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: «Програмне забезпечення сервісу обліку робочого часу бригад»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи к.т.н., с.н.с.	_____ (підпис)	**Юлія БОЯРІНОВА**
Виконав	_____ (підпис)	**Богдан КОВАЛЕНКО**

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

Коваленку Богдану Ігоровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення сервісу обліку робочого часу бригад»

затверджена наказом від «14» квітня 2026 р. №1026 «С»
Термін подання завершеної роботи на кафедру «22 травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області обліку робочого часу бригад та визначення проблем ручної фіксації змін на віддалених робочих об'єктах.
2. Дослідження існуючих програмних рішень для обліку робочого часу, GPS-контролю, геозон, мобільного табелювання та формування звітності.
3. Формування функціональних і нефункціональних вимог до сервісу обліку робочого часу бригад з урахуванням ролей бригадира та працівника.
4. Проєктування інформаційного забезпечення системи, зокрема логічної та фізичної структури бази даних для збереження користувачів, бригад, робочих об'єктів, QR-кодів, змін і подій сканування.
5. Розроблення клієнт-серверної архітектури програмного забезпечення, Android-застосунку, серверної частини, механізму QR-фіксації, перевірки геолокації та локального збереження подій.
6. Тестування основних функцій системи, визначення вимог до впровадження, експлуатації та складу інсталяційного пакету.

Перелік графічних документів (за необхідності).

_____

Дата видачі завдання «16»вересня 2025 р.

Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Юлія БОЯРІНОВА**
Завдання взяв до виконання	_____ (підпис)	**Богдан КОВАЛЕНКО**

ЗМІСТ

ЗМІСТ 3
ВСТУП 5
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1 Опис предметної області 8
1.2 Огляд інформаційних джерел та існуючих рішень 10
1.3 Вимоги до розроблюваної системи 15
1.4 Моделювання предметної області 19
1.5 Постановка завдання 26
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 28
2.1 Логічна модель даних 28
2.2 Вибір системи управління базами даних 31
2.3 Розробка структури інформаційної бази 32
2.4 Схема та фізична структура бази даних 39
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 44
3.1 Абстракції предметної області 44
3.2 Моделювання структури та взаємодії об’єктів 49
3.4 Компонентна структура системи 56
3.5 Вибір інструментарію для створення прикладного програмного забезпечення 62
3.6 Алгоритмізація та програмування програмних модулів 63
4.1 Тестування систем 72
4.2 Вимоги до апаратного та програмного забезпечення 83
4.3 Склад інсталяційного пакету 86
ВИСНОВКИ 89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 91
ДОДАТКИ 94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - інтерфейс програмування застосунків

APK - інсталяційний файл Android-застосунку

GPS - глобальна система позиціювання

MVVM - архітектурний шаблон Model–View–ViewModel

ORM - об’єктно-реляційне відображення

QR - двовимірний матричний код швидкого відгуку

REST - архітектурний стиль взаємодії вебсервісів

Room - бібліотека локального збереження даних Android Jetpack

SQL - мова програмування запитів

СУБД - система керування базами даних

Offline-first (офлайн-перший підхід) - це архітектура розробки, де інтернет використовується лише для фонової синхронізації, а базовий функціонал працює локально.

ВСТУП

Практична цінність теми в тому, що для невеликих і середніх підрядних бригад контроль робочого часу потребує великих витрат, дисципліни, доброчестності виконання робіт та коректності внутрішньої звітності. Якщо відмітки про початок і завершення роботи ведуться вручну, керівник бригади змушений витрачати додатковий час на звіряння даних, а в результаті отримує менш надійні дані для управлінських рішень.

Особливість теми полягає у поєднанні кількох технологічних механізмів у єдине прикладне рішення. QR-код використовується для швидкої ідентифікації об’єкта, геолокація - для перевірки фактичного перебування працівника в допустимій зоні робочого об’єкту, а локальне сховище - для накопичення подій, у разі нестабільного інтернет-з’єднання. Разом ці засоби механізми створюють застосунок, орієнтований на реальні польові умови роботи.

На відміну від універсальних систем трекінгу часу, які орієнтовані на офісний формат, загальне табелювання без прив’язки до місця виконання робіт, складний інтерфейс розроблювана система враховує специфіку саме виїзних малих та середніх робочих бригад. Принципово важливими є координати об’єкта, оперативна фіксація зміни без зайвих дій працівника та можливість отримати підсумковий звіт в зручному для бригадира вигляді.

Окремої уваги потребує відсутність інтернет-з’єднання. Значна частина робочих об’єктів розташовується там, де мобільне покриття нестійке або відсутнє. За таких умов застосунок повинен не втрачати події, а коректно зберігати їх на пристрої з подальшою синхронізацією із сервером, саме цей механізм відрізняє застосунок.

Науково-практична новизна роботи полягає не у винайденні нового способу табелювання, а в поєднанні сучасних мобільних та серверних засобів у межах одної системи, адаптованої до потреб робочих бригад. Цінність результату роботи застосунка визначається тим, що реалізовані механізми придумані для практикичного застосування невеликих та середніх бригад, які мають порівняно малі здібності у застосуванні програмних застосунків.

Ручне ведення звітності породжує кілька типових проблем. Насамперед бригадир витрачає час на перевірку відміток і перенесення інформації в підсумкові таблиці. Крім того, зростає ймовірність розбіжностей у часі початку та завершення роботи, дублювання записів та втрати частини даних. За відсутності геоприв’язки доволі складно і ресурсозатратно підтвердити, що працівник дійсно перебував у межах потрібного об’єкта.

Мета роботи заключається у розробленні програмного забезпечення сервісу обліку робочого часу бригад, яке забезпечує фіксацію відкриття і закриття змін за допомогою QR-коду та геолокації, підтримує локальне збереження подій і надає зручні засоби формування звітності.

Для досягнення поставленої вище мети треба проаналізувати і реалізувати пункти нижче:

- дослідити предметну область;
- сформувати функціональні й нефункціональні вимоги;
- побудувати UML-моделі та модель даних;
- обґрунтувати вибір технологій;
- реалізувати мобільний клієнт і серверну частину;
- спроєктувати базу даних;
- провести тестування та підготувати рекомендації щодо інсталяції і розгортання застосунка.

Об’єктом дослідження є процес обліку робочого часу працівників робочих бригад. Предметом дослідження є методи, моделі та програмні засоби автоматизації фіксації змін, перевірки перебування на об’єкті, локального збереження подій і синхронізації даних у клієнт-серверній системі.

У виконаній роботі застосовано наступне: методи системного аналізу, об’єктно-орієнтованого проєктування, UML-моделювання, реляційного проєктування даних, мобільної та серверної розробки а також сценарного функціонального тестування. Практична реалізація програмного забезпечення виконана з допомогою: Kotlin, Jetpack Compose, Room, WorkManager, FastAPI, SQLAlchemy та PostgreSQL.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сервіс обліку робочого часу бригад призначений для автоматизації фіксації відвідування працівниками робочих об’єктів, тобто контролю їх фактичної присутності на місці виконання робіт та формування звітності. Така система є актуальною для бригад, у яких працівники виконують завдання не тільки в межах одного місця, а на багатьох об’єктах. До таких сфер належать будівництво, монтажні та сервісні роботи, технічне обслуговування, виїзна інженерна діяльність, ремонтні та польові роботи.

У більшості випадків облік робочого часу бригад ведеться вручну або напівручним методом. Для цього використовують паперові журнали, телефонні дзвінки, повідомлення у месенджерах і електронні таблиці. Такий підхід не дозволяє достовірно підтвердити, що працівник дійсно прибув на потрібний об’єкт і розпочав роботу саме там, де було заплановано. Крім того, ручний облік створює ризик помилок, дублювання записів, зміщення дат і втрати частини інформації, махінацій а головне , що підготовка зведеної звітності потребує велику кількість часу і ресурсів бригадира.

У межах предметної області можна виділити дві основні ролі користувачів - бригадир і працівник. Бригадир відповідає за організацію роботи на об’єктах, створює записи про робочі об’єкти, визначає або уточнює їх місцезнаходження, генерує QR-коди, контролює історію змін і формує підсумкові звіти. Працівник використовує систему для входу до облікового запису, сканування QR-коду, відкриття та закриття зміни,

перегляду власної історії змін, сумарно відпрацьованого часу, а також контактних даних бригадира.

Першою ключовою сутністю предметної області є робочий об’єкт. Він характеризується назвою, адресою, координатами місця виконання робіт та допустимим радіусом перевірки. Саме до нього прив’язується QR-код, який використовується для швидкої і точної ідентифікації місця роботи. Наявність координат об’єкта дозволяє системі не лише реєструвати сам факт сканування коду, а й перевіряти, чи справді працівник перебуває в допустимій зоні біля цього об’єкта.

Другою ключовою сутністю є зміна. Зміна відкривається у момент сканування коду, а це означає початок роботи працівника на конкретному об’єкті й закривається після сканування та завершення робіт. Для кожної зміни необхідно зберігати час відкриття, час закриття, статус, прив’язку до працівника, прив’язку до об’єкта, а також дані, потрібні для перевірки й синхронізації. Саме на основі накопичених змін надалі формується звітність: перелік змін по працівнику, перелік змін по об’єкту, сумарний час роботи, а також узагальнені звіти для бригадира.

Важливою особливістю предметної області є необхідність роботи за умов нестабільного або відсутнього інтернет-з’єднання. У реальних умовах робочі об’єкти можуть розташовуватися в місцях зі слабким мобільним покриттям, також варто не забувати в який час ми живемо – війна і блекаути. Через це система повинна підтримувати локальне збереження подій на пристрої працівника та подальшу синхронізацію із сервером після відновлення мережі[1]. Без цієї властивості сервіс був би непридатний для реального використання.

Отже, сервіс обліку робочого часу бригад є інформаційною системою, що поєднує реєстрацію користувачів, облік робочих об’єктів, контроль місця знаходження працівників, фіксацію змін, локальне зберігання подій та формування звітності. Автоматизація цих процесів дає змогу збільшити

достовірність даних, зменшити обсяг ручної роботи та автоматизувати звітність.

1.2 Огляд інформаційних джерел та існуючих рішень

Під час аналізу наявних рішень варто враховувати не лише підтримку обліку робочого часу, а наскільки рішення відповідає реальному сценарію роботи бригади. Для виїзного формату необхідні геоприв’язка до об’єкта, простота реєстрації зміни, компакність, зрозуміла звітність і можливість роботи при нестабільному інтернет-з’єднанні.

На ринку вже існують універсальні системи обліку часу, мобільні сервіси з GPS-моніторингом і корпоративні платформи табелювання. Однак більшість із них або орієнтована на офісну модель роботи, або передбачає інший формат впровадження, а не тий, що потрібен бригаді з обмеженими ресурсами. В основному системи надають дуже поглиблену аналітику, але не пропонують швидкого механізму фіксації початку та завершення робіт безпосередньо на фізичному об’єкті.

Додатковим критерієм оцінювання є довіра до відміток. У частині сервісів достатньо натиснути кнопку початку роботи, але така дія сама по собі не доводить реальної присутності. Для нашої предметної області цього недостатньо, тому потрібне поєднання ідентифікації об’єкта через QR-код і перевірки геолокації.

Порівняння ручного табелювання з готовими SaaS-продуктами натякає на висновок. Паперові та напівручні підходи прості, але ненадійні та трудомісткі. Хмарні сервіси забезпечують зручну аналітику, однак не завжди покривають специфіку локальних об’єктів і в основному не переносять проблем з інтернет-з’єднанням.

Перевагою готових продуктів це продвинута інфраструктура, мобільні застосунки і розвинені звітні можливості. Разом із тим вони вимагають дорогої підписки, містять функції, надлишкові для типових бригад.

Jibble

Jibble є хмарною системою обліку робочого часу та присутності, яка підтримує мобільне і веб-табелювання, геолокацію, QR-ідентифікацію, офлайн-фіксацію відміток та побудову табелів і звітів а також робота без інтернет-з'єднання з подальшим збереженням даних.[1]

На рисунку 1.1 представлено головну сторінку конкуретна Jibble

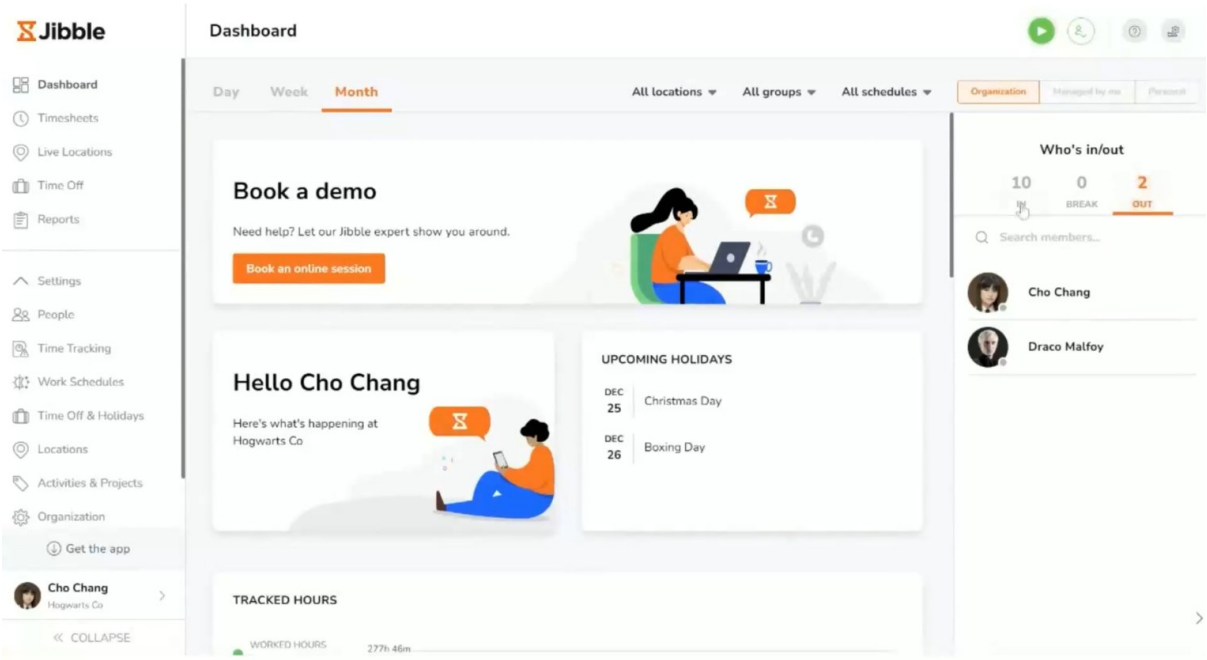


Рис 1.1 інтерфейс системи Jibble

Переваги:

- підтримує QR-відмітку часу;
- система підтримує геолокацію, що дозволяє обмежити відмітку робочого часу конкретним місцем;[2]
- деталізовані робочі табелі та експорт звітів;
- підтримується офлайн-робота з подальшим збереженням відміток;[3]

- базовий функціонал є безкоштовним для необмеженої кількості користувачів.

  Недоліки:

- система не орієнтована на польові робочі або будівельни об’єкти;
- сервіс зосереджений на табелюванні та звітах, тоді як у розроблюваній системі важливою є прив’язка до конкретного робочого об’єкта, який самостійно створює бригадир;
- для бригади надлишковий функціонал такого рішення ускладняє інтерфейс і впровадження - прості робітники просто не розберуться.

ClockShark

ClockShark - програмне рішення для обліку часу польових працівників, яке орієнтоване насамперед на будівельні та сервісні команди.[4] Мобільнитй застосунок підтримує відкриття та закриття зміни, GPS-фіксацію, побудову маршруту працівника протягом дня, роботу без покриття мережі та подальшу синхронізацію після повернення інтернет-з’єднання. Окремо підкреслюється використання GPS Time Clock, це зв’язок відпрацьованого часу з конкретними роботами та підтримка офлайн-режиму.

На рисунку 1.2 представлено інтерфейс ClockShark

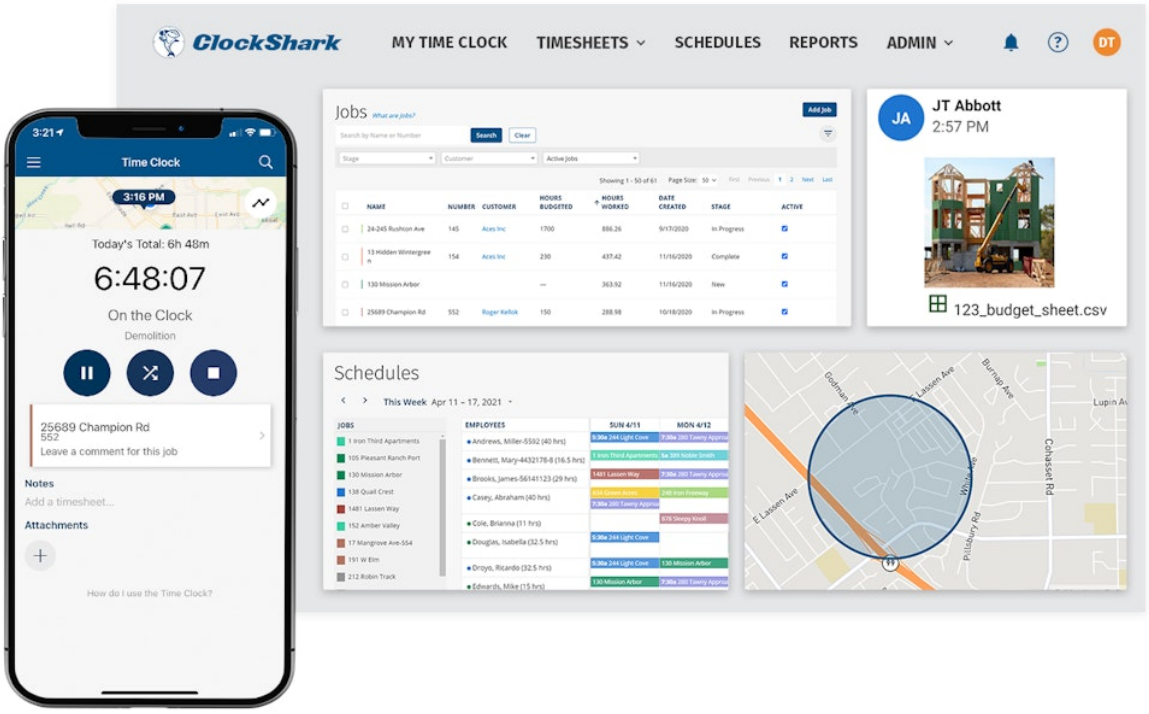


Рис 1.2 інтерфейс системи ClockShark

Переваги:

- система орієнтована на виїзні бригади та польових працівників;[5]
- підтримується GPS-фіксація початку, завершення і процесу роботи;
- наявна підтримка офлайн-обліку з подальшою передачею даних на сервер;[6].
- можна прив’язувати відпрацьований час до конкретних робіт або об’єктамів, що дозволяє контролювати трудовитрати.

Недоліки:

- основний акцент зроблено на GPS-табелюванні та відстеженні місцеположення, а не на сценарії швидкої фіксації зміни через QR-код;
- безперервне GPS-спостереження є надлишковими для задач малих бригад, яким потрібно лише підтвердження факту прибуття працівника на місце роботи;

- ціна є головним мінусом, для малих бригад вона дуже агресивна, та застосунок не має Української локалізації.

Connecteam

Connecteam є багатофункціональною платформою для управління персоналом, у складі якої є модуль Time Clock.[7] Підтримка мобільної відмітки часу, цифрових геозон, GPS-фіксації місця входу та виходу, автоматичного закриття зміни після виходу із заданої геозони, а також облік часу за виконаною роботою, проєктом або клієнтом.[8] Зазначено, що система може зберігати локаційні штампи входу/виходу і маршрут переміщення працівника.

На рисунку 1.3 представлено інтерфейс Connecteam

Рис 1.3 інтерфейс системи Connecteam

Переваги:

- відмітка часу з мобільного застосунку;

- геозона дозволяє обмежувати вхід і вихід із зміни лише визначеним місцем;
- система може автоматично завершувати зміну після виходу працівника за межі робочої зони;
- час можна відносити до певної роботи, проєкту або клієнта;
- розширена звітність і додаткові механізми управління мобільними командами.

Недоліки:

- система є завеликою для малих бригад, її функціональність надмірна;
- залежність від точності геолокації та налаштувань мобільного пристрою ускладнює використання в польових умовах, особливо за слабкого GPS-сигналу; наприклад, коли користувачі не можуть відмітитися навіть усередині геозони, якщо некоректно налаштована точна локація;
- можливість зберігати маршрути переміщення працівника для обліку робочого часу бригади надлишкова і делікатна з точки зору приватності, адже працівники завжди вільно пересуваються.

Тобто головна перевага запропонованого застосунока – це орієнтація на мінімальний набір можливостей для малої або середньої бригади, а саме: реєстрацію користувачів, облік об’єктів, QR-фіксацію змін, геолокаційну перевірку, офлайн-накопичення подій та формування простих звітів, нативно зрозумілий інтерфейс.

1.3 Вимоги до розроблюваної системи

На основі аналізу: предметної області, робочого процессу бригад та недоліків існуючих підходів до обліку робочого часу, сформовано низку

вимог до розроблюваної програмної системи. Вони визначають функціональні можливості програмного застосунку, умови його використання, особливості взаємодії користувачів із системою, а також вимоги до якості.

Розроблюване програмне забезпечення повинно забезпечувати автоматизацію обліку робочого часу працівників, які виконують роботи на визначених бригадиром об’єктах, контроль фактичної присутності працівника на місці виконання робіт в кінці та на початку зміни, накопичення історії змін та формування підсумкової звітності для перегляду. На відміну від загальних систем обліку часу, у даній роботі акцент робиться саме на мобільному сценарії використання, геолокаційній прив’язці до робочого об’єкта та можливості відмічатися за умов нестабільного інтернет-з’єднання.

У застосунку передбачаються дві основні ролі користувачів: бригадир і працівник. Кожна з цих ролей має свої доступні дії, тому програмна система повинна підтримувати розмежування прав доступу відповідно до ролей.

До функціональних вимог системи належать:

- реєстрація користувачів із прив’язкою ролі;
- авторизація в системі за логіном і паролем;
- перегляд та редагування профілю;
- збереження персональних даних користувача, зокрема прізвища, імені, контактного номера телефону, посади та ролі;
- створення бригадиром нового робочого об’єкта;
- введення та збереження атрибутів робочого об’єкта, а саме назви, адреси, координат місця розташування та допустимого радіуса перевірки;
- генерування QR-коду для кожного робочого об’єкта;

- перегляд створених робочих об’єктів;
- можливість поширення QR-коду об’єкта;
- сканування QR-коду працівником;
- відкриття зміни після успішного сканування QR-коду та перевірка місцеположення;
- збереження сканування у локальному сховищі;
- синхронізація накопичених локально сканувать із сервером;
- закриття зміни після повторного сканування;
- збереження часу відкриття та закриття зміни;
- збереження статусу зміни;
- ведення історії змін для окремо для кожного працівника;
- розрахунок сумарно відпрацьованого часу;
- формування звітів;
- відображення для працівника контактних даних бригадира об’єкта;
- завершення сеансу роботи користувача із системою.

В системі має бути механізм перевірки коректності введених даних. Це означає, що обов’язкові поля не повинні залишатися порожніми, номер телефону повинен бути в допустимому форматі, а текстові поля, що описують посаду, не повинні приймати некоректні значення. Такі обмеження є важливими не лише з точки зору зручності, а й для забезпечення цілісності інформації в системі.

До нефункціональних вимог належать вимоги, що визначають якість:

- зрозумілий інтерфейс користувача;
- швидке виконання основних операцій, а саме входу до системи, відкриття зміни, перегляду змін та формування звіту;
- працювати на Android-пристроях, які відповідають мінімальним технічним вимогам;
- стабільна робота в умовах відсутності інтернет-з’єднання;

- збереження локально до їх синхронізації із сервером;
- розмежування доступу до функцій системи відповідно до ролі користувача;
- читабельне та зручне подання звітної інформації;
- достатній рівень надійності;
- базовий рівень захисту даних користувача.

Важливою вимогою до системи є підтримка офлайн-сценарію. Для предметної області, яка розглядається в роботі, ця вимога є основною, оскільки робочі об’єкти часто знаходяться в місцях зі слабким мобільним покриттям. За таких умов система не повинна втрачати можливість фіксації подій. Вона має зберігати інформацію локально, а після появи зв’язку виконувати синхронізацію без дублювання або втрати даних.

До окремої групи відносяться вимоги до інформації, що зберігаються та обробляється в системі. А саме у базі даних та локальному сховищі повинні бути:

- дані про користувачів системи;
- дані про ролі користувачів;
- дані про робочі об’єкти;
- координати місця виконання робіт;
- QR-ідентифікатори об’єктів;
- відомості про зміни;
- часові параметри відкриття та закриття змін;
- відомості про геолокаційні перевірки;
- інформація про локально збережені події, що очікують синхронізації;
- дані для формування підсумкових звітів.

Не меншу увагу слід приділити вимогам до звітності, оскільки саме звітна інформація є одним із ключових результатів успішного функціонування системи, тобто:

- перегляд історії змін конкретного працівника;

- визначення сумарного відпрацьованого часу;
- формування переліку змін за конкретним об’єктом;
- формування зведеного звіту для бригадира;
- відображення часу початку та завершення роботи;
- відображення тривалості роботи по кожній зміні;
- можливість передачі або поширення звіту в зручному для користувача вигляді.

Таким чином, сформовані вимоги охоплюють як функціональні можливості системи, так і якісні характеристики її роботи. Надалі вимоги виступають основою для побудови логічної моделі даних, проєктування структури програмного забезпечення, визначення архітектури взаємодії клієнтської та серверної частин, а також реалізації конкретних програмних модулів. Їх визначення на ранньому етапі дозволяє зменшити ризик помилок під час подальшого проєктування та забезпечує якість і відповідність програмного продукту потребам.

1.4 Моделювання предметної області

Моделювання предметної області - це етап переходу від загального опису проблеми до подання майбутнього програмного застосунка. Воно надає змогу впорядкувати уявлення про ролі користувачів, функції системи, послідовність виконання операцій та взаємодію між окремими елементами сервісу. У межах розробки моделювання використовується для покращення логіки обліку робочого часу бригад, а також для подальшого успішного переходу до проєктування структури програмного забезпечення.

У предметній області застосунка було виділено дві основні ролі: працівник і бригадир. Працівник взаємодіє із системою під час фіксації

власного робочого часу на об’єкті, тоді як бригадир організує роботу бригади, створює об’єкти і контролює результати роботи. Окрім ролей користувачів є наступні сутності: робочий об’єкт, QR-код, геолокаційні дані, локальне сховище та запис про зміну.

Для опису цих сутностей у використано UML-діаграми, які дозволяють оглянути предметну область з різних точок зору. Діаграма прецедентів відображає перелік функцій, доступних окремим акторам, відповідно до їх ролей. Діаграма активності показує алгоритм виконання ключовогих процесів з перевіроками та альтернативниими гілоками. Діаграма послідовності використовується для відображення послідовної логіки взаємодії учасників системи у межах конкретного сценарію. Сукупність цих моделей забезпечує достатній рівень якості предметної області для успішного переходу до наступних етапів проєктування.

1.4.1 Діаграма прецедентів. Діаграма дозволяє визначити, які саме дії виконує кожен учасник процесу, а також показати функціональні межі програмного продукту.[28]

На рисунку 1.4 представлено діаграму прецедентів

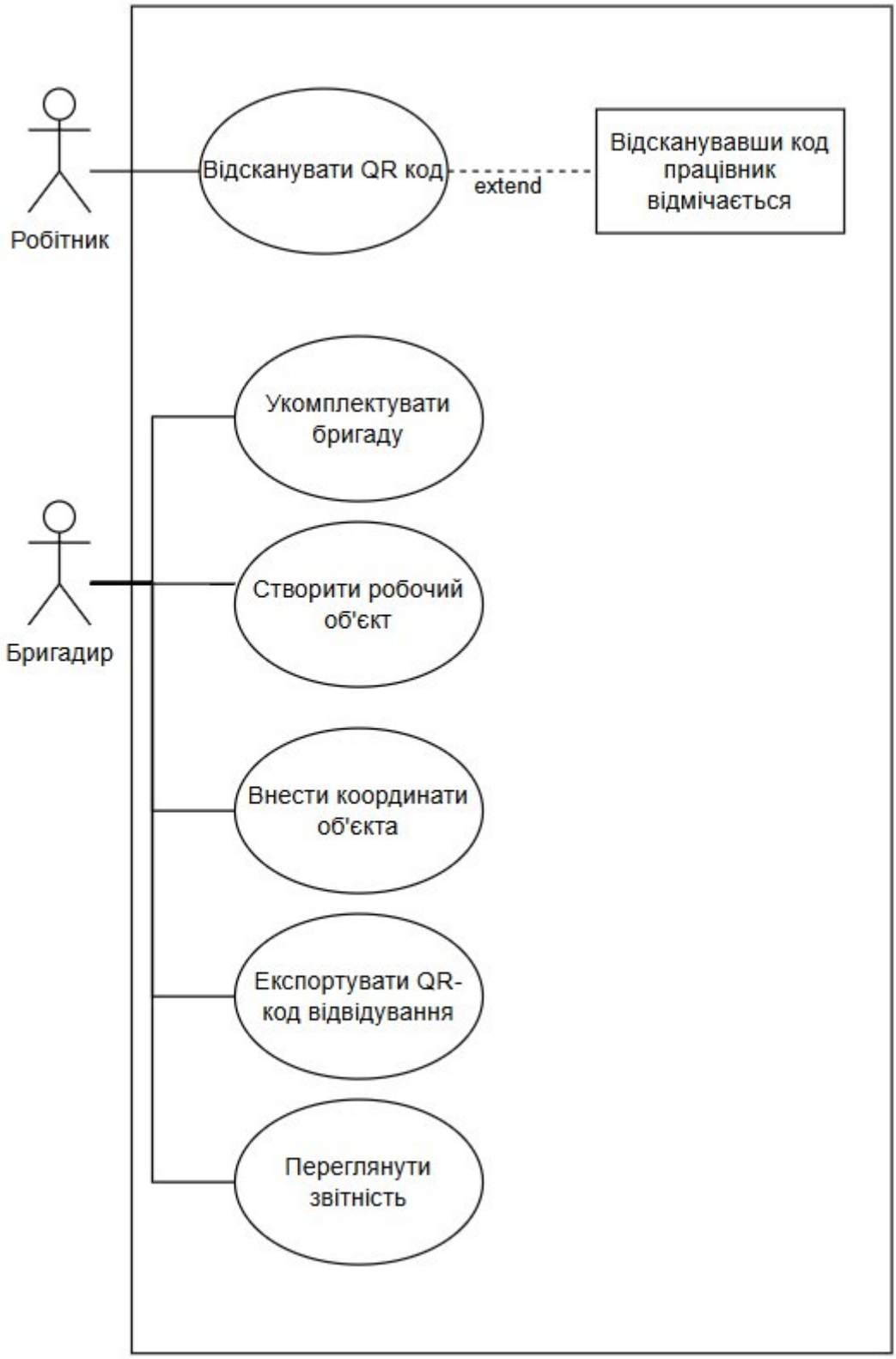


Рис 1.4 Діаграма прецедентів

На рис 1.4 відображено двох акторів: робітника та бригадира. Робітник взаємодіє із системою через сканування QR-коду. У межах цього сценарію сканування коду - дія, після якої виконується відмітка працівника в системі.

Бригадир взаємодіє із системою значно шриже, бо в його потреби входить попереднья організаця об’єкта. На діаграмі є такі прецеденти, як укомплектування бригади, створення робочого об’єкта, внесення координат об’єкта, експортування QR-коду відвідування та перегляд звітності. Вона відображають дії бригадира, які створюють умови для фіксації початку та кінця зміни.

Таким чином, діаграма прецедентів показує, що робітник у системі виконує операцію сканування, а бригадир - налаштування, організацію та звітність. Це дозволяє розмежувати відповідальність користувачів і визначити основну суть функціонування застосунка.

1.4.2 Діаграма активності. Діаграма активності застосовується для моделювання процесу відмітки працівника, а також для відображення підготовки бригадира.[29]

На рисунку 1.5 представлено діаграму активності

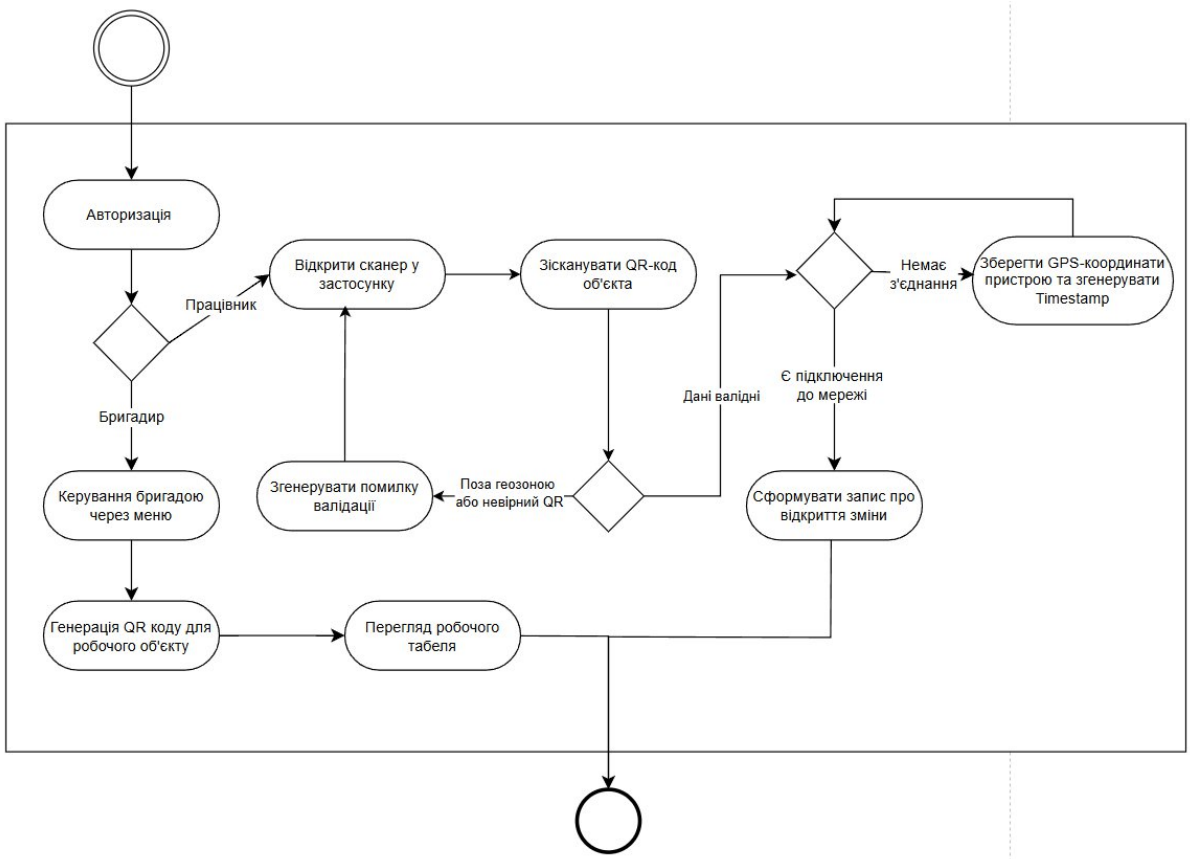


Рис 1.5 Діаграма активності

На початку процесу відбувається авторизація користувача в системі. Після чого відбувається розгалуження за ролями. Якщо в систему входить бригадир - він переходить до функцій керування бригадою через меню, після чого генерує QR-код робочого об’єкта та переглядає робочий табель. А коли авторизується працівник, то відкриває сканер у мобільному застосунку та сканує QR-код. Після цього система виконує перевірку отриманих даних. У разі, якщо код є некоректним або працівник перебуває поза допустимим радіусом геолокації, відображається помилка валідації. Якщо ж дані коректні, то перевіряється наявність з’єднання.

При наявності інтернет-з’єднанні формується запис про відкриття зміни. Якщо воно відсутнє, система зберігає GPS-координати пристрою та генерує часову позначку, що дозволяє синхронізувати подію пізніше. Після

обробки інформація потрапляє до звіту, що означає успішний результат роботи застосунка.

Отже, діаграма активності показує не тільки базовий сценарій роботи працівника, а й важливі перевірки та розгалуження, пов’язані з коректністю QR-коду і розташуванням, та станом інернет-з’єднання.

1.4.3 Діаграма послідовності. Діаграма послідовності моделює сценарії відкриття і закриття зміни працівником із врахуванням перевірки QR-коду, визначення координат, локального збереження події сканування та її подальшої синхронізації.[30]

На рисунку 1.6 представлено діаграму послідовності

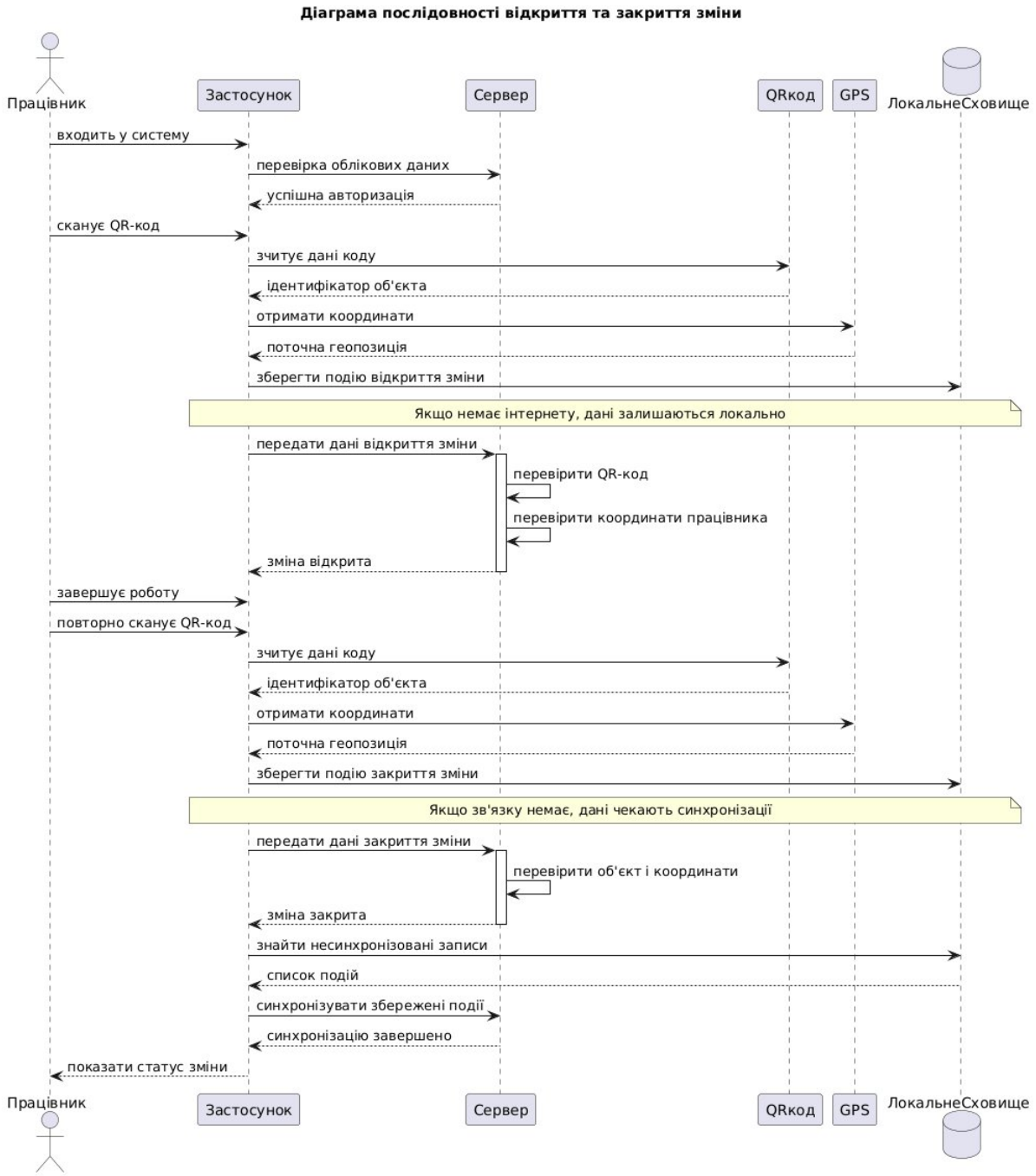


Рис 1.6 Діаграма послідовності

На рисунку 1.6 показано взаємодію між працівником, мобільний застосуноком і сервером, QR-кодом, GPS-модулем і локальним сховищем. Спочатку працівник входить до системи, після чого застосунок передає дані для перевірки на сервер. Після успішної авторизації працівник сканує QR-

код. Після чого застосунок отримує ідентифікатор об’єкта, та звіряє поточні координати з допомогою GPS-модуля.

Якщо зв’язок із сервером відсутній, дані тимчасово зберігаються локально. А коли зв’язок є, то застосунок передає зміну статусу зміни на сервер відразу. Після успішної операції сервер повертає підтвердження, що зміна відкрита.

Друга частина діаграми відображає завершення роботи працівника після повторного сканування QR-коду, застосунок знову отримує дані об’єкта, координати пристрою та формує подію закриття зміни. Ця подія також зберігається локально за відсутності інтернет-з’єднання і передається на сервер. Після перевірки сервер повертає підтвердження закриття зміни. Додатково на діаграмі показано механізм пошуку несинхронізованих записів у локальному сховищі, передачі їх на сервер і завершення синхронізації. Наприкінці працівник може переглянути актуальний статус зміни.

1.5 Постановка завдання

Постановка завдання має бути такою, щоб на її основі можна було перейти до проєктування даних і реалізації програмних модулів. Саме тому вона включає не тільки мету розробки, а й низку інформації, яка піддається обробці, бізнес-операції, що мають бути автоматизовані, а також результати, які система видає користувачу.

Потрібно побудувати систему, що складається з мобільного застосунку та серверної частини. Мобільний застосунок має забезпечувати користувачу наступне: сканування QR-коду, отримання геолокації, локальне збереження даних та перегляд історії змін. Сервер обробляє

авторизацію, коректность запитів, збереження змін, список робочих об’єктів і формування звітів.

Одним із ключів до успіху є реальність системи. Тобто, розроблене рішення повинно бути не проста теоретично обґрунтованим, а запускатися як готовий програмний застосунок, підтримвати основні ролі та задовольняти базовий сценарій: створення об’єкта бригадиром, генерація QR-коду, відкриття зміни працівником, закриття зміни працівником, формування звіту бригадиром.

Отже, рамки всієї подальшої роботи наступні: у розділі 2 буде розроблена придатна до реалізації модель даних, у третьому розділі - обґрунтована архітектура та реалізація, а четвертому - перевірка працездатності та вимоги до впровадження. Така послідовність забезпечує логічну цілісність пояснювальної записки.

Результатом розробки має бути MVP, у вигляді APK, придатний для демонстрації на реальному Android-пристрої: запуск серверної частини, авторизація користувачів, створення об’єкта бригадиром, генерація QR-коду, сканування QR-коду працівником, перевірка геолокації, відкриття і закриття зміни, а також ручний запуск синхронізації.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Важливим критерієм якості логічної моделі є відсутність дублювання даних. Це досягається завдяки виокремленню довідкових сутностей, використанню зовнішніх ключів та збереження кожного факту лише в одному місці. Тобто, параметри робочого об’єкта не дублюються всередині таблиці, а з’єднані з допомогою ідентифікатора об’єкта. Аналогічно роль користувача не потребує створення дублікату таблиць для працівника і бригадира, оскільки вона задається атрибутом ролі в межах сутності користувача.

Слід враховувати наявність нереляційності, тобто частина інформації в системі проходить як JSON-повідомлення між клієнтом і сервером та як тимчасова локальна структура в мобільному застосунку. Однак це подання використовуються лише на рівні транспортування або кешування і не суперечить реляційності основного сховища даних. Тобто, логічна модель поєднує реляційість із додатковим поданнями на рівні інтеграції.

Під час побудови ER-діаграми треба реалізувати логічні аспекти: один користувач може мати багато змін, один робочий об’єкт може бути пов’язаний із багатьма змінами, а події синхронізації логічно належать до конкретного пристрою або користувацького сценарію. Такі основні сутності: бригада, користувач, робочий об’єкт, QR-код, зміна та подія сканування. Дані сутності відповідають предметній області та дають змогу уникнути дублювання інформації.

Працівник і бригадир зберігаються в одній таблиці користувачів, а відмінність між ними задається роллю. Це спрощує авторизацію і дає змогу використовувати один механізм облікових записів. Робочий об’єкт пов’язаний з бригадою і користувачем, який його створив. QR-код належить конкретному робочому об’єкту. Зміна пов’язує працівника, робочий об’єкт і QR-код. Сканування зберігає фактичну дію працівника, координати, відстань до об’єкта та результат перевірки.

На рисунку 1.7 представлено ER-діаграму

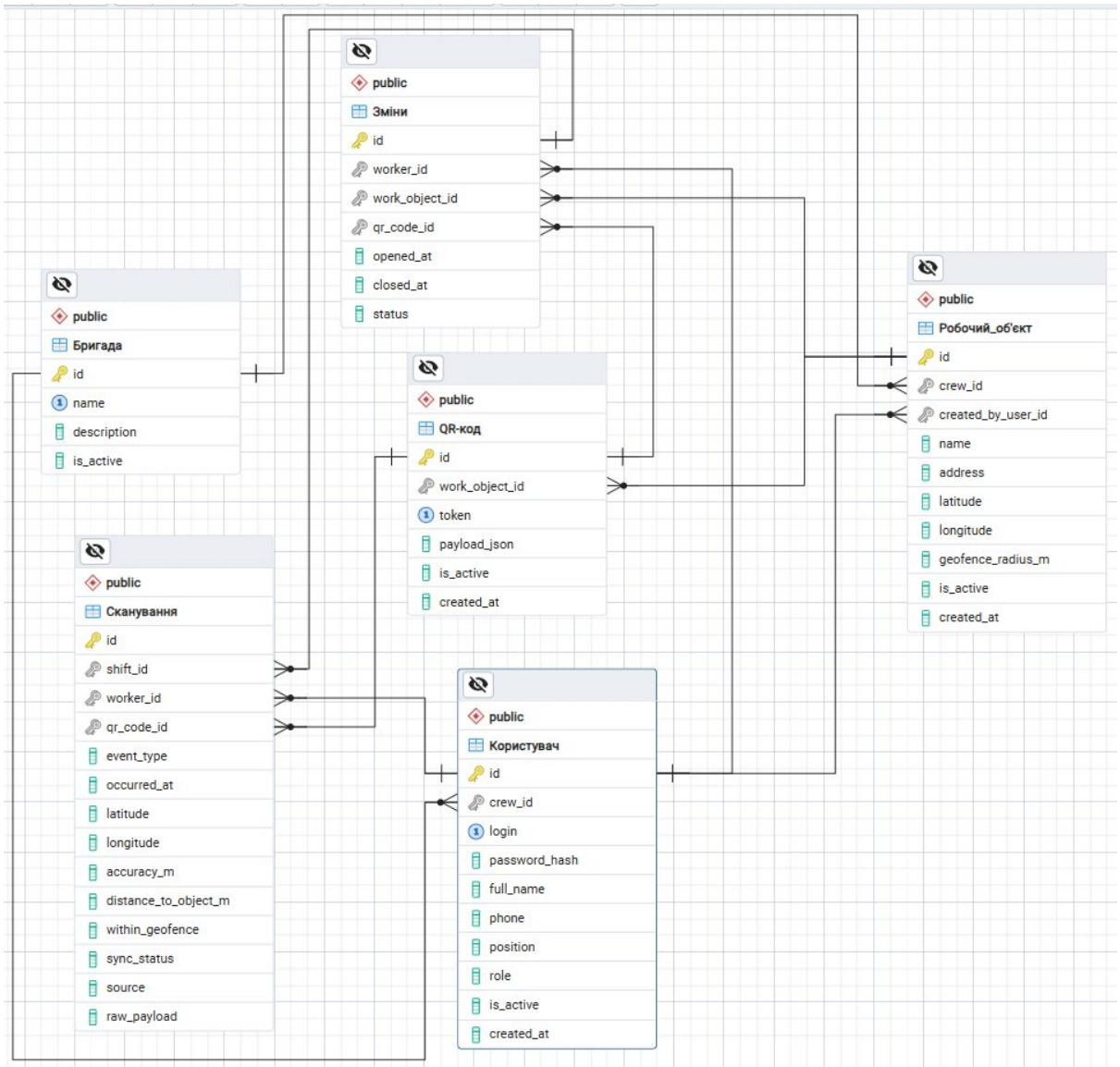


Рис 1.7 ER-діаграма

Розроблена логічна модель даних відповідає вимогам реляційної структури, оскільки основні об’єкти предметної області подані у вигляді окремих сутностей, між якими встановлено чіткі зв’язки. Кожна сутність у фізичній структурі бази даних відображається окремою таблицею, а зв’язки між ними реалізуються за допомогою первинних і зовнішніх ключів.

Модель відповідає першій нормальній формі, адже атрибути таблиць є атомарними[9]. Повторювані групи або списки значень у межах одного поля для основних сутностей не використовуються.

Друга нормальна форма також дотримана, бо в таблицях використовуються прості первинні ключі, а всі неключові атрибути повністю залежать від первинного ключа відповідної таблиці. У таблиці shifts час відкриття, час закриття та статус характеризують конкретну зміну, а не окремо працівника чи робочий об’єкт.

Третя нормальна форма забезпечується відсутністю транзитивних залежностей між неключовими атрибутами[9]. Дані про бригаду не дублюються в таблиці користувачів, а зберігаються окремо в таблиці crews. Дані про робочий об’єкт не дублюються в таблиці змін, а підключаються через зовнішній ключ work_object_id. Дані про QR-код також винесено в окрему таблицю qr_codes, що дозволяє не повторювати їх у кожній зміні або події сканування.

Отже, основна модель бази даних є реляційною та відповідає третій нормальній формі.

Також у системі присутні допоміжні напівструктуровані елементи, JSON-вміст QR-коду може передаватися між мобільним клієнтом і сервером, а локальна база мобільного застосунку тимчасово зберігає відкладені події сканування для подальшої синхронізації.

2.2 Вибір системи управління базами даних

Вибір системи бази даних для програмного застосунку керувався не тільки загальними характеристиками продуктивності, а і особливостями архітектури системи та сценаріями її практичного використання. Оскільки застосунок реалізовано як мобільну клієнт-серверну систему, у ньому використовуються два рівні збереження даних: централізоване серверне сховище та локальне сховище на мобільному пристрої.

На серверному рівні до сховища даних основні вимоги полягають в надійності, цілісності, підтримки зовнішніх ключів, транзакційності, масштабованості та гнучкості для подальшого супроводу. Тому обрано PostgreSQL, офіційна документація PostgreSQL визначає її як потужну об’єктно-реляційну систему керування базами даних, що поєднує підтримку стандартного SQL із засобами безпечного збереження і масштабування для великих навантажень. PostgreSQL також відома своєю надійністю, цілісністю даних, розвиненим набором типів даних, підтримкою обмежень, наявністю індексів і транзакцій та механізмами керування доступом.

Вибір PostgreSQL для серверної частини є доцільним з тьрох основних причин. По-перше, логічна модель розроблюваного застосунка є реляційною: у ній присутні чітко визначені сутності, зовнішні ключі та зв’язки типу «один-до-багатьох». По-друге, для сервера важливе забезпечення цілісності даних на рівні схеми, обмеження унікальності, контроль допустимих значень, зовнішні ключі та коректна робота транзакцій. По-третє, система повинна підтримувати не лише зберігання записів користувача, об’єкти та зміни, а й історію подій сканування, тобто передбачає поступове зростання обсягів даних. У таких умовах використання повноцінної серверної реляційної СУБД є більш

виправданим, ніж використання файлового сховища або спрощеного локального механізму.

Також важливо, що в межах системи частина інформації передається між клієнтом і сервером у форматі JSON, а окремі поля містять напівструктуроване навантаження, він використовується для тимчасового збереження окремих повідомлень. PostgreSQL підтримує як класичні реляційні механізми, так і розширені типи даних, включно з документними структурами JSON/JSONB.

Для локального збереження даних на мобільному пристрої використовується Room[17]. Room – це не окрема СУБД у класичному розумінні, це бібліотека Android Jetpack, яка надає рівень абстракції над SQLite і дозволяє зручно працювати з локальною базою даних. Room/SQLite використовується як локальне сховище для тимчасового накопичення подій, що виникають на мобільному пристрої за відсутності зв’язку. Обмін інформацією між цими рівнями відбувається через серверний API у форматі JSON, цей канал виконує лише транспорт даних.

Отже, обраний підхід є обґрунтованим як з точки зору логіки предметної області, так і з точки зору практичної реалізації MVP-версії системи. Він дозволяє поєднати переваги надійної серверної реляційної СУБД із можливістю локальної автономної роботи мобільного клієнта, що є принципово важливим для сервісу обліку робочого часу бригад.

2.3 Розробка структури інформаційної бази

Після побудови логічної моделі йде етап розробки структури інформаційної бази. На цьому етапі сутності, визначені в ER-моделі,

переходять у фізичні об’єкти бази даних: таблиці, поля, первинні ключі, зовнішні ключі, обмеження цілісності та індекси.

Так як розроблюваний сервіс обліку робочого часу бригад має клієнт-серверну архітектуру, інформаційна база складається з двох рівнів. Перший рівень - серверна база даних PostgreSQL, у якій зберігаються основні дані системи. До них належать відомості про бригади, користувачів, робочі об’єкти, QR-коди, зміни та події сканування. Другий рівень - локальна база даних Room/SQLite на мобільному пристрої. Вона використовується для тимчасового збереження подій сканування, які не вдалося одразу передати на сервер через відсутність або нестабільність інтернет-з’єднання.

2.3.1 Створення серверної бази даних

Для розгортання серверної бази даних використовується PostgreSQL. У проєкті база створюється під час запуску контейнера Docker. У файлі docker-compose.yml задаються назва бази даних, користувач, пароль, порт підключення та шлях до SQL-сценарію ініціалізації структури бази. Лістинг А.1 Налаштування контейнера PostgreSQL для створення бази даних.

У наведеному фрагменті база даних отримує назву crewtimekeeping. Користувач crewuser використовується серверною частиною для підключення до СУБД. Файл schema_postgres.sql автоматично виконується під час першої ініціалізації контейнера, тому таблиці та індекси створюються без ручного введення SQL-команд у середовищі адміністрування бази даних.

На рівні серверного застосунку підключення до бази даних реалізовано через SQLAlchemy. Для цього створюється об’єкт engine, фабрика сесій SessionLocal та базовий клас моделей Base. Лістинг А.2 Налаштування підключення серверної частини до бази даних.

Під час запуску FastAPI-застосунку виконується створення таблиць за ORM-моделями, якщо відповідні таблиці ще не існують. Це дозволяє

підтримувати однакову структуру між програмним кодом і базою даних. Лістинг А.3 Автоматичне створення таблиць під час запуску серверного застосунку.

Такий підхід є зручним для MVP-версії системи, оскільки структура бази даних описана одночасно у SQL-сценарії та ORM-моделях. SQL-сценарій використовується для ініціалізації PostgreSQL, а ORM-моделі застосовуються серверною частиною для роботи з цими таблицями через об’єктно-реляційне відображення.

2.3.2 Створення основних таблиць серверної бази даних

Першою створюється таблиця crews, яка відповідає сутності «бригада». Вона використовується для логічного групування працівників і робочих об’єктів. Первинним ключем є поле id. Поле name є обов’язковим і унікальним, що не дозволяє створювати кілька бригад з однаковою назвою. Лістинг А.4 Створення таблиці бригад.

Далі створюється таблиця users, у якій зберігаються всі користувачі системи. У межах однієї таблиці зберігаються як бригадири, так і працівники. Розмежування виконується через поле role, для якого встановлено обмеження CHECK. Це дозволяє зберігати лише два допустимі значення ролі: FOREMAN або WORKER. Лістинг А.5 Створення таблиці користувачів.

Поле crew_id є зовнішнім ключем до таблиці crews. Для нього використано дію ON DELETE SET NULL, тобто при видаленні бригади користувач не видаляється з бази, а лише втрачає прив’язку до неї. Це рішення зменшує ризик втрати історичних даних про користувача.

Таблиця work_objects зберігає інформацію про робочі об’єкти. Для кожного об’єкта фіксується назва, адреса, географічні координати та допустимий радіус перевірки присутності працівника. Також зберігається

користувач, який створив об’єкт. Лістинг А.6 Створення таблиці робочих об’єктів.

Для поля geofence_radius_m встановлено обмеження CHECK (geofence_radius_m > 0). Це потрібно для захисту від некоректних значень радіуса, оскільки нульовий або від’ємний радіус не має сенсу для перевірки геозони.

Таблиця qr_codes використовується для збереження QR-кодів, прив’язаних до конкретних робочих об’єктів. Кожен QR-код має унікальний токен і JSON-вміст, який передається мобільному застосунку під час сканування. Лістинг А.7 Створення таблиці QR-кодів.

Для зв’язку з робочим об’єктом використано ON DELETE CASCADE. Це означає, що при видаленні робочого об’єкта пов’язані з ним QR-коди також видаляються. Таке рішення є логічним, оскільки QR-код без робочого об’єкта втрачає практичний зміст.

Таблиця shifts зберігає інформацію про робочі зміни. Вона пов’язує працівника, робочий об’єкт і QR-код, за допомогою якого було відкрито або закрито зміну. Лістинг А.8 Створення таблиці змін.

Поле status обмежене двома значеннями: OPEN і CLOSED. Це дозволяє чітко визначати, чи зміна ще триває, чи вже завершена. Для зовнішніх ключів використано ON DELETE RESTRICT, тому система не дозволяє видалити користувача, об’єкт або QR-код, якщо з ними вже пов’язана історія змін.

Таблиця scan_events є операційною таблицею, у якій зберігаються фактичні події сканування QR-коду. На відміну від таблиці shifts, вона фіксує не лише результат зміни, а кожну окрему дію працівника: відкриття зміни, закриття зміни або відхилення події через перебування поза допустимою геозоною. Лістинг А.6 Створення таблиці подій сканування.

У цій таблиці зберігаються координати працівника, точність геолокації, відстань до об’єкта, результат перевірки геозони та початковий

вміст QR-коду. Поле event_type показує тип події, а поле sync_status відображає стан її обробки сервером. Поле source дає змогу відрізняти події, що були надіслані одразу з мобільного застосунку, від подій, які спочатку були накопичені локально і синхронізовані пізніше.

2.3.3 Створення індексів

Для підвищення швидкодії запитів у базі даних створено індекси. Вони застосовуються до тих полів, які часто використовуються під час пошуку, авторизації та отримання історії змін. Лістинг А.10 Створення індексів у серверній базі даних.

Індекс idx_users_login прискорює пошук користувача під час входу в систему. Індекс idx_shifts_worker_status потрібний для швидкого пошуку відкритих або закритих змін конкретного працівника. Індекс idx_scan_events_shift_id використовується для швидкого отримання подій, пов’язаних із певною зміною.

2.3.4 Відображення таблиць у серверному програмному коді

Крім SQL-сценарію, структура серверної бази даних описана в ORM-моделях SQLAlchemy. Це дозволяє працювати з таблицями як із класами Python, не формуючи всі SQL-запити вручну. Наприклад, таблиця бригад представлена класом Crew. Лістинг А.11 ORM-модель таблиці бригад.

Таблиця користувачів представлена класом User. У ньому визначено основні поля облікового запису, роль користувача, зв’язок із бригадою, створеними об’єктами, змінами та подіями сканування. Лістинг А.12 ORM-модель таблиці користувачів.

Для робочих змін використовується клас Shift, який відображає зв’язок між працівником, об’єктом і QR-кодом. Лістинг А.13 ORM-модель таблиці змін.

Така реалізація забезпечує відповідність між структурою бази даних і програмними класами серверної частини. Завдяки цьому сервер може виконувати операції створення робочих об’єктів, генерації QR-кодів, відкриття та закриття змін, а також формування звітності на основі реляційних даних.

2.3.5 Створення локальної бази даних мобільного застосунку

Окрему частину інформаційної бази становить локальна база даних мобільного застосунку. Вона створюється за допомогою бібліотеки Room, яка є надбудовою над SQLite для Android. Локальна база не дублює всю серверну структуру, а зберігає лише ті події сканування, які потрібно передати на сервер після відновлення мережевого з’єднання.

Для цього створено сутність PendingScanEntity, яка відповідає локальній таблиці pending_scans. Лістинг А.14 Опис локальної таблиці відкладених подій сканування.

У цій таблиці зберігається JSON-вміст QR-коду, час події, координати працівника, точність геолокації та джерело події. Поле source за замовчуванням має значення OFFLINE, що дозволяє серверу розуміти, що подія була створена в автономному режимі.

Для доступу до локальної таблиці створено DAO-інтерфейс PendingScanDao. Лістинг А.15 DAO-інтерфейс для роботи з локальними подіями.

Метод insert додає подію до локальної черги. Метод getAll отримує всі події у порядку їх створення. Метод deleteByIds видаляє події після успішної синхронізації із сервером. Метод count використовується для відображення кількості подій, які ще очікують на передачу. Клас локальної бази даних має такий вигляд: Лістинг А.16 Опис локальної бази даних Room.

У мобільному застосунку локальна база використовується під час спроби синхронізації події сканування. Якщо сервер недоступний або виникає помилка мережі, подія не втрачається, а записується до таблиці pending_scans.Лістинг А.17 Збереження події сканування у локальну базу при відсутності зв’язку.

Після відновлення підключення застосунок отримує всі локально збережені події, послідовно надсилає їх на сервер і видаляє лише ті записи, які були успішно синхронізовані. Лістинг А.18 Синхронізація локально збережених подій із сервером.

Така структура локальної бази є мінімальною, але достатньою для підтримки offline-first логіки. Мобільний застосунок не зберігає повну копію серверної бази, а накопичує тільки критично важливі події, які можуть бути втрачені через відсутність зв’язку.

2.3.6 Узагальнення структури інформаційної бази

У результаті розробки структури інформаційної бази було створено серверну базу PostgreSQL і локальну базу Room/SQLite. Серверна база містить шість основних таблиць: crews, users, work_objects, qr_codes, shifts і scan_events. Вони відображають основні сутності предметної області та забезпечують збереження як довідкових, так і операційних даних.

Цілісність серверної бази забезпечується первинними ключами, зовнішніми ключами, обмеженнями CHECK, унікальними обмеженнями та індексами. Завдяки цьому зменшується ризик появи некоректних даних, наприклад користувача з невідомою роллю, робочого об’єкта з недопустимим радіусом геозони або зміни з некоректним статусом.

Локальна база даних мобільного застосунку містить таблицю pending_scans, яка реалізує чергу відкладених подій. Вона дозволяє зберігати сканування QR-коду на пристрої працівника і передавати його на сервер пізніше. Отже, розроблена структура інформаційної бази відповідає

логіці предметної області, підтримує клієнт-серверну архітектуру системи та забезпечує стабільну роботу застосунку навіть за нестабільного інтернет-з’єднання.

2.4 Схема та фізична структура бази даних

Фізична структура бази даних розробленого програмного забезпечення відображена у схемі PostgreSQL, наведеній у Додатку Б. На цій схемі показано основні таблиці бази даних, їхні поля, типи даних, первинні ключі та зв’язки між таблицями. Саме за цією схемою можна простежити, як логічна модель, описана в попередніх підрозділах, була реалізована на рівні конкретної реляційної бази даних.

Як видно з Додатку Б, база даних складається з шести основних таблиць: Бригада, Користувач, Робочий_об’єкт, QR-код, Зміни та Сканування. Кожна з цих таблиць відповідає окремій сутності предметної області сервісу обліку робочого часу бригад. Для однозначної ідентифікації записів у кожній таблиці використовується поле id, яке має тип serial і виконує роль первинного ключа.

Відповідно до схеми, наведеної у Додатку Б, таблиця Бригада використовується для збереження інформації про робочі бригади. У ній зберігаються назва бригади, її опис та ознака активності. Поле name містить назву бригади, поле description використовується для короткого текстового опису, а поле is_active показує, чи є бригада активною в системі. Такий підхід дозволяє не видаляти бригаду фізично з бази даних, а лише змінювати її статус.

Таблиця Користувач, яка також показана в Додатку Б, призначена для збереження облікових записів користувачів системи. У ній містяться логін, хеш пароля, ПІБ користувача, номер телефону, посада, роль, статус

активності та дата створення запису. Поле full_name використовується для збереження ПІБ працівника або бригадира. Поле login потрібне для авторизації в системі, а password_hash зберігає не сам пароль, а його захищене хешоване представлення. Поле role визначає роль користувача, наприклад працівник або бригадир. Через поле crew_id таблиця Користувач пов’язується з таблицею Бригада, що дозволяє визначити належність користувача до конкретної бригади.

Згідно з Додатком Б, таблиця Робочий_об’єкт зберігає інформацію про місця виконання робіт. У ній фіксуються назва об’єкта, адреса, географічні координати, допустимий радіус геозони, ознака активності та дата створення запису. Поля latitude і longitude зберігають широту та довготу робочого об’єкта. Поле geofence_radius_m визначає допустимий радіус перебування працівника біля об’єкта в метрах. Саме ці дані використовуються під час перевірки, чи справді працівник перебуває біля робочого об’єкта під час відкриття або закриття зміни. Поле created_by_user_id показує, який користувач створив запис про об’єкт.

Таблиця QR-код, представлена у схемі в Додатку Б, використовується для збереження QR-кодів, прив’язаних до робочих об’єктів. Кожен QR-код має унікальний токен, службовий JSON-вміст, ознаку активності та дату створення. Поле work_object_id пов’язує QR-код із конкретним робочим об’єктом. Поле token використовується як унікальний ідентифікатор QR-коду, а payload_json містить службові дані, які зчитуються мобільним застосунком під час сканування.

Як показано в Додатку Б, таблиця Зміни є однією з центральних операційних таблиць бази даних. Вона зберігає інформацію про робочі зміни працівників. У таблиці фіксується працівник, робочий об’єкт, QR-код, дата й час відкриття зміни, дата й час її закриття, а також поточний статус. Поле worker_id посилається на користувача, який відкрив зміну. Поле work_object_id вказує, на якому об’єкті виконується робота. Поле

qr_code_id визначає QR-код, за яким була відкрита або закрита зміна. Поле status використовується для збереження стану зміни, наприклад відкрита або закрита.

Таблиця Сканування, наведена у Додатку Б, призначена для збереження окремих подій сканування QR-коду. Вона фіксує кожну дію працівника, пов’язану зі скануванням: відкриття зміни, закриття зміни або відхилену спробу. У таблиці зберігаються тип події, час сканування, координати працівника, точність геолокації, відстань до робочого об’єкта, результат перевірки геозони, статус синхронізації, джерело події та початкові службові дані. Поля latitude і longitude зберігають фактичні координати працівника під час сканування. Поле distance_to_object_m показує відстань від працівника до робочого об’єкта, а within_geofence визначає, чи перебував працівник у межах допустимої геозони.

Зв’язки між таблицями у фізичній структурі бази даних також відображені в Додатку Б. Таблиця Користувач пов’язана з таблицею Бригада через поле crew_id. Таблиця Робочий_об’єкт пов’язана з бригадою через crew_id і з користувачем через created_by_user_id. Таблиця QR-код пов’язана з робочим об’єктом через work_object_id. Таблиця Зміни об’єднує користувача, робочий об’єкт і QR-код, тому вона забезпечує основну логіку обліку робочого часу. Таблиця Сканування пов’язана зі зміною, користувачем і QR-кодом, що дозволяє зберігати детальну історію дій працівника.

У фізичній структурі бази даних, показаній у Додатку Б, використано кілька типів даних PostgreSQL. Тип serial застосовується для полів id у всіх основних таблицях. Він використовується для автоматичного створення унікального числового ідентифікатора запису.

Тип integer використовується для зовнішніх ключів, зокрема crew_id, worker_id, work_object_id, qr_code_id і shift_id. Через ці поля реалізуються зв’язки між таблицями. Наприклад, crew_id у таблиці Користувач вказує на

відповідну бригаду, а worker_id у таблицях Зміни та Сканування вказує на конкретного працівника.

Тип character varying застосовується для текстових даних обмеженої довжини. У таблиці Користувач він використовується для логіна, хешу пароля, ПІБ, номера телефону, посади та ролі. У таблиці Робочий_об’єкт цей тип використовується для назви та адреси об’єкта. У таблиці Зміни він застосовується для статусу зміни, а в таблиці Сканування - для типу події, статусу синхронізації та джерела події.

Тип text використовується для довших текстових значень. У таблиці Бригада він застосовується для опису бригади. У таблиці QR-код тип text використовується для поля payload_json, де зберігається службовий JSON-вміст QR-коду. У таблиці Сканування поле raw_payload також має тип text і призначене для збереження початкових службових даних події.

Тип boolean використовується для логічних значень. Поля is_active у таблицях Бригада, Користувач, Робочий_об’єкт і QR-код показують, чи активний відповідний запис. Поле within_geofence у таблиці Сканування показує результат перевірки геозони: працівник перебував у дозволеній зоні або поза нею.

Тип timestamp without time zone використовується для збереження дати та часу. У таблицях Користувач, Робочий_об’єкт і QR-код поле created_at фіксує момент створення запису. У таблиці Зміни поля opened_at і closed_at зберігають час відкриття та закриття зміни. У таблиці Сканування поле occurred_at визначає час фактичного сканування QR-коду.

Тип double precision використовується для числових значень із дробовою частиною. У таблиці Робочий_об’єкт він застосовується для координат latitude, longitude і радіуса геозони geofence_radius_m. У таблиці Сканування цей тип використовується для координат працівника, точності геолокації accuracy_m і відстані до робочого об’єкта distance_to_object_m.

Отже, згідно зі схемою, наведеною у Додатку Б, фізична структура бази даних відповідає логіці роботи сервісу обліку робочого часу бригад. Довідкові таблиці зберігають інформацію про бригади, користувачів, робочі об’єкти та QR-коди, а операційні таблиці Зміни і Сканування забезпечують фіксацію фактичного робочого часу. Використання первинних і зовнішніх ключів підтримує цілісність даних, а обрані типи даних відповідають характеру інформації, яка зберігається в системі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

Абстракція являється узагальненим образом реального об’єкта, процесу або дії, з якими працює система. Для сервісу обліку робочого часу бригад є наступні абстракції: бригадир, бригада, працівник, обліковий запис користувача, робочий об’єкт, QR-код, зміна, геолокація та звіт.

Абстракції предметної області потрібні для того, щоб описати логіку роботи системи незалежно від конкретної реалізації інтерфейсу, бази даних або програмного коду. Вони показують, які об’єкти беруть участь у роботі застосунку, які дані вони містять і які дії можуть виконувати. На рис. 3.1 наведено основні абстракції предметної області сервісу обліку робочого часу бригад.

На рисунку 3.1 представлено абстракції предметної області

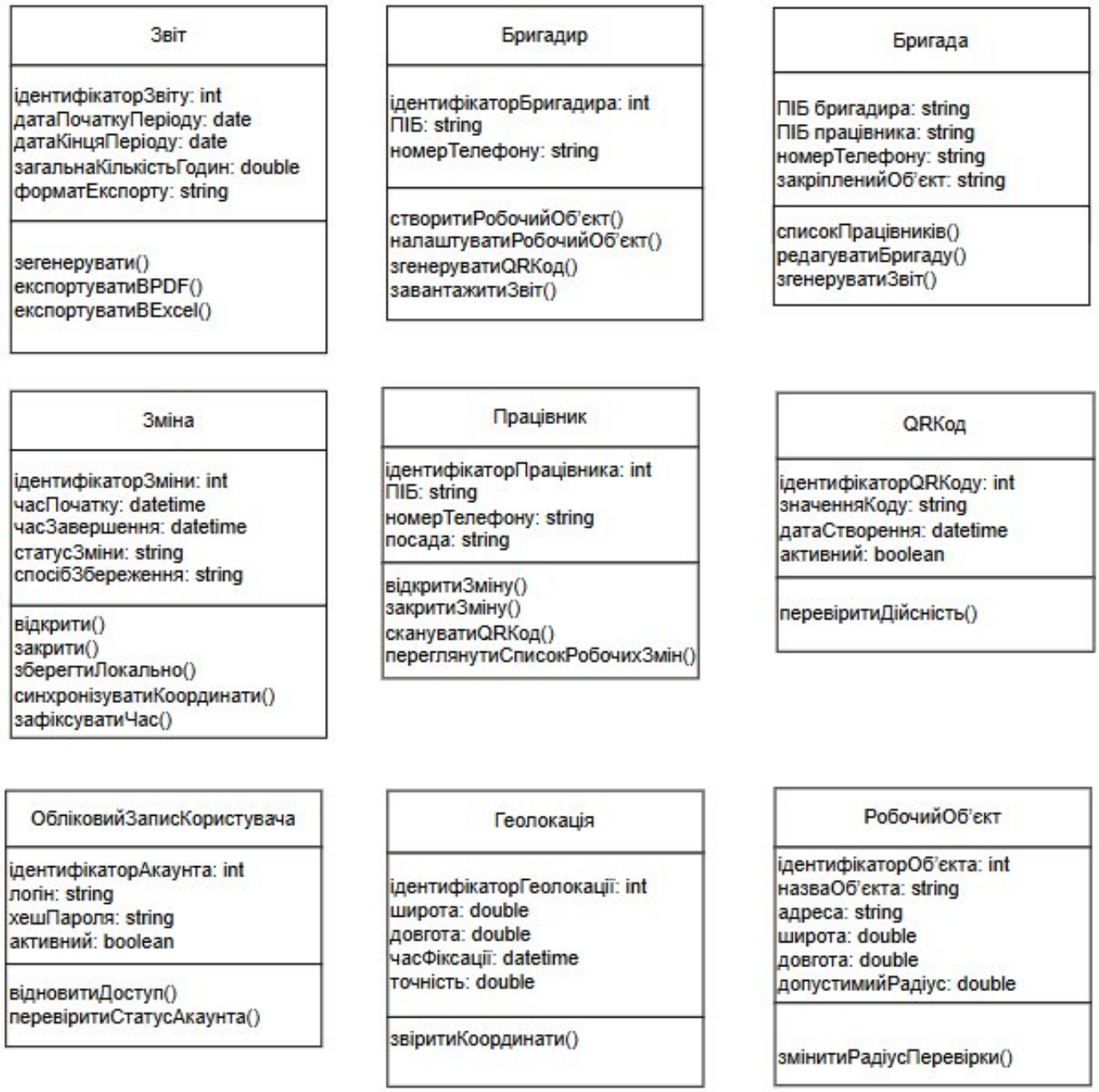


Рис 3.1 Абстракції предметної області

Таблиця 3.1

Абстракції предметної області

Абстракція	**Призначення**	**Основні дані**
Бригадир	Користувач, який керує робочими об’єктами, QR-кодами та звітами.	Ідентифікатор бригадира, ПІБ, номер телефону.
Бригада	Група працівників, організована для виконання робіт на закріплених об’єктах.	Дані бригадира, дані працівників, номер телефону, закріплений об’єкт.
Працівник	Користувач, який відкриває та закриває робочі зміни через сканування QR-коду.	Ідентифікатор працівника, ПІБ, номер телефону, посада.
Обліковий запис користувача	Дані для авторизації та перевірки доступу користувача до системи.	Ідентифікатор акаунта, логін, хеш пароля, ознака активності.
Робочий об’єкт	Місце виконання робіт, для якого задаються координати та допустимий радіус перевірки.	Ідентифікатор об’єкта, назва, адреса, широта, довгота, допустимий радіус.
QR-код	Засіб ідентифікації робочого об’єкта під час відкриття або закриття зміни.	Ідентифікатор QR-коду, значення коду, дата створення, ознака активності.
Зміна	Період роботи працівника на конкретному робочому об’єкті.	Ідентифікатор зміни, час початку, час завершення, статус зміни, спосіб збереження.
Геолокація	Дані про фактичне місцезнаходження працівника під час сканування QR-коду.	Ідентифікатор геолокації, широта, довгота, час фіксації, точність.
Звіт	Узагальнені дані про робочий час працівників за вибраний період.	Ідентифікатор звіту, дата початку періоду, дата кінця періоду, загальна кількість годин, формат експорту.

Абстракція Бригадир описує користувача, який має розширені можливості в системі. Бригадир створює робочі об’єкти, налаштовує їхні параметри, генерує QR-коди та завантажує звіти. Для цієї абстракції важливими є ідентифікатор бригадира, ПІБ і номер телефону, оскільки ці дані дозволяють визначити відповідальну особу, яка організовує роботу бригади.

Абстракція Бригада описує групу працівників, які виконують роботи на певному робочому об’єкті або групі об’єктів. Вона використовується для логічного об’єднання працівників і спрощення контролю робочого часу. У межах цієї абстракції враховуються дані бригадира, працівників, контактний номер телефону та закріплений робочий об’єкт. Основними діями є перегляд списку працівників, редагування даних бригади та формування звіту.

Абстракція Працівник описує користувача, який безпосередньо виконує роботу на об’єкті. Працівник використовує мобільний застосунок для відкриття та закриття зміни. Для цього він сканує QR-код, після чого система перевіряє його геолокацію. Основними діями працівника є відкриття зміни, закриття зміни, сканування QR-коду та перегляд списку власних робочих змін.

Абстракція Обліковий запис користувача використовується для організації доступу до системи. Вона містить логін, хеш пароля та ознаку активності. Логін потрібний для входу в застосунок, а хеш пароля використовується для безпечного збереження облікових даних. Ознака активності дозволяє визначити, чи може користувач працювати із системою. Для цієї абстракції передбачено відновлення доступу та перевірку статусу акаунта.

Абстракція Робочий об’єкт описує місце виконання робіт. Для кожного об’єкта зберігаються назва, адреса, географічні координати та допустимий радіус перевірки. Поля широти й довготи потрібні для визначення місцезнаходження об’єкта, а допустимий радіус використовується для перевірки перебування працівника в межах геозони. Основною дією цієї абстракції є зміна радіуса перевірки.

Абстракція QR-код використовується для ідентифікації робочого об’єкта. Кожен QR-код має ідентифікатор, значення коду, дату створення та ознаку активності. Під час сканування мобільний застосунок отримує

значення QR-коду й передає його на сервер. Після цього система перевіряє дійсність QR-коду, його активність і зв’язок із робочим об’єктом.

Абстракція Зміна є центральною для обліку робочого часу. Вона описує період роботи працівника на конкретному об’єкті. Зміна має час початку, час завершення, статус і спосіб збереження. Якщо працівник сканує QR-код і проходить перевірку геолокації, система відкриває зміну. Якщо зміна вже відкрита, повторне коректне сканування використовується для її закриття. Спосіб збереження показує, чи дані були одразу передані на сервер, чи тимчасово збережені локально на мобільному пристрої.

Абстракція Геолокація використовується для фіксації фактичного місцезнаходження працівника під час сканування QR-коду. Вона містить широту, довготу, час фіксації координат і точність. Ці дані потрібні для перевірки того, чи перебуває працівник у межах допустимого радіуса робочого об’єкта. Метод звірки координат порівнює координати працівника з координатами об’єкта та визначає можливість відкриття або закриття зміни.

Абстракція Звіт описує результат узагальнення даних про робочі зміни. Звіт формується за певний період і містить дату початку, дату кінця, загальну кількість годин та формат експорту. Бригадир може згенерувати звіт, експортувати його у PDF або Excel і використати для контролю робочого часу працівників.

Отже, сформовані абстракції відображають основні об’єкти й процеси сервісу обліку робочого часу бригад. Вони створюють основу для подальшого моделювання структури та взаємодії об’єктів, розробки програмних модулів і реалізації бізнес-логіки застосунку.

3.2 Моделювання структури та взаємодії об’єктів

Після визначення абстракцій предметної області було виконано моделювання структури та взаємодії об’єктів програмного забезпечення. Для цього використано загальну діаграму класів і дві діаграми кооперацій, які деталізують окремі функціональні сценарії роботи системи.

Діаграма класів відображає статичну структуру системи: основні класи, їхні атрибути, методи та зв’язки між ними.[31] Діаграми кооперацій показують, як ці об’єкти взаємодіють між собою під час виконання конкретних дій користувача. У межах розроблюваного сервісу основними сценаріями є керування бригадою, створення робочого об’єкта, генерація QR-коду, а також відкриття і закриття робочої зміни. [31–33]

3.2.1 Загальна структура об’єктів системи

Загальна структура програмного забезпечення подана у вигляді діаграми класів на рис. 3.2.

На рисунку 3.2 Діаграма класів сервісу обліку робочого часу бригад

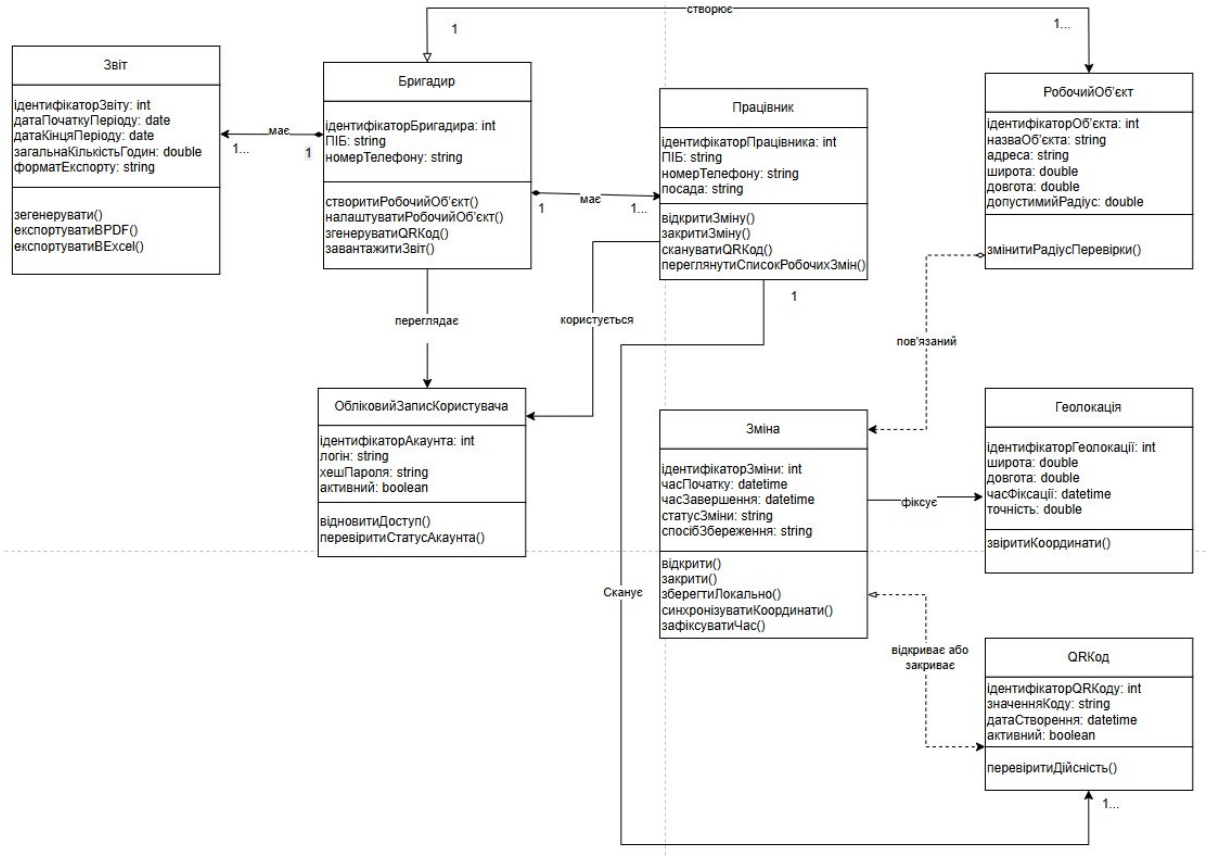


Рис 3.2 Діаграма класів

На діаграмі класів відображено такі основні класи: Бригадир, Бригада, Працівник, ОбліковийЗаписКористувача, РобочийОб’єкт, QRКод, Зміна, Геолокація та Звіт. Ці класи відповідають основним об’єктам і процесам предметної області.

Клас Бригадир описує користувача, який має права керування. Він створює робочі об’єкти, налаштовує їхні параметри, генерує QR-коди та завантажує звіти. Основними атрибутами класу є ідентифікатор бригадира, ПІБ і номер телефону.

Клас Працівник описує користувача, який фіксує власний робочий час. Працівник сканує QR-код, відкриває або закриває зміну та переглядає список своїх робочих змін. До основних атрибутів належать ідентифікатор працівника, ПІБ, номер телефону та посада.

Клас Бригада використовується для групування працівників. Він пов’язує бригадира, працівників і закріплений робочий об’єкт. Основні дії цього класу - перегляд списку працівників, редагування даних бригади та формування звіту.

Клас ОбліковийЗаписКористувача відповідає за дані авторизації. Він містить логін, хеш пароля та ознаку активності. Винесення облікового запису в окремий клас дозволяє відокремити персональні дані користувача від даних, які потрібні для входу в систему.

Клас РобочийОб’єкт описує місце виконання робіт. Він містить назву, адресу, координати та допустимий радіус перевірки. Ці дані використовуються під час перевірки геолокації працівника.

Клас QRКод є засобом ідентифікації робочого об’єкта. Він містить значення коду, дату створення та ознаку активності. Перед відкриттям або закриттям зміни система перевіряє дійсність QR-коду.

Клас Зміна описує період роботи працівника. Він містить час початку, час завершення, статус зміни та спосіб збереження. Цей клас є центральним для обліку робочого часу.

Клас Геолокація зберігає координати працівника під час сканування QR-коду. Він містить широту, довготу, час фіксації та точність. Метод звірки координат використовується для перевірки перебування працівника в межах допустимого радіуса робочого об’єкта.

Клас Звіт використовується для формування узагальнених даних про робочий час. Він містить період звіту, загальну кількість годин і формат експорту. Звіт може бути експортований у PDF або Excel.

3.2.2 Взаємозв’язки між класами

Зв’язки між класами відображають основну логіку роботи системи. Бригадир створює один або кілька робочих об’єктів і формує звіти. Також він пов’язаний із бригадою, оскільки саме бригадир відповідає за її організацію.

Бригада об’єднує працівників і може бути пов’язана із закріпленим робочим об’єктом. Один бригадир може керувати кількома працівниками, а кожен працівник належить до певної бригади.

Працівник користується обліковим записом для входу в систему. Після авторизації він може сканувати QR-код, переглядати власні зміни, відкривати або закривати зміну. Зміна створюється на основі даних працівника, QR-коду, робочого об’єкта та геолокації.

РобочийОб’єкт пов’язаний із QR-кодом, оскільки кожен QR-код використовується для ідентифікації конкретного місця виконання робіт. Також робочий об’єкт має координати та допустимий радіус, які використовуються при перевірці геолокації.

Зміна пов’язана з геолокацією, оскільки відкриття або закриття зміни залежить не лише від QR-коду, а й від фактичного місцезнаходження працівника. Якщо координати працівника не відповідають допустимій зоні, система не повинна підтверджувати зміну.

Таким чином, діаграма класів показує, що система складається з двох логічних частин. Перша частина відповідає за організацію роботи: бригадир, бригада, працівники, робочі об’єкти та QR-коди. Друга частина відповідає за облік робочого часу: зміна, геолокація та звітність.

3.2.3 Кооперація “Керування бригадою”

Для деталізації сценарію керування бригадою побудовано діаграму кооперації, наведену на рис. 3.3.

На рисунку 3.3 Діаграма кооперації “Керування бригадою

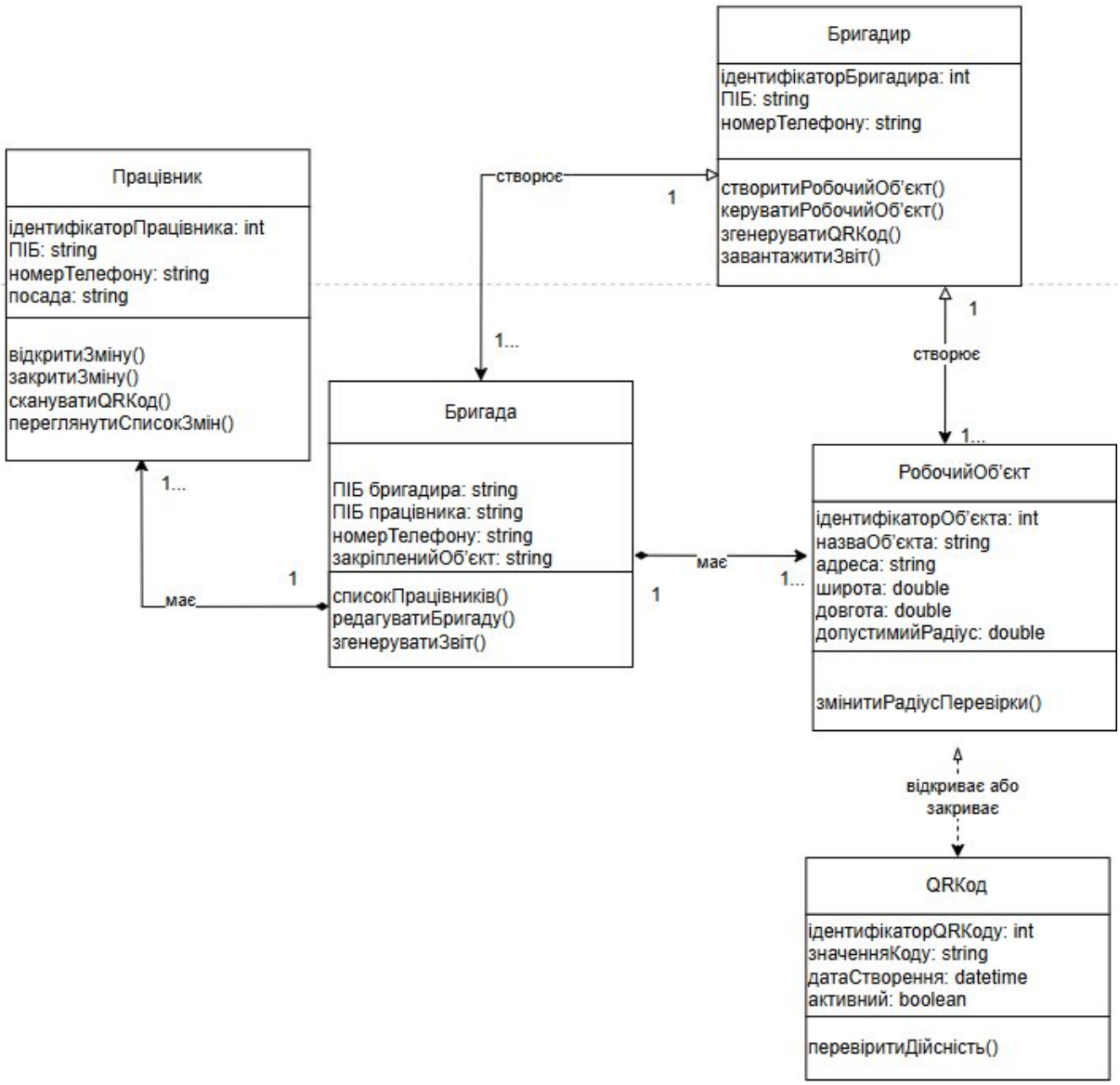


Рис 3.3 Діаграма кооперації “Керування бригадою”

У цій кооперації основним об’єктом є Бригадир. Він створює або редагує Бригаду, до якої входять працівники. Бригада пов’язується з робочим об’єктом, на якому виконуються роботи.

Після створення робочого об’єкта бригадир задає його параметри: назву, адресу, широту, довготу та допустимий радіус перевірки. Ці дані потрібні для подальшої перевірки місцезнаходження працівника під час відкриття або закриття зміни.

Далі для робочого об’єкта створюється QRКод. Він використовується працівником у мобільному застосунку. Фактично QR-код є зв’язком між фізичним місцем виконання робіт і цифровою логікою системи.

Логіка кооперації така: бригадир створює бригаду, додає або переглядає працівників, створює робочий об’єкт, налаштовує його параметри та генерує QR-код. Після цього працівники можуть використовувати QR-код для фіксації робочого часу.

3.2.4 Кооперація “Відкриття та закриття зміни”

Другий важливий сценарій роботи системи - відкриття та закриття робочої зміни. Його подано на рис. 3.4.

На рисунку 3.4 Діаграма кооперації “Відкриття та закриття зміни”

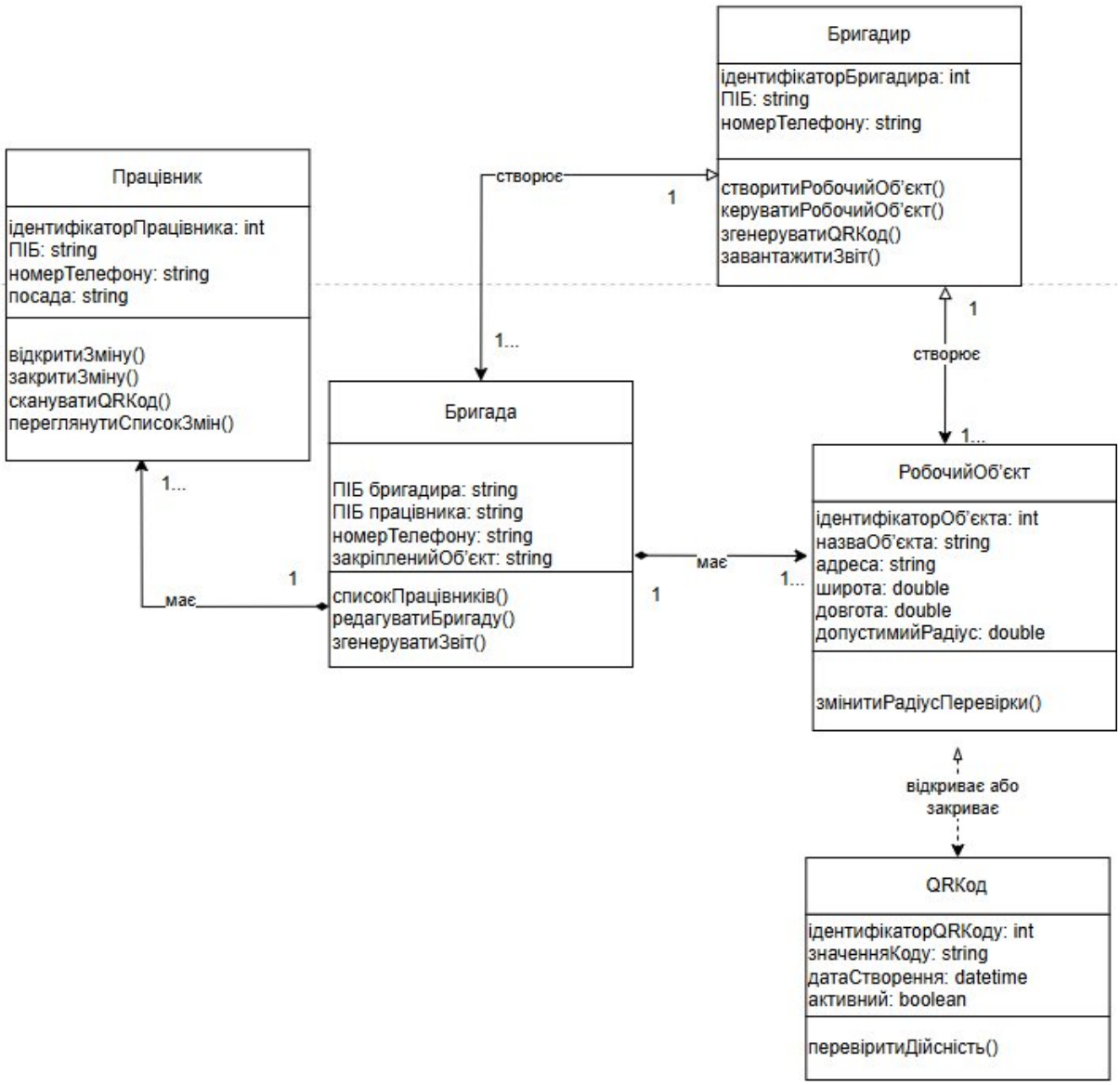


Рис 3.4 Діаграма кооперації “Відкриття та закриття зміни”

У цьому сценарії основним ініціатором є Працівник. Він сканує QRКод, після чого система перевіряє його дійсність. Якщо QR-код активний і пов’язаний із робочим об’єктом, система переходить до перевірки геолокації.

Клас Геолокація фіксує координати працівника, час фіксації та точність. Далі виконується звірка координат працівника з координатами

робочого об’єкта. Якщо працівник перебуває в межах допустимого радіуса, система дозволяє виконати операцію зі зміною.

Клас Зміна відповідає за фіксацію часу роботи. Якщо у працівника немає відкритої зміни для відповідного об’єкта, система відкриває нову зміну та записує час початку. Якщо зміна вже відкрита, повторне коректне сканування закриває зміну та записує час завершення.

У разі відсутності стабільного інтернет-з’єднання дані про зміну можуть бути тимчасово збережені локально. Після відновлення зв’язку вони синхронізуються із сервером. Це дозволяє не втрачати інформацію про фактичний робочий час.

3.2.5 Узагальнення результатів моделювання

У результаті моделювання було визначено склад системи та основні сценарії взаємодії її об’єктів. Діаграма класів показує загальну структуру програмного забезпечення, а діаграми кооперацій уточнюють логіку виконання окремих процесів.

Кооперація “Керування бригадою” описує підготовчий етап роботи системи: створення бригади, додавання працівників, створення робочого об’єкта та генерацію QR-коду. Кооперація “Відкриття та закриття зміни” описує основний робочий процес: сканування QR-коду, перевірку геолокації та фіксацію часу роботи.

Отже, побудовані моделі показують, що програмне забезпечення має логічно розділену структуру. Організаційна частина системи відповідає за бригади, працівників, робочі об’єкти та QR-коди, а операційна частина - за відкриття й закриття змін, перевірку координат і формування звітності.

3.4 Компонентна структура системи

Компонентна структура програмного забезпечення визначає склад основних програмних частин системи та залежності між ними.[32] Для сервісу обліку робочого часу бригад така структура є важливою, оскільки система складається з мобільного Android-застосунку, серверного REST API, локального сховища на пристрої та серверної бази даних.

Діаграму компонентів програмного забезпечення наведено в Додатку В. На ній показано поділ системи на три основні рівні: рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Такий поділ дозволяє відокремити інтерфейс користувача від логіки обробки даних і від механізмів збереження інформації.

3.4.1 Загальна характеристика компонентної структури

Як видно з Додатку В, із системою працюють два типи користувачів: Бригадир і Працівник. Обидва користувачі взаємодіють із мобільним застосунком через компонент MainActivity.kt, який є початковою точкою входу в Android-застосунок.

Після запуску застосунку керування передається до компонента CrewTimekeepingApp.kt. Він відповідає за відображення основних екранів, форм, вкладок, профілю, списку змін, звітів та елементів входу в систему. Тобто цей компонент належить до рівня представлення і забезпечує взаємодію користувача з програмним забезпеченням.

Подальша обробка дій користувача передається до рівня бізнес-логіки. Саме на цьому рівні визначається, яку операцію потрібно виконати: авторизувати користувача, створити робочий об’єкт, згенерувати QR-код, обробити сканування, перевірити геолокацію або отримати звіт.

3.4.2 Рівень представлення

Рівень представлення містить компоненти MainActivity.kt і CrewTimekeepingApp.kt. Їхнє основне призначення — забезпечити графічний інтерфейс мобільного застосунку.

Компонент MainActivity.kt відповідає за запуск застосунку та підключення основного інтерфейсу. Через нього користувач потрапляє до мобільного застосунку після відкриття програми на Android-пристрої.

Компонент CrewTimekeepingApp.kt містить екрани та форми, з якими працюють бригадир і працівник. Для бригадира це екрани створення робочих об’єктів, перегляду QR-кодів і звітності. Для працівника це екрани сканування QR-коду, перегляду змін і роботи з профілем.

Рівень представлення не повинен напряму працювати з базою даних або серверною логікою. Його завдання — отримати дію користувача, відобразити результат і передати запит на обробку до відповідних компонентів бізнес-логіки.

3.4.3 Рівень бізнес-логіки

Рівень бізнес-логіки об’єднує компоненти, які відповідають за правила роботи системи. У мобільній частині до нього належать MainViewModel.kt, AppRepository.kt, GeoUtils.kt, LocationHelper.kt і QrBitmapEncoder.kt.

Компонент MainViewModel.kt керує станом інтерфейсу. Він отримує дії від екранів застосунку, викликає потрібні методи репозиторію та оновлює дані, які бачить користувач. Наприклад, після входу користувача він зберігає стан авторизації, роль користувача та повідомлення про результат операції.

Компонент AppRepository.kt є проміжним шаром між інтерфейсом, серверним API та локальним сховищем. Він відповідає за виконання запитів до сервера, збереження відкладених подій сканування і синхронізацію даних після відновлення з’єднання.

Компоненти GeoUtils.kt і LocationHelper.kt відповідають за роботу з геолокацією[20]. Вони використовуються для отримання координат працівника та обчислення відстані до робочого об’єкта. Ця логіка потрібна для перевірки, чи перебуває працівник у межах допустимого радіуса під час відкриття або закриття зміни.

Компонент QrBitmapEncoder.kt використовується для формування QR-коду у графічному вигляді[21]. Він потрібний у сценарії, коли бригадир створює робочий об’єкт і система генерує для нього QR-код.

У серверній частині до рівня бізнес-логіки належать main.py, файли маршрутів API та службові модулі services.py, auth.py, deps.py. Компонент main.py є точкою входу серверного REST API. Файли routes/auth.py, routes/work_objects.py, routes/shifts.py, routes/sync.py, routes/reports.py описують маршрути для авторизації, роботи з об’єктами, змінами, синхронізацією та звітами.

Файл services.py містить основну серверну логіку. У ньому реалізуються операції створення робочого об’єкта, генерації QR-коду, обробки сканування, відкриття або закриття зміни та формування звіту. Компонент auth.py відповідає за авторизацію, а deps.py — за допоміжні залежності, наприклад отримання поточного користувача.

3.4.4 Рівень доступу до даних

Рівень доступу до даних містить компоненти, які відповідають за передавання, збереження та отримання інформації. У мобільній частині до нього належать ApiService.kt, AuthInterceptor.kt, SessionManager.kt, AppDatabase.kt, PendingScanDao.kt і PendingScanEntity.kt.

Компонент ApiService.kt описує HTTP-запити до серверного API. Через нього мобільний застосунок виконує авторизацію, створює робочі об’єкти, передає події сканування, отримує список змін і звітні дані.

Компонент AuthInterceptor.kt додає токен авторизації до запитів.[25] Це потрібно для доступу до захищених маршрутів API, які не повинні бути доступні неавторизованим користувачам.

Компонент SessionManager.kt зберігає дані сесії користувача, зокрема токен, роль, ім’я та інші службові дані. Завдяки цьому користувачу не потрібно повторно вводити облікові дані після кожного запуску застосунку.

Компоненти AppDatabase.kt, PendingScanDao.kt і PendingScanEntity.kt відповідають за локальне збереження відкладених подій сканування. Вони використовуються тоді, коли працівник виконав сканування QR-коду, але мобільний застосунок не зміг одразу передати подію на сервер. Після відновлення з’єднання ці події синхронізуються.

У серверній частині рівень доступу до даних представлений компонентами database.py, models.py і schemas.py. Компонент database.py відповідає за підключення до бази даних. У models.py описуються ORM-моделі, які відповідають таблицям бази даних[26]. У schemas.py описуються структури вхідних і вихідних даних API[27].

3.4.5 Взаємодія компонентів системи

Взаємодія компонентів, показана в Додатку В, починається з дії користувача. Бригадир або працівник відкриває мобільний застосунок через MainActivity.kt. Далі інтерфейсний компонент CrewTimekeepingApp.kt передає дію користувача до MainViewModel.kt.

MainViewModel.kt визначає, яку операцію потрібно виконати, і звертається до AppRepository.kt. Якщо дія потребує роботи з сервером, репозиторій використовує ApiService.kt. Перед відправленням запиту компонент AuthInterceptor.kt додає авторизаційний токен.[23] Якщо дія потребує локального збереження, репозиторій звертається до AppDatabase.kt і пов’язаних компонентів локального сховища.

На сервері запит потрапляє до main.py, після чого передається до відповідного маршруту API. Наприклад, авторизація обробляється через routes/auth.py, створення робочого об’єкта — через routes/work_objects.py, сканування QR-коду — через routes/sync.py, а звітність — через routes/reports.py.

Після маршруту запит передається до серверної логіки. Компонент services.py виконує основні перевірки та операції: перевіряє QR-код, визначає робочий об’єкт, обчислює відстань до нього, відкриває або закриває зміну, формує звіт. Для роботи з базою даних сервер використовує database.py, models.py і schemas.py.

Завершальним компонентом є база даних crewtimekeeping, структура якої створюється за сценарієм schema_postgres.sql. У ній зберігаються користувачі, бригади, робочі об’єкти, QR-коди, зміни та події сканування.

3.4.6 Висновки щодо компонентної структури

Компонентна структура, наведена в Додатку В, показує логічний поділ системи на рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Така побудова спрощує підтримку програмного забезпечення, оскільки кожен компонент має чітко визначену відповідальність.

Рівень представлення відповідає за взаємодію з користувачем. Рівень бізнес-логіки реалізує правила роботи системи. Рівень доступу до даних забезпечує обмін із сервером, локальне збереження подій і роботу з базою даних. Завдяки цьому система залишається структурованою, а її окремі частини можна змінювати або розширювати без повного переписування всього застосунку.

3.5 Вибір інструментарію для створення прикладного програмного забезпечення

Для створення прикладного програмного забезпечення було обрано інструменти, які відповідають клієнт-серверній архітектурі системи та підтримують роботу мобільного застосунку, серверної частини і бази даних.

Клієнтська частина розроблена для платформи Android мовою Kotlin[14]. Ця мова обрана через її офіційну підтримку в Android-розробці, зручний синтаксис і хорошу сумісність із сучасними бібліотеками Android. Для побудови інтерфейсу використовується підхід на основі Kotlin-файлів застосунку, зокрема MainActivity.kt і CrewTimekeepingApp.kt [16].

Для роботи з локальними даними на мобільному пристрої використано Room/SQLite. Room спрощує роботу з SQLite-базою через DAO-інтерфейси та сутності.[17] У цьому проєкті локальна база потрібна для збереження відкладених подій сканування QR-коду, якщо немає стабільного інтернет-з’єднання.

Серверна частина реалізована мовою Python із використанням FastAPI. FastAPI обрано через просту організацію REST API, підтримку валідації даних і зручну роботу з маршрутами[24]. Запуск серверної частини виконується через Uvicorn, який використовується як ASGI-сервер.

Для роботи серверної частини з базою даних використано SQLAlchemy[26]. Цей інструмент дозволяє описувати таблиці бази даних у вигляді ORM-моделей і працювати з ними через об’єктно-реляційне відображення. Для опису структур вхідних і вихідних даних API використовуються схеми, які відокремлюють внутрішню модель бази від даних, що передаються клієнту.

Основною серверною СУБД обрано PostgreSQL[11]. Вона використовується для централізованого збереження даних про користувачів, бригади, робочі об’єкти, QR-коди, зміни та події сканування. PostgreSQL підходить для цієї системи, оскільки підтримує реляційну модель, зовнішні ключі, обмеження цілісності, індекси та стабільну роботу з транзакційними даними[10].

Для обміну даними між мобільним застосунком і сервером використовується формат JSON через HTTP-запити. Такий підхід є зручним для REST API, оскільки дозволяє передавати структуровані дані між Android-клієнтом і серверною частиною.

Для контейнеризації серверної інфраструктури використано Docker. Він спрощує запуск PostgreSQL і серверної частини в однаковому середовищі, що зменшує ризик помилок під час розгортання.

Отже, обраний інструментарій охоплює всі частини системи: Kotlin використовується для мобільного застосунку, Room/SQLite — для локального збереження даних, Python і FastAPI — для серверної логіки, PostgreSQL — для основної бази даних, а Docker — для розгортання та ізоляції середовища. Такий набір засобів є достатнім для реалізації сервісу обліку робочого часу бригад.

3.6 Алгоритмізація та програмування програмних модулів

На етапі програмування система була розділена на окремі модулі, кожен із яких виконує конкретну функцію. Такий поділ спрощує розробку, тестування та подальше розширення програмного забезпечення. Повні фрагменти програмного коду основних модулів наведено в Додатку Е.

3.6.1 Загальний склад програмних модулі. Програмне забезпечення сервісу обліку робочого часу бригад складається з клієнтської та серверної частини. Клієнтська частина реалізована у вигляді Android-застосунку, а серверна частина — у вигляді REST API на FastAPI.

Основні програмні модулі системи наведено в табл. 3.2.

Таблиця 3.2 – Основні програмні модулі системи

Модуль	Файли реалізації	Призначення
Модуль запуску серверної частини	main.py	Створення FastAPI-застосунку, підключення маршрутів API.
Модуль авторизації	auth.py, routes/auth.py, deps.py	Реєстрація, вхід, перевірка токена та ролі користувача.
Модуль роботи з об’єктами	routes/work_objects.py, services.py	Створення робочих об’єктів і генерація QR-кодів.
Модуль обліку змін	routes/sync.py, routes/shifts.py, services.py	Обробка сканування QR-коду, відкриття та закриття змін.
Модуль звітності	routes/reports.py, services.py	Формування звітів для бригадира.
Модуль моделей даних	models.py, schemas.py, database.py	Опис ORM-моделей, схем API та підключення до бази даних.
Модуль Android-застосунку	CrewTimekeepingApplication.kt, MainActivity.kt	Ініціалізація застосунку, Retrofit, Room і репозиторію.
Модуль інтерфейсу та стану	CrewTimekeepingApp.kt, MainViewModel.kt	Відображення екранів і керування станом застосунку.
Модуль клієнтської логіки	AppRepository.kt, ApiService.kt	Обмін із сервером, робота з профілем,

		об’єктами, змінами та звітами.
Модуль локального збереження	AppDatabase.kt, PendingScanDao.kt, PendingScanEntity.kt	Збереження відкладених подій сканування.
Модуль геолокації та QR-кодів	GeoUtils.kt, LocationHelper.kt, QrBitmapEncoder.kt	Обробка координат, отримання геопозиції та робота з QR-кодами.

3.6.2 Модуль запуску серверної частини

Серверна частина запускається через файл main.py. У цьому модулі створюється екземпляр FastAPI-застосунку, підключаються маршрути API та налаштовується CORS. Також під час запуску виконується створення таблиць бази даних на основі ORM-моделей. Код модуля запуску серверної частини наведено в Додатку Е, лістинг Е.1. Основний алгоритм роботи модуля:

- створити FastAPI-застосунок;
- підключити маршрути авторизації, робочих об’єктів, синхронізації, змін і звітів;
- налаштувати доступ клієнтського застосунку через CORS;
- під час запуску створити таблиці бази даних, якщо вони ще не існують;
- надати службовий маршрут /health для перевірки стану сервера.

Цей модуль є точкою входу до серверної частини та об’єднує всі API-маршрути в один застосунок.

3.6.3 Модуль авторизації користувачів

Модуль авторизації відповідає за реєстрацію, вхід у систему, створення JWT-токена та перевірку поточного користувача. Він складається з файлів auth.py, routes/auth.py і deps.py.

У файлі auth.py реалізовано хешування пароля, перевірку пароля, створення та декодування токена. У файлі routes/auth.py описано API-маршрути для входу, реєстрації, отримання поточного профілю та редагування профілю. Файл deps.py використовується для отримання поточного користувача з токена та перевірки ролі. Код модуля авторизації наведено в Додатку Е, лістинг Е.2. Алгоритм входу користувача в систему має такий вигляд:

- користувач вводить логін і пароль;
- сервер нормалізує логін і шукає користувача в базі даних;
- якщо користувача не знайдено або пароль неправильний, повертається помилка;
- якщо дані правильні, сервер створює JWT-токен;
- токен повертається мобільному застосунку;
- надалі мобільний застосунок додає цей токен до захищених запитів.

Такий підхід дозволяє відокремити відкриті операції, наприклад реєстрацію, від захищених операцій, які доступні тільки авторизованим користувачам.

3.6.4 Модуль створення робочого об’єкта та QR-коду

Модуль роботи з об’єктами відповідає за створення робочих об’єктів бригадиром і генерацію QR-коду для кожного об’єкта. API-маршрути описані у файлі routes/work_objects.py, а основна логіка реалізована у функції create_work_object() файлу services.py. Код створення робочого об’єкта та QR-коду наведено в Додатку Е, лістинг Е.3. Алгоритм створення робочого об’єкта:

- перевірити, що поточний користувач має роль FOREMAN;
- отримати від клієнта назву, адресу, координати та радіус геозони;
- створити запис робочого об’єкта в базі даних;
- згенерувати унікальний токен QR-коду;

- сформувати JSON-вміст QR-коду з токеном, ідентифікатором об’єкта, координатами та радіусом;
- створити запис QR-коду в базі даних;
- повернути клієнту дані об’єкта та JSON-вміст QR-коду.

Цей модуль забезпечує підготовчий етап роботи системи. Після створення об’єкта бригадир може передати або роздрукувати QR-код, а працівник використовує його для відкриття чи закриття зміни.

3.6.5 Модуль обробки сканування та обліку зміни

Основна логіка обліку робочого часу реалізована у функції process_scan_event() файлу services.py. API-маршрут для передавання події сканування описано у файлі routes/sync.py. Список змін працівника повертається через маршрут routes/shifts.py. Код модуля обробки сканування наведено в Додатку Е, лістинг Е.4. Алгоритм обробки сканування QR-коду:

- перевірити, що поточний користувач має роль WORKER;
- отримати JSON-вміст QR-коду, час події, координати працівника та точність геолокації;
- перевірити коректність JSON-вмісту QR-коду;
- знайти QR-код у базі даних за токеном;
- отримати робочий об’єкт, до якого прив’язаний QR-код;
- обчислити відстань між координатами працівника та координатами об’єкта;
- перевірити, чи перебуває працівник у межах допустимого радіуса;
- якщо працівник поза геозоною, зберегти подію з типом REJECT;
- якщо працівник у межах геозони, перевірити наявність відкритої зміни;
- якщо відкритої зміни немає, створити нову зміну зі статусом OPEN;
- якщо відкрита зміна є, закрити її та встановити статус CLOSED;

- зберегти подію сканування в таблиці scan_events;
- повернути клієнту результат операції.

Для обчислення відстані використовується формула гаверсинуса. Вона дозволяє визначити відстань між двома точками за широтою та довготою. На сервері це реалізовано у функції haversine_distance_m(), а на мобільному клієнті — у GeoUtils.kt.

3.6.6 Модуль локального збереження та синхронізації

Для підтримки роботи в умовах нестабільного інтернет-з'єднання в мобільному застосунку реалізовано модуль локального збереження подій. Він складається з файлів PendingScanEntity.kt, PendingScanDao.kt, AppDatabase.kt і відповідної логіки в AppRepository.kt. Код локального збереження та синхронізації наведено в Додатку Е, лістинг Е.5. Алгоритм роботи offline-first логіки:

- працівник сканує QR-код;
- мобільний застосунок отримує координати працівника;
- застосунок намагається передати подію на сервер;
- якщо сервер доступний, подія одразу обробляється;
- якщо сервер недоступний, подія записується в локальну таблицю pending_scans;
- застосунок періодично перевіряє локальну чергу;
- після відновлення зв'язку відкладені події надсилаються на сервер;
- успішно синхронізовані записи видаляються з локальної бази.

Для автоматичної повторної синхронізації використовується WorkManager.[18] У застосунку передбачено періодичну синхронізацію, а також одноразовий запуск синхронізації після старту програми.

3.6.7 Модуль стану мобільного застосунку.

Керування станом мобільного застосунку реалізовано у файлі MainViewModel.kt. Цей модуль є посередником між інтерфейсом користувача та репозиторієм. Він зберігає поточний стан екранів, повідомлення для користувача, дані профілю, список об’єктів, список змін, звіт бригадира та кількість відкладених подій.

Код модуля стану мобільного застосунку наведено в Додатку Е, лістинг Е.6. Основні функції модуля:

- login() — виконує вхід користувача;
- register() — виконує реєстрацію;
- createWorkObject() — створює робочий об’єкт;
- processQrScan() — обробляє сканування QR-коду;
- loadShifts() — завантажує список змін працівника;
- loadForemanReport() — завантажує звіт бригадира;
- syncPending() — запускає синхронізацію відкладених подій;
- logout() — очищує сесію користувача.

У методі processQrScan() спочатку перевіряється QR-вміст і локально обчислюється відстань до об’єкта. Якщо працівник перебуває поза допустимою геозоною, користувачу одразу показується повідомлення. Якщо перевірка успішна, подія передається до репозиторію для синхронізації із сервером або локального збереження.

3.6.8 Модуль обміну даними з API

Обмін даними між Android-застосунком і сервером реалізовано через ApiService.kt та AppRepository.kt. У ApiService.kt описано HTTP-запити до серверного API, а AppRepository.kt об’єднує роботу з API, локальною базою та сесією користувача.[22] Код модуля обміну даними наведено в Додатку Е, лістинг Е.7. У застосунку використовуються такі основні API-запити:

- auth/login — вхід користувача;

- auth/register — реєстрація користувача;
- auth/me — отримання даних поточного користувача;
- work-objects — створення робочого об’єкта;
- work-objects/mine — отримання об’єктів бригадира;
- shifts/me — отримання змін працівника;
- sync/scan-event — синхронізація події сканування;
- reports/foreman — отримання звіту бригадира.

Для додавання токена авторизації до запитів використовується AuthInterceptor.kt. Дані сесії зберігаються в SessionManager.kt.

3.6.9 Модуль формування звітності

Формування звіту для бригадира реалізовано у функції build_foreman_report() файлу services.py, а API-маршрут описано у routes/reports.py. Код модуля звітності наведено в Додатку Е, лістинг Е.8. Алгоритм формування звіту:

- визначити поточного бригадира;
- отримати всі робочі об’єкти, створені цим бригадиром;
- отримати зміни, пов’язані з цими об’єктами;
- для кожної закритої зміни обчислити тривалість у хвилинах;
- підрахувати кількість відкритих і закритих змін;
- підрахувати загальну тривалість роботи;
- сформувати список записів звіту;
- повернути узагальнені дані мобільному застосунку.

Цей модуль дає бригадиру можливість контролювати фактичний робочий час працівників і бачити стан виконання робіт на створених ним об’єктах.

3.6.10 Узагальнення результатів програмування модулів

У результаті алгоритмізації та програмування було реалізовано основні модулі системи: авторизацію, роботу з об’єктами, генерацію QR-кодів, обробку сканування, перевірку геолокації, відкриття та закриття змін, локальне збереження подій, синхронізацію і формування звітності.

Серверні модулі відповідають за перевірку користувачів, виконання бізнес-логіки та збереження даних у PostgreSQL. Мобільні модулі відповідають за інтерфейс, роботу з QR-кодом, отримання геолокації, передачу даних на сервер і локальне збереження подій при відсутності зв’язку.

Отже, програмні модулі реалізують повний цикл роботи системи: від створення робочого об’єкта бригадиром до сканування QR-коду працівником, фіксації зміни, синхронізації даних і формування звіту. Повні лістинги ключових модулів наведено в Додатку Е.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування систем

Тестування системи виконувалося для перевірки коректності роботи основних функцій програмного забезпечення сервісу обліку робочого часу бригад. Оскільки система має клієнт-серверну архітектуру, перевірка проводилася на рівні мобільного застосунку, серверного API, локального збереження даних і серверної бази PostgreSQL. Під час тестування використовувалися такі методи: функціональне тестування, модульне тестування, інтеграційне тестування, системне тестування. [34]

Першим етапом було перевірено авторизацію користувача. Для цього в мобільному застосунку вводилися логін і пароль зареєстрованого користувача. Якщо дані були правильними, система відкривала головний екран відповідно до ролі користувача. Для бригадира були доступні функції створення робочого об’єкта, генерації QR-коду та перегляду звіту. Для працівника були доступні функції сканування QR-коду, відкриття або закриття зміни та перегляду власних змін.

На рисунку 4.1 Вікно авторизації

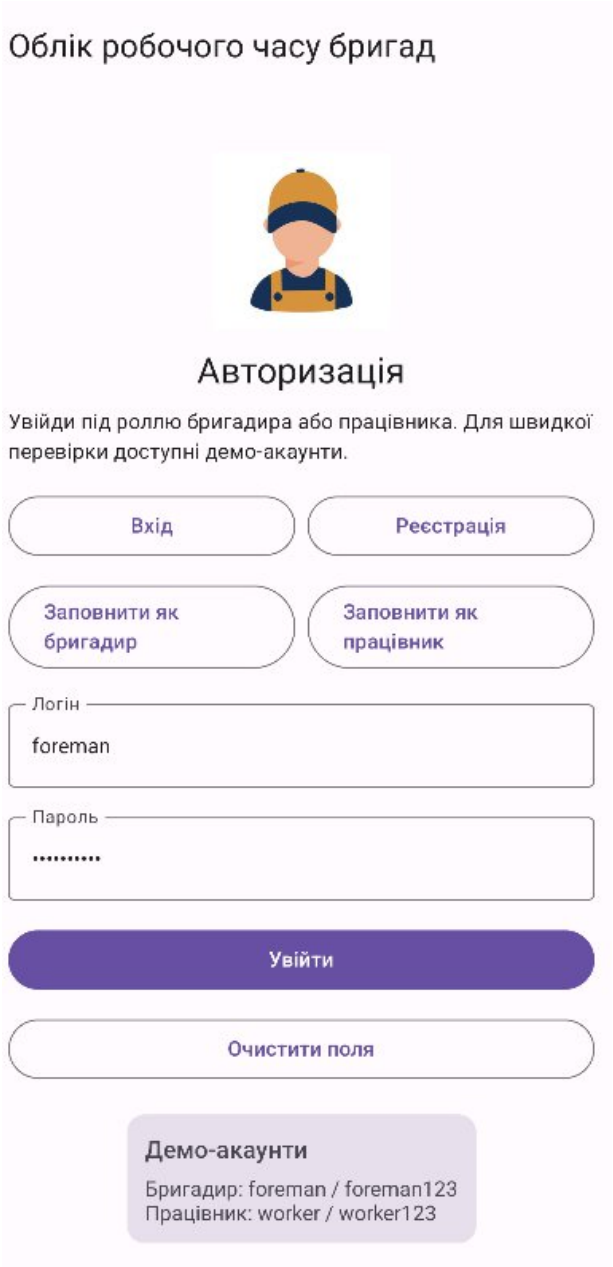


Рис 4.1 Вікно авторизації

Додатково перевірялася авторизація з неправильними даними. У разі введення неправильного пароля система не виконувала вхід і показувала повідомлення про помилку. Це підтверджує, що доступ до функцій системи можливий тільки після успішної перевірки облікових даних.

Після авторизації бригадира було перевірено створення робочого об’єкта. У застосунку вводилися назва об’єкта, адреса, географічні

координати та допустимий радіус перевірки. Після збереження об’єкта система створювала відповідний запис на сервері та генерувала QR-код, який прив’язувався до цього об’єкта.

На рисунку 4.2 Створення робочого об’єкта бригадиром

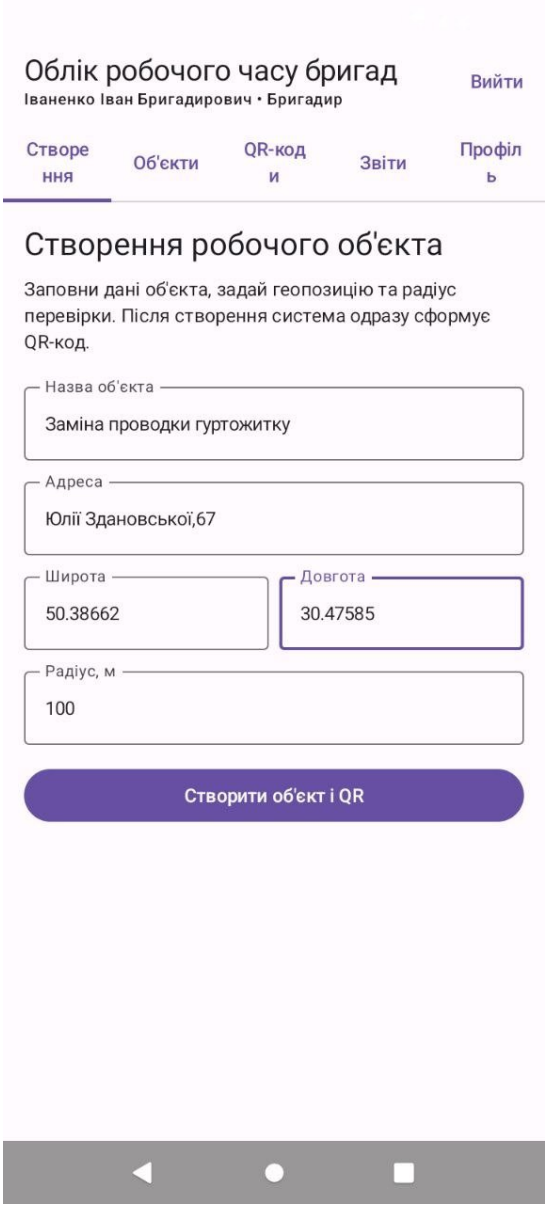


Рис 4.2 Створення робочого об’єкта бригадиром

На рисунку 4.3 Згенерований QR-код для робочого об’єкта

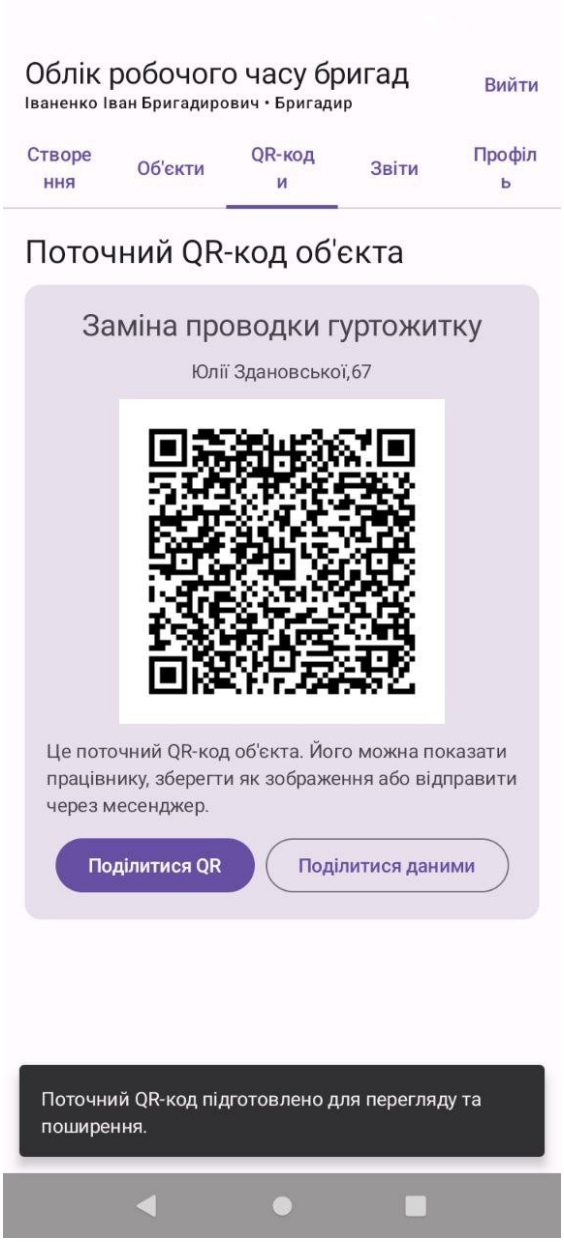


Рис 4.3 згенерований QR-код для робочого об’єкта

На цьому етапі перевірялося, чи правильно передаються дані з мобільного застосунку на сервер, чи створюється запис у базі даних і чи формується QR-код із необхідним службовим вмістом. У результаті тестування було підтверджено, що створений QR-код містить дані, потрібні для подальшого відкриття або закриття зміни працівником.

Наступним етапом було протестовано відкриття зміни працівником. Працівник авторизувався в мобільному застосунку, сканував QR-код робочого об’єкта, після чого

система отримувала його поточні координати. Якщо працівник перебував у межах допустимого радіуса, система відкривала зміну та фіксувала час початку роботи.

На рисунку 4.4 Сканування QR-коду працівником

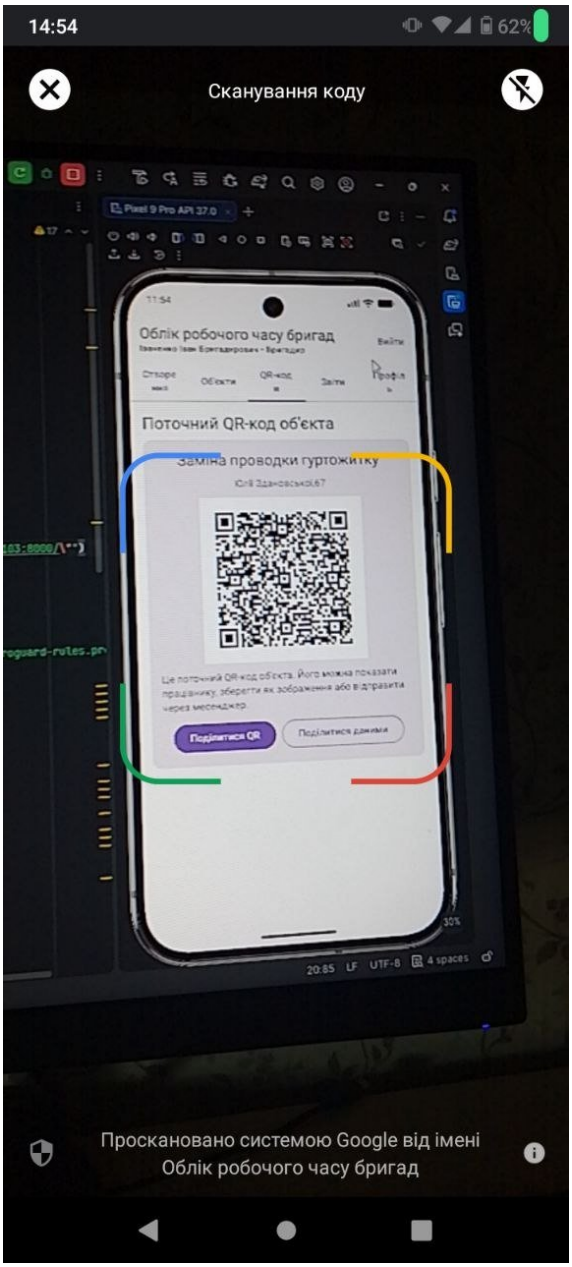


Рис 4.4 Сканування QR-коду працівником

На рисунку 4.5 Повідомлення про успішне відкриття зміни

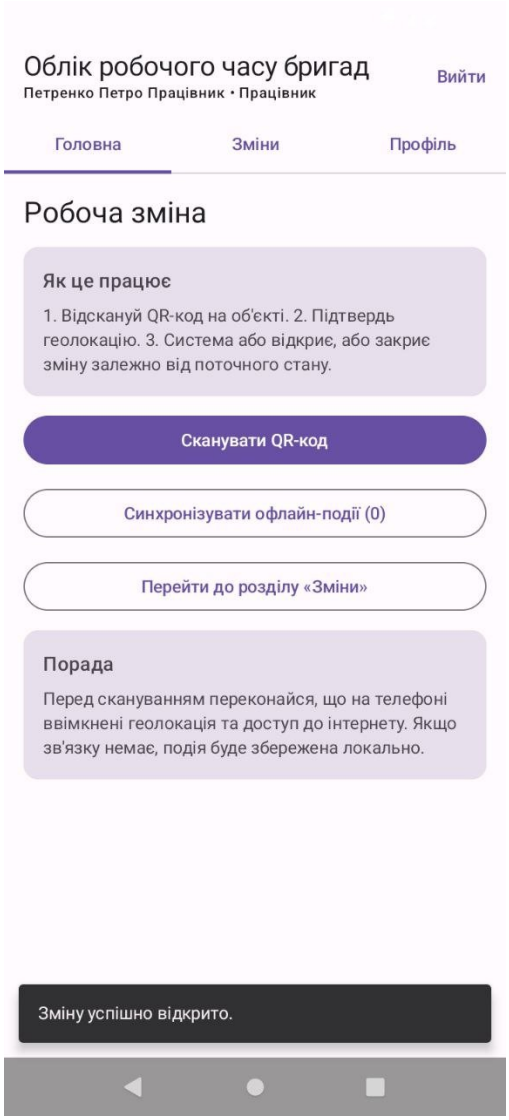


Рис 4.5 Повідомлення про успішне відкриття зміни

Під час перевірки було встановлено, що система коректно визначає робочий об’єкт за QR-кодом, отримує координати працівника та обчислює відстань до об’єкта. Якщо відстань не перевищує допустимий радіус, створюється нова зміна зі статусом OPEN.

Після цього було перевірено закриття зміни. Для цього працівник повторно сканував QR-код того самого робочого об’єкта. Якщо для цього працівника вже існувала відкрита зміна, система не створювала нову, а завершувала поточну зміну, записувала час завершення та змінювала статус на CLOSED.

На рисунку 4.6 Повідомлення про успішне закриття зміни

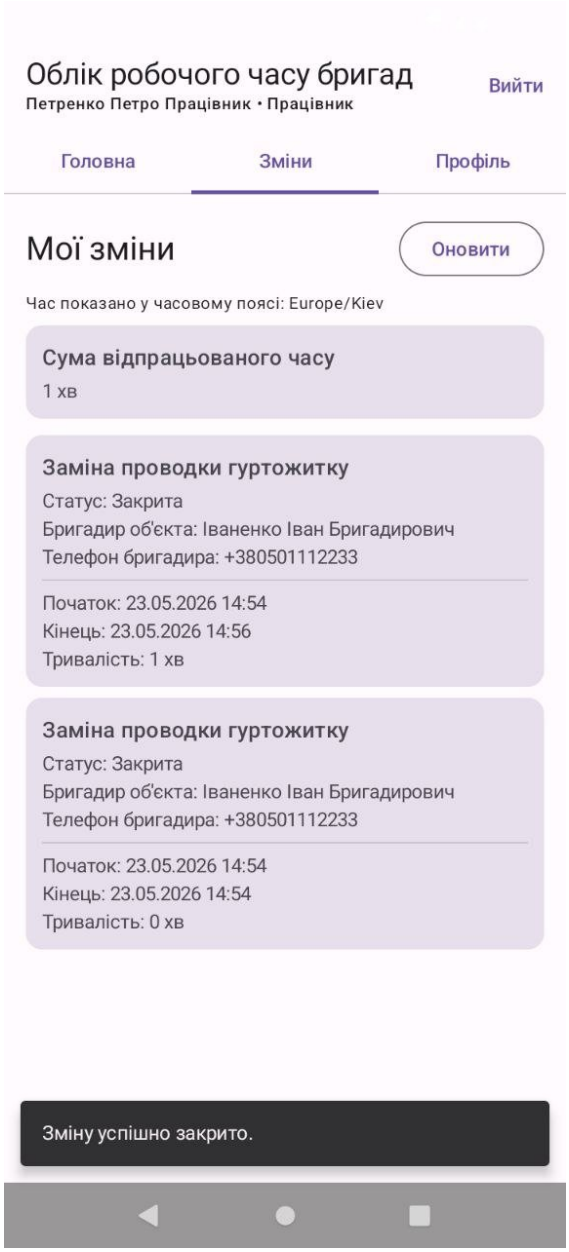


Рис 4.6 Повідомлення про успішне закриття зміни

Окремо перевірявся сценарій, коли працівник перебуває поза допустимою геозоною. У такому випадку система обчислювала відстань між координатами працівника та координатами робочого об’єкта. Якщо відстань була більшою за встановлений радіус, зміна не відкривалася і користувач отримував повідомлення про неможливість виконання дії.

На рисунку 4.7 Перебування поза допустимою геозоною

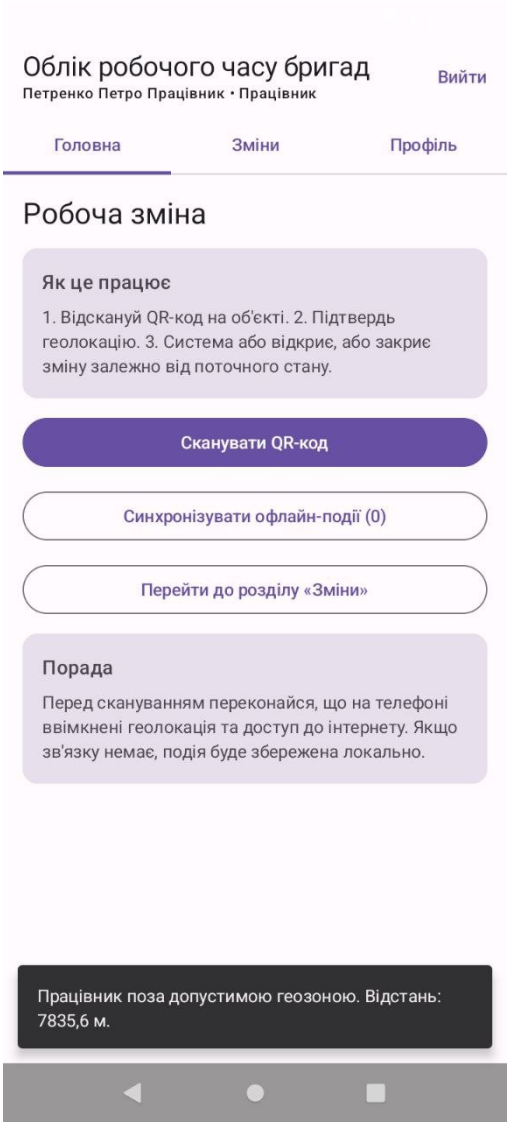


Рис 4.7 Перебування поза допустимою геозоною

Такий результат є коректним, оскільки система повинна запобігати фіксації робочого часу без фактичної присутності працівника на об’єкті. Це одна з ключових перевірок для даного програмного забезпечення.

Також було протестовано роботу системи при відсутності інтернет-з’єднання. Під час сканування QR-коду мобільний застосунок намагався передати подію на сервер. Якщо сервер був недоступний, подія не втрачалася, а зберігалася в локальній базі даних у таблиці відкладених

сканувань. Після відновлення підключення подія автоматично передавалася на сервер і видалялася з локального сховища після успішної синхронізації.

На рисунку 4.8 Збереження події сканування в локальній черзі

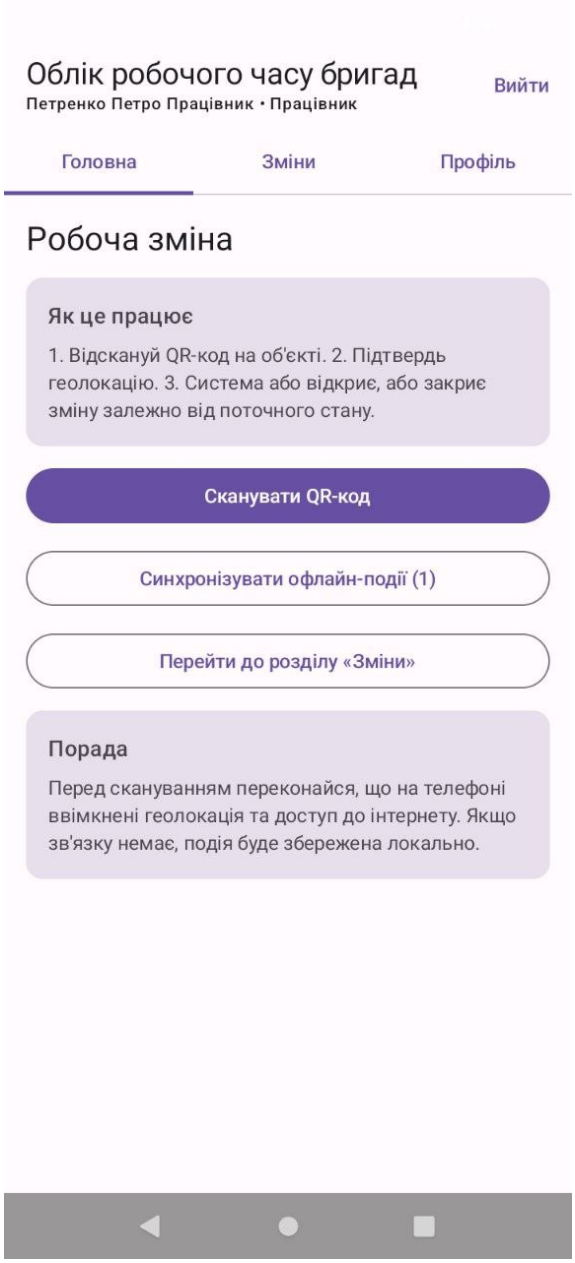


Рис 4.8 збереження події сканування в локальній черзі

Цей сценарій підтверджує працездатність offline-first логіки. Вона важлива для практичного використання системи, оскільки працівники можуть виконувати роботи на об’єктах із нестабільним мобільним інтернетом.

Після відкриття та закриття кількох змін було перевірено перегляд списку змін працівника. У застосунку відображалися зміни із зазначенням робочого об’єкта, часу відкриття, часу закриття та поточного статусу. Це дозволяє працівнику контролювати власну історію роботи.

На рисунку 4.9 Перегляд списку робочих змін працівника

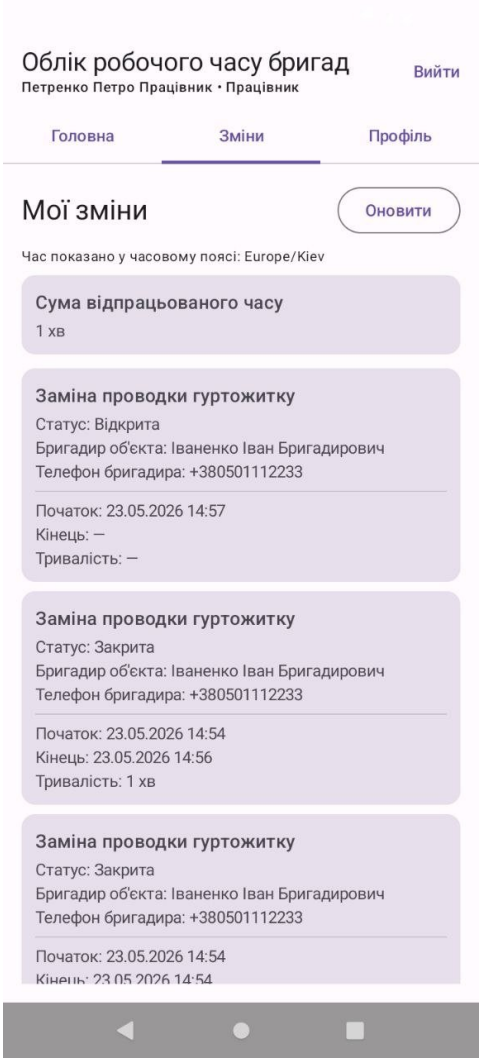


Рис 4.9 Перегляд списку робочих змін працівника

На завершальному етапі було протестовано формування звіту бригадира. Бригадир відкривав розділ звітності, після чого система отримувала з сервера узагальнену інформацію про робочі об’єкти, кількість змін, відкриті та закриті зміни, а також загальну тривалість роботи працівників.

На рисунку 4.10 Перегляд звіту бригадира

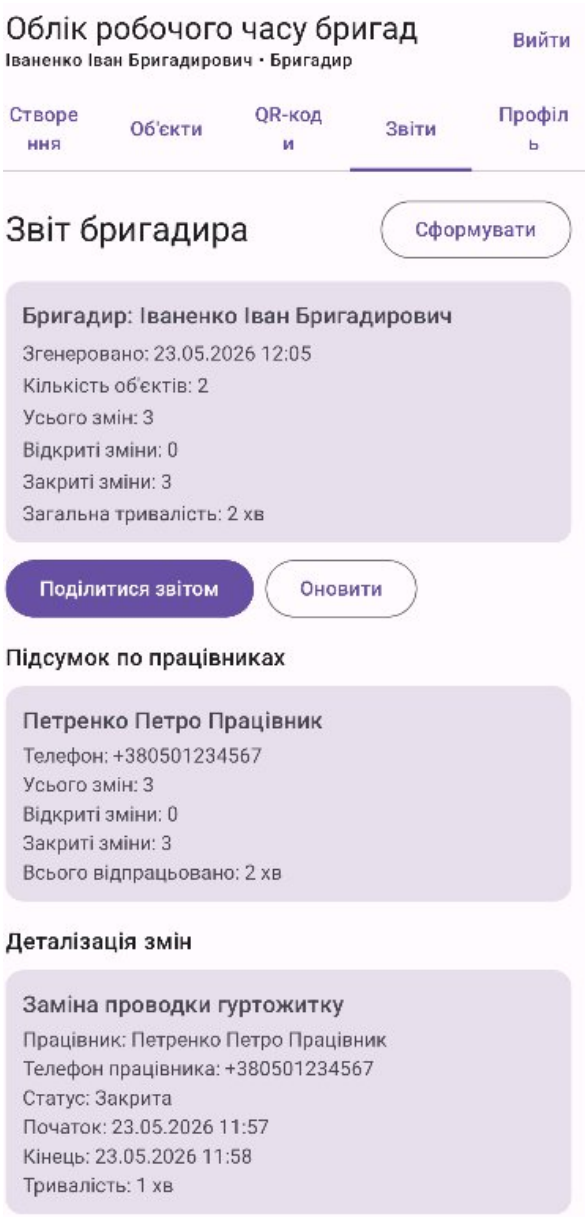


Рис 4.10 Перегляд звіту бригадира

Під час тестування звітності було перевірено, чи враховуються тільки ті робочі об’єкти, які належать відповідному бригадиру, і чи правильно обчислюється тривалість закритих змін. Це підтверджує коректність зв’язку між користувачем, робочими об’єктами, змінами та звітними даними.

За результатами тестування встановлено, що основні функції системи працюють відповідно до очікуваної логіки. Система забезпечує авторизацію користувачів, розмежування ролей, створення робочих

об’єктів, генерацію QR-кодів, перевірку геолокації, відкриття та закриття змін, локальне збереження подій і синхронізацію із сервером. Отримані результати підтверджують готовність програмного забезпечення до дослідної експлуатації.

4.2 Вимоги до апаратного та програмного забезпечення

Для впровадження та експлуатації програмного забезпечення сервісу обліку робочого часу бригад необхідно забезпечити роботу трьох основних частин системи: мобільного Android-застосунку, серверної частини та серверної бази даних. Система має клієнт-серверну архітектуру, тому мобільний застосунок не працює напряму з базою даних, а передає запити до серверного API.

Діаграма розгортання, наведена у Додатку Ж, показує розміщення мобільного застосунку, серверної частини та бази даних.[33] На ній видно, що користувачі системи: бригадир і працівник, працюють із мобільним застосунком CrewTimekeeping.apk, встановленим на Android-пристрій. Мобільний пристрій використовує камеру для сканування QR-коду, GPS для отримання координат і локальну базу SQLite/Room для тимчасового збереження відкладених подій сканування.

Згідно з Додатком Ж, мобільний застосунок обмінюється даними із сервером застосунку через мережеве з’єднання у форматі HTTPS / JSON. Сервер застосунку виконує FastAPI-застосунок, запущений через Uvicorn, обробляє запити клієнта, перевіряє авторизацію, виконує бізнес-логіку та передає дані до PostgreSQL. Сервер бази даних зберігає основні дані системи: користувачів, бригади, робочі об’єкти, QR-коди, зміни та події сканування.

Для стабільної роботи мобільного застосунку рекомендовано використовувати Android-пристрій із такими мінімальними характеристиками:

- операційна система Android 8.0 або новіша;
- оперативна пам’ять від 2 ГБ;
- вільне місце у внутрішній пам’яті від 200 МБ;
- наявність камери для сканування QR-кодів;
- наявність GPS-модуля для фіксації геолокації;
- доступ до інтернету через Wi-Fi або мобільну мережу;
- дозвіл застосунку на використання камери, геолокації та мережі.

Мобільний пристрій використовується працівником для відкриття та закриття робочої зміни, а бригадиром — для створення робочих об’єктів, перегляду QR-кодів і звітної інформації. Якщо підключення до сервера тимчасово відсутнє, застосунок зберігає подію сканування локально в SQLite/Room і передає її на сервер після відновлення зв’язку.

Для серверної частини рекомендовано використовувати такі мінімальні апаратні характеристики:

- процесор із 2 ядрами;
- оперативна пам’ять від 4 ГБ;
- накопичувач SSD від 20 ГБ;
- стабільне підключення до мережі;
- можливість відкриття порту для API-запитів;
- доступ до сервера бази даних PostgreSQL.

Сервер застосунку може бути розгорнутий на окремій фізичній машині, віртуальному сервері або локальному комп’ютері під час тестової експлуатації. У тестовому режимі сервер застосунку і сервер бази даних можуть працювати на одному пристрої. Для реального впровадження доцільніше розміщувати їх окремо, щоб зменшити навантаження та спростити адміністрування.

До програмного забезпечення серверної частини належать:

- Python 3.11 або сумісна версія;
- FastAPI для реалізації REST API;
- Uvicorn для запуску серверного застосунку;
- SQLAlchemy для роботи з базою даних;
- PostgreSQL для централізованого збереження даних;
- Docker за потреби для спрощення розгортання;
- pgAdmin або інший клієнт PostgreSQL для адміністрування бази даних.

Сервер бази даних повинен забезпечувати надійне збереження інформації. Для цього необхідно передбачити регулярне резервне копіювання, контроль доступу до PostgreSQL і обмеження прямого доступу до бази з боку мобільних клієнтів. Мобільний застосунок має взаємодіяти з даними тільки через серверне API.

До мережевого середовища висуваються такі вимоги:

- доступ мобільних пристроїв до сервера застосунку через інтернет або локальну мережу;
- використання захищеного з’єднання HTTPS для передавання авторизаційних і робочих даних;
- доступ сервера застосунку до PostgreSQL через внутрішню мережу або захищене підключення;
- стабільна робота API для синхронізації подій сканування;
- можливість повторної передачі даних після тимчасової втрати з’єднання.

Отже, для експлуатації системи необхідні Android-пристрої з камерою та GPS, сервер для запуску FastAPI-застосунку, сервер PostgreSQL і мережевий доступ між клієнтською та серверною частиною. Така конфігурація відповідає схемі розгортання, наведеній у Додатку Ж, і забезпечує роботу основних функцій системи: авторизацію користувачів,

створення робочих об’єктів, сканування QR-кодів, перевірку геолокації, облік змін і формування звітів.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення призначений для розгортання клієнтської та серверної частин сервісу обліку робочого часу бригад. Його склад визначено відповідно до діаграми розгортання, наведеної в Додатку Ж. Методичні рекомендації вимагають чітко визначати склад інсталяційного пакету, а для мобільного застосунку передбачати можливість встановлення і тестування на реальному пристрої у вигляді .apk-файлу.

До складу інсталяційного пакету розробленої системи входять такі компоненти:

- файл мобільного застосунку CrewTimekeeping.apk, який встановлюється на Android-пристрій бригадира або працівника;
- каталог backend/, який містить серверну частину системи на FastAPI;
- файл requirements.txt, у якому наведено Python-залежності серверної частини;
- файл main.py, який є точкою входу серверного застосунку;
- каталог routes/, що містить маршрути API для авторизації, робочих об’єктів, змін, синхронізації та звітності;
- файли services.py, auth.py, deps.py, які реалізують бізнес-логіку, авторизацію та службові залежності;
- файли database.py, models.py, schemas.py, які забезпечують підключення до бази даних, опис ORM-моделей і схем обміну даними;
- SQL-скрипт schema_postgres.sql, призначений для створення структури бази даних PostgreSQL;

- файл docker-compose.yml, який може використовуватися для швидкого запуску PostgreSQL у контейнері;[35]
- конфігураційний файл .env, у якому задаються параметри підключення до бази даних, секретний ключ і службові налаштування сервера;
- файл RUN_MANUAL.md або інструкція користувача, де описано порядок запуску серверної частини, встановлення мобільного застосунку та перевірки основних функцій.

Окремо до інсталяційного пакету може додаватися вихідний код Android-застосунку в каталозі android-app/. Він потрібний не для звичайного користувача, а для повторної збірки .apk-файлу, наприклад у разі зміни адреси серверного API. У поточній реалізації адреса сервера задається на етапі збірки застосунку, тому перед формуванням фінального APK необхідно вказати актуальну адресу серверної частини.

План інсталяції системи передбачає послідовне розгортання серверної частини, бази даних і мобільного застосунку.

Спочатку виконується підготовка сервера бази даних. Для цього встановлюється PostgreSQL або запускається контейнер через docker-compose.yml. Після запуску PostgreSQL створюється база даних crewtimekeeping, користувач бази даних і необхідні таблиці за допомогою SQL-скрипта schema_postgres.sql.

Далі виконується встановлення серверної частини. Каталог backend/ копіюється на сервер або хостинг. Після цього створюється віртуальне середовище Python, встановлюються залежності з файлу requirements.txt, налаштовується файл .env і запускається FastAPI-застосунок через Uvicorn.

Для експлуатаційного середовища серверну частину доцільно запускати як постійну службу або контейнер, щоб API автоматично відновлювалося після перезапуску сервера.

Після запуску серверної частини перевіряється доступність API. Для цього можна відкрити службову сторінку документації FastAPI або виконати тестовий запит до сервера. Якщо сервер відповідає, можна переходити до встановлення мобільного застосунку.

Перед встановленням мобільного застосунку необхідно переконатися, що файл CrewTimekeeping.apk зібраний із правильною адресою серверного API. Якщо застосунок встановлюється на реальний телефон, адреса сервера має бути доступною з цього пристрою через інтернет або локальну мережу.

Після цього файл CrewTimekeeping.apk передається на Android-пристрій і встановлюється. Під час першого запуску користувач повинен надати застосунку дозволи на використання камери та геолокації. Камера потрібна для сканування QR-кодів, а геолокація — для перевірки перебування працівника в межах допустимого радіуса робочого об’єкта.

Після інсталяції виконується перевірка працездатності системи. Бригадир входить у систему, створює робочий об’єкт і генерує QR-код. Працівник входить у застосунок, сканує QR-код, після чого система перевіряє геолокацію та відкриває або закриває робочу зміну. Якщо інтернет-з’єднання тимчасово відсутнє, подія сканування зберігається локально, а після відновлення зв’язку синхронізується із сервером.

Отже, інсталяційний пакет охоплює всі компоненти, необхідні для запуску системи: мобільний APK-файл, серверну частину, SQL-скрипт бази даних, конфігураційні файли та інструкцію з розгортання. Такий склад пакету дозволяє встановити систему на реальний Android-пристрій, розгорнути серверну частину та забезпечити взаємодію клієнта з базою даних через серверне API.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено програмне забезпечення сервісу обліку робочого часу бригад. Система призначена для фіксації початку та завершення робочої зміни працівника на конкретному робочому об’єкті.

На початку роботи проаналізовано предметну область і визначено, що ручний облік робочого часу має багато недоліків: дані можна легко записати з помилкою, важко підтвердити фактичну присутність працівника на об’єкті, а формування звітів займає додатковий час. Тому було вирішено створити сервіс, який автоматизує цей процес.

У роботі розглянуто існуючі рішення для обліку часу, зокрема системи з GPS-контролем, геозонами та мобільним табелюванням. На основі цього визначено, які функції потрібні для власної системи: авторизація користувачів, робота з ролями бригадира і працівника, створення робочих об’єктів, генерація QR-кодів, сканування QR-коду, перевірка геолокації, відкриття та закриття зміни, локальне збереження подій і формування звітів.

Було спроєктовано базу даних системи. Основними таблицями є користувачі, бригади, робочі об’єкти, QR-коди, зміни та події сканування. Для серверного збереження даних використано PostgreSQL, а для локального збереження відкладених подій на Android-пристрої — SQLite через Room. Такий підхід дозволяє не втрачати події сканування, якщо під час роботи немає стабільного інтернет-з’єднання.

Програмне забезпечення реалізовано за клієнт-серверною архітектурою. Клієнтська частина створена як Android-застосунок мовою Kotlin. Серверна частина реалізована мовою Python із використанням FastAPI. Обмін даними між мобільним застосунком і сервером виконується через REST API у форматі JSON.

У процесі розробки було реалізовано основні модулі системи: модуль авторизації, модуль роботи з робочими об’єктами, модуль створення QR-коду, модуль сканування, модуль перевірки координат, модуль відкриття та закриття зміни, модуль локального збереження, модуль синхронізації та модуль звітності.

Виконано моделювання системи за допомогою UML-діаграм. У роботі подано діаграми, які описують структуру бази даних, абстракції предметної області, взаємодію об’єктів, компонентну структуру та розгортання системи. Це допомогло показати не тільки кодову реалізацію, а й загальну логіку роботи програмного забезпечення.

Під час тестування було перевірено основні сценарії роботи системи: вхід користувача, створення робочого об’єкта, генерація QR-коду, сканування QR-коду працівником, відкриття і закриття зміни, перевірка перебування в межах геозони, локальне збереження події та формування звіту бригадира. За результатами тестування основні функції працюють відповідно до очікуваної логіки.

Також було визначено вимоги до апаратного та програмного забезпечення. Для роботи системи потрібен Android-пристрій із камерою, GPS і доступом до мережі, сервер для FastAPI-застосунку та база даних PostgreSQL. Окремо описано склад інсталяційного пакету, до якого входять APK-файл, серверна частина, SQL-скрипт бази даних і конфігураційні файли.

Отже, мету роботи досягнуто. Було створено програмне забезпечення, яке дозволяє бригадиру створювати робочі об’єкти та переглядати звіти, а працівнику — фіксувати робочий час через QR-код із перевіркою геолокації. Розроблена система може використовуватися в тестовому режимі для невеликих і середніх бригад, які працюють на будівельних, монтажних або сервісних об’єктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jibble. Time Tracking Software. URL: https://www.jibble.io/time-tracking-software (дата звернення: 12.03.2026).
2. Jibble. Time tracking software with geofencing. URL: https://www.jibble.io/time-tracking-software-geofencing (дата звернення: 12.03.2026).
3. Jibble. Track time without internet connection. URL: https://www.jibble.io/help/track-time-without-internet-connection (дата звернення: 13.03.2026).
4. ClockShark. GPS Time Tracking Software. URL: https://www.clockshark.com/gps-time-tracking-software (дата звернення: 14.03.2026).
5. ClockShark. Construction Time Tracking Software. URL: https://www.clockshark.com/construction-time-tracking-software (дата звернення: 14.03.2026).
6. ClockShark. Does ClockShark work offline? URL: https://help.clockshark.com/en/articles/397677-does-clockshark-work-offline (дата звернення: 15.03.2026).
7. Connecteam. Employee Time Clock App. URL: https://connecteam.com/employee-time-clock/ (дата звернення: 16.03.2026).
8. Connecteam. GPS Time Clock App. URL: https://connecteam.com/gps-time-clock-app/ (дата звернення: 16.03.2026).
9. IBM. What Is Database Normalization? URL: https://www.ibm.com/think/topics/database-normalization (дата звернення: 18.03.2026).
10. PostgreSQL Documentation. Constraints. URL: https://www.postgresql.org/docs/current/ddl-constraints.html (дата звернення: 19.03.2026).
11. PostgreSQL. About PostgreSQL. URL: https://www.postgresql.org/about/ (дата звернення: 20.03.2026).
12. PostgreSQL Documentation. CREATE TABLE. URL: https://www.postgresql.org/docs/current/sql-createtable.html (дата звернення: 20.03.2026).

13. Android Developers. Guide to app architecture. URL: https://developer.android.com/topic/architecture (дата звернення: 23.03.2026).
14. Kotlin Documentation. URL: https://kotlinlang.org/docs/home.html (дата звернення: 24.03.2026).
15. Android Developers. Build an offline-first app. URL: https://developer.android.com/topic/architecture/data-layer/offline-first (дата звернення: 24.03.2026).
16. Android Developers. Jetpack Compose. URL: https://developer.android.com/compose (дата звернення: 25.03.2026).
17. Android Developers. Save data in a local database using Room. URL: https://developer.android.com/training/data-storage/room (дата звернення: 26.03.2026).
18. Android Developers. WorkManager. URL: https://developer.android.com/jetpack/androidx/releases/work (дата звернення: 26.03.2026).
19. Android Developers. Request app permissions. URL: https://developer.android.com/training/permissions/requesting (дата звернення: 27.03.2026).
20. Google for Developers. Fused Location Provider API. URL: https://developers.google.com/location-context/fused-location-provider (дата звернення: 27.03.2026).
21. ZXing. QRCodeWriter API. URL: https://zxing.github.io/zxing/apidocs/com/google/zxing/qrcode/QRCodeWriter.html (дата звернення: 28.03.2026).
22. Retrofit. Introduction. URL: https://square.github.io/retrofit/ (дата звернення: 29.03.2026).
23. OkHttp. Interceptors. URL: https://square.github.io/okhttp/features/interceptors/ (дата звернення: 29.03.2026).
24. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 02.04.2026).
25. FastAPI Documentation. OAuth2 with Password and Bearer with JWT tokens. URL: https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/ (дата звернення: 03.04.2026).

26. SQLAlchemy Documentation. ORM Quick Start. URL: https://docs.sqlalchemy.org/orm/quickstart.html (дата звернення: 04.04.2026).
27. Pydantic Documentation. URL: https://pydantic.dev/docs/ (дата звернення: 05.04.2026).
28. PlantUML. Use Case Diagram Syntax. URL: https://plantuml.com/use-case-diagram (дата звернення: 08.04.2026).
29. PlantUML. Activity Diagram Syntax. URL: https://plantuml.com/activity-diagram-beta (дата звернення: 09.04.2026).
30. PlantUML. Sequence Diagram Syntax. URL: https://plantuml.com/sequence-diagram (дата звернення: 10.04.2026).
31. PlantUML. Class Diagram Syntax. URL: https://plantuml.com/class-diagram (дата звернення: 10.04.2026).
32. PlantUML. Component Diagram Syntax. URL: https://plantuml.com/component-diagram (дата звернення: 11.04.2026).
33. PlantUML. Deployment Diagram Syntax. URL: https://plantuml.com/deployment-diagram (дата звернення: 12.04.2026).
34. ISO. ISO/IEC/IEEE 29119-1:2013 Software and systems engineering — Software testing. URL: https://www.iso.org/standard/45142.html (дата звернення: 16.05.2026).
35. Docker Docs. Define and manage volumes in Docker Compose. URL: https://docs.docker.com/reference/compose-file/volumes/ (дата звернення: 20.05.2026).

ДОДАТКИ

ДОДАТОК А

Лістинги коду розробленої бази даних:

Лістинг А.1 Налаштування контейнера PostgreSQL для створення бази даних

```
services:
  postgres:
    image: postgres:16
    container_name: crew_timekeeping_postgres
    environment:
      POSTGRES_DB: crewtimekeeping
      POSTGRES_USER: crewuser
      POSTGRES_PASSWORD: crewpass
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data [35]
      - ./database/schema_postgres.sql:/docker-entrypoint-
initdb.d/schema_postgres.sql:ro

volumes:
  postgres_data:
```

Лістинг А.2 Налаштування підключення серверної частини до бази даних

```
from sqlalchemy import create_engine
    from sqlalchemy.orm import declarative_base, sessionmaker

    from app.config import settings

    connect_args = {"check_same_thread": False} if
settings.database_url.startswith("sqlite") else {}

    engine = create_engine(
        settings.database_url,
        echo=False,
        future=True,
        connect_args=connect_args
    )

    SessionLocal = sessionmaker(
        bind=engine,
        autoflush=False,
        autocommit=False,
        future=True
    )

    Base = declarative_base()
```

Лістинг А.3 Автоматичне створення таблиць під час запуску серверного застосунку

```
from contextlib import asynccontextmanager
from fastapi import FastAPI

from app.database import Base, engine

@asynccontextmanager
async def lifespan(app: FastAPI):
    Base.metadata.create_all(bind=engine)
    yield

app = FastAPI(
    title=settings.app_name,
    version="1.0.0",
    lifespan=lifespan
)
```

Лістинг А.4 Створення таблиці бригад

```
CREATE TABLE crews (
    id SERIAL PRIMARY KEY,
    name VARCHAR(120) NOT NULL UNIQUE,
    description TEXT,
    is_active BOOLEAN NOT NULL DEFAULT TRUE
);
```

Лістинг А.5 Створення таблиці користувачів

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    crew_id INTEGER REFERENCES crews(id) ON DELETE SET
NULL,
    login VARCHAR(64) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(120) NOT NULL,
    phone VARCHAR(32),
    position VARCHAR(64),
    role VARCHAR(20) NOT NULL CHECK (role IN ('FOREMAN',
'WORKER')),
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
```

Лістинг А.6 Створення таблиці робочих об’єктів

```
CREATE TABLE work_objects (
    id SERIAL PRIMARY KEY,
    crew_id INTEGER REFERENCES crews(id) ON DELETE SET
NULL,
    created_by_user_id INTEGER NOT NULL REFERENCES users(id)
ON DELETE RESTRICT,
    name VARCHAR(150) NOT NULL,
    address VARCHAR(255) NOT NULL,
    latitude DOUBLE PRECISION NOT NULL,
    longitude DOUBLE PRECISION NOT NULL,
```

```
    geofence_radius_m DOUBLE PRECISION NOT NULL CHECK
(geofence_radius_m > 0),
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
```

Лістинг А.7 Створення таблиці QR-кодів

```
CREATE TABLE qr_codes (
    id SERIAL PRIMARY KEY,
    work_object_id INTEGER NOT NULL REFERENCES
work_objects(id) ON DELETE CASCADE,
    token VARCHAR(120) NOT NULL UNIQUE,
    payload_json TEXT NOT NULL,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
```

Лістинг А.8 Створення таблиці змін

```
CREATE TABLE shifts (
    id SERIAL PRIMARY KEY,
    worker_id INTEGER NOT NULL REFERENCES users(id) ON
DELETE RESTRICT,
    work_object_id INTEGER NOT NULL REFERENCES
work_objects(id) ON DELETE RESTRICT,
```

```
    qr_code_id INTEGER NOT NULL REFERENCES qr_codes(id) ON
DELETE RESTRICT,
    opened_at TIMESTAMP NOT NULL,
    closed_at TIMESTAMP NULL,
    status VARCHAR(20) NOT NULL CHECK (status IN ('OPEN',
'CLOSED'))
);
```

Лістинг А.9 Створення таблиці подій сканування

```
CREATE TABLE scan_events (
  id SERIAL PRIMARY KEY,
  shift_id INTEGER REFERENCES shifts(id) ON DELETE SET
NULL,
  worker_id INTEGER NOT NULL REFERENCES users(id) ON
DELETE RESTRICT,
  qr_code_id INTEGER NOT NULL REFERENCES qr_codes(id) ON
DELETE RESTRICT,
  event_type VARCHAR(10) NOT NULL CHECK (event_type IN
('OPEN', 'CLOSE', 'REJECT')),
  occurred_at TIMESTAMP NOT NULL,
  latitude DOUBLE PRECISION NOT NULL,
  longitude DOUBLE PRECISION NOT NULL,
  accuracy_m DOUBLE PRECISION NULL,
  distance_to_object_m DOUBLE PRECISION NOT NULL,
  within_geofence BOOLEAN NOT NULL,
  sync_status VARCHAR(20) NOT NULL DEFAULT 'PROCESSED'
    CHECK (sync_status IN ('RECEIVED', 'PROCESSED', 'FAILED')),
  source VARCHAR(20) NOT NULL DEFAULT 'MOBILE',
```

```
    raw_payload TEXT NULL
);
```

Лістинг А.10 Створення індексів у серверній базі даних

```
CREATE INDEX idx_users_login ON users(login);
CREATE INDEX idx_shifts_worker_status ON shifts(worker_id, status);
CREATE INDEX idx_scan_events_shift_id ON scan_events(shift_id);
```

Лістинг А.11 ORM-модель таблиці бригад

```
class Crew(Base):
    __tablename__ = "crews"

    id: Mapped[int] = mapped_column(Integer, primary_key=True,
index=True)
    name: Mapped[str] = mapped_column(String(120), nullable=False,
unique=True)
    description: Mapped[str | None] = mapped_column(Text,
nullable=True)
    is_active: Mapped[bool] = mapped_column(Boolean, default=True,
nullable=False)

    members: Mapped[list["User"]] =
relationship(back_populates="crew")
    work_objects: Mapped[list["WorkObject"]] =
relationship(back_populates="crew")
```

Лістинг А.12 ORM-модель таблиці користувачів

```
class User(Base):
    __tablename__ = "users"

    id: Mapped[int] = mapped_column(Integer, primary_key=True, index=True)
    crew_id: Mapped[int | None] = mapped_column(ForeignKey("crews.id"), nullable=True)
    login: Mapped[str] = mapped_column(String(64), nullable=False, unique=True, index=True)
    password_hash: Mapped[str] = mapped_column(String(255), nullable=False)
    full_name: Mapped[str] = mapped_column(String(120), nullable=False)
    phone: Mapped[str | None] = mapped_column(String(32), nullable=True)
    position: Mapped[str | None] = mapped_column(String(64), nullable=True)
    role: Mapped[UserRole] = mapped_column(Enum(UserRole), nullable=False)
    is_active: Mapped[bool] = mapped_column(Boolean, default=True, nullable=False)
    created_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.utcnow, nullable=False)

    crew: Mapped["Crew | None"] = relationship(back_populates="members")
    created_objects: Mapped[list["WorkObject"]] = relationship(back_populates="created_by_user")
```

```
    shifts: Mapped[list["Shift"]] = relationship(back_populates="worker")
    scan_events: Mapped[list["ScanEvent"]] =
relationship(back_populates="worker")
```

Лістинг А.13 ORM-модель таблиці змін

```
class Shift(Base):
    __tablename__ = "shifts"

    id: Mapped[int] = mapped_column(Integer, primary_key=True,
index=True)
    worker_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
nullable=False, index=True)
    work_object_id: Mapped[int] =
mapped_column(ForeignKey("work_objects.id"), nullable=False, index=True)
    qr_code_id: Mapped[int] =
mapped_column(ForeignKey("qr_codes.id"), nullable=False)
    opened_at: Mapped[datetime] = mapped_column(DateTime,
nullable=False)
    closed_at: Mapped[datetime | None] = mapped_column(DateTime,
nullable=True)
    status: Mapped[ShiftStatus] = mapped_column(Enum(ShiftStatus),
default=ShiftStatus.OPEN, nullable=False)

    worker: Mapped["User"] = relationship(back_populates="shifts")
    work_object: Mapped["WorkObject"] =
relationship(back_populates="shifts")
    qr_code: Mapped["QrCode"] = relationship(back_populates="shifts")
```

```
    scan_events: Mapped[list["ScanEvent"]] =
relationship(back_populates="shift")
```

Лістинг А.14 Опис локальної таблиці відкладених подій сканування

```
package com.example.crewtimekeeping.data.local

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "pending_scans")
data class PendingScanEntity(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val qrPayloadJson: String,
    val occurredAtIso: String,
    val latitude: Double,
    val longitude: Double,
    val accuracyM: Double?,
    val source: String = "OFFLINE"
)
```

Лістинг А.15 DAO-інтерфейс для роботи з локальними подіями

```
package com.example.crewtimekeeping.data.local

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
```

```
import androidx.room.Query

@Dao
interface PendingScanDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(entity: PendingScanEntity)

    @Query("SELECT * FROM pending_scans ORDER BY id ASC")
    suspend fun getAll(): List<PendingScanEntity>

    @Query("DELETE FROM pending_scans WHERE id IN (:ids)")
    suspend fun deleteByIds(ids: List<Long>)

    @Query("SELECT COUNT(*) FROM pending_scans")
    suspend fun count(): Int
}
```

Лістинг А.16 Опис локальної бази даних Room

```
package com.example.crewtimekeeping.data.local

import androidx.room.Database
import androidx.room.RoomDatabase

@Database(
    entities = [PendingScanEntity::class],
    version = 1,
    exportSchema = false
)
```

```
abstract class AppDatabase : RoomDatabase() {
    abstract fun pendingScanDao(): PendingScanDao
}
```

Лістинг А.17 Збереження події сканування у локальну базу при відсутності зв’язку

```
suspend fun syncScanOrQueue(
    qrPayloadJson: String,
    occurredAtIso: String,
    latitude: Double,
    longitude: Double,
    accuracy: Double?
): Result<ScanSyncResponse> {
    return try {
        val response = api.syncScan(
            ScanSyncRequest(
                qr_payload_json = qrPayloadJson,
                occurred_at = occurredAtIso,
                latitude = latitude,
                longitude = longitude,
                accuracy_m = accuracy,
                source = "ONLINE"
            )
        )
        Result.success(response)
    } catch (ex: Exception) {
        pendingScanDao.insert(
            PendingScanEntity(
```

```
                qrPayloadJson = qrPayloadJson,
                occurredAtIso = occurredAtIso,
                latitude = latitude,
                longitude = longitude,
                accuracyM = accuracy,
                source = "OFFLINE"
            )
        )
        Result.failure(ex)
    }
}
```

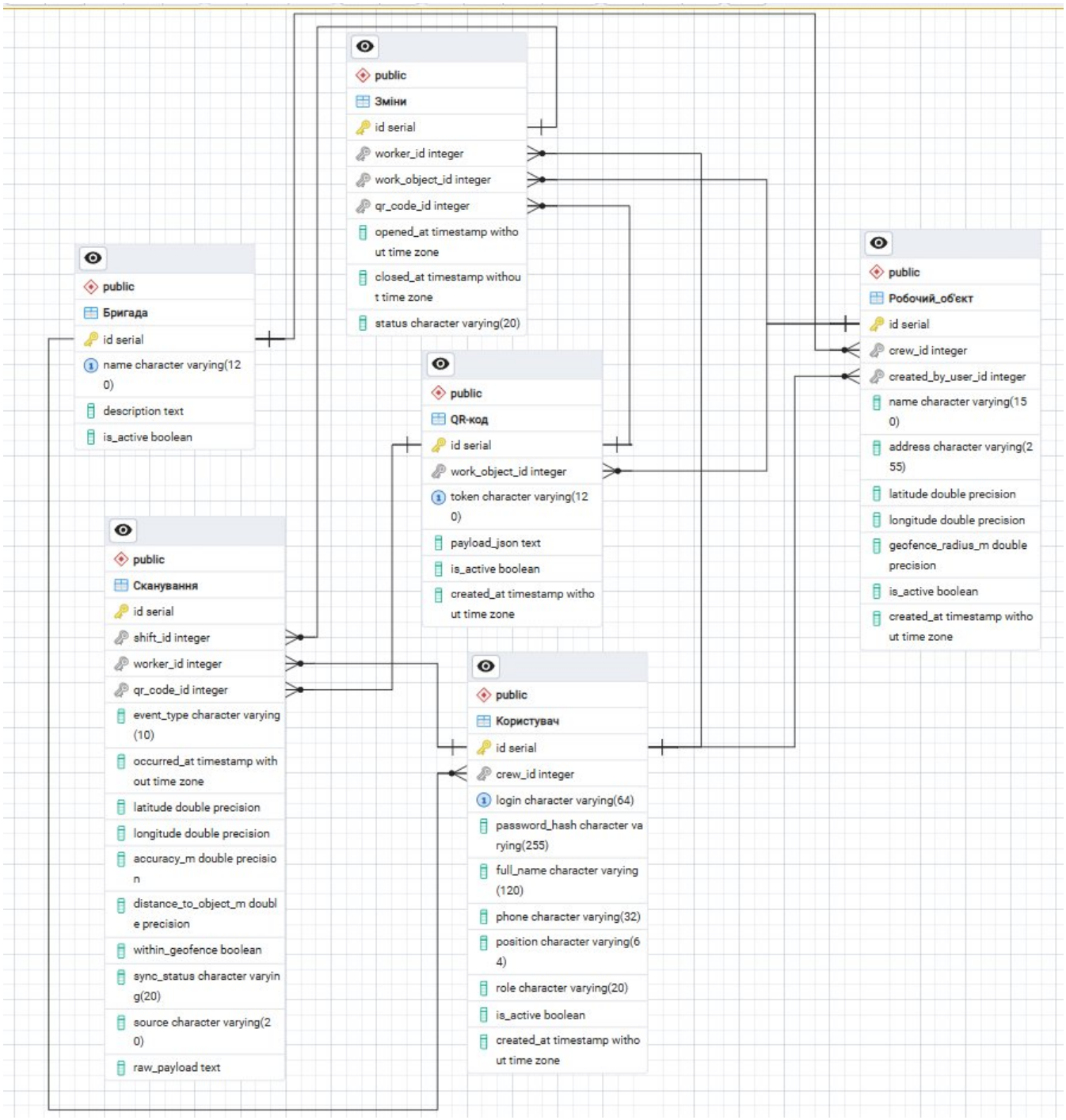
Лістинг А.18 Синхронізація локально збережених подій із сервером

```
suspend fun syncPendingScans(): Int {
    sessionManager.token() ?: return 0
    val pending = pendingScanDao.getAll()
    if (pending.isEmpty()) return 0

    val syncedIds = mutableListOf<Long>()
    for (item in pending) {
        try {
            api.syncScan(
                ScanSyncRequest(
                    qr_payload_json = item.qrPayloadJson,
                    occurred_at = item.occurredAtIso,
                    latitude = item.latitude,
                    longitude = item.longitude,
```

```
                accuracy_m = item.accuracyM,
                source = item.source
            )
        )
        syncedIds += item.id
    } catch (_: Exception) {
        // запис залишається в локальній базі для наступної спроби
    }
}

if (syncedIds.isNotEmpty()) {
    pendingScanDao.deleteByIds(syncedIds)
}
return syncedIds.size
}
```

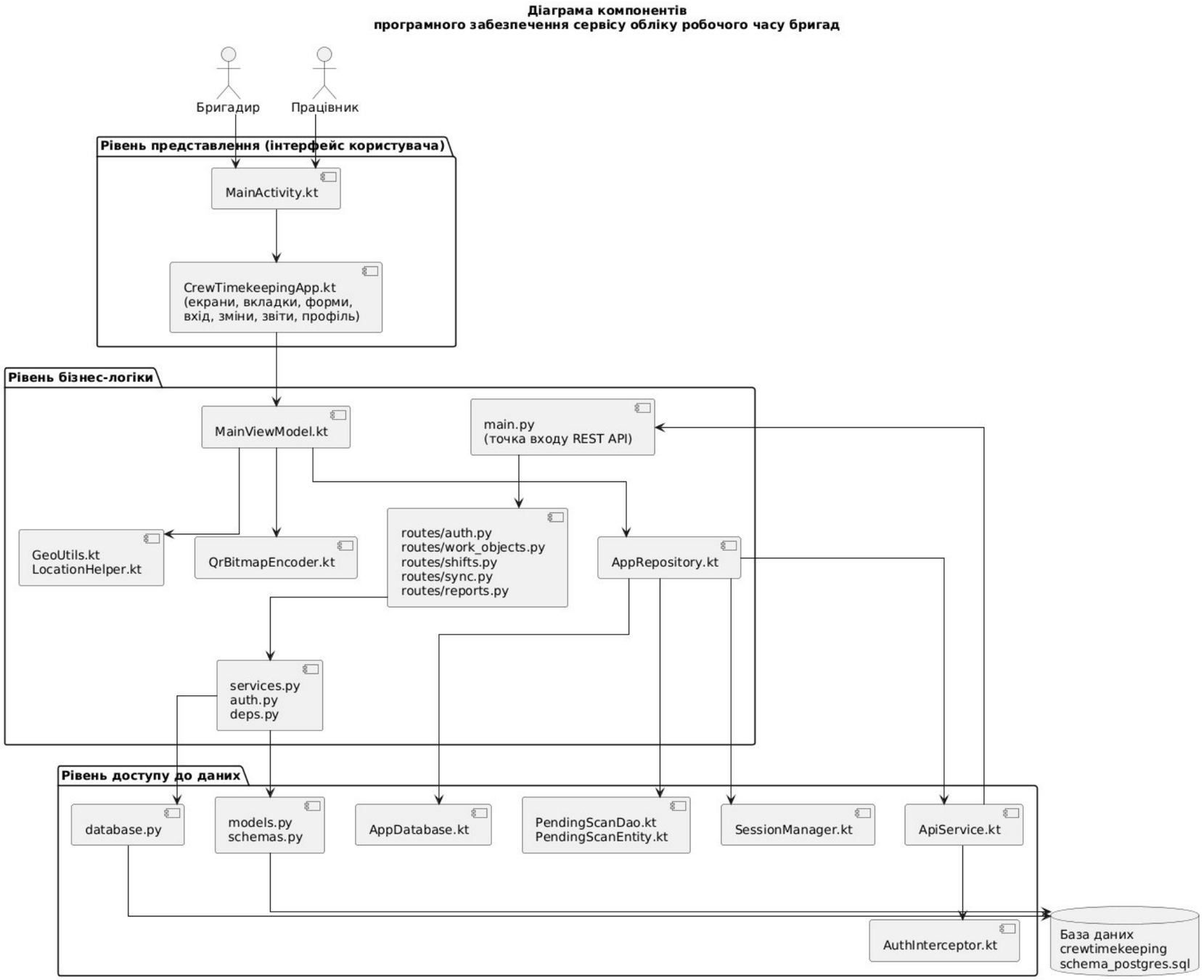
ДОДАТОК Б

Логічна модель бази даних



ДОДАТОК В

Діаграма компонентів



ДОДАТОК Г

Лістинги коду розробленої бази даних:

лістинг Е.1 ORM-моделі серверної частини

```
class UserRole(str, enum.Enum):
    FOREMAN = "FOREMAN"
    WORKER = "WORKER"

class ShiftStatus(str, enum.Enum):
    OPEN = "OPEN"
    CLOSED = "CLOSED"

class SyncStatus(str, enum.Enum):
    RECEIVED = "RECEIVED"
    PROCESSED = "PROCESSED"
    FAILED = "FAILED"

class Crew(Base):
    __tablename__ = "crews"
    id = mapped_column(Integer, primary_key=True, index=True)
    name = mapped_column(String(120), nullable=False, unique=True)
    description = mapped_column(Text, nullable=True)
    is_active = mapped_column(Boolean, default=True, nullable=False)

class User(Base):
    __tablename__ = "users"
    id = mapped_column(Integer, primary_key=True, index=True)
```

```
    crew_id = mapped_column(ForeignKey("crews.id"), nullable=True)
    login = mapped_column(String(64), nullable=False, unique=True,
index=True)
    password_hash = mapped_column(String(255), nullable=False)
    full_name = mapped_column(String(120), nullable=False)
    phone = mapped_column(String(32), nullable=True)
    position = mapped_column(String(64), nullable=True)
    role = mapped_column(Enum(UserRole), nullable=False)
    is_active = mapped_column(Boolean, default=True, nullable=False)
    created_at = mapped_column(DateTime, default=datetime.utcnow,
nullable=False)

class WorkObject(Base):
    __tablename__ = "work_objects"
    id = mapped_column(Integer, primary_key=True, index=True)
    crew_id = mapped_column(ForeignKey("crews.id"), nullable=True)
    created_by_user_id = mapped_column(ForeignKey("users.id"),
nullable=False)
    name = mapped_column(String(150), nullable=False)
    address = mapped_column(String(255), nullable=False)
    latitude = mapped_column(Float, nullable=False)
    longitude = mapped_column(Float, nullable=False)
    geofence_radius_m = mapped_column(Float, nullable=False)
    is_active = mapped_column(Boolean, default=True, nullable=False)
    created_at = mapped_column(DateTime, default=datetime.utcnow,
nullable=False)

class QrCode(Base):
    __tablename__ = "qr_codes"
```

```
    id = mapped_column(Integer, primary_key=True, index=True)
    work_object_id = mapped_column(ForeignKey("work_objects.id"),
nullable=False)
    token = mapped_column(String(120), nullable=False, unique=True)
    payload_json = mapped_column(Text, nullable=False)
    is_active = mapped_column(Boolean, default=True, nullable=False)
    created_at = mapped_column(DateTime, default=datetime.utcnow,
nullable=False)

class Shift(Base):
    __tablename__ = "shifts"
    id = mapped_column(Integer, primary_key=True, index=True)
    worker_id = mapped_column(ForeignKey("users.id"), nullable=False,
index=True)
    work_object_id = mapped_column(ForeignKey("work_objects.id"),
nullable=False)
    qr_code_id = mapped_column(ForeignKey("qr_codes.id"), nullable=False)
    opened_at = mapped_column(DateTime, nullable=False)
    closed_at = mapped_column(DateTime, nullable=True)
    status = mapped_column(Enum(ShiftStatus), default=ShiftStatus.OPEN,
nullable=False)

class ScanEvent(Base):
    __tablename__ = "scan_events"
    id = mapped_column(Integer, primary_key=True, index=True)
    shift_id = mapped_column(ForeignKey("shifts.id"), nullable=True)
    worker_id = mapped_column(ForeignKey("users.id"), nullable=False)
    qr_code_id = mapped_column(ForeignKey("qr_codes.id"), nullable=False)
    event_type = mapped_column(String(10), nullable=False)
```

```
    occurred_at = mapped_column(DateTime, nullable=False)
    latitude = mapped_column(Float, nullable=False)
    longitude = mapped_column(Float, nullable=False)
    accuracy_m = mapped_column(Float, nullable=True)
    distance_to_object_m = mapped_column(Float, nullable=False)
    within_geofence = mapped_column(Boolean, nullable=False)
    sync_status = mapped_column(Enum(SyncStatus),
default=SyncStatus.PROCESSED)
    source = mapped_column(String(20), default="MOBILE", nullable=False)
    raw_payload = mapped_column(Text, nullable=True)
```

лістинг Е.2 Схеми обміну даними API

```
class TokenResponse(BaseModel):
    access_token: str
    token_type: str = "bearer"

class LoginRequest(BaseModel):
    login: str
    password: str

class RegisterRequest(BaseModel):
    login: str = Field(min_length=3, max_length=64)
    password: str = Field(min_length=6, max_length=72)
    full_name: str = Field(min_length=2, max_length=120)
    role: Literal["FOREMAN", "WORKER"] = "WORKER"
    phone: str | None = Field(default=None, max_length=32)
    position: str | None = Field(default=None, max_length=64)
```

```
class UserResponse(BaseModel):
    id: int
    login: str
    full_name: str
    phone: str | None = None
    position: str | None = None
    role: str
    crew_id: int | None = None
    model_config = ConfigDict(from_attributes=True)

class WorkObjectCreateRequest(BaseModel):
    name: str = Field(min_length=2, max_length=150)
    address: str = Field(min_length=4, max_length=255)
    latitude: float
    longitude: float
    geofence_radius_m: float = Field(gt=5, le=5000)
    crew_id: int | None = None

class QrPayload(BaseModel):
    token: str
    work_object_id: int
    name: str
    latitude: float
    longitude: float
    radius_m: float

class ScanSyncRequest(BaseModel):
    qr_payload_json: str
    occurred_at: datetime
```

```
    latitude: float
    longitude: float
    accuracy_m: float | None = None
    source: Literal["ONLINE", "OFFLINE"] = "ONLINE"

class ScanSyncResponse(BaseModel):
    action: Literal["OPENED", "CLOSED", "REJECTED"]
    message: str
    shift_id: int | None = None
    within_geofence: bool
    distance_to_object_m: float

class ForemanReportItem(BaseModel):
    shift_id: int
    worker_name: str
    worker_phone: str | None = None
    work_object_name: str
    opened_at: datetime
    closed_at: datetime | None = None
    status: str
    duration_minutes: int | None = None

class ForemanReportResponse(BaseModel):
    generated_at: datetime
    foreman_name: str
    total_objects: int
    total_shifts: int
    open_shifts: int
    closed_shifts: int
```

```
    total_duration_minutes: int
    items: list[ForemanReportItem]
```

лістинг Е.3 Основні серверні алгоритми

```
def haversine_distance_m(lat1, lon1, lat2, lon2):
    r = 6371000
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    d_phi = math.radians(lat2 - lat1)
    d_lambda = math.radians(lon2 - lon1)
    a = math.sin(d_phi / 2) ** 2 + math.cos(phi1) * math.cos(phi2) *
math.sin(d_lambda / 2) ** 2
    return 2 * r * math.atan2(math.sqrt(a), math.sqrt(1 - a))

def create_work_object(db, current_user, payload):
    if current_user.role != UserRole.FOREMAN:
        raise HTTPException(status_code=403, detail="Only foreman can create
work objects")

    work_object = WorkObject(
        crew_id=payload.crew_id or current_user.crew_id,
        created_by_user_id=current_user.id,
        name=payload.name,
        address=payload.address,
        latitude=payload.latitude,
        longitude=payload.longitude,
        geofence_radius_m=payload.geofence_radius_m
    )
```

```
db.add(work_object)
db.flush()

token = str(uuid.uuid4())
qr_payload = QrPayload(
    token=token,
    work_object_id=work_object.id,
    name=work_object.name,
    latitude=work_object.latitude,
    longitude=work_object.longitude,
    radius_m=work_object.geofence_radius_m
)
db.add(QrCode(
    work_object_id=work_object.id,
    token=token,
    payload_json=qr_payload.model_dump_json()
))
db.commit()

return WorkObjectResponse(
    id=work_object.id,
    name=work_object.name,
    address=work_object.address,
    latitude=work_object.latitude,
    longitude=work_object.longitude,
    geofence_radius_m=work_object.geofence_radius_m,
    qr_payload_json=qr_payload.model_dump_json(),
    qr_token=token
)
```

```
def process_scan_event(db, current_user, payload):
    if current_user.role != UserRole.WORKER:
        raise HTTPException(status_code=403, detail="Only workers can send
scan events")

    qr_payload = QrPayload.model_validate_json(payload.qr_payload_json)
    qr_code = db.execute(
        select(QrCode).where(QrCode.token == qr_payload.token,
QrCode.is_active.is_(True))
    ).scalar_one_or_none()

    if not qr_code:
        raise HTTPException(status_code=404, detail="QR code not found")

    work_object = db.get(WorkObject, qr_code.work_object_id)
    distance = haversine_distance_m(
        payload.latitude,
        payload.longitude,
        work_object.latitude,
        work_object.longitude
    )
    within_geofence = distance <= work_object.geofence_radius_m

    if not within_geofence:
        db.add(ScanEvent(
            shift_id=None,
            worker_id=current_user.id,
            qr_code_id=qr_code.id,
            event_type="REJECT",
```

```
            occurred_at=payload.occurred_at,
            latitude=payload.latitude,
            longitude=payload.longitude,
            distance_to_object_m=distance,
            within_geofence=False,
            source=payload.source,
            raw_payload=payload.qr_payload_json
        ))
        db.commit()
        return ScanSyncResponse(
            action="REJECTED",
            message="Працівник знаходиться поза межами допустимої
геозони.",
            shift_id=None,
            within_geofence=False,
            distance_to_object_m=round(distance, 2)
        )

    active_shift = db.execute(
        select(Shift).where(
            Shift.worker_id == current_user.id,
            Shift.work_object_id == work_object.id,
            Shift.status == ShiftStatus.OPEN
        )
    ).scalar_one_or_none()

    if active_shift is None:
        shift = Shift(
            worker_id=current_user.id,
```

```
            work_object_id=work_object.id,
            qr_code_id=qr_code.id,
            opened_at=payload.occurred_at,
            status=ShiftStatus.OPEN
        )
        db.add(shift)
        db.flush()
        event_type = "OPEN"
        action = "OPENED"
        message = "Зміну успішно відкрито."
    else:
        active_shift.closed_at = payload.occurred_at
        active_shift.status = ShiftStatus.CLOSED
        shift = active_shift
        event_type = "CLOSE"
        action = "CLOSED"
        message = "Зміну успішно закрито."

    db.add(ScanEvent(
        shift_id=shift.id,
        worker_id=current_user.id,
        qr_code_id=qr_code.id,
        event_type=event_type,
        occurred_at=payload.occurred_at,
        latitude=payload.latitude,
        longitude=payload.longitude,
        distance_to_object_m=distance,
        within_geofence=True,
        source=payload.source,
```

```
        raw_payload=payload.qr_payload_json
    ))
    db.commit()

    return ScanSyncResponse(
        action=action,
        message=message,
        shift_id=shift.id,
        within_geofence=True,
        distance_to_object_m=round(distance, 2)
    )
```

лістинг Е.4 Моделі даних Android-клієнта

```
data class TokenResponse(val access_token: String, val token_type: String)

data class LoginRequest(val login: String, val password: String)

data class RegisterRequest(
    val login: String,
    val password: String,
    val full_name: String,
    val role: String,
    val phone: String? = null,
    val position: String? = null
)

data class UserResponse(
    val id: Int,
    val login: String,
```

```
    val full_name: String,
    val phone: String?,
    val position: String?,
    val role: String,
    val crew_id: Int?
)

data class WorkObjectCreateRequest(
    val name: String,
    val address: String,
    val latitude: Double,
    val longitude: Double,
    val geofence_radius_m: Double,
    val crew_id: Int? = null
)

data class WorkObjectResponse(
    val id: Int,
    val name: String,
    val address: String,
    val latitude: Double,
    val longitude: Double,
    val geofence_radius_m: Double,
    val qr_payload_json: String,
    val qr_token: String
)

data class ShiftResponse(
    val id: Int,
```

```
    val worker_id: Int,
    val work_object_id: Int,
    val work_object_name: String,
    val foreman_name: String,
    val foreman_phone: String?,
    val opened_at: String,
    val closed_at: String?,
    val status: String
)

data class ScanSyncRequest(
    val qr_payload_json: String,
    val occurred_at: String,
    val latitude: Double,
    val longitude: Double,
    val accuracy_m: Double?,
    val source: String
)

data class ScanSyncResponse(
    val action: String,
    val message: String,
    val shift_id: Int?,
    val within_geofence: Boolean,
    val distance_to_object_m: Double
)
```

лістинг Е.5 Інтерфейс обміну з REST API

```
interface ApiService {
    @POST("auth/login")
    suspend fun login(@Body body: LoginRequest): TokenResponse

    @POST("auth/register")
    suspend fun register(@Body body: RegisterRequest): TokenResponse

    @GET("auth/me")
    suspend fun me(): UserResponse

    @PATCH("auth/me")
    suspend fun updateMe(@Body body: UpdateProfileRequest): UserResponse

    @POST("work-objects")
    suspend fun createWorkObject(@Body body: WorkObjectCreateRequest):
WorkObjectResponse

    @GET("work-objects/mine")
    suspend fun myObjects(): List<WorkObjectResponse>

    @GET("shifts/me")
    suspend fun myShifts(): List<ShiftResponse>

    @POST("sync/scan-event")
    suspend fun syncScan(@Body body: ScanSyncRequest): ScanSyncResponse

    @GET("reports/foreman")
    suspend fun foremanReport(): ForemanReportResponse
}
```

лістинг Е.6 Локальне збереження відкладених сканувань

```
@Entity(tableName = "pending_scans")
data class PendingScanEntity(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val qrPayloadJson: String,
    val occurredAtIso: String,
    val latitude: Double,
    val longitude: Double,
    val accuracyM: Double?,
    val source: String = "OFFLINE"
)

@Dao
interface PendingScanDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(entity: PendingScanEntity)

    @Query("SELECT * FROM pending_scans ORDER BY id ASC")
    suspend fun getAll(): List<PendingScanEntity>

    @Query("DELETE FROM pending_scans WHERE id IN (:ids)")
    suspend fun deleteByIds(ids: List<Long>)

    @Query("SELECT COUNT(*) FROM pending_scans")
    suspend fun count(): Int
}

@Database(
    entities = [PendingScanEntity::class],
```

```
        version = 1,
        exportSchema = false
    )
    abstract class AppDatabase : RoomDatabase() {
        abstract fun pendingScanDao(): PendingScanDao
    }
```

лістинг Е.7 Репозиторій мобільного застосунку

```
class AppRepository(
    private val api: ApiService,
    private val pendingScanDao: PendingScanDao,
    private val sessionManager: SessionManager
) {
    suspend fun login(login: String, password: String): Result<UserResponse> =
runCatching {
        val token = api.login(LoginRequest(login, password))
        sessionManager.saveSession(token.access_token, -1, "", "")
        val me = api.me()
        sessionManager.saveSession(token.access_token, me.id, me.full_name,
me.role, me.login)
        me
    }

    suspend fun createWorkObject(
        name: String,
        address: String,
        latitude: Double,
        longitude: Double,
```

```
    radius: Double
): Result<WorkObjectResponse> = runCatching {
    api.createWorkObject(
        WorkObjectCreateRequest(name, address, latitude, longitude, radius)
    )
}

suspend fun syncScanOrQueue(
    qrPayloadJson: String,
    occurredAtIso: String,
    latitude: Double,
    longitude: Double,
    accuracy: Double?
): Result<ScanSyncResponse> {
    return try {
        val response = api.syncScan(
            ScanSyncRequest(
                qr_payload_json = qrPayloadJson,
                occurred_at = occurredAtIso,
                latitude = latitude,
                longitude = longitude,
                accuracy_m = accuracy,
                source = "ONLINE"
            )
        )
        Result.success(response)
    } catch (ex: Exception) {
        pendingScanDao.insert(
            PendingScanEntity(
```

```
                qrPayloadJson = qrPayloadJson,
                occurredAtIso = occurredAtIso,
                latitude = latitude,
                longitude = longitude,
                accuracyM = accuracy,
                source = "OFFLINE"
            )
        )
        Result.failure(ex)
    }
}

suspend fun syncPendingScans(): Int {
    val pending = pendingScanDao.getAll()
    val syncedIds = mutableListOf<Long>()

    for (item in pending) {
        try {
            api.syncScan(
                ScanSyncRequest(
                    qr_payload_json = item.qrPayloadJson,
                    occurred_at = item.occurredAtIso,
                    latitude = item.latitude,
                    longitude = item.longitude,
                    accuracy_m = item.accuracyM,
                    source = item.source
                )
            )
            syncedIds += item.id
```

```
            } catch (_: Exception) {
            }
        }

        if (syncedIds.isNotEmpty()) {
            pendingScanDao.deleteByIds(syncedIds)
        }
        return syncedIds.size
    }
}
```

лістинг Е.8 Модель стану інтерфейсу та ViewModel

```
data class AppUiState(
    val isLoading: Boolean = false,
    val isLoggedIn: Boolean = false,
    val fullName: String = "",
    val login: String = "",
    val phone: String = "",
    val position: String = "",
    val role: String = "",
    val message: String? = null,
    val foremanObjects: List<WorkObjectResponse> = emptyList(),
    val shifts: List<ShiftResponse> = emptyList(),
    val selectedQrPayload: String? = null,
    val selectedWorkObject: WorkObjectResponse? = null,
    val foremanReport: ForemanReportResponse? = null,
    val pendingCount: Int = 0
)
```

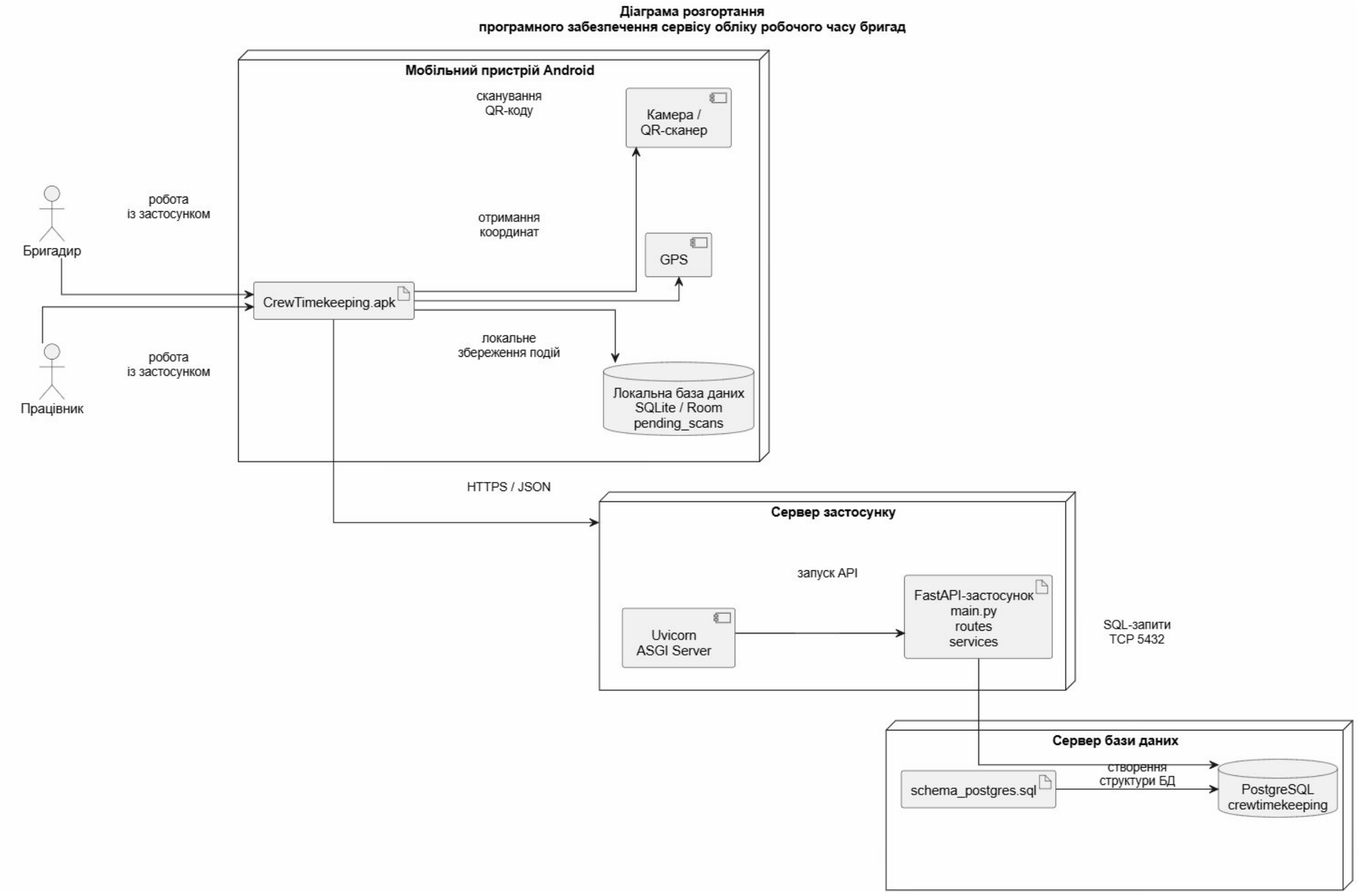
```
class MainViewModel(
    private val repository: AppRepository
) : ViewModel() {
    private val _uiState = MutableStateFlow(AppUiState())
    val uiState: StateFlow<AppUiState> = _uiState.asStateFlow()

    fun login(login: String, password: String) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true)
            repository.login(login, password)
                .onSuccess { user ->
                    _uiState.value = _uiState.value.copy(
                        isLoading = false,
                        isLoggedIn = true,
                        fullName = user.full_name,
                        login = user.login,
                        role = user.role,
                        message = "Вхід успішний."
                    )
                }
                .onFailure {
                    _uiState.value = _uiState.value.copy(
                        isLoading = false,
                        message = "Помилка входу."
                    )
                }
        }
    }
```

```
    fun createWorkObject(name: String, address: String, latitude: Double,
longitude: Double, radius: Double) {
        viewModelScope.launch {
            repository.createWorkObject(name, address, latitude, longitude, radius)
                .onSuccess { item ->
                    _uiState.value = _uiState.value.copy(
                        selectedQrPayload = item.qr_payload_json,
                        selectedWorkObject = item,
                        message = "Об'єкт створено. QR готовий до друку."
                    )
                }
        }
    }

    fun processQrScan(rawQr: String, location: LocationSnapshot) {
        viewModelScope.launch {
            val payload = GeoUtils.parseQrPayload(rawQr)
            val distance = GeoUtils.haversineDistanceMeters(
                location.latitude,
                location.longitude,
                payload.latitude,
                payload.longitude
            )

            if (distance > payload.radiusM) {
                _uiState.value = _uiState.value.copy(
                    message = "Працівник поза допустимою геозоною."
                )
```

```
                return@launch
            }

            repository.syncScanOrQueue(
                qrPayloadJson = rawQr,
                occurredAtIso = Instant.now().toString(),
                latitude = location.latitude,
                longitude = location.longitude,
                accuracy = location.accuracy
            )
        }
    }
}
```

ДОДАТОК Ж

Діаграма розгортання



ДОДАТОК К

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ
ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Коваленко Богдан Ігорович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення сервісу обліку робочого часу бригад» 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. **Коваленко Богдан Ігорович,**___ (підпис)