Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої
освіти бакалавра
зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка системи автоматизації Телеграм-бота для адміністрування інфраструктурних об’єктів

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань

Студент гр. ІПЗ-23ск _____ / М. О. Волков /

Керівник
кваліфікаційної роботи _____ / І. А. Котов /

В.о. завідувача кафедри _____ / А. М. Стрюк /

Кривий Ріг
2026

Криворізький національний університет
Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. зав. кафедри
_____ А. М. Стрюк
«___» _____ 20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу
студенту групи ІПЗ-23ск Волкова Максима Олексійовича

Тема: Розробка системи автоматизації Телеграм-бота для адміністрування інфраструктурних об’єктів затверджено наказом по КНУ № 283с від 28.05.26.

1. Термін подання студентом закінченої роботи: «8» червня 2026 р.
2. Вихідні дані по роботі: розроблюваний програмний комплекс має допомагати у виконанні обов’язків співробітників промислового підприємства при роботі в умовах з низькою надійністю каналів зв’язку.
3. Зміст пояснювальної записки: виконати аналіз предметної області, спроектувати систему для первинного документообігу, розробити програмну реалізацію програмного комплексу.
4. Перелік ілюстративного матеріалу: блок схеми алгоритмів, знімки інтерфейсів, UML діаграми, знімки прикладів.

Календарный план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	09.02.2026 – 18.02.2026
2	Підготовка матеріалів першого розділу роботи	19.02.2026 – 23.02.2026
3	Підготовка матеріалів другого розділу роботи	24.02.2026 – 03.03.2026
4	Розробка моделі та практичний аналіз умов на підприємстві	04.03.2026 – 13.03.2026
5	Підготовка матеріалів третього розділу роботи	14.03.2026 – 07.04.2026
6	Розробка мобільного програмного комплексу для первинного документообігу промислового підприємства в умовах низької надійності каналів передачі даних	08.04.2026 – 17.04.2026
7	Оформлення пояснювальної записки	18.04.2026 – 02.05.2026

Дата видачі завдання: «02» лютого 2026 р.

Студент _____ / М. О. Волков /

Керівник роботи _____ / І. А. Котов /

РЕФЕРАТ

АВТОМАТИЗАЦІЯ, АДМІНІСТРУВАННЯ, АНАЛІТИКА, БАЗА ДАНИХ, ІНФРАСТРУКТУРНІ ОБ'ЄКТИ, ПОШУК ІНФОРМАЦІЇ, ТЕЛЕГРАМ-БОТ.

Пояснювальна записка: 90 с., 9 табл., 18 рис., 1 дод., 28 джерел.

Метою кваліфікаційної роботи є розробка системи автоматизації у вигляді Телеграм-бота для адміністрування та системного обліку інфраструктурних об’єктів.

Головною задачею при розробці програмного комплексу є створення зручного інструменту, що поєднує функції бази даних та інтерактивної візуалізації для швидкого доступу до технічної інформації. Під час роботи було проведено аналіз предметної області, виявлено потреби у систематизації даних про будинки, ключі та доступи.

Було спроєктовано структуру бази даних та алгоритми пошуку, що дозволяють знаходити об'єкти за адресою, геопозицією або іншою інформацією. Реалізовано функціонал для повного адміністрування записів (додавання, редагування, видалення) та додавання медіафайлів.

Після аналізу вимог було впроваджено модуль аналітики, який дозволяє відстежувати активність користувачів та виявляти найбільш популярні об'єкти. Це допомагає ефективно керувати інформаційною базою та пріоритезувати обслуговування.

Як результат, ми маємо програмний комплекс на базі Телеграм-бота для ефективного адміністрування інфраструктурних об’єктів та оперативного доступу до даних.

ABSTRACT

TELEGRAM BOT, AUTOMATION, INFRASTRUCTURE OBJECTS, ADMINISTRATION, DATABASE, ANALYTICS, INFORMATION SEARCH.

Explanatory note: 90 p., 9 tabl., 18 fig., 1 app., 28 sources.

The purpose of the qualification work is the development of an automation system in the form of a Telegram bot for the administration and systematic accounting of infrastructure objects.

The main task in the development of the software complex is the creation of a convenient tool that combines database functions and interactive visualization for quick access to technical information. During the work, an analysis of the subject area was conducted, and the needs for systematizing data on houses, keys, and access were identified.

The database structure and search algorithms were designed, allowing objects to be found by address, geolocation, or other information. Functionality for full record administration (adding, editing, deleting) and adding media files was implemented.

After analyzing the requirements, an analytics module was implemented, which allows tracking user activity and identifying the most popular objects. This helps to effectively manage the information base and prioritize maintenance.

As a result, we have a software complex based on a Telegram bot for effective administration of infrastructure objects and operational access to data.

ЗМІСТ

ВСТУП

У наш час ефективне функціонування будь-якої сервісної або обслуговуючої компанії неможливе без швидкого доступу до технічної інформації. Підприємства, що займаються обслуговуванням інфраструктурних об’єктів (будинків, комунікацій, обладнання), щодня оперують величезними масивами даних: від адрес та геопозицій до специфічних відомостей про ключі доступу, коди домофонів та контактні дані відповідальних осіб.

Проте на багатьох підприємствах облік цієї інформації досі ведеться застарілими методами: у паперових журналах, розрізнених Excel-таблицях або ж передається в телефонному режимі. Це призводить до значних втрат часу, коли майстер або інженер, знаходячись на об’єкті, змушений витрачати дорогоцінні хвилини на пошук потрібного ключа чи уточнення деталей доступу у диспетчера. Географічна роздробленість об’єктів інфраструктури вимагає рішення, яке завжди буде «під рукою» у співробітника.

Враховуючи повсюдне використання смартфонів та мобільного інтернету, найбільш раціональним шляхом вирішення цієї проблеми є використання месенджерів як платформи для корпоративних інформаційних систем. Особливої популярності набув Telegram завдяки своїй доступності, швидкості та потужному API для розробки ботів.

Всі ці фактори призводять до потреби у створенні спеціалізованої системи автоматизації — Телеграм-бота, який виконуватиме роль єдиної мобільної бази даних для персоналу. Така система дозволяє відмовитися від громіздкого програмного забезпечення на користь легкого та зручного інтерфейсу, що не потребує додаткового навчання користувачів.

Головною задачею при розробці такої системи є створення інструменту, що поєднує в собі функції надійної бази даних, інтерактивного каталогу та засобу візуалізації. Бот повинен забезпечувати миттєвий пошук потрібного будинку за будь-яким відомим параметром (адресою, телефоном,

геопозицією) та надавати вичерпну інформацію, включаючи фотографії обладнання та дані про ключі.

Метою даної роботи є розробка та програмна реалізація системи автоматизації Телеграм-бота для системного обліку та адміністрування інфраструктурних об’єктів. Важливим аспектом є не лише відображення даних, але й можливість їх повного адміністрування (CRUD-операції) та збір аналітики для керівництва.

Як результат роботи ми отримуємо готовий програмний комплекс, який вирішує проблему децентралізації даних та забезпечує оперативний доступ до інформації про інфраструктуру в режимі 24/7.

Даний програмний комплекс допоможе підприємству оптимізувати роботу виїзного персоналу, підвищити швидкість реагування на запити та надати керівництву інструменти аналітики для виявлення найбільш навантажених об’єктів та ефективного управління ресурсами.

1 АНАЛІЗ МЕТОДІВ АВТОМАТИЗАЦІЇ АДМІНІСТРУВАННЯ ІНФРАСТРУКТУРНИХ ОБ'ЄКТІВ

1.1 Аналіз основних положень організації обліку та адміністрування об'єктів інфраструктури

Робота будь-якої сервісної організації або компанії, що займається обслуговуванням інфраструктури, потребує особливого відношення до систем, що забезпечують управлінську діяльність та оперативний доступ до технічної інформації. Розвиток інформаційної епохи передбачає перехід від статичних баз даних (таблиць, журналів) до інтерактивних систем, що забезпечують миттєву взаємодію з даними. Ці фактори призводять до необхідності впровадження автоматизованих систем на базі сучасних месенджерів та хмарних технологій. Наявність великої кількості розподілених об'єктів (будинків, під'їздів, обладнання) потребує потужних та мобільних інструментів для підтримки високої швидкості обслуговування та адміністрування [2, 3].

Ключовим завданням сучасних організацій є створення єдиного інформаційного простору. Найважливіше це питання для підприємств, співробітники яких постійно переміщуються містом та потребують актуальних даних «тут і зараз».

Основою управлінської діяльності в такій системі є інформаційний об'єкт (запис у базі даних). За аналогією з документом, інформаційний об'єкт — це структурований набір даних, що має унікальні ідентифікатори та властивості. У контексті даної роботи, під об'єктом розуміється цифрова сутність, що описує елемент інфраструктури.

Згідно зі статтею 1 Закону України «Про інформацію», інформація — це будь-які відомості та/або дані, які можуть бути збережені на матеріальних носіях або відображені в електронному вигляді. У розроблюваній системі

основним носієм інформації виступає хмарна база даних, а інтерфейсом доступу — Телеграм-бот.

Адміністрування інфраструктури — це набір бізнес-процесів, пов'язаних із додаванням, оновленням, пошуком та аналізом даних про об'єкти.

Всі дані в системі мають своє призначення, і для коректної організації їх розподіляють на інформаційні потоки. Основними потоками в системі Телеграм-бота є:

- вхідні потоки (Запити): команди користувачів на пошук інформації (за адресою, телефоном, геопозицією);
- вихідні потоки (Відповіді): структуровані картки об'єктів, фотографії, контактні дані, що надсилаються ботом;
- внутрішні потоки (Адміністрування): процеси оновлення бази даних, додавання нових ключів, редагування описів, збір логів та аналітики.

Більшу частину інформаційного навантаження складають так звані об'єкти обліку — записи, що містять головні відомості про інфраструктуру. До них відносять: будинки, під'їзди, ключі доступу, технічні фотографії, контакти відповідальних осіб.

Рисунок 1.1 – Концептуальна модель сучасної системи адміністрування інфраструктури та її інформаційних потоків

Для кожного об'єкта в системі визначено набір обов'язкових атрибутів (реквізитів), необхідних для його ідентифікації:

- унікальний ідентифікатор (ID);
- адреса (Вулиця, Номер будинку);
- геопозиція (координати для відображення на мапі);
- технічні параметри (тип обладнання, коди доступу);
- медіа-дані (фотографії фасадів, обладнання).

Відсутність хоча б одного з ключових атрибутів робить запис неможливим для пошуку, тому система повинна мати механізми валідації даних.

Перелік основних об'єктів системи та їх суть наведено в таблиці 1.1.

Таблиця 1.1 – Перелік основних об'єктів системи додатку

№	Назва об'єкта	Суть та призначення
1	Будинок	Основна одиниця обліку. Зберігає прив'язку до геопозиції, адреси та загального опису об'єкта.
2	Ключ / Код доступу	Інформація про способи фізичного доступу до об'єкта (коди домофонів, номери ключів).
3	Контактна особа	Дані про голів ОСББ, відповідальних мешканців або технічний персонал (телефон, ім'я).
4	Медіа-файл	Фотографії розташування обладнання, фасадів або схем, що допомагають візуально ідентифікувати об'єкт.
5	Лог подій (Аналітика)	Службовий запис, що фіксує, хто, коли та який об'єкт шукав. Використовується для побудови рейтингів активності.

Адміністрування характеризується таким показником як активність звернень, що вказує на кількість запитів до бота за певні проміжки часу. Завдяки модулю аналітики можна визначати "гарячі точки" (будинки, що обслуговуються найчастіше) та оптимізувати маршрути персоналу.

Життєвий цикл інформації в боті складається з таких етапів:

- первинне внесення даних адміністратором (створення картки будинку);
- збагачення даними (додавання фото, ключів);
- пошук та використання даних користувачами (читання);
- актуалізація (редагування застарілої інформації);
- видалення або архівування (при відключенні об'єкта).
-

Рисунок 1.2 – Життєвий цикл інформаційного об'єкта в системі адміністрування

На відміну від паперового документообігу, де оновлення інформації вимагає фізичної заміни носіїв, система на базі Телеграм-бота дозволяє вносити зміни миттєво. Всі користувачі отримують доступ до оновлених даних одразу після їх збереження в базі.

На даному етапі стає зрозумілим, що система управління повинна складатися з двох інтерфейсів: користувацького (для швидкого пошуку) та адміністративного (для керування контентом). Це означає, що кожен співробітник, залежно від своїх прав доступу, отримує відповідний набір інструментів.

Звідси можна зробити висновок, що якісний, централізований облік інфраструктури забезпечується використанням єдиної бази даних з доступом через API месенджера. Це призводить до значного підвищення ефективності

роботи персоналу, зменшення часу на пошук ключів та адрес, а також дозволяє керівництву отримувати прозору статистику роботи.

Працювати з даними через мобільний інтерфейс Телеграм-бота наразі виглядає найбільш ефективним способом організації роботи виїзного персоналу, що поєднує в собі мобільність, простоту використання та захищеність даних.

1.2 Аналіз каналів передачі інформації в розподілених системах адміністрування інфраструктури

З практичної точки зору, сучасні сервісні компанії та підприємства, що обслуговують інфраструктуру (інтернет-провайдери, служби обслуговування домофонів, ОСББ, комунальні служби), мають мережу об'єктів, які географічно розкидані по всьому місту або навіть регіону. Це призводить до складнощів у контролі доступу до цих об'єктів та актуалізації технічної інформації про них. Це спонукає до появи розподілених мобільних систем, які дозволяють керувати інфраструктурою віддалено [11].

Будь-яка система адміністрування використовує декілька каналів передачі даних. У контексті обслуговування інфраструктури можна виділити такі види зв’язку:

- паперові журнали (видача ключів під розпис);
- телефонний зв'язок (дзвінки диспетчеру);
- месенджери в режимі "чат" (Viber/WhatsApp групи);
- спеціалізоване програмне забезпечення (API, боти).

Канали передачі інформації в таких системах можна поділити на дві основні групи: неструктуровані (аналогові/ручні) та структуровані (цифрові/автоматизовані). До неструктурованих відносяться: паперові носії, усні вказівки, телефонні розмови, текстові повідомлення у загальних чатах (де інформацію важко знайти). Під структурованими цифровими каналами мають

на увазі взаємодію через API, веб-інтерфейси та чат-боти, де запит і відповідь мають чіткий формат.

Рисунок 1.3 – Класифікація каналів передачі інформації в системах адміністрування інфраструктури

Основними напрямками передачі інформації при обслуговуванні інфраструктури є:

- зв’язок "Диспетчер – Виїзний майстер";
- зв’язок "Адміністратор – База даних";
- зв’язок "Майстер – База даних" (отримання довідкової інформації).

Розглянемо типовий приклад робочого процесу на підприємстві, що обслуговує житлові будинки, для аналізу ефективності існуючих каналів зв'язку.

Сценарій: Майстру необхідно провести ремонтні роботи обладнання, яке знаходиться на горищі або в підвалі житлового будинку. Майстер прибуває за адресою. Під'їзд зачинено на домофон, а ключ від підвалу знаходиться у

відповідальної особи (наприклад, голови ОСББ) або захований у спеціальному місці (наприклад, у щитовій сусіднього під'їзду).

При використанні традиційних каналів: Майстер телефонує диспетчеру. Диспетчер шукає інформацію в Excel-таблиці або паперовому журналі. Це займає час. Часто виявляється, що номер телефону контактної особи змінився, або код домофону застарів. Майстер змушений чекати, передзвонювати, або їхати в офіс за фізичним ключем. Якщо диспетчер зайнятий іншим дзвінком — робота простоює. Після завершення робіт майстер повинен знову зателефонувати і відзвітувати, або записати зміни (наприклад, "змінив код замка") у свій блокнот, щоб потім, можливо, передати це в офіс. Часто ця інформація губиться, і база даних стає неактуальною.

При використанні Телеграм-бота: Майстер відправляє боту геолокацію або вводить номер будинку. Система миттєво (через API) звертається до бази даних і видає картку об'єкта: актуальний код домофону, фотографію місця, де лежить ключ, та контактний номер голови ОСББ. Якщо майстер змінює замок, він тут же через меню бота оновлює інформацію в базі.

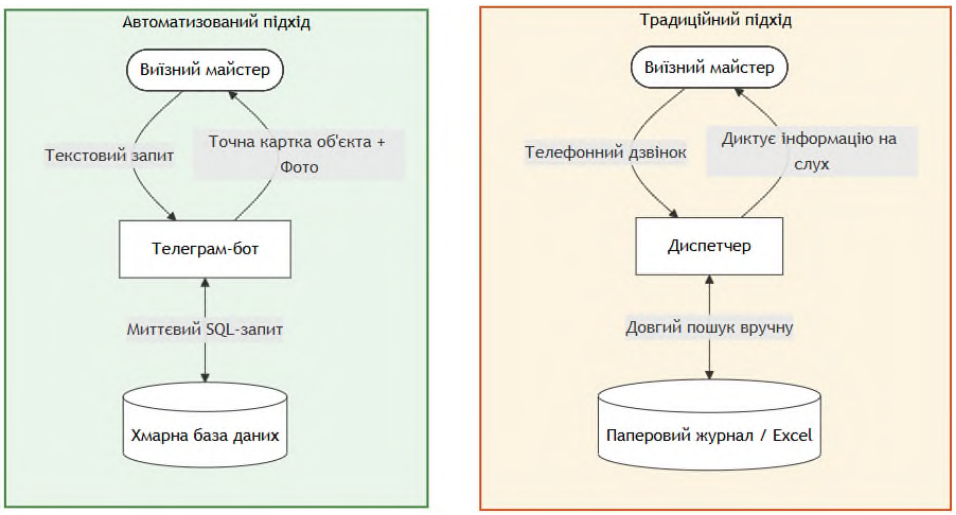

Рисунок 1.4 – Порівняльна схема каналів передачі інформації при традиційному та автоматизованому підходах

Як бачимо, в реальних умовах використання застарілих каналів передачі інформації (телефон, папір) призводить до наступних проблем:

- «Людський фактор»: помилки при диктуванні кодів або телефонів;
- втрата часу: очікування на лінії, пошук інформації в розрізнених файлах;
- деактуалізація даних: зміни на об'єктах не вносяться в центральну базу вчасно;
- відсутність історії: неможливо відстежити, хто і коли брав ключі або запитував доступ;
- залежність від офісу: неможливість отримати дані у неробочий час (вночі або у вихідні при аваріях).

Альтернативою на сьогоднішній час є використання Телеграм-ботів як клієнтської частини розподіленої системи. Вони використовують надійні цифрові канали (мобільний інтернет, HTTPS протокол), забезпечують авторизацію користувачів та доступність 24/7.

Порівняння традиційного (ручного/телефонного) методу адміністрування та автоматизованого (через бот) представлено в таблиці 1.2. Виходячи з цього, можна прийти до висновку, що автоматизація через месенджер є найефективнішим способом управління розподіленою інфраструктурою. Такий підхід дозволяє суттєво скоротити час на пошук необхідних технічних даних та мінімізує кількість критичних помилок, спричинених людським фактором. Крім того, цілодобовий доступ до централізованої бази об'єктів знижує навантаження на диспетчерський центр і підвищує загальну продуктивність роботи виїзного персоналу підприємства [10, 15].

Таблиця 1.2 – Порівняльна характеристика методів доступу до інформації

№	Критерії порівняння	Ручне/Телефонне адміністрування	Автоматизоване (Телеграм-бот)
1	Швидкість отримання даних	Від 2 до 15 хвилин (залежить від завантаженості диспетчера)	1-3 секунди (залежить від швидкості інтернету)
2	Доступність	Обмежена робочим часом диспетчерської служби	Цілодобово (24/7)
3	Актуальність даних	Низька (оновлення відбувається із затримкою)	Висока (зміни вступають в силу миттєво)
4	Точність інформації	Можливі помилки при передачі "на слух"	Виключено спотворення даних при передачі
5	Вимоги до обладнання	Телефонний зв'язок	Смартфон з встановленим месенджером
6	Безпека	Низька (важко перевірити, хто телефонує)	Середня/Висока (авторизація за номером телефону або ID акаунту)
7	Мультимедійні можливості	Відсутні (неможливо передати фото по телефону)	Можливість передачі фотографій, геолокацій, файлів
8	Аналітика	Відсутня або потребує ручного підрахунку	Автоматичний збір статистики запитів та активності

Використовуючи цифрові системи на базі ботів, ми вирішуємо більшість проблем логістики та інформування [25]. Проте виникають нові виклики, які необхідно врахувати при розробці:

- залежність від наявності мобільного інтернету (наприклад, у підвальних приміщеннях);
- залежність від працездатності серверів Telegram API;
- необхідність забезпечення безпеки даних при втраті смартфону співробітником.

З урахуванням даного порівняння, можна прийти до висновку, що розробка системи автоматизації на базі Телеграм-бота є оптимальним рішенням для підвищення ефективності обслуговування інфраструктурних об'єктів.

1.3 Аналіз існуючих рішень для автоматизації обліку та диспетчеризації

Для правильного вибору платформи розробки та розуміння необхідного функціоналу системи, необхідно проаналізувати існуючі на ринку рішення, які використовуються сервісними компаніями для обліку об'єктів та управління персоналом.

На сьогоднішній день для адміністрування інфраструктурних об'єктів (будинків, ключів, обладнання) найчастіше використовуються такі підходи:

- Табличні редактори (MS Excel, Google Sheets);
- універсальні CRM-системи (Bitrix24, AmoCRM, Zoho);
- спеціалізовані сервіси для ОСББ та комунальних служб («Мій Дім», «ОСББ 365»);
- робочі чати у месенджерах (Viber, WhatsApp, Telegram).

Табличні редактори (MS Excel, Google Sheets) залишаються найпоширенішим інструментом ведення первинного обліку на невеликих та

середніх підприємствах. Їхня головна перевага полягає в універсальності, низькому порозі входження та відсутності додаткових витрат на розгортання спеціалізованого програмного забезпечення. Базові офісні пакети дозволяють швидко створити реєстр інфраструктури, список адрес, ключів та контактних даних.

Проте, незважаючи на ефективність у стаціонарних офісних умовах, використання електронних таблиць виїзним персоналом має ряд критичних недоліків. Інтерфейс таких програм абсолютно не адаптований для смартфонів: необхідність постійного масштабування екрана та незручна навігація ускладнюють оперативний пошук і внесення даних «на ходу».

Крім того, табличний формат не забезпечує належної валідації введеної інформації (що призводить до помилок) та не дозволяє зручно інтегрувати технічні фотографії об'єктів. З точки зору безпеки, спільний мобільний доступ до єдиного файлу створює високий ризик випадкового пошкодження або незворотного видалення критично важливих даних співробітниками [4, 17].

Страна	Регион	Населенный Пункт
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Звенигородка
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Золотоноша
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев
Украина	Черкасская область	Канев

Рисунок 1.5 – Приклад відображення таблиці бази даних об'єктів

Універсальні CRM-системи – потужні програмні комплекси для управління бізнесом. Вони мають модулі для ведення бази клієнтів (у нашому випадку – мешканців), постановки задач майстрам та зберігання документів. Основна перевага – широкі можливості інтеграції та наявність мобільних додатків. Проте такі системи часто є «перевантаженими» для простого технічного персоналу. Майстру, якому потрібно просто знайти код від під'їзду, доводиться проходити складну авторизацію та навігацію по меню. Крім того, вартість ліцензії для кожного співробітника є досить високою, а стабільна робота мобільного додатку вимагає якісного інтернет-з'єднання та потужного смартфону.

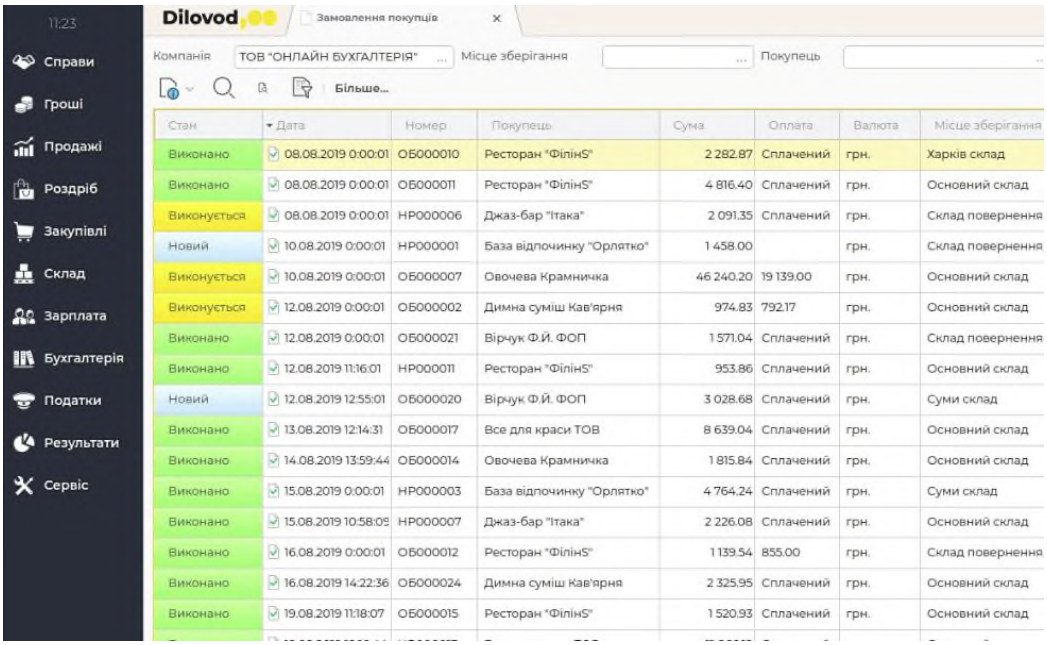

Рисунок 1.6 – Перевантажений користувацький інтерфейс додатку універсальної CRM-системи

Спеціалізовані сервіси (наприклад, платформи «Мій Дім», «ОСББ 365») – це комплексні програмні продукти, розроблені спеціально для управління житловим фондом. Вони чудово справляються з веденням обліку мешканців, генерацією квитанцій, контролем нарахувань та оплат. Проте їхній функціонал здебільшого орієнтований на бухгалтерський та абонентський облік, залишаючи оперативну технічну інвентаризацію на другому плані.

Для виїзного технічного персоналу використання таких систем часто є незручним. Їхні інтерфейси перевантажені зайвими модулями (наприклад, фінансовими звітами, підсистемами онлайн-голосувань чи реєстром боржників), що значно уповільнює навігацію та ускладнює оперативний пошук потрібної адреси з мобільного пристрою. Вузькопрофільні технічні функції, такі як облік конкретних номерів ключів, точні описи схем розташування обладнання в підвалах чи на горищах, а також миттєва фотофіксація технічного стану, реалізовані в них досить поверхнево або відсутні зовсім.

Крім того, більшість таких платформ є пропрієтарними (закритими) комерційними рішеннями, які розповсюджуються за моделлю SaaS (Software as a Service). Це створює дві суттєві проблеми: по-перше, їхню жорстку архітектуру вкрай важко інтегрувати з іншими системами або адаптувати під специфічні нетипові бізнес-процеси конкретної сервісної компанії; по-друге, необхідність сплати регулярних ліцензійних платежів (за кожного користувача або за кількість доданих будинків) робить використання таких масивних комплексів економічно недоцільним, якщо підприємству потрібен лише швидкий довідник для технічного персоналу.

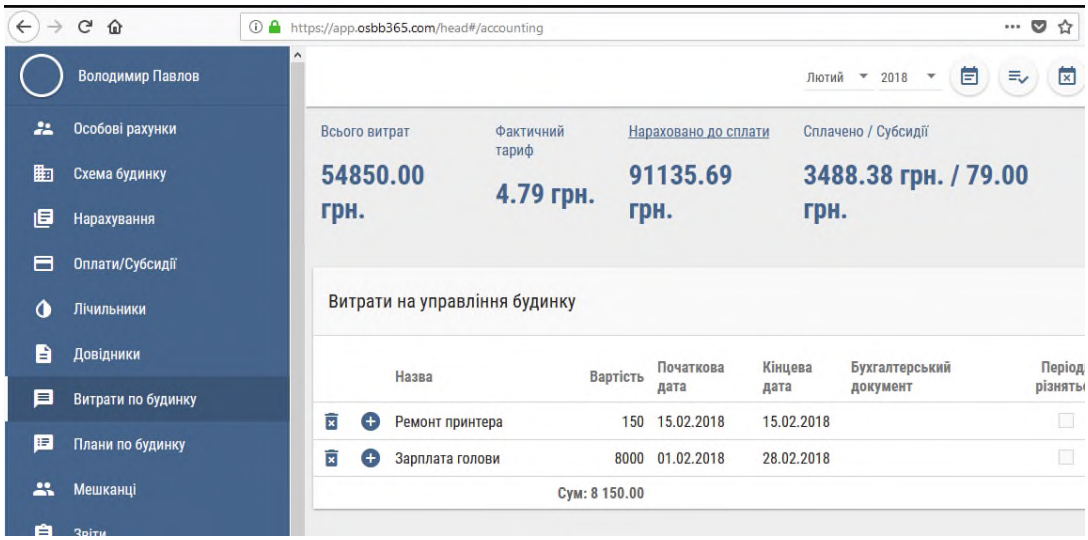

Рисунок 1.7 – Інтерфейс спеціалізованої програми для ОСББ з акцентом на фінансовий облік

Робочі чати у месенджерах (Viber, Telegram) – це найбільш «народний» метод комунікації. Створюються групи («Заявки», «Ключі»), куди скидається вся інформація. Інтерфейс месенджера є звичним для всіх. Проте такий підхід має суттєвий недолік – відсутність структури. Повідомлення про важливий код доступу швидко губиться в потоці листування. Знайти фотографію обладнання, зроблену півроку тому, практично неможливо. Відсутня база даних як така, інформація зберігається хаотично [14, 28].

Згідно з проведеним аналізом, можна дійти висновку, що існуючі рішення мають суттєві обмеження для вирішення поставленої задачі:

- Excel/Google Sheets – незручні для мобільного використання та не дозволяють працювати з медіа-файлами (фото);
- CRM-системи – занадто дорогі та складні для лінійного персоналу, вимагають навчання та потужних гаджетів;
- спеціалізоване ПЗ – фокусується на фінансах, а не на технічних деталях доступу до об'єктів;
- чати – створюють інформаційний хаос і не дають можливості швидкого пошуку.

На підприємстві, для якого виконується розробка, раніше використовувався комбінований метод (Excel для диспетчера + Viber чати для майстрів). Це призводило до того, що інформація про зміну кодів домофонів або нові ключі часто не вносилася в таблицю вчасно, а фотографії обладнання губилися в історії повідомлень. Через це виникали додаткові затрати часу на повторні виїзди та телефонні дзвінки.

Враховуючи відсутність на ринку ідеального готового рішення, яке б поєднувало простоту месенджера зі структурністю бази даних, керівництвом було прийнято рішення про розробку власної системи автоматизації на базі Telegram-бота.

Такий підхід дозволяє:

- використати звичний інтерфейс (не потрібно навчати персонал);
- забезпечити швидку роботу навіть при слабкому інтернеті (текстовий трафік мінімальний);
- створити структуровану базу даних, де кожен будинок має свою «картку» з фото та ключами;
- уникнути витрат на ліцензії дорогого ПЗ.

Таким чином, розробка власного Телеграм-бота є економічно та технічно обґрунтованим кроком для оптимізації процесів адміністрування інфраструктурних об'єктів.

1.4. Формування вимог до системи автоматизації адміністрування інфраструктури

Згідно потреб сервісної компанії, що обслуговує значну кількість розподілених об'єктів (житлових будинків), виникають специфічні вимоги до програмного забезпечення. Основним критерієм для прийняття системи в роботу є її мобільність, простота використання для технічного персоналу та швидкість доступу до даних.

По-перше, в умовах роботи виїзного персоналу (майстрів, інженерів) з’являється потреба в миттєвому отриманні інформації безпосередньо на об'єкті. Особливо це важливо у випадках аварійних виїздів, коли час на пошук ключів від підвалу або коду від домофону критично впливає на швидкість локалізації аварії. Майстер не повинен витрачати час на дзвінки диспетчеру та очікування на лінії.

По-друге, компанія повинна забезпечити конфіденційність технічних даних. Коди доступу до під'їздів, місця знаходження ключів та особисті телефони голів ОСББ не повинні бути доступні стороннім особам. При використанні паперових записників або загальних чатів у месенджерах, звільнений співробітник залишається з повною базою даних на своєму

телефоні. Необхідна система, яка дозволяє централізовано керувати правами доступу та миттєво блокувати доступ звільненим працівникам.

По-третє, коли виникає потреба оперативного оновлення інформації (наприклад, зміна коду на дверях), виникає проблема синхронізації даних. Якщо майстер змінив замок і записав новий код лише у свій блокнот, наступна зміна, яка приїде на цей об'єкт (наприклад, вночі), не зможе потрапити всередину. Виникає ситуація, коли диспетчер має застарілі дані, а актуальна інформація розпорошена по особистих записах співробітників. Необхідна єдина хмарна база даних, яка оновлюється в реальному часі.

По-четверте, із зростанням кількості об'єктів збільшується навантаження на диспетчерську службу. Якщо кожен майстер дзвонить диспетчеру, щоб дізнатися код, лінія постійно зайнята, а диспетчер виконує роль "голосового інтерфейсу" до Excel-таблиці. Для забезпечення ефективності роботи потрібно автоматизувати процес видачі довідкової інформації, щоб диспетчер займався лише нестандартними ситуаціями, а рутинні запити обробляв бот.

По-п’яте, специфіка роботи в підвальних приміщеннях, ліфтових шахтах та на технічних поверхах часто передбачає погану якість мобільного зв'язку. Програмний комплекс повинен бути "легким" та передавати мінімум трафіку, щоб працювати навіть при слабкому сигналі EDGE/2G, де завантаження важких веб-сторінок CRM-систем неможливе.

Попередньо керівництвом було визначено наступні вимоги до розроблюваного Телеграм-бота:

- можливість швидкого пошуку об'єкта за адресою або геолокацією;
- можливість роботи при слабкому інтернет-з'єднанні;
- зручний інтерфейс, що не потребує навчання персоналу;
- можливість прикріплення фотоматеріалів (схеми, фото обладнання);
- централізоване адміністрування прав доступу.

Детальний аналіз умов праці показав, що співробітники користуються різними моделями смартфонів (Android, iOS), часто бюджетного сегменту. Тому розробка окремого нативного мобільного додатку є недоцільною через високу вартість розробки та підтримки під різні ОС. Використання кросплатформного рішення на базі Telegram є оптимальним, оскільки месенджер вже встановлений у всіх співробітників і працює стабільно на будь-якому "залізі".

Отже, можна сформулювати основні вимоги до програмного забезпечення, які для зручності представлені в таблиці 1.3.

Таблиця 1.3 – Основні вимоги до системи автоматизації

№	Вимога	Пояснення
1	Кросплатформність	Система повинна працювати на будь-якому смартфоні (Android/iOS) та комп'ютері без встановлення додаткового ПЗ.
2	Мінімізація трафіку	Інтерфейс повинен завантажуватися миттєво навіть при поганому зв'язку (текстовий режим, оптимізовані фото).
3	Централізація даних	Всі зміни (нові коди, ключі) повинні миттєво потрапляти в єдину базу даних і ставати доступними всім користувачам.
4	Простота адміністрування	Інтерфейс додавання та редагування об'єктів повинен бути інтуїтивно зрозумілим для адміністратора без технічної освіти.
5	Медіа-можливості	Можливість зберігати та переглядати фотографії місць встановлення обладнання та схованок з ключами.

Продовження таблиці 1.3.

6	Економічна ефективність	Відсутність плати за ліцензії для кожного користувача та мінімальні витрати на сервер.
7	Масштабованість	Можливість легкого додавання нових міст, районів та типів об'єктів без переписання коду.

Більшість з цих вимог частково перекривають CRM-системи, але вони провалюються по пунктах "Економічна ефективність" та "Мінімізація трафіку". Таблиці Excel не задовольняють вимоги "Медіа-можливості" та "Централізація даних" (у мобільному сценарії).

Таким чином виявляється, що необхідно розробити власне рішення, яке буде одночасно задовольняти специфічні потреби:

- робота "в полях" з мобільного;
- безкоштовна платформа для клієнта;
- підтримка фото та геолокації.

Спираючись на дані обставини, з’являється задача на розробку спеціалізованого Телеграм-бота, який стане єдиним вікном доступу до інфраструктурної бази даних компанії. Це дозволить поєднати надійність бази даних SQL із зручністю месенджера Telegram.

1.5. Задача розробки системи автоматизації Телеграм-бота для адміністрування інфраструктурних об’єктів

Головною метою даної роботи є підвищення ефективності роботи виїзного персоналу сервісної компанії шляхом розробки мобільної та відмовостійкої системи доступу до технічної інформації.

Виходячи з поставленої мети, основними задачами розробки, які необхідно вирішити в ході виконання кваліфікаційної роботи, є:

- Провести розробку моделі і засобів передачі даних системи в умовах низької надійності каналів зв'язку.
- Спроєктувати структуру розподіленої реляційної бази даних для зберігання інформації про інфраструктурні об'єкти та користувачів.
- Розробити структуру та схему взаємодії компонентів (модулів) автоматизованої системи.
- Побудувати функціональну схему станів програмного комплексу Телеграм-бота.
- Розробити алгоритми функціонування системи, зокрема алгоритми ідентифікації користувачів та інтелектуального пошуку інформації.
- Здійснити вибір мови програмування, фреймворку та технологічного стеку для реалізації проєкту.
- Виконати програмну реалізацію розроблених моделей та бази даних.
- Провести тестування програмного комплексу та оцінити його працездатність у реальних умовах експлуатації.

Для забезпечення виконання цих задач необхідно дотримуватися чітких вимог до системи. Базуючись на аналізі предметної області, було сформовано перелік основних функцій, які повинен виконувати програмний комплекс:

- авторизація користувачів: перевірка прав доступу за унікальним ідентифікатором Telegram ID;
- інтелектуальний пошук: реалізація алгоритмів пошуку об'єктів за текстовим запитом (адреса, назва вулиці) та за геопозицією;
- управління контентом (CRUD): створення, редагування та перегляд карток будинків, під'їздів та технічних приміщень безпосередньо через інтерфейс бота;
- робота з медіа-файлами: можливість завантаження та відображення фотографій;
- адміністрування доступів: можливість додавати та блокувати користувачів через панель адміністратора;

– аналітика та логування: автоматична фіксація всіх дій користувачів (хто, коли і який об'єкт переглядав) для формування статистики та контролю;

– забезпечення відмовостійкості: коректна обробка помилок при відсутності даних або збоях мережі.

Структура інформаційних об'єктів розроблена відповідно до специфіки обслуговування житлового фонду. Основні сутності та їх логічні атрибути представлені в таблиці 1.4.

Таблиця 1.4 – Структура основних сутностей системи

№	Структурна одиниця	Поле (Атрибут)	Тип поля
1	Користувач (User)	Telegram ID	Ціле число (BigInt)
		Ім'я (Username)	Текст
2	Вулиця (Street)	Номер телефону	Текст
3	Будинок (House)	Роль (Адмін/Користувач)	Логічне / Enum
4	Доступ / Ключ (Access)	Статус (Активний/Блок)	Логічне
		Назва	Текст
5	Контактна особа	Район міста	Текст
		Номер будинку	Текст
		ID Вулиці	Зовнішній ключ
		Гео-координати (Lat/Long)	Дійсні числа (Float)
		Опис / Примітка	Текст
		ID Будинку	Зовнішній ключ

Продовження таблиці 1.4.

6	Лог активності (Log)	Номер під'їзду	Ціле число
		Код домофону	Текст
		Опис розташування ключа	Текст
		Фото-підказка (File ID)	Текст (ідентифікатор файлу)
		ID Будинку	Зовнішній ключ
		Посада (Голова ОСББ тощо)	Текст

Клієнтська частина системи (Front-end) реалізується на базі месенджера Telegram, що автоматично забезпечує кросплатформеність. Бот повинен коректно працювати на смартфонах з ОС Android та iOS, а також на десктопних версіях (Windows/macOS/Linux). Це знімає необхідність розробки окремих мобільних додатків та дозволяє охопити 100% персоналу.

Конкретних вимог до візуального стилю керівництвом не висувалося, оскільки інтерфейс обмежений можливостями Telegram API. Проте основними вимогами до UX (користувацького досвіду) є:

- мінімізація введення тексту: максимальне використання кнопок (Inline Keyboards) та навігаційних меню для вибору дій;
- швидкість реакції: відповідь бота на запит не повинна перевищувати 1-2 секунди;
- інтуїтивність: нові співробітники повинні мати змогу користуватися ботом без попереднього інструктажу.

Визначених задач та сформованих вимог достатньо для початку проєктування архітектури та програмної реалізації системи автоматизації, що вирішуватиме проблему оперативного доступу до інформації про інфраструктурні об'єкти.

2 РОЗРОБКА СТРУКТУР І МОДЕЛЕЙ ФУНКЦІОНУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ ТЕЛЕГРАМ-БОТА

2.1 Розробка моделі і засобів передачі даних системи адміністрування інфраструктурних об'єктів

Як було визначено у першому розділі, ефективне функціонування будь-якої сервісної компанії неможливе без швидкого доступу до технічної інформації. При цьому специфіка роботи виїзного персоналу часто передбачає знаходження у підвальних приміщеннях або на технічних поверхах, де спостерігається погана якість мобільного зв'язку. Тому архітектура розроблюваної системи будується навколо інфраструктури месенджера Telegram, який оптимізовано для роботи при слабкому інтернет-з'єднанні.

Модель передачі даних у розроблюваному програмному комплексі складається з трьох основних логічних вузлів:

- клієнтська частина (Front-end): мобільний додаток або десктопна версія месенджера Telegram на смартфоні чи комп'ютері співробітника;
- транспортний рівень: хмарні сервери Telegram, які шифрують та маршрутизують повідомлення за допомогою протоколу MTProto;
- серверна частина (Back-end): сервер підприємства або локальна машина, де розгорнуто логіку бота (модулі обробки команд) та базу даних інфраструктурних об'єктів.

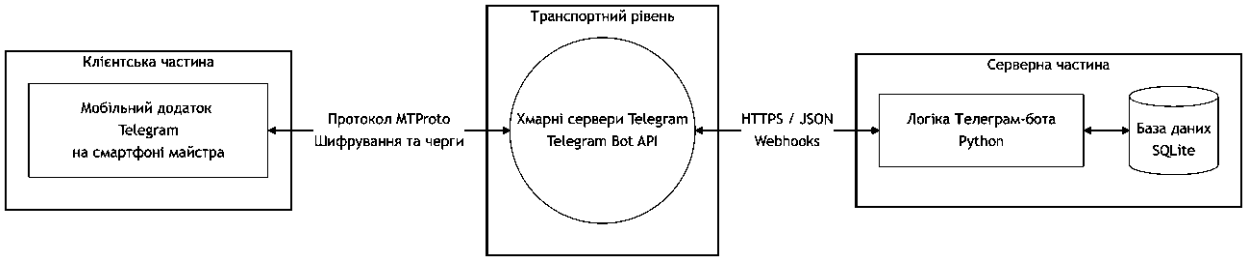

Рисунок 2.1 – Структурна модель передачі даних та взаємодії компонентів у системі Телеграм-бота

Головною перевагою такої моделі для умов низької надійності каналів зв'язку є механізм черг повідомлень на стороні клієнта. Якщо майстер вносить зміни до бази даних (наприклад, додає новий об'єкт або редагує існуючий), перебуваючи у зоні відсутності мережі, месенджер зберігає запит локально. Відправка пакету даних на сервер відбувається автоматично у фоновому режимі одразу після відновлення мінімального зв'язку. Це знімає необхідність розробки власних складних алгоритмів повторної відправки пакетів, які б вимагалися при створенні окремого нативного додатку.

Взаємодія між серверами Telegram та розробленим бекендом відбувається через Telegram Bot API з використанням протоколу HTTPS. Обмін даними здійснюється у форматі JSON (JavaScript Object Notation), що робить пакети даних легкими та зручними для парсингу.

З точки зору безпеки та передачі даних, система використовує модель авторизації на основі унікального ідентифікатора користувача Telegram ID. Тобто замість традиційної передачі логіна та пароля по мережі, система звіряє ID вхідного повідомлення зі своїм модулем авторизації, що значно прискорює процес обробки запитів та підвищує надійність.

Для реалізації програмних модулів обробки даних на стороні сервера використовується мова програмування Python. Передача даних від Telegram API до нашого серверного коду може бути реалізована за двома основними моделями:

- Long Polling (Довге опитування): програма періодично звертається до серверів Telegram з питанням про наявність нових подій;
- Webhooks: сервери Telegram самостійно ініціюють POST-запит на сервер підприємства в момент настання події.

Обидва методи забезпечують надійну доставку інформації. Таким чином, розроблена модель гарантує захищену та стійку до обривів зв'язку передачу технічної інформації без необхідності створення складних систем синхронізації «з нуля» [5, 16].

2.2 Розробка структури розподіленої бази даних програмного комплексу

Для забезпечення надійності, оперативності обліку інфраструктурних об'єктів та швидкого пошуку інформації розроблюваний програмний комплекс потребує використання реляційної бази даних.

Враховуючи архітектуру системи (Телеграм-бот), відсутність потреби у надвисокій кількості одночасних транзакцій на запис та вимогу до простоти розгортання, в якості системи управління базами даних (СУБД) було обрано SQLite. Вона інтегрована безпосередньо у програмний код на мові Python, працює локально та зберігає всі дані у єдиному файлі, що значно спрощує процеси резервного копіювання та перенесення системи на інший сервер.

Внутрішня структура бази даних розроблена з урахуванням принципів нормалізації для уникнення дублювання інформації та забезпечення її цілісності. База даних складається з п'яти основних таблиць, які відповідають за збереження об'єктів інфраструктури, медіа-файлів, прав доступу та логування дій користувачів. Логічну структуру бази даних та зв'язки між її сутностями (ER-діаграму) наведено на рисунку 2.2. [13, 18].

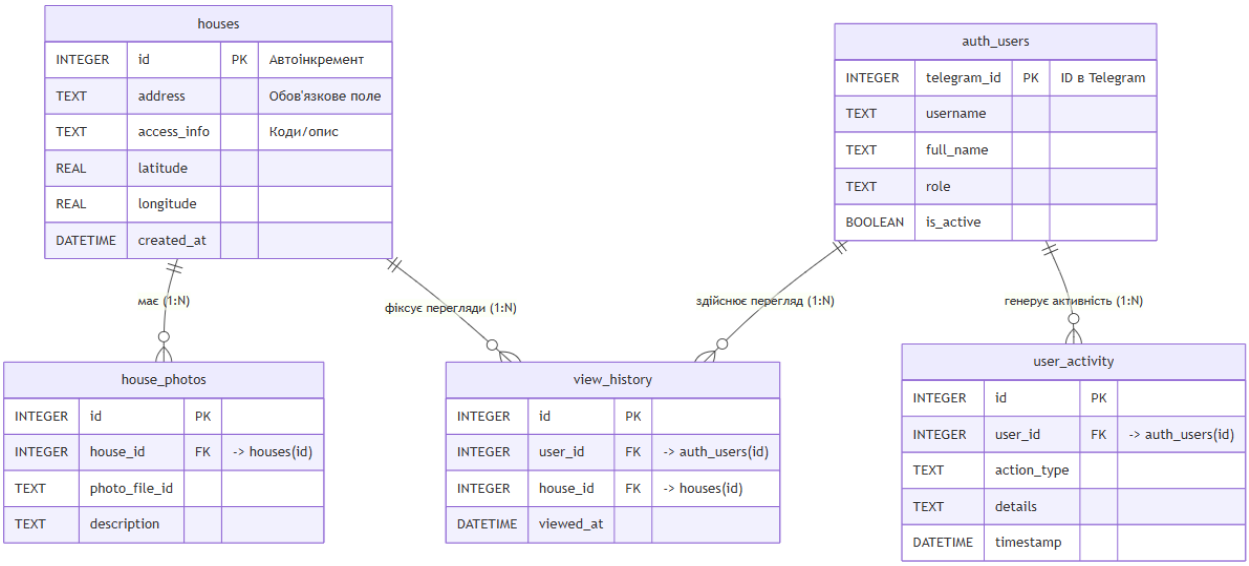

Рисунок 2.2 – ER-діаграма бази даних системи адміністрування інфраструктурних об'єктів

Основним об'єктом обліку в системі є інформація про будинок або інфраструктурний вузол. Для її збереження розроблено таблицю houses. Ця таблиця містить ключові ідентифікатори та текстову інформацію про доступи (коди домофонів, опис місцезнаходження ключів). Структура таблиці представлена в таблиці 2.1.

Таблиця 2.1 – Структура таблиці houses (Будинки)

№	Поле	Тип даних	Опис та обмеження
1	id	INTEGER	Первинний ключ (Primary Key), автоінкремент
2	address	TEXT	Адреса об'єкта (обов'язкове поле)
3	access_info	TEXT	Технічна інформація, коди, опис доступу
4	latitude	REAL	Географічна широта (для відображення на мапі)
5	longitude	REAL	Географічна довгота
6	created_at	DATETIME	Дата та час створення запису

Оскільки Telegram API працює з медіа-файлами за допомогою спеціальних внутрішніх ідентифікаторів (file_id), зберігати самі зображення

безпосередньо в базі даних недоцільно. Для реалізації можливості прикріплення фотографій до карток будинків створено підпорядковану таблицю house_photos, яка зв'язана з головною таблицею відношенням «один-до-багатьох» (один будинок може мати декілька фотографій). Її структура наведена в таблиці 2.2.

Таблиця 2.2 – Структура таблиці house_photos (Фотографії)

№	Поле	Тип даних	Опис та обмеження
1	id	INTEGER	Первинний ключ
2	house_id	INTEGER	Зовнішній ключ (Foreign Key), посилання на houses(id)
3	photo_file_id	TEXT	Унікальний ідентифікатор файлу на серверах Telegram
4	description	TEXT	Короткий опис фотографії (наприклад, "Щитова")

Для забезпечення безпеки та розмежування прав доступу (адміністратор/майстер) використовується таблиця auth_users. Замість класичних логінів та паролів ідентифікація відбувається за унікальним telegram_id. Якщо користувача немає в цій таблиці, бот ігнорує його команди.

Таблиця 2.3 – Структура таблиці auth_users (Користувачі)

№	Поле	Тип даних	Опис та обмеження	Поле
1	telegram_id	INTEGER	Первинний ключ, унікальний ID користувача в Telegram	telegram_id
2	username	TEXT	Нікнейм користувача (опціонально)	username
3	full_name	TEXT	Ім'я та прізвище співробітника	full_name
4	role	TEXT	Роль у системі (наприклад, 'admin', 'user')	role
5	is_active	BOOLEAN	Статус доступу (1 - активний, 0 - заблокований)	is_active

Важливою вимогою до системи була наявність модулю аналітики та контролю. Для цього спроєктовано дві таблиці для збору статистики: user_activity та view_history. Таблиця view_history фіксує безпосередні запити до карток об'єктів, що дозволяє виявляти найбільш популярні або проблемні адреси, тоді як user_activity відповідає за глобальне логування дій персоналу (авторизація, додавання нових будинків, помилки).

Таблиця 2.4 – Структура таблиці view_history (Історія переглядів)

№	Поле	Тип даних	Опис та обмеження
1	id	INTEGER	Первинний ключ
2	user_id	INTEGER	Зовнішній ключ, посилання на auth_users(telegram_id)
3	house_id	INTEGER	Зовнішній ключ, посилання на houses(id)
4	viewed_at	DATETIME	Точний час перегляду інформації

Таблиця user_activity (таблиця 2.5) відповідає за глобальне логування ключових дій персоналу (авторизація, додавання або видалення будинків, помилки при пошуку). Це дозволяє адміністраторам відстежувати зміни в базі даних та швидко реагувати на несанкціоновані дії.

Таблиця 2.5 – Структура таблиці user_activity (Активність користувачів)

№	Поле	Тип даних	Опис та обмеження
1	id	INTEGER	Первинний ключ
2	user_id	INTEGER	Зовнішній ключ, посилання на auth_users(telegram_id)
3	action_type	TEXT	Тип виконаної дії (наприклад, додавання об'єкта)
4	details	TEXT	Деталі дії або системне повідомлення
5	timestamp	DATETIME	Час виконання дії

Усі звернення до бази даних інкапсульовані всередині спеціальних класів-репозиторіїв на мові Python. Оскільки Telegram-бот працює в асинхронному режимі, для уникнення блокувань бази даних (database locked) використовуються відповідні асинхронні драйвери для роботи з SQLite (наприклад, aiosqlite).

Таким чином, розроблена структура бази даних повністю покриває потреби зберігання інформації про інфраструктуру, дозволяє масштабувати дані та забезпечує надійний рівень контролю за діями виїзного персоналу.

2.3 Розробка структури і схеми взаємодії компонентів автоматизованої системи

Визначимо основні високорівневі компоненти розроблюваного програмного комплексу:

- клієнтський інтерфейс (мобільний або десктопний додаток Telegram);
- проміжний транспортний рівень (Telegram Bot API);
- серверна частина (додаток, написаний мовою Python);
- локальна база даних (СУБД SQLite).
- Для забезпечення виконання всіх необхідних операцій та реалізації бізнес-логіки системи, серверну частину бота розділено на наступний перелік функціональних модулів:
- модуль авторизації та контролю доступу;
- модуль обробки повідомлень та станів (Handlers);
- модуль взаємодії з базою даних;
- модуль логування та аналітики;
- модуль формування користувацького інтерфейсу.

Модуль формування користувацького інтерфейсу відповідає за створення візуальних елементів управління на стороні клієнта. Оскільки

Telegram не має класичного графічного інтерфейсу (GUI), цей модуль генерує інтерактивні клавіатури двох типів: Reply (кнопки замість клавіатури) та Inline (кнопки під повідомленнями). Це мінімізує необхідність ручного введення тексту майстром та зводить управління базою до кількох натискань [1].

Модуль авторизації та контролю доступу є критично важливим компонентом безпеки. Він відповідає за перевірку кожного вхідного запиту. Модуль звіряє унікальний telegram_id користувача із білим списком у базі даних. Крім того, модуль розмежовує права доступу: звичайні користувачі (майстри) можуть лише переглядати та редагувати технічні дані, тоді як адміністратори мають доступ до функцій видалення об'єктів, вивантаження бази даних та перегляду статистики [19].

Модуль взаємодії з базою даних виконує роль прошарку (Data Access Layer) між логікою програми та фізичним файлом SQLite. Він забезпечує безпосереднє з'єднання, формування SQL-запитів, безпечне читання, запис та оновлення інформації про будинки, ключі та медіа-файли [12, 22].

Модуль логування та аналітики працює у фоновому режимі та автоматично фіксує всі ключові події. Кожен запит на пошук адреси, успішна авторизація чи зміна коду домофону записується цим модулем до відповідних таблиць (view_history, user_activity). Це дозволяє керівництву отримувати вивантаження у форматі CSV для аналізу активності.

Модуль обробки повідомлень та станів (Handlers) — це головний контролер системи (Controller у парадигмі MVC). Він приймає оброблені дані від Telegram API та вирішує, яку дію виконати. Для реалізації складних багато крокових операцій (наприклад, додавання нового будинку: крок 1 – ввести вулицю, крок 2 – номер будинку, крок 3 – опис) цей модуль використовує кінцевий автомат станів (Finite State Machine, FSM).

Повна схема взаємодії компонентів та модулів програмного комплексу представлена на рис. 2.3.

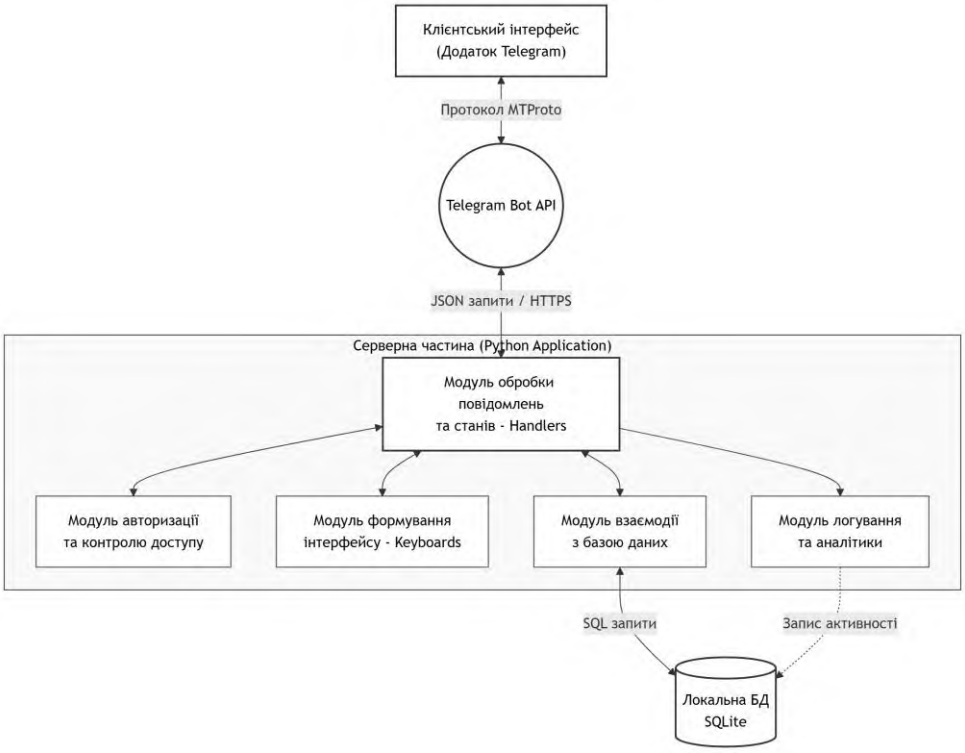

Рисунок 2.3 – Структурна схема взаємодії модулів серверної частини Телеграм-бота

Всі модулі серверної частини тісно пов’язані один з одним, проте мають чітку зону відповідальності, що створює прозору архітектуру та дозволяє легко масштабувати систему у майбутньому (наприклад, додавати нові типи інфраструктурних об'єктів).

Життєвий цикл обробки запиту відбувається наступним чином. Користувач через клієнтський додаток Telegram натискає кнопку або відправляє текстове повідомлення. Ця дія через Telegram Bot API надходить до серверної частини у вигляді JSON-об'єкта.

Першим спрацьовує модуль обробки повідомлень, який негайно передає дані користувача до модуля авторизації. Якщо перевірка telegram_id проходить успішно, система перевіряє поточний стан користувача (FSM). Залежно від команди, ініціюється звернення до модуля взаємодії з БД для пошуку потрібної адреси або збереження нових даних. Отриманий результат передається до модуля формування інтерфейсу, який «обгортає» текст у зручну картку з кнопками. Одночасно з цим модуль аналітики записує факт

виконання операції. Зрештою, сформована відповідь відправляється назад користувачеві через API месенджера.

Такий підхід дозволяє обробляти запити в асинхронному режимі та забезпечує високу швидкість реакції системи (до 1-2 секунд) навіть за умов низької надійності каналів зв'язку на стороні клієнта.

2.4 Функціональна схема автоматизованої системи адміністрування інфраструктурних об'єктів

Програмний комплекс Телеграм-бота під час своєї роботи функціонально може переходити між низкою ключових станів залежно від етапу виконання програми та дій користувача. Оскільки система реалізована на базі асинхронного фреймворку aiogram, управління станами відбувається за допомогою вбудованого механізму кінцевого автомата (Finite State Machine – FSM).

Функціонально роботу системи можна поділити на такі ключові стадії (стани):

- не авторизовано (початок роботи);
- перевірка авторизації (введення пароля / реєстрація імені);
- головне меню (очікування вибору дії);
- додавання об'єкта (послідовне введення вулиці, будинку, опису, ключа);
- пошук об'єкта (очікування пошукового запиту);
- редагування/видалення об'єкта;
- робота з медіа-файлами (додавання або перегляд фото);
- збереження даних та логування активності.

Авторизація реалізується через перевірку наявності унікального telegram_id користувача в базі даних. При першому запуску (команда /start) система перевіряє статус користувача. Якщо користувач не авторизований, бот

запитує пароль та унікальне ім'я (username) для подальшої ідентифікації в системі логування. Стан Авторизовано відкриває доступ до головного меню (навігаційної клавіатури).

З головного меню користувач може ініціювати різні бізнес-процеси за допомогою кнопок (наприклад, «🏠Додати дім🏠», «🔍Знайти дім🔍», «📖Список домов📖» тощо). При виборі певної дії система переводить користувача у відповідний стан FSM.

Розглянемо процес додавання нового інфраструктурного об'єкта. При натисканні кнопки додавання, система переходить у стан waiting_for_street (очікування назви вулиці). Після введення даних бот перемикається у стан waiting_for_house_number (очікування номеру будинку), потім – waiting_for_house_info (опис) та waiting_for_key_number (номер ключа). Якщо всі дані введено коректно, відбувається транзакція збереження до БД (INSERT INTO houses), логування дії користувача до таблиці активності, створення резервної копії бази даних (backup_database), після чого стан FSM очищується (state.clear()), і користувач повертається до головного меню [8, 9].

Будь-який процес (додавання, редагування, пошук) може бути перерваний користувачем на будь-якому кроці за допомогою кнопки «🔄Перезапустить🔄». При її натисканні контролер негайно скидає поточний стан автомата та повертає систему до базового стану очікування команд головного меню. Це забезпечує високий рівень відмовостійкості та захищає систему від «зависань» у випадку помилкового введення даних користувачем.

Детально функціональну схему станів програмного комплексу представлено на рис. 2.4.

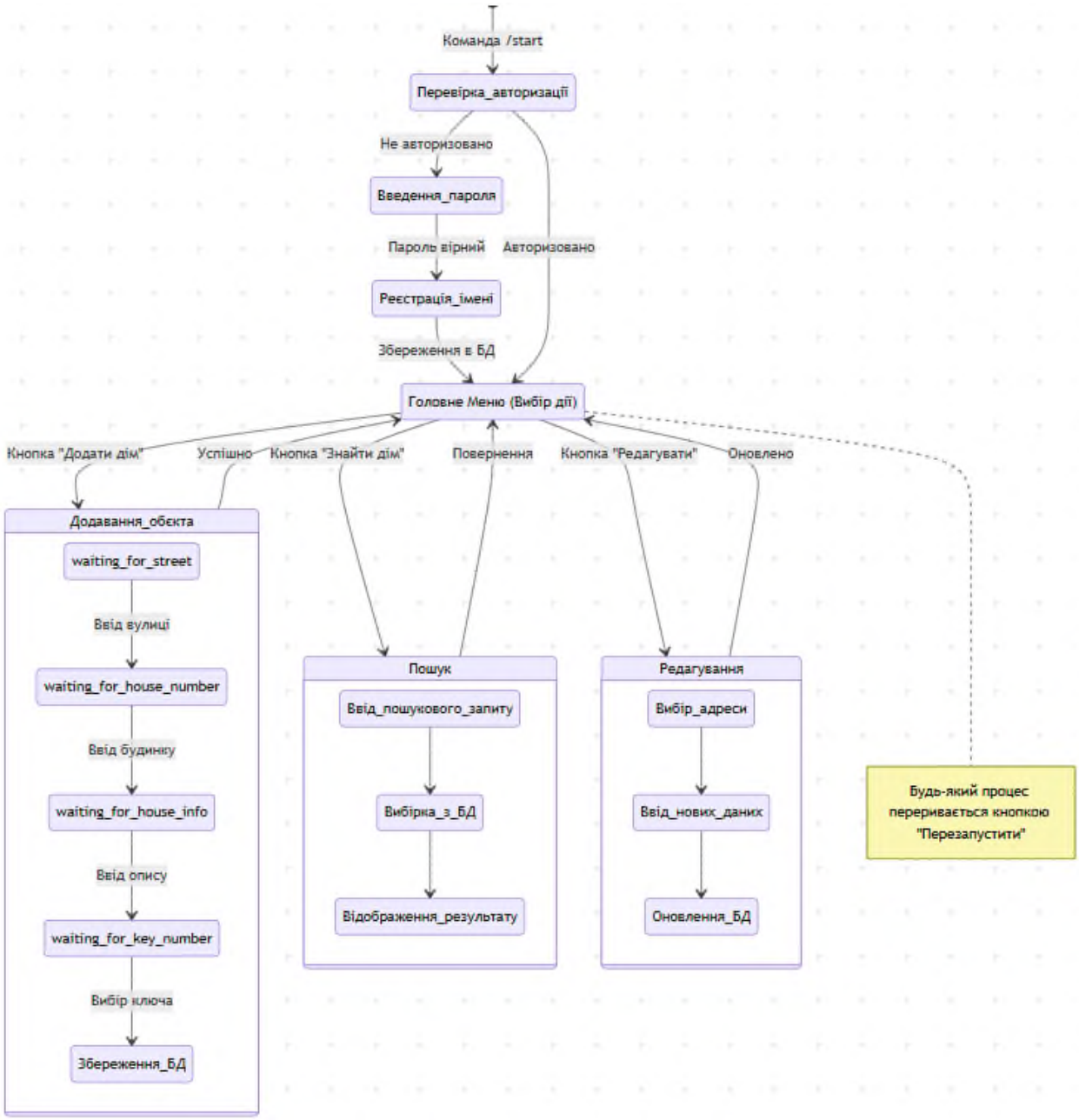

Рисунок 2.4 – Функціональна схема станів програмного комплексу Телеграм-бота

2.5 Розробка алгоритмів функціонування Телеграм-бота

Для забезпечення безперебійної, ефективної та, перш за все, безпечної роботи програмного комплексу було спроєктовано та реалізовано низку алгоритмів, що відповідають за багаторівневий контроль доступу, валідацію

вхідних даних та швидку селекцію технічної інформації з реляційної бази даних.

Оскільки розроблений Телеграм-бот оперує конфіденційними даними підприємства, що становлять комерційну та службову таємницю (зокрема, цифрові коди доступів до під'їздів та щитових, точне розташування схованок із ключами, а також персональні контактні дані голів ОСББ та відповідальних осіб), центральним елементом безпеки є алгоритм авторизації та багаторівневої ідентифікації користувачів. Цей алгоритм виконує роль «інформаційного шлюзу», що повністю блокує доступ до функціоналу бота для неавторизованих осіб.

Алгоритм ініціюється автоматично при першому зверненні клієнта до бота (стандартна команда месенджера /start). На цьому етапі програмний комплекс за допомогою методів Telegram API отримує об'єкт Message, з якого витягується унікальний числовий ідентифікатор акаунта (telegram_id). Отримане значення передається до модуля auth.py, де формується параметризований SQL-запит до таблиці auth_users. Використання параметризації на цьому етапі є критично важливим, оскільки це унеможливлює проведення атак типу SQL-ін'єкцій.

Якщо за результатом запиту ідентифікатор знайдено в «білому списку», система автоматично визначає роль користувача (звичайний співробітник або адміністратор) за допомогою функції is_admin. Авторизований користувач миттєво отримує доступ до головного навігаційного меню, згенерованого з урахуванням його прав доступу.

У разі, якщо ідентифікатор користувача відсутній у базі даних, система переходить у стан очікування авторизаційного токена. Бот надсилає запит на введення пароля доступу, який жорстко заданий у конфігураційних файлах системи. Введене текстове значення проходить через функцію verify_password, яка здійснює посимвольне порівняння з еталонним паролем підприємства.

При успішній верифікації пароля активується механізм реєстрації нового профілю. Система переводить користувача у стан FSM (Finite State Machine) waiting_for_username, де бот пропонує ввести реальне ім’я та прізвище співробітника для забезпечення прозорого логування дій у майбутньому. Після отримання цих даних викликається функція save_user, яка в асинхронному режимі записує новий telegram_id, нікнейм та вказане ім’я до бази даних users.db, надаючи користувачеві базовий рівень прав доступу. Блок-схему цього багатокрокового алгоритму авторизації наведено на рис. 2.5.

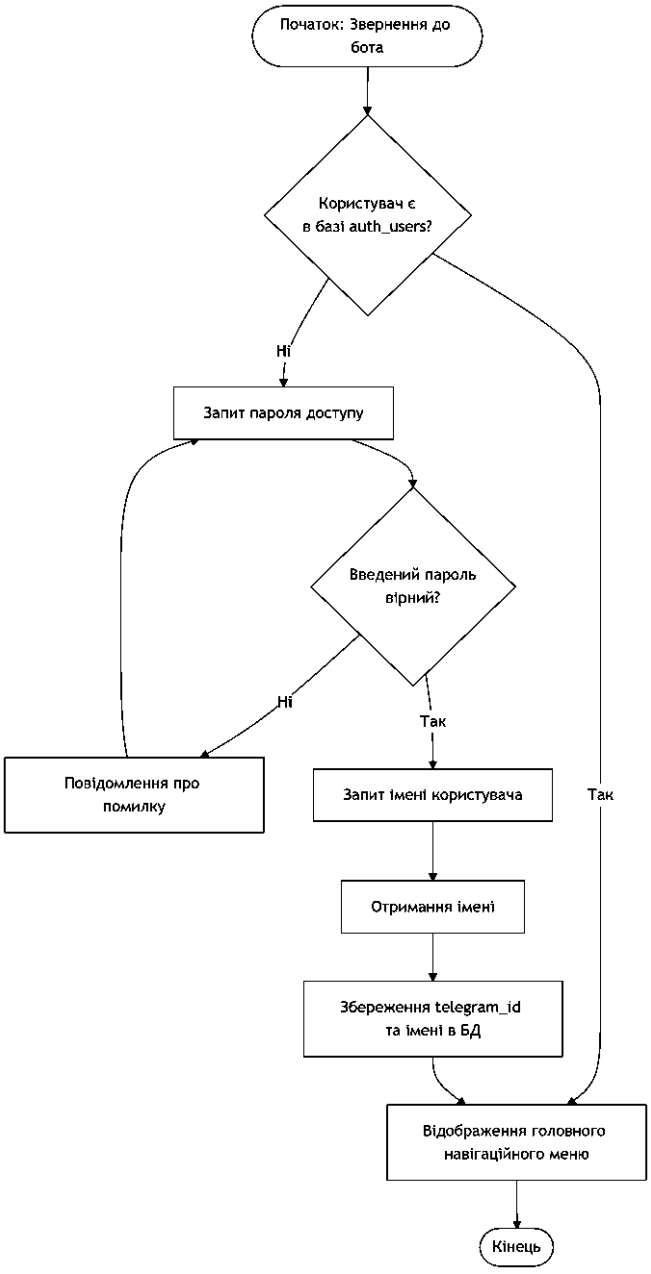

Рисунок 2.5 – Блок-схема алгоритму авторизації користувача у системі

Іншим важливим компонентом системи є алгоритм інтелектуального пошуку об'єктів за текстовим описом. На практиці часто виникають ситуації, коли виїзний майстер знає лише часткову або непряму інформацію про об'єкт: наприклад, номер телефону відповідальної особи, фрагмент імені голови ОСББ або специфічну технічну деталь, зафіксовану в примітках (наприклад, «ліфтерка», «вхід з двору» або «щитова на 2 поверсі»).

Алгоритм пошуку реалізований за принципом динамічного формування SQL-запитів, що дозволяє системі адаптуватися до запитів будь-якої складності. Процес обробки починається з нормалізації вхідних даних: отриманий від користувача пошуковий рядок очищується від зайвих пробілів та розбивається на масив окремих лексем (слів) за допомогою методу split(). Далі програма ітераційно проходить по кожному слову та генерує для нього індивідуальну умову пошуку.

Для забезпечення високої гнучкості системи умови налаштовуються за допомогою оператора LIKE із використанням масок %. Пошук здійснюється одночасно у двох полях — description (технічний опис) та street (назва вулиці). Важливою технічною особливістю є використання SQL-функції LOWER(), що приводить і вхідний запит, і дані в таблиці до нижнього регістру. Це робить пошук регістронезалежним, що критично важливо при швидкому введенні тексту з мобільного пристрою.

З точки зору кібербезпеки, алгоритм використовує механізм параметризованих запитів. Замість прямої вставки тексту в SQL-команду, програма передає пошукові слова як окремі параметри кортежу, що повністю нівелює ризик проведення атак типу SQL-ін'єкція.

Ключовою архітектурною перевагою алгоритму є логічне об’єднання всіх згенерованих умов оператором AND. Це означає, що результуюча вибірка міститиме лише ті записи, які відповідають усім введеним словам одночасно. Такий підхід дозволяє майстру поступово уточнювати пошук, додаючи нові

слова, що значно підвищує релевантність результатів і зменшує обсяг вихідного трафіку.

Для покращення користувацького досвіду (UX) в алгоритм інтегровано допоміжну функцію get_context_around_search(). Вона аналізує знайдений опис і виводить у повідомлення бота не весь текст, а лише фрагмент навколо знайденого ключового слова. Це дозволяє користувачеві миттєво зрозуміти контекст знайденої інформації без необхідності перечитувати великі блоки технічних приміток. Блок-схему алгоритму інтелектуального пошуку наведено на рис. 2.6.

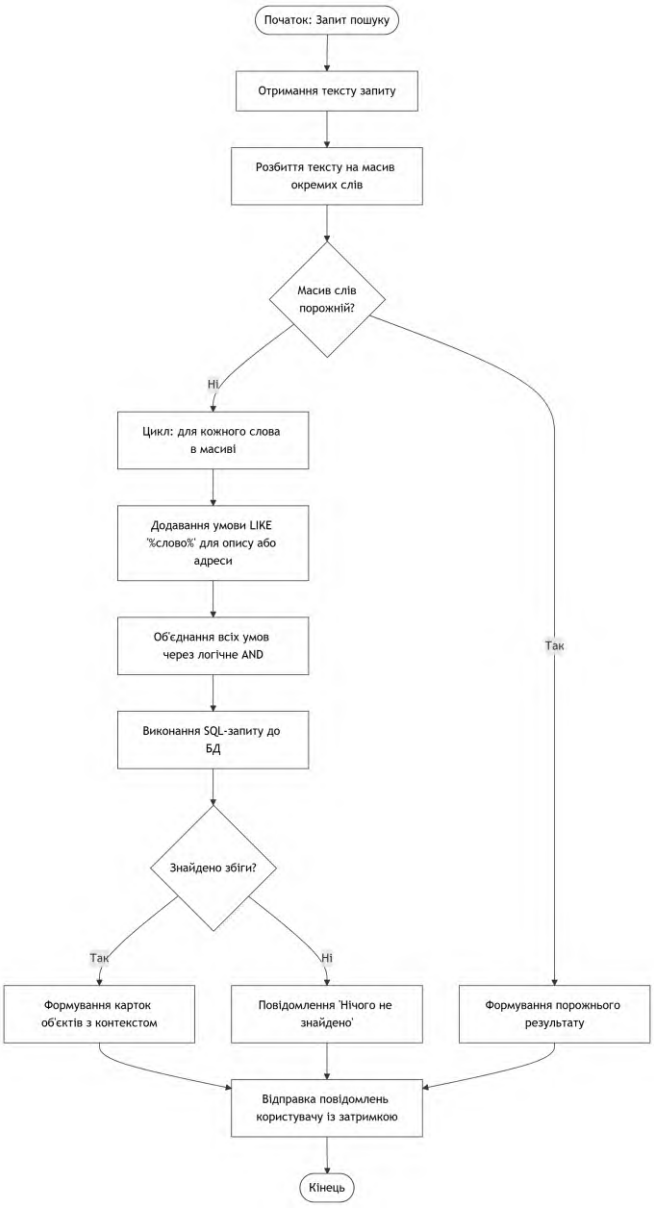

Рисунок 2.6 – Блок-схема алгоритму інтелектуального пошуку інфраструктурного об'єкта

Після успішного отримання результатів з бази даних, алгоритм обробляє кожний знайдений запис, виділяє контекст навколо знайденого слова для зручного попереднього перегляду та формує відповідь у вигляді серії повідомлень з кнопками "Детальніше". Щоб уникнути блокування зі сторони Telegram API через перевищення ліміту відправки повідомлень (Flood Control), алгоритм використовує асинхронну затримку (asyncio.sleep) між відправкою кожного результату.

2.6 Розробка моделі інтерфейсу розподіленого програмного комплексу

Оскільки розроблюваний програмний комплекс функціонує на базі месенджера Telegram, класичне розуміння графічного інтерфейсу користувача (GUI) з його вікнами, складними формами та спливаючими повідомленнями тут замінюється концепцією розмовного інтерфейсу (CUI — Conversational User Interface). Такий підхід дозволяє перенести акцент з візуальної складності на лінійну логіку взаємодії, що є критично важливим для технічного персоналу, який працює в мобільних умовах.

Користувацький інтерфейс будується з використанням стандартних, але гнучких компонентів Telegram API:

- текстові повідомлення — виступають основним каналом передачі технічної інформації та інструкцій від бота;
- звичайні клавіатури (Reply Keyboards) — замінюють стандартну клавіатуру пристрою на набір статичних кнопок, що забезпечують швидку навігацію між основними розділами системи («Додати будинок», «Пошук», «Статистика»);
- вбудовані клавіатури (Inline Keyboards) — інтегруються безпосередньо у тіло повідомлення, дозволяючи виконувати

контекстні дії (редагування, видалення або перегляд фото) безпосередньо в межах конкретного об’єкта.

Такий підхід забезпечує абсолютну кросплатформеність та ідентичний вигляд інтерфейсу на будь-яких пристроях — від смартфонів до десктопних клієнтів, незалежно від операційної системи.

Початковим та критично важливим етапом взаємодії користувача з системою є процес авторизації, який виконує функцію безпекового шлюзу. Інтерфейс авторизації реалізований у форматі інтерактивного діалогу: бот блокує доступ до всіх функцій, доки не отримає валідний пароль доступу. У разі введення некоректних даних система миттєво видає візуальне попередження за допомогою емодзі EMOJI_ERROR (❌).

При успішній верифікації пароля бот ініціює запит на введення імені співробітника. Це необхідно для персоналізації сесії та коректного логування дій у базі даних. Тільки після збереження ідентифікаційних даних система генерує головне навігаційне меню та відкриває доступ до технічної бази об'єктів (рис. 2.7).

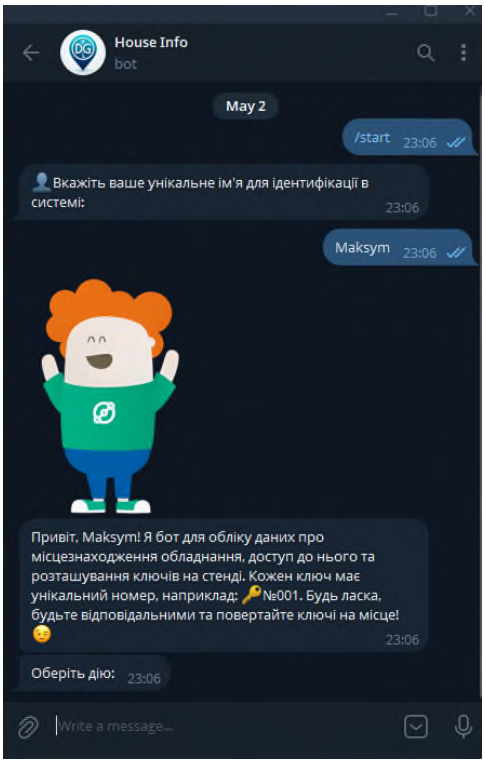

Рисунок 2.7 – Інтерфейс авторизації користувача у системі Телеграм-бота

Після ідентифікації інтерфейс системи перемикається на головний навігаційний екран. Меню реалізоване у вигляді закріпленої клавіатури (ReplyKeyboardMarkup), яка замінює стандартну клавіатуру смартфона. Згідно з розробленою логікою, головне меню містить наступні українськомовні елементи керування:

– «🏠 Додати будинок 🏠» та «📖 Список будинків 📖» — основні функції для розширення бази та перегляду наявних об'єктів.

– «🔍 Знайти будинок 🔍» та «🔎 Знайти за описом 🔎» — інструменти швидкого пошуку за адресою або технічними характеристиками.

– «🖍️ Редагувати 🖍️» та «❌ Видалити ❌» — функції для актуалізації інформації.

– «🕰️ Історія 🕰️», «📈 Моя статистика 📈» та «🏆 Топ адрес 🏆» — блок аналітики та контролю активності персоналу.

– «🔄 Перезапустити 🔄» — критично важлива кнопка, яка присутня у всіх багатокрокових сценаріях для миттєвого скидання стану (FSM) та повернення до початку.

Скріншот головного навігаційного меню наведено на рис. 2.8.

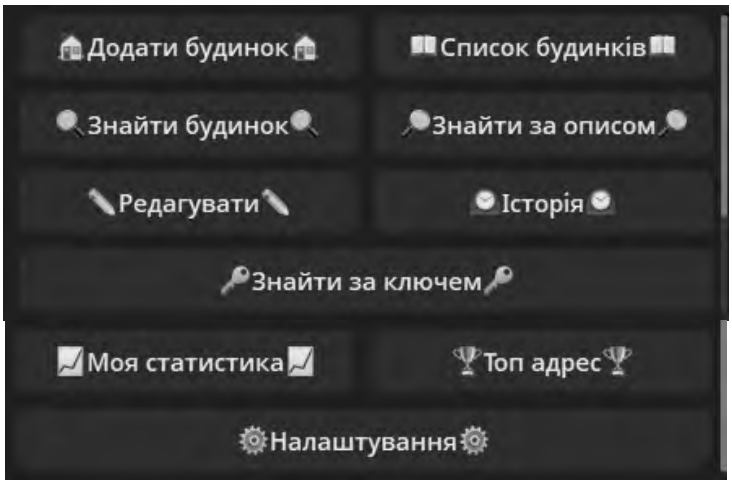

Рисунок 2.8 – Головне навігаційне меню Телеграм-бота

Інтерфейс відображення об'єктів (Картка будинку). Модель представлення даних про інфраструктурний об'єкт реалізована у вигляді структурованого текстового блоку (рис. 2.9). Для покращення читабельності використано марковані списки та емодзі:

- 📍 — для позначення адреси;
- 🔑 — для виділення інформації про ключі та коди;
- 📝 — для технічного опису.

Під кожною карткою об'єкта генерується InlineKeyboardMarkup. Це дозволяє реалізувати контекстне управління: користувач може натиснути «🖼️Фото» для перегляду медіа-файлів саме цього будинку або «❌Видалити❌» (для адміністраторів), не захаращуючи при цьому історію чату новими командами.

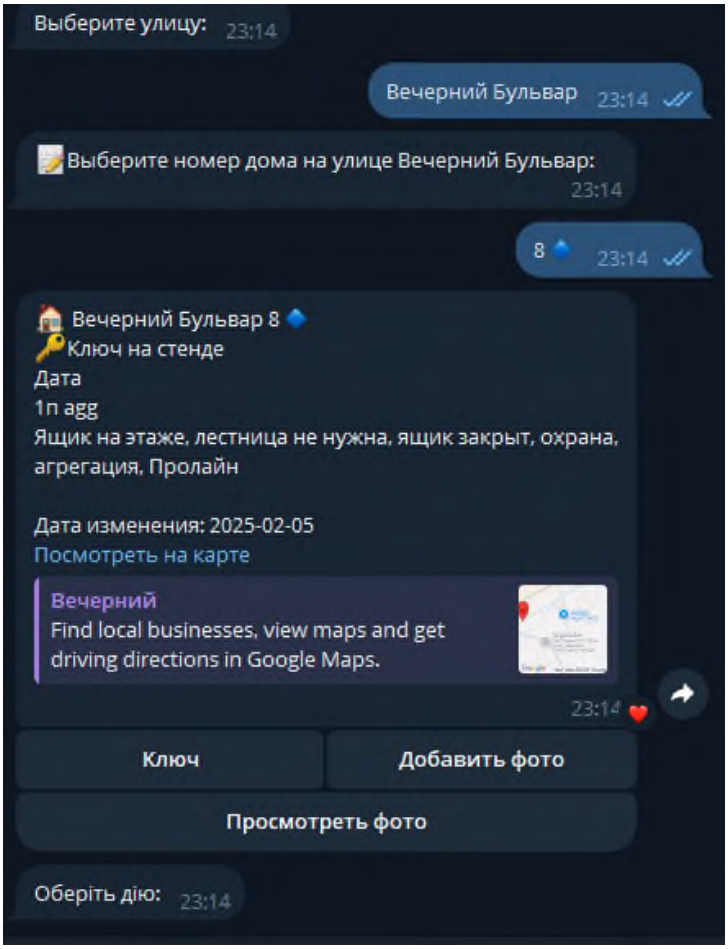

Рисунок 2.9 – Картка об'єкта з інтерактивними кнопками управління

Форми введення та редагування. Процес створення нового елемента (додавання будинку) реалізований як покроковий діалог. Замість однієї складної форми, як у традиційних додатках, бот пропонує послідовно ввести вулицю, номер будинку та опис. На етапі вибору вулиці інтерфейс пропонує готові варіанти з бази даних, що прискорює роботу та валідує дані вже на етапі введення, що зображено на рисунку 2.10.

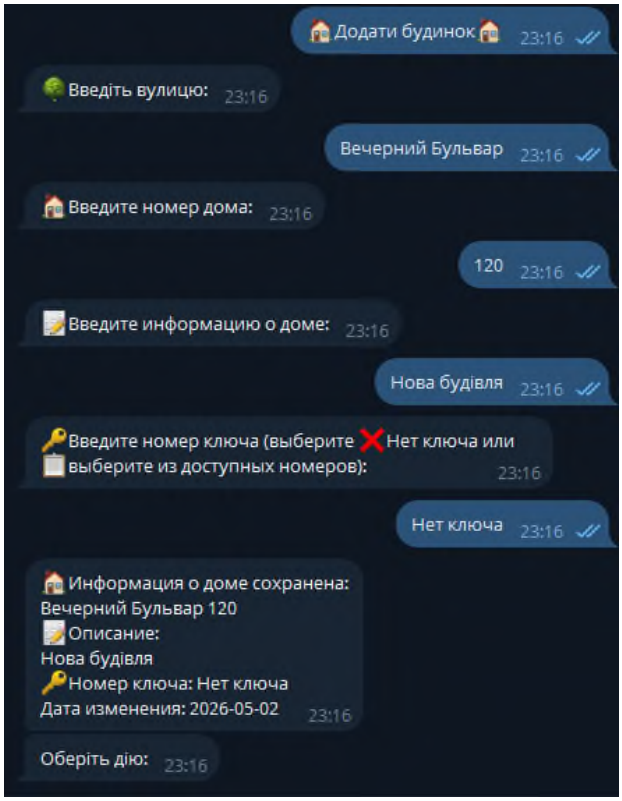

Рисунок 2.10 – Додавання нового об’єкту

Загалом, модель інтерфейсу Телеграм-бота побудована таким чином, щоб забезпечити максимальну швидкість доступу до даних (1-2 секунди) при мінімальному навантаженні на канал зв'язку, що повністю відповідає завданням розробки розподіленої системи адміністрування.

3 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ТЕЛЕГРАМ-БОТА ДЛЯ АДМІНІСТРУВАННЯ ІНФРАСТРУКТУРНИХ ОБ'ЄКТІВ

3.1 Розробка бази даних і забезпечення розподіленого доступу до масивів даних

Серед багатьох можливих варіантів для реалізації бази даних серверної частини Телеграм-бота найбільш оптимальним рішенням є легка реляційна система управління базами даних SQLite. На відміну від важких клієнт-серверних СУБД (таких як PostgreSQL або MySQL), SQLite не вимагає налаштування окремого сервера, а зберігає всі дані у звичайних локальних файлах (у даному проєкті це файли houses.db та users.db). Оскільки програмний комплекс орієнтований на роботу з текстовими даними, кодами та посиланнями на медіа-файли, використання цієї системи забезпечує необхідну швидкодію, мінімальне споживання оперативної пам'яті сервера та максимальну зручність при створенні резервних копій.

Згідно з логічною структурою, розробленою в розділі 2, було здійснено фізичне проєктування таблиць бази даних. При розробці програмного комплексу мовою Python для взаємодії з БД використовується вбудована бібліотека sqlite3 [6, 23].

Оскільки Телеграм-бот працює в асинхронному режимі і може одночасно отримувати запити від десятків виїзних майстрів, критично важливим було вирішення проблеми розподіленого доступу. SQLite за замовчуванням блокує базу даних при спробі одночасного запису з різних потоків. Для вирішення цієї проблеми у програмному коді (модуль db_context.py) було розроблено спеціальний менеджер контексту get_db_connection. Він ініціалізує з'єднання з параметром check_same_thread=False, що дозволяє безпечно використовувати одне

підключення різними асинхронними задачами. Менеджер контексту автоматично відкриває курсор для виконання SQL-запиту, фіксує зміни (commit) та гарантовано закриває з'єднання у блоці finally після завершення операції, запобігаючи виникненню помилки "database is locked".

Загальний вигляд, фізичні типи даних та зв’язки між таблицями бази даних програмного комплексу представлено на рис. 3.1 у вигляді діаграми.

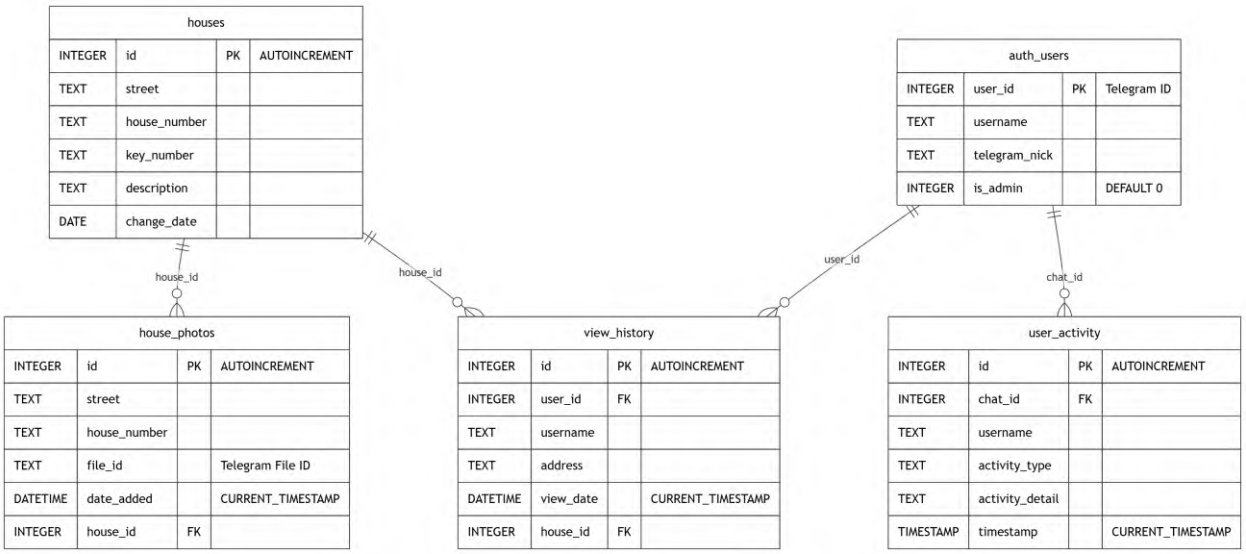

Рисунок 3.1 – Фізична діаграма зв’язків таблиць бази даних Телеграм-бота

Ще однією важливою архітектурною особливістю розробленої підсистеми роботи з даними є механізм автоматичної ініціалізації та міграцій. Модулі бази даних (наприклад, функція initialize_auth_table) містять SQL-інструкції CREATE TABLE IF NOT EXISTS. Завдяки цьому, при першому запуску програми на новому сервері система самостійно формує всю необхідну структуру таблиць. Більше того, реалізовано механізм базових міграцій через перехоплення виключення sqlite3.OperationalError (наприклад, при спробі додати нову колонку is_admin до існуючої таблиці), що робить процес оновлення програмного продукту безшовним і не потребує втручання адміністратора бази даних.

3.2 Розробка програмних модулів розподіленого доступу до даних

Розробка серверної частини програмного комплексу проводиться у сучасному інтегрованому середовищі розробки (IDE) з використанням мови програмування Python. Вибір даної мови зумовлений її високою гнучкістю, наявністю потужних бібліотек для взаємодії з мережевими протоколами та внутрішніми стандартами, що дозволяють швидко прототипувати та масштабувати інформаційні системи. В якості основного фреймворку для взаємодії з Telegram Bot API було обрано бібліотеку aiogram. Її головною перевагою є нативна підтримка асинхронного вводу-виводу (asyncio). В умовах розподіленої системи, коли десятки виїзних майстрів можуть одночасно надсилати запити до сервера, класичний синхронний підхід призвів би до блокування основного потоку виконання програми. Асинхронна модель дозволяє серверу обробляти наступні повідомлення, поки попередні очікують на відповідь від бази даних або мережі, що кардинально підвищує пропускну здатність системи.

З урахуванням того, що система обробки даних базується на легкій СУБД SQLite, виникла необхідність вирішення класичної проблеми розподіленого доступу: блокування файлу бази даних при одночасних транзакціях з різних асинхронних потоків (помилка database is locked). Для забезпечення стабільної роботи без використання важких ORM-систем (Object-Relational Mapping), було розроблено власний архітектурний прошарок доступу до даних (Data Access Layer).

Основою цього прошарку є менеджер контексту (Context Manager), реалізований у модулі db_context.py. На рис. 3.2 зображено програмний код даного механізму.

```
@contextmanager  1 usage
def get_db_connection(db_path : str =DATABASE_PATH):
    conn = None
    try:
        conn = sqlite3.connect(db_path, check_same_thread=False)
        cursor = conn.cursor()
        yield conn, cursor
        conn.commit()
    except sqlite3.Error as e:
        if conn:
            conn.rollback()
        raise e
    finally:
        if conn:
            conn.close()
```

Рисунок 3.2 – Реалізація менеджера контексту для безпечного керування з'єднаннями з базою даних

Як можна побачити з наведеного лістингу, функція ініціалізує підключення з критично важливим параметром check_same_thread=False, який дозволяє передавати об'єкт з'єднання між різними асинхронними задачами. Використання конструкції try...yield...finally гарантує, що після виконання SQL-запиту системою буде автоматично викликано метод commit() для збереження змін, а у разі виникнення помилки — метод rollback() для відміни транзакції та збереження цілісності даних. Закриття з'єднання (conn.close()) відбувається гарантовано у будь-якому випадку, що виключає виникнення «витоків пам'яті» (memory leaks).

Для безпосереднього виконання запитів до бази даних було створено клас-обгортку (модуль database.py). Це дозволило абстрагувати бізнес-логіку програми від низькорівневих SQL-команд. Програмний код універсальної функції виконання запитів наведено на рис. 3.3.

```
def execute_query(query, params : Any =()):
    """Выполняет запрос к базе данных."""
    conn, cursor = get_connection()
    cursor.execute(query, params)
    result = cursor.fetchall()
    conn.commit()
    conn.close()
    return result
```

Рисунок 3.3 – Програмний модуль абстрагування SQL-запитів

Дана функція приймає текстовий SQL-запит та кортеж параметрів (params). Використання параметризованих запитів (підстановка значень через символ ?) є обов'язковою вимогою кібербезпеки, оскільки це повністю виключає можливість проведення атак типу SQL-ін'єкція (SQL Injection) при введенні майстрами текстових даних з мобільного пристрою.

Наступним важливим етапом розробки модулів доступу до даних стала реалізація механізму автоматичної міграції та ініціалізації структури бази даних. Оскільки програмний комплекс повинен легко розгортатися на нових серверах, він не потребує попереднього створення файлів БД вручну. На рис. 3.4 представлено програмний код ініціалізації таблиці активності користувачів.

```
def initialize_user_activity():  2+ usages
    """Создает таблицу user_activity, если она не существует."""
    conn, cursor = get_connection()
    if conn is not None:
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS user_activity (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                chat_id INTEGER,
                username TEXT,
                activity_type TEXT,
                activity_detail TEXT,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        """)
        conn.commit()
        conn.close()
```

Рисунок 3.4 – Код модуля автоматичної ініціалізації таблиць бази даних

Завдяки конструкції CREATE TABLE IF NOT EXISTS, система при кожному запуску перевіряє цілісність внутрішньої структури файлу SQLite. Якщо необхідної таблиці немає, вона створюється динамічно з усіма необхідними типами даних (INTEGER, TEXT, TIMESTAMP). Це значно спрощує процес оновлення програмного продукту на підприємстві.

Оскільки програмний комплекс розробляється для умов, де надійність є ключовим фактором, до модулів доступу до даних було інтегровано підсистему автоматичного резервного копіювання. З огляду на те, що база даних SQLite зберігається як звичайний файл, процес бекапу реалізовано на рівні файлової системи. Код модуля резервного копіювання наведено на рис. 3.5.

```
def backup_database():  2+ usages
    """Создает резервную копию базы данных."""
    try:
        current_time = datetime.now().strftime("%Y%m%d%H%M%S")
        backup_folder = 'backups'

        # Создание папки для резервных копий, если её нет
        if not os.path.exists(backup_folder):
            os.makedirs(backup_folder)

        backup_filename = os.path.join(backup_folder, f"backup_{current_time}.db")

        # Создание резервной копии базы данных
        shutil.copy2(DATABASE_PATH, backup_filename)

        logger.info(f"База данных успешно скопирована в {backup_filename}")
    except Exception as e:
        logger.error(f"Ошибка при создании резервной копии базы данных: {str(e)}")
```

Рисунок 3.5 – Програмна реалізація модуля резервного копіювання бази даних

Цей модуль використовує стандартну бібліотеку shutil для фізичного копіювання файлу бази даних до окремої директорії backups. Кожній копії автоматично присвоюється унікальне ім'я, згенероване на основі поточного часу (datetime.now()). Ця функція викликається системою перед кожною деструктивною операцією (наприклад, перед видаленням об'єкта

інфраструктури), що гарантує можливість відновлення втраченої інформації у разі непередбачених збоїв або помилкових дій персоналу.

Таким чином, розроблена архітектура програмних модулів доступу до даних повністю задовольняє вимоги підприємства щодо швидкодії, безпеки та надійності зберігання інформації.

3.3 Розробка програмних модулів автоматизованої системи

Програмний комплекс Телеграм-бота за своєю архітектурою є подієво-орієнтованою системою, яка базується на обробці вхідних оновлень (Updates) від серверів Telegram. На відміну від класичних клієнт-серверних систем з REST API, де взаємодія ініціюється клієнтом через HTTP-запити, Телеграм-бот використовує механізм «довгих опитувань» (Long Polling) або веб-хуків, де центральним вузлом обробки є диспетчер (Dispatcher).

Диспетчер виконує роль головного маршрутизатора, який розподіляє вхідні повідомлення, натискання кнопок та завантажені файли між відповідними програмними модулями — обробниками (Handlers). У розробленому комплексі реалізовано чіткий поділ логіки, що відповідає парадигмі, близькій до MVC (Model-View-Controller), де обробники виступають у ролі контролерів. На рис. 3.6 представлено фрагмент коду реєстрації основних обробників команд у головному модулі програми.

```
# Регистрация обработчиков команд
@dp.message(CommandStart())
async def start_handler(message: Message, state: FSMContext):
    await send_welcome(message, bot, state)


@dp.message(Command("help"))
async def help_handler(message: Message):
    await send_help(message, bot)
```

Рисунок 3.6 – Програмна реєстрація обробників подій у диспетчері

Однією з фундаментальних функцій системи є вибірка та відображення списків об'єктів інфраструктури. Цей модуль забезпечує читання масивів даних із таблиці houses. Оскільки об'єм даних може бути значним, реалізація передбачає асинхронне отримання даних, що дозволяє боту залишатися чуйним до інших користувачів під час виконання важких SQL-запитів. Код модуля отримання повного списку будинків наведено на рис. 3.7.

```
@dp.message(lambda m {text} : m.text and m.text.lower() == '📖список будинків📖')
async def list_houses_handler(message: Message, state: FSMContext):
    await list_houses(message, bot, state)
```

Рисунок 3.7 – Реалізація модуля отримання списку об'єктів з бази даних

Для забезпечення оперативного доступу до конкретної інформації було розроблено модуль ідентифікації та отримання даних одного запису. Цей механізм спрацьовує, коли майстер вводить конкретну адресу або вибирає об'єкт із запропонованого списку. Система ініціює запит до БД з використанням унікального ідентифікатора об'єкта або параметризованого пошуку за текстовим рядком. Якщо об'єкт знайдено, модуль формує структуровану відповідь, додаючи до неї Inline-кнопки для подальшої взаємодії (редагування або видалення). Програмний код запиту даних конкретного об'єкта представлено на рис. 3.8.

```
@dp.message(lambda m {text} : m.text and m.text.lower() != '🔄перезапустити🔄', StateFilter("waiting_for_search_street"))
async def process_search_street_state(message: Message, state: FSMContext):
    from houses import process_search_street
    await process_search_street(message, message.chat.id, bot, state)
```

Рисунок 3.8 – Програмний код модуля детального пошуку за ідентифікатором

Процес створення нових записів (додавання будинку) є найбільш складною частиною автоматизованої системи, оскільки він вимагає

послідовного збору даних від користувача. Для цього використано механізм FSM (Finite State Machine). Кожен крок введення (вулиця, номер будинку, опис) є окремим станом системи. Модуль створення записів зчитує дані з тимчасового сховища станів (state.get_data()) і після завершення діалогу формує фінальний SQL-запит INSERT. Прикладом реалізації фінального етапу створення об'єкта є код, наведений на рис. 3.9

```
async def save_key_number(message: Message, chat_id, street, house_number, description, bot, state=None):  2+ usages
    """Сохраняет номер ключа и информацию о доме."""
    if message.text and message.text.lower() == RESTART_BUTTON.lower():
        from handlers import stop_process
        await stop_process(message, bot, state)
    else:
        key_number = message.text.strip()
        if key_number.lower() == NO_KEY_TEXT.lower():
            key_number = None
        change_date = datetime.now().strftime("%Y-%m-%d")
        conn, cursor = get_connection()
        try:
            cursor.execute('INSERT INTO houses (chat_id, street, house_number, description, key_number, change_date)
                           (chat_id, street, house_number, description, key_number, change_date))
            conn.commit()

            username = get_username(chat_id)
            log_user_activity(chat_id, username,  activity_type: "добавил дом",  activity_detail: f"{street} {house_number}")

            backup_database()

            await bot.send_message(
                chat_id,
```

Рисунок 3.9 – Код модуля створення нового інфраструктурного об'єкта

Завершальним елементом функціональності є модуль редагування наявних записів. Логіка цього модуля поєднує в собі механізми пошуку та оновлення даних. Система спочатку завантажує поточні дані об'єкта, пропонує користувачу вибрати поле для зміни (наприклад, «Змінити код домофону»), а потім виконує асинхронний запис оновлених значень. Для гарантування безпеки цей модуль обов'язково викликає функцію резервного копіювання перед внесенням змін. Програмну реалізацію оновлення записів наведено на рис. 3.10.

```
async def edit_house(message: Message, bot, state=None):  3+ usages
    """Начинает процесс редактирования дома."""
    chat_id = message.chat.id
    conn, cursor = get_connection()
    try:
        cursor.execute('SELECT DISTINCT street FROM houses ORDER BY street')
        streets = cursor.fetchall()
        if streets:
            markup = create_street_keyboard(streets, row_width=2)
            await message.answer( text: "🏠Выберите улицу для редактирования:", reply_markup=markup)
            if state:
                await state.set_state("waiting_for_edit_street")
        else:
            await message.answer("Нет сохраненных домов.")
            await show_navigation_menu(message, bot)
    except Exception as e:
        await message.answer(f"Ошибка при получении списка улиц: {e}")
        await show_navigation_menu(message, bot)
    finally:
        conn.close()
```

Рисунок 3.10 – Програмна реалізація модуля редагування даних

Важливою особливістю всіх розроблених модулів є вбудована обробка виключень. У разі виникнення помилок з'єднання з базою даних або перевищення лімітів Telegram API (Flood limits), система не припиняє роботу, а інформує користувача про затримку та автоматично повторює запит через певний проміжок часу. Це забезпечує високу стабільність комплексу в реальних промислових умовах.

3.4 Розробка елементів ергономічного користувацького інтерфейсу

Проєктування та розробка користувацького інтерфейсу програмного комплексу базувалася на принципах мінімалізму та максимальної швидкості доступу до критично важливої інформації в польових умовах. Оскільки основними користувачами системи є виїзні майстри, інтерфейс було адаптовано під концепцію «керування одним пальцем», що реалізується через розмовний інтерфейс (Conversational User Interface, CUI). На відміну від традиційних мобільних застосунків, де інтерфейс описується мовами розмітки

на кшталт XAML, у даному проєкті побудова графічних елементів реалізована за допомогою програмних засобів фреймворку aiogram, які динамічно формують об’єкти інтерфейсу на стороні сервера [24].

Для забезпечення високого рівня ергономіки було розроблено спеціалізований модуль keyboard_builder.py, який відповідає за автоматичну генерацію та масштабування кнопок. Програмний код цього модуля, що демонструє використання методів для створення адаптивних клавіатур, представлено на рис. 3.11.

```
def create_keyboard_with_restart(items, row_width : int =1, one_time : bool =False):  3+ usages
    keyboard = []
    keyboard.append([KeyboardButton(text=RESTART_BUTTON)])

    if row_width == 1:
        for item in items:
            keyboard.append([KeyboardButton(text=str(item))])
    elif row_width == 2:
        for i in range(0, len(items), 2):
            buttons = [KeyboardButton(text=str(items[i]))]
            if i + 1 < len(items):
                buttons.append(KeyboardButton(text=str(items[i + 1])))
            keyboard.append(buttons)
    elif row_width == 3:
        for i in range(0, len(items), 3):
            buttons = [KeyboardButton(text=str(items[j])) for j in range(i, min(i + 3, len(items)))]
            keyboard.append(buttons)
    else:
        # Для других значений используем стандартный подход
        for item in items:
            keyboard.append([KeyboardButton(text=str(item))])

    return ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True, one_time_keyboard=one_time)
```

Рисунок 3.11 – Програмна реалізація методів динамічної побудови ергономічних клавіатур

Основними інструментами забезпечення взаємодії в системі виступають наступні типи елементів:

- статичні навігаційні клавіатури (ReplyKeyboardMarkup) — закріплені в нижній частині екрана кнопки, що мають параметр

resize_keyboard=True для автоматичної адаптації під роздільну здатність дисплея смартфона;

- контекстні вбудовані кнопки (InlineKeyboardMarkup) — динамічні елементи, що прив'язані до конкретних повідомлень і дозволяють виконувати операції (редагування, видалення, перегляд фото) без необхідності введення текстових команд;
- візуальні індикатори (Emoji-маркери) — використання стандартизованих емодзі для швидкого візуального сканування інформації (наприклад, 🏠 для адрес, 🔑 для ключів, ❌ для помилок), що знижує когнітивне навантаження на користувача;
- універсальна кнопка переривання («🔄 Перезапустити 🔄») — ергономічне рішення, що дозволяє користувачеві миттєво вийти з будь-якого багатокрокового стану (FSM) та повернутися до головного меню у разі помилкового введення.

Знімок екрана початкового етапу взаємодії — вікна авторизації — наведено на рис. 3.12. Ергономіка цього екрана полягає у відсутності будь-яких відволікаючих елементів: користувач бачить лише чітку інструкцію та поле для введення, що мінімізує час на вхід у систему.

Рисунок 3.12 – Реалізація ергономічного вікна авторизації та реєстрації

оловна форма програмного комплексу (рис. 3.13) спроєктована таким чином, щоб найбільш вживані функції знаходилися у верхній частині клавіатури. Для адміністраторів система автоматично доповнює інтерфейс службовим блоком кнопок («💾 База даних 💾», «👥 Користувачі 👥»), відокремлюючи їх візуальним роздільником від основних функцій.

Рисунок 3.13 – Основна форма головного меню з розподілом функціональних зон

Для відображення технічної інформації було розроблено формат «Картки об’єкта» (рис. 3.14). Ергономіка цього вікна забезпечується використанням HTML-розмітки для виділення жирним шрифтом ключових адрес та інтеграцією гіперпосилань для швидкого переходу до Google Maps. Inline-кнопки під повідомленням дозволяють майстру миттєво отримати доступ до фотографій або номерів ключів, не втрачаючи фокусу на основних даних об’єкта.

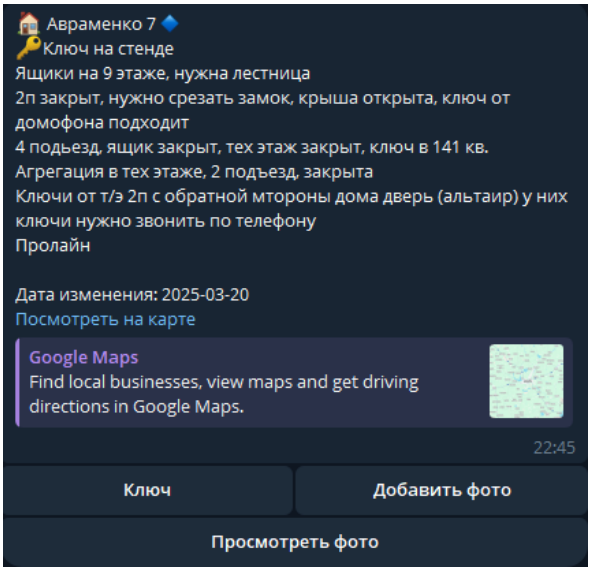

Рисунок 3.14 – Інформаційна картка об’єкта з інтегрованими елементами швидкого доступу

Тестування розробленого інтерфейсу на різних пристроях (смартфонах з діагоналлю від 5.5" до 6.7" та планшетах) підтвердило, що обрана модель розмовного інтерфейсу забезпечує високу швидкість роботи персоналу та виключає необхідність тривалого навчання нових співробітників завдяки своїй інтуїтивності та відповідності стандартам сучасних мобільних застосунків [27].

Окремої уваги при розробці інтерфейсу заслуговує реалізація багатокрокових сценарних операцій, зокрема процесу додавання нового інфраструктурного об'єкта. У класичних програмах для цього використовується єдина громіздка форма з багатьма текстовими полями. У розробленому комплексі цей процес адаптовано під мобільний формат і розбито на послідовний діалог за допомогою машини станів (FSM).

Бот послідовно надсилає користувачеві короткі та зрозумілі інструкції («Введіть вулицю», «Введіть номер будинку», «Введіть опис»), очікуючи на відповідь перед переходом до наступного кроку. На кожному етапі на екрані залишається закріпленою кнопка «🔄 Перезапустити 🔄», що є важливим елементом ергономіки: вона дозволяє майстру в один клік скасувати операцію,

якщо він припустився помилки, не змушуючи його проходити весь ланцюжок до кінця. Процес покрокової взаємодії при додаванні об'єкта представлено на рис. 3.15.

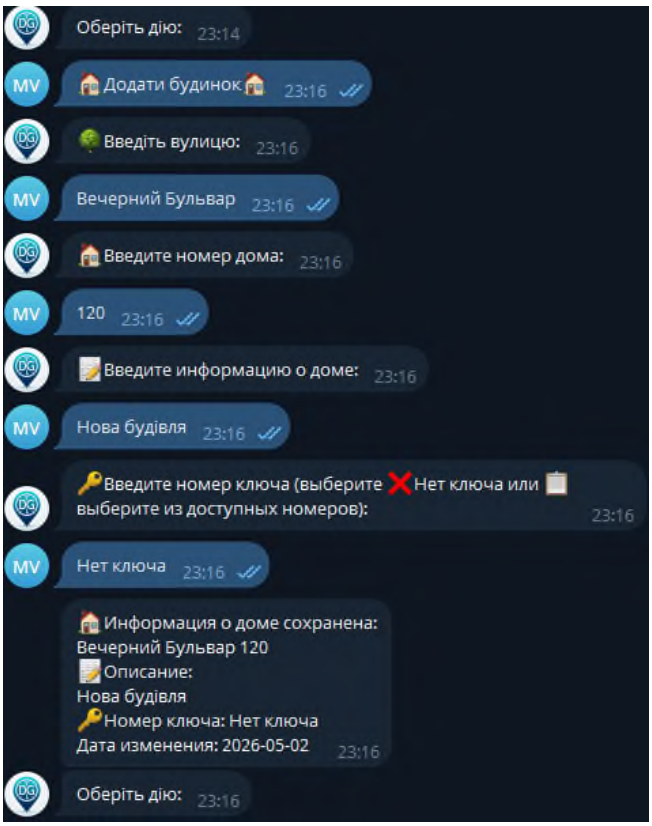

Рисунок 3.15 – Інтерфейс покрокового діалогового введення даних (FSM)

Ще одним важливим модулем користувацького інтерфейсу є блок адміністрування та налаштувань. Оскільки система повинна бути гнучкою, для користувачів з розширеними правами (адміністраторів) передбачено окремі діалогові вікна для управління персоналом. Наприклад, інтерфейс керування користувачами реалізовано у вигляді списку карток, де кожна картка містить інформацію про співробітника та набір контекстних Inline-кнопок («➕ Підвищити», «➖ Понизити», «✏️ Змінити ім'я»). Це дозволяє керівнику змінювати права доступу підлеглих безпосередньо з екрана смартфона кількома дотиками. Знімок екрана інтерфейсу керування користувачами зображено на рис. 3.16.

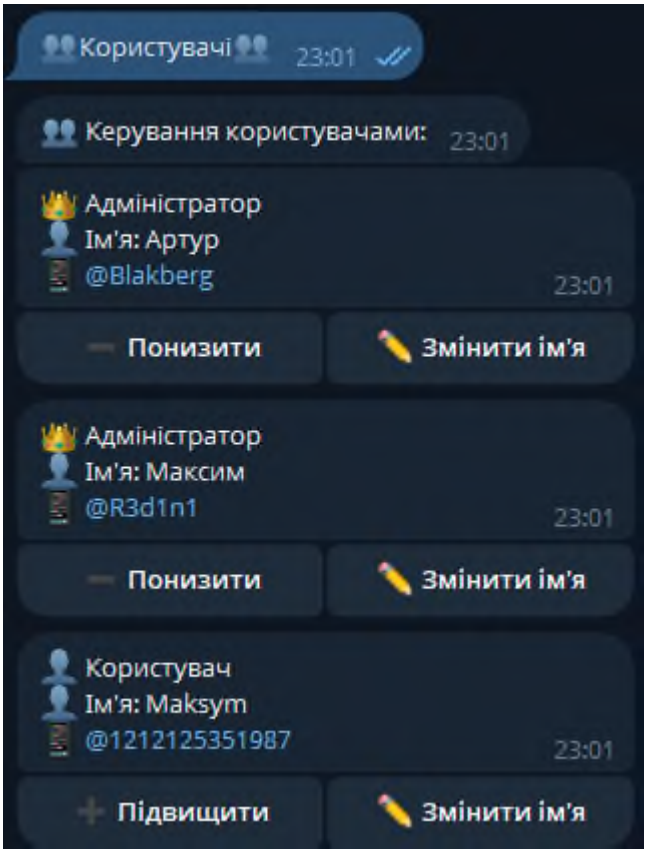

Рисунок 3.16 – Інтерфейс модуля адміністрування та управління правами доступу

Також в інтерфейсі передбачено вікно персональної статистики співробітника (рис. 3.17). При виклику цієї функції система формує структуроване зведення з використанням емодзі, яке наочно демонструє продуктивність майстра: кількість проглянутих адрес, доданих об'єктів та завантажених фотографій. Цей елемент інтерфейсу виконує мотиваційну функцію та дозволяє співробітникам самостійно відстежувати свою активність у системі.

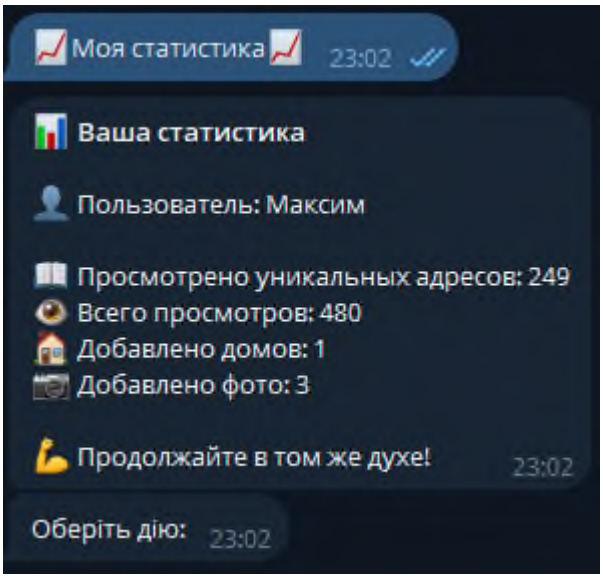

Рисунок 3.17 – Вікно відображення персональної статистики користувача

Підсумовуючи, можна стверджувати, що розроблений користувацький інтерфейс Телеграм-бота повністю відповідає сучасним вимогам до ергономіки мобільних додатків. Він позбавлений візуального сміття, оптимізований для роботи в умовах обмеженого часу та нестабільного зв'язку, а також забезпечує інтуїтивно зрозумілу взаємодію з базою даних підприємства без необхідності проведення додаткового навчання персоналу.

3.5 Програмні засоби забезпечення надійності та безпеки даних в розподілених каналах обміну інформацією

На відміну від класичних клієнт-серверних додатків, де розробник має самостійно реалізовувати алгоритми шифрування та стиснення трафіку, розроблюваний програмний комплекс використовує готову інфраструктуру Telegram. Надійність та конфіденційність передачі даних між мобільним клієнтом та серверами Telegram забезпечується криптографічним протоколом MTProto. Подальша передача запитів від серверів Telegram до серверної частини бота (API) здійснюється через захищене з'єднання HTTPS, що виключає можливість перехоплення конфіденційної інформації [20, 21].

Основним завданням забезпечення надійності на стороні сервера є гарантування цілісності та збереження даних при апаратних або програмних збоях. Для цього в системі реалізовано модуль автоматичного резервного копіювання бази даних SQLite [7]. Програмна реалізація цього механізму, що використовує системний час для генерації унікальних копій, наведена нижче:

```
def backup_database():
    """Створює резервну копію бази даних."""
    if not os.path.exists('backups'):
        os.makedirs('backups')
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    backup_path = f"backups/houses_backup_{timestamp}.db"
        try:
        shutil.copy2(DATABASE_PATH, backup_path)
        logger.info(f"Резервна копія створена: {backup_path}")
    except Exception as e:
        logger.error(f"Помилка при створенні резервної копії: {e}")
```

Дана функція викликається системою перед кожною деструктивною операцією або внесенням значних змін до записів об'єктів. Використання стандартної бібліотеки shutil.copy2() дозволяє копіювати файл разом із метаданими, що забезпечує швидке відновлення працездатності системи.

Ще одним важливим елементом надійності є забезпечення безпечного багатопотокового доступу до бази даних. Оскільки бот працює в асинхронному режимі, критично важливо уникати блокування файлу БД при одночасних запитах. Для цього було розроблено менеджер контексту, який гарантує коректне відкриття та закриття з'єднань:

```
@contextmanager
def get_db_connection(db_path=DATABASE_PATH):
    """Менеджер контексту для отримання з'єднання з базою даних."""
    conn = sqlite3.connect(db_path, check_same_thread=False)
    cursor = conn.cursor()
```

```
try:
    yield conn, cursor
    conn.commit()
except Exception as e:
    conn.rollback()
    raise e
finally:
    conn.close()
```

Цей алгоритм забезпечує автоматичний відкат транзакції (rollback) у разі виникнення помилки під час виконання SQL-запиту, що запобігає пошкодженню структури бази даних.

Для захисту від надлишкового мережевого навантаження та запобігання блокуванню з боку Telegram API (Flood Control), у системі реалізовано механізм обмеження частоти запитів (Throttling). Нижче представлений фрагмент коду, що контролює інтервали між зверненнями користувачів до важких операцій пошуку:

```
last_call_time = 0
async def list_houses(message: Message, bot: Bot):
    global last_call_time
    current_time = time.time()
      if current_time - last_call_time < THROTTLE_TIME:
        await message.answer("⚠️ Занадто багато запитів. Зачекайте кілька секунд.")
        return
      last_call_time = current_time
```

Використання алгоритмів резервного копіювання у поєднанні з програмним контролем транзакцій та частоти запитів забезпечує високу відмовостійкість серверної частини. Система здатна стабільно обробляти інформаційні потоки навіть за умов низької якості каналів зв'язку, що є характерним для експлуатації віддалених інфраструктурних об'єктів.

ВИСНОВОК

Цілю роботи була розробка мобільного програмного комплексу для управління первинним документообігом в промислових умовах низької надійності каналів зв’язку.

На Метою кваліфікаційної роботи була розробка автоматизованої системи адміністрування інфраструктурних об'єктів на базі месенджера Telegram для оптимізації роботи виїзного технічного персоналу сервісних підприємств.

На початку роботи було проведено ґрунтовний аналіз предметної області та існуючих методів управління розподіленою інфраструктурою (будинками, під'їздами, технічними приміщеннями). Виявлено критичні недоліки традиційного (паперового та табличного) обліку в польових умовах, а також проаналізовано наявні на ринку спеціалізовані рішення, які виявилися економічно недоцільними та перевантаженими зайвим функціоналом для мобільного використання.

З огляду на відсутність оптимального готового програмного забезпечення, що відповідало б вимогам швидкодії та простоти доступу, було сформовано завдання на розробку власного програмного комплексу. На етапі проєктування було розроблено структуру реляційної бази даних, визначено основні інформаційні потоки та створено модель ергономічного розмовного користувацького інтерфейсу (CUI).

Після затвердження концептуальної моделі розпочався пошук актуальних технологій. Для реалізації було обрано мову програмування Python та асинхронний фреймворк aiogram. У процесі розробки було реалізовано складні алгоритми інтелектуального пошуку об'єктів, систему багаторівневої авторизації, управління станами (FSM) та механізми забезпечення надійності даних (автоматичне резервне копіювання, захист від мережевого флуду).

По завершенні розробки було отриямано готовий, відмовостійкий програмний продукт. Комплекс успішно пройшов тестування на різних

платформах (Android, iOS, Desktop), підтвердивши високу швидкість відгуку та зручність використання. Впровадження розробленого Telegram-бота дозволяє підприємству централізувати технічну інформацію, скоротити час реагування майстрів на об'єктах та мінімізувати помилки, спричинені людським фактором, що загалом підвищує економічну та операційну ефективність компанії [26].

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Про інформацію: Закон України від 02.10.1992 № 2657-XII. Ред. від 01.01.2024.

2. Таненбаум Е., ван Стеен М. Розподілені системи: принципи та парадигми. — К.: Видавничий дім «SoftPress», 2021. — 912 с.

3. Соммервілл І. Інженерія програмного забезпечення. — 10-те вид. — М.: Вільямс, 2018. — 816 с.

4. Шилдт Г. Python: повне керівництво. — К.: Діалектика, 2022. — 720 с.

5. Лютц М. Вивчаємо Python. — 5-те вид. — СПб.: Символ-Плюс, 2020. — 1160 с.

6. Фаулер М. Рефакторинг. Покращення дизайну існуючого коду. — К.: Діалектика, 2019. — 448 с.

7. Гамма Е., Хелм Р. Прийоми об'єктно-орієнтованого проектування. Патерни проектування. — СПб.: Пітер, 2021. — 368 с.

8. Купер А., Рейман Р. Про інтерфейс. Основи проектування взаємодії. — 4-те вид. — СПб.: Пітер, 2022. — 720 с.

9. Круг С. Не змушуйте мене думати. Здоровий глузд у веб-юзабіліті. — М.: Манн, Іванов і Фербер, 2020. — 256 с.

10. Гілл Т. Розробка ботів для месенджерів. — К.: Видавництво «Техніка», 2021. — 320 с.

11. Марченко О. І. Архітектура інформаційних систем. — К.: КНУ, 2019. — 412 с.

12. Керніган Б., Пайк Р. Практика програмування. — СПб.: Невський Діалект, 2021. — 288 с.

13. Дейт К. Дж. Вступ до систем баз даних. — 8-ме вид. — М.: Вільямс, 2017. — 1328 с.

14. Клементс П. Архітектура програмного забезпечення на практиці. — 3-тє вид. — К.: Діасофт, 2021. — 608 с.

15. Документація Telegram Bot API. URL: https://core.telegram.org/bots/api (дата звернення: 15.04.2026).

16. Документація бібліотеки Aiogram. URL: https://docs.aiogram.dev/ (дата звернення: 12.04.2026).

17. Документація мови програмування Python. URL: https://docs.python.org/3/ (дата звернення: 10.04.2026).

18. Документація SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 18.04.2026).

19. Шнайєр Б. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти на мові С. — М.: Тріумф, 2018. — 816 с.

20. Таненбаум Е. Комп'ютерні мережі. — 6-те вид. — СПб.: Пітер, 2023. — 992 с.

21. Гоффман Е. Безпека веб-застосунків. — К.: МК-Прес, 2020. — 336 с.

22. Мартін Р. Чистий код. Створення, аналіз і рефакторинг. — СПб.: Пітер, 2022. — 464 с.

23. Макконнелл С. Досконалий код. Практичне керівництво з розробки програмного забезпечення. — К.: Діалектика, 2021. — 896 с.

24. Нільсен Я. Ергономіка веб-інтерфейсів. — М.: Вільямс, 2019. — 400 с.

25. Брук Ф. Міфічний людино-місяць. Нариси з системного програмування. — СПб.: Символ-Плюс, 2020. — 304 с.

26. Канер С. Тестування програмного забезпечення. — К.: Діасофт, 2021. — 544 с.

27. Бейзер Б. Тестування чорного ящика. Технології тестування програм та систем. — СПб.: Пітер, 2018. — 320 с.

28. Орлов С. А. Технологія розробки програмного забезпечення. — СПб.: Пітер, 2021. — 608 с.

ДОДАТОК А

```
import os
import sys
import sqlite3
import shutil
import asyncio
import logging
from datetime import datetime, timedelta
from contextlib import contextmanager
from aiogram import Bot, Dispatcher
from aiogram.filters import Command, CommandStart, StateFilter
from aiogram.types import Message, CallbackQuery, ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup, InlineKeyboardButton
from aiogram.fsm.context import FSMContext
from aiogram.fsm.storage.memory import MemoryStorage
#
# ⚙️ БЛОК НАЛАШТУВАНЬ ТА КОНСТАНТ (config.py та constants.py)
#
# --- Основні конфігураційні змінні бота (config) ---
BOT_TOKEN = "6360068248:AAHYzCEAKh4kKMZQirZ7PsF3C-8_BBi-nDc"
DATABASE_PATH = r'houses.db'  # Шлях до бази даних будинків
USERS_DATABASE_PATH = r'users.db'  # Шлях до бази даних користувачів
ADMIN_CHAT_ID = 405597723
ADMIN_USERNAMES = ["R3d1n1", "Blakberg"]
# Координати міста Кривий Ріг (для розрахунків або карт)
KR_COORDINATES = (47.9104, 33.3911)
# Пароль для первинної авторизації
AUTH_PASSWORD = "DataKey99"
PAGE_SIZE = 5
THROTTLE_TIME = 2
STICKER_ID = "CAACAgIAAxkBAAEKsfplSAKqklR1jICM72Kj1tY323MD-QAC4DkAAmSrQUp2q8rxlnYi9TME"
# --- Текстові константи та емодзі для інтерфейсу (constants) ---
RESTART_BUTTON = "🔄 Перезапустить 🔄"
ADD_HOUSE_BUTTON = "🏠 Добавить дом 🏠"
LIST_HOUSES_BUTTON = "📖 Список домов 📖"
SEARCH_HOUSE_BUTTON = "🔍 Найти дом 🔍"
HISTORY_BUTTON = "🕰️ История 🕰️"
EDIT_BUTTON = "🖍️ Редактировать 🖍️"
DELETE_BUTTON = "❌ Удалить ❌"
NO_KEY_TEXT = "Нет ключа"
CURRENT_TEXT = "Текущий"
CURRENT_DESCRIPTION_TEXT = "Текущее описание"
YES_TEXT = "Да"
NO_TEXT = "Нет"
# Емодзі для форматування тексту
EMOJI_HOUSE = "🏠"
```

```
EMOJI_KEY = " 🔑 "
EMOJI_STREET = " 🌳 "
EMOJI_EDIT = " 🖍️ "
EMOJI_DELETE = " 🗑️ "
EMOJI_SUCCESS = " ✅ "
EMOJI_ERROR = " ❌ "
EMOJI_INFO = " 📝 "
EMOJI_USER = " 👤 "
EMOJI_ADMIN = " 👑 "
EMOJI_CALENDAR = " 🗓️ "
EMOJI_LOCATION = " 📍 "
#
==========================================================================
# 📝 БЛОК ЛОГУВАННЯ (logger_config.py)
#
==========================================================================
# Створюємо папку для логів, якщо вона відсутня
if not os.path.exists('logs'):
    os.makedirs('logs')
def setup_logger(name='bot'):
    """Налаштовує та повертає логер для запису подій системи."""
    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)
    if logger.handlers:
        return logger
    formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S'
    )
     # Вивід логів у консоль
    console_handler = logging.StreamHandler(sys.stdout)
    console_handler.setLevel(logging.INFO)
    console_handler.setFormatter(formatter)
    logger.addHandler(console_handler)
        # Збереження логів у файл з розбивкою по днях
    log_filename = f'logs/bot_{datetime.now().strftime("%Y%m%d")}.log'
    file_handler = logging.FileHandler(log_filename, encoding='utf-8')
    file_handler.setLevel(logging.DEBUG)
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    return logger
logger = setup_logger()
#
==========================================================================
# 💾 БЛОК БАЗИ ДАНИХ ТА КОНТЕКСТНИХ МЕНЕДЖЕРІВ (db_context.py,
database.py)
#
==========================================================================
@contextmanager
def get_db_connection(db_path=DATABASE_PATH):
```

```
        """Контекстний менеджер для безпечного підключення до БД (автоматичне закриття)."""
        conn = None
        try:
            conn = sqlite3.connect(db_path, check_same_thread=False)
            cursor = conn.cursor()
            yield conn, cursor
            conn.commit()
        except sqlite3.Error as e:
            if conn:
                conn.rollback()
            raise e
        finally:
            if conn:
                conn.close()
    @contextmanager
    def get_users_db_connection():
        """Спеціальний контекстний менеджер для БД користувачів."""
        with get_db_connection(USERS_DATABASE_PATH) as (conn, cursor):
            yield conn, cursor
    def get_connection():
        """Застарілий метод отримання з'єднання (краще використовувати get_db_connection)."""
        conn = sqlite3.connect(DATABASE_PATH, check_same_thread=False)
        cursor = conn.cursor()
        return conn, cursor
    def execute_query(query, params=()):
        """Виконує SQL-запит та повертає результат."""
        conn, cursor = get_connection()
        cursor.execute(query, params)
        result = cursor.fetchall()
        conn.commit()
        conn.close()
        return result
    def backup_database():
        """Створює резервну копію головної бази даних."""
        try:
            current_time = datetime.now().strftime("%Y%m%d%H%M%S")
            backup_folder = 'backups'
    def get_users_connection_old():
        """Старий метод для отримання підключення до БД користувачів."""
        conn = sqlite3.connect(USERS_DATABASE_PATH, check_same_thread=False)
        cursor = conn.cursor()
        return conn, cursor

    def initialize_auth_table():
        """Ініціалізація таблиці користувачів та міграція стовпця 'is_admin'."""
        conn, cursor = get_users_connection_old()
        try:
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS auth_users (
                    user_id INTEGER PRIMARY KEY,
```

```
                username TEXT,
                telegram_nick TEXT,
                is_admin INTEGER DEFAULT 0
            )
        """)

        try:
            cursor.execute("ALTER TABLE auth_users ADD COLUMN is_admin INTEGER
DEFAULT 0")
        except sqlite3.OperationalError:
            pass  # Стовпець вже існує

        for admin_username in ADMIN_USERNAMES:
            cursor.execute("""
                UPDATE auth_users
                SET is_admin = 1
                WHERE telegram_nick = ? OR username = ?
            """, (admin_username, admin_username))

        conn.commit()
    except sqlite3.Error as e:
        logger.error(f"Помилка ініціалізації таблиці auth_users: {e}")
    finally:
        conn.close()

def is_user_authenticated(user_id):
    """Перевіряє, чи користувач вже є в базі."""
    conn, cursor = get_users_connection_old()
    cursor.execute("SELECT * FROM auth_users WHERE user_id=?", (user_id,))
    result = cursor.fetchone() is not None
    conn.close()
    return result

def is_admin(user_id):
    """Перевіряє права адміністратора для конкретного користувача."""
    conn, cursor = get_users_connection_old()
    try:
        cursor.execute("SELECT is_admin FROM auth_users WHERE user_id=?", (user_id,))
        result = cursor.fetchone()
        if result:
            return bool(result[0])
        cursor.execute("SELECT telegram_nick FROM auth_users WHERE user_id=?", (user_id,))
        user_data = cursor.fetchone()
        if user_data and user_data[0] in ADMIN_USERNAMES:
            return True
        return False
    finally:
        conn.close()

def set_admin(user_id, is_admin_value=True):
    conn, cursor = get_users_connection_old()
```

```
    try:
        cursor.execute("UPDATE auth_users SET is_admin = ? WHERE user_id = ?",
                       (1 if is_admin_value else 0, user_id))
        conn.commit()
        return True
    except sqlite3.Error as e:
        conn.rollback()
        logger.error(f"Помилка призначення адміністратора: {e}")
        return False
    finally:
        conn.close()

def get_user_by_username_or_nick(username_or_nick):
    conn, cursor = get_users_connection_old()
    try:
        cursor.execute("""
            SELECT user_id, username, telegram_nick
            FROM auth_users
            WHERE username = ? OR telegram_nick = ?
        """, (username_or_nick, username_or_nick))
        result = cursor.fetchone()
        return result
    finally:
        conn.close()

def get_username(user_id):
    conn, cursor = get_users_connection_old()
    cursor.execute('SELECT username FROM auth_users WHERE user_id = ?', (user_id,))
    username = cursor.fetchone()
    conn.close()
    return username[0] if username else "Unknown"

def verify_password(password):
    """Перевірка коректності введеного пароля."""
    return password.strip() == AUTH_PASSWORD

def save_user(user_id, username, telegram_nick):
    """Збереження або оновлення даних користувача."""
    initialize_auth_table()
    conn, cursor = get_users_connection_old()
    try:
        cursor.execute("SELECT user_id, is_admin FROM auth_users WHERE user_id=?",
(user_id,))
        existing_user = cursor.fetchone()

        if existing_user:
            is_admin_value = existing_user[1] if len(existing_user) > 1 else 0
            cursor.execute("""
                UPDATE auth_users
                SET username=?, telegram_nick=?, is_admin=?
                WHERE user_id=?
            """, (username, telegram_nick, is_admin_value, user_id))
```

```
        else:
            is_admin_value = 0
            cursor.execute("""
                INSERT INTO auth_users (user_id, username, telegram_nick, is_admin)
                VALUES (?, ?, ?, ?)
            """, (user_id, username, telegram_nick, is_admin_value))
        conn.commit()
    except sqlite3.Error as e:
        conn.rollback()
        logger.error(f"Помилка збереження користувача: {e}")
    finally:
        conn.close()

def update_username(user_id, new_username):
    conn, cursor = get_users_connection_old()
    cursor.execute("UPDATE auth_users SET username=? WHERE user_id=?", (new_username,
user_id))
    conn.commit()
    conn.close()


#
================================================================================
# 📈 БЛОК АКТИВНОСТІ (activity.py)
#
================================================================================

def initialize_user_activity():
    conn, cursor = get_connection()
    if conn is not None:
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS user_activity (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                chat_id INTEGER,
                username TEXT,
                activity_type TEXT,
                activity_detail TEXT,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        """)
        conn.commit()
        conn.close()

def log_user_activity(chat_id, username, activity_type, activity_detail):
    """Логування дій користувача (додавання будинку, фото тощо)."""
    initialize_user_activity()
    conn, cursor = get_connection()
    try:
        current_timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        cursor.execute(
            "INSERT INTO user_activity (chat_id, username, activity_type, activity_detail,
timestamp) VALUES (?, ?, ?, ?, ?)",
```

```
            (chat_id, username, activity_type, activity_detail, current_timestamp)
        )
        conn.commit()

        message = (
            f"🔔 Новая запись активности:\n"
            f"👤 Пользователь: {username}\n"
            f"📋 Тип активности: {activity_type}\n"
            f"📝 Детали: {activity_detail}\n"
            f"🕒 Время: {current_timestamp}"
        )

        if chat_id != ADMIN_CHAT_ID and activity_type != "просмотр дома":
            print(f"Активность для отправки администратору: {message}")

    except sqlite3.Error as e:
        print(f"Помилка логування активності: {e}")
        conn.rollback()
    finally:
        conn.close()

def log_view(user_id, username, address, view_date):
    """Записує факт перегляду інформації про конкретну адресу."""
    conn, cursor = get_connection()
    try:
        cursor.execute("INSERT INTO view_history (user_id, username, address, view_date)
VALUES (?, ?, ?, ?)",
                       (user_id, username, address, view_date))
        conn.commit()
    except sqlite3.Error as e:
        print(f"Помилка логування перегляду: {e}")
        conn.rollback()
    finally:
        conn.close()

def get_activity_stats(offset=0):
    """Отримання статистики дій користувачів за місяць."""
    conn, cursor = get_connection()
    try:
        today = datetime.now()
        first_day_of_this_month = today.replace(day=1)
        target_month_start = first_day_of_this_month -
timedelta(days=first_day_of_this_month.day - 1) - timedelta(weeks=4 * offset)
        next_month_start = target_month_start.replace(day=1) + timedelta(days=32)
        target_month_end = next_month_start.replace(day=1) - timedelta(seconds=1)

        cursor.execute("""
            SELECT username,
                   SUM(CASE WHEN activity_type = 'просмотр дома' THEN 1 ELSE 0 END) AS
view_count,
```

```
            SUM(CASE WHEN activity_type = 'добавил дом' THEN 1 ELSE 0 END) AS
add_count,
            SUM(CASE WHEN activity_type = 'редактирование дома' THEN 1 ELSE 0 END)
AS edit_count,
            SUM(CASE WHEN activity_type = 'Добавление фото' THEN 1 ELSE 0 END) AS
photo_add_count
        FROM user_activity
        WHERE timestamp BETWEEN ? AND ?
        GROUP BY username
    """, (target_month_start, target_month_end))
    stats = cursor.fetchall()
    return stats, target_month_start.strftime('%Y-%m')
  except sqlite3.Error as e:
    print(f"Помилка отримання статистики: {e}")
    return [], datetime.now().strftime('%Y-%m')
  finally:
    conn.close()

def get_month_activity(offset=0):
  conn, cursor = get_connection()
  try:
    today = datetime.now()
    first_day_of_this_month = today.replace(day=1)
    target_month_start = first_day_of_this_month -
timedelta(days=first_day_of_this_month.day - 1) - timedelta(weeks=4 * offset)
    next_month_start = target_month_start.replace(day=1) + timedelta(days=32)
    target_month_end = next_month_start.replace(day=1) - timedelta(seconds=1)

    cursor.execute("""
        SELECT username, COUNT(view_date) AS view_count
        FROM view_history
        WHERE view_date BETWEEN ? AND ?
        GROUP BY username
        ORDER BY view_count DESC
    """, (target_month_start, target_month_end))
    stats = cursor.fetchall()
    return stats, target_month_start.strftime('%Y-%m')
  except sqlite3.Error as e:
    print(f"Помилка отримання переглядів: {e}")
    return [], datetime.now().strftime('%Y-%m')
  finally:
    conn.close()

def get_user_personal_stats(user_id):
  """Збирає особисту статистику по кожному конкретному користувачу."""
  conn, cursor = get_connection()
  try:
    users_conn, users_cursor = get_users_db_connection()
    users_cursor.execute('SELECT username FROM auth_users WHERE user_id = ?',
(user_id,))
    username_result = users_cursor.fetchone()
    username = username_result[0] if username_result else "Unknown"
```

```
        cursor.execute("""
            SELECT COUNT(DISTINCT address) AS unique_addresses,
                   COUNT(*) AS total_views
            FROM view_history
            WHERE user_id = ?
        """, (user_id,))
        view_stats = cursor.fetchone()
        unique_addresses = view_stats[0] if view_stats else 0
        total_views = view_stats[1] if view_stats else 0

        cursor.execute("""
            SELECT COUNT(*) AS add_count
            FROM user_activity
            WHERE chat_id = ? AND activity_type = 'добавил дом'
        """, (user_id,))
        add_result = cursor.fetchone()
        houses_added = add_result[0] if add_result else 0

        cursor.execute("""
            SELECT COUNT(*) AS photo_count
            FROM user_activity
            WHERE chat_id = ? AND activity_type = 'Добавление фото'
        """, (user_id,))
        photo_result = cursor.fetchone()
        photos_added = photo_result[0] if photo_result else 0

        return {
            'username': username,
            'unique_addresses': unique_addresses,
            'total_views': total_views,
            'houses_added': houses_added,
            'photos_added': photos_added
        }
    except sqlite3.Error as e:
        print(f"Помилка збору особистої статистики: {e}")
        return None
    finally:
        conn.close()


#
# ============================================================================
# 🎛️ БЛОК МЕНЮ ТА ІНТЕРФЕЙСУ (menu.py)
#
# ============================================================================

async def show_navigation_menu(message, bot):
    """Створює головне меню бота, залежно від статусу (адмін чи звичайний
користувач)."""
    if is_user_authenticated(message.from_user.id):
        keyboard = []
```

```
    if message.text and (message.text.lower() == '/add' or message.text.lower() == '/s'):
        keyboard.append([KeyboardButton(text='🔄 Перезапустить 🔄')])
    else:
        keyboard.append([
            KeyboardButton(text='🏠 Добавить дом 🏠'),
            KeyboardButton(text='📖 Список домов 📖')
        ])
        keyboard.append([
            KeyboardButton(text='🔍 Найти дом 🔍'),
            KeyboardButton(text='🔎 Найти по описанию 🔎')
        ])
        keyboard.append([
            KeyboardButton(text='🖍️ Редактировать 🖍️'),
            KeyboardButton(text='🕰️ История 🕰️')
        ])
        keyboard.append([KeyboardButton(text='🔑 Найти по ключу 🔑')])
        keyboard.append([
            KeyboardButton(text='📈 Моя статистика 📈'),
            KeyboardButton(text='🏆 Топ адресов 🏆')
        ])
        keyboard.append([KeyboardButton(text='⚙️ Настройки ⚙️')])

        # --- Панель адміністратора ---
        if is_admin(message.from_user.id):
            keyboard.append([KeyboardButton(text='————— 👑 АДМИН —————')])
            keyboard.append([
                KeyboardButton(text='❌ Удалить адрес ❌'),
                KeyboardButton(text='📸 Удалить фото 📸')
            ])
            keyboard.append([
                KeyboardButton(text='💾 База данных 💾'),
                KeyboardButton(text='📁 Активность 📁')
            ])
            keyboard.append([
                KeyboardButton(text='📊 Статистика 📊'),
                KeyboardButton(text='👥 Пользователи 👥')
            ])

        markup = ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
        await bot.send_message(message.chat.id, "Выберите действие:", reply_markup=markup)
    else:
        # Імпорт локальний, оскільки send_welcome знаходиться в іншому файлі
        from handlers import send_welcome
        await send_welcome(message, bot)

async def show_add_menu(message, bot):
    keyboard = [[KeyboardButton(text='🔄 Перезапустить 🔄')]]
    markup = ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
    await bot.send_message(message.chat.id, " 🌳 Введите улицу:", reply_markup=markup)
```

```
async def show_search_menu(message, bot):
    keyboard = [[KeyboardButton(text='🔄 Перезапустить 🔄')]]
    markup = ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
    await bot.send_message(message.chat.id, "Введите улицу для поиска:",
reply_markup=markup)

async def show_admin_menu(message, bot):
    keyboard = [
        [KeyboardButton(text='➕ Повысить пользователя ➕')],
        [KeyboardButton(text='➖ Понизить пользователя ➖')],
        [KeyboardButton(text='🔙 Назад 🔙')]
    ]
    markup = ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
    await bot.send_message(message.chat.id, "👑 Управление администраторами:",
reply_markup=markup)

async def show_settings_menu(message, bot):
    keyboard = [
        [KeyboardButton(text='🖍️ Изменить имя 🖍️')],
        [KeyboardButton(text='❓ Помощь ❓')],
        [KeyboardButton(text='🔙 Назад 🔙')]
    ]
    markup = ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
    await bot.send_message(message.chat.id, "⚙️ Настройки:", reply_markup=markup)


#
================================================================================
# 🖼️ БЛОК РОБОТИ З ФОТОГРАФІЯМИ (photos.py)
#
================================================================================

async def save_house_photo(message: Message, street, house_number, bot):
    """Збереження file_id фотографії в базу та логування цієї події."""
    if message.photo:
        file_id = message.photo[-1].file_id
        conn, cursor = get_connection()
        if conn is None:
            await message.answer("Ошибка: не удалось подключиться к базе данных.")
            return

        try:
            cursor.execute('SELECT 1 FROM houses WHERE street = ? AND house_number = ?',
(street, house_number))
            if cursor.fetchone() is None:
                await message.answer("Ошибка: дом не найден. Сначала добавьте дом.")
                conn.close()
                return
```

```
            cursor.execute(
                'INSERT INTO house_photos (street, house_number, file_id, date_added) VALUES
(?, ?, ?, CURRENT_TIMESTAMP)',
    await delete_photos_command(message, bot, state)

@dp.message(lambda m: m.text and m.text.lower() == '🏆 топ адресов 🏆')
async def popularity_button_handler(message: Message):
    from handlers import send_popularity_reports
    await send_popularity_reports(message, bot)

@dp.message(lambda m: m.text and m.text.lower() == '⚙️ настройки ⚙️')
async def settings_button_handler(message: Message):
    await show_settings_menu(message, bot)

@dp.message(lambda m: m.text and m.text.lower() == '🖍️ изменить имя 🖍️')
async def edit_name_button_handler(message: Message, state: FSMContext):
    from handlers import start_edit_name
    await start_edit_name(message, bot, state)

@dp.message(lambda m: m.text and m.text.lower() == '❓ помощь ❓')
async def help_button_handler(message: Message):
    from handlers import send_help
    await send_help(message, bot)

@dp.message(lambda m: m.text and m.text.lower() == '💾 база данных 💾')
async def database_button_handler(message: Message):
    if not is_admin(message.from_user.id):
        await message.answer("❌ У вас нет прав для выполнения этой команды.")
        await show_navigation_menu(message, bot)
        return
    from handlers import send_database
    await send_database(message, bot)

@dp.message(lambda m: m.text and m.text.lower() == '📁 активность 📁')
async def activity_button_handler(message: Message):
    if not is_admin(message.from_user.id):
        await message.answer("❌ У вас нет прав для выполнения этой команды.")
        await show_navigation_menu(message, bot)
        return
    from handlers import send_activity_database
    await send_activity_database(message, bot)

@dp.message(lambda m: m.text and m.text.lower() == '🔙 назад 🔙')
async def back_button_handler(message: Message):
    await show_navigation_menu(message, bot)

# --- Обробники станів вводу тексту (пошук, імена) ---
@dp.message(lambda m: m.text and m.text.lower() != '🔙 назад 🔙',
StateFilter("waiting_for_description_search"))
async def process_description_search_state(message: Message, state: FSMContext):
```

```
    from handlers import process_description_search
    await process_description_search(message, bot, state)

@dp.message(lambda m: m.text and m.text.lower() != '⬅️ назад ⬅️',
StateFilter("waiting_for_key_search"))
async def process_key_search_state(message: Message, state: FSMContext):
    from handlers import process_key_search
    await process_key_search(message, bot, state)

@dp.message(lambda m: m.text, StateFilter("waiting_for_username"))
async def process_username_state(message: Message, state: FSMContext):
    from handlers import send_welcome_message
    username = message.text.strip()
    telegram_nick = message.from_user.username
    save_user(message.from_user.id, username, telegram_nick)
    await state.clear()
    await send_welcome_message(message, bot)

@dp.message(lambda m: m.text, StateFilter("waiting_for_edit_name"))
async def process_edit_name_state(message: Message, state: FSMContext):
    from handlers import process_edit_name
    await process_edit_name(message, bot, state)

@dp.message(lambda m: m.text, StateFilter("waiting_for_edit_user_name"))
async def process_edit_user_name_state(message: Message, state: FSMContext):
    from handlers import process_edit_user_name
    data = await state.get_data()
    user_id = data.get('edit_user_id')
    if user_id:
        await process_edit_user_name(message, bot, user_id, state)
    else:
        await message.answer("Ошибка: не найден ID пользователя.")
        await state.clear()

@dp.message(lambda m: m.text, StateFilter("waiting_for_address_selection"))
async def process_address_selection_state(message: Message, state: FSMContext):
    from handlers import process_selected_address
    data = await state.get_data()
    addresses = data.get('addresses', [])
    current_page = data.get('current_page', 0)
    await process_selected_address(message, addresses, current_page, bot, state)

@dp.message(lambda m: m.photo, StateFilter("waiting_for_photo"))
async def process_photo_state(message: Message, state: FSMContext):
    data = await state.get_data()
    street = data.get('street')
    house_number = data.get('house_number')
    if street and house_number:
        await save_house_photo(message, street, house_number, bot)
        await message.answer(f"Фото для дома {street} {house_number} успешно сохранено!")
    else:
        await message.answer("Ошибка: не найдены данные адреса.")
```

```
    await state.clear()
    await show_navigation_menu(message, bot)

# --- Обробники Inline-кнопок (Callback Queries) ---
@dp.callback_query(lambda c: c.data and c.data.startswith("add_photo"))
async def add_photo_callback(callback: CallbackQuery, state: FSMContext):
    await handle_add_photo_callback(callback, bot, state)

@dp.callback_query(lambda c: c.data and c.data.startswith("view_photos"))
async def view_photos_callback(callback: CallbackQuery):
    await handle_view_photos_callback(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("view_house"))
async def view_house_callback_handler(callback: CallbackQuery):
    """Обробляє натискання на кнопку перегляду інформації про конкретний будинок."""
    try:
        _, street, house_number = callback.data.split(":")

        query = "SELECT street, house_number, description, change_date, key_number FROM
houses WHERE street = ? AND house_number = ?"
        result = execute_query(query, (street, house_number))

        if result:
            street, house_number, description, change_date, key_number = result[0]

            result_text = f"🏠 <b>{street}, {house_number}</b>\n\n"
            if key_number:
                result_text += "🔑 Ключ на стенде\n\n"
            result_text += f"📝 Описание:\n{description}\n\n"
            result_text += f"🗓️ Дата изменения: {change_date}"

            from utils import generate_google_maps_link
            google_maps_link = generate_google_maps_link(street, house_number)
            result_text += f'\n\n<a href="{google_maps_link}">📍 Посмотреть на карте</a>'

            keyboard = InlineKeyboardMarkup(inline_keyboard=[])

            if key_number:
                key_button = InlineKeyboardButton(text='🔑 Показать ключ',
callback_data=f'key_pressed:{street}:{house_number}')
                keyboard.inline_keyboard.append([key_button])

            add_photo_button = InlineKeyboardButton(text="📷 Добавить фото",
callback_data=f"add_photo:{street}:{house_number}")
            keyboard.inline_keyboard.append([add_photo_button])

            conn, cursor = get_connection()
            cursor.execute('SELECT file_id FROM house_photos WHERE street = ? AND
house_number = ?', (street, house_number))
            photo_data = cursor.fetchall()
            conn.close()
```

```
            if photo_data:
                view_photos_button = InlineKeyboardButton(text="🖼️ Просмотреть фото",
callback_data=f"view_photos:{street}:{house_number}")
                keyboard.inline_keyboard.append([view_photos_button])

            await callback.message.answer(result_text, reply_markup=keyboard)
        else:
            await callback.message.answer("❌ Дом не найден.")
    except Exception as e:
        await callback.message.answer(f"❌ Произошла ошибка: {e}")
    finally:
        await callback.answer()

@dp.callback_query(lambda c: c.data and c.data.startswith('key_pressed'))
async def key_pressed_callback_handler(callback: CallbackQuery):
    from handlers import handle_key_button_pressed
    await handle_key_button_pressed(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("prev_act_"))
async def prev_act_callback_handler(callback: CallbackQuery):
    from handlers import handle_previous_activity_stats
    await handle_previous_activity_stats(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("prev_stats_"))
async def prev_stats_callback_handler(callback: CallbackQuery):
    from handlers import handle_previous_month_stats
    await handle_previous_month_stats(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("promote_user:"))
async def promote_user_callback_handler(callback: CallbackQuery):
    from handlers import handle_promote_user_callback
    await handle_promote_user_callback(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("demote_user:"))
async def demote_user_callback_handler(callback: CallbackQuery):
    from handlers import handle_demote_user_callback
    await handle_demote_user_callback(callback, bot)

@dp.callback_query(lambda c: c.data and c.data.startswith("edit_user_name:"))
async def edit_user_name_callback_handler(callback: CallbackQuery, state: FSMContext):
    from handlers import handle_edit_user_name_callback
    await handle_edit_user_name_callback(callback, bot, state)


def get_bot():
    return bot

# Головна функція для ініціалізації та старту поллінгу (polling)
async def main():
    try:
```

```
        initialize_auth_table()
        logger.info("Бот запущен...")
        await dp.start_polling(bot, allowed_updates=dp.resolve_used_update_types())
    except KeyboardInterrupt:
        logger.info("Бот остановлен пользователем")
    except asyncio.CancelledError:
        logger.info("Операция отменена")
    except Exception as e:
        logger.error(f"Критическая ошибка: {e}", exc_info=True)
    finally:
        try:
        except Exception as e:
            logger.warning(f"Ошибка при остановке polling: {e}")
        finally:
            try:
                await bot.session.close()
            except Exception as e:
                logger.warning(f"Ошибка при закрытии сессии: {e}")

if __name__ == "__main__":
    asyncio.run(main())
```