest): Response<Unit>

@DELETE("activity_participant")
suspend fun leaveActivity(
    @Query("activity_id") activityIdEq: String,
    @Query("user_id")     userIdEq: String
): Response<Unit>

@GET("activity_participant")
suspend fun getParticipants(
    @Query("activity_id") activityIdEq: String
): Response<List<ParticipantEntry>>

@GET("activity_participant")
suspend fun getParticipantsBatch(
    @Query("activity_id") activityIdsIn: String,
    @Query("select")      select: String = "activity_id"
): Response<List<ParticipantEntry>>

@GET("activity_participant")
suspend fun getParticipantsWithNames(
    @Query("activity_id") activityIdEq: String,
    @Query("select")      select: String = "user_id,app_user(user_name)"
```

```
): Response<List<ParticipantWithName>>

@GET("host")
suspend fun checkIsHost(
    @Query("activity_id") activityIdEq: String,
    @Query("user_id")     userIdEq: String
): Response<List<HostEntry>>

@GET("host")
suspend fun getHostWithName(
    @Query("activity_id") activityIdEq: String,
    @Query("select")      select: String = "user_id,app_user(user_name)"
): Response<List<HostWithName>>

// Чат
@GET("message")
suspend fun getMessages(
    @Query("activity_id") activityIdEq: String,
    @Query("order")       order: String = "created_at.asc"
): Response<List<Message>>

@GET("message")
suspend fun getChatMessages(
    @Query("activity_id") activityIdEq: String,
    @Query("select")      select: String = "*,app_user(user_name)",
    @Query("order")       order: String  = "created_at.asc"
): Response<List<MessageWithSender>>

@POST("message")
suspend fun sendMessage(@Body message: NewMessage): Response<Unit>
```

```
    // Пароль активності
    @POST("rpc/check_activity_password")
    suspend fun checkActivityPassword(
        @Body request: CheckPasswordRequest
    ): Response<Boolean>

    // Архів
    @POST("archived_activity")
    suspend fun archiveActivity(@Body body: ArchiveActivityRequest):
Response<Unit>

    @DELETE("archived_activity")
    suspend fun unarchiveActivity(
        @Query("user_id")     userIdEq: String,
        @Query("activity_id") activityIdEq: String
    ): Response<Unit>

    @GET("archived_activity")
    suspend fun getMyArchivedActivities(
        @Query("user_id") userIdEq: String,
        @Query("select")  select: String = "activity_id,activity(*)"
    ): Response<List<ArchivedWithActivity>>

    @GET("archived_activity")
    suspend fun checkArchived(
        @Query("user_id")     userIdEq: String,
        @Query("activity_id") activityIdEq: String,
        @Query("select")      select: String = "activity_id"
    ): Response<List<ArchiveActivityRequest>>
```

```
@GET("archived_activity")
suspend fun getArchivedIds(
    @Query("user_id") userIdEq: String,
    @Query("select")  select: String = "activity_id"
): Response<List<ArchiveActivityRequest>>

// Сповіщення
@GET("notification")
suspend fun getNotifications(
    @Query("user_id") userIdEq: String,
    @Query("order")   order: String = "created_at.desc"
): Response<List<NotificationItem>>

@GET("notification")
suspend fun getUnreadNotifications(
    @Query("user_id") userIdEq: String,
    @Query("is_read") isReadEq: String = "eq.false",
    @Query("select")  select: String   = "notification_id"
): Response<List<NotificationIdOnly>>

@PATCH("notification")
suspend fun markAllNotificationsRead(
    @Query("user_id") userIdEq: String,
    @Query("is_read") isReadEq: String = "eq.false",
    @Body update: MarkReadRequest = MarkReadRequest()
): Response<Unit>

// Статистика
@GET("daily_activity_report")
```

```
    suspend fun getDailyStatistics(): Response<List<DailyReportItem>>

    // Теги
    @GET("tag")
    suspend fun getTags(): Response<List<TagItem>>

    @POST("activity_tag")
    suspend fun addActivityTag(@Body request: ActivityTagRequest):
Response<Unit>

    @GET("activity_tag")
    suspend fun getActivityTags(
        @Query("activity_id") activityIdEq: String,
        @Query("select")      select: String = "tag_id,tag(tag_name,tag_name_en)"
    ): Response<List<ActivityTagWithName>>

    @DELETE("activity_tag")
    suspend fun deleteActivityTags(
        @Query("activity_id") activityIdEq: String
    ): Response<Unit>

    @GET("activity_tag")
    suspend fun getActivitiesByTag(
        @Query("tag_id") tagIdEq: String,
        @Query("select") select: String = "activity_id"
    ): Response<List<ActivityTagEntry>>

    // Відгуки
    @POST("review")
    suspend fun submitReview(@Body review: NewReview): Response<Unit>
```

```
@GET("review")
suspend fun getMyReview(
    @Query("reviewer_id") reviewerIdEq: String,
    @Query("activity_id") activityIdEq: String,
    @Query("select")      select: String = "review_id,rating"
): Response<List<ReviewEntry>>

@GET("review")
suspend fun getHostReviews(
    @Query("host_id") hostIdEq: String,
    @Query("select")  select: String = "rating"
): Response<List<ReviewEntry>>

// Карта
@GET("activity_map_view")
suspend fun getActivityMapItems(): Response<List<ActivityMapItem>>

// Адміністрування (police)
@PATCH("app_user")
suspend fun blockUser(
    @Query("user_id") userIdEq: String,
    @Body update: BlockUserRequest
): Response<Unit>

@GET("activity")
suspend fun getPendingActivities(
    @Query("status") statusEq: String = "eq.pending",
    @Query("order")  order: String    = "created_at.asc"
): Response<List<ActivityItem>>
```

```
    @PATCH("activity")
    suspend fun updateActivityStatus(
        @Query("activity_id") activityIdEq: String,
        @Body update: ActivityStatusUpdate
    ): Response<Unit>

    @POST("activity_validation")
    suspend fun saveValidation(@Body request: ValidationRequest): Response<Unit>

    @GET("activity_validation")
    suspend fun getValidation(
        @Query("activity_id") activityIdEq: String,
        @Query("order")       order: String = "created_at.desc",
        @Query("limit")       limit: Int    = 1
    ): Response<List<ValidationEntry>>
}
```

Автентифікація - MainActivity.kt

```
package com.actimapyPack.actimapy

import android.Manifest
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.lifecycleScope
import com.actimapyPack.actimapy.data.model.UpsertProfileRequest
import com.actimapyPack.actimapy.data.network.RetrofitHelper
import com.actimapyPack.actimapy.data.network.TokenStore
import com.actimapyPack.actimapy.databinding.ActivityMainBinding
import com.auth0.android.Auth0
import com.auth0.android.authentication.AuthenticationException
import com.auth0.android.callback.Callback
import com.auth0.android.provider.WebAuthProvider
import com.auth0.android.result.Credentials
import kotlinx.coroutines.launch

class MainActivity : AppCompatActivity() {

    override fun attachBaseContext(newBase: Context) {
        super.attachBaseContext(LocaleHelper.wrap(newBase))
```

```
}

private lateinit var binding: ActivityMainBinding
private lateinit var account: Auth0

private val notifPermLauncher = registerForActivityResult(
    ActivityResultContracts.RequestPermission()
) { /* granted/denied */ }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    account = Auth0.getInstance(
        getString(R.string.com_auth0_client_id),
        getString(R.string.com_auth0_domain)
    )

    requestNotificationPermissionIfNeeded()
    binding.buttonLogin.setOnClickListener { login() }
}

private fun requestNotificationPermissionIfNeeded() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        if (checkSelfPermission(Manifest.permission.POST_NOTIFICATIONS)
            != PackageManager.PERMISSION_GRANTED
        ) {
            notifPermLauncher.launch(Manifest.permission.POST_NOTIFICATIONS)
```

```
        }
    }
}

private fun login() {
    WebAuthProvider.login(account)
        .withScheme(getString(R.string.com_auth0_scheme))
        .withAudience("actimapy")
        .start(this, object : Callback<Credentials, AuthenticationException> {

            override fun onSuccess(result: Credentials) {
                val user = User(result.idToken)
                TokenStore.accessToken = result.accessToken
                TokenStore.userId      = user.id
                TokenStore.role        = parseJwtClaim(result.accessToken, "role")

                userIsAuthenticated = true

                AppPreferences.setAccessToken(this@MainActivity,
result.accessToken)
                AppPreferences.setUserId(this@MainActivity, user.id)

                if (BuildConfig.DEBUG) {
                    Log.d("AUTH", "Logged in: sub=${user.id},
role=${TokenStore.role}")
                }

                syncProfile(user)
            }
```

```
            override fun onFailure(error: AuthenticationException) {
                Log.e("AUTH", "Login failed", error)
                Toast.makeText(this@MainActivity, "Помилка входу",
Toast.LENGTH_SHORT).show()
            }
        })
    }

    // Upsert профілю в БД після успішного логіну
    private fun syncProfile(user: User) {
        lifecycleScope.launch {
            try {
                RetrofitHelper.api.upsertProfile(
                    UpsertProfileRequest(
                        email = user.email,
                        name  = user.name.ifBlank { user.email }
                    )
                )
            } catch (e: Exception) {
                Log.e("AUTH", "syncProfile failed: ${e.message}")
            }
            navigateToMain()
        }
    }

    private fun navigateToMain() {
        startActivity(Intent(this, MainScreen::class.java))
        finish()
    }
```

```
    // Витягує claim з payload JWT (лише для UI, без верифікації підпису)
    private fun parseJwtClaim(token: String, claim: String): String? {
        return try {
            val payload = token.split(".").getOrNull(1) ?: return null
            val json = String(
                android.util.Base64.decode(
                    payload,
                    android.util.Base64.URL_SAFE or android.util.Base64.NO_PADDING
                )
            )
            org.json.JSONObject(json).optString(claim).ifEmpty { null }
        } catch (e: Exception) {
            Log.e("AUTH", "JWT parse failed: ${e.message}")
            null
        }
    }
}
```

Карта активностей - MapsFragment.kt (ключові методи)

```
// Завантаження маркерів з геофільтром або без (якщо GPS недоступний)
private fun loadMarkers() {
    val loc = getDeviceLatLon()
    lifecycleScope.launch {
        try {
            val items: List<ActivityMapItem> = if (loc != null) {
                val resp = RetrofitHelper.api.getNearbyActivities(
                    loc.first, loc.second, activeRadiusM
                )
                if (resp.isSuccessful) {
                    resp.body()?.mapNotNull { it.toMapItem() } ?: emptyList()
                } else emptyList()
            } else {
                val resp = RetrofitHelper.api.getActivityMapItems()
                if (resp.isSuccessful) resp.body() ?: emptyList()
                else emptyList()
            }
            allItems = items
            applyFiltersAndRender()
        } catch (e: Exception) {
            Log.e("MAPS", "loadMarkers error", e)
        }
    }
}

// Клієнтська фільтрація: пошук + тег
private fun applyFiltersAndRender() {
    filterJob?.cancel()
    filterJob = lifecycleScope.launch {
```

```
        try {
            val afterSearch = if (searchQuery.isBlank()) allItems
            else allItems.filter {
                it.name.contains(searchQuery, ignoreCase = true) ||
                it.locationName?.contains(searchQuery, ignoreCase = true) == true
            }
            renderMarkers(applyTagFilter(afterSearch))
        } catch (e: Exception) {
            Log.e("MAPS", "applyFiltersAndRender error", e)
        }
    }
}

// Отримання найточнішої доступної геолокації
private fun getDeviceLatLon(): Pair<Double, Double>? {
    if (!hasLocationPermission()) return null
    val lm = requireContext().getSystemService<LocationManager>() ?: return null
    var best: Location? = null
    for (provider in listOf(LocationManager.GPS_PROVIDER,
LocationManager.NETWORK_PROVIDER)) {
        val loc = try {
            lm.getLastKnownLocation(provider)
        } catch (_: SecurityException) { null }
        if (loc != null && (best == null || loc.accuracy < best.accuracy)) best = loc
    }
    return best?.let { it.latitude to it.longitude }
}

// Рендер кластерів на карті
private fun renderMarkers(items: List<ActivityMapItem>) {
```

```
    if (_binding == null || !::clusterManager.isInitialized) return
    clusterHandler.removeCallbacksAndMessages(null)
    clusterManager.clearItems()
    if (items.isEmpty()) {
        scheduleCluster()
        binding.cardEmptyState.visibility = View.VISIBLE
        return
    }
    binding.cardEmptyState.visibility = View.GONE
    val clusterItems = items.map { ActivityClusterItem(it) }
    clusterManager.addItems(clusterItems)
    scheduleCluster()
    val boundsBuilder = LatLngBounds.Builder()
    clusterItems.forEach { boundsBuilder.include(it.position) }
    if (items.size == 1) {

mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(clusterItems[0].posit
ion, 14f))
    } else {
        try {
            mMap.animateCamera(
                CameraUpdateFactory.newLatLngBounds(boundsBuilder.build(), 120)
            )
        } catch (_: Exception) { }
    }
}
```

SQL-скрипти бази даних PostgreSQL

Увімкнення розширень та допоміжна функція JWT

```
-- Розширення
CREATE EXTENSION IF NOT EXISTS pgcrypto;   -- bcrypt, gen_random_uuid()
CREATE EXTENSION IF NOT EXISTS postgis;    -- просторові типи та функції


-- Зчитування user_id з JWT-токена Auth0
CREATE OR REPLACE FUNCTION auth.uid() RETURNS text
LANGUAGE sql STABLE AS
$$
  SELECT current_setting('request.jwt.claims', true)::json->>'sub';
$$;
```

Хешування пароля активності при INSERT/UPDATE

```
CREATE OR REPLACE FUNCTION hash_activity_password()
RETURNS TRIGGER LANGUAGE plpgsql AS
$$
BEGIN
  IF NEW.password IS NOT NULL AND NEW.password <> '' THEN
    NEW.password := crypt(NEW.password, gen_salt('bf'));
  END IF;
  RETURN NEW;
END;
$$;
```

RPC upsert_profile - створення або пропуск профілю

```
CREATE OR REPLACE FUNCTION upsert_profile(
    p_email text,
    p_name  text
) RETURNS void
LANGUAGE plpgsql SECURITY DEFINER AS
$$
```

```
DECLARE
  v_uid text := auth.uid();
BEGIN
  INSERT INTO app_user(user_id, user_email, user_name)
  VALUES (v_uid, p_email, p_name)
  ON CONFLICT (user_id) DO NOTHING;
END;
$$;
```

RPC check_activity_password - перевірка пароля

```
CREATE OR REPLACE FUNCTION check_activity_password(
    act_id  uuid,
    entered text
) RETURNS boolean
LANGUAGE sql STABLE SECURITY DEFINER AS
$$
  SELECT CASE
    WHEN password IS NULL OR password = '' THEN true
    ELSE password = crypt(entered, password)
  END
  FROM activity
  WHERE activity_id = act_id;
$$;
```

Row-Level Security (RLS) - основні політики

```
-- Увімкнення RLS
ALTER TABLE activity            ENABLE ROW LEVEL SECURITY;
ALTER TABLE activity_participant ENABLE ROW LEVEL SECURITY;
ALTER TABLE host                ENABLE ROW LEVEL SECURITY;
ALTER TABLE message             ENABLE ROW LEVEL SECURITY;
ALTER TABLE review              ENABLE ROW LEVEL SECURITY;
ALTER TABLE notification        ENABLE ROW LEVEL SECURITY;
```

```
-- Активності: читати можуть всі, змінювати — лише хост
CREATE POLICY activity_select ON activity
  FOR SELECT USING (status = 'approved');

CREATE POLICY activity_insert ON activity
  FOR INSERT WITH CHECK (auth.uid() IS NOT NULL);

CREATE POLICY activity_update ON activity
  FOR UPDATE USING (
    EXISTS (
      SELECT 1 FROM host
      WHERE host.activity_id = activity.activity_id
        AND host.user_id = auth.uid()
    )
  );

-- Учасники: читати може будь-хто авторизований
CREATE POLICY participant_select ON activity_participant
  FOR SELECT USING (auth.uid() IS NOT NULL);

CREATE POLICY participant_insert ON activity_participant
  FOR INSERT WITH CHECK (user_id = auth.uid());

CREATE POLICY participant_delete ON activity_participant
  FOR DELETE USING (
    user_id = auth.uid()
    OR EXISTS (
      SELECT 1 FROM host
      WHERE host.activity_id = activity_participant.activity_id
```

```
      AND host.user_id = auth.uid()
    )
  );

-- Повідомлення: лише учасники та хост бачать чат активності
CREATE POLICY message_select ON message
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM activity_participant p
      WHERE p.activity_id = message.activity_id
        AND p.user_id = auth.uid()
    )
    OR EXISTS (
      SELECT 1 FROM host h
      WHERE h.activity_id = message.activity_id
        AND h.user_id = auth.uid()
    )
  );

CREATE POLICY message_insert ON message
  FOR INSERT WITH CHECK (user_id = auth.uid());

-- Сповіщення: лише власні
CREATE POLICY notification_own ON notification
  FOR ALL USING (user_id = auth.uid());

-- Відгуки: писати може лише учасник, що не є хостом
CREATE POLICY review_insert ON review
  FOR INSERT WITH CHECK (
    reviewer_id = auth.uid()
```

```
  AND EXISTS (
    SELECT 1 FROM activity_participant p
    WHERE p.activity_id = review.activity_id
      AND p.user_id = auth.uid()
  )
  AND NOT EXISTS (
    SELECT 1 FROM host h
    WHERE h.activity_id = review.activity_id
      AND h.user_id = auth.uid()
  )
);
```

Додаток Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Конєвєга Захар Сергійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система для підтримки соціальної взаємодії користувачів на основі геолокаційних сервісів» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«____»_____2026 р. _____ **Захар КОНЄВЄГА**