МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»

Факультет комп'ютерних наук та технологій

Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Олена ГРІНЕНКО
«_____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Методика та програмний продукт на базі Telegram-бота для внутрішніх потреб підприємства.

Виконавець: Ковальчук Володимир Володимирович

Керівник: к. т. н., доцент Борковська Любов Олексіївна

Нормоконтролер: к. т. н., доцент Борковська Любов Олексіївна

Київ 2025

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»
Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олена ГРІНЕНКО
«_____» _____ 2025 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Ковальчука Володимира Володимировича

1. Тема кваліфікаційної роботи: «Методика та програмний продукт на базі Telegram-бота для внутрішніх потреб підприємства.» затверджена наказом ректора від _____

2. Термін виконання проекту: з 29.10.2025 р. по 14.12.2025 р.

3. Вихідні дані до роботи: розробити програмний засіб за допомогою середовища об'єктно-орієнтованого програмування IntelliJ IDEA.

4. Зміст пояснювальної записки:
1. Дослідження предметної області системи з оглядом конкурентів.
2. Аналіз вимог та технологій для розробки подібних програмних продуктів.
3. Структура бази даних та самого програмного продукту.
4. Прототип telegram-боту.
5. Висновки, щодо проведеної роботи.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
1. Таблиця ризиків проекту.
2. Функціональні можливості програмного продукту.
3. ER-діаграма
4. Діаграма варіантів використання
5. Діаграма класів
6. Діаграма послідовності
7. Демонстрація роботи програми.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи	29.09-01.10.2025	Виконано
2.	Ознайомлення з постановкою задачі, вивчення інформаційних джерел та складання плану роботи.	02.10-05.10.2025	Виконано
2.	Підготовка 1 розділу та подання його керівнику	06.10-19.10.2025	Виконано
3.	Підготовка 2 розділу та подання його керівнику	20.10-02.11.2025	Виконано
4.	Підготовка 3 розділу та подання його керівнику	03.12-16.11.2025	Виконано
5.	Підготовка 4 розділу і висновків по роботі та подання їх керівнику	17.11-30.11.2025	Виконано
6.	Загальне редагування пояснювальної записки, графічного матеріалу. Представлення роботи для перевірки на академічну доброчесність. Проходження нормоконтролю.	01.12-07.12.2025	Виконано
7.	Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	08.12-14.12.2025	Виконано
8.	Попередній захист (представлення електронної версії пояснювальної записки, презентації, позитивного відгуку керівника).	08.12-14.12.2025	Виконано
9.	Рецензування кваліфікаційної роботи	15.12-18.12.2025	Виконано
10.	Здача секретарю ЕК пояснювальної записки: електронної версії кваліфікаційної роботи; презентації доповіді; відгуку керівника, рецензії; результату проходження перевірки на плагіат; довідки про успішність, декларації про академічну доброчесність.	15.12-18.12.2025	Виконано
11.	Захист кваліфікаційної роботи перед екзаменаційною комісією	23.12.2025	Виконано

Дата видачі завдання 12.05.2025 р.

Керівник кваліфікаційної роботи:
к. т. н., доцент Любов БОРКОВСЬКА

Завдання прийняв до виконання: Володимир КОВАЛЬЧУК

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Методика та програмний продукт на базі Telegram-бота для внутрішніх потреб підприємства»:95 сторінок, 64 рисунків, 5 таблиць, 5 діаграм, 9 використаних джерел, 0 додатків.

Об'єкт дослідження – процеси організації внутрішніх бізнес-процесів підприємства та засоби їх автоматизації за допомогою інформаційних систем.

Мета кваліфікаційної роботи – розробка методики та програмного продукту у вигляді Telegram-бота, який забезпечує автоматизацію внутрішніх операцій підприємства, інтеграцію з наявними інформаційними ресурсами та оптимізацію взаємодії персоналу.

Методи дослідження – системний аналіз бізнес-процесів, методи проєктування інформаційних систем, UML-моделювання, аналіз та порівняння архітектур телеграм-ботів, методи об'єктно-орієнтованого програмування, прототипування, тестування програмного забезпечення.

Практичне значення роботи полягає у створенні Telegram-бота для внутрішніх потреб підприємства, який дозволяє автоматизувати типові операції: автоматичну генерацію заяв на відпустку або відрядження, пошук та замовлення товарів, друк документів, бронювання місць для парковки, реєстрація заявок на IT-підтримку. Запропоноване рішення може бути використане в будь-яких організаціях, де необхідна оптимізація внутрішніх робочих процесів та зменшення навантаження на адміністративний персонал.

Умови проведення дослідження – розробка та тестування здійснювалися в операційній системі Windows 11. Програмний продукт створено у середовищі IntelliJ IDEA, з використанням мови програмування JavaFX, для бази даних SQLlite.

Ключові слова: ТЕЛЕГРАМ-БОТ, АВТОМАТИЗАЦІЯ ПІДПРИЄМСТВА, ВНУТРІШНІ БІЗНЕС-ПРОЦЕСИ, ПРОГРАМНИЙ ПРОДУКТ, ОБЛІК ТА ЗАМОВЛЕННЯ, ІНФОРМАЦІЙНІ СИСТЕМИ ПІДПРИЄМСТВА, АВТОМАТИЗАЦІЯ РОБОЧИХ ПРОЦЕСІВ.

ABSTRACT

Explanatory note to the qualification work "Methodology and software product based on Telegram-bot for internal needs of the enterprise": 95 pages, 64 figures, 5 tables, 5 diagrams, 9 sources used, 0 appendices.

The object of the research is the processes of organizing internal business processes of the enterprise and means of their automation using information systems.

The purpose of the qualification work is to develop a methodology and software product in the form of a Telegram-bot, which provides automation of internal operations of the enterprise, integration with existing information resources and optimization of personnel interaction.

Research methods are system analysis of business processes, methods of designing information systems, UML modeling, analysis and comparison of Telegram-bot architectures, methods of object-oriented programming, prototyping, software testing.

The practical significance of the work lies in the creation of a Telegram bot for the internal needs of the enterprise, which allows you to automate typical operations: automatic generation of vacation or business trip applications, search and order of goods, printing documents, booking parking spaces, registering applications for IT support. The proposed solution can be used in any organizations where optimization of internal workflows and reduction of the burden on administrative personnel is necessary.

Research conditions - development and testing were carried out in the Windows 11 operating system. The software product was created in the IntelliJ IDEA environment, using the JavaFX programming language, for the SQLlite database.

Keywords: TELEGRAM BOT, ENTERPRISE AUTOMATION, INTERNAL BUSINESS PROCESSES, SOFTWARE PRODUCT, ACCOUNTING

AND ORDERING, ENTERPRISE INFORMATION SYSTEMS, AUTOMATION OF WORK PROCESSES

ЗМІСТ

ВСТУП

Сучасні умови розвитку цифрових технологій вимагають від компаній постійного вдосконалення внутрішніх процесів та автоматизації рутинних завдань. Одним із актуальних інструментів для цього є програмні застосунки у вигляді чат-ботів, які дозволяють значно зменшити навантаження на персонал, забезпечити швидкий доступ до інформації та налагодити ефективну взаємодію всередині організації. Telegram, як одна з найбільш популярних платформ обміну повідомленнями, надає відкритий програмний інтерфейс, що дозволяє розробникам створювати ботів різного рівня складності. Використання цієї платформи є доцільним завдяки її поширеності, гнучкості, простоті інтеграції та високому рівню безпеки.

У межах даної роботи розглядається процес проектування та розробки Telegram-бота для внутрішніх потреб компанії. На відміну від комерційних чат-ботів, орієнтованих на клієнтів, цей бот спрямований на забезпечення внутрішньої комунікації, обліку даних і підтримки управлінських процесів.

Метою роботи є розробити методику та програмний продукт у вигляді Telegram-бота, призначеного для автоматизації внутрішніх процесів підприємства.

Об’єкт дослідження – процеси організації та автоматизації внутрішніх бізнес-операцій підприємства.

Предмет дослідження – методи та програмні засоби, які забезпечують автоматизовану взаємодію персоналу через Telegram-бот.

Методи дослідження. У роботі застосовано методи системного аналізу, UML-моделювання, методи об’єктно-орієнтованого проектування, прототипування та тестування програмного забезпечення.

Практичне значення полягає у створенні Telegram-бота для внутрішніх потреб підприємства, який дозволяє автоматизувати типові операції: автоматичну генерацію заяв на відпустку або відрядження, пошук та замовлення товарів, друк документів, бронювання місць для парковки, реєстрація заявок на IT-підтримку. Запропоноване рішення може бути

використане в будь-яких організаціях, де необхідна оптимізація внутрішніх робочих процесів та зменшення навантаження на адміністративний персонал.

За матеріалами дослідження опубліковано 1 тез доповідей, на тему: «Digitalization of Internal Business Processes Using Intelligent Chatbots: Development Methodology and Social Impact» на науковій конференції IT&IS-2025 .

Таким чином, робота має на меті створення ефективного, масштабованого та безпечного рішення, яке відповідатиме сучасним вимогам до інформаційних систем та забезпечить надійний інструмент для автоматизації внутрішніх бізнес-процесів.

РОЗДІЛ 1

ІНІЦІАЦІЯ

1.1. Дослідження предметної області

Відповідно до сучасних тенденцій та практик ведення бізнесу ефективність діяльності підприємства залежить від рівня автоматизації внутрішніх бізнес-процесів та швидкості комунікацій між працівниками. Зростання об'єму інформації, кількості завдань та обов'язків збільшують навантаження на працівників компанії, що призводить до емоційного вигорання і великого навантаження. Зменшення навантаження на працівників та спрощення робочих процесів потребує використання зручних цифрових інструментів, що дозволяють мінімізувати ручну працю та зменшити навантаження на персонал. Одним із сучасних тенденцій цифровізації бізнесу є впровадження інформаційних систем на базі чат-ботів. Використання ботів у робочому середовищі дає такі можливість, як:

- ✓ Спростити оперативну комунікацію між співробітниками;
- ✓ Створити централізований канал обміну даними;
- ✓ Автоматизувати рутинні процеси;
- ✓ Підвищити рівень дисципліни та контроль виконання завдань;
- ✓ Інтегруватися з іншими внутрішніми системами (CRM, ERP, кадровий облік).

Сучасним та зручним інструментом є Telegram. Це популярний застосунок, що надає відкритий і добре задокументований API для створення чат-ботів та має такі переваги як:

- ✓ кросплатформність;
- ✓ висока швидкість передачі даних;
- ✓ підтримка роботи з базами даних через API;
- ✓ можливість створення інтерактивних меню, кнопок, inline-запитів;
- ✓ відсутність вартості за користування платформою;
- ✓ широкі можливості інтеграції з бізнес-процесами.

Для підприємства Telegram-бот може прекрасно слугувати внутрішньою інформаційною системою. Основні сценарії використання:

- ✓ інформування персоналу;
- ✓ збір та обробка даних;
- ✓ управління завданнями;
- ✓ довідкові сервіси;
- ✓ інтеграція з корпоративними сервісами.

У рамках цього дослідження предметна область буде реалізована через Telegram-бота з адміністративною панеллю на JavaFX.

Telegram-бот – забезпечує взаємодію користувачів з системою. Він приймає повідомлення, обробляє команди та надсилає відповіді.

Адміністративна панель – використовується IT-відділом для моніторингу запитів, управління внутрішньою інформацією, контролю доступу співробітників.

База даних – зберігає інформацію про співробітників, заявки, завдання, внутрішні документи.

Дослідження предметної області вказує нам, що створення корпоративного Telegram-бота дозволить підприємству отримати універсальний та економічно доцільний інструмент внутрішньої комунікації, який поєднує всі необхідні фактори як: простоту використання, зручність мобільного доступу та наявність зручного інтерфейсу для адміністрування ботом.

1.2. Визначення контексту проекту розробки

Розробка чат-бота для внутрішнього середовища компанії зумовлена потребою в оптимізації комунікації між співробітниками, а також у автоматизації робочих процесів, та спрощення рутинних справ. У сучасних умовах більшість організацій накопичує значний обсяг правил і внутрішньої документації Усі ці дані часто зберігаються в різних системах, що ускладнює їх пошук.

Запровадження чат-бота дозволяє централізувати цей процес і створити інструмент швидкої взаємодії з внутрішніми ресурсами компанії. У бізнес-контексті він виступає невід'ємним помічником, який знімає навантаження з менеджерів і фахівців технічної підтримки, допомагаючи працівникам самостійно отримати відповіді на поширені питання. У перспективі це сприяє зменшенню кількості рутинних завдань, економії робочого часу та підвищенню загальної продуктивності компанії.

З технічного боку чат-бот реалізується за допомогою JavaFX, що надає широкі можливості для створення сучасного графічного інтерфейсу та забезпечує зручний додаток для користувачів. Застосунок можна інтегрувати з внутрішніми базами даних і корпоративними сервісами, що дозволить у реальному часі надавати актуальну інформацію. Такий підхід забезпечує гнучкість у масштабуванні системи, адже бот може поступово доповнюватися новими модулями, що розширюватимуть його функціональність.

Основними користувачами чат-бота є співробітники компанії, які отримують можливість швидко взаємодіяти з внутрішніми системами. Це можуть бути як нові працівники, які ще не орієнтуються у внутрішніх процесах і потребують допомоги, так і досвідчені спеціалісти. Крім цього, чат-ботом користуються й адміністратори системи, які здійснюють його налаштування, оновлюють базу знань та інтегрують нові функції.

Важливо визначити межі розроблюваної системи. Чат-бот працює виключно у внутрішньому середовищі компанії та не призначений для зовнішніх клієнтів. Сам бот не виступає заміною інформаційних систем компанії, а є додатковим інструментом, який полегшує роботу працівників і дозволяє співробітникам працювати ефективніше. Його завданням є не створення нових інформаційних потоків, а оптимізація вже існуючих.

Окремо слід відзначити взаємодію чат-бота з іншими системами. Він може підключатися до баз даних для пошуку довідкової інформації, інтегруватися з HR-системами для уточнення питань щодо відпусток або лікарняних, працювати з системами документообігу для надання швидкого

доступу до потрібних файлів і форм, а також взаємодіяти з інструментами технічної підтримки. Це створює єдине інформаційне середовище, де співробітники отримують можливість швидко та безпосередньо працювати з даними, які потрібні для виконання їхніх завдань.

Отже контекстом розробки чат-бота є створення внутрішнього цифрового інструменту, який полегшує робочі процеси співробітників компанії, зменшує навантаження на адміністраторів і водночас сприяє ефективнішій організації внутрішніх бізнес-процесів.

1.3. Техніко-економічне обґрунтування доцільності розробки

В великих та малих компаніях працівники щодня стикаються з повторюваними запитаннями: де знайти потрібний документ, як правильно оформити відпустку, до кого звернутися з технічними проблемами, чи буде парковочне місце біля роботи та де мені знайти вільний принтер. Витрачаючи час на пошук цієї інформації або на звернення до колег, працівники знижують власну продуктивність. Особливо це відчутно у великих компаніях, де інформаційні потоки розгалужені й зберігаються у різних місцях. Створення внутрішнього чат-бота допомагає вирішити ці проблеми. Бот стає універсальним «помічником» для швидкого доступу до довідкових матеріалів та автоматизованих відповідей, що зменшить навантаження на адміністративний та технічний персонал, які раніше витрачали час на пояснення типових речей, і дозволить співробітникам отримувати відповіді одразу, без зайвих затримок.

З економічної точки зору розробка чат-бота дозволяє компанії зекономити час, який співробітники витрачають на зайві рухи під час роботи, що призведе до збільшення продуктивності працівника. Коли мова йде не про одну людину а сотні людей, які збережуть час на рутинні завдання перетворюються на значний приріст продуктивності та скорочення витрат.

З технічної точки зору використання JavaFX робить систему доступною для впровадження без великих інвестицій у нове обладнання чи

складні програмні комплекси. Чат-бот може бути встановлений на робочих місцях співробітників і поступово вдосконалюватися. Він не потребує значних витрат на підтримку, оскільки адміністратори лише час від часу оновлюють його базу знань і додають нові функції.

Отже, доцільність розробки пояснюється тим, що чат-бот допомагає економити час співробітників і зменшувати рутину, що призводить до приросту продуктивності працівників компанії та зменшення навантаження.

1.4. Формування цілей

1.4.1. Головна мета проекту

Головною метою даного проекту є розробка внутрішньої інформаційної системи у вигляді чат-бота на основі JavaFX, який буде використовуватись в середині компанії та слугувати невід'ємним помічником працівників. Його основне призначення є автоматизація внутрішніх бізнес-процесів, спрощення щоденних викликів працівників, зниження навантаження на адміністраторів та створення невід'ємного помічника для щоденної роботи працівників. Чат-бот має забезпечити співробітникам швидкі відповіді на стандартні питання (графік роботи, доступ до документів, внутрішні регламенти, контакти відповідальних осіб), а також інтегруватися з існуючими внутрішніми системами, такими як HR-модулі чи база документів. У перспективі він стане «цифровим асистентом» працівників, що спрощує виконання рутинних завдань і дозволяє зосередитися на ключових робочих процесах.

1.4.2. Основні цілі

Основними цілями проекту можна сформулювати так:

Автоматизація обробки типових внутрішніх запитів. Створення системи, яка здатна швидко надавати співробітникам відповіді на стандартні питання – наприклад, як оформити відпустку, до кого звернутися з технічними проблемами, які документи потрібні для звіту тощо.

Оптимізація бізнес-процесів компанії. Зменшення навантаження на

відділи адміністративної підтримки, HR та технічної допомоги. Це дозволяє спеціалістам концентруватися на складних завданнях, а співробітники отримують базову інформацію самостійно через чат-бота.

Підтримка внутрішньої комунікації. Чат-бот може інформувати працівників про важливі події компанії.

Збір і збереження даних. Система зберігатиме історію запитів та відгуків, що дозволить аналізувати найчастіші запити співробітників і постійно оновлюватись відповідно до відгуків, щоб ставати більш досконалим помічником.

Гнучкість і масштабованість. Проект передбачає можливість поступового розширення функціоналу чат-бота: інтеграція з системами документообігу, автоматизація внутрішніх погоджень, підключення до корпоративної пошти тощо.

Простота використання. Система має бути зручною та інтуїтивною, щоб будь-який співробітник міг швидко навчитися користуватися чат-ботом без додаткових інструкцій.

1.4.3. SMART-цілі

Сформуємо цілі проекту за принципом SMART, адаптованим під задачі які мають вирішуватись:

S (Specific): створити чат-бота для автоматизації внутрішніх запитів співробітників і підтримки внутрішньої комунікації.

M (Measurable): автоматизувати щонайменше 50% стандартних звернень до HR, адміністрації чи IT-відділу протягом перших 3 місяців після запуску.

A (Achievable): розробка можлива у межах 2–3 місяців роботи невеликої команди (1–2 розробники, 1 аналітик).

R (Relevant): зниження навантаження на адміністрацію та підвищення продуктивності співробітників завдяки автоматизації робочих процесів.

T (Time-bound): запуск системи у тестову експлуатацію протягом 3 місяців, перехід у промислову експлуатацію – до 6 місяців.

1.5. Визначення ключових стейкхолдерів проекту

Щоб чат-бот дійсно став корисним інструментом для компанії, важливо заздалегідь визначити всіх зацікавлених осіб, які впливають на його створення, розвиток і подальше використання. Це допоможе краще зрозуміти очікування, інтереси та рівень зацікавленості кожного учасника.

1. Замовник

Керівництво виступає головним замовником, адже саме воно визначає головні цілі проекту і виділяє матеріальні ресурси для успішної реалізації проекту. Замовник особливо зацікавлений в розробці, адже йому необхідно щоб чат-бот допоміг знизити витрати на внутрішні процеси та підвищив продуктивність роботу працівників, що значно підвисить дохід. Замовник очікує від проекту, що система має бути вигідною, безпечною й готовою до масштабування, якщо в майбутньому виникне потреба додати нові функції.

2. Користувачі

Користувачами виступають в нашій ситуації самі працівники, для яких в більшій мірі і створюється чат-бот, які щодня будуть взаємодіяти з ботом. Користувачам важливо, щоб інструмент був простим і зрозумілим: щоб можна було швидко знайти потрібну інформацію, отримати нагадування чи автоматизувати рутинні дії, такі як забронювати парковку, знайти вільний принтер, подати вірно написану заяву на відпустку. Їхні очікування: інтуїтивний інтерфейс, стабільна робота і можливість отримати відповіді без зайвого звернення до колег.

3. Команда розробки

Розробники це команда яка безпосередньо створює чат-бота. Сюди входять самі програмісти, тестувальники й аналітики. Для них головним є чітко сформульовані вимоги, вчасні рішення від замовника та реалістичні строки. Команда розробки хоче бачити зрозумілі задачі для створення продукту який буде реалізовувати і відповідати всі очікуванням. Немало важливим є саме фінансування, адже команда має бути впевнена в тому, що замовник буде надавати фінансові ресурси які були закладені на створення

проекту

4. IT-відділ компанії

IT-фахівці відповідають за технічну підтримку інфраструктури та інтеграцію чат-бота з існуючими корпоративними системами. Вони зацікавлені в тому, щоб бот працював безпечно, відповідав внутрішнім IT-стандартам і був керованим. Очікування IT-відділу: доступна документація, можливість адміністрування та масштабування системи.

5. Постачальники технологій

Даним стекхолдером виступають компанії чи сервіси, які надають додаткові інструменти такі як хостинг, резервне копіювання чи аналітичні сервіси. Вони хочуть чітких вимог і своєчасної оплати за свої послуги.

1.6. Аналіз і опис цільової аудиторії проекту

Щоб чат-бот дійсно приносив користь і став зручним інструментом у щоденній роботі, необхідно розібратися, хто саме буде ним користуватися та які потреби має ця аудиторія. Важливо проаналізувати потреби кожної групи користувачів, що вони хочуть бачити в створеним програмному продукті, для яких задач він буде використовуватись та на які процеси потрібно акцентувати увагу при створенні проекту. Це дозволяє зробити систему не абстрактною, а максимально адаптованою до реальних умов роботи всередині компанії.

1. Співробітники компанії.

Для даного проекту є найважливішою групою користувачів саме працівники компанії, так як саме вони найбільше будуть ним користуватись. Щодня дана група працює з значною кількістю рутинних завдань: пошук товарів, уточнення інформації про колег, надсилання запитів у бухгалтерію чи кадровий відділ, погодження звітів або документів. На всі ці дії витрачається багато часу, і часто доводиться звертатися до інших працівників або писати листи.

Провівши аналіз можна виділити такі основні потреби: швидкий доступ до внутрішньої інформації, можливість автоматизувати прості запити,

автоматизувати документообіг.

Очікування: чат-бот має бути простим, зрозумілим навіть для тих, хто не дуже «дружить» з технологіями; він має працювати швидко, без зависань, і бути доступним з таких девайсів як ноутбук, планшет, ПК, та телефон.

2. Керівний склад та менеджери

Ця група користувачів має інші потреби. Для керівників і менеджерів важливі швидкість доступу до даних і можливість оперативно знайти контакти необхідно підлеглого. Даній групі необхідно бачити роботу IT- відділу та відгуки про даного помічника.

Дана група має такі потреби: короткі аналітичні звіти прямо у чаті, швидко отримати контактні дані необхідного підлеглого.

Очікування: чат-бот повинен бути максимально надійним, працювати без перебоїв, інтегруватися з існуючими корпоративними системами і надавати структуровану інформацію у зрозумілому вигляді.

3. Нові співробітники та стажери

Для новачків у компанії чат-бот стане невід'ємним помічником у адаптації у новому робочому середовищі. Часто адаптація нових працівників займає чимало часу, адже потрібно знайти правила та інструкції компанії.

Потребами даної групи є: швидке ознайомлення з корпоративними політиками, доступ до бази правил та інструкцій, можливість знайти контакти керівників і колег.

Очікування: покрокові підказки, простий інтерфейс, зрозуміла структура меню. Ідеально, якщо бот зможе відповідати на типові питання новачків.

4. IT-відділ та технічні адміністратори

IT-фахівці не є менш зацікавленими користувачами. Для них чат-бот слугує як прилад для отримання заявок на допогу, отримання пропозицій для покращення даного програмного застосунку, а головною задачою це є технічна підтримка та подальше доопрацювання проекту.

Потреби: зручні інструменти моніторингу, отримання повідомлень на

запит на допогу, зручний інтерфейс для адміністрування та підтримки чату.

Очікування: система має бути гнучкою у налаштуванні, масштабованою, а також забезпечувати необхідний інтерфейс для адміністрування чату.

Узагальнений портрет користувача

Вік: від 20 до 55 років.

Досвід роботи з технологіями: більшість користувачів маю навички та досвід з користуванням месенджерами та офісними програмами.

Пристрої доступу: переважно смартфони на основі операційнії систем iOS, Android, рідше — десктопна версія Telegram.

Основна мотивація: економія часу, зручність, швидке отримання відповіді без потреби писати листи чи звертатись до друзів\колег.

Фактори успіху: інтуїтивний інтерфейс, мінімальна кількість дій для доступу до інформації, швидка реакція системи, надійний захист даних.

1.7. Проведення аналізу основних конкурентів

Для розуміння ситуації яка на разі є на ринку подібних програмних продуктів, їх переваг та недоліків, та фішок якими вони знаходять нових користувачів проведемо аналіз конкурентів. Аналіз конкурентів надасть можливість створити унікальні функції і при розробці не повторити типових помилок. Найбільшу увагу приділим увагу інструментам які вже використовуються для корпоративних комунікацій та автоматизації бізнес-процесів.

1. Slack

Skack це хмарна платформа яка слугує для корпоративного використання малими та великими компаніями. В Google Play даний програмний продукт має більше 10млн. завантажень та більше 150тис. відгуків, більшість з яких є позитивними, що вказує на те що програмний продукт гарно працює та має великий попит.

При аналізі даної хмарної платформи можна

визначити такі сильні сторони:

- ✓ Великий вибір інтеграцій, таких як CRM, таск-трекери, хмарні сервіси.
- ✓ Зручний пошук по історії повідомлень, що є досить корисним при великих обговореннях та дискусіях.
- ✓ Розвинена екосистема ботів і додатків.

Але нажаль в нашому світі нічого немає досконалого то при аналізі було виявлено такі недоліки, як:

- ✓ Висока вартість підписок для великої кількості користувачів, що не дає можливість використовувати їх всім охочим, адже деякі компанії не готові витрачати великі фінансові ресурси для внутрішньої комунікації працівників.
- ✓ Необхідність додаткового навчання персоналу, для досвідчених користувачів це не є обов'язковим, а для новачків потрібно проводити тренінги у використанні даної хмарної платформи, що знов несе витрачання фінансових ресурсів так і зайвого часу.
- ✓ Немає українськомовного інтерфейсу, що для людей які не володіють англійською мовою чи володіють нею на низькому рівні приведуть до не зручності користування.

2. Microsoft Teams

Microsoft Teams це корпоративна платформа для чату, відеоконференцій та інтеграції з Microsoft 365, яка має значну популярність у навчальних закладах України, в великих та малих компаніях по всьому світі. В Google Play даний програмний продукт має більше 500млн. завантажень та більше 8млн. відгуків, більшість з яких є позитивними, що вказує на якість створеного програмного продукту який значно користується попитом у всьому світі.

Проаналізувавши дану платформу та спираючись на досвід користування нею, можу визначити такі сильні сторони:

- ✓ Інтеграція з пакетами Office 365 і SharePoint, що є великим плюсом

для тих компаній в яких більшість роботи пов’язані з програмними продуктами від Microsoft.

✓ Розширені можливості для відеозв’язку, що є значними перевагами для проведення конференцій та нарад .

✓ Вбудовані засоби безпеки та управління правами доступу, що дає можливість безпечно передавати корпоративні документи без остереження виливу до зловмисників.

Але також було виділено на мою суб’єктивну думку такі слабкі сторони:

✓ Високі ліцензійні витрати, що може викликати відсторонення від даної платформи малих компаній та тих компаній які не маю зайвих ресурсів для комунікації працівників.

✓ Залежність від інфраструктури Microsoft.

3.Google Workspace

Google Workspace є хмарною системою для корпоративного спілкування та співпраці та має інтеграцію з самим Gmail, GoogleDrive, Calendar. Проаналізувавши інтернет було знайдено таку інформацію що на кінець 2023 року 1.8млн. людей використовували дану систему, що є значним показником у кількості користувачів які обрали дану систему, хоч поступається в певній мірі своїм конкурентам.

При проведенні аналізу даної хмарної системи можна визначити такі переваги як:

✓ Простота у використанні, яка зумовлена тип що є можливість користуватись хмарною системою з мобільних пристроїв.

✓ Потужні можливості спільної роботи з документами, яка реалізована в більшій мірі за допомогою інтеграції з Google Docs та Google Drive.

✓ Гнучкі інструменти керування користувачами.

Але окрім сильних сторін було визначено й слабкі сторони, такі як:

✓ Необхідність оплати підписок для всіх користувачів, що є значним мінусом бо не всі компанії мають фінансові ресурси для забезпечення зручної комунікації в компанії.

✓ Обмежена кастомізація ботів під специфічні внутрішні процеси, що не дає змогу створювати таких ботів які були б корисними під певні процеси.

✓ Залежність від стабільного інтернету, так Google Workspace це є хмарною системою, що в разі слабкого інтернет-з'єднання може обмежити можливість користування.

4. Власні корпоративні боти в різних месенджерах

Так як корпоративні чат-боти значно набирають популярності в компаніях, то попит на них тільки росте. На жаль корпоративними чат-ботами я не користувався, але було знайдено людину яка надала особистий відгук про чат-бот, який використовується в її компанії, на надали таку інформацію:

Що сильними сторонами є:

✓ Інтуїтивний інтерфейс що значно спрощує опанування даного продукту.

✓ Безліч можливостей для розвитку та адаптації під потреби які постійно змінюються, що є значним плюсом так як значно менше ресурсів витрачається на доопрацювання наявного програмного продукту чим на створення нового.

✓ Також в їхньому боті було реалізовано така функції як: «Погода», хоч здавалось би така дрібниця, але на думку користувачів це значно спрощує ранок, бо не доводиться забивати великий запит, а достатньо натиснути 3 рази на екран і погода в твоєму місті вже у тебе.

Слабкими сторонами чат ботів є:

✓ Обмежені можливості інтеграції з системами які вже наявні в компанії, що може спричинювати значні незручності.

✓ Менше інструментів для адміністрування, що обмежує можливості адміністраторів та інших людей які займаються адмініструванням чат-боту.

Порівняльний аналіз конкурентів (Таблиця 1)

Таблиця 1

Порівняльний аналіз конкурентів

Критерій	**Slack**	**Microsoft Teams**	**Google Chat**	**Telegram-бот (проєкт)**
Вартість впровадження	Висока	Висока	Середня	Низька (безкоштовний месенджер)
Простота інтеграції	Висока	Середня	Середня	Висока (гнучкі API)
Мобільність	Висока	Висока	Висока	Висока
Гнучкість налаштувань	Середня	Середня	Обмежена	Дуже висока (повна кастомізація під потреби підприємства)
Потреба у навчанні	Середня	Висока	Низька	Низька

В результаті проведення аналізу конкурентів можна сказати, що:

Основною конкурентною перевагою Telegram-бота є його безкоштовність, адже Telegram є безкоштовним месенджером, простота впровадження та використання, а також великі можливості API для реалізації будь якого функціоналу в залежності від потреб замовника. Ключовими відмінностями від можливих аналогів на ринку є те, що Telegram-бот не потребує дорогих підписок і не вимагає додаткового програмного забезпечення, що зменшує зайві фінансові витрати компанії. Ризиками є потреба у власній службі підтримці яка безпосередньо в майбутньому буде масштабувати бот та впроваджувати новий функціонал відповідно до вимог

користувачів та безпеки. Сама ж розробка особистого корпоративного Telegram-боту дозволить компанії отримати власний засіб корпоративного спілкування з безліччю корисних функцій, при том заощаджуючи фінансові ресурси в порівнянні з використанням закордонних ресурсів.

1.8. Формування ризиків проекту та оцінки їх ймовірностей

Аналіз ризиків має важливу роль в процесі створення програмного продукту, адже надає можливість заздалегідь виявити можливі перепони які можуть здійснити негативний вплив на розробку боту і його впровадження для внутрішніх потреб підприємства. Для кращого розуміння кожного ризику, ймовірності його впливу та наслідків та стратегії реагування на нього створимо таблицю(Таблиця 2), яка матиме такі стовбці:

1. Ризик(Сама назва ризику);
2. Ймовірність виникнення (Позначається як: Низька, Середня, Висока);
3. Вплив на проект (Позначається як: Низький, Середній, Високий);
4. Опис і можливі наслідки (Яка містить короткий опис та наслідки які може спричинити даний ризик);
5. Стратегії реагування (запобігання, мінімізація, план дій у разі настання).

Таблиця 2

Основні ризики проекту

№	Ризик	Ймовірність	Вплив	Опис і можливі наслідки	Стратегії реагування
1	Технічні збої або проблеми з Telegram API	Середня	Високий	Нестабільність роботи або зміни в API можуть спричинити	Постійний моніторинг змін API, резервне копіювання коду, план швидкого

				зупинку функцій бота.	оновлення.
2	Витік або несанкціонований доступ до даних	Середня	Високий	Можливі фінансові та репутаційні збитки, порушення законодавства (GDPR, українські норми).	Використання шифрування, багаторівнева авторизація, регулярні аудити безпеки.
3	Затримки у розробці через недостатнє планування	Середня	Середній	Перевищення строків впровадження, збільшення витрат.	Детальне планування етапів, гнучке управління проєктом (Agile/Scrum), буфер часу.
4	Низький рівень залучення кінцевих користувачів	Середня	Середній	Недостатнє використання бота після впровадження, низька окупність.	Проведення навчання персоналу, збирання зворотного зв’язку, адаптація функціоналу.
5	Нестабільне інтернет-з’єднання у співробітників	Низька	Середній	Перешкоди в доступі до сервісу в окремих підрозділах.	Оптимізація роботи бота для низькошвидкісних мереж, можливість кешування важливих даних.
6	Зміни вимог замовника в процесі розробки	Середня	Середній	Переробка коду, збільшення бюджету й	Використання гнучких методологій (Agile), регулярні

				строків.	зустрічі з замовником, фіксація вимог у документації.
7	Недостатня компетенція команди розробників	Низька	Високий	Помилки в архітектурі, складнощі у масштабуванні.	Попередній відбір кваліфікованих фахівців, навчання, технічні рев'ю коду.
8	Правові зміни у сфері захисту персональних даних	Низька	Середній	Необхідність термінового доопрацювання системи для відповідності новим нормам.	Постійний моніторинг законодавства, юридичний супровід проекту.

Якщо проаналізувати створену таблицю то ризики можна розбити умовно на 3 групи, такі як критичні, помірні та низькі, до критичних ризиків відносяться технічні збої Telegram API та можливий витік даних. На дану групу ризиків слід звернути найбільшу увагу, адже витік даних не припустимий, тож їх необхідно контролювати постійно та впроваджувати шифрування та план аварійного відновлення.

До помірних ризиків віднесемо затримку у розробці, зміни вимог замовника, низька активність користувачів. Для зниження можливості їх виникнення слід впроваджувати гнучке управління ботом та часто збирати відгуки користувачів для кращого розуміння що користувачам не подобається і на що слід акцентувати увагу в подальшому розвитку бота.

До низьких ризиків віднесемо правові зміни та проблеми з інтернет-з'єднанням, проте вони вимагають періодичного моніторингу та вчасного реагування.

1.9. Опис концепції програмного забезпечення як інформаційної системи

Концепція програмного забезпечення Telegram-бота, який

розробляється для внутрішніх потреб підприємства. Система створюється з метою автоматизації робочих процесів тож повинна забезпечити безпечну та зручну комунікацію працівників та автоматизувати рутинні процеси.

1. Загальна ідея

Дане програмне забезпечення реалізується у вигляді інформаційної системи. Telegram-бот інтегрується з базою даних яка містить в собі вся необхідна інформація для успішного функціонування самого бота так і для адміністративною панеллю.

Система працює у хмарному середовищі, як і сам Telegram в якому і буде функціонувати бот, що надає можливість в подальшому масштабуватись і дає можливість отримати до неї доступ будь-звідки . Комунікація здійснюється через Telegram, який має значний попит в Україні, тож процес впровадження даного програмного продукту не є складним.

2. Функціональні можливості

Основні функції Telegram-бота як інформаційної системи:

Комунікаційний модуль: надсилання правил та розпоряджень, надсилання зворотного зв’язку про бот та запити на допомогу.

Довідковий модуль: швидкий доступ до контактів працівників компанії, внутрішніх інструкцій та правил.

Сервісний модуль: автоматизація рутинних завдань, таких як заявки на відпустку, замовлення товарів.

Додаткові функції: перегляд погоди на вулиці, бронювання парко місця на території компанії, перегляд днів народжень.

3. Архітектурна модель

Клієнтська частина: мобільний та десктопний додаток Telegram, яким користуються співробітники.

Серверна частина:

Telegram Bot API — обробка вхідних та вихідних повідомлень.

Прикладний сервер, що реалізована за допомогою мови програмування JavaFX— реалізація бізнес-логіки, обробка запитів, авторизація користувачів.

База даних, що створена за допомогою мови програмування MySQL — зберігання користувацьких профілів, історії запитів, документації, списків замовлень.

Інтеграційний шар — конектори до корпоративних систем.

Хмарна інфраструктура: використання надійного хостингу Azure для забезпечення безперервності роботи боту.

4. Інформаційні потоки

Вхідні дані: запити користувачів, заявки на замовлення товарів, оновлення з панелі адміністрування, звіти працівників.

Обробка: бізнес-логіка сервера, що відповідає за аутентифікацію, перевірка прав доступу та формування відповідей.

Вихідні дані: повідомлення бота, сформовані заявки на відпустку, заявки на замовлення товарів.

5. Безпека та захист даних

- ✓ Використання шифрування TLS при обміні даними.
- ✓ Авторизація за доступом, користувач може користуватись ботом виключно якщо його номер телефону доданий в систему адміністратором.
- ✓ Розмежування прав доступу залежно від ролей.
- ✓ Регулярне резервне копіювання.

6. Переваги концепції

Низький поріг входу: немає необхідності встановлювати незнайомі програмні застосунки та витрачати час на їх опанування так як використовується знайомий месенджер.

Гнучкість: можливість швидкої адаптації до функціональних потреб користувачів.

Масштабованість: підтримка необмеженої кількості користувачів без зайвої трати фінансових ресурсів.

Швидке впровадження: мінімальні терміни розгортання порівняно з цілими програмними застосунками.

1.10. Проведення обстеження та моделювання IT-потреб предметної області

В процесі обстеження предметної області було виявлено, що в багатьох компаніях наразі комунікація в компанії та доступ до правил і інструкцій не завжди організовані ефективно та зручно. Працівникам доводиться витрачати багато часу на пошук необхідної інформації для роботи, пошук місця для друку паперів та як вірно оформлювати заяви на відпустку, що призводить до втрати ефективності працівника. Даний фактор спричинює зменшення продуктивності підприємства та тягне за собою спад продуктивності.

Також існуючі системи, якими користуються співробітники, мають певні недоліки:

- ✓ відсутність єдиної точки доступу до корпоративної інформації;
- ✓ складні або незручні інтерфейси, що ускладнюють використання;
- ✓ проблеми з актуальністю даних та синхронізацією між різними внутрішніми сервісами;
- ✓ залежність від «людського фактора», коли для вирішення простих питань потрібна участь інших працівників.

На базі цього обстеження було сформовано наступні IT-потреби:

Єдина інформаційна система: створення внутрішнього чат-бота, який стане універсальним інструментом для пошуку та отримання необхідної інформації.

Підвищення стабільності та надійності доступу до даних: чат-бот має працювати швидко та безпечно.

Зручний інтерфейс: реалізація інтуїтивно зрозумілого текстового чату, який можна інтегрувати у звичні для співробітників корпоративні месенджери або внутрішній портал.

Автоматизація рутинних завдань: забезпечення можливості надання інструкцій, пошуку контактів, створення звітів, створення заявок на допомогу, замовлення товарів.

Збір статистики та аналітика: впровадження механізмів для аналізу

запитів співробітників з метою вдосконалення роботи бота та оптимізації внутрішніх процесів.

Зменшення впливу людського фактору: мінімізація потреби у залученні колег для вирішення стандартних питань завдяки автоматичним відповідям та інтеграції з базами знань.

Інтеграція з іншими системами компанії: підключення чат-бота до бази даних для створення єдиної точки доступу до інформації.

Безпека та конфіденційність: реалізація контролю доступу, ведення журналів запитів і дотримання внутрішніх політик безпеки компанії.

Висновок до розділу 1

В першому розділі було проведено ініціацію проекту, яка дала безліч корисної інформації, що позитивно вплине на подальшу розробку telegram боту. Спочатку було досліджено саму предметну область, що дало зрозуміти які саме рутинні робочі процеси потребують автоматизації в компанії. Аналіз предметної області показав, що компанія потребує сучасного інструменту автоматизації робочих процесів для підвищення продуктивності роботи працівників, полегшення їх рутинних завдань, що позитивно вплине на дохід компанії. Далі визначили межі його реалізації, учасників та зовнішні фактори, які можуть впливати на успіх створеного боту. На основі отриманих даних провели техніко-економічне обґрунтування, в якому було доведено та обґрунтовано доцільність створення програмного боту на основі месенджера, оскільки воно надає можливість компанії отримати чудовий інструмент автоматизації робочих процесів, з меншими витратами, чим в разі використання інших закордонних аналогів. В подальшому сформували головні цілі розробки та визначили ключових стейкхолдерів, які можуть внести вплив на проект або користуватимуться готовим продуктом, а також описано цільову аудиторію майбутнього програмного забезпечення. Аналіз конкурентів та готових рішень, який показав, що більшість із них не враховують специфіку внутрішніх процесів компанії, що розробка власного

програмного продукту є доцільним. У рамках роботи також виявлено основні ризики та оцінено їх імовірність, що дозволить підготуватися до можливих проблем у процесі реалізації. Опис концепції програмного забезпечення сформував загальне уявлення про її функціональні можливості, логіку роботи та роль у внутрішній інфраструктурі підприємства. В заключному етапу було проведено обстеження та моделювання IT-потреб предметної області, яке надало можливість конкретно визначити запити майбутніх користувачів, що їм необхідно та що б вони хотіли автоматизувати за допомогою чат боту.

Отже результати ініціації показали актуальність і необхідність розробки. Було отримано цілісне уявлення про завдання розробленого програмного забезпечення, головних зацікавлених сторін та визначено ризики проекту. Отримана інформація створює міцний фундамент для створення якісного програмного продукту який буде корисним та знайде своїх споживачів.

РОЗДІЛ 2

ПЛАНУВАННЯ ТА АРХІТЕКТУРА

2.1. Виявлення і опис вимог користувачів

Для створення успішного чат-боту який стане незамінним помічником у щоденній роботі працівників компанії проведемо виявлення і опис очікувань користувачів. Для цього проведемо опитування працівників компанії стосовно того, які робочі процеси найбільше займають часу, щоб на їх думку можна було б полегшити за допомогою автоматизації. У результаті аналізу отриманих даних в процесі опитування можем сформувати такі основні вимоги користувачів:

Простота використання.

Співробітники очікують, що чат-бот буде працювати у звичному для них середовищі, а його інтерфейс буде простим та інтуїтивним, що надасть можливість опанувати даний програмний продукт без необхідності проходження додаткового навчання.

Швидкий доступ до інформації.

Співробітники хочуть мати можливість швидко отримувати відповіді на такі питання, як: правила компанії, внутрішні інструкції, контакти колег.

Автоматизація рутинних процесів.

Більшість опитаних вважає що задачі, які сьогодні виконуються вручну , наприклад подання заяви на відпустку, замовлення товару, мають бути доступними в автоматизованому режимі через чат-бот.

Актуальність даних.

Майбутні користувачі очікують від боту, що він надаватиме лише перевірену й актуальну інформацію. Це особливо важливо для інструкцій та правил компанії.

Підтримка різних сценаріїв використання.

Бот має бути універсальним і задовольняти різні потреби від довідкових задач до організаційних .

Доступність з різних пристроїв.

Для опитаних працівників, важливим пунктом було те, що даний продукт має бути кросплатформеним, тобто доступний з будь яких пристроїв чи з телефону чи з ноутбука або комп'ютера.

Надійність і безпека.

Користувачі очікують стабільної роботи системи та впевненості у тому, що їхні дані захищені. Це включає авторизацію, розмежування доступів та захист конфіденційної інформації.

2.2. Технічні вимоги до інформаційної системи на базі Telegram-бота

Сформуємо технічні вимоги до інформаційної системи.

Функціональні вимоги:

Функціональні вимоги є корисними в процесі розробки програмного продукту, адже визначають, що система повинна виконувати.

- ✓ авторизація користувачів через Telegram ID;
- ✓ підтримка основних команд Telegram-бота (/start, /help, /info та інші бізнес-команди);
- ✓ збереження усіх повідомлень і дій користувачів у базі даних;
- ✓ можливість адміністратору переглядати інформацію про користувачів у JavaFX-застосунку;
- ✓ можливість редагування даних (додавання нових користувачів, зміна статусу, блокування тощо);
- ✓ формування статистичних звітів (кількість користувачів, активність, використання функцій бота);
- ✓ підтримка багатокористувацького режиму (одночасна робота декількох користувачів з ботом).

Для кращого розуміння визначених функціональних вимог, сформуємо таблицю(Таблиця 3) в якій буде вказано назву самої вимоги та короткий її опис:

Таблиця 2.1.1.

Функціональні вимоги

№	Вимога	Опис
1	Авторизація користувача	Використання Telegram ID для ідентифікації
2	Обробка команд	Підтримка базових і бізнес-команд
3	Збереження даних	Усі дії та повідомлення зберігаються в БД
4	Адміністрування	JavaFX-застосунок для роботи з даними
5	Звіти	Формування статистики та моніторинг активності

Нефункціональні вимоги

Нефункціональні вимоги допомагають сформувати загальні характеристики створюваної системи, які забезпечать такі ключові фактори як надійність та зручність використання.

Основні вимоги:

Продуктивність – система повинна працювати швидко, обробляючи запити користувачів у прийнятний час (не більше декількох секунд), навіть при значному навантаженні.

Надійність – Telegram-бот має бути доступний цілодобово, без збоїв та втрати даних, а у випадку збоїв – мати можливість відновлення.

Масштабованість – у майбутньому систему можна розширювати, додаючи нові функції або збільшуючи кількість користувачів без суттєвої перебудови архітектури.

Зручність використання – інтерфейс адміністративного застосунку на JavaFX повинен бути простим і зрозумілим навіть для користувачів без глибоких технічних знань.

Сумісність – система повинна працювати на найпоширеніших операційних системах (Windows, Linux) і легко інтегруватися з іншими сервісами у разі потреби.

Безпека – усі дані користувачів і службова інформація мають бути

захищені від несанкціонованого доступу, з використанням шифрування, авторизації та ведення журналів дій.

Для візуалізації нефункціональним вимог, подамо у вигляді таблиці (Таблиця 4) яка включатиме назву нефункціональної вимоги та її опис.

Таблиця 2.1.2

Нефункціональні вимоги

Категорія	Опис вимог
Продуктивність	Система повинна швидко реагувати на запити користувачів, забезпечуючи стабільну роботу навіть при великій кількості одночасних звернень.
Надійність	Бот і база даних повинні працювати безперервно, з мінімальними простоями. У випадку помилки має бути можливість швидкого відновлення.
Масштабованість	Архітектура системи повинна дозволяти розширення функціоналу та збільшення кількості користувачів без значних змін у структурі.
Зручність	Адміністративний інтерфейс має бути інтуїтивно зрозумілим і доступним для працівників, які не є IT-фахівцями.
Сумісність	Програмне забезпечення повинно працювати на основних ОС та підтримувати взаємодію з іншими корпоративними системами.
Безпека	Забезпечення захисту персональних даних, використання авторизації, паролів і журналів для моніторингу дій користувачів.

Вимоги до бази даних

Для забезпечення зберігання та обробки інформації реалізуємо базу даних SQLlite, яка є найбільш поширеною та стабільною системою управління базами даних. Сам SQLlite підтримує стандарт SQL, яка забезпечує високу швидкість обробку запитів і має гнучкість на високому рівні для інтеграції з

додатками що реалізуються на мові Java.

Створена база даних має включати основні таблиці:

- ✓ employees – таблиця для зберігання розширеної інформації про співробітників. Таблиця використовується для авторизації, визначення ролей та персоналізації взаємодії з ботом.
- ✓ departments – довідкова таблиця департаментів компанії. Використовується для зв'язку співробітників із конкретним підрозділом.
- ✓ shops – таблиця для зберігання інформації про магазини. Забезпечує можливість прив'язування працівників до конкретного місця роботи.
- ✓ suppliers – таблиця постачальників із зазначенням їх назви та контактних даних. Використовується під час роботи з товарами.
- ✓ products – таблиця товарів, що містить: штрихкод, назву, постачальника, ціну та поточні залишки. Дозволяє здійснювати пошук товарів та формувати замовлення.
- ✓ orders – таблиця для зберігання замовлень на товари, створених працівниками. Містить ID співробітника, товар, кількість, дату створення та статус замовлення.
- ✓ it_tickets – таблиця заявок у IT-відділ. Зберігає текст повідомлення, шлях до фото, рейтинг, дату створення, статус, а також інформацію про автора й відповідального спеціаліста.
- ✓ parking – таблиця для обліку паркомісць, де визначається статус зайнятості, користувач, який зайняв місце, та номер місця. Забезпечує роботу модуля бронювання паркування.
- ✓ assets – таблиця, у якій зберігаються корпоративні активи (обладнання) з прив'язкою до відповідальних співробітників.
- ✓ applications – таблиця, що містить перелік шаблонів заяв, які працівники можуть використовувати.

Основні вимоги до організації бази даних:

- ✓ Використання первинних ключів (PRIMARY KEY) у всіх

таблицях для унікальної ідентифікації записів.

- ✓ Реалізація зовнішніх ключів (FOREIGN KEY) для забезпечення зв'язків між таблицями (наприклад, повідомлення повинні мати зв'язок з користувачем, який їх надіслав).
- ✓ Забезпечення індексації для швидкого пошуку, особливо в таблицях з великим обсягом даних (наприклад, messages).
- ✓ Організація регулярного резервного копіювання даних, щоб запобігти їх втраті у випадку технічних проблем або збоїв.
- ✓ Використання методів шифрування конфіденційної інформації (наприклад, паролів, ключів або інших чутливих даних).
- ✓ Оптимізація структури таблиць для забезпечення швидкої обробки транзакцій та зручності адміністрування.
- ✓ Забезпечення масштабованості: структура бази даних має дозволяти додавання нових таблиць і полів без порушення цілісності системи.

Таким чином, база даних виступає ключовим елементом інформаційної системи, що відповідає за зберігання усіх необхідних даних, що забезпечують стабільність роботи Telegram-бота та адміністративного застосунку.

Вимоги до продуктивності

Продуктивність системи є важливим критерієм її ефективності, оскільки від цього безпосередньо залежить якість взаємодії користувачів з Telegram-ботом та швидкість виконання адміністративних операцій.

Основні вимоги до продуктивності:

1. Система повинна підтримувати одночасну роботу не менше 200–300 користувачів без помітного зниження швидкості. Це забезпечить можливість масштабного використання продукту у медичних закладах або компаніях із великою кількістю працівників.
2. Час відповіді Telegram-бота на повідомлення користувача має становити не більше 2–3 секунд, щоб уникнути затримок і зробити взаємодію комфортною.
3. Усі SQL-запити до бази даних повинні бути оптимізованими:

- ✓ використання індексів для прискорення пошуку;
- ✓ уникнення надмірно складних запитів;
- ✓ застосування кешування при повторюваних операціях.

4. Система повинна мати можливість масштабування, тобто розширення ресурсів у випадку зростання навантаження. горизонтального або вертикального масштабування бази даних.

5. Важливо забезпечити стабільність роботи у режимі 24/7, щоб користувачі могли взаємодіяти з ботом у будь-який час доби.

6. При збільшенні кількості даних у базі система повинна зберігати стабільний рівень швидкодії, не допускаючи значного погіршення продуктивності.

2.3. Формування концепції (стратегії) створення програмного застосунку

Створення програмного застосунку у вигляді Telegram-бота потребує чіткої стратегії.

Основні підходи та кроки сформованої концепції:

Визначення цілей та завдань системи.

На першому етапі формуємо головну мету проекту: створення внутрішнього інструмента для автоматизації роботи компанії, який дозволяє швидко обмінюватися інформацією, вести облік дій та працювати з користувачами. Завданням якої є зробити систему зручною, зрозумілою та такою, що легко підтримується.

Вибір архітектури.

Для зручності обслуговування та розширення буде використана модульна архітектура. Telegram-бот відповідає за взаємодію з користувачами, JavaFX — за візуальну частину для адміністраторів, а SQLlite виступає як надійне сховище даних. Такий поділ дозволить гнучко змінювати окремі компоненти без повного переписування системи.

Розробка бази даних.

Передбачено створення реляційної БД, що міститиме інформацію про користувачів, повідомлення, команди, налаштування та журнали подій. Правильне проєктування БД є основою продуктивності системи, адже саме там зберігатимуться всі важливі дані.

Інтеграція з Telegram API.

Бот буде побудований на основі офіційного Telegram Bot API. Це забезпечить надійну обробку повідомлень та команд, а також можливість масштабування у разі зростання кількості користувачів.

Розробка бізнес-логіки.

Основна логіка роботи передбачатиме обробку команд, збереження історії взаємодії, ведення журналів, контроль налаштувань та можливість адміністрування. Важливо, щоб бот міг працювати автономно та не потребував постійного втручання людини.

Розробка адміністративного інтерфейсу.

JavaFX використовується для створення графічного інтерфейсу адміністратора. Він дозволить керувати користувачами, переглядати повідомлення, змінювати налаштування та відслідковувати активність системи.

Тестування та відлагодження.

На кожному етапі необхідно проводити тестування — як окремих модулів, так і всієї системи. Особливу увагу слід приділити перевірці надійності роботи з базою даних, швидкості відповіді та стійкості до помилок.

Впровадження та навчання персоналу.

Після завершення розробки бот буде поступово впроваджений у роботу компанії. Адміністраторам та працівникам буде надано інструкції щодо користування системою.

Підтримка та розвиток.

У подальшому передбачено можливість масштабування системи, додавання нових функцій та розширення бази даних відповідно до потреб компанії.

2.4. Вибір і обґрунтування методології і стеки технологій для архітектурного і детального проектування та необхідні інструментальні засоби

Для розробки Telegram-бота як внутрішньої інформаційної системи компанії було вирішено обрати об’єктно-орієнтовану методологія у поєднанні з реляційною моделлю даних.

Об’єктно-орієнтований підхід дозволить нам структурувати бізнес-логіку, виділяти основні класи та легко масштабувати та підтримувати код, що в подальшому наддасть можливість легко додавати новий функціонал. Використання реляційної бази даних забезпечить цілісність і надійність збереження даних та дозволить уникати дублювання, підтримувати зв’язки між таблицями та виконувати складні SQL-запити. Поєднання цих двох методологій зробить систему більш гнучкою, прозорою й стійкою до змін у майбутньому.

Стек технологій

1. Система управління базами даних:

SQLlite — відкрита реляційна СУБД, перевагами якої є: висока продуктивність, надійність, простота використання та велика спільнота підтримки. SQLlite добре інтегрується з Java через JDBC та Hibernate.

2. Мова програмування для бізнес-логіки:

Java — основна мова для реалізації логіки Telegram-бота. Має стабільність, кросплатформеність та велику кількість бібліотек. Для роботи з Telegram використовується TelegramBots Java API.

3. Графічний інтерфейс адміністратора:

JavaFX — сучасний фреймворк для побудови десктопних додатків. Дозволить створити зручний адміністративний інтерфейс із таблицями, формами, графіками та панелями управління.

4. Фреймворки та бібліотеки:

TelegramBots API — офіційна Java-бібліотека для роботи з Telegram Bot

API.

5. Інструменти для тестування:

JUnit — для модульного тестування бізнес-логіки.

Postman — для перевірки REST-запитів і взаємодії бот ↔ сервер ↔ база даних.

Apache JMeter — для навантажувального тестування, щоб оцінити швидкодію бота при зростанні кількості користувачів.

6. Інструменти для управління базою даних:

SQLlite — для проєктування, адміністрування та візуалізації бази даних.

7. Інструменти для управління кодом та процесом розробки:

Git + GitHub/GitLab — для контролю версій і командної роботи.

Maven/Gradle — для керування залежностями та збирання проєкту.

IntelliJ IDEA — середовище розробки (IDE), яке забезпечує швидкість, підказки коду та інтеграцію з базою.

Для візуалізації стек технологій було вирішено створити таблицю(Таблиця 5), яка включатиме назву компонента, технологію та призначення:

Таблиця 2.1.3.

Стек технології

Компонент	**Технологія / Інструмент**	**Призначення**
СУБД	SQLlite	зберігання та обробка даних
ORM	Hibernate	робота з БД через Java-класи
Мова програмування	Java	бізнес-логіка системи
Інтерфейс адміністратора	JavaFX	GUI для адміністраторів
API для Telegram	TelegramBots Java	інтеграція з Telegram

	API	
Тестування	JUnit, Postman, Apache JMeter	модульне, інтеграційне та навантажувальне тестування
Управління БД	SQLlite	проєктування та версіонування БД
Контроль версій	Git + GitHub/GitLab	командна робота та збереження коду
Інструменти розробки	IntelliJ IDEA / Eclipse, Maven/Gradle	зручна робота з кодом і залежностями

Висновок до розділу 2

У розділі 2 було проведено сформовано концепції створення програмного застосунку з адміністративним інтерфейсом на JavaFX та реляційною базою даних MySQL, та визначено технічні та функціональні вимоги, описали стратегію розробки, а також підібрано стек технологій і необхідні інструменти. Обґрунтували доцільність використання об'єктно-орієнтованої методології для побудови бізнес-логіки та реляційної моделі даних для надійного збереження інформації. Такий підхід дозволяє створити масштабовану, захищену та продуктивну систему, яка буде ефективно виконувати внутрішні завдання компанії.

Вибраний стек технологій забезпечує:

- ✓ стабільність і надійність роботи системи;
- ✓ простоту адміністрування та підтримки;
- ✓ можливість швидкого масштабування;
- ✓ зручну інтеграцію з Telegram для взаємодії з користувачами.

РОЗДІЛ 3

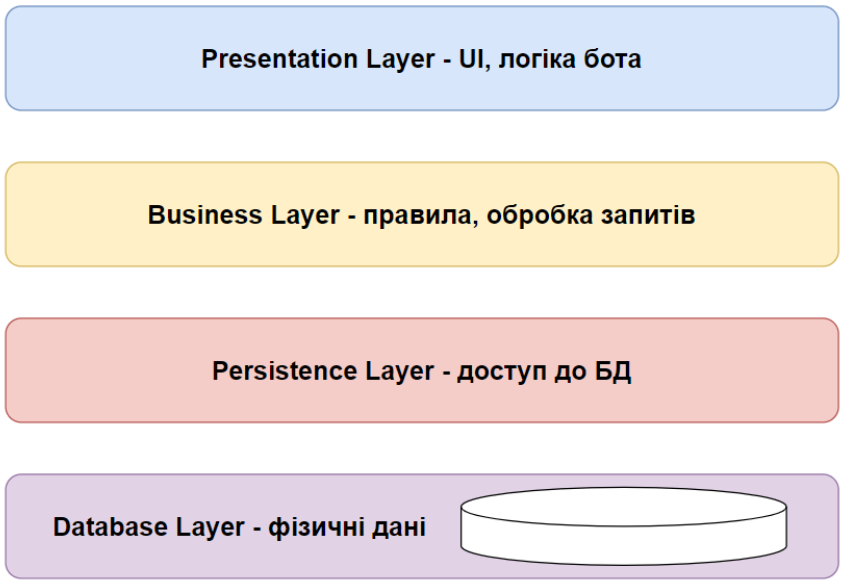
ПЕРВИННЕ ПРОЕКТУВАННЯ

3.1. Архітектура проекту

Для забезпечення високої якості розроблюваного програмного продукту необхідно чітко усвідомлювати з яких компонентів буде він складатись, які проблеми даний програмний продукт буде вирішувати та які рутинні робочі процеси буде автоматизовувати, для цього були розроблені діаграма, такі як:

- ✓ ER-діаграма
- ✓ Діаграма варіантів використання;
- ✓ Діаграма класів;
- ✓ Діаграма послідовності;
- ✓ Діаграма станів.

Але для того щоб діаграми були виконані вірно і відповідали дійсності слід на сам перед обрати архітектуру створюваного програмного продукту. На мій погляд для даного програмного продукту найліпше підходить 4-рівнева архітектура (Діаграм 3.1.1.).



Діаграма 3.1.1. 4-рівнева архітектура

Можна виділити таки переваги 4-рівневої архітектури:

1. Чітке розділення відповідальностей. Кожен рівень реалізує власну

роль, а саме: інтерфейс взаємодіє з користувачем, бізнес-логіка відповідає за правила, рівень доступу до даних управляє запитами, база даних лише зберігає інформацію. Чітке розділення відповідальностей робить систему досить структурованою і легкою в розумінні.

2. Спрощення підтримки та модернізації. Зміни всередині одного рівня не впливають на інші, що дозволяє переробляти логіку замовлень, переписати Telegram-бот або замінити SQLite на PostgreSQL, при тому не доведеться перероблювати інший рівень, що є важливим для телеграм ботів які постійно адаптуються та розширяються.

3. Масштабованість системи. 4-рівнева архітектура дозволяє легко додавати нові функції. Система розширюється горизонтально, без переписування ядра.

4. Підвищена надійність та стабільність Завдяки розмежуванню рівнів помилка на UI не впливає на БД, збій у бізнес-логіці не шкодить фізичним даним, а Persistence-level перехоплює проблеми доступу. Це робить чат-бот більш стійким до помилок.

5. Розширюваність під майбутні технології. 4-рівневі архітектури легко адаптувати під: інтеграцію з LLM, RAG-модулі, аналітику, мікросервісні компоненти. Що дає можливість модернізувати до «розумного» бота без архітектурного рефакторингу.

6. Підвищена безпека системи. UI та бізнес-логіка не мають прямого доступу до бази даних, що зменшує ризик SQL-ін'єкцій, неконтрольованих операцій, та спрощує процес обмежування прав. Persistence-рівень гарантує безпечний доступ до збережених даних.

7. Можливість паралельної розробки. Команда розробників може працювати одночасно, що прискорює створення та оновлення проекту.

8. Підтримка SOLID-принципів. 4-рівневий підхід природно реалізує:

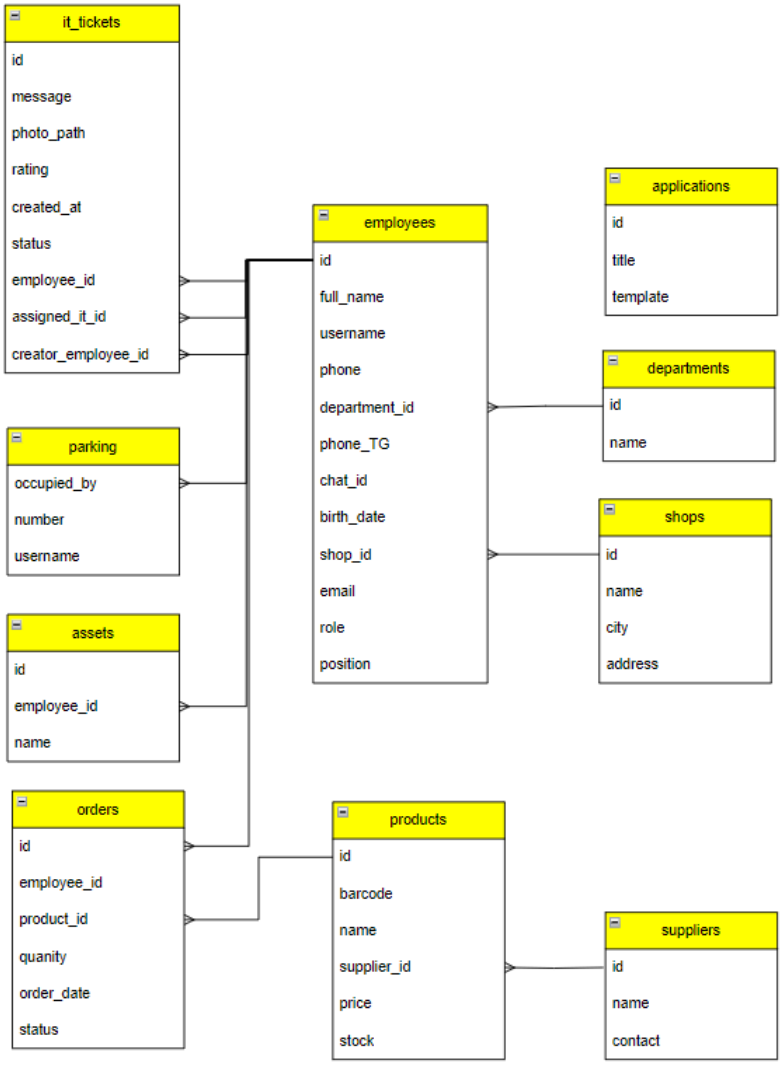
- ✓ SRP — кожен рівень має одну відповідальність;
- ✓ OCP — можна розширювати, не змінюючи існуюче;
- ✓ DIP — бізнес-логіка працює через абстракції, а не напряму з

базою.

9. Висока повторюваність та універсальність. Багато функцій можна повторно використовувати, такі як базові сервіси, модельні класи, логіку роботи з БД, що пришвидшує створення нових чат-ботів того ж типу.

3.2. ER-діаграма

Ключовим етапом первинного проектування є створення ER-діаграми, оскільки вона домогає визначити структуру предметної області та взаємозв'язків між ними. Вона дозволяє виявити помилки на ранньому етапі розробки програмного продукт та надасть можливість уникнути дублювання даних. Тож було вирішено створити ER-діаграму (Діаграма 3.2.1.) щоб уникнути вищенаведених факторів.



Діаграма 3.2.1. ER-діаграма

Опис ER-діаграми:

Створена ER-діаграма бази даних відображає основні сутності внутрішньої платформи та зв’язки між ними. Основною таблицею є Employees, що містить необхідні дані про працівників, такі як:

- ✓ Ідентифікатор
- ✓ Повне ім’я;
- ✓ Контактна інформація;
- ✓ Посада;
- ✓ Роль;
- ✓ Належність до відділу та магазину.

Від неї походять основні зв’язки системи.

Таблиця IT_Tickets використовується для зберігання заявок на IT-підтримку . Вона містить три зовнішні ключі на Employees: автор заявки, виконавець та працівник, який ініціював звернення. Кожен працівник може створювати багато заявок, а також отримувати їх на виконання.

Сутність Parking забезпечує облік паркомісць на парковкі компанії. Кожне місце пов’язане з одним працівником через поле occupied_by.

Сутність Assets зберігає дані про матеріальні активи та обладнання, закріплені за працівниками.

Сутність Orders відображає внутрішні замовлення товарів: кожне замовлення пов’язане з конкретним працівником та конкретним товаром.

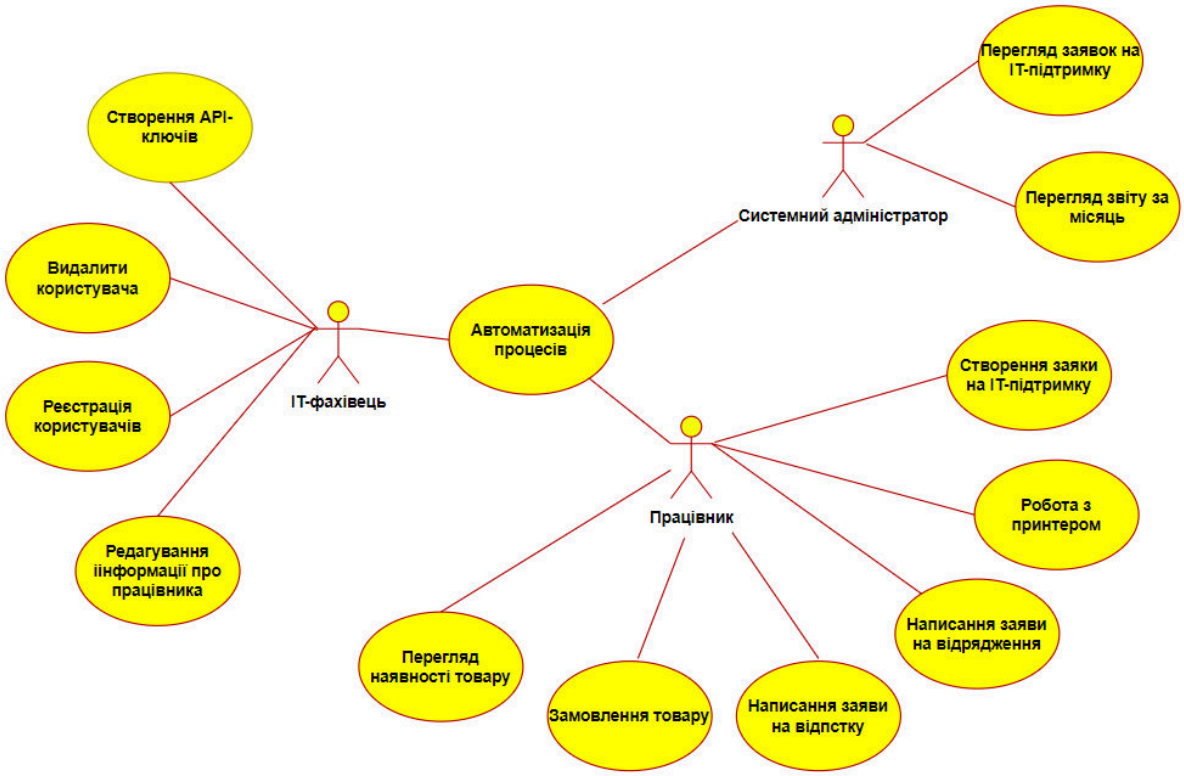
Сутність Products містить перелік товарів, кожен із яких має постачальника

Сутності Departments та Shops визначають організаційну структуру: працівник належить до одного відділу та одного магазину, а кожен підрозділ може містити багато працівників.

Сутність Applications є каталогом шаблонів внутрішніх документів і не має зовнішніх зв’язків з іншими таблицями.

3.3. Діаграма варіантів використання

Для відображення варіантів взаємодії користувачів програмного продукту з її функціональними можливостями було вирішено створити діаграму варіантів використання (Діаграма 3.3.1). Вона дає змогу визначити основних акторів, визначити їх цілі та встановити межі програмного продукту. Сама ж діаграма допоможе сформувати чітке розуміння вимог до розроблюваної системи, структурувати функції та забезпечити створення програмного продукту, який буде відповідати очікуванням як замовника так і майбутнім користувачам. Надалі можете побачити розроблену діаграму з її акторами та варіантами використання:



Діаграма 3.3.1 Діаграма варіантів використання.

Опис діаграми варіантів використання:

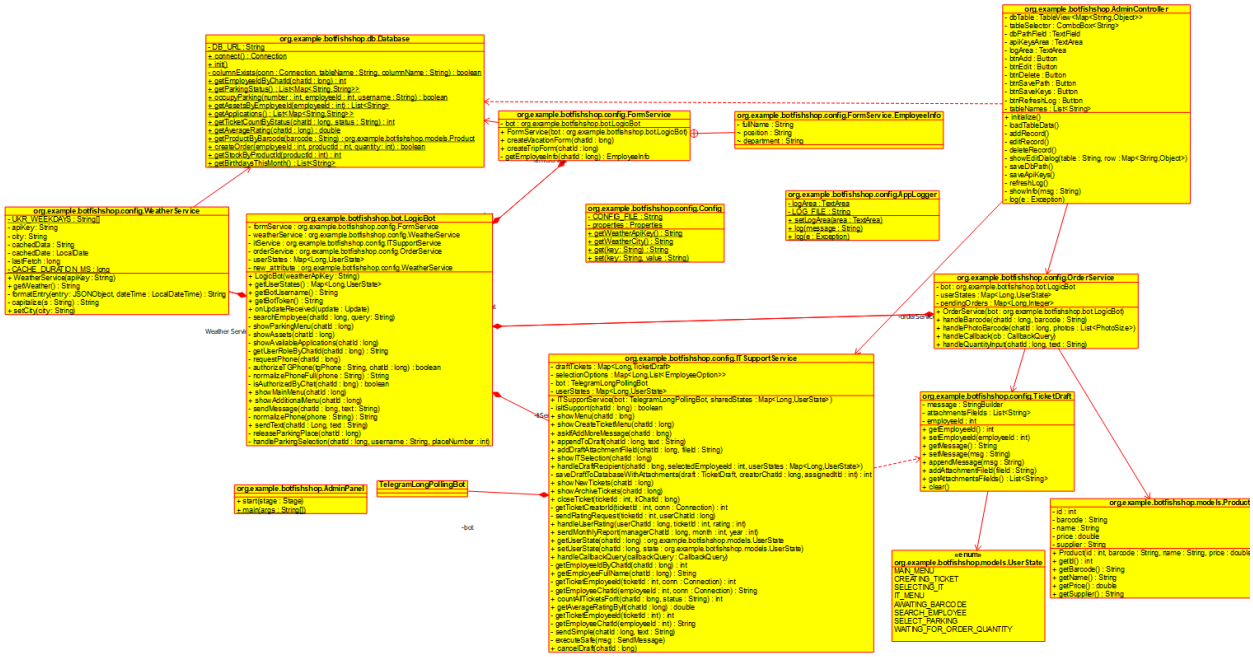
Діаграма варіантів використання відображає структуру взаємодії користувачів із системою та показує, які функції доступні кожному типу користувачів. У центрі діаграми розташована сама система, а навколо неї — три основні актори: IT-фахівець, Системний адміністратор та Працівник.

Кожен із них має власний набір можливостей, які відповідають їхнім обов'язкам та ролі в організації. IT-фахівець взаємодіє з функціональністю, що стосується керування користувачами та технічних аспектів роботи системи. Він може здійснювати реєстрацію нових користувачів, редагувати інформацію про працівників, видаляти користувачів та створювати API-ключі, які потрібні для інтеграції зовнішніх сервісів або автоматизації процесів обміну даними. Такий набір функцій підкреслює його технічну роль та відповідальність за підтримку коректного функціонування системи. Системний адміністратор відповідає за контроль роботи системи та моніторинг діяльності користувачів. Йому доступні такі можливості, як перегляд заявок на IT-підтримку, що дозволяє відстежувати та контролювати процес обробки технічних звернень від працівників, а також перегляд місячних звітів, у яких міститься аналітична інформація про навантаження, ефективність роботи або статистику по зверненням. Функції системного адміністратора зосереджені на аналізі та управлінні, що підкреслює його координуючу роль. Працівник — це основний користувач, який взаємодіє із системою для виконання внутрішніх задач компанії. Він може переглядати наявність товару, створювати замовлення, подавати заяви на відпустку та відрядження, працювати з принтером наприклад, друкувати документи, а також створювати заявки на IT-підтримку, коли виникають технічні проблеми. Для працівника система слугує інструментом, що спрощує повсякденні процедури й дозволяє швидко виконувати адміністративні та операційні завдання. Загалом діаграма демонструє чіткий розподіл доступних можливостей між різними типами користувачів та відображає структуру автоматизованих процесів у межах підприємства.

3.4. Діаграма класів

Після того як було зрозуміло процеси які потрібно автоматизувати з діаграми варіантів використання, було побудовано наступну діаграму, а саме діаграму класів (Діаграма 3.4.1.), для відображення структури проекту,

основних класів системи, їх властивостей та методів взаємодії між ними. Створення даної діаграми є важливим етапом в первинному проектуванні, оскільки саме вона забезпечує цілісне розуміння внутрішньої структури програмної системи та слугує основою для подальшої реалізації програмного продукту.



Діагарма 3.4.1. Діаграма класів.

Опис діаграми класів:

Діаграма класів відображає архітектуру корпоративного чат-бота, що складається з центрального логічного модуля, сервісних компонентів, моделі даних, шарів доступу до бази та адміністративної підсистеми, розглянем їх більш детально:

1) **LogicBot** — виступає центральним елементом. Через нього проходять основні сценарії: вибір меню, обробка команд, виклики сервісів (WeatherService, FormService, ITSupportService, OrderService). Він не зберігає даних, а лише керує логікою й переходами станів користувача.

2) **Database** — єдиний клас доступу до SQLite: отримання співробітників, товарів, замовлень, створення записів, оновлення полів.

Він використовується сервісами (OrderService, FormService, ITSupportService) для читання/запису даних.

3) Сервіси бізнес-логіки:

✓ **WeatherService** —Повертає погоду, кешує дані, форматує відповідь. Використовується LogicBot.

✓ **OrderService** — Логіка замовлень: вибір товарів, фото баркоду, підтвердження кількості, виклик Database, передача Callback'ів.

✓ **FormService** — Створення заявок/форм, зчитування даних співробітника, передача в Database.

4) ITSupportService — Найбільший сервіс: тікети, статуси, призначення відповідальних, історія змін, перелік співробітників, інтеграція з Database, Callback'ами та TelegramLongPollingBot.

5) **UserState** — Перелік станів чат-бота: вибір меню, створення тікету, сканування баркоду, пошук працівника, введення кількості товару. Використовується LogicBot, OrderService, ITSupportService.

6) Дані сутностей:

1) **Product** — Містить інформацію про товар: баркод, назву, ціну, постачальника.

2) **TicketDraft** — Чернетка тікета: повідомлення, вкладення, employeeId, поля. Використовується ITSupportService.

3) **EmployeeInfo** — Ім'я, посада, відділ — використовується FormService.

7) Telegram-інфраструктура

✓ **TelegramLongPollingBot** — Приймає повідомлення, передає їх у LogicBot. Нічого сам не обробляє — лише пересилає.

✓ **AdminController** — Графічна панель для керування: таблиця даних, CRUD-операції, фільтри. Працює напряму з Database.

Окрім даних класів на діаграмі класів можемо також бачити не зв'язені класи, але це не є помилкою адже:

1) **AdminPanel** — окремий GUI-модуль, який не залежить від логіки Telegram-бота.

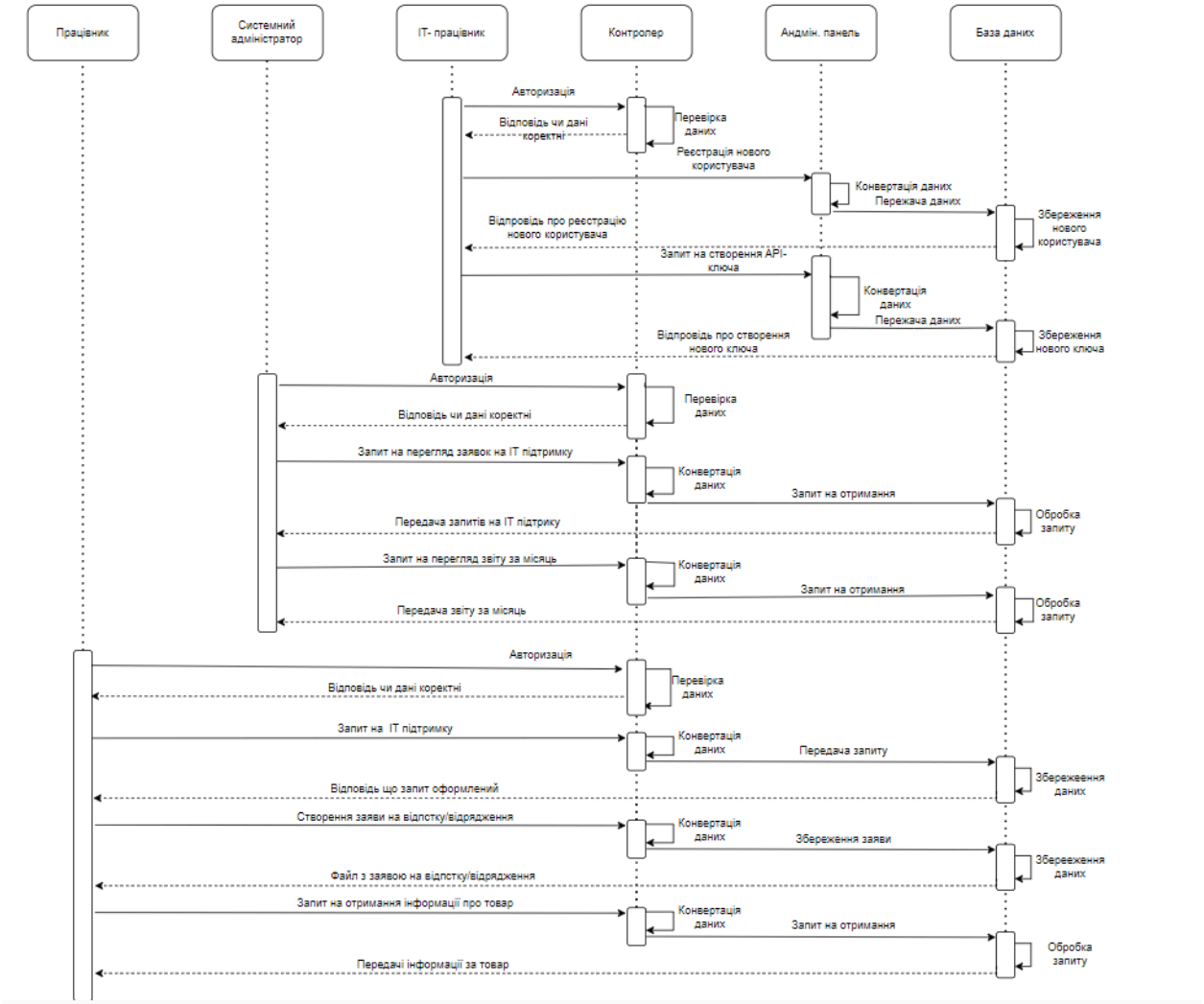
2) **TelegramLongPollingBot** — точка входу Telegram, яку сервісна логіка не викликає назад напряму.

3) **UserState** — перелік констант, який логічно не наслідується, тому не має стрілок.

4) **Config**, **AppLogger** — утилітарні класи, які використовуються статично, тому UML не показує зв’язків.

3.5. Діаграма послідовності

Після того як було створено та описано діаграму класів можна перейти до неменше важливого пункту первинного проектування, а саме до створення діаграми послідовності (Діаграма 3.5.1.). Призначенням даної діаграми є відображення динамічної взаємодії між об’єктами системи під час її роботи. Створення та використання діаграми послідовності допомогає нам деталізувати поведінку програмного продукту, уточнити взаємодію компонентів. Тож перейдемо до її створення:



Діаграма 3.5.1. Діагарма послідовності.

Опис діаграми послідовності:

Працівник ↔ Контролер: Тут відбувається передача номера телефону користувача до контролера, який здійснює перевірку коректності введених даних. Після обробки інформації контролер повертає відповідь про те, чи є дані коректними.

ІТ-працівник → Контролер: Тут здійснюється передача даних, необхідних для реєстрації нового користувача. Контролер перевіряє коректність введених даних та формує запит на створення нового облікового запису.

Контролер → Адмін. панель: Тут контролер передає інформацію для здійснення конвертації даних у формат, необхідний для подальшого збереження.

Адмін. панель → База даних: Відбувається передача вже конвертованих даних до бази даних, де вони зберігаються як новий користувач системи.

База даних → IT-працівник: Після успішного збереження база даних формує відповідь та надсилає IT-працівнику повідомлення про те, що новий користувач був успішно зареєстрований.

IT-працівник → Контролер: Відбувається передача запиту на створення нового API-ключа. Контролер перевіряє передані дані та формує відповідний запит.

Контролер → Адмін. панель: Контролер передає інформацію для конвертації та підготовки до збереження ключа.

Адмін. панель → База даних: Тут база даних отримує конвертовані дані та зберігає створений API-ключ.

База даних → IT-працівник: Після успішного створення та збереження нового ключа база даних надсилає відповідь IT-працівнику.

Системний адміністратор ↔ Контролер: На цьому етапі адміністратор передає контролеру номер телефону. Контролер перевіряє коректність даних та повертає відповідь про успішну або неуспішну авторизацію.

Системний адміністратор → Контролер: Тут адміністратор передає запит на перегляд заявок на IT-підтримку.

Контролер → Адмін. панель: Контролер передає відповідний запит для його конвертації та підготовки.

Адмін. панель → База даних: Тут відбувається передача запиту до бази даних для отримання всіх заявок.

База даних → Системний адміністратор: База даних повертає адміністратору інформацію про всі заявки на IT-підтримку.

Системний адміністратор → Контролер: Адміністратор формує запит на отримання звіту за місяць, який передається контролеру.

Контролер → Адмін. панель: Запит проходить процес конвертації та підготовки до передачі.

Адмін. панель → База даних: Відправляється запит на отримання місячного звіту.

База даних → Системний адміністратор: База даних повертає адміністратору сформований звіт за місяць.

Працівник ↔ Контролер: Працівник вводить свої облікові дані, а контролер перевіряє їх коректність та повідомляє результат авторизації.

Працівник → Контролер: Тут працівник передає запит на створення заявки у ІТ-підтримку.

Контролер → Адмін. панель: Контролер передає дані на конвертацію для їх подальшої обробки.

Адмін. панель → База даних: Тут виконується збереження заявки на ІТ-підтримку у базі даних.

База даних → Працівник: База повідомляє працівнику, що заявка успішно створена.

Працівник → Контролер: Працівник формує та надсилає запит на створення заяви про відпустку або відрядження.

Контролер → Адмін. панель: Дані заявки конвертуються перед передачею до бази даних.

Адмін. панель → База даних: Тут зберігається сформована заява.

База даних → Працівник: База даних повідомляє про успішне збереження заяви.

Працівник → Контролер: Працівник формує запит на отримання інформації про товар.

Контролер → Адмін. панель: Адмін-панель виконує необхідну конвертацію запиту.

Адмін. панель → База даних: Формується і надсилається запит на отримання даних про товар.

База даних → Працівник: Після обробки книги запиту база повертає інформацію про товар працівнику.

Висновок до розділу 3

У даному розділі було проведено первинне проектування системи. На основі визначених вимог, що були сформовані в попередніх розділах побудували архітектуру проекту, що відображає структуру його основних компонентів та принципи їх взаємодії. Створена ER-діаграма дозволила сформувати логічну модель бази даних і визначити основні сутності та зв’язки між ними, та навели короткий опис їх. Діаграма варіантів використання дала змогу описати та наочно зобразити функціональні можливості системи та роль користувачів у її роботі. Діаграма класів відобразила внутрішню структуру системи, її класи та об’єкти, атрибути та зв’язки між класами, а діаграма послідовності деталізувала динамічну взаємодію компонентів під час виконання основних сценаріїв. Сукупність отриманих моделей формує основу для подальшої розробки та дозволяє забезпечити узгодженість між вимогами, архітектурою та реалізацією програмного продукту.

РОЗДІЛ 4

ПРОТОТИПУВАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1. Створення прототипу БД.

На першому етапі створено клас Database (Рис 4.1.1. Лістинг коду 1), що реалізує підключення до SQLite, відкриття з'єднання, виконання SQL-запитів та створення основних таблиць. Це фундаментальний компонент, на якому ґрунтується робота всіх сервісів.

```
public class Database {
    private static final String DB_URL = "jdbc:sqlite:company.db";

    public static Connection connect() throws SQLException {
        return DriverManager.getConnection(DB_URL);
    }

    // Ініт бази, створюється автоматом при запуску бота
    public static void init() {
        try (Connection conn = connect(); Statement st =
conn.createStatement()) {
            st.executeUpdate("PRAGMA foreign_keys = ON");
```

Рис 4.1.1. Лістинг коду 1

Після того як сам клас, який відповідає за реалізацію підключення до SQLite, відкриття з'єднання, виконання SQL-запитів та створення основних таблиць було створено можна перейти до створення базових таблиць працівників, відділів кадрів та магазинів. Вони є центральними сутностями бізнес-логіки: співробітники прив'язуються до магазинів і відділів, а їх chat-ID використовується ботом для ідентифікації. (Рис 4.1.2. Лістинг коду 2)

```
st.executeUpdate("CREATE TABLE IF NOT EXISTS employees (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "full_name TEXT NOT NULL, " +
        "username TEXT UNIQUE NOT NULL, " +
        "phone TEXT UNIQUE NOT NULL, " +
        "phone_TG TEXT UNIQUE, " +
        "chat_id TEXT UNIQUE, " +
        "department_id INTEGER, " +
        "shop_id INTEGER, " +
        "birth_date DATE, " +
        "email TEXT, " +
        "role TEXT, " +
        "FOREIGN KEY(department_id) REFERENCES departments(id), " +
        "FOREIGN KEY(shop_id) REFERENCES shops(id))");

st.executeUpdate("CREATE TABLE IF NOT EXISTS departments (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL)");

st.executeUpdate("CREATE TABLE IF NOT EXISTS shops (" +
```

```
        "id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, city TEXT,
address TEXT)");
```

Рис 4.1.2. Лістинг коду 2

Після визначення працівників було створено блок таблиць, які відповідають за товарні операції: постачальники, товари, замовлення (Рис 4.1.3. Лістинг коду 3). Ці сутності формують логіку модулів складу та продажу.

```
st.executeUpdate("CREATE TABLE IF NOT EXISTS suppliers (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, contact
TEXT)");

st.executeUpdate("CREATE TABLE IF NOT EXISTS products (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, barcode TEXT UNIQUE NOT NULL,
name TEXT NOT NULL, " +
        "supplier_id INTEGER, price REAL, FOREIGN KEY(supplier_id) REFERENCES
suppliers(id))");

st.executeUpdate("CREATE TABLE IF NOT EXISTS orders (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, employee_id INTEGER,
product_id INTEGER, quantity INTEGER, " +
        "order_date DATETIME DEFAULT CURRENT_TIMESTAMP, status TEXT DEFAULT
'NEW', " +
        "FOREIGN KEY(employee_id) REFERENCES employees(id), FOREIGN
KEY(product_id) REFERENCES products(id))");
```

Рис 4.1.3. Лістинг коду 3

Далі створив таблицю it_tickets (Рис 4.1.4. Лістинг коду 4), для зберігання звернень працівників на IT-підтримку і додано можливість призначення заявки конкретному IT-спеціалісту та фіксацію працівника який звернувсі за допомогою.

```
st.executeUpdate("CREATE TABLE IF NOT EXISTS it_tickets (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, employee_id INTEGER, message
TEXT, photo_path TEXT, rating INTEGER, " +
        "created_at DATETIME DEFAULT CURRENT_TIMESTAMP, status TEXT DEFAULT
'OPEN', " +
        "FOREIGN KEY(employee_id) REFERENCES employees(id))"); if
(!columnExists(conn, "it_tickets", "assigned_it_id")) {
    st.executeUpdate("ALTER TABLE it_tickets ADD COLUMN assigned_it_id
INTEGER");
    System.out.println("Column 'assigned_it_id' added to it_tickets ✅");
}
if (!columnExists(conn, "it_tickets", "creator_employee_id")) {
    st.executeUpdate("ALTER TABLE it_tickets ADD COLUMN creator_employee_id
INTEGER");
    System.out.println("Column 'creator_employee_id' added to it_tickets
✅");
```

Рис 4.1.4. Лістинг коду 4

На наступному етапі було створено таблиці, які реалізують додаткові сервіси корпоративного бота: бронювання паркувальних місць, таблиця

основних засобів та шаблони заяв (Рис 4.1.5. Лістинг коду 5).

```
// Таблиця парковки
st.executeUpdate("CREATE TABLE IF NOT EXISTS parking (" +
        "number INTEGER PRIMARY KEY, " +
        "occupied_by INTEGER, " +
        "username TEXT, " +
        "FOREIGN KEY(occupied_by) REFERENCES employees(id))");

// Таблиця основних засобів
st.executeUpdate("CREATE TABLE IF NOT EXISTS assets (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "name TEXT NOT NULL, " +
        "employee_id INTEGER, " +
        "FOREIGN KEY(employee_id) REFERENCES employees(id))");

// Таблиця заяв
st.executeUpdate("CREATE TABLE IF NOT EXISTS applications (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "title TEXT NOT NULL, " +
        "template TEXT)");
```

Рис 4.1.5. Лістинг коду 5

Після створення структури бази даних були розроблені допоміжні методи, які забезпечують доступ до даних та виконання бізнес-операцій. Вони формують ключовий функціонал Telegram-бота, хоча й не змінюють структуру БД. Їхнє призначення — отримання інформації, перевірка наявності колонок, створення замовлень, керування паркуванням, визначення відповідальних працівників та формування статистики. Розглянемо деякі з них:

1. **Перевірка існування колонки у таблиці**

Метод використовується для безпечного додавання нових полів без втрати даних, що дозволяє еволюційно розвивати структуру бази (Рис 4.1.6. Лістинг коду 6).

```
private static boolean columnExists(Connection conn, String tableName, String
columnName) {
    String sql = "PRAGMA table_info(" + tableName + ")";
    try (Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
        while (rs.next()) {
            String name = rs.getString("name");
            if (name.equalsIgnoreCase(columnName)) {
                return true;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

Рис 4.1.6. Лістинг коду 6

2. Отримання ID працівника за chat_id

Метод використовується ботом для ідентифікації користувача Telegram у корпоративній системі (Рис 4.1.7 Лістинг коду 7).

```
public static int getEmployeeIdByChatId(long chatId) {
    String sql = "SELECT id FROM employees WHERE chat_id = ?";
    try (Connection conn = connect();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, String.valueOf(chatId));
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return rs.getInt("id");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}
```

Рис 4.1.7 Лістинг коду 7

3. Операції з парковкою

Методи дозволяють отримувати статус паркувальних місць та бронювати їх (Рис 4.1.8. Лістинг коду 8).

```
public static List<Map<String, String>> getParkingStatus() {
    List<Map<String, String>> list = new ArrayList<>();
    String sql = "SELECT number, occupied_by, username FROM parking";
    try (Connection conn = connect();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
        while (rs.next()) {
            Map<String, String> map = new HashMap<>();
            map.put("number", String.valueOf(rs.getInt("number")));
            map.put("occupied_by", rs.getString("occupied_by"));
            map.put("username", rs.getString("username"));
            list.add(map);
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return list;
}

public static boolean occupyParking(int number, int employeeId, String
username) {
    String sql = "UPDATE parking SET occupied_by = ?, username = ? WHERE
number = ? AND occupied_by IS NULL";
    try (Connection conn = connect();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setInt(1, employeeId);
        ps.setString(2, username);
        ps.setInt(3, number);
        return ps.executeUpdate() > 0;
    } catch (SQLException e) { e.printStackTrace(); }
    return false;
}
```

Рис 4.1.8. Лістинг коду 8

4. Отримання списку заяв

```
public static List<Map<String, String>> getApplications() {
    List<Map<String, String>> list = new ArrayList<>();
    String sql = "SELECT id, title FROM applications";
    try (Connection conn = connect();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
        while (rs.next()) {
            Map<String, String> map = new HashMap<>();
            map.put("id", String.valueOf(rs.getInt("id")));
            map.put("title", rs.getString("title"));
            list.add(map);
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return list;
}
```

Рис 4.1.9. Лістинг коду 9

5. Методи статистики IT-заявок

```
public static Product getProductByBarcode(String barcode) {
    String sql = "SELECT p.id, p.barcode, p.name, p.price, s.name AS supplier
" +
            "FROM products p " +
            "LEFT JOIN suppliers s ON p.supplier_id = s.id " +
            "WHERE p.barcode = ?";
    try (Connection conn = connect();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, barcode);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return new Product(
                    rs.getInt("id"),
                    rs.getString("barcode"),
                    rs.getString("name"),
                    rs.getDouble("price"),
                    rs.getString("supplier")
            );
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

Рис 4.1.10. Лістинг коду 10

Окрім розглянутих вище функцій, у прототипі бази даних були реалізовані додаткові утилітарні методи, які забезпечують розширений функціонал корпоративного чат-бота. До них належать операції отримання активів працівника, пошук товару за штрихкодом, створення внутрішніх замовлень, визначення залишків продукції та формування списку співробітників, які мають день народження у поточному місяці. Методи працюють через стандартні SQL-запити та гарантують швидкий доступ до структурованої інформації, необхідної для відповіді на запити користувача. Функція отримання активів повертає всі основні засоби, закріплені за

працівником, що дозволяє автоматизувати інвентаризацію. Метод пошуку товару за штрихкодом використовується для спрощення процесів обліку та замовлення продукції. Створення замовлення реалізовано як проста транзакційна операція вставки, що дозволяє працівникам оформлювати внутрішні запити на необхідні матеріали. Додатково реалізовано отримання залишків товару, яке дозволяє перевіряти наявність товарів перед оформленням замовлення. Функція формування переліку днів народжень у поточному місяці забезпечує інформаційну підтримку HR-підрозділу та використовується для автоматичних щотижневих або щомісячних повідомлень у чаті.

4.2. Створення бізнес логіки

Бізнес-логіка системи реалізує центральні правила роботи Telegram-бота для внутрішніх процесів компанії: обробку заявок, дії працівників, маршрутизацію, автоматичні повідомлення та формування відповідей. Вона охоплює процеси, які поєднують взаємодію з базою даних, правила переходу між статусами, автоматичні перевірки, фільтрацію даних, логування та шаблонізацію відповідей.

Перший клас з якого почнемо створення бізнес логіки буде **AppLogger**, який реалізує механізм логування подій у застосунку та забезпечує централізоване ведення журналу роботи системи (Рис 4.2.1.Лістинг коду 11). Основним завданням класу є фіксація діагностичних повідомлень, інформаційних подій, а також виняткових ситуацій під час виконання програми.

```
ublic class AppLogger {
    private static TextArea logArea;
    private static final String LOG_FILE = "bot.log";

    public static void setLogArea(TextArea area) {
        logArea = area;
    }

    public static void log(String message) {
        String time = LocalDateTime.now().toString();
        String line = "[" + time + "] " + message;

        // В файл
```

```
        try (PrintWriter pw = new PrintWriter(new FileWriter(LOG_FILE,
true))) {
            pw.println(line);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // В TextArea в FX потоке
        if (logArea != null) {
            Platform.runLater(() -> {
                logArea.appendText(line + "\n");
            });
        }

        System.out.println(line); // в консоль тоже
    }

    public static void log(Exception e) {
        log(e.toString());
        for (StackTraceElement el : e.getStackTrace()) {
            log("    at " + el.toString());
        }
    }
}
```

Рис 4.2.1. Лістинг коду 11

Функціональність класу охоплює декілька напрямів:

1) Запис логів у файл - тобто повідомлення фіксуються у зовнішньому лог-файлі bot.log.

2) Виведення логів у графічний інтерфейс - тобто клас підтримує інтеграцію з JavaFX-компонентом TextArea, який у реальному часі додаються нові повідомлення. Оновлення виконується у JavaFX-потоці за допомогою Platform.runLater(), що гарантує коректну взаємодію з UI.

3) Логування винятків, наявний окремий метод для обробки об’єктів типу Exception. Він записує текст винятку, а також повний стек-трейс, що значно полегшує пошук причин збоїв.

Наступним класом є Клас **Config** відповідає за централізоване зчитування та збереження конфігураційних параметрів системи (Рис 4.2.2 Лістинг коду 12) . Його призначення — забезпечити зручний доступ до налаштувань застосунку, що зберігаються у файлі config.properties, та забезпечити їхню зміну без потреби перекомпіляції програми.

```
public class Config {
    private static final String CONFIG_FILE = "config.properties";
    private static Properties properties;

    static {
        properties = new Properties();
```

```
        try {
            properties.load(new FileInputStream(CONFIG_FILE));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String getWeatherApiKey() {
        return properties.getProperty("weather.apiKey", "");
    }

    public static String getWeatherCity() {
        return properties.getProperty("weather.city", "Київ");
    }

    public static String get(String key) {
        return properties.getProperty(key);
    }

    public static void set(String key, String value) {
        properties.setProperty(key, value);
        try (FileOutputStream fos = new FileOutputStream(CONFIG_FILE)) {
            properties.store(fos, null);
            AppLogger.log("Оновлено ключ: " + key + "=" + value);
        } catch (IOException e) { AppLogger.log(e); }
    }

}
```

Рис 4.2.2 Лістинг коду 12

В даному класі наявні такі методи як:

1) get(key) — зчитує значення будь-якого параметра.
2) set(key, value) — оновлює параметр у конфігураційному файлі.
3) getWeatherApiKey() — повертає ключ для API погоди.
4) getWeatherCity() — повертає місто, для якого отримується погода.

Клас **FormService** реалізує функціональність автоматичного формування службових документів для співробітників шляхом інтеграції Telegram-бота з базою даних та механізмами роботи з офісними файлами у форматі DOCX (Рис 4.2.3. Лістинг коду13) . Даний клас містить в собі такі основноні методи як:

1) createVacationForm(chatId) — формує заяву на відпустку.

```
public void createVacationForm(long chatId) {
    try {
        EmployeeInfo emp = getEmployeeInfo(chatId);
        if (emp == null) {
            bot.sendText(chatId, "❌ Не вдалося знайти ваш профіль у базі.");
            return;
        }

        XWPFDocument doc = new XWPFDocument();
        XWPFParagraph title = doc.createParagraph();
```

```
        title.setAlignment(ParagraphAlignment.CENTER);
        XWPFRun runTitle = title.createRun();
        runTitle.setBold(true);
        runTitle.setFontSize(14);
        runTitle.setText("ЗАЯВА");

        XWPFParagraph body = doc.createParagraph();
        body.setAlignment(ParagraphAlignment.LEFT);
        XWPFRun run = body.createRun();
        run.setFontSize(12);
        run.setText("\nПрошу надати мені відпустку з " +

LocalDate.now().plusDays(1).format(DateTimeFormatter.ofPattern("dd.MM.yyyy"))
+
                " по " +

LocalDate.now().plusDays(14).format(DateTimeFormatter.ofPattern("dd.MM.yyyy")
) +
                ".\n\n");

        run.setText("ПІБ: " + emp.fullName + "\n");
        run.setText("Посада: " + emp.position + "\n");
        run.setText("Дата подання: " +
LocalDate.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy")) + "\n");

        XWPFParagraph sign = doc.createParagraph();
        sign.setAlignment(ParagraphAlignment.RIGHT);
        XWPFRun runSign = sign.createRun();
        runSign.setText("Підпис: _____");

        String fileName = "Vacation_" + emp.fullName.replace(" ", "_") +
".docx";
        File file = new File(fileName);
        try (FileOutputStream fos = new FileOutputStream(file)) {
            doc.write(fos);
        }
        doc.close();

        SendDocument sendDoc = new SendDocument();
        sendDoc.setChatId(String.valueOf(chatId));
        sendDoc.setDocument(new InputFile(file));
        bot.execute(sendDoc);

        file.delete(); // можно убрать, если хочешь сохранять файлы

    } catch (Exception e) {
        e.printStackTrace();
        bot.sendText(chatId, "⚠ Помилка при створенні заяви на відпустку.");
    }
}
```

Рис 4.2.3. Лістинг коду13

createTripForm(chatId) — формує заяву на відрядження (Рис 4.2.4. Лістинг коду 14).

```
public void createTripForm(long chatId) {
    try {
        EmployeeInfo emp = getEmployeeInfo(chatId);
        if (emp == null) {
            bot.sendText(chatId, "❌ Не вдалося знайти ваш профіль у базі.");
            return;
        }
```

```
        XWPFDocument doc = new XWPFDocument();
        XWPFParagraph title = doc.createParagraph();
        title.setAlignment(ParagraphAlignment.CENTER);
        XWPFRun runTitle = title.createRun();
        runTitle.setBold(true);
        runTitle.setFontSize(14);
        runTitle.setText("ЗАЯВА НА ВІДРЯДЖЕННЯ");

        XWPFParagraph body = doc.createParagraph();
        body.setAlignment(ParagraphAlignment.LEFT);
        XWPFRun run = body.createRun();
        run.setFontSize(12);
        run.setText("\nПрошу направити мене у службове відрядження до м. Київ
" +
                "з " +
LocalDate.now().plusDays(1).format(DateTimeFormatter.ofPattern("dd.MM.yyyy"))
+
                " по " +
LocalDate.now().plusDays(5).format(DateTimeFormatter.ofPattern("dd.MM.yyyy"))
+
                " для виконання службових обов'язків.\n\n");

        run.setText("ПІБ: " + emp.fullName + "\n");
        run.setText("Посада: " + emp.position + "\n");
        run.setText("Дата подання: " +
LocalDate.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy")) + "\n");

        XWPFParagraph sign = doc.createParagraph();
        sign.setAlignment(ParagraphAlignment.RIGHT);
        XWPFRun runSign = sign.createRun();
        runSign.setText("Підпис: _____");

        String fileName = "Trip_" + emp.fullName.replace(" ", "_") + ".docx";
        File file = new File(fileName);
        try (FileOutputStream fos = new FileOutputStream(file)) {
            doc.write(fos);
        }
        doc.close();

        SendDocument sendDoc = new SendDocument();
        sendDoc.setChatId(String.valueOf(chatId));
        sendDoc.setDocument(new InputFile(file));
        bot.execute(sendDoc);

        file.delete();

    } catch (Exception e) {
        e.printStackTrace();
        bot.sendText(chatId, "⚠ Помилка при створенні заяви на
відрядження.");
    }
}
```

Рис 4.2.4. Лістинг коду14

2) getEmployeeInfo(chatId) — отримує дані співробітника з БД (ПІБ, посада, департамент)(Рис 4.2.5. Лістинг коду 15).

```
private EmployeeInfo getEmployeeInfo(long chatId) {
    try (Connection conn = Database.connect();
         PreparedStatement ps = conn.prepareStatement(
                 "SELECT e.full_name, e.position, d.name AS department " +
```

```
                        "FROM employees e " +
                        "LEFT JOIN departments d ON e.department_id = d.id "
+
                        "WHERE e.chat_id = ?"
         )) {
        ps.setString(1, String.valueOf(chatId));
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            EmployeeInfo info = new EmployeeInfo();
            info.fullName = rs.getString("full_name");
            info.position = rs.getString("position");
            info.department = rs.getString("department"); // теперь это
название департамента
            return info;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
```

Рис 4.2.5. Лістинг коду 15

Клас **ITSupportService** є центральним компонентом бізнес-логіки підсистеми обробки заявок технічної підтримки в Telegram-боті (Рис 4.2.6. Лістинг коду 16). Він відповідає за повний життєвий цикл IT-заявки — від створення чернетки користувачем до її закриття спеціалістом та подальшої оцінки якості виконання. Клас забезпечує інтеграцію користувацьких дій із базою даних, Telegram API та механізмами взаємодії з інтерфейсом у вигляді інлайн-кнопок. Даний клас має в собі такі основні методи:

1) showMenu(chatId) — показує головне меню IT із статистикою (нові заявки, архів, середній рейтинг).

```
public void showMenu(long chatId) {
    if (!isItSupport(chatId)) {
        sendSimple(chatId, "📝 Ви можете лише залишити заявку. Меню IT
недоступне.");
        return;
    }
    String fullName = getEmployeeFullName(chatId);
    if (fullName == null) fullName = "Співробітник";

    // Витягуємо статистику з БД
    int newCount = Database.getTicketCountByStatus(chatId, "OPEN");
    int archiveCount = Database.getTicketCountByStatus(chatId, "CLOSED");
    double avgRating = Database.getAverageRating(chatId);

    String text = "💻 Вітаю, " + fullName + "!\n\n" +
            "📊 Статистика:\n" +
            "🔘 Нових заявок: " + newCount + "\n" +
            "📂 Архів (закритих): " + archiveCount + "\n" +
            "⭐ Середній рейтинг: " +
            (avgRating > 0 ? String.format("%.2f", avgRating) : "нема
```

```
оцінок") +
            "\n\nОберіть дію:";

    SendMessage msg = new SendMessage(String.valueOf(chatId), text);

    // Кнопки динамічно з урахуванням кількості
    InlineKeyboardMarkup markup = new InlineKeyboardMarkup();

    InlineKeyboardButton btnNew = InlineKeyboardButton.builder()
            .text("📄 Нові заявки (" + newCount + ")")
            .callbackData("menu_new")
            .build();

    InlineKeyboardButton btnArchive = InlineKeyboardButton.builder()
            .text("🗂 Архів (" + archiveCount + ")")
            .callbackData("menu_archive")
            .build();

    InlineKeyboardButton btnReport = InlineKeyboardButton.builder()
            .text("📊 Звіт за місяць")
            .callbackData("menu_report")
            .build();

    markup.setKeyboard(List.of(
            List.of(btnNew, btnArchive),
            List.of(btnReport)
    ));

    msg.setReplyMarkup(markup);

    executeSafe(msg);
}
```

Рис 4.2.6 Лістинг коду 16

2) showCreateTicketMenu(chatId) — створює нову чернетку заявки.

3) appendToDraft(chatId, text) — додає повідомлення до чернетки.

4) addDraftAttachmentFileId(chatId, fileId) — додає файл до чернетки.

5) showITSelection(chatId) — показує список IT-співробітників для призначення заявки.

6) handleDraftRecipient(chatId, selectedEmployeeId, userStates) — обробляє вибір IT-співробітника та зберігає заявку в БД.

7) closeTicket(ticketId, itChatId) — закриває заявку та пропонує оцінку користувачу.

8) sendMonthlyReport(managerChatId, month, year) — експортує заявки за місяць у Excel.

9) handleCallbackQuery(callbackQuery) — обробка callback-подій (закриття заявки, оцінка, вибір IT, меню).

Клас **OrderService** реалізує бізнес-логіку обробки товарів та

оформлення замовлень у Telegram-боті. Він забезпечує взаємодію користувача зі складською системою магазину, включаючи пошук товару за штрихкодом, перегляд інформації про товар, перегляд наявності товару та оформлення замовлення. Клас також підтримує розпізнавання штрихкодів із фото, що значно підвищує зручність користувача. Данйи клас включає в собі такі основні методи як:

1) handleBarcode(chatId, barcode) — обробляє текстовий штрихкод, показує інформацію про товар (Рис 4.2.7. Лістинг коду 17).

```
public void handleBarcode(long chatId, String barcode) {
    Product product = Database.getProductByBarcode(barcode);
    if (product == null) {
        bot.sendText(chatId, "❌ Товар зі штрихкодом " + barcode + " не
знайдено.");
        return;
    }
```

Рис 4.2.7. Лістинг коду 17

2) handlePhotoBarcode(chatId, photos) — розпізнає штрихкод із фотографії (Рис 4.2.8. Лістинг коду 18).

```
public void handlePhotoBarcode(long chatId, List<PhotoSize> photos) {
    try {
        PhotoSize largest = photos.stream()
                .max(Comparator.comparing(PhotoSize::getFileSize))
                .orElse(null);

        if (largest == null) {
            bot.sendText(chatId, "❌ Не вдалося отримати фото.");
            return;
        }
```

Рис 4.2.8. Лістинг коду 18

3) handleCallback(callback) — обробляє натискання inline-кнопок (Рис 4.2.9. Лістинг коду 19).

```
public void handleCallback(CallbackQuery cb) {
    String data = cb.getData();
    long chatId = cb.getMessage().getChatId();

    if (data.startsWith("order_")) {
        int productId = Integer.parseInt(data.split("_")[1]);
        // Сохраняем товар и переводим пользователя в состояние ожидания
количества
        pendingOrders.put(chatId, productId);
        userStates.put(chatId, UserState.WAITING_FOR_ORDER_QUANTITY);
        bot.sendText(chatId, "📄 Введіть кількість для замовлення:");
        return;
    }

    if (data.startsWith("stock_")) {
        int productId = Integer.parseInt(data.split("_")[1]);
```

```
        int stock = Database.getStockByProductId(productId);
        bot.sendText(chatId, "📊 Залишок по товару: " + stock + " шт.");
        return;
    }

    if ("main_menu".equals(data)) {
        userStates.put(chatId, UserState.MAIN_MENU);
        bot.showMainMenu(chatId);
    }
```

Рис 4.2.9. Лістинг коду 19

4) handleQuantityInput(chatId, text) — обробляє введення кількості товару та створює замовлення (Рис 4.2.10 Лістинг коду20).

```
public void handleQuantityInput(long chatId, String text) {
    if (userStates.getOrDefault(chatId, UserState.MAIN_MENU) !=
UserState.WAITING_FOR_ORDER_QUANTITY) return;

    try {
        int quantity = Integer.parseInt(text);
        if (quantity <= 0) throw new NumberFormatException();

        Integer productId = pendingOrders.get(chatId);
        if (productId == null) {
            bot.sendText(chatId, "❌ Сталася помилка. Спробуйте ще раз.");
            userStates.put(chatId, UserState.MAIN_MENU);
            return;
        }

        int empId = Database.getEmployeeIdByChatId(chatId);
        if (Database.createOrder(empId, productId, quantity)) {
            bot.sendText(chatId, "✅ Замовлення на " + quantity + " шт.
оформлено!");
        } else {
            bot.sendText(chatId, "❌ Помилка при створенні замовлення.");
        }

        pendingOrders.remove(chatId);
        userStates.put(chatId, UserState.MAIN_MENU);
        bot.showMainMenu(chatId);

    } catch (NumberFormatException e) {
        bot.sendText(chatId, "❌ Будь ласка, введіть коректне число для
кількості.");
    }
```

Рис 4.2.10. Лістинг коду 20

Клас **TicketDraft** реалізує модель чернетки службового звернення (тикету) в системі. Його основне призначення — тимчасово зберігати дані, які користувач або співробітник вводить перед остаточним створенням та відправленням тикету. Даний клас включає такі основні методи, як:

1) getMessage() — задає повний текст звернення (Рис 4.2.11. Лістинг коду21).

```
public String getMessage() { return message.toString(); }

public void setMessage(String msg) {
    message.setLength(0);
    if (msg != null) message.append(msg);
}
```

Рис 4.2.11. Лістинг коду 21

2) appendMessage() — додає рядок у кінець звернення (Рис 4.2.12. Лістинг коду 22).

```
public void appendMessage(String msg) {
    if (msg != null && !msg.isEmpty()) {
        if (message.length() > 0) message.append("\n");
        message.append(msg);
    }
}
```

Рис 4.2.12. Лістинг коду 22

3) addAttachmentFileId(fileId) — додає ID вкладення (фото, документ) (Рис 4.2.13.Лістинг коду 23).

```
public void addAttachmentFileId(String fileId) {
    if (fileId != null && !fileId.isEmpty()) attachmentsFileIds.add(fileId);
}
```

Рис 4.2.13. Лістинг коду 23

4) getAttachmentsFileIds() — повертає список прикріплених файлів (Рис 4.2.14. Лістинг коду 24).

```
public List<String> getAttachmentsFileIds() { return attachmentsFileIds; }
```

Рис 4.2.14. Лістинг коду 24

5) clear() — очищає чернетку (текст, файли, ID співробітника) (Рис 4.2.15. Лістинг коду 25).

```
public void clear() {
    message.setLength(0);
    attachmentsFileIds.clear();
    employeeId = 0;
}
```

Рис 4.2.15. Лістинг коду 25

Клас **WeatherService** реалізує сервіс отримання та форматування даних про погоду із зовнішнього API OpenWeatherMap. Його основним призначенням є надання користувачеві актуального прогнозу погоди у зручному текстовому форматі з локалізованими назвами днів тижня, емодзі-піктограмами та іншими характеристиками. Даний клас містить в собі такі методи:

1) getWeather()- отримує або повертає закешований прогноз погоди;

враховує оновлення раз на 10 хвилин (Рис 4.2.16. Лістинг коду 26).

```
public synchronized String getWeather() {
    long now = System.currentTimeMillis();
    LocalDate today = LocalDate.now();

    if (cachedData != null && cachedDate != null
            && cachedDate.equals(today)
            && (now - lastFetch < CACHE_DURATION_MS)) {
        return cachedData + "\n\n(Оновлюється раз на 10 хвилин)";
    }

    try {
        ...

        return cachedData;

    } catch (Exception e) {
        e.printStackTrace();
        return "❌ Не вдалося отримати погоду";
    }
}
```

Рис 4.2.16. Лістинг коду 26

2) formatEntry(entry, dateTime)- форматує окремий запис погоди (температура, вітер, хмарність) (Рис 4.2.17. Лістинг коду 27).

```
private String formatEntry(JSONObject entry, LocalDateTime dateTime) {
    JSONObject main = entry.getJSONObject("main");
    JSONArray weatherArr = entry.getJSONArray("weather");
    JSONObject weather = weatherArr.getJSONObject(0);
    JSONObject wind = entry.getJSONObject("wind");

    double temp = main.getDouble("temp");
    int humidity = main.getInt("humidity");
    String desc = weather.getString("description").toLowerCase();
    double windSpeed = wind.getDouble("speed");

    String emoji = switch (desc) {
        case "ясно","чисте небо" -> "☀";
        case "хмарно", "легка хмарність", "рвані хмари", "уривчасті хмари", "кілька
хмар" -> "☁";
        case "дощ", "легкий дощ", "помірний дощ" -> "🌧";
        case "сніг" -> "❄";
        case "туман" -> "🌫";
        case "гроза" -> "⛈";
        default -> "";
    };
```

Рис 4.2.17. Лістинг коду 27

3) setCity(city)- змінює місто та скидає кеш (Рис 4.2.18. Лістинг коду 28).

```
public void setCity(String city) {
    if (city != null && !city.trim().isEmpty()) {
        this.city = city.trim();
    }
    this.cachedData = null; // очищаємо кеш
}
```

Рис 4.2.18. Лістинг коду 28

Отже, розроблена бізнес-логіка боту забезпечують повний цикл

взаємодії користувача з системою. Класи, які її реалізують, відповідають за обробку повідомлень, управління станами користувачів, збереження проміжних даних, формування службових запитів і роботу з базою даних. Кожен клас має визначену відповідальність: від зберігання та обробки чернетки заявки до маршрутизації команд, управління сеансами та взаємодії з файлами. Даний підхід дозволяє досягти розподілу обов'язків, підвищити гнучкість і спростити підтримку проекту. У результаті розроблена бізнес-логіка забезпечує надійне, передбачуване та зручне функціонування Telegram-боту, що є важливим чинником для його подальшої експлуатації та розширення.

4.3. Створення Telegram-бота

Розпочинати розробку самого бота будем з створення класу LogicBot, який і буде виступати як сам телеграм бот, в який додаємо поля — сервіси та карту станів користувачів (Рис 4.3.1. Лістинг коду 29).

```
public class LogicBot extends TelegramLongPollingBot {

    private final FormService formService;
    private final WeatherService weatherService;
    private final ITSupportService itService;
    private final OrderService orderService;
    private final Map<Long, UserState> userStates = new HashMap<>();
```

Рис 4.3.1. Лістинг коду 29

Далі у конструкторі класу створимо екземпляри служб та передаємо необхідні параметри: API-ключ погоди, посилання на поточний борт так картку станів (Рис 4.3.2. Лістинг коду 30).

```
public LogicBot(String weatherApiKey) {
    this.weatherService = new WeatherService(weatherApiKey);
    this.itService = new ITSupportService(this, userStates);
    this.orderService = new OrderService(this);
    this.formService = new FormService(this);
}
```

Рис 4.3.2. Лістинг коду 30

Для повноцінної інтеграції з платформою Telegram були перевизначені методи getBotUsername(), getBotToken() та onUpdateReceived()(Рис 4.. Лістинг коду 31). Перші два методи забезпечують ідентифікацію бота, а третій — прийом і первинну обробку всіх вхідних подій.

```
@Override
public String getBotUsername() {
    return "FishingCompany";
}

@Override
public String getBotToken() {
    return "7954415758:AAGdI-ctoT6-UMCWCGdVppj37bXZJo5J-RQ";
}
```

Рис 4.3.3. Лістинг коду 31

Після чого переходим до реалізації методу onUpdateReceived(), який реалізу повноцінну бізнес-логіку(Рис 4.3.4. Лістинг коду 32), тобто в ньому:

- ✓ Обробляється callback-кнопок таких як: замовлення, склад, заяви, паркомісця;
- ✓ Авторизація користувача за номером телефону;
- ✓ Обробка станів машини станів, таких як: створення IT-тикету, вибір парковки, введення штрихкоду товару;
- ✓ Формування меню та підменю;
- ✓ Маршрутизація запитів до відповідних сервісів;

Для демонстрації наведу реалізацію одного методу, так як реалізація всієї бізнес логіки знаходиться в окремому рівні архітектури і її вже було розглянуто.

```
@Override
    public void onUpdateReceived(Update update) {
        if (update.hasCallbackQuery()) {
            CallbackQuery cb = update.getCallbackQuery();
            if (cb.getData().startsWith("order_") ||
cb.getData().startsWith("stock_") || cb.getData().equals("main_menu")) {
                orderService.handleCallback(cb);
                return;
            }
            String data = cb.getData();

            if (data.startsWith("park_")) {
                int placeNumber = Integer.parseInt(data.replace("park_",
""));
                handleParkingSelection(cb.getMessage().getChatId(),
cb.getFrom().getUserName(), placeNumber);
                return;
            }

// Обработка кнопок из раздела "Заяви"
            if (data.equals("form_vacation")) {
                sendMessage(cb.getMessage().getChatId(), "🏖 Створюю заяву на
відпустку...");
                formService.createVacationForm(cb.getMessage().getChatId());
                return;
            }
```

```
            if (data.equals("form_trip")) {
                sendMessage(cb.getMessage().getChatId(), "🚗 Створюю заяву на
відрядження...");
                formService.createTripForm(cb.getMessage().getChatId());
                return;
            }


            itService.handleCallbackQuery(cb);
            return;
        }

        if (!update.hasMessage()) return;
        Message msg = update.getMessage();
        long chatId = msg.getChatId();
```

Рис 4.3.4. Лістинг коду 32

Після того як основний метод onUpdateReceived(), був повністю реалізований переходим до реалізації допоміжних бізнес функцій, таких як:

1) Пошук співробітника у базі даних

Метод здійснює пошук за ПІБ або номером телефону, формує структуровану відповідь та повертає її користувачу (Рис 4.3.5. Лістинг коду 33).

```
private void searchEmployee(long chatId, String query) {
    String sql = "SELECT full_name, phone, email, role FROM employees WHERE
full_name LIKE ? OR phone LIKE ?";
    try (Connection conn = Database.connect();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, "%" + query + "%");
        ps.setString(2, "%" + query + "%");
        ResultSet rs = ps.executeQuery();
        StringBuilder sb = new StringBuilder();
        while (rs.next()) {
            sb.append("ФИО: ").append(rs.getString("full_name")).append("\n")
                    .append("Посада:
").append(rs.getString("position")).append("\n")
                    .append("Телефон:
").append(rs.getString("phone")).append("\n")
                    .append("Пошта:
").append(rs.getString("email")).append("\n");
        }
        if (sb.length() == 0) sendMessage(chatId, "Сотрудник не найден.");
        else sendMessage(chatId, sb.toString());
    } catch (SQLException e) { e.printStackTrace(); sendMessage(chatId, "❌
Ошибка поиска."); }
}
```

Рис 4.3.5. Лістинг коду 33

2) Робота з заявами

Метод формує клавіатуру вибору типів заяв та відправляє її користувачу(Рис 4.3.6. Лістинг коду 34).

```
private void showAvailableApplications(long chatId) {
    String text = "📄 Оберіть тип заяви:";

    InlineKeyboardMarkup markup = new InlineKeyboardMarkup(List.of(
            List.of(
                    InlineKeyboardButton.builder()
                            .text("🏖 Заява на відпустку")
                            .callbackData("form_vacation")
                            .build()
            ),
            List.of(
                    InlineKeyboardButton.builder()
                            .text("🚗 Заява на відрядження")
                            .callbackData("form_trip")
                            .build()
            ),
            List.of(
                    InlineKeyboardButton.builder()
                            .text("🔙 Головне меню")
                            .callbackData("main_menu")
                            .build()
            )
    ));

    SendMessage msg = new SendMessage(String.valueOf(chatId), text);
    msg.setReplyMarkup(markup);
    try {
        execute(msg);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Рис 4.3.6. Лістинг коду 34

3) Авторизація за телефоном

Метод перевіряє, чи існує користувач у базі даних та прив’язує Telegram-номер до його облікового запису(Рис 4.3.7. Лістинг коду 35).

```
private boolean authorizeTGPhone(String tgPhone, long chatId) {
    String normalizedTG = normalizePhoneFull(tgPhone);
    try (Connection conn = Database.connect()) {
        String sql = "SELECT id, phone, phone_TG FROM employees";
        try (Statement st = conn.createStatement(); ResultSet rs =
st.executeQuery(sql)) {
            while (rs.next()) {
                int id = rs.getInt("id");
                String dbPhone = rs.getString("phone");
                String dbTG = rs.getString("phone_TG");
                if (normalizedTG.equals(normalizePhoneFull(dbPhone)) ||
normalizedTG.equals(normalizePhoneFull(dbTG))) {
                    try (PreparedStatement update =
conn.prepareStatement("UPDATE employees SET phone_TG = ?, chat_id = ? WHERE
id = ?")) {
                        update.setString(1, tgPhone);
                        update.setString(2, String.valueOf(chatId));
                        update.setInt(3, id);
                        update.executeUpdate();
                    }
                    return true;
```

```
                }
            }
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return false;
}
```

Рис 4.3.7. Лістинг коду 35

Окрім наведених функцій, у класі LogicBot були реалізовані й інші допоміжні методи, що забезпечують обробку бізнес-логіки:

1) керування замовленнями;
2) робота з IT-тикетами;
3) управління паркомісцями;
4) обробка штрихкодів товарів.

Всі методи формують повноцінний функціональний модуль Telegram-бота, який взаємодіє з базою даних, користувачами та різними сервісами системи.

4.4. Створення панелі адміністрування Telegram-бота

Починати розробку панелі адміністрування з створення класу AdminController (Рис 4.4.1. Лістинг коду 36). Даний клас відповідає за керування графічним інтерфейсом панелі адміністрування, що забезпечує роботу з таблицями бази даних, редагуванням записів, переглядом логів та конфігураційними параметрами.

```
public class AdminController {

    @FXML private TableView<Map<String, Object>> dbTable;
    @FXML private ComboBox<String> tableSelector;
    @FXML private TextField dbPathField;
    @FXML private TextArea apiKeysArea;
    @FXML private TextArea logArea;

    @FXML private Button btnAdd;
    @FXML private Button btnEdit;
    @FXML private Button btnDelete;
    @FXML private Button btnSavePath;
    @FXML private Button btnSaveKeys;
    @FXML private Button btnRefreshLog;
```

Рис 4.4.1. Лістинг коду 36

Наступною дією реалізації адмін. панелі буде створення методу initialize(), який завантажує назви з таблиці, встановлює дію при зміні вибору

таблиці, прив’язує дії до кнопок: додати, редагувати, видалити, оновити логи, зберегти шлях і ключі та встановлює початковий шлях(Рис 4.4.2. Лістинг коду 37).

```
public void initialize() {
    dbPathField.setText("jdbc:sqlite:company.db");
    tableSelector.setItems(FXCollections.observableArrayList(tableNames));
    tableSelector.setOnAction(e -> loadTableData());

    btnSavePath.setOnAction(e -> saveDbPath());
    btnSaveKeys.setOnAction(e -> saveApiKeys());
    btnRefreshLog.setOnAction(e -> refreshLog());
    btnAdd.setOnAction(e -> addRecord());
    btnEdit.setOnAction(e -> editRecord());
    btnDelete.setOnAction(e -> deleteRecord());
}
```

Рис 4.4.2. Лістинг коду 37

Далалі розробляємо метод loadTableData() який отримує назву вибраної таблиці, виконує SQL-запит і динамічно формує колонки TableView, що дозволяє працювати з будь-якою таблицею без змін коду (Рис 4.4.3. Лістинг коду 38).

```
private void loadTableData() {
    String table = tableSelector.getValue();
    if (table == null) return;

    try (Connection conn = Database.connect();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery("SELECT * FROM " + table)) {

        // --- Отримуємо назви колонок ---
        ResultSetMetaData meta = rs.getMetaData();
        int cols = meta.getColumnCount();
        List<String> columnNames = new ArrayList<>();
        for (int i = 1; i <= cols; i++) {
            columnNames.add(meta.getColumnName(i));
        }

        // --- Створюємо колонки TableView ---
        dbTable.getColumns().clear();
        for (String colName : columnNames) {
            TableColumn<Map<String, Object>, String> col = new
TableColumn<>(colName);
            col.setCellValueFactory(cell -> {
                Object val = cell.getValue().get(colName);
                return new SimpleStringProperty(val == null ? "" :
val.toString());
            });
            dbTable.getColumns().add(col);
        }

        // --- Читаємо всі дані ---
        ObservableList<Map<String, Object>> data =
FXCollections.observableArrayList();
        while (rs.next()) {
```

```
            Map<String, Object> row = new HashMap<>();
            for (String colName : columnNames) {
                row.put(colName, rs.getObject(colName));
            }
            data.add(row);
        }

        dbTable.setItems(data);

    } catch (SQLException e) {
        log(e);
        showInfo("Помилка при завантаженні таблиці: " + e.getMessage());
    }
}
```

Рис 4.4.3. Лістинг коду 38

Після реалізації метод loadTableData(), створюємо метод addRecord(), який перевіряє чи обрана таблиця та відкриває діалог без початкових даних (Рис 4.4.4. Лістинг коду 39).

```
private void addRecord() {
    String table = tableSelector.getValue();
    if (table == null) { showInfo("Виберіть таблицю!"); return; }
    showEditDialog(table, null);
}
```

Рис 4.4.4. Лістинг коду 39

Після реалізуєм метод editRecord(), котрий отримує виділений запис з таблиці і якщо запис вибрано, то відкриває діалог з попередньо заповненими полями, та метод deleteRecord(),котрий перевіряє, чи вибрано таблицю та запис, виконує SQL-запит та оновлює дані таблиці після видалення (Рис 4.4.5. Лістинг коду 40).

```
private void editRecord() {
    String table = tableSelector.getValue();
    if (table == null) { showInfo("Виберіть таблицю!"); return; }
    Map<String,Object> row = dbTable.getSelectionModel().getSelectedItem();
    if (row == null) { showInfo("Виберіть запис для редагування"); return; }
    showEditDialog(table, row);
}
private void deleteRecord() {
    String table = tableSelector.getValue();
    if (table == null) { showInfo("Виберіть таблицю!"); return; }
    Map<String,Object> row = dbTable.getSelectionModel().getSelectedItem();
    if (row == null) { showInfo("Виберіть запис для видалення"); return; }

    try (Connection conn = Database.connect();
         PreparedStatement ps = conn.prepareStatement("DELETE FROM "+table+"
WHERE id=?")) {
        ps.setObject(1, row.get("id"));
        int res = ps.executeUpdate();
        showInfo("Видалено записів: " + res);
        loadTableData();
    } catch (SQLException e) {
```

```
            log(e);
            showInfo("Помилка при видаленні: " + e.getMessage());
        }
    }
```

Рис 4.4.5. Лістинг коду 40

Далі реалізуєм метод showEditDialog() створює динамічну форму для внесення даних (Рис 4.4.6. Лістинг коду 41). Даний метод отримує структуру таблиці, створює форму, відображає її у діалоговому вікні та вставляє дані в таблицю при збереженні.

```
private void showEditDialog(String table, Map<String,Object> row) {
    try (Connection conn = Database.connect();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery("SELECT * FROM " + table + " LIMIT
1")) {

        ResultSetMetaData meta = rs.getMetaData();
        int cols = meta.getColumnCount();
        List<String> columnNames = new ArrayList<>();
        for (int i = 1; i <= cols; i++)
columnNames.add(meta.getColumnName(i));

        Dialog<Map<String,Object>> dialog = new Dialog<>();
        dialog.setTitle(row == null ? "Додати запис" : "Редагувати запис");
        ButtonType okButtonType = new ButtonType("Зберегти",
ButtonBar.ButtonData.OK_DONE);
        dialog.getDialogPane().getButtonTypes().addAll(okButtonType,
ButtonType.CANCEL);

        GridPane grid = new GridPane();
        grid.setVgap(5);
        grid.setHgap(10);
        Map<String, TextField> fields = new HashMap<>();

        int rowIndex = 0;
        for (String colName : columnNames) {
            if (colName.equalsIgnoreCase("id")) continue;
            Label label = new Label(colName);
            TextField tf = new TextField();
            if (row != null && row.get(colName) != null)
tf.setText(row.get(colName).toString());
            fields.put(colName, tf);
            grid.add(label, 0, rowIndex);
            grid.add(tf, 1, rowIndex);
            rowIndex++;
        }

        dialog.getDialogPane().setContent(grid);
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == okButtonType) {
                Map<String,Object> result = new HashMap<>();
                fields.forEach((k,v)-> result.put(k, v.getText()));
                return result;
            }
            return null;
        });
```

```
        Optional<Map<String,Object>> result = dialog.showAndWait();
        result.ifPresent(data -> {
            try {
                if (row == null) {
                    // Insert
                    StringBuilder colsPart = new StringBuilder();
                    StringBuilder valsPart = new StringBuilder();
                    for (String col : data.keySet()) {
                        colsPart.append(col).append(",");
                        valsPart.append("?,");
                    }
                    colsPart.setLength(colsPart.length()-1);
                    valsPart.setLength(valsPart.length()-1);
                    String sql = "INSERT INTO "+table+" ("+colsPart+") VALUES
("+valsPart+")";
                    try (PreparedStatement ps = conn.prepareStatement(sql)) {
                        int idx=1;
                        for (String col : data.keySet()) ps.setString(idx++,
data.get(col).toString());
                        ps.executeUpdate();
                    }
                } else {
                    // Update
                    StringBuilder setPart = new StringBuilder();
                    for (String col : data.keySet())
setPart.append(col).append("=?,");
                    setPart.setLength(setPart.length()-1);
                    String sql = "UPDATE "+table+" SET "+setPart+" WHERE
id=?";
                    try (PreparedStatement ps = conn.prepareStatement(sql)) {
                        int idx=1;
                        for (String col : data.keySet()) ps.setString(idx++,
data.get(col).toString());
                        ps.setObject(idx, row.get("id"));
                        ps.executeUpdate();
                    }
                }
                loadTableData();
            } catch (SQLException e) {
                log(e);
                showInfo("Помилка при збереженні запису: " + e.getMessage());
            }
        });

    } catch (SQLException e){
        log(e);
        showInfo("Помилка при відкритті діалогу: " + e.getMessage());
    }
}
```

Рис 4.4.6. Лістинг коду 41

На прикінці реалізуєм службові методи, котрі показують інформаційне вікно та записують помилки через AppLogger і виводить трасування(Рис 4.4.7. Лістинг коду 42).

```
private void showInfo(String msg){
    Alert alert = new Alert(Alert.AlertType.INFORMATION, msg, ButtonType.OK);
    alert.showAndWait();
}
```

```
private void log(Exception e){
    AppLogger.log(e);
    e.printStackTrace();
}
```

Рис 4.4.7. Лістинг коду 42

В результаті було створено панель адміністрування Telegram-бота, котра забезпечує централізований контроль над усіма процесами системи. Завдяки такій структурі, створили гнучку, модульну та масштабовану архітектуру, у якій кожен компонент може незалежно доповнюватися або змінюватися. Панель адміністрування робить роботу з ботом значно ефективнішою, а майбутнє масштабування та трансформацію значно простішими та швидшими.

4.5. Результат прототипування

В результаті прототипування телеграм-боту BotFishShop було було отримано структуровану і масштабовану архітектуру (Рис 4.5.1), що включає:

- ✓ bot-рівень для взаємодії з Telegram API;
- ✓ config-сервіси для реалізації бізнес-логіки;
- ✓ db-шар для доступу до зашифрованої бази даних;
- ✓ tools - що містять допоміжні утиліти та панель адміністратора.

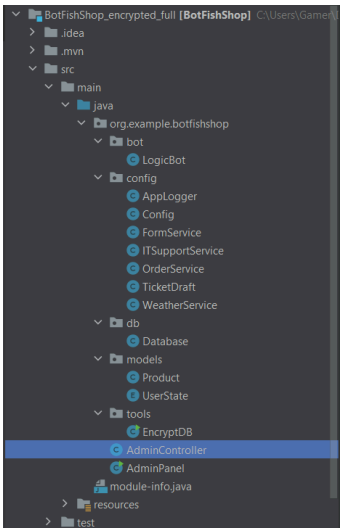


Рис 4.5.1. Структура проекту

Створена структура проекту забезпечує чітке розмежування на логічні модулі, що дає змогу чітко розмежувати відповідальності між класами та уникнути змішування бізнес-логіки з технічною реалізацією. Така організація

робить систему значно простішою для розуміння та розширення новими функціями. У результаті структура проекту виходить масштабованою, безпечною та добре організованою, що позитивно впливає на якість і швидкість розробки, а також позитивно впливає на масштабованість і легкість інтеграції нового функціоналу. Розроблений прототип після компіляції має такий вигляд:

1) Панель адміністрування (Рис 4.5.2)

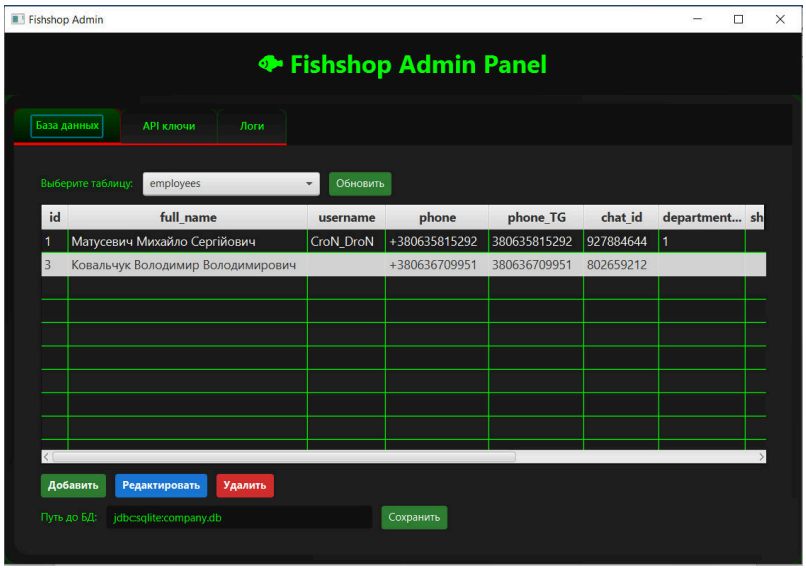


Рис 4.5.1. Панель адміністрування

Як можем бачити на рис. 4.5.2 панель адміністрування має три ключових вікна, таких як:

- ✓ База даних;
- ✓ API ключі;
- ✓ Логи.

У вікні база даних ми маємо випливаюче вікно вибору необхідної таблиці (Рис. 4.5.3).

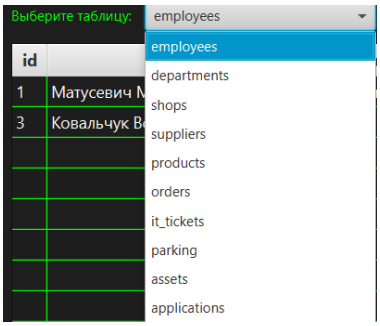


Рис. 4.5.2. Випливаюче вікно вибору таблиці

Та п’ять функціональних кнопок, таких як:

1) Обновить.
2) Додати.

При натисканні кнопки «додати», перед адміністратором відкривається вікно (Рис. 4.5.4) «додавання запису», в якому необхідно вручну вввести всі необхідні дані для реєстрації запису.

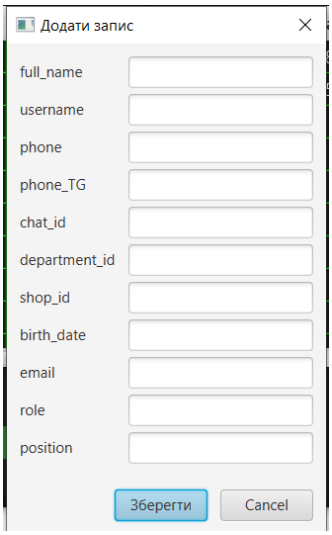


Рис. 4.5.3. Вікно додавання запису

3) Редагувати.

Для редагування запису, необхідно обрати вже наявну запис та натиснути клавішу «Редагувати», після чого перед адміністратором з’явиться вікно «Редагування запису»(Рис. 4.5.5), де можна змінити дані та натиснути «зберегти» після введення актуальної інформації

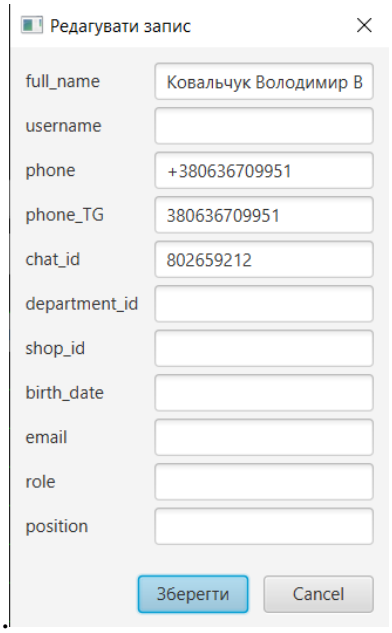


Рис. 4.4.5. Вікно редагування запису

В разі якщо запис не був вибраний то адмін панель видасть помилку (Рис. 4.5.6)

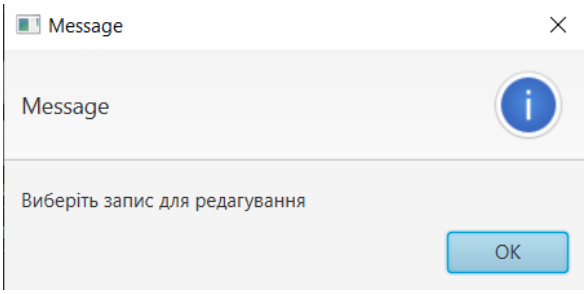


Рис. 4.5.5. Попереження що слід вибрати запис

4) Видалити.

Для видалення слід вибрати запис, в разі якщо запис не був обраний то панель видає попередження (Рис. 4.5.7), та натиснути кнопку «видалити», після чого адміністративна панель видасть повідомлення(Рис. 4.5.7).

Рис. 4.5.6. Повідомлення про видалення запису

5) Зберегти.

2) Корпоративний телеграм бот.

Для початку роботи з телеграм ботом слід прописати команду «\start», після чого телеграм бот запросить мобільний номер телефону для авторизації (Рис. 4.5.8).

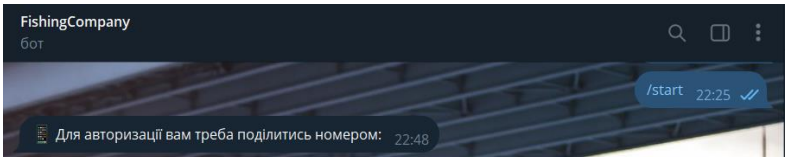


Рис. 4.5.7. Початок роботи з ботом.

В разі якщо користувач не знайдений то бот надасть повідомлення «Ваш номер не знайдено. Прошу звернутися до адміністратора », а в разі успішної авторизації бот надасть повідомлення «Вітаю. Доступ до функцій відкритий».

Після авторизації перед користувачем з'явиться меню(Рис. 4.9).

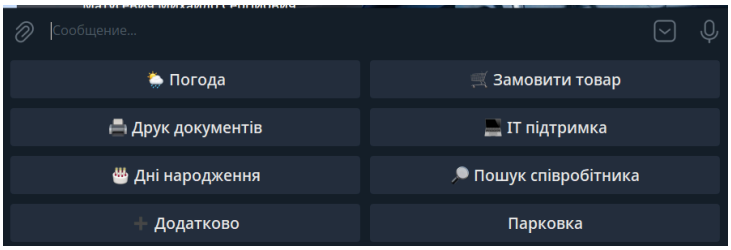


Рис. 4.5.8. Меню вибору функціоналу

Як можемо бачити меню надає 8 кнопок, таких як:

1) Погода

При натисканні на погоду бот надає інформацію про погоду(Рис. 4.5.10)

Рис. 4.5.9. Відповідь на кнопку «Погода»

2) Замовити товар

При натисканні на кнопку «замовити товар» бот просить надати код (Рис. 4.5.11)

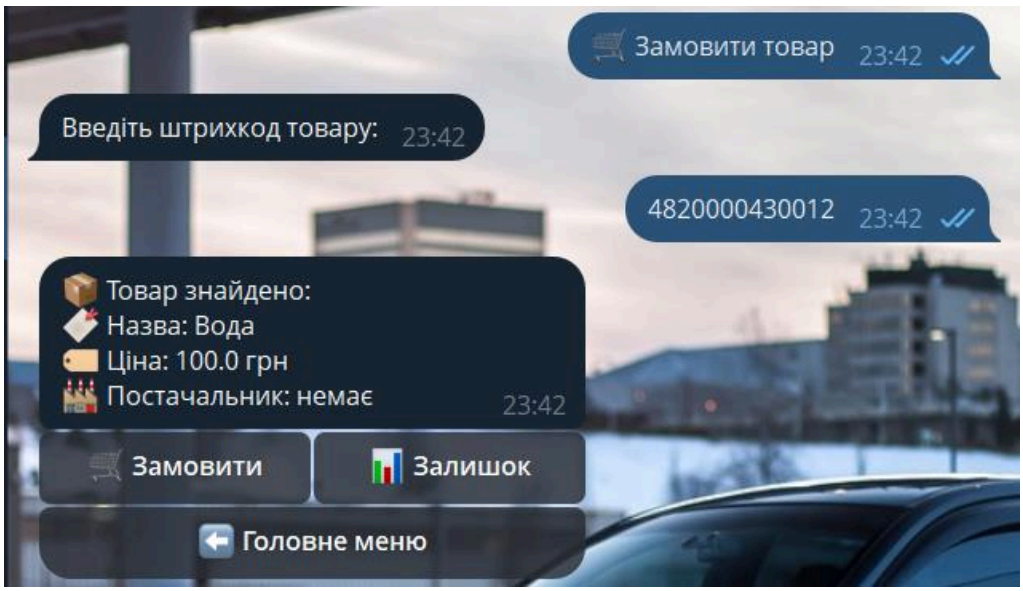


Рис. 4.5.10. Замовлення товару

3) IT-підтримка

При натисканні на кнопку «IT підтримка» бот просить надати опис проблеми, фото або документ. Після того як опис був наданий натискаєм на кнопку «Додати ще», після натискання отримуємо список фахівців які можуть допомогти у вирішенні проблеми, обираємо фахівця і отримаємо відповідь про те що заявка створена (Рис. 4.5.12).

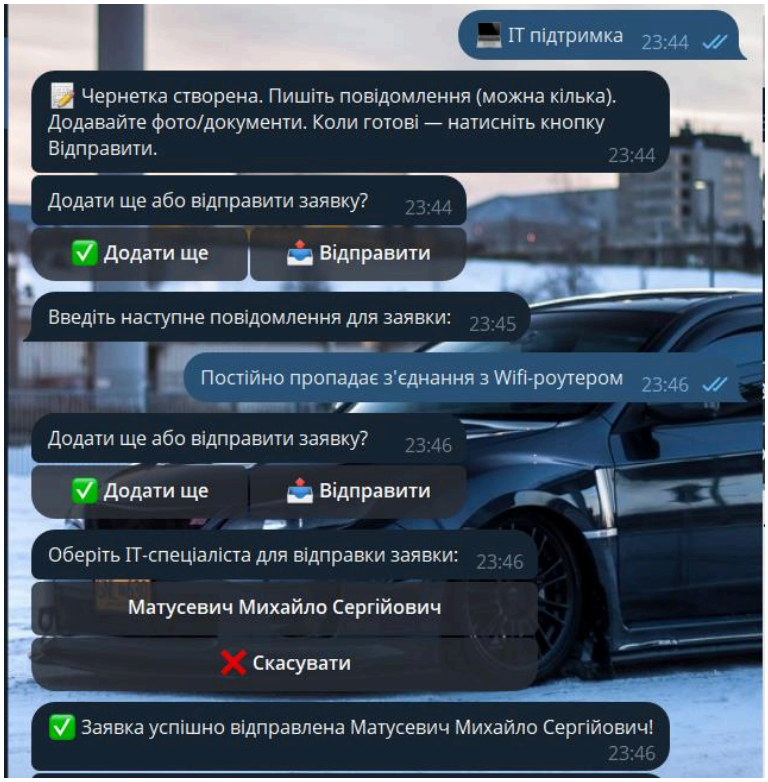


Рис. 4.5.11. Створення заявки на IT підтримку

4) Дні народження

При натисканні на кнопку «Дні народження», бот надає імена та дати народження колег які цього місяця святкують день народження, в разі якщо в місяці немає днів народжень бот видає повідомлення: «У цьому місяці немає днів народження» (Рис. 4.5.13).

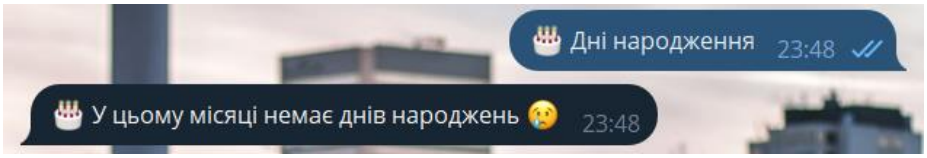


Рис. 4.5.12. Реакція бота на кнопку "Дні народження"

5) Пошук співробітника

При обранні кнопки «Пошук співробітників», бот просить ввести ім’я або номер телефону співробітника, після веденння даних бот надає ФІО працівника, посаду, телефон та пошту (Рис. 4.5.14).

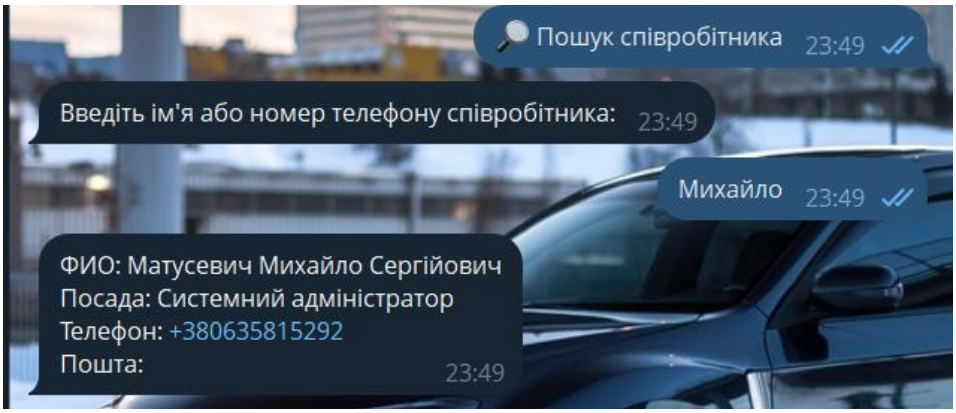


Рис. 4.13. Пошук співробітників

6) Парковка

При натисканні кнопки «Парковка», бот видає інформацію про кількість місць на парковці та надає інформацію про зайняті місця та ким вони зайняті та просить обрати місце, при обранні зайнятого місця бот видає «Місце вже зайняте», та повторно надає можливість обрати місце, при обранні вільного місця бот видає інформацію про те що місце успішно було зайняте(Рис. 4.5.15).

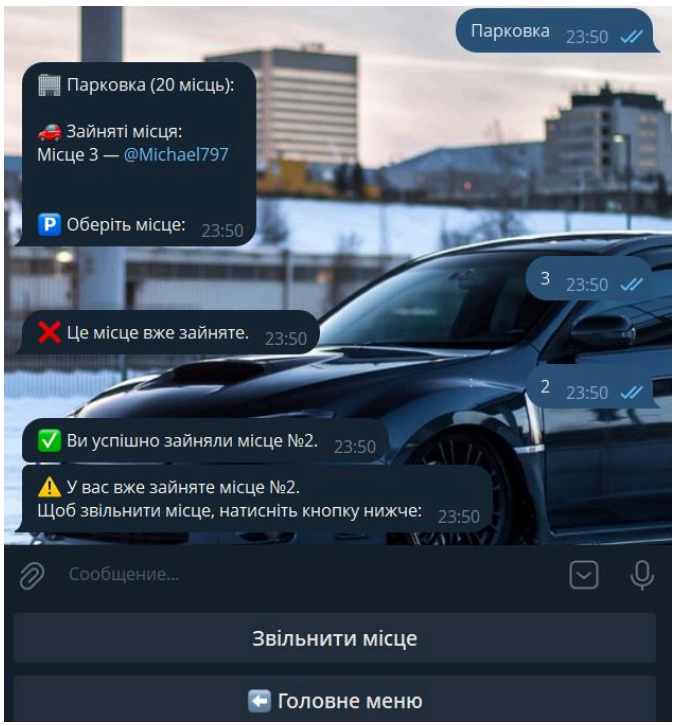


Рис. 4.5.14. Парковка

7) Додатково

При натисканні кнопки додатково, бот видає меню (Рис. 4.5.16) яке складається з чотирьох кнопок:

1. Заяви;
2. Основні засоби;
3. Налаштування;
4. Назад.

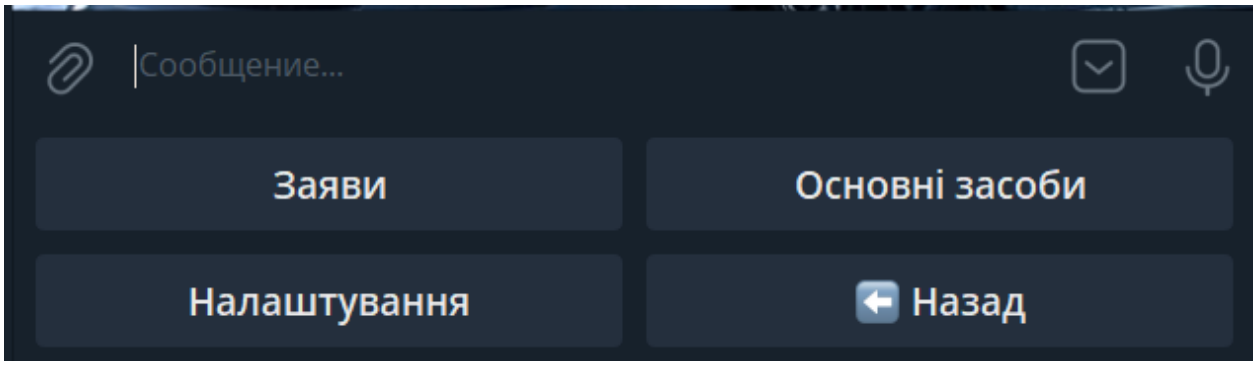


Рис. 4.5.15. Меню додатково

При натисканні заяви,бот просить вибрати тип заяви «Заява на відпустку» або «Заява на відрядженння». Після обрання типу заяви бот генерує заяву та видає ворд документ(Рис. 4.5.17).

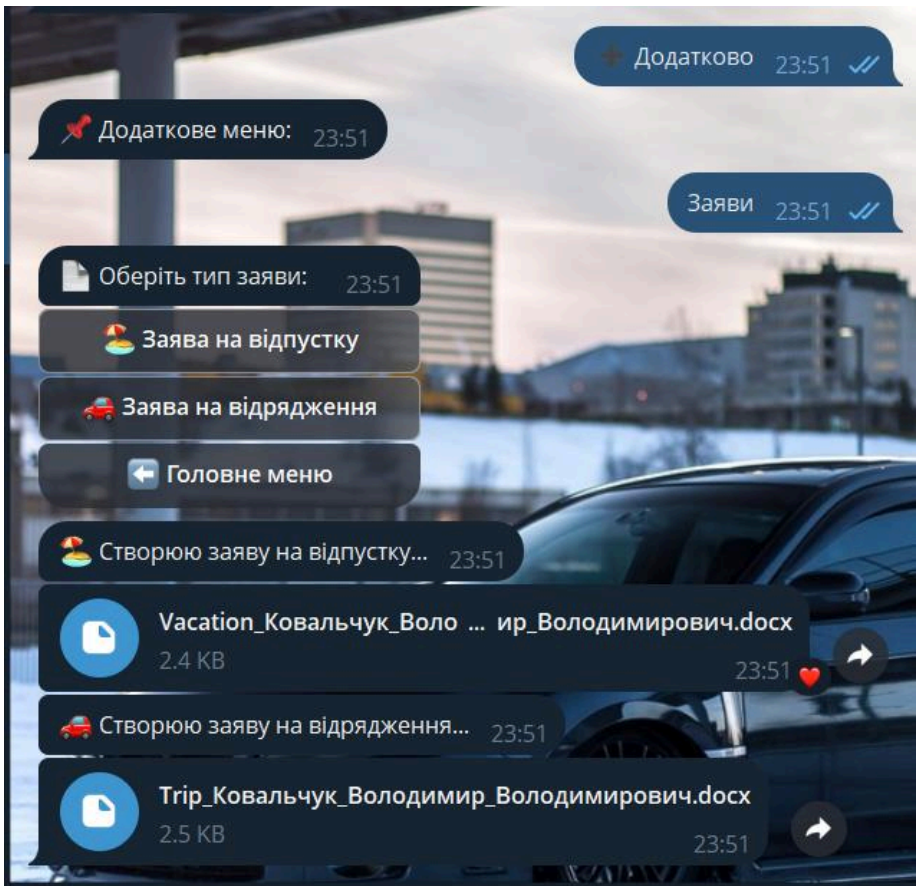


Рис. 4.5.16. Створення заяви

Висновок до розділу 4

В ході виконання розділу провели повний цикл прототипування програмного засобу. Розпочали розділ з створення прототипу бази даних, який містить необхідні таблиці, зв’язки та первинні механізми взаємодії з даними. Це дозволило забезпечити узгодженість структури з вимогами предметної області та підготувати основу для реалізації бізнес-логіки. На другому етапі було розроблено бізнес-логіку, яка забезпечує виконання операцій із даними, обробку запитів користувача, застосування правил предметної області та взаємодію між різними модулями. Реалізація бізнес-логіки дала змогу сформувати структурований та масштабований функціональний шар, придатний для подальшого розширення. Після реалізації бізнес логіки перейшли до створення Telegram-бота, який виступає основним інтерфейсом взаємодії користувачів із системою. Реалізовано модульну структуру, підключено бізнес-логіку та інтерпретовано сценарії обробки команд і

повідомлень. Це дозволило перевірити коректність роботи сервісів у реальному сценарії використання.

Також розроблено адміністративну панель Telegram-бота, яка забезпечує керування товарами, замовленнями та користувачами. Адмін-панель дозволила протестувати можливості системи з точки зору оператора та оцінити зручність управління процесами.

На прикінці розділу було проведено оцінювання результатів прототипування, а саме запущено телеграм боту та проведено ручне тестування яке вказало на те що весь розроблений функціонал працює вірно та без помилок.

ЗАГАЛЬНИЙ ВИСНОВОК ПО РОБОТІ

В процесі виконання кваліфікаційної роботи було проведено дослідження предметної області та визначено основні проблеми, з якими стикаються організації у використанні застарілих або недосконалих програмних засобів. Було сформульовано цілі проекту, визначено стейкхолдерів і цільову аудиторію, а також описано ключові ризики. На основі аналізу конкурентних рішень і потреб користувачів сформовано концепцію інформаційної системи та окреслено вимоги до її функціоналу. Далі проведено конкретизацію технічних і нефункціональних вимог до майбутньої системи, а також описано вимоги до бази даних. Сформовано стратегію створення застосунку, яка включає вибір архітектури, методології та визначення стеку технологій. Було обґрунтовано доцільність використання JavaFX для реалізації бізнес-логіки й адміністративного інтерфейсу, SQLlite як реляційної СУБД, а також інструментів для тестування, управління версіями й оптимізації продуктивності.

Після того було проведено первинне проектування, в якому було підібрано та описано архітектуру проекту, та створені ER-діаграма, діаграма варіантів використання, діаграма класів та послідовності, які стали в пригоді при створенні прототипу, адже надали візуальне бачення прототипу та зменшили можливість дублювання класів та функцій.

Створили прототип бази даних, з якою працюватиме прототип telegram-боту, створено та описано бізнес логіку та написано клас який реалізував взаємозв’язок бази даних, бізнес логіки та функціональних кнопок. Розроблена панель адміністрування спростила робочий процес адміністратора, адже надала інтуїтивний та простий засіб для підтримки боту. На прикінці роботи провели аналіз виконаної роботи, оглянули прототип програмного продукту та протестували його функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

2. Oracle. Java Platform, Standard Edition Documentation. [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/javase/

3. MySQL Documentation. MySQL Reference Manual. Oracle Corporation, 2023. – [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/

4. JavaFX Documentation. OpenJFX Project. [Електронний ресурс]. – Режим доступу: https://openjfx.io/

5. Telegram Bot API. Official Documentation. [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api

6. Hibernate ORM. User Guide. Red Hat, 2023. – [Електронний ресурс]. – Режим доступу: https://hibernate.org/orm/documentation/

7. Sommerville I. Software Engineering. 10th edition. Pearson, 2016.

8. Pressman R. S. Software Engineering: A Practitioner’s Approach. 8th edition. McGraw-Hill, 2014.

9. Apache Software Foundation. Apache JMeter User Manual. [Електронний ресурс]. – Режим доступу: https://jmeter.apache.org/