Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу»

Виконав: студент 4 курсу
група 4ПІ-21б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Богач І.І.
(прізвище та ініціали)
Керівник:д.т.н., проф. каф. ПЗ Ліщинська Л.Б.
(прізвище та ініціали)
Рецензент: к.т.н. доц. каф. ОТ Черняк О.І.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
(прізвище та ініціали)
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Богачу Ігнатію Івановичу

1. Тема роботи – «Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу».

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: тип програми – мобільний застосунок; модель розробки – ітеративна; засоби організації даних програми – фреймворки Room та DataStore; вхідні дані: дані клієнтів, інформація про завдання, події, дзвінки, угоди; вихідні дані: відображення списків завдань, подій, дзвінків, угод, формування статистики на основі цих даних; середовище розробки – Android Studio; мова програмування – Kotlin, програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку CRM-систем для малого бізнесу; розробка структури, моделі та алгоритмів системи; розробка системи; тестування CRM-системи для малого бізнесу; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів; використані технології; розробка методу офлайн-менеджера валют;

розробка методу формування пріоритетів завдань; розробка моделі застосунку; тестування застосунку; висновки; апробація результатів роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л.Б., д.т.н., професор кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку CRM-систем для малого бізнесу	25.03.2025 — 03.04.2025	вик
2	Розробка структури та моделі системи	04.04.2025 — 11.04.2025	вик
3	Розробка методу офлайн-менеджера валют	12.04.2025 — 16.04.2025	вик
4	Розробка алгоритмів роботи застосунку	17.04.2025 — 27.04.2025	вик
5	Розробка системи	28.04.2025 — 18.05.2025	вик
6	Тестування CRM-системи для малого бізнесу	19.05.2025 — 26.05.2025	вик
7	Оформлення матеріалів до захисту БКР	27.05.2025 — 30.05.2025	вик

Студент (підпис) **Богач І.І.** (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи (підпис) **Ліщинська Л.Б.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004:005.9

Богач І.І. Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 86 с.

На укр. мові. Бібліогр. : 21 назв.; рис.: 35; табл.: 10.

У бакалаврській кваліфікаційній роботі було визначено доцільність та мету розробки, яка полягає в удосконаленні бізнес-процесів малого бізнесу за рахунок створення мобільного застосунку – спеціалізованої CRM-системи. Для досягнення поставленої мети було сформульовано задачі дослідження, проведено аналіз стану питання розробки CRM-систем, визначено сучасні підходи та методи до реалізації подібних систем.

У ході роботи було розроблено структуру CRM-системи; розроблено алгоритм автоматичного формування пріоритетів завдань на основі історії користувача та змісту завдань. розроблено офлайн-менеджер валют.

Програмне забезпечення розроблено із використанням середовища розробки Android Studio та мови програмування Kotlin. У процесі розробки використовувались сучасні бібліотеки та інструменти, такі як Room, Hilt, DataStore, Material Components та Chaquopy. Під час тестування було перевірено працездатність основного функціоналу системи та підтверджено коректність реалізованих методів.

У результаті виконання роботи розроблено мобільний застосунок, що дозволяє ефективно керувати взаємодією з клієнтами, контролювати процеси продажів та підвищувати продуктивність малого бізнесу.

Ключові слова: CRM-система, мобільний застосунок, Kotlin, угоди, завдання, клієнти, малий бізнес.

ABSTRACT

UDC 004:005.9

Bohach I.I. Management system for programming customer relationships in CRM for small business. Vinnytsia : VNTU, 2025. 86 p.

In Ukrainian language. Bibliobrapher: 21 title; fig.: 35; tabl.: 10.

In the bachelor's thesis, the feasibility and purpose of the development was determined, which is to improve the business processes of small businesses by creating a mobile application - a specialized CRM system. To achieve this goal, the research objectives were formulated, the state of the art of CRM system development was analyzed, and modern approaches and methods for implementing such systems were identified.

In the course of the work, the structure of the CRM system was developed; an algorithm for automatically prioritizing tasks based on the user's history and the content of tasks was developed. an offline currency manager was developed.

The software was developed using the Android Studio development environment and the Kotlin programming language. Modern libraries and tools such as Room, Hilt, DataStore, Material Components, and Chaquopy were used in the development process. During the testing, we checked the operability of the main system functionality and confirmed the correctness of the implemented methods.

As a result of the work, a mobile application has been developed that allows you to effectively manage customer interaction, control sales processes, and increase the productivity of small businesses.

Keywords: CRM system, mobile application, Kotlin, transactions, tasks, customers, small business.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасних умовах конкуренції малий бізнес все частіше звертається до спеціалізованих рішень для автоматизації взаємодії з клієнтами та оптимізації внутрішніх процесів. Впровадження CRM-систем допомагає стандартизувати роботу відділів продажів, маркетингу та підтримки, що призводить до підвищення ефективності і лояльності клієнтів . При цьому вже 51 % малих підприємств, які використали CRM, відзначили зростання конверсії лідів, а 72 % опитаних власників вважають такі рішення ефективними для досягнення бізнес-цілей [1]. Водночас глобальний ринок CRM-систем у 2024 р. оцінювався майже в 101,4 млрд USD з прогнозованим ростом до 112,9 млрд USD у 2025 р. і подальшим зростанням до 262,7 млрд USD до 2032 р. [2]. Проте близько 50 % проектів впровадження CRM стикаються з невдачею через відсутність координації між відділами [3], що вказує на необхідність детального опрацювання вимог і адаптації рішення під специфіку малого бізнесу.

Customer Relationship Management (CRM) охоплює практики, стратегії та технології для аналізу взаємодії підприємства з поточними та потенційними клієнтами. CRM-системи забезпечують централізоване зберігання даних про контрагента, історію комунікацій та аналітику ефективності продажів, що дозволяє підвищити якість обслуговування та швидкість обробки запитів [4]. Для малих підприємств, де ресурси на підтримку клієнтської бази обмежені, впровадження CRM стає ключовим інструментом для цифрової трансформації та зростання конкурентоспроможності.

Актуальність розробки CRM-системи для малого бізнесу полягає в тому, що CRM дозволяє автоматизувати рутинні операції – нагадування, сегментацію контактів, відстеження угод, що скорочує час на адміністративні завдання і дозволяє зосередитися на стратегії продажів. Крім цього, аналітичні можливості системи дають змогу відстежувати ефективність маркетингових кампаній і швидко реагувати на зміни поведінки клієнтів. Також інтеграція інструментів штучного інтелекту (наприклад, чат-ботів) вже планується 80% компаній, що використовують CRM, що підвищує швидкість реагу-

вання на запити та покращує клієнтський досвід.

Глобальний ринок CRM продовжує стрімко зростати: у 2024 р. він становив $101,41 млрд, а вже в 2025 р. очікується збільшення до $112,91 млрд. Водночас малий бізнес активно освоює хмарні CRM-рішення: вже 87 % підприємств використовують хмарні платформи, а серед тих, хто використовує мобільні CRM, 65 % досягають своїх продажних цілей [5]. Це свідчить про те, що сфера CRM-додатків знаходиться на етапі зрілості, але продовжує впроваджувати нові функції для задоволення потреб різних сегментів бізнесу.

Отже, актуальною є розробка системи управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є удосконалення бізнес-процесів малого бізнесу шляхом розробки спеціалізованої CRM-системи, що дозволяє вести облік угод, зустрічей з клієнтами та справ.

Відповідно до поставленої мети потрібно вирішити такі **завдання**:

- розробити структуру CRM-системи для малого бізнесу;
- розробити інтерфейс користувача застосунку;
- розробити модель системи;
- розробити алгоритми роботи системи;
- розробити функціонал системи;
- провести тестування розробленої системи.

Об'єктом дослідження є процеси управління взаємовідносинами з клієнтами у CRM для малого бізнесу.

Предметом дослідження є методи і засоби управління взаємовідносинами з клієнтами у CRM для малого бізнесу.

Методи дослідження. У процесі досліджень використовувались такі методи дослідження: аналіз аналогів та порівняння їх з власною розробкою для постановки задач розробки; теорія чисел та чисельних методів; теорія алгори-

тмів для розробки та візуалізації алгоритмів роботи програмного забезпечення; елементи математичної статистики для генерування статистичних даних у застосунку; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Удосконалено метод адаптивного формування пріоритетів завдань на основі базового семантичного та статистичного аналізу ключових слів, що дозволяє автоматично визначати важливість нових завдань.

2. Удосконалено метод офлайн-менеджменту курсів валют, який поєднує локальне кешування із парсингом даних, що дозволяє підвищити стійкість застосунку, забезпечуючи гарантований доступ до даних навіть за відсутності підключення до мережі.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській кваліфікаційній роботі, отримано автором самостійно. У наукових працях, опублікованих у співавторстві, автору належать такі результати: аналіз ролі CRM-систем у цифровій трансформації малого бізнесу.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи доповідалися на: XXV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів «Стан, досягнення і перспективи інформаційних систем і технологій» (Одеса, 2025 р.).

Публікації. За тематикою дослідження опубліковано 1 наукову працю у збірнику матеріалів конференції [6].

1 АНАЛІЗ СТАНУ РОЗВИТКУ CRM-СИСТЕМ ДЛЯ МАЛОГО БІЗНЕСУ

1.1 Аналіз предметної галузі

Сучасний ринок систем управління взаємовідносин з клієнтами переживає період активної трансформації, спричиненої зростаючими потребами малого бізнесу в доступних та ефективних інструментах автоматизації. Згідно з дослідженнями аналітичних агенцій, ринок CRM-систем демонструє стабільне щорічне зростання на рівні 12-14%, що свідчить про зростаючий попит на подібні рішення [7]. Особливої актуальності набуває сегмент CRM-систем, спеціально адаптованих для потреб малого бізнесу, який історично мав обмежений доступ до подібного програмного забезпечення через високу вартість та складність впровадження.

Основною тенденцією в розробці CRM для малого бізнесу стала демократизація технологій, що виявляється у створенні рішень з моделлю SaaS (Software as a Service), яка не потребує значних початкових інвестицій та складної технічної інфраструктури. Дана модель дозволяє малому бізнесу отримати доступ до функціоналу, який раніше був прерогативою виключно корпоративного сегменту, що суттєво вирівнює конкурентні можливості на ринку.

Аналіз актуальних розробок у сфері CRM для малого бізнесу виявляє кілька домінуючих технологічних трендів. Інтеграція штучного інтелекту та машинного навчання стає стандартом галузі, що дозволяє системам автоматично аналізувати поведінку клієнтів, прогнозувати вірогідність закриття угод та пропонувати персоналізовані рекомендації щодо взаємодії з потенційними покупцями. Впровадження цих технологій суттєво підвищує ефективність процесів продажу та маркетингу, дозволяючи малому бізнесу з обмеженими ресурсами досягати результатів, порівнянних з більшими конкурентами.

Мобільна доступність CRM-систем також стала критичним компонентом сучасних розробок. Згідно з дослідженнями, близько 65% представників малого бізнесу використовують мобільні пристрої для доступу до бізнес-додатків, що

зумовлює необхідність оптимізації CRM-систем для роботи на різних пристроях[8]. Розробники активно впроваджують стратегію "mobile-first", що передбачає проектування інтерфейсів з пріоритетом мобільного використання, забезпечуючи повноцінний функціонал навіть на пристроях з обмеженим розміром екрану.

Сучасна розробка CRM-систем для малого бізнесу характеризується переходом до мікросервісної архітектури, яка забезпечує гнучкість, масштабованість та можливість поетапного впровадження функціоналу. Даний підхід дозволяє малому бізнесу починати з базових функцій, поступово додаючи нові можливості без необхідності повної заміни системи, що суттєво знижує ризики та вартість впровадження.

Важливим аспектом архітектурних рішень стала також інтеграційна відкритість, що реалізується через розвинуті API (Application Programming Interface) та готові конектори до популярних бізнес-сервісів. Дослідження показують, що малий бізнес використовує в середньому 7-10 різних програмних рішень для управління операційною діяльністю, і безперешкодна інтеграція між ними стає вирішальним фактором ефективності. Сучасні CRM-системи перетворюються на центральні хаби, що об'єднують дані з різних джерел, забезпечуючи цілісний погляд на клієнта та бізнес-процеси.

Незважаючи на значний прогрес, розробка та впровадження CRM-систем для малого бізнесу зіштовхуються з низкою викликів. Першою суттєвою проблемою залишається баланс між функціональністю та простотою використання. Дослідження свідчать, що близько 43% впроваджень CRM у малому бізнесі зазнають невдачі через складність інтерфейсів та необхідність тривалого навчання персоналу, що відволікає обмежені ресурси від основної діяльності [9].

Іншим викликом є забезпечення інформаційної безпеки при роботі з клієнтськими даними. З огляду на посилення регуляторних вимог щодо захисту персональних даних (GDPR, CCPA та інші), розробникам CRM доводиться впроваджувати комплексні механізми безпеки, зберігаючи при цьому доступність рішень для малого бізнесу з обмеженими технічними компетенціями. Ця

дилема призводить до необхідності розробки інтуїтивно зрозумілих інструментів управління безпекою, які не вимагають спеціалізованих знань.

Аналіз актуальних тенденцій дозволяє визначити кілька перспективних напрямків розвитку CRM-систем для малого бізнесу. Омніканальність стає новим стандартом, що передбачає безперешкодну інтеграцію різних каналів комунікації (електронна пошта, соціальні мережі, месенджери, телефонія) в єдиний інтерфейс. Дослідження показують, що клієнти використовують в середньому 3-4 різні канали при взаємодії з бізнесом, і здатність забезпечити цілісний досвід через усі ці канали стає конкурентною перевагою.

Предиктивна аналітика на основі штучного інтелекту також становить значний потенціал для розвитку CRM-систем. Технології, що дозволяють прогнозувати поведінку клієнтів, виявляти риски відтоку та ідентифікувати можливості для додаткових продажів, стають доступними для малого бізнесу через готові рішення, що не потребують спеціалізованих знань у сфері аналізу даних.

Економічна доцільність впровадження CRM-систем у малому бізнесі залишається критичним питанням, що впливає на розвиток галузі. Дослідження показують, що середнє повернення інвестицій (ROI) при впровадженні CRM становить близько 5,60 долара на кожен вкладений долар, проте цей показник суттєво варіюється залежно від галузі та ефективності впровадження. Розробники CRM активно працюють над створенням моделей ціноутворення, що відповідають специфіці малого бізнесу, включаючи гнучкі тарифні плани, можливість оплати лише за фактично використаний функціонал та відсутність довгострокових контрактних зобов'язань.

Отже, поточний стан розробки CRM-систем для малого бізнесу характеризується динамічним розвитком, що відповідає зростаючим потребам ринку в ефективних інструментах автоматизації взаємодії з клієнтами. Тому актуальною є розробка такої системи.

1.2 Порівняльний аналіз аналогів

Для визначення функціоналу власного функціоналу, дослідження потреб у функціоналі для системи управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу, проведемо порівняльний аналіз таких існуючих рішень: Pipedrive, Zoho CRM, Freshsales, Microsoft Dynamics 365.

Pipedrive [10] — система, орієнтована на оптимізацію процесу продажів. Система розроблена з урахуванням потреб відділів продажів, що дозволяє ефективно управляти угодами на різних етапах. Функціонал Pipedrive зосереджений у першу чергу на відстеженна активностей з продажу, управлінні угодами та контактами, а також прогнозуванні результатів. Однак при цьому Pipedrive має недолік, який полягає у обмеженому функціоналі для автоматизації маркетингу та обслуговування.

Інтерфейс Pipedrive наведено на рисунку 1.1.

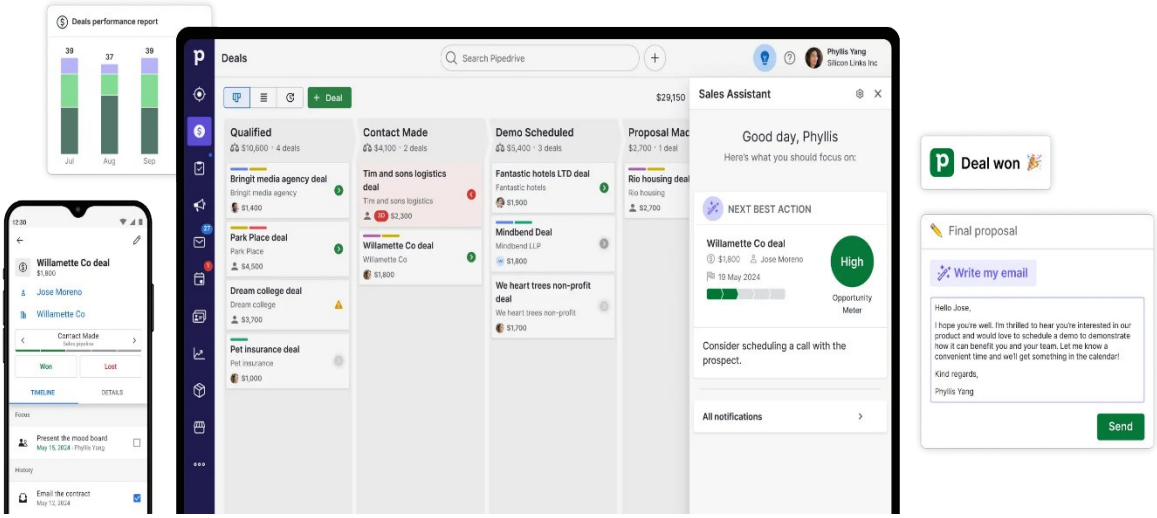

Рисунок 1.1 — Інтерфейс Pipedrive

Zoho CRM [11] — система, що пропонує широкий спектр функцій для управління процесами продажів, автоматизації маркетингу та аналітики клієнтських даних.

Zoho CRM забезпечує значні можливості для налаштування робочих процесів, включаючи створення власних модулів та полів, налаштування автоматизації та розробку персоналізованих звітів. Екосистема Zoho включає більше 40

інтегрованих бізнес-додатків, що забезпечує обмін даними між різними аспектами діяльності компанії.

Інтерфейс Zoho CRM наведено на рисунку 1.2.

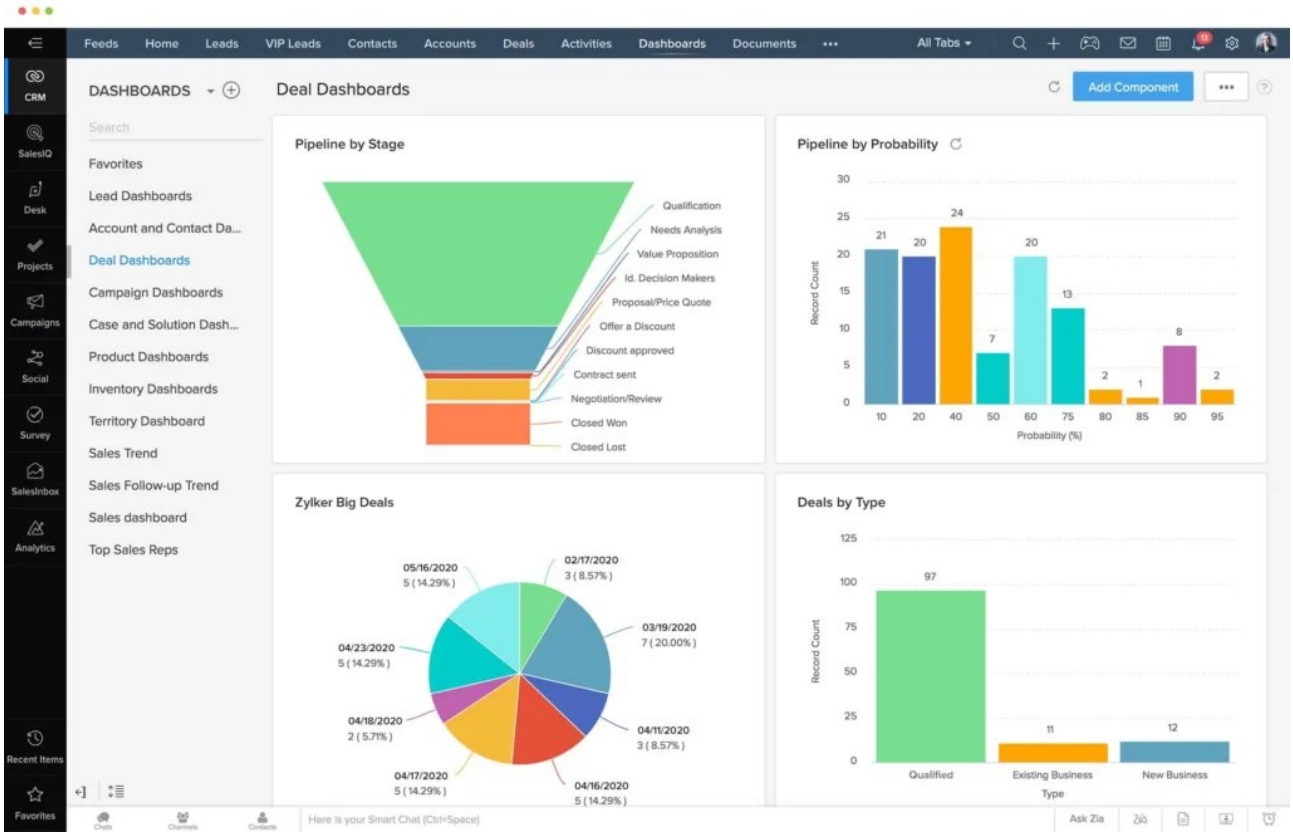

Рисунок 1.2 — Інтерфейс Zoho CRM

Freshsales [12] являє собою CRM-рішення, яке поєднує функціональність управління продажами з елементами автоматизації маркетингу. Ключовою перевагою Freshsales є вбудована система штучного інтелекту Freddy AI, яка допомагає аналізувати поведінку клієнтів, прогнозувати угоди та автоматизувати рутинні завдання. Система забезпечує комплексний підхід до управління клієнтським циклом, включаючи лідогенерацію, кваліфікацію, конверсію та утримання клієнтів.

Інтерфейс Freshsales наведено на рисунку 1.3.

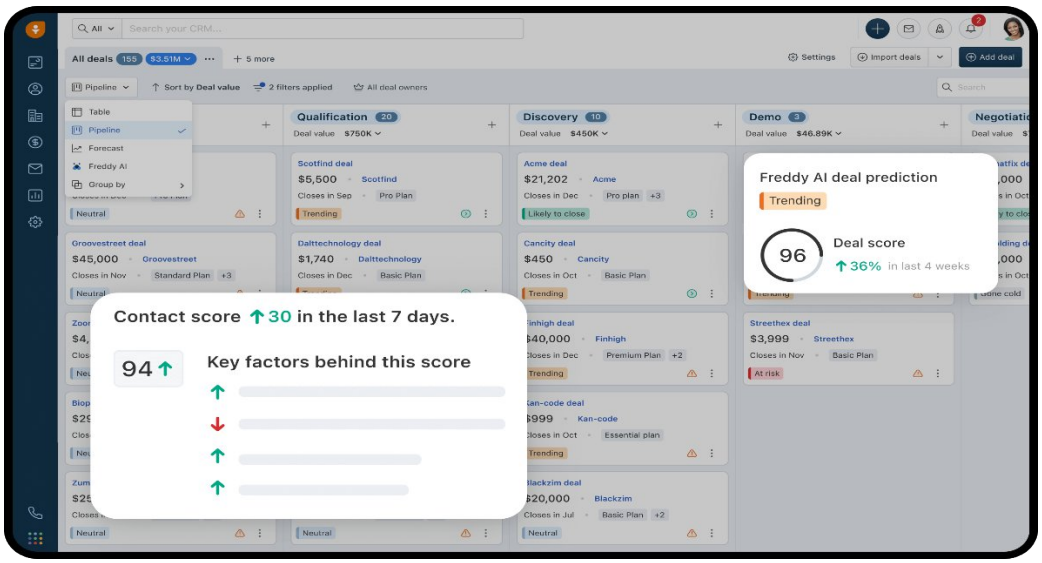

Рисунок 1.3 — Інтерфейс Freshsales

Microsoft Dynamics 365 [13] — корпоративне CRM-рішення, адаптоване для потреб малого та середнього бізнесу. Система інтегрується з екосистемою Microsoft, забезпечуючи безперешкодну взаємодію з такими продуктами, як Office 365, Outlook та Power BI. Функціонал включає управління продажами, маркетингом, обслуговуванням клієнтів та бізнес-аналітикою. Система пропонує широкі можливості для налаштування та масштабування відповідно до зростаючих потреб бізнесу.

Інтерфейс Microsoft Dynamics 365 наведено на рисунку 1.4.

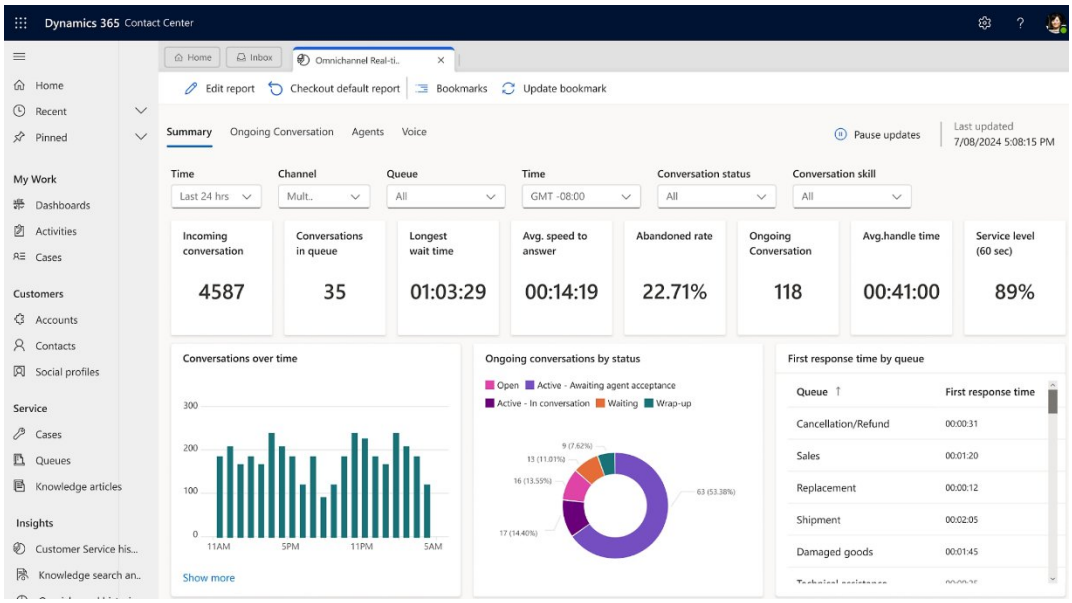

Рисунок 1.4 — Інтерфейс Microsoft Dynamics 365

З метою порівняння функціоналу розглянутих CRM-систем, та власної розробки, було сформовано таблицю 1.1.

Таблиця 1.1 — Порівняльний аналіз аналогів

Характеристика	Dynamics 365	Freshsales	Zoho CRM	Pipedrive	Власна розробка
Управління контактами	+	+	+	+	+
Управління угодами/продажами	+	+	+	+	+
Розширена аналітика та звітність	+	-	+	-	+
Мобільний додаток	+	+	+	+	+
Можливість налаштування бізнес-процесів	+	+	+	+	+
Функціонал для малого бізнесу	-	+	+	+	+
Автоматичне формування пріоритетів завдань	-	-	-	-	+
Сумарний бал	5	5	6	5	7

Власна розробка CRM-системи є актуальною завдяки можливості повного налаштування під специфічні потреби бізнесу. На відміну від готових рішень, вона дозволяє інтегрувати унікальні бізнес-процеси, використовувати штучний інтелект для автоматизації та аналітики, а також створювати функціонал, який максимально відповідає масштабу та цілям компанії. Завдяки цьому власна розробка отримала найвищий сумарний бал і демонструє найкращу гнучкість та перспективність у довгостроковій перспективі.

1.3 Аналіз методів розв’язання задачі

Розробка системи управління взаємовідносин з клієнтами для малого бізнесу вимагає комплексного підходу, що враховує обмеженість ресурсів, специфічні бізнес-процеси та потреби цільової аудиторії. Розглянемо основні

методологічні підходи до вирішення цієї задачі, їхні переваги, недоліки та сфери застосування.

Монолітний підхід передбачає розробку CRM-системи як єдиного цілісного додатку, де всі функціональні модулі тісно інтегровані [14]. Цей метод характеризується відносною простотою впровадження та управління на початкових етапах.

Переваги монолітної архітектури включають нижчу початкову складність розробки, спрощене розгортання та тестування, а також високу продуктивність завдяки відсутності мережевих затримок між компонентами. Для малого бізнесу з обмеженими технічними ресурсами даний підхід може бути оптимальним на початкових етапах.

Однак монолітна архітектура має суттєві обмеження щодо масштабованості та гнучкості. Із зростанням функціональності системи збільшується складність розробки та підтримки, знижується швидкість впровадження змін та зростає ризик виникнення дефектів при модифікації коду.

Мікросервісний підхід передбачає декомпозицію CRM-системи на набір незалежних сервісів, кожен з яких відповідає за певну бізнес-функцію (управління контактами, продажами, маркетингом тощо) [15]. Сервіси можуть розроблятися, розгортатися та масштабуватися незалежно один від одного.

Цей метод забезпечує високу гнучкість розробки, можливість поетапного впровадження функціональності та ефективне масштабування системи. Для малого бізнесу це означає можливість починати з базового набору функцій, поступово додаючи нові можливості відповідно до зростання потреб.

Проте мікросервісна архітектура вимагає більш складної інфраструктури, збільшує початкові витрати на розробку та потребує вищої технічної експертизи для впровадження та підтримки. Ці фактори можуть стати обмежуючими для малого бізнесу з обмеженим бюджетом та відсутністю власної технічної команди.

Гібридна архітектура поєднує елементи монолітного та мікросервісного підходів, дозволяючи досягти балансу між простотою впровадження та гнучкі-

стю розвитку. У контексті CRM для малого бізнесу це може реалізуватися як розробка ключових функцій у вигляді монолітного ядра з можливістю інтеграції додаткових функцій через незалежні сервіси.

Цей метод дозволяє отримати швидкий результат із базовим функціоналом та забезпечити гнучкість подальшого розвитку. Для малого бізнесу гібридний підхід часто є оптимальним, оскільки дозволяє мінімізувати початкові витрати при збереженні перспектив масштабування, тому прийнято рішення розробити систему саме з використанням цього підходу.

Порівняльну характеристику описаних підходів наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика методів розробки

Критерій	Монолітна архітектура	Мікросервісна архітектура	Гібридна архітектура
Простота реалізації	Висока	Низька	Середня
Швидкість початкового впровадження	Висока	Низька	Висока
Гнучкість і масштабованість	Низька	Висока	Середня
Інфраструктурні вимоги	Низькі	Високі	Середні
Технічні вимоги до команди	Невисокі	Високі	Середні
Можливість поетапного розвитку	Обмежена	Висока	Висока
Стабільність та цілісність	Висока	Середня	Висока

З огляду на потреби малого бізнесу, обмеженість ресурсів і необхідність швидкого запуску базового функціоналу, монолітний підхід є найбільш доцільним для запланованої розробки CRM-системи. Він дозволяє мінімізувати витрати на старті, швидко отримати працездатний продукт і забезпечити стабільну роботу системи без складних інфраструктурних рішень. Попри обмеження в масштабованості, монолітна архітектура є цілком виправданим вибором для перших етапів, особливо у випадку, коли розвиток функціоналу пла-

нується поступово й без залучення великої команди. У майбутньому можливий перехід до гібридної або мікросервісної моделі при зростанні потреб і технічних можливостей.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану CRM-систем для малого бізнесу, порівняння аналогів було поставлено такі задачі дослідження:

- розробити структуру CRM-системи для малого бізнесу;
- розробити інтерфейс користувача застосунку;
- розробити модель системи;
- розробити алгоритми роботи системи;
- розробити функціонал системи;
- провести тестування розробленої системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

Було здійснено формулювання цілей і постановку задач дослідження. Для цього проаналізовано стан розвитку CRM-систем для малого бізнесу, проведено порівняння з аналогами, розглянуто сильні та слабкі сторони актуальних рішень. На основі цього сформовано перелік задач, які відповідають реальним потребам малого бізнесу.

2 РОЗРОБКА СТРУКТУРИ, МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ

2.1 Розробка структури системи

Структура системи організована відповідно до принципів модульної архітектури та слідує шаблону MVVM (Model-View-ViewModel), що забезпечує чіткий розподіл відповідальностей між компонентами.

Структура додатку розподілена на логічні модулі, що відповідають за різні аспекти функціональності системи. Модуль "data" відповідає за роботу з даними і містить підмодулі "currency" та "db". Підмодуль "currency" забезпечує функціональність для роботи з валютами та обмінними курсами через класи CurrencyManager, ExchangeRateApiService та ExchangeRateResponse, що вказує на інтеграцію із зовнішніми API для отримання актуальних курсів валют.

Модуль "db" є ключовою частиною додатку, що реалізує доступ до локальної бази даних. Він містить клас AppDatabase, який служить основою для локального зберігання даних, а також набір класів DAO (Data Access Object) та сутностей для кожного типу бізнес-об'єктів: клієнтів, дзвінків, завдань, подій та угод.

Модуль "di" (Dependency Injection) представлений класом DatabaseModule для використання підходу ін’єкції залежностей для забезпечення гнучкості та тестованості додатку.

Модуль "repository" реалізує шар доступу до даних відповідно до шаблону Repository, абстрагуючи джерела даних від інших компонентів системи. Він включає окремі репозиторії для кожного типу бізнес-об'єктів: CallRepository, ClientRepository, DealRepository, EventRepository та TaskRepository, що забезпечує чітке розмежування відповідальностей.

Модуль "ui" відповідає за представлення інформації користувачеві. Він поділений на підмодулі відповідно до функціональних екранів додатку: contacts, dashboard, deals та home. Кожен з цих підмодулів містить компоненти, необхідні для відображення та взаємодії з відповідними даними: фрагменти, адаптери, моделі представлення та елементи списків. Підмодуль "home"

додатково містить компоненти для роботи з календарем та відображення різних типів активностей клієнтів.

Структура застосунку наведена на рисунку 2.1.

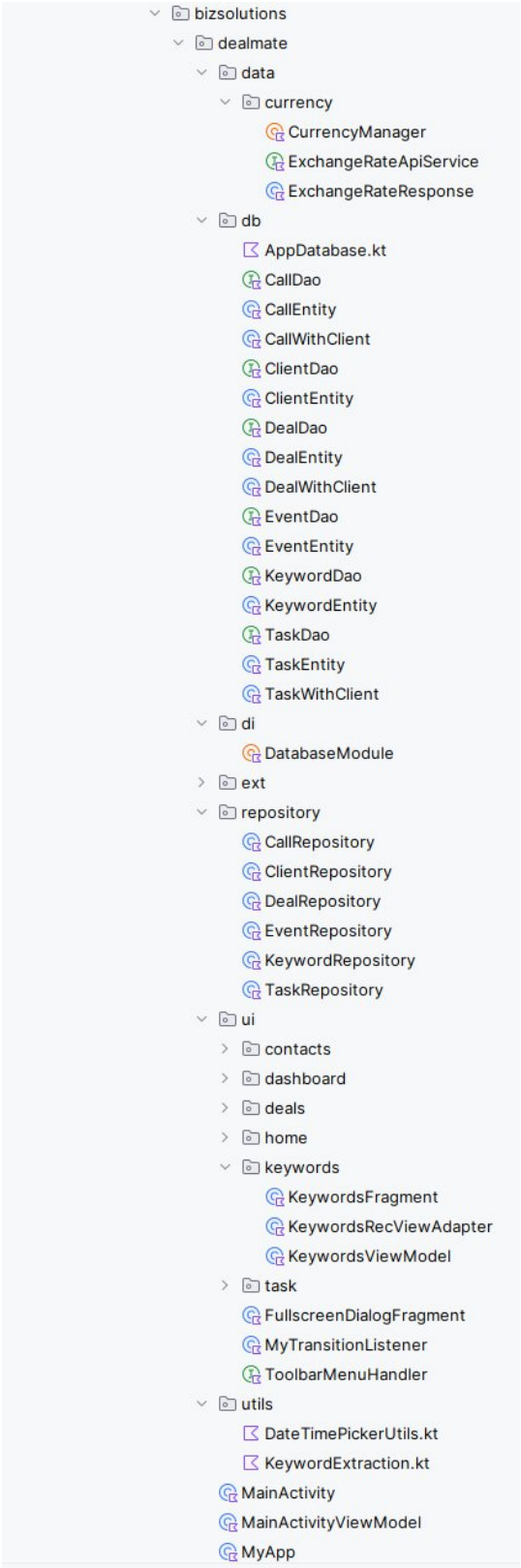

Рисунок 2.1 — Структура системи

Отже, розроблена структура системи містить чітке розділення відповідальностей між компонентами, що забезпечує масштабованість та легкість підтримки коду.

2.2 Розробка моделі системи

Модель системи включає п’ять головних екранів системи та зовнішні бібліотеки, від яких кожен із них залежить. Система поєднує функціонал панелі управління, головного екрану, управління угодами й контактами.

Dashboard відповідає за візуалізацію агрегованої бізнес-статистики (поточний баланс, загальна кількість угод, графіки активності). Для взаємодії з віддаленим API він використовує Retrofit (оголошення REST-інтерфейсів), виконує запити всередині Coroutines для неблокуючої асинхронної роботи, а отриманий JSON перетворює в об’єкти моделі через Gson.

Home показує список останніх подій, дзвінків та нагадувань. Він також використовує coroutines для асинхронного завантаження даних (локальних і мережевих) та застосовує Gson для серіалізації налаштувань користувача й кеш-даних у DataStore.

Deals реалізує екран управління угодами: список, фільтри, сортування та контекстні дії. Для збереження й читання інформації він використовує Room, а оновлення інтерфейсу відбувається завдяки LiveData, яка автоматично повідомляє про зміни в базі даних.

Contacts дозволяє переглядати та редагувати клієнтську базу. Аналогічно до модуля Deals, він працює з Room для зберігання сутностей і використовує LiveData для спостереження за змінами в таблиці клієнтів, що забезпечує своєчасне оновлення RecyclerView на екрані.

Calendar відображає нескінченну стрічку дат із можливістю вибору дня для перегляду подій. Перетворення позиції у списку та навпаки реалізовано через Java Time API (LocalDate, ChronoUnit): це дозволяє точно обчислювати офсет від базової дати та синхронізувати відображення з RecyclerView.

Модель системи наведена на рисунку 2.2.

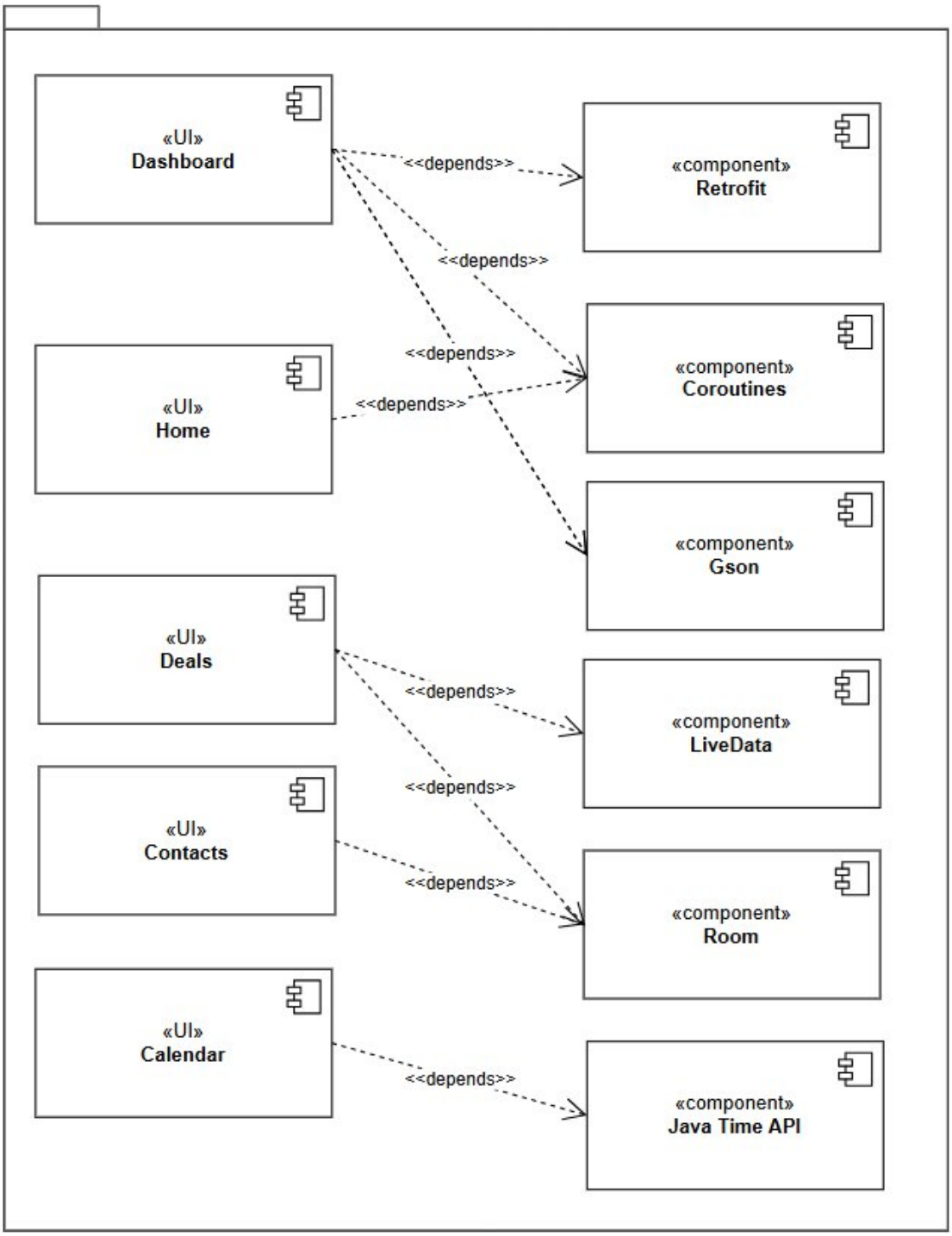

Рисунок 2.2 — Модель системи

Діаграма класів відображає фундаментальну доменну модель застосунку DealMate, де ключовими сутностями є ClientEntity та DealEntity. Клас ClientEntity має поля id, name і phone, що повністю описують профіль клієнта в системі. Клас DealEntity містить поля id, clientId (зовнішній ключ на клієнта), amount (сума угоди) та date (дата угоди), що дає змогу зберігати історію фінансових

транзакцій із прив’язкою до конкретного клієнта.

Інтерфейси ClientDao та DealDao виступають шаром доступу до бази даних Room, забезпечуючи CRUD-операції над сутностями. У діаграмі чітко показано, що ClientDao читає і записує об’єкти ClientEntity, а DealDao—об’єкти DealEntity та агреговані результати через DealWithClient. Відношення «один-до-багатьох» між клієнтом і угодами проілюстроване зв’язком ClientEntity “1” – “0..*” DealEntity, що підкреслює, як кожен клієнт може мати довільну кількість угод.

Діаграма класів наведена на рисунку 2.3.

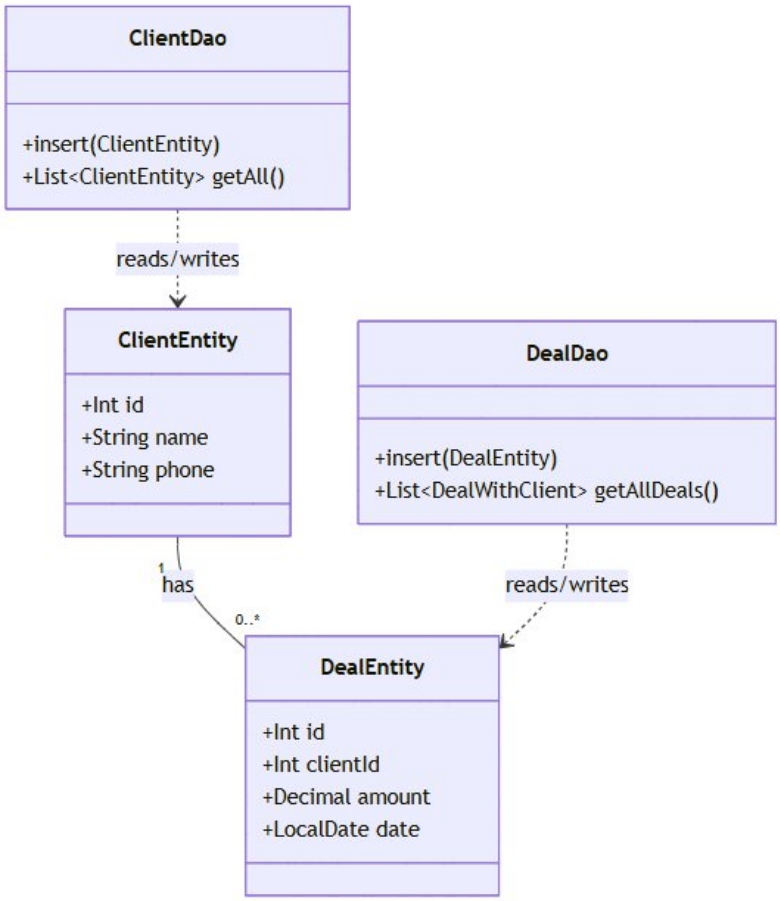

Рисунок 2.3 — Діаграма класів

Таким чином, розроблена модель системи формує функціонал системи, де кожен екран залежить тільки від тих компонентів і технологій, що необхідні для його роботи, забезпечуючи модульність, тестованість і зручність подальшого розширення.

2.3 Розробка методу формування пріоритетів завдань

У процесі управління завданнями в CRM-системі важливу роль відіграє ефективне визначення пріоритетів. Завдання, пов’язані з клієнтами, можуть мати різний рівень терміновості й важливості залежно від змісту, історії взаємодій або контексту. Ручне встановлення пріоритетів часто є суб’єктивним і не завжди ефективним. У зв’язку з цим виникає потреба в автоматизованому та адаптивному підході до формування пріоритетів на основі попередньої статистики виконання схожих завдань.

Щоб автоматизувати й адаптувати цей процес, було розроблено метод формування пріоритетів завдань, який базується на припущенні, що деякі слова або словосполучення в описах завдань мають підвищену інформативність щодо терміновості. Наприклад, ключові слова на кшталт “терміново”, “звіт клієнту”, “дедлайн” частіше пов’язані з високим рівнем пріоритету, тоді як слова “нагадати”, “перевірити” можуть мати нижчу пріоритетність.

Для виявлення таких закономірностей у системі зберігається статистика по кожному ключовому слову:

- кількість завершених завдань, де воно фігурує;
- кількість відкладень таких завдань;
- дата останнього використання слова.

На основі цих даних для кожного ключового слова обчислюється коефіцієнт терміновості. Чим частіше завдання з певним словом відкладалися або з'являлися нещодавно, тим більша ймовірність того, що нове завдання з подібним змістом буде важливим.

Пріоритет завдання визначається середнім значенням таких коефіцієнтів для всіх релевантних ключових слів, виявлених у назві або описі завдання. Потім це значення порівнюється з наперед заданими порогами для трьох рівнів пріоритету:

1) високий пріоритет – високий ризик відкладання або терміновість завдання;

2) середній пріоритет – нейтральна чи змішана статистика;

3) низький пріоритет – ключові слова, які містяться у завдання, що зазвичай виконуються без затримок.

Блок-схема роботи описаного методу наведена на рисунку 2.4.

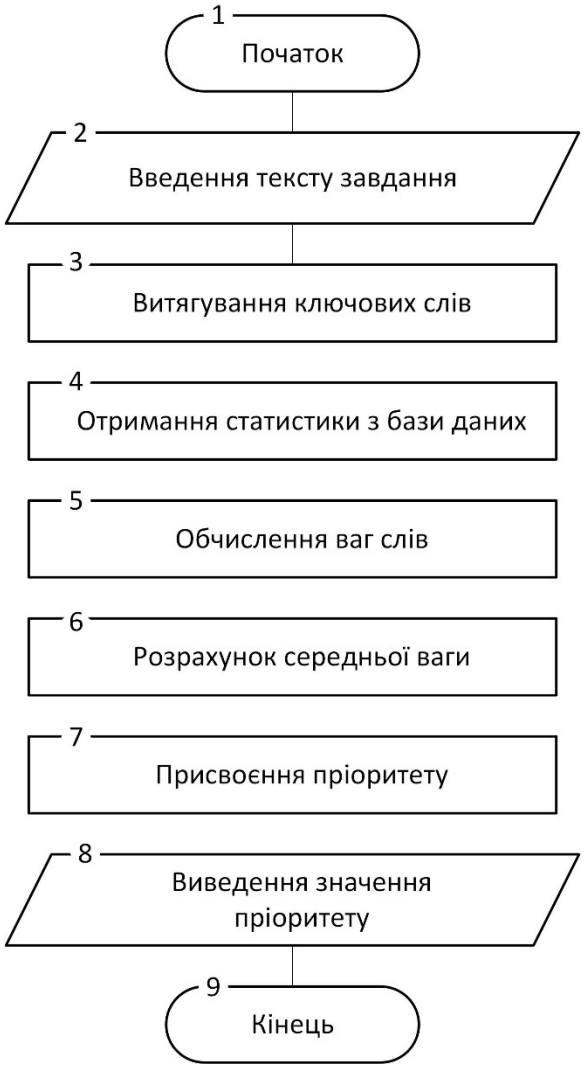

Рисунок 2.4 – Блок-схема методу формування пріоритетів завдань

Для реалізації методу передбачено такі етапи:

1) завдання аналізується за допомогою простих лінгвістичних правил (наприклад, розбиття на фрази по пробілах та пунктуації).

2) для знайдених слів виконується запит до бази даних, де зберігається інформація про попередній досвід взаємодії з подібними завданнями.

3) на основі кількості відкладань, виконань та часу останнього використання визначається ваговий коефіцієнт кожного слова.

4) всі коефіцієнти усереднюються, після чого значення порівнюється з встановленими порогами для визначення остаточного пріоритету.

Завдяки оновленню статистики після кожного завершеного або відкладеного завдання, метод адаптується до поточних користувацьких звичок. Система з часом стає точнішою без необхідності ручного втручання.

Таким чином, запропонований метод дозволяє автоматизувати процес визначення пріоритету завдань на основі об’єктивних показників. Такий підхід знижує суб’єктивність, оптимізує розподіл часу та підвищує продуктивність користувача.

2.4 Розробка методу офлайн-менеджера валют

Метод офлайн-менеджера валют забезпечує постійний доступ до актуальних курсів навіть за відсутності мережі або при тимчасових відмовах API. Він поєднує в собі AndroidX DataStore для локального кешування відповіді від сервера та захисний парсинг через Gson: маючи закешований JSON-рядок, система спершу намагається його розібрати й одразу повернути як карту курсів, а при будь-яких помилках (прострочений кеш, некоректний формат, викидання винятку) — автоматично звертається до віддаленого API.

Таке рішення вирізняється надійністю, адже за класичного підходу “якщо немає інтернету, нічого не працює”, в той час як цей метод гарантує повернення хоча б останніх відомих даних. Механізм fallback-парсингу знижує ризик краху застосунку через пошкоджений або частково завантажений кеш, підвищуючи стійкість і комфорт користувача.

Виконання роботи методу складається з таких кроків:

1. Менеджер зчитує з DataStore два значення: JSON-рядок з раніше закешованими курсами та мітку часу останнього оновлення.

2. Обчислюється, чи минуло більше 24 годин від останнього запису, або чи сам JSON відсутній.

3. Якщо кеш існує і він актуальний, відбувається перехід до кроку 4. Інакше виконується асинхронний запит до віддаленого сервісу через Retrofit у ко-

рутині, результат перетворюється в ExchangeRateResponse.

4. Менеджер намагається обробити JSON за допомогою Gson. У разі успішної обробки, отримані курси валют повертаються відразу в інтерфейс користувача. У разі помилки відбувається автоматичний повторний запит до API.

5. Якщо викликано API (у кроці 3 або через помилку парсингу), менеджер записує свіжий JSON і нову мітку часу в DataStore.

6. Отриманий результат (карта валют і курсів) повертається для відображення користувачеві.

Блок-схема роботи цього методу наведена на рисунку 2.5.

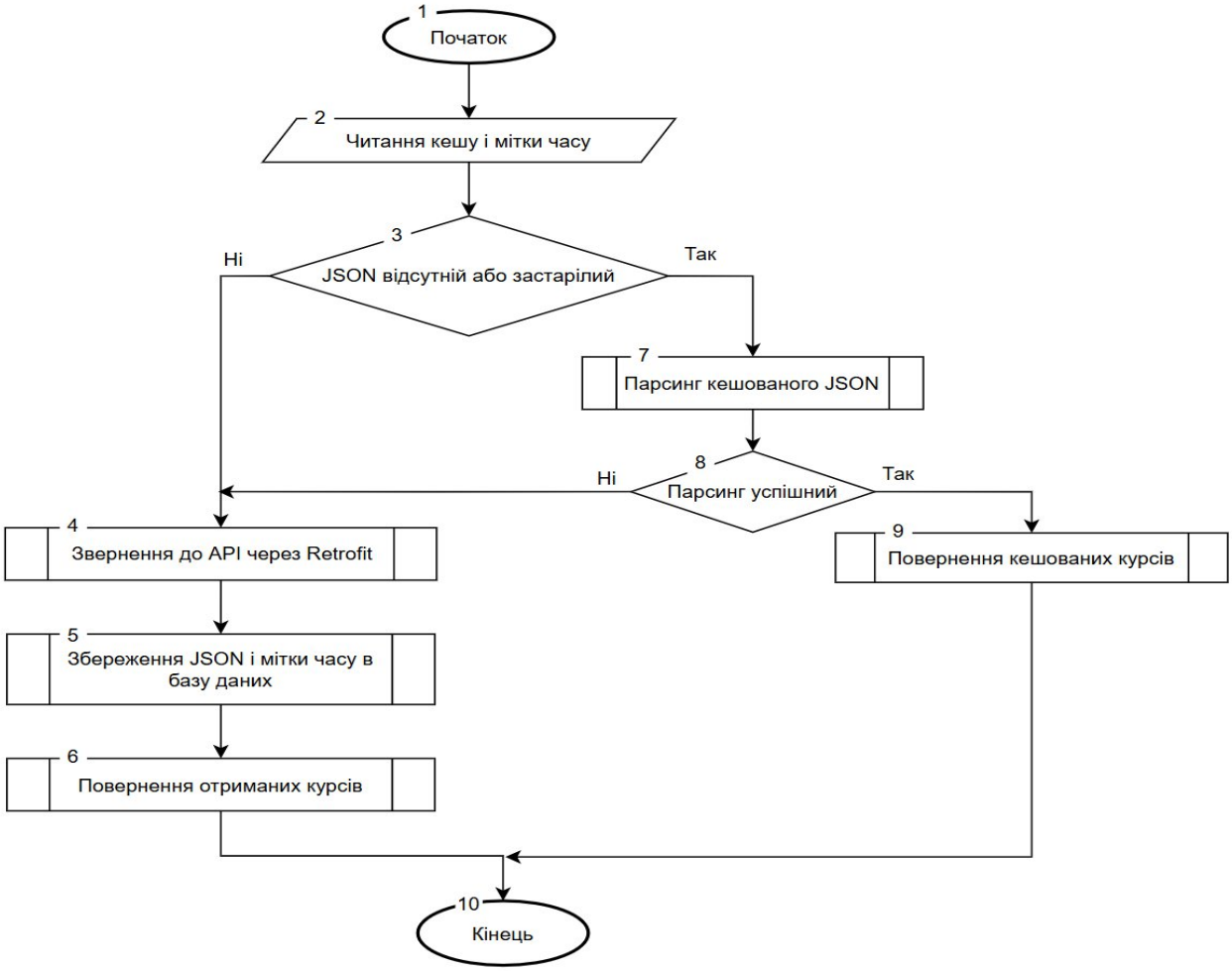

Рисунок 2.5 – Блок-схема роботи методу офлайн-менеджера валют

Завдяки цьому методу менеджер валют забезпечує надійність роботи та стійкість до помилок. Користувачі отримують останні доступні курси валют під

час виконання запиту, система автоматично відновлюється при пошкодженні кешу й оновлює інформацію лише за реально необхідних умов, що суттєво підвищує загальну стабільність застосунку.

2.5 Розробка алгоритмів роботи застосунку

Алгоритм конвертації валют полягає у перерахунку суми з однієї валюти в іншу на основі актуальних курсів обміну. Метод convertCurrency спочатку знаходить курси для вихідної та цільової валюти у внутрішньому словнику курсів, отриманих із зовнішнього API. Якщо обидва курси знайдені, то сума переводиться спочатку у базову валюту (USD чи EUR залежно від API) і потім у цільову валюту. У разі відсутності курсів обміну метод повертає початкову суму без змін. Таким чином, алгоритм забезпечує гнучку конвертацію валют у будь-якому напрямку залежно від актуальних даних.

Блок-схема алгоритму конвертації валют наведена на рисунку 2.6.

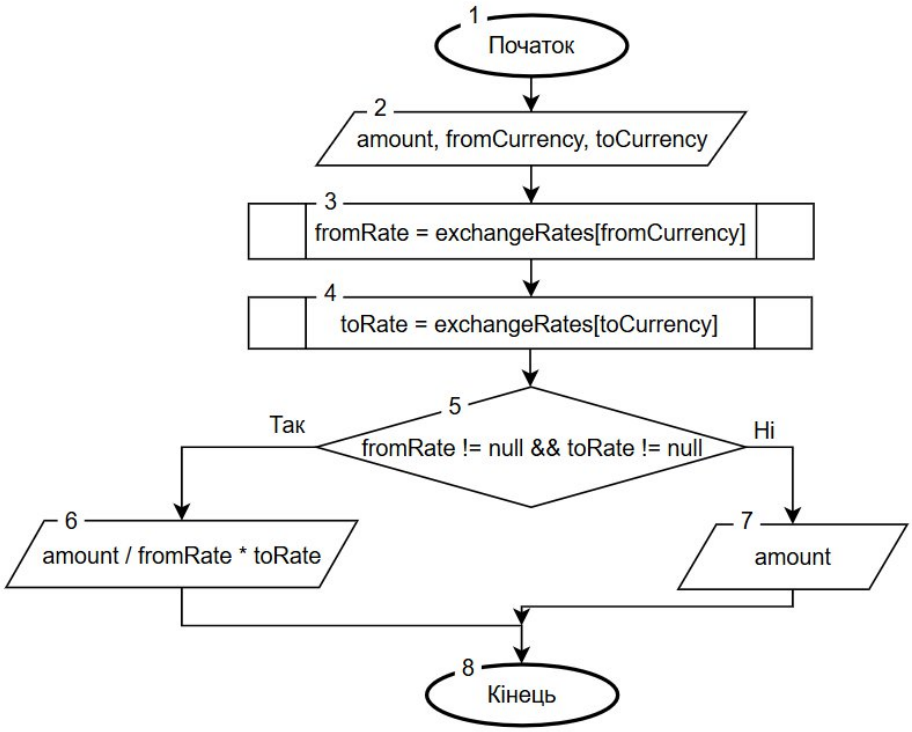

Рисунок 2.6 – Блок-схема алгоритму конвертації валют

Алгоритм отримання угод, пов'язаних із конкретним клієнтом, реалі-

зований за допомогою запиту до локальної бази даних через Room DAO. Метод getDealsByClientId отримує ідентифікатор клієнта і виконує SQL-запит до таблиці deals, вибираючи всі записи, де clientId збігається з переданим значенням. Це дозволяє легко збирати усю історію угод клієнта для подальшого відображення у додатку або аналізу. Завдяки використанню корутин метод працює асинхронно, що не блокує основний потік програми.

Блок-схема алгоритму отримання списку угод для конкретного клієнта наведена на рисунку 2.7.

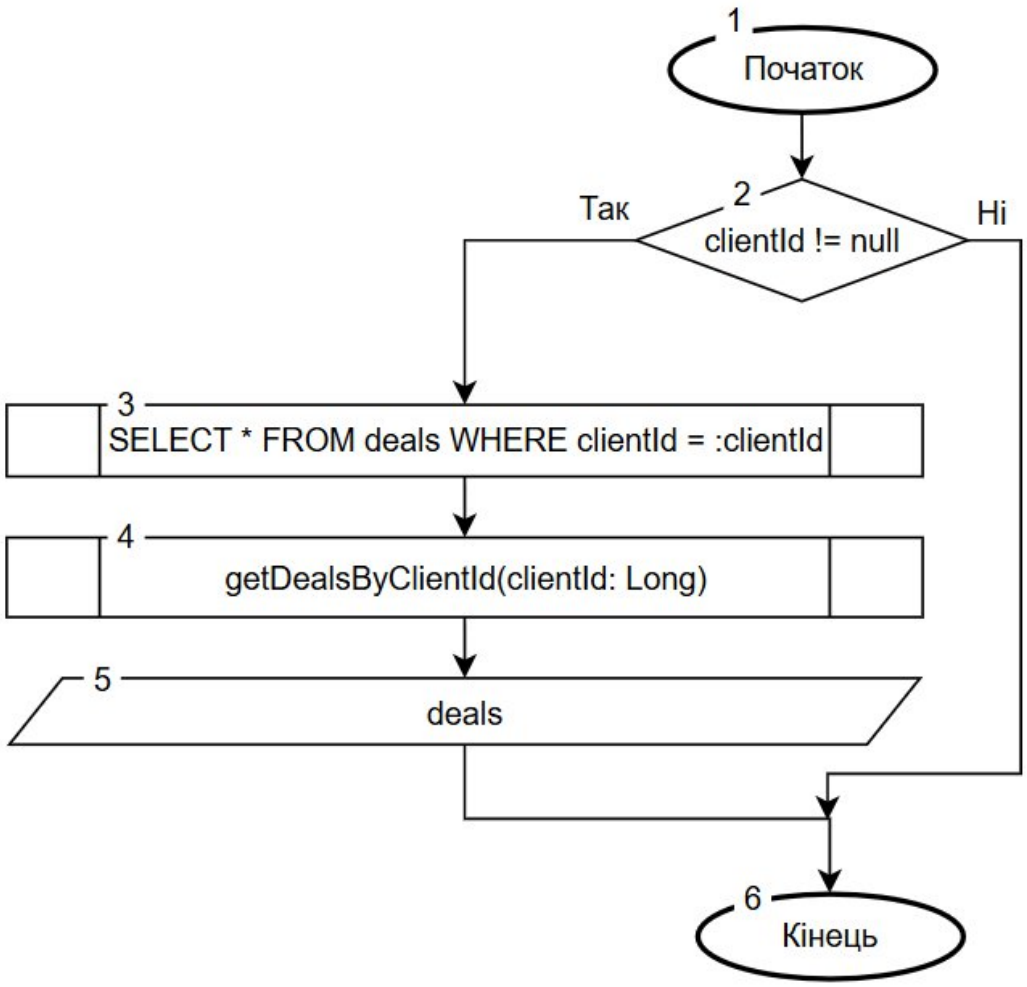

Рисунок 2.7 – Блок-схема алгоритму отримання списку угод для конкретного клієнта

Алгоритм підрахунку активних завдань полягає у виконанні SQL-запиту, який рахує записи у таблиці tasks для конкретного клієнта, де поле isCompleted

має значення 0 (тобто завдання ще не завершене). Метод getActiveTasksCountForClient викликає відповідний запит у DAO і повертає загальну кількість таких завдань. Цей підрахунок корисний для побудови звітів чи оперативного відображення навантаження на клієнта в інтерфейсі програми.

Блок-схема цього алгоритму наведена на рисунку 2.8.

Діаграма послідовності запиту курсів демонструє детальний сценарій отримання курсів валют при відкритті відповідного екрану. Спочатку користувач ініціює відкриття екрану в MainActivity, що призводить до виклику CurrencyManager.getRates(context). Далі менеджер перевіряє наявність і «свіжість» кешованого JSON у DataStore: у разі позитивного результату він читає дані із сховища, парсить їх через Gson і повертає карту курсів.

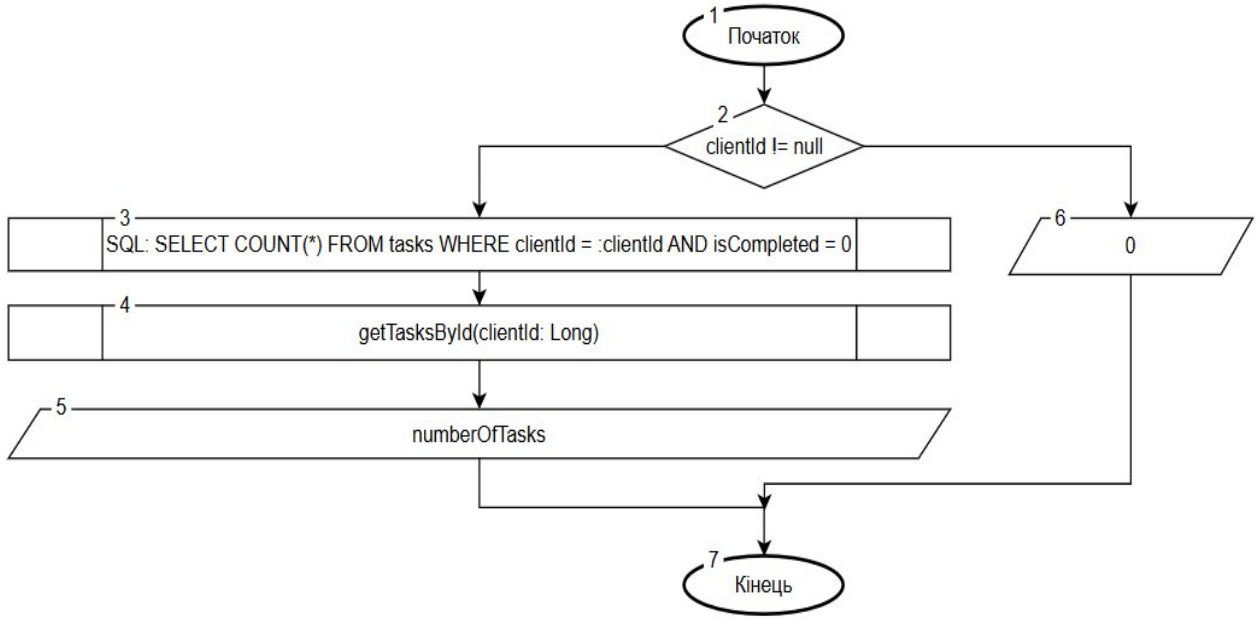

Рисунок 2.8 – Блок-схема алгоритму підрахунку активних завдань

Якщо ж кеш відсутній або сплив його термін актуальності у 24 години, менеджер звертається до віддаленого API (ExchangeRateApiService) через Retrofit. Отримавши відповідь, він зберігає її в DataStore (JSON + мітка часу) і повертає у інтерфейс користувача. Такий підхід гарантує і прозорість послідовності викликів, і коректну обробку всіх сценаріїв. Діаграма послідовності запиту курсів наведена на рисунку 2.9.

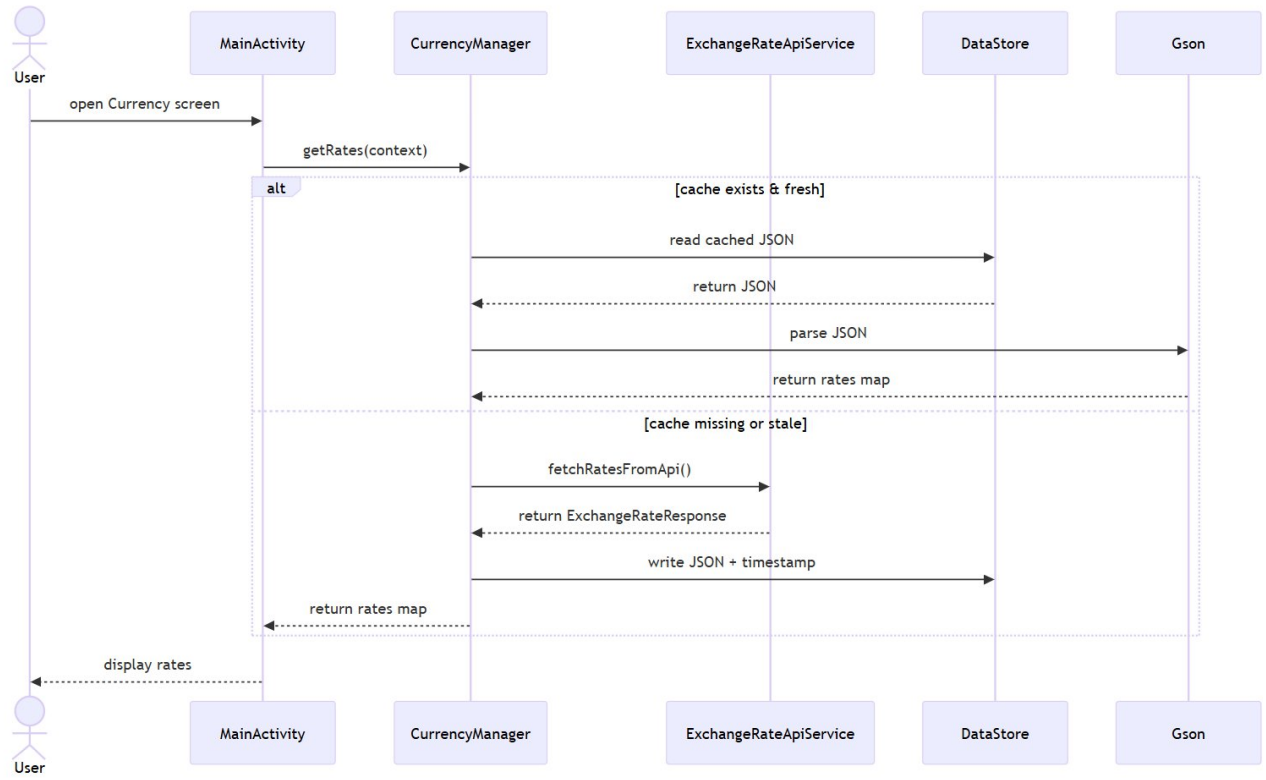

Рисунок 2.9 — Діаграма послідовності запиту курсів

Таким чином, розроблені алгоритми реалізують основний функціонал CRM-системи для малого бізнесу.

2.6 Висновки

Для вирішення задачі розробки структури CRM-системи було реалізовано архітектуру застосунку за шаблоном MVVM з чітким поділом на модулі, що відповідають за дані, логіку та інтерфейс. Модель системи розроблено на основі кількох ключових сутностей з урахуванням їх зв’язків. Розроблені алгоритми роботи системи включають метод формування пріоритетів завдань на основі ключових слів і статистики виконання, а також метод офлайн-менеджера валют, який гарантує стійкий доступ до курсів навіть за відсутності мережі. Це забезпечило логічну і функціональну базу для реалізації системи.

3 РОЗРОБКА СИСТЕМИ

3.1 Варіантний аналіз та вибір мови програмування

Kotlin [16] – сучасна статично типізована мова програмування, розроблена компанією JetBrains, що працює на віртуальній машині Java (JVM). Це офіційна мовою для розробки Android-додатків, проте також широко використовується для серверних застосунків. Kotlin надає розробникам суттєві переваги при створенні CRM-систем завдяки своїй виразності та лаконічності. Мова пропонує функціональні можливості, такі як функції вищого порядку, лямбда-вирази та неявну типізацію, що дозволяє писати більш чіткий та підтримуваний код. Важливою перевагою є повна сумісність з Java, що забезпечує доступ до всіх існуючих Java-бібліотек та фреймворків.

Для розробки CRM-систем Kotlin пропонує низку ефективних інструментів. Null-безпека на рівні мови значно зменшує ризик виникнення помилок NullPointerException, що підвищує стабільність системи. Корутини Kotlin забезпечують елегантний підхід до асинхронного програмування, що критично важливо для CRM з високими вимогами до продуктивності та чутливості інтерфейсу.

JavaScript [17] з використанням Node.js є популярним вибором для розробки повноцінних веб-застосунків, включаючи CRM-системи. Ця мова дозволяє використовувати один і той самий код як на клієнтській, так і на серверній стороні. Node.js забезпечує неблокуючу модель введення/виведення, що робить його ефективним для систем з високим навантаженням. Екосистема npm містить тисячі готових пакетів та бібліотек для різних функцій CRM, таких як аутентифікація, обробка платежів, інтеграція з зовнішніми API тощо. Для створення CRM-систем JavaScript пропонує широкий вибір фреймворків, таких як Express.js для серверної частини та React або Angular для клієнтської. Динамічна типізація JavaScript забезпечує швидку розробку прототипів, хоча може створювати складнощі при масштабуванні складних проектів.

Python [18] – інтерпретована мова програмування високого рівня з

простим синтаксисом та значним фокусом на читабельності коду.

Python пропонує фреймворки для веб-розробки, таких як Django та Flask, які включають готові компоненти для аутентифікації користувачів, адміністративні панелі та інструменти для роботи з базами даних. Мова відзначається потужними можливостями для аналізу даних та машинного навчання (з бібліотеками, такими як Pandas, NumPy та scikit-learn), що може бути використано для розробки аналітичних модулів CRM.

У таблиці 3.1 наведено порівняння функціональних можливостей описаних мов програмування.

Таблиця 3.1 – Порівняння мов програмування для розробки CRM-системи для малого бізнесу

Критерії	Kotlin	JavaScript	Python
Статична типізація	+	-	-
Безпека від NullPointerException	+	-	-
Підтримка функціонального програмування	+	+	+
Асинхронне програмування	+	+	+
Підтримка мікросервісної архітектури	+	+	+
Ефективна робота з базами даних	+	+	+
Бібліотеки для мультиплатформної розробки	+	+	-
Сумарний бал	7	5	4

Таким чином, Kotlin є найкращим вибором для розроби системи управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

3.2 Розробка бази даних

База даних складається з п'яти основних таблиць, які забезпечують функціональність управління клієнтами, завданнями, подіями, дзвінками та угодами.

Таблиця clients є центральною у структурі бази даних та містить інформацію про клієнтів компанії. Ця таблиця пов'язана з іншими таблицями (tasks, calls, deals) зв'язками "один-до-багатьох", що дозволяє відстежувати всю взаємодію з кожним клієнтом. Поля таблиці clients наведені у таблиці 3.2.

Таблиця 3.2 — Поля таблиці clients

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT, NOT NULL	Унікальний ідентифікатор клієнта
name	TEXT	NOT NULL	Ім'я клієнта
email	TEXT	NOT NULL	Електронна адреса клієнта
phone	TEXT	NOT NULL	Телефонний номер клієнта

Таблиця tasks зберігає інформацію про завдання, які необхідно виконати стосовно певних клієнтів. Кожне завдання має пріоритет та статус виконання, а також прив'язане до клієнта. Поля таблиці clients наведені у таблиці 3.3.

Таблиця 3.3 — Поля таблиці tasks

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT, NOT NULL	Унікальний ідентифікатор завдання
title	TEXT	NOT NULL	Назва або опис завдання
date	INTEGER	NOT NULL	Дата виконання завдання (у форматі timestamp)

Продовження таблиці 3.3

Поле	Тип даних	Обмеження	Опис
priority	INTEGER	NOT NULL	Пріоритет завдання (числове значення)
clientId	INTEGER	NOT NULL, FOREIGN KEY	Ідентифікатор клієнта, до якого відноситься завдання
completed	INTEGER	NOT NULL	Статус виконання завдання (0 - не виконано, 1 - виконано)

Таблиця events містить дані про заплановані події в календарі. На відміну від інших таблиць, події не прив'язані до конкретних клієнтів, що дозволяє використовувати їх для внутрішніх заходів компанії або загальних подій. Поля таблиці clients наведені у таблиці 3.4.

Таблиця 3.4 — Поля таблиці events

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT, NOT NULL	Унікальний ідентифікатор події
title	TEXT	NOT NULL	Назва або опис події
timeStart	INTEGER	NOT NULL	Час початку події (у форматі timestamp)
timeEnd	INTEGER	NOT NULL	Час закінчення події (у форматі timestamp)
date	INTEGER	NOT NULL	Дата події (у форматі timestamp)
completed	INTEGER	NOT NULL	Статус виконання події (0 - не відбулася, 1 - відбулася)

Таблиця calls відстежує заплановані телефонні дзвінки клієнтам, включаючи час та дату дзвінка, а також його статус виконання. Поля таблиці clients наведені у таблиці 3.5.

Таблиця 3.5 — Поля таблиці calls

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT, NOT NULL	Унікальний ідентифікатор дзвінка
title	TEXT	NOT NULL	Тема або мета дзвінка
time	INTEGER	NOT NULL	Час дзвінка (у форматі timestamp)
date	INTEGER	NOT NULL	Дата дзвінка (у форматі timestamp)
clientId	INTEGER	NOT NULL, FOREIGN KEY	Ідентифікатор клієнта, якому здійснюється дзвінок

Таблиця deals відстежує комерційні угоди з клієнтами, включаючи інформацію про суму, валюту та дату угоди. Поля таблиці deals наведені у таблиці 3.6.

Таблиця 3.6 — Поля таблиці deals

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT, NOT NULL	Унікальний ідентифікатор угоди
title	TEXT	NOT NULL	Назва або опис угоди
amount	INTEGER	NOT NULL	Сума угоди
currency	TEXT	NOT NULL	Валюта угоди
clientId	INTEGER	NOT NULL, FOREIGN KEY	Ідентифікатор клієнта, з яким укладено угоду

Поле	Тип даних	Обмеження	Опис
date	INTEGER	NOT NULL, DEFAULT 0	Дата укладання угоди (у форматі timestamp)

На рисунку 3.1 наведено схему бази даних, що відображає зв’язки між описаними таблицями.

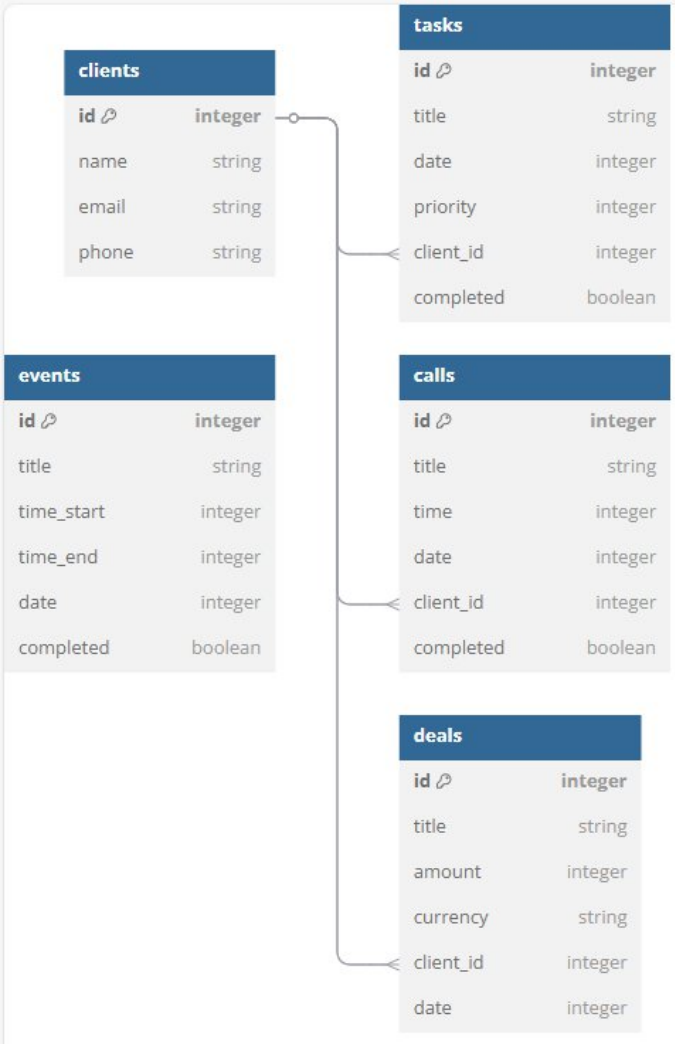

Рисунок 3.1 — Схема бази даних

Розроблена база даних забезпечує основні функції CRM-системи для малого бізнесу, дозволяючи відстежувати клієнтів, керувати завданнями, планувати події та дзвінки, а також фіксувати комерційні угоди. Використання зовнішніх ключів та індексів оптимізує продуктивність запитів та забезпечує цілісність даних.

3.3 Розробка головного модуля застосунку

Головний модуль застосунку дозволяє переглядати заплановані події та завдання на кожен день, створювати нові та відмічати виконання поставлених завдань.

Бізнес-логіка цього модуля реалізована у класі DayViewModel, який викликає запити з бази даних для додавання, редагування, вилучення записів про заплановані зустрічі, завдання та дзвінки, а також отримує записи про уже існуючі. Місце класу DayViewModel в архітектурі застосунку наведено на рисунку 3.2. Лістинг класу подано у додатку В.

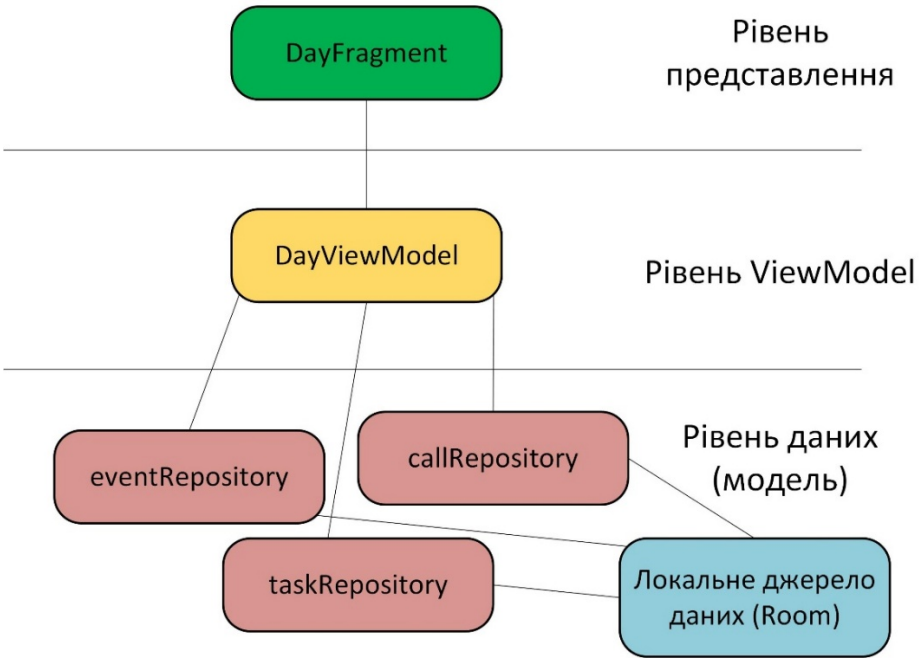

Рисунок 3.2 — Архітектура головного модуля

Структурна схема інтерфейсу головного модуля застосунку наведена на рисунку 3.3.

Елементи головного вікна системи:

1. Календар.

2. Кнопка додавання подій.

3. Список подій.

4. Кнопка додавання завдань.

5. Список завдань

6. Навігаційна панель.

У розробленому застосунку для реалізації зручного та ефективного відображення інформації про щоденні завдання, дзвінки та події на головному екрані використовується компонент ViewPager2 (рисунок 3.4). Основною ідеєю є організація інтерфейсу у вигляді горизонтальної каруселі фрагментів, де кожен фрагмент відповідає певній календарній даті.

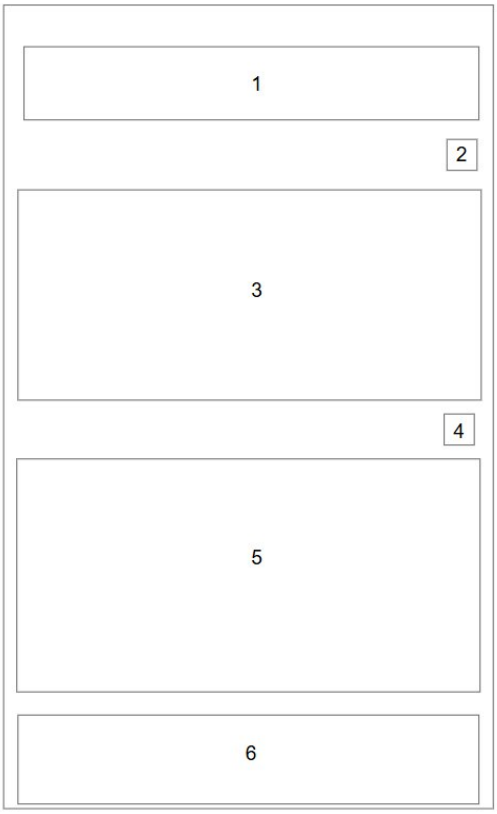

Рисунок 3.3 — Структурна схема інтерфейсу головного модуля системи

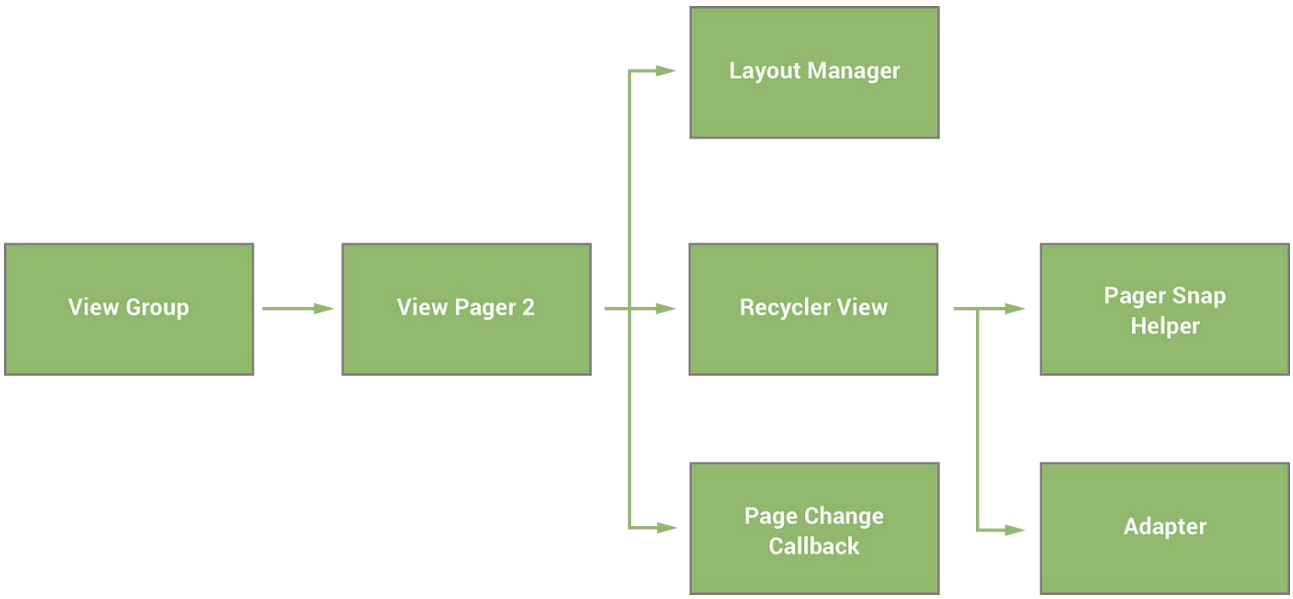

Рисунок 3.4 – Структурна схема ViewPager2

На кожній сторінці ViewPager2 відображається окремий DayFragment, у якому згруповано усі дані, пов’язані з конкретним днем: список запланованих завдань, запланованих дзвінків та подій. Таким чином, користувач може переглядати інформацію по днях, просто гортуючи сторінки вліво або вправо. Такий підхід імітує безперервну шкалу часу і дозволяє зручно переходити між минулими, поточними та майбутніми датами.

Для зберігання прив’язки між позицією у ViewPager2 та відповідною датою реалізовано спеціальну ViewModel, яка дозволяє отримувати об’єкт LocalDate для будь-якої позиції та навпаки. Це дозволяє легко реалізувати такі дії, як повернення до поточної дати або відкриття сторінки за вибраною датою з календаря.

Завдяки ViewPager2 досягається інтуїтивно зрозумілий, адаптивний інтерфейс, який підтримує як динамічне завантаження фрагментів, так і ефективне повторне використання вмісту. Крім того, використання FragmentStateAdapter забезпечує збереження стану кожної сторінки, що позитивно впливає на продуктивність та зручність користування.

Розроблений головний модуль застосунку дозволяє вести облік усіх бізнес-справ та процесів у малому бізнесі, планувати зустрічі, дзвінки та завдання.

3.4 Реалізація методу формування пріоритетів завдань

Однією з ключових функцій розробленої CRM-системи є адаптивне формування пріоритетів завдань на основі аналізу їх вмісту. Цей метод дозволяє автоматично визначати ступінь важливості завдання, що сприяє підвищенню ефективності планування та розстановки пріоритетів.

Основою запропонованого методу є використання евристичного підходу до оцінки терміновості завдання, який базується на аналізі ключових слів у його заголовку або описі. Для витягу ключових слів з тексту завдання використовується алгоритм RAKE (Rapid Automatic Keyword Extraction) (рисунок 3.5). RAKE — це відомий алгоритм обробки природної мови, що виконує безнаглядове виділення важливих фраз на основі статистичних і структурних особливостей тексту. Алгоритм не потребує попереднього навчання і працює з використанням списку стоп-слів, що дозволяє ефективно виділяти змістовні фрази.

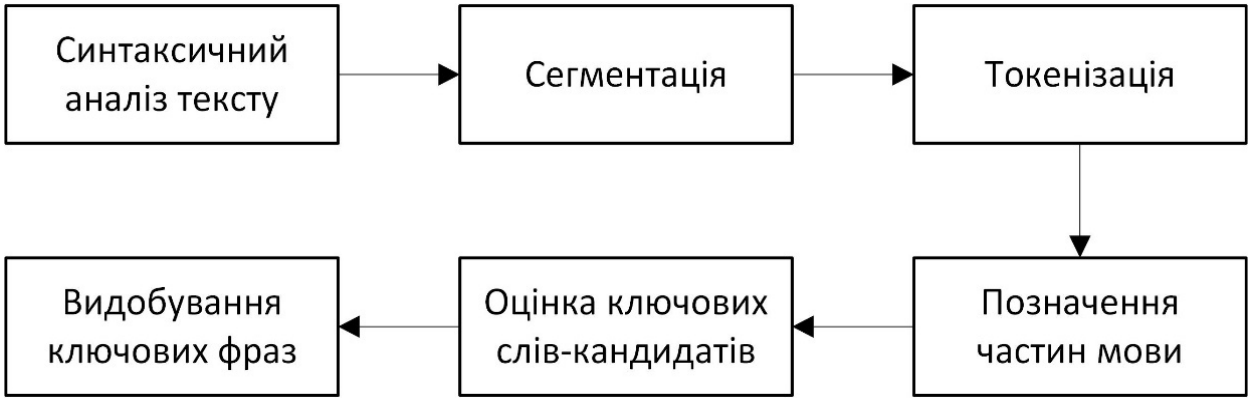

Рисунок 3.5 – Алгоритм RAKE

Оскільки RAKE реалізовано мовою Python, для інтеграції алгоритму в Android-додаток, написаний Kotlin, було використано Chaquopy — інструмент, що дозволяє запускати Python-код у межах Android-застосунку. Таким чином, обробка ключових слів відбувається на Python-стороні, а результат повертається у Kotlin-код для подальшого використання.

На етапі ініціалізації RAKE із Python-файлу, застосунок довантажує необхідні ресурси (наприклад, корпус стоп-слів із бібліотеки NLTK), виконує оброб-

ку введеного тексту, а потім повертає список ключових фраз до основного застосунку.

Розроблений алгоритм працює за такими етапами:

1) Витяг ключових слів: зі заголовку чи опису нового завдання витягуються найбільш змістовні фрази за допомогою RAKE.

2) Отримання статистики: по кожному ключовому слову з локальної бази даних витягується статистика — кількість випадків, коли слово було асоційоване з відкладеним або виконаним завданням.

3) Обчислення оцінки: для кожного слова обчислюється коефіцієнт терміновості як відношення postponed / (postponed + completed), а також додається бонус, якщо слово було використано нещодавно.

4) Агрегація: обчислюється середнє значення оцінки серед усіх ключових слів.

5) Призначення пріоритету: залежно від середньої оцінки призначається рівень пріоритету як показано у таблиці 3.7.

Таблиця 3.7 – Призначення пріоритету завдання

Середня оцінка	Пріоритет
≥ 0.7	Високий
≥ 0.4	Середній
< 0.4	Низький

Такий підхід дозволяє автоматизовано оцінювати терміновість завдань ще на етапі їх створення, спираючись на статистичний аналіз попередніх дій користувача. Це сприяє кращому плануванню, зменшенню кількості відкладених справ і підвищенню загальної ефективності управління завданнями.

3.5 Розробка менеджера валют та модуля угод

Модуль угод та доходів дозволяє керувати укладеними угодами для бізнесу з метою їх подальшої обробки та виконання, збирання історії виконаних угод для їх подальшої аналітики та визначення ефективності й прибутковості бізнесу.

Для управління угодами реалізовано клас DealsFragment, який реалізує інтерфейс застосунку та ключову бізнес-логіку. Функція calculateIncome(), що належить цьому класу, дозволяє підраховувати увесь дохід з наявних в базі даних виконаних угод. Вона отримує список угод, та проходить через нього циклом, і для кожного елементу списку отримує валюту, в якій здійснювалась угода, переводить її за актуальним курсом, отримує суму доходу у доларах США та додає до суми доходів. Функція calculateIncome() наведена на рисунку 3.6.

```
private fun calculateIncome(list: List<DealWithClient>): Float? {
    var income = 0f

    list.forEach {
        val currencyCode = it.deal.currency.uppercase()
        val currencyRate = CurrencyManager.currencyRates?.get(currencyCode)?.toFloat()
        if (currencyRate == null) return null

        val amount = it.deal.amount
        income += amount / currencyRate
    }

    return income
}
```

Рисунок 3.6 — Функція calculateIncome() для підрахунку доходів

Для підрахунку доходу за місяць реалізовано клас MonthlyIncome. Він містить два поля: month та total, які відповідно зберігають місяць та дохід за цей місяць, та функцію getMonthlyIncome, яка групує усі існуючі записи про доходи за місяцями, та повертає інформацію про той місяць, який запитує користувач. Клас MonthlyIncome наведено на рисунку 3.7.

```
data class MonthlyIncome(
    val month: YearMonth,
    val total: Int
) {
    companion object {
        fun getMonthlyIncome(deals: List<DealWithClient>): List<MonthlyIncome> {
            return deals
                .groupBy { YearMonth.from(it.deal.date) }
                .map { (month, dealsInMonth) ->
                    MonthlyIncome(
                        month = month,
                        total = dealsInMonth.sumOf { it.deal.amount }
                    )
                }
                .sortedBy { it.month }
        }
    }
}
```

Рисунок 3.7 — Клас MonthlyIncome

Вікно укладених угод організоване у вигляді списку укладених угод, відсортованих у календарному порядку і згрупованих за місяцями. Для опису кожної угоди розроблено інтерфейс, структурна схема якого наведена на рисунку 3.8.

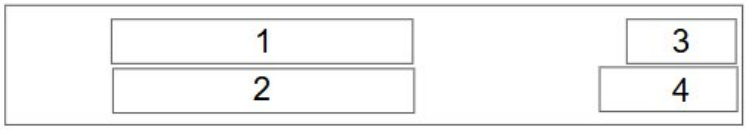

Рисунок 3.8 — Структурна схема інтерфейсу елемента запису про угоду

Складові елемента запису про угоду:

1. Назва угоди.
2. Працівник, що вів угоду.
3. Дата угоди.
4. Сума угоди.

Структурна схема інтерфейсу вікна укладених угод наведена на рисунку 3.9.

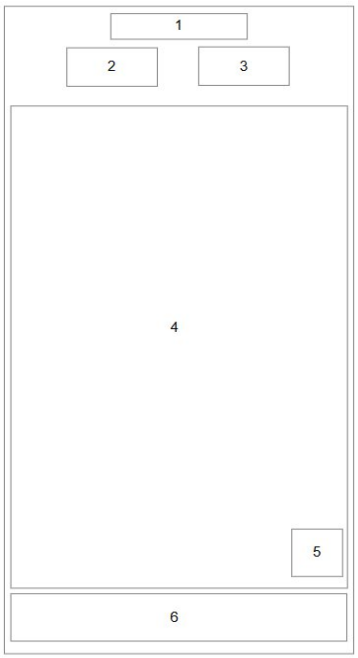

Рисунок 3.9 — Структурна схема вікна угод

Елементи вікна угод:

1. Назва вікна.

2. Загальна сума доходу.

3. Дохід за останній місяць.

4. Список останніх угод.

5. Кнопка «Додати угоду».

6. Навігаційна панель.

У фрагменті зі списком угод реалізовано систему відображення фінансових показників, яка забезпечує зручну візуалізацію доходів у єдиній валюті — доларах США. Оскільки самі угоди можуть бути створені у різних валютах, система використовує офлайн-конвертер валют, що працює на основі локально кешованих курсів. Це дозволяє уникати проблем із відображенням актуального доходу при відсутності мережі або тимчасовій недоступності API.

Особливості реалізації підходу полягають в наступному:

1) Ініціалізація менеджера курсів відбувається один раз при запуску застосунку. Він автоматично зчитує курси валют з DataStore, а в разі їхньої відсутності або застарілості – виконує фоновий запит до API через Retrofit.

2) Кешовані дані зберігаються у вигляді JSON-рядка, що містить об’єкт з ключами-кодами валют та відповідними курсами. Ці дані надалі використовуються для перерахунку кожної суми угоди у долари США.

3) Надійність обробки реалізована завдяки використанню Gson для парсингу кешованих даних. У разі виникнення винятку або пошкодженого формату виконується fallback-запит до API з подальшим оновленням кешу.

4) Конвертація угод у долари відбувається безпосередньо у фрагменті, де відображаються списки. Для кожної угоди береться її сума, курс відповідної валюти з менеджера, після чого обчислюється еквівалент у USD.

5) Верхня панель доходу динамічно оновлюється при кожному зміненні списку або нових даних. Вона містить загальний дохід у USD та дохід за поточний місяць у USD. Значення обчислюються на основі сконвертованих у долари угод, фільтрованих за відповідними датами.

Розроблений модуль угод та конвертер валют дозволяють відстежувати дохід з виконаних угод, самі угоди, отримувати інформацію про доходи по місяцям для кращої звітності.

3.6 Висновки

Задачу розробки інтерфейсу користувача було вирішено з урахуванням потреб малого бізнесу, що дозволяє користувачеві отримати доступ до списку завдань, дзвінків, подій та угод у зручному форматі. Було розроблено функціонал системи: керування записами, адаптивне визначення пріоритетів завдань, стійкий до помилок обробник курсів валют. Усі компоненти працюють узгоджено, забезпечуючи користувачеві повноцінну CRM-систему.

4 ТЕСТУВАННЯ CRM-СИСТЕМИ ДЛЯ МАЛОГО БІЗНЕСУ

4.1 Аналіз методів тестування

Для перевірки коректності роботи репозиторіїв, DAO та ViewModel доцільно використовувати юніт тести на основі JUnit [19] і Mockito (або MockK) [20]. Репозиторій можна тестувати, підміняючи реальний DealDao чи ClientDao макетами, і перевіряти, що методи insert(), getAllDeals() або getAllClients() коректно повертають й обробляють дані.

Робота з локальною базою даних потребує перевірки міграцій та запитів. Використовуючи вбудовану базу даних, розробляються інтеграційні тести, які створюють екземпляр AppDatabase, виконують CRUD операції через DAO і порівнюють результати з очікуваними. Це дозволяє виявити помилки в анотаціях @Relation, некоректні SQL запити або проблеми з версіями схем.

Для перевірки реального досвіду користувача (натискання кнопка, прокрутка списків, навігація між фрагментами) використовується Espresso [21]. Наприклад, тест може відкривати ContactsFragment, додавати новий контакт через діалогову форму, перевіряти, що запис з’явився в RecyclerView, а потім видаляти його. Такі тести проганяються на емуляторі або реальному пристрої, вони гарантують, що дані зі ViewModel дійсно прив’язуються до компонентів інтерфейсу.

Robolectric [22] дозволяє виконувати тестування логіки інтерфейсу користувача на JVM без емулятора. Це пришвидшує зворотний зв’язок під час розробки, але не замінює повноцінні Espresso тести.

Для комплексної перевірки застосунку (від логіки бази даних і API) використовується UI Automator [23]. Це забезпечує, що застосунок працює під різними версіями Android і на різних екранах, але такий процес вимагає суттєвих ресурсів і часу на виконання.

Кожен із цих підходів потрібно комбінувати: юніт тести забезпечать міцний фундамент бізнес логіки, інструментовані та Robolectric тести — швидкий відгук по інтерфейсу користувача.

Порівняльну характеристику описаних підходів до тестування наведено у таблиці 4.1.

Таблиця 4.1 – Порівняльна характеристика підходів до тестування

Метод тестування	Юніт тести	Інтеграційні тести	Інструментовані тести	Robolectric тести	UI Automator
Рівень	Логіка/бізнес-рівень	База даних	Інтерфейс користувача	UI логіка на JVM	Енд-ту-енд
Призначення	Перевірка окремих компонентів (ViewModel, репозиторії, DAO)	Перевірка реальної взаємодії з базою даних (CRUD, міграції, зв’язки)	Тестування взаємодії користувача з інтерфейсом	Перевірка логіки UI без емулятора	Перевірка роботи додатку як цілісної системи на реальному пристрої/емуляторі
Інструменти	JUnit, Mockito, MockK	JUnit, Room (AppDatabase)	Espresso	Robolectric	UI Automator
Переваги	Швидкі, ізольовані, легко підтримувати	Дозволяють виявити помилки в запитах і структурах таблиць	Реалістична поведінка, тестування всієї зв’язки ViewModel ↔ UI	Швидше виконання, не потребує пристрою	Перевірка застосунку в реальних умовах
Недоліки	Не охоплюють реальну взаємодію з інтерфейсом чи БД	Потребують більше часу на запуск	Потрібен емулятор або пристрій, повільніші за юніт тести	Не повністю відображає реальну поведінку Android	Дуже повільне виконання, складність написання та підтримки

Таким чином, комплексний підхід до тестування є критично важливим для забезпечення якості розроблюваного CRM-застосунку. Юніт тести дозволяють швидко виявити помилки у бізнес-логіці та взаємодії між компонентами, інструментовані тести — перевірити інтеграцію та поведінку інтерфейсу на реальних пристроях, а Robolectric забезпечує оперативну перевірку змін без запуску емулятора. Використання UI Automator гарантує стабільність роботи в умовах, максимально наближених до користувацьких. Такий підхід дозволяє ефективно підтримувати якість коду, швидко виявляти дефекти та зменшити ризики під час релізу нових версій програми.

4.2 Тестування системи

На рисунку 4.1 наведено головний «домашній» екран системи. Він містить календар, події на обраний день, завдання з пріоритетністю кожного, дозволяє додавати нові події та завдання.

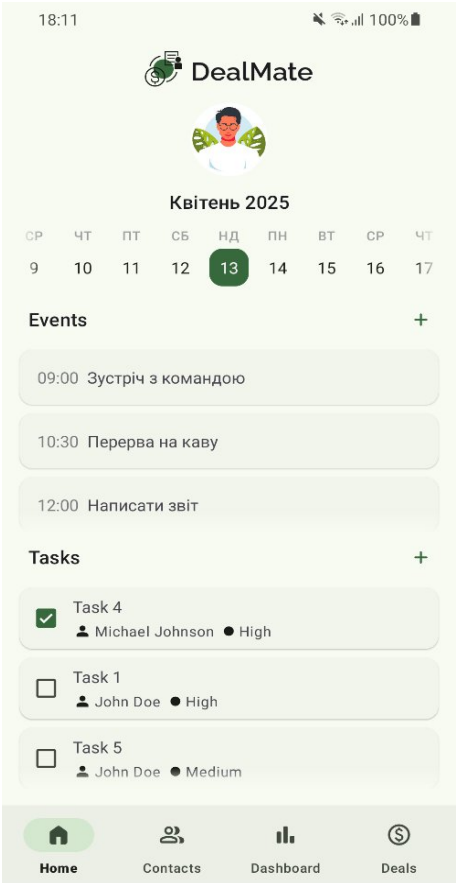

Рисунок 4.1 — Головне вікно системи

Крім цього, міститься також список нещодавніх дзвінків (рисунок 4.2).

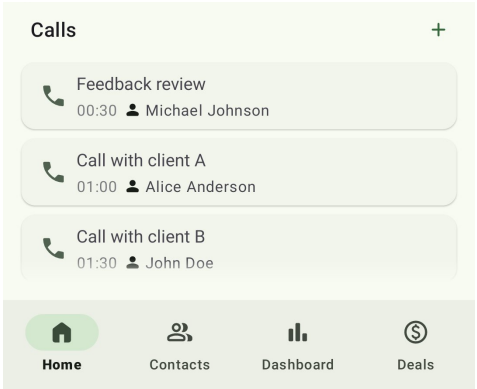

Рисунок 4.2 — Головне вікно системи (продовження)

При натисканні на події, відображаються її деталі, такі як дата та час завдання, та функціонал для редагування, видалення і позначення події як завершеної (рисунок 4.3).

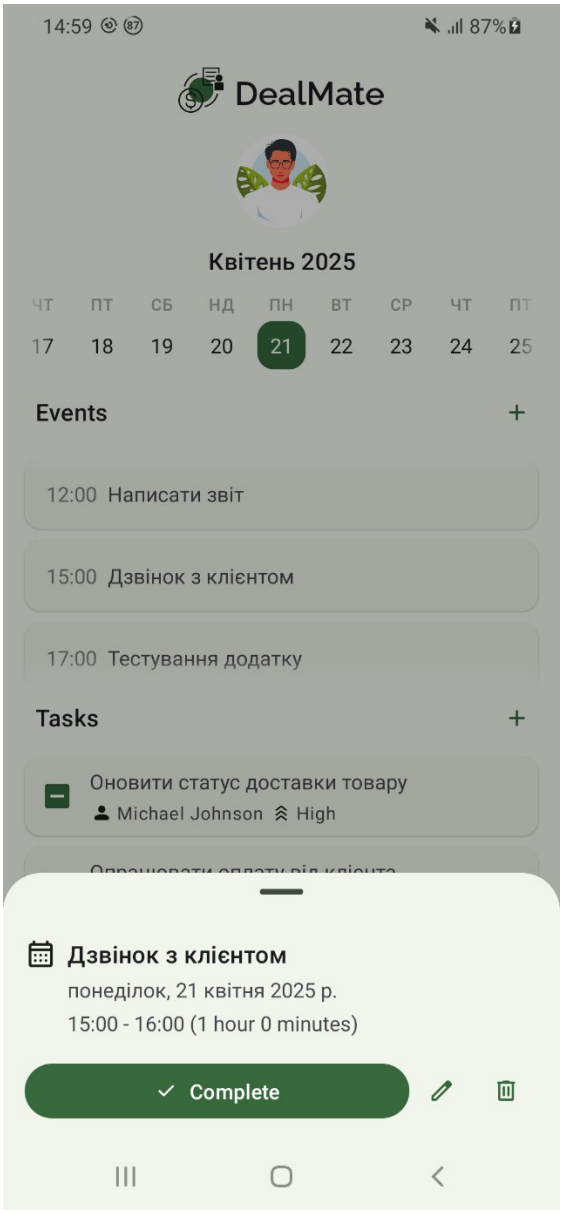

Рисунок 4.3 – Перегляд деталей події

При натисканні на завдання, відбувається перехід на новий екран, у якому відображається опис, дата завдання, пріоритет, клієнт, якого стосується завдання, та прогрес виконання. Аналогічно вікну з переглядом деталей події, запис про завдання можна видалити чи відредагувати (рисунок 4.4).

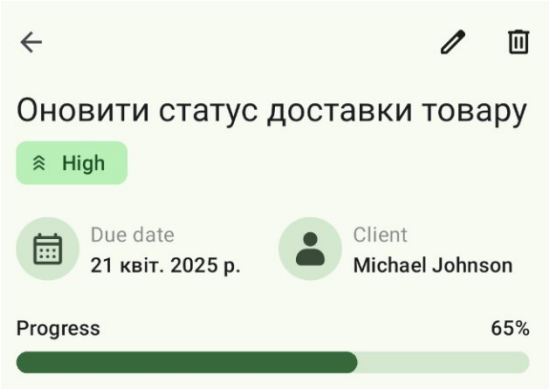

Рисунок 4.4 – Перегляд деталей завдання

Під час тестування модуля відображення списку ключових слів було перевірено коректність завантаження та візуалізації даних, що зберігаються у базі даних. Основною метою тестування було впевнитися, що користувач має доступ до повної інформації щодо кожного ключового слова, зокрема кількості завершених та відкладених завдань.

У результаті тестування застосунок успішно відобрає список ключових слів у зручному для перегляду форматі як показано на рисунку 4.5.

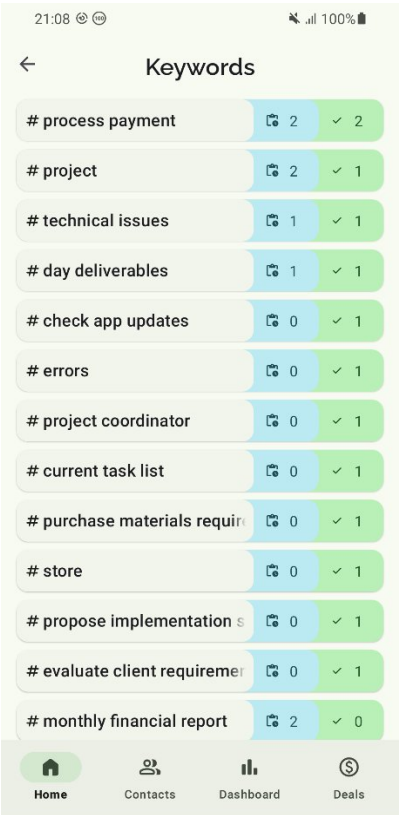

Рисунок 4.5 – Перегляд списку ключових слів

Окрім перевірки відображення ключових слів, під час тестування було також перевірено функцію автоматичного визначення пріоритету завдання. Ця функція покладається на наявні в базі даних ключові слова, щоб проаналізувати текст нового завдання й запропонувати відповідний рівень пріоритету (високий, середній або низький).

У рамках тестування було створено нове завдання з описом, який містив ключові слова, що вже присутні у базі (рисунок 4.6). Після додавання завдання система автоматично виконала аналіз його тексту, виявила ключові слова та, на основі заданої евристики, правильно визначила рівень пріоритету. Отриманий результат відповідав очікуванням, виходячи з того, які слова містив текст і яку статистику вони мали.

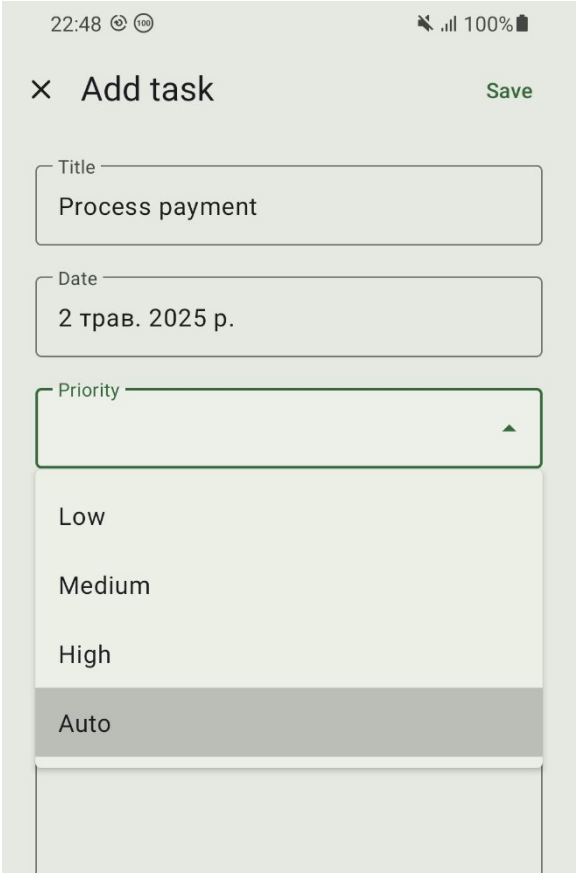

Рисунок 4.6 – Створення нового завдання

Таким чином, функція автоматичного визначення пріоритету завдань була успішно протестована в умовах, наближених до реального використання.

Екран контактів дозволяє вести записи контактів усіх необхідних співробітників та клієнтів. Записи з контактами містять імена контактів, номери телефонів, електронні адреси (рисунок 4.7).

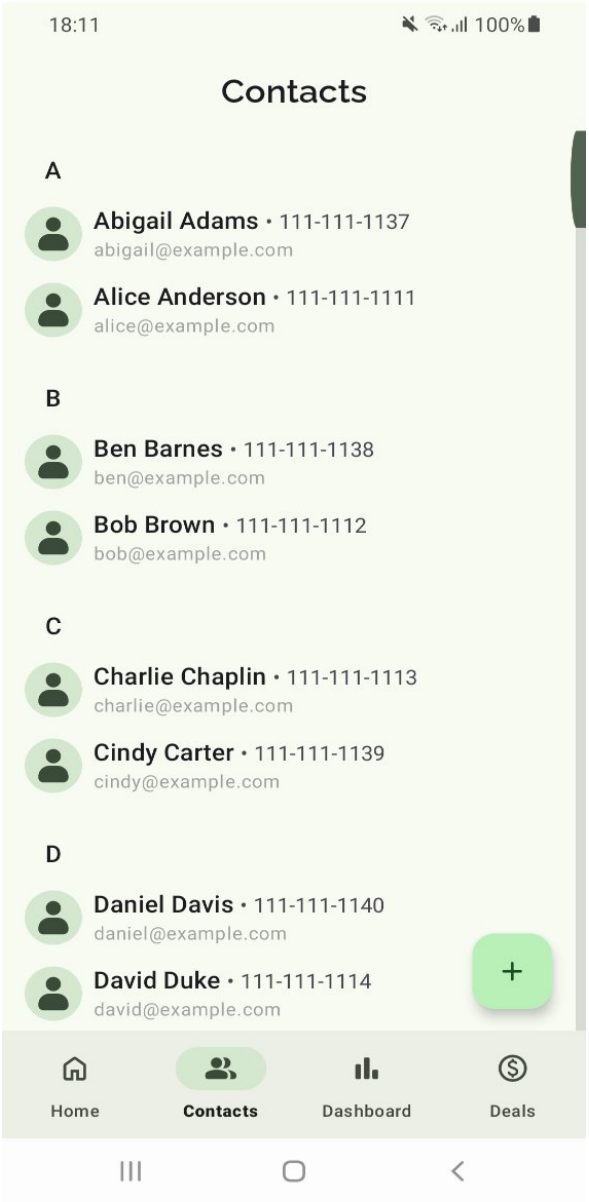

Рисунок 4.7 — Вікно контактів

Після натискання кнопки «+» у розділі дзвінків відкрився екран додавання нового дзвінка (рисунок 4.8). Було заповнено всі необхідні поля, після чого запис успішно зберігся, і новий дзвінок відобразився у загальному списку. Це підтверджує, що функція додавання дзвінків працює стабільно та без помилок.

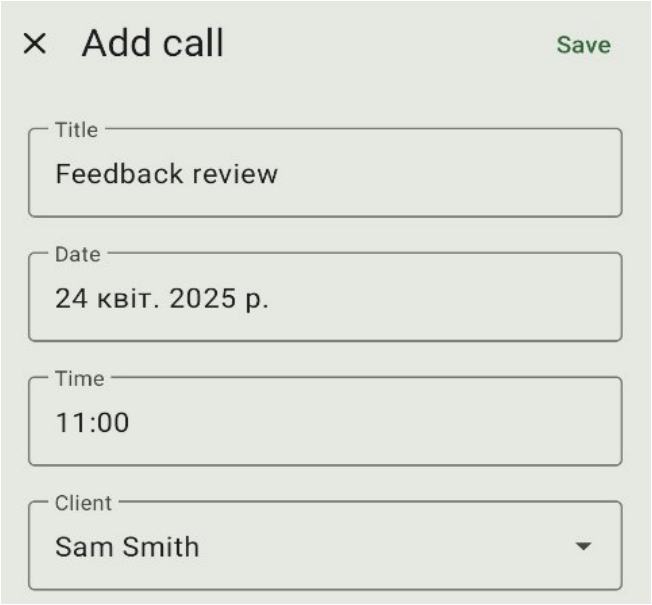

Рисунок 4.8 — Додавання запису про новий дзвінок

Аналогічно, під час додавання нового клієнта за допомогою екрану Add Client (рисунок 4.9), дані були успішно передані до бази даних. Новий клієнт з’явився у відповідному розділі CRM-системи, що свідчить про коректну роботу механізмів введення, обробки та збереження інформації.

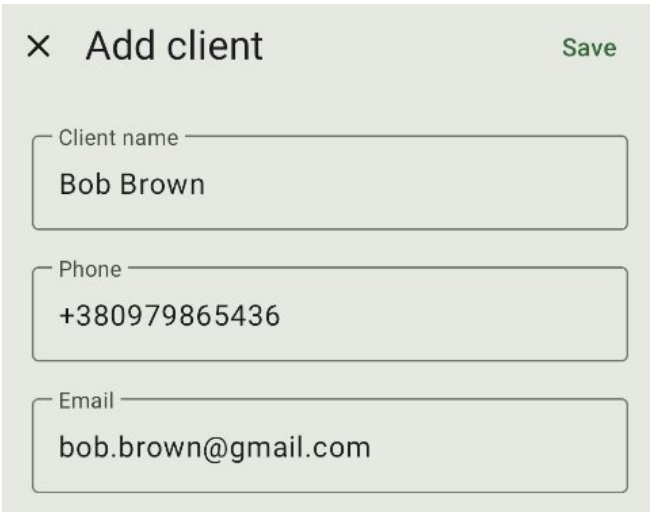

Рисунок 4.9 — Додавання нового клієнта

Екран укладених угод дозволяє вести облік усіх угод, що були укладені, клієнтів, з якими вони укладені, суму угоди та дату. Також вверху екрану міститься загальна сума угод за місяць та за весь час (рисунок 4.10).

Рисунок 4.10 — Вікно укладених угод

Для додавання нової угоди, натиснемо кнопку «+» у правій нижній частині екрану, після чого заповнимо форму, як на рисунку 4.9.

У результаті тестування функції додавання нової угоди було підтверджено її працездатність і зручність використання. Після натискання кнопки «+» у правому нижньому куті відкрилася форма створення угоди (рисунок 4.11), що відповідає очікуваному сценарію користувача.

Усі необхідні поля форми були успішно заповнені, після чого інформація збереглася в базу даних, і нова угода з’явилася у списку. Візуальне відображення, логіка переходів між екранами та збереження даних працювали без помилок.

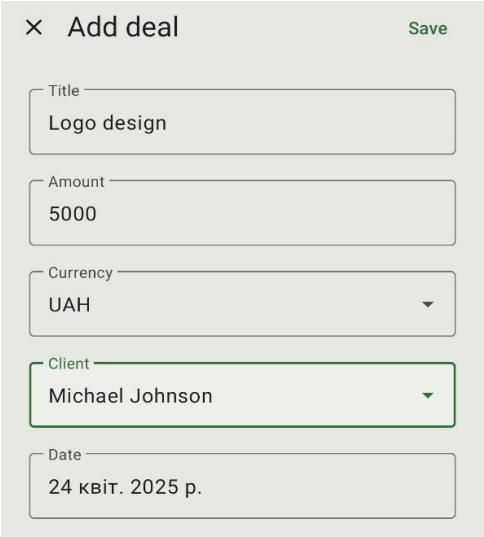

Рисунок 4.11 — Додавання нової угоди

Екран Dashboard містить більш детальну інформацію про прибутки бізнесу, а саме середній дохід за місяць, кількість укладених угод, основна валюта, у якій укладаються угоди, та найприбутковіший місяць. Крім цього містяться графіки для візуалізації даних, а саме стовпчата діаграма з прибутками по місяцях, та кругова діаграма з розподілом валют, у яких було укладено угоди. Для підрахунку загального прибутку, усі валюти переводяться за курсом, отриманим зі спеціального API інтерфейсу (рисунок 4.12, 4.13).

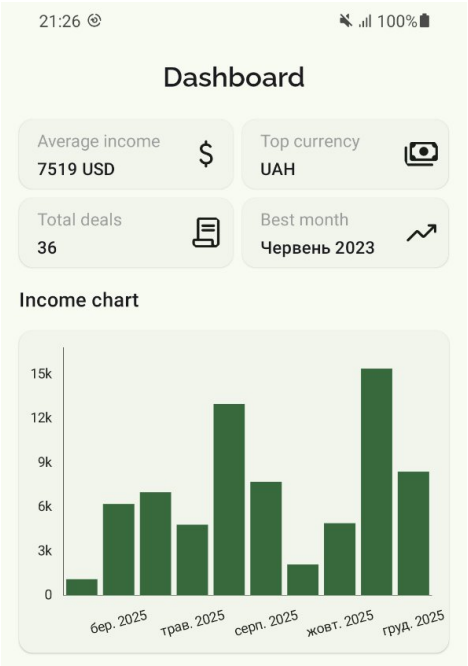

Рисунок 4.12 — Екран Dashboard

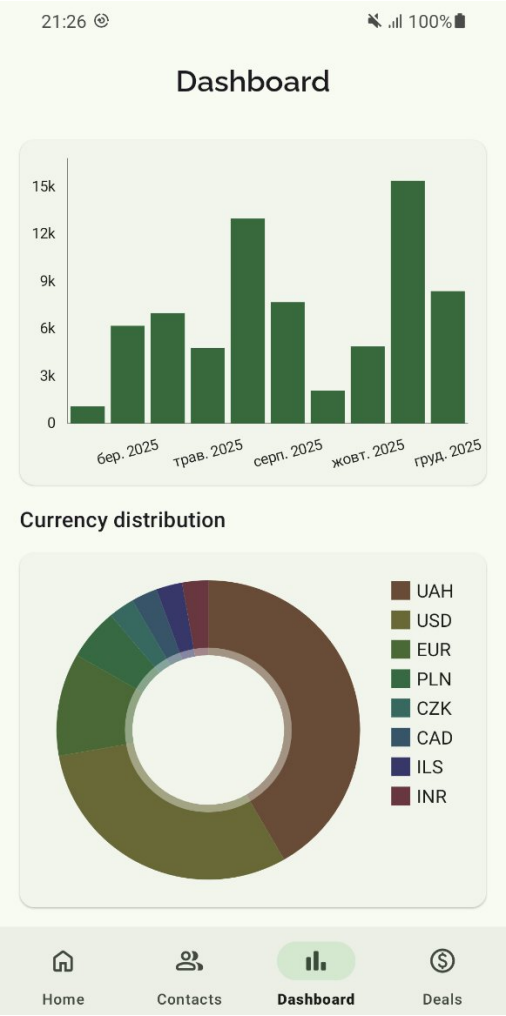

Рисунок 4.13 – Екран Dashboard (продовження)

Таким чином, було проведено тестування CRM-системи для малого бізнесу. Було продемонстровано її функціонал. Розроблена система виконує поставлені на початку розробки завдання.

4.3 Висновки

Для тестування розробленої системи було здійснено перевірку роботи інтерфейсу, бази даних, алгоритмів формування пріоритетів і роботи з курсами валют. За результатами тестування розроблена система виконує всі поставлені задачі.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено засоби CRM-системи для малого бізнесу. Для розробки використовувалося середовище розробки Android Studio. Розроблене ПЗ має на меті удосконалення бізнес-процесів малого бізнесу шляхом розробки спеціалізованої CRM-системи, що дозволяє вести облік угод, зустрічей з клієнтами та справ.

Було здійснено формулювання цілей і постановку задач дослідження. Для цього проаналізовано стан розвитку CRM-систем для малого бізнесу, проведено порівняння з аналогами, розглянуто сильні та слабкі сторони актуальних рішень. На основі цього сформовано перелік задач, які відповідають реальним потребам малого бізнесу.

Для вирішення задачі розробки структури CRM-системи було реалізовано архітектуру застосунку за шаблоном MVVM з чітким поділом на модулі, що відповідають за дані, логіку та інтерфейс. Модель системи розроблено на основі кількох ключових сутностей з урахуванням їх зв’язків. Розроблені алгоритми роботи системи включають метод формування пріоритетів завдань на основі ключових слів і статистики виконання, а також метод офлайн-менеджера валют, який гарантує стійкий доступ до курсів навіть за відсутності мережі. Це забезпечило логічну і функціональну базу для реалізації системи.

Задачу розробки інтерфейсу користувача було вирішено з урахуванням потреб малого бізнесу, що дозволяє користувачеві отримати доступ до списку завдань, дзвінків, подій та угод у зручному форматі. Було розроблено функціонал системи: керування записами, адаптивне визначення пріоритетів завдань, стійкий до помилок обробник курсів валют. Усі компоненти працюють узгоджено, забезпечуючи користувачеві повноцінну CRM-систему.

Для тестування розробленої системи було здійснено перевірку роботи інтерфейсу, бази даних, алгоритмів формування пріоритетів і роботи з курсами валют. За результатами тестування розроблена система виконує всі поставлені задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CRM Software Benefits for Small Businesses. Business News Daily. URL: https://www.businessnewsdaily.com/15963-benefits-of-crm.html (дата звернення: 13.03.2025).

2. Amancio C. M., Caballero B. 28 CRM Statistics Every Small Business Should Know for 2024. Fit Small Business. URL: https://fitsmallbusiness.com/crm-statistics (дата звернення: 13.03.2025).

3. 43 Need to Know CRM Statistics for 2024 | LinkPoint360. LinkPoint360. URL: https://www.linkpoint360.com/crm-statistics/ (дата звернення: 13.03.2025).

4. Lawler J. P. Case Study of Customer Relationship Management (CRM) in the On-Line Affluent Financial Services Market. Journal of Internet Commerce. 2005. Т. 4, № 4. С. 153–164. URL: https://doi.org/10.1300/j179v04n04_10 (дата звернення: 15.03.2025).

5. Lemhandez S. 20 CRM statistics for 2025. URL: https://www.folk.app/articles/20-crm-statistics-for-2024 (дата звернення: 16.03.2025).

6. Богач І.І., Ліщинська Л.Б. Роль CRM-систем у цифровій трансформації малого бізнесу : збірник матеріалів XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса : Видавництво ОНТУ, 2025. с. 94-96.

7. The 11 Greatest Benefits of CRM Platforms. Salesforce. URL: https://www.salesforce.com/crm/benefits-of-crm/ (дата звернення: 17.03.2025).

8. Team K. 22 Eye-Opening CRM Statistics You Should Know for 2024. Kixie PowerCall – Better Sales Made Simple. URL: https://www.kixie.com/sales-blog/22-eye-opening-crm-stats-you-should-know-for-2024/ (дата звернення: 19.03.2025).

9. Customer Relationship Management (CRM) Market Size. URL: https://www.fortunebusinessinsights.com/customer-relationship-management-crm-market-103418 (дата звернення: 21.03.2025).

10. Pipedrive. URL: https://www.pipedrive.com/uk (дата звернення: 25.03.2025).

11. ZOHO. URL: https://www.zoho.com/crm/ (дата звернення: 26.03.2025).

12. Freshworks. URL: https://www.freshworks.com/crm/sales/ (дата звернення: 26.03.2025).

13. Dynamics 365. URL: https://www.microsoft.com/en-us/dynamics-365 (дата звернення: 27.03.2025).

14. Монолітна архітектура ПЗ. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura (дата звернення: 14.05.2025).

15. Мікросервісна архітектура. Основні концепції та виклики. URL: https://foxminded.ua/mikroservisna-arkhitektura (дата звернення: 14.05.2025).

16. Faisal Islam. Kotlin from Scratch: A Project-Based Introduction for the Intrepid Programmer. Сан-Франциско: No Starch Press, 2025. 432с.

17. Mary Delamater. Murach's Modern JavaScript: Beginner to Pro. Фресно: Mike Murach and Associates Inc, 2024. 602с.

18. L. Ramalho. Fluent Python. Clear, Concise, and Effective Programming. Farnham: O'Reilly Media, 2024. 1012с.

19. Mockk. URL: https://mockk.io/ (дата звернення: 09.04.2025).

20. Shekhar Gulati. Java Unit Testing with JUnit 5: Test Driven Development with JUnit 5. New York: Apress, 2017. 164с.

21. Denys Zelenchuk. Android Espresso Revealed: Writing Automated UI Tests. New York: Apress, 2019. 325с.

22. Roboelectric. URL: https://robolectric.org/ (дата звернення: 10.04.2025)

23. Gerardus Blokdyk. UI Automation A Complete Guide - 2020 Edition. London: 5STARCooks, 2021. 306с.

ДОДАТКИ

ДОДАТОК А

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.т.н., проф. Л.Б. Ліщинська
«24» березня 2025 р.
Виконав:
студент гр. 4ПІ-21б І.І. Богач
«24» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу».

Галузь застосування – мобільні технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є удосконалення бізнес-процесів малого бізнесу шляхом розробки спеціалізованої CRM-системи, що дозволяє вести облік угод, зустрічей з клієнтами та справ.

Призначення роботи – розробка та реалізація програмного забезпечення системи управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Amancio C. M., Caballero B. 28 CRM Statistics Every Small Business Should Know for 2024. Fit Small Business. URL: https://fitsmallbusiness.com/crm-statistics (дата звернення: 13.03.2025).

2. Lawler J. P. Case Study of Customer Relationship Management (CRM) in the On-Line Affluent Financial Services Market. Journal of Internet Commerce. 2005. Т. 4, № 4. С. 153–164. URL: https://doi.org/10.1300/j179v04n04_10 (дата звернення: 15.03.2025).

3. Faisal Islam. Kotlin from Scratch: A Project-Based Introduction for the Intrepid Programmer. Сан-Франциско: No Starch Press, 2025. 432с.

5. Технічні вимоги

Тип програми – мобільний застосунок; модель розробки – ітеративна; засоби організації даних програми – фреймворки Room та DataStore; вхідні дані: дані клієнтів, інформація про завдання, події, дзвінки, угоди; вихідні дані: відображення списків завдань, подій, дзвінків, угод, формування статистики на основі цих даних; середовище розробки – Android Studio; мова програмування – Kotlin, програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку CRM-систем для малого бізнесу	25.03.2025 — 03.04.2025
2	Розробка структури та моделі системи	04.04.2025 — 11.04.2025
3	Розробка методу офлайн-менеджера валют	12.04.2025 — 16.04.2025
4	Розробка алгоритмів роботи застосунку	17.04.2025 — 27.04.2025
5	Розробка системи	28.04.2025 — 18.05.2025
6	Тестування CRM-системи для малого бізнесу	19.05.2025 — 26.05.2025
7	Оформлення матеріалів до захисту БКР	27.05.2025 — 30.05.2025

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

ДОДАТОК Б

Протокол перевірки кваліфікаційної роботи

на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу»
Тип роботи: бакалаврська кваліфікаційна робота
Підрозділ: кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism __3,9____%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ліщинська Л.Б. д.т.н., проф.каф.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Богач І.І.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг програми

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        enableEdgeToEdge()

        ViewCompat.setOnApplyWindowInsetsListener(binding.root) { view, windowInsets ->
            val insets = windowInsets.getInsets(WindowInsetsCompat.Type.systemBars())

            binding.toolbar.updateLayoutParams<MarginLayoutParams> {
                topMargin = insets.top
            }
            binding.activityMainBottomNavBar.updatePadding(bottom = insets.bottom)

            view.updateLayoutParams<MarginLayoutParams> {
                leftMargin = insets.left
                rightMargin = insets.right
```

```
            }

            WindowInsetsCompat.CONSUMED
        }

        val navView: BottomNavigationView = binding.activityMainBottomNavBar

        val navController = findNavController(R.id.activity_main_nav_host_fragment).apply {
            val navGraph = navInflater.inflate(R.navigation.mobile_navigation)
            graph = navGraph

            addOnDestinationChangedListener { _, destination, _ ->
                binding.homeAppTitleLayout.isVisible = destination.id == R.id.navigation_home
            }
        }

        val appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.navigation_home, R.id.navigation_contacts, R.id.navigation_dashboard, R.id.navigation_deals
            )
        )

        binding.toolbar.setupWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }
}
```

```
@Module
@InstallIn(SingletonComponent::class)
object DatabaseModule {

    @Provides
    @Singleton
    fun provideDatabase(@ApplicationContext context: Context): AppDatabase =
        AppDatabase.getInstance(context)

    @Provides
    fun provideClientDao(database: AppDatabase): ClientDao = database.clientDao()
    @Provides
    fun provideClientRepository(dao: ClientDao): ClientRepository = ClientRepository(dao)

    @Provides
    fun provideEventDao(database: AppDatabase): EventDao = database.eventDao()
    @Provides
    fun provideEventRepository(dao: EventDao): EventRepository = EventRepository(dao)

    @Provides
    fun provideTaskDao(database: AppDatabase): TaskDao = database.taskDao()
    @Provides
    fun provideTaskRepository(dao: TaskDao): TaskRepository = TaskRepository(dao)

    @Provides
    fun provideCallDao(database: AppDatabase): CallDao = database.callDao()
```

```
    @Provides
    fun provideCallRepository(dao: CallDao): CallRepository = CallRepository(dao)

    @Provides
    fun provideDealDao(database: AppDatabase): DealDao = database.dealDao()
    @Provides
    fun provideDealRepository(dao: DealDao): DealRepository = DealRepository(dao)
}

object CurrencyManager {
    private const val TAG = "CurrencyManager"
    private const val BASE_URL = "https://v6.exchangerate-api.com/v6/"
    private const val API_KEY = "2d830d478520c737f2dc2cec"
    private const val PREF_EXCHANGE_RATES = "exchange_rates"
    private const val PREF_LAST_UPDATE = "last_update"

    var currencyRates: Map<String, Double>? = null

    fun init(context: Context) {
        CoroutineScope(Dispatchers.IO).launch {
            currencyRates = getRates(context)
        }
    }

    private suspend fun getRates(context: Context): Map<String, Double>? {
        val prefs = context.dataStore.data.first()
        val jsonString = prefs[stringPreferencesKey(PREF_EXCHANGE_RATES)]
```

```
    val lastUpdate = prefs[longPreferencesKey(PREF_LAST_UPDATE)] ?: 0L
    val isOutdated = System.currentTimeMillis() - lastUpdate > 24 * 60 * 60 * 1000

    return if (jsonString == null || isOutdated) {
        fetchRatesFromApi(context)
    } else {
        try {
            val response = Gson().fromJson(jsonString, ExchangeRateResponse::class.java)
            response.conversionRates
        } catch (e: Exception) {
            Log.e(TAG, "Failed to parse cached rates: ${e.message}")
            fetchRatesFromApi(context)
        }
    }
}

private suspend fun fetchRatesFromApi(context: Context): Map<String, Double>? {
    return try {
        val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val service = retrofit.create(ExchangeRateApiService::class.java)
        val response = service.getRates(API_KEY)

        if (response.result == "success") {
```

```
            val jsonString = Gson().toJson(response)

            context.dataStore.edit { prefs ->
                prefs[stringPreferencesKey(PREF_EXCHANGE_RATES)] = jsonString
                prefs[longPreferencesKey(PREF_LAST_UPDATE)] = System.currentTimeMillis()
            }

            Log.d(TAG, "Currency rates loaded successfully.")
            response.conversionRates
        } else {
            Log.e(TAG, "Currency API returned failure result.")
            null
        }
    } catch (e: HttpException) {
        Log.e(TAG, "HTTP error: ${e.code()} ${e.message()}")
        null
    } catch (e: Exception) {
        Log.e(TAG, "Failed to fetch currency rates: ${e.message}")
        null
    }
  }
}

@Database(entities = [
  ClientEntity::class,
  TaskEntity::class,
  EventEntity::class,
```

```
    CallEntity::class,
    DealEntity::class],
    version = 2, exportSchema = true,
    autoMigrations = [
        AutoMigration(from = 1, to = 2),
    ])
@TypeConverters(MyTypeConverters::class)
abstract class AppDatabase : RoomDatabase() {
    abstract fun clientDao(): ClientDao
    abstract fun eventDao(): EventDao
    abstract fun taskDao(): TaskDao
    abstract fun callDao(): CallDao
    abstract fun dealDao(): DealDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "app_database"
                ).build()
                INSTANCE = instance
                instance
```

```
            }
        }
    }
}

private class MyTypeConverters {
    @TypeConverter
    fun localDateFromTimestamp(value: Long?): LocalDate? {
        return value?.let { LocalDate.ofEpochDay(value) }
    }

    @TypeConverter
    fun localDateToTimestamp(date: LocalDate?): Long? {
        return date?.toEpochDay()
    }

    @TypeConverter
    fun localTimeFromTimestamp(value: Int?): LocalTime? {
        return value?.let { LocalTime.ofSecondOfDay(value.toLong()) }
    }

    @TypeConverter
    fun localTimeToTimestamp(time: LocalTime?): Int? {
        return time?.toSecondOfDay()
    }
}
```

```
{
 "formatVersion": 1,
 "database": {
  "version": 2,
  "identityHash": "081c60c096d5be7ae6f9794aa5e974e0",
  "entities": [
   {
    "tableName": "clients",
    "createSql": "CREATE TABLE IF NOT EXISTS `${TABLE_NAME}` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `name` TEXT NOT NULL, `email` TEXT NOT NULL, `phone` TEXT NOT NULL)",
    "fields": [
     {
      "fieldPath": "id",
      "columnName": "id",
      "affinity": "INTEGER",
      "notNull": true
     },
     {
      "fieldPath": "name",
      "columnName": "name",
      "affinity": "TEXT",
      "notNull": true
     },
     {
      "fieldPath": "email",
      "columnName": "email",
      "affinity": "TEXT",
```

```
        "notNull": true
      },
      {
        "fieldPath": "phone",
        "columnName": "phone",
        "affinity": "TEXT",
        "notNull": true
      }
    ],
    "primaryKey": {
      "autoGenerate": true,
      "columnNames": [
        "id"
      ]
    },
    "indices": [],
    "foreignKeys": []
  },
  {
    "tableName": "tasks",
    "createSql": "CREATE TABLE IF NOT EXISTS `${TABLE_NAME}` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `title` TEXT NOT NULL, `date` INTEGER NOT NULL, `priority` INTEGER NOT NULL, `clientId` INTEGER NOT NULL, `completed` INTEGER NOT NULL, FOREIGN KEY(`clientId`) REFERENCES `clients`(`id`) ON UPDATE NO ACTION ON DELETE CASCADE )",
    "fields": [
      {
        "fieldPath": "id",
```

```
  "columnName": "id",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "title",
  "columnName": "title",
  "affinity": "TEXT",
  "notNull": true
},
{
  "fieldPath": "date",
  "columnName": "date",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "priority",
  "columnName": "priority",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "clientId",
  "columnName": "clientId",
  "affinity": "INTEGER",
  "notNull": true
```

```
      },
      {
        "fieldPath": "completed",
        "columnName": "completed",
        "affinity": "INTEGER",
        "notNull": true
      }
    ],
    "primaryKey": {
      "autoGenerate": true,
      "columnNames": [
        "id"
      ]
    },
    "indices": [
      {
        "name": "index_tasks_clientId",
        "unique": false,
        "columnNames": [
          "clientId"
        ],
        "orders": [],
        "createSql": "CREATE INDEX IF NOT EXISTS `index_tasks_clientId` ON `${TABLE_NAME}` (`clientId`)"
      }
    ],
    "foreignKeys": [
```

```
        {
          "table": "clients",
          "onDelete": "CASCADE",
          "onUpdate": "NO ACTION",
          "columns": [
            "clientId"
          ],
          "referencedColumns": [
            "id"
          ]
        }
      ]
    },
    {
      "tableName": "events",
      "createSql": "CREATE TABLE IF NOT EXISTS `${TABLE_NAME}` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `title` TEXT NOT NULL, `timeStart` INTEGER NOT NULL, `timeEnd` INTEGER NOT NULL, `date` INTEGER NOT NULL, `completed` INTEGER NOT NULL)",
      "fields": [
        {
          "fieldPath": "id",
          "columnName": "id",
          "affinity": "INTEGER",
          "notNull": true
        },
        {
          "fieldPath": "title",
```

```
  "columnName": "title",
  "affinity": "TEXT",
  "notNull": true
},
{
  "fieldPath": "timeStart",
  "columnName": "timeStart",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "timeEnd",
  "columnName": "timeEnd",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "date",
  "columnName": "date",
  "affinity": "INTEGER",
  "notNull": true
},
{
  "fieldPath": "completed",
  "columnName": "completed",
  "affinity": "INTEGER",
  "notNull": true
```

```
      }
    ],
    "primaryKey": {
      "autoGenerate": true,
      "columnNames": [
        "id"
      ]
    },
    "indices": [],
    "foreignKeys": []
  },
  {
    "tableName": "calls",
    "createSql": "CREATE TABLE IF NOT EXISTS `${TABLE_NAME}` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `title` TEXT NOT NULL, `time` INTEGER NOT NULL, `date` INTEGER NOT NULL, `clientId` INTEGER NOT NULL, `completed` INTEGER NOT NULL, FOREIGN KEY(`clientId`) REFERENCES `clients`(`id`) ON UPDATE NO ACTION ON DELETE CASCADE )",
    "fields": [
      {
        "fieldPath": "id",
        "columnName": "id",
        "affinity": "INTEGER",
        "notNull": true
      },
      {
        "fieldPath": "title",
        "columnName": "title",
```

```
  "affinity": "TEXT",
  "notNull": true
 },
 {
  "fieldPath": "time",
  "columnName": "time",
  "affinity": "INTEGER",
  "notNull": true
 },
 {
  "fieldPath": "date",
  "columnName": "date",
  "affinity": "INTEGER",
  "notNull": true
 },
 {
  "fieldPath": "clientId",
  "columnName": "clientId",
  "affinity": "INTEGER",
  "notNull": true
 },
 {
  "fieldPath": "completed",
  "columnName": "completed",
  "affinity": "INTEGER",
  "notNull": true
 }
```

```
],
"primaryKey": {
 "autoGenerate": true,
 "columnNames": [
  "id"
 ]
},
"indices": [
 {
  "name": "index_calls_clientId",
  "unique": false,
  "columnNames": [
   "clientId"
  ],
  "orders": [],
  "createSql": "CREATE INDEX IF NOT EXISTS `index_calls_clientId` ON `${TABLE_NAME}` (`clientId`)"
 }
],
"foreignKeys": [
 {
  "table": "clients",
  "onDelete": "CASCADE",
  "onUpdate": "NO ACTION",
  "columns": [
   "clientId"
  ],
```

```
        "referencedColumns": [
          "id"
        ]
      }
    ]
  },
  {
    "tableName": "deals",
    "createSql": "CREATE TABLE IF NOT EXISTS `${TABLE_NAME}` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `title` TEXT NOT NULL, `amount` INTEGER NOT NULL, `currency` TEXT NOT NULL, `clientId` INTEGER NOT NULL, `date` INTEGER NOT NULL DEFAULT 0, FOREIGN KEY(`clientId`) REFERENCES `clients`(`id`) ON UPDATE NO ACTION ON DELETE CASCADE )",
    "fields": [
      {
        "fieldPath": "id",
        "columnName": "id",
        "affinity": "INTEGER",
        "notNull": true
      },
      {
        "fieldPath": "title",
        "columnName": "title",
        "affinity": "TEXT",
        "notNull": true
      },
      {
        "fieldPath": "amount",
```

```
      "columnName": "amount",
      "affinity": "INTEGER",
      "notNull": true
    },
    {
      "fieldPath": "currency",
      "columnName": "currency",
      "affinity": "TEXT",
      "notNull": true
    },
    {
      "fieldPath": "clientId",
      "columnName": "clientId",
      "affinity": "INTEGER",
      "notNull": true
    },
    {
      "fieldPath": "date",
      "columnName": "date",
      "affinity": "INTEGER",
      "notNull": true,
      "defaultValue": "0"
    }
  ],
  "primaryKey": {
    "autoGenerate": true,
    "columnNames": [
```

```
      "id"
    ]
  },
  "indices": [
    {
      "name": "index_deals_clientId",
      "unique": false,
      "columnNames": [
        "clientId"
      ],
      "orders": [],
      "createSql": "CREATE INDEX IF NOT EXISTS `index_deals_clientId` ON `${TABLE_NAME}` (`clientId`)"
    }
  ],
  "foreignKeys": [
    {
      "table": "clients",
      "onDelete": "CASCADE",
      "onUpdate": "NO ACTION",
      "columns": [
        "clientId"
      ],
      "referencedColumns": [
        "id"
      ]
    }
```

```
        ]
      }
    ],
    "views": [],
    "setupQueries": [
      "CREATE TABLE IF NOT EXISTS room_master_table (id INTEGER PRIMARY KEY,identity_hash TEXT)",
      "INSERT OR REPLACE INTO room_master_table (id,identity_hash) VALUES(42, '081c60c096d5be7ae6f9794aa5e974e0')"
    ]
  }
}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

СИСТЕМА УПРАВЛІННЯ ПРОГРАМУВАННЯ ВЗАЄМОВІДНОСИН З КЛІЄНТАМИ У CRM ДЛЯ МАЛОГО БІЗНЕСУ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Система управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу»

Автор: ст.групи 4ПІ-21б Богач І.І.
Науковий керівник: д.т.н., проф. каф. ПЗ Ліщинська Л.Б.

Вінниця - 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

У сучасних умовах конкуренції малий бізнес все частіше звертається до спеціалізованих рішень для автоматизації взаємодії з клієнтами та оптимізації внутрішніх процесів. Впровадження CRM-систем допомагає стандартизувати роботу відділів продажів, маркетингу та підтримки, що призводить до підвищення ефективності і лояльності клієнтів . При цьому вже 51 % малих підприємств, які використали CRM, відзначили зростання конверсії лідів, а 72 % опитаних власників вважають такі рішення ефективними для досягнення бізнес-цілей. Водночас глобальний ринок CRM-систем у 2024 р. оцінювався майже в 101,4 млрд USD з прогнозованим ростом до 112,9 млрд USD у 2025 р. і подальшим зростанням до 262,7 млрд USD до 2032 р. Проте близько 50 % проектів впровадження CRM стикаються з невдачею через відсутність координації між відділами, що вказує на необхідність детального опрацювання вимог і адаптації рішення під специфіку малого бізнесу.

Customer Relationship Management (CRM) охоплює практики, стратегії та технології для аналізу взаємодії підприємства з поточними та потенційними клієнтами. CRM-системи забезпечують централізоване зберігання даних про контрагента, історію комунікацій та аналітику ефективності продажів, що дозволяє підвищити якість обслуговування та швидкість обробки запитів. Для малих підприємств, де ресурси на підтримку клієнтської бази обмежені, впровадження CRM стає ключовим інструментом для цифрової трансформації та зростання конкурентоспроможності.

Актуальність розробки CRM-системи для малого бізнесу полягає в тому, що CRM дозволяє автоматизувати рутинні операції – нагадування, сегментацію контактів, відстеження угод, що скорочує час на адміністративні завдання і дозволяє зосередитися на стратегії продажів. Крім цього, аналітичні можливості системи дають змогу відстежувати ефективність маркетингових кампаній і швидко реагувати на зміни поведінки клієнтів. Також інтеграція інструментів штучного інтелекту (наприклад, чат-ботів) вже планується 80% компаній, що використовують CRM, що підвищує швидкість реагування на запити та покращує клієнтський досвід.

Глобальний ринок CRM продовжує стрімко зростати: у 2024 р. він становив $101,41 млрд, а вже в 2025 р. очікується збільшення до $112,91 млрд. Водночас малий бізнес активно освоює хмарні CRM-рішення: вже 87 % підприємств використовують хмарні платформи, а серед тих, хто використовує мобільні CRM, 65 % досягають своїх продажних цілей. Це свідчить про те, що сфера CRM-додатків знаходиться на етапі зрілості, але продовжує впроваджувати нові функції для задоволення потреб різних сегментів бізнесу.

Отже, актуальною є розробка системи управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

Мета та завдання дослідження. Метою роботи є удосконалення бізнес-процесів малого бізнесу шляхом розробки спеціалізованої CRM-системи, що дозволяє вести облік угод, зустрічей з клієнтами та справ.
Об’єктом дослідження є процеси управління взаємовідносинами з клієнтами у CRM для малого бізнесу.
Предметом дослідження є методи і засоби управління взаємовідносинами з клієнтами у CRM для малого бізнесу.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Завдання дослідження

Відповідно до поставленої мети потрібно вирішити такі завдання:

- розробити структуру CRM-системи для малого бізнесу;
- розробити інтерфейс користувача застосунку;
- розробити модель системи;
- розробити алгоритми роботи системи;
- розробити функціонал системи;
- провести тестування розробленої системи.

Рисунок Г.4 – Завдання дослідження

Наукова новизна

1. Удосконалено метод адаптивного формування пріоритетів завдань на основі базового семантичного та статистичного аналізу ключових слів, що дозволяє автоматично визначати важливість нових завдань.
2. Удосконалено метод офлайн-менеджменту курсів валют, який поєднує локальне кешування із парсингом даних, що дозволяє підвищити стійкість застосунку, забезпечуючи гарантований доступ до даних навіть за відсутності підключення до мережі.

Рисунок Г.5 — Наукова новизна

Практична цінність отриманих результатів

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби управління програмування взаємовідносин з клієнтами у CRM для малого бізнесу.

Рисунок Г.6 — Практична цінність отриманих результатів

Аналоги

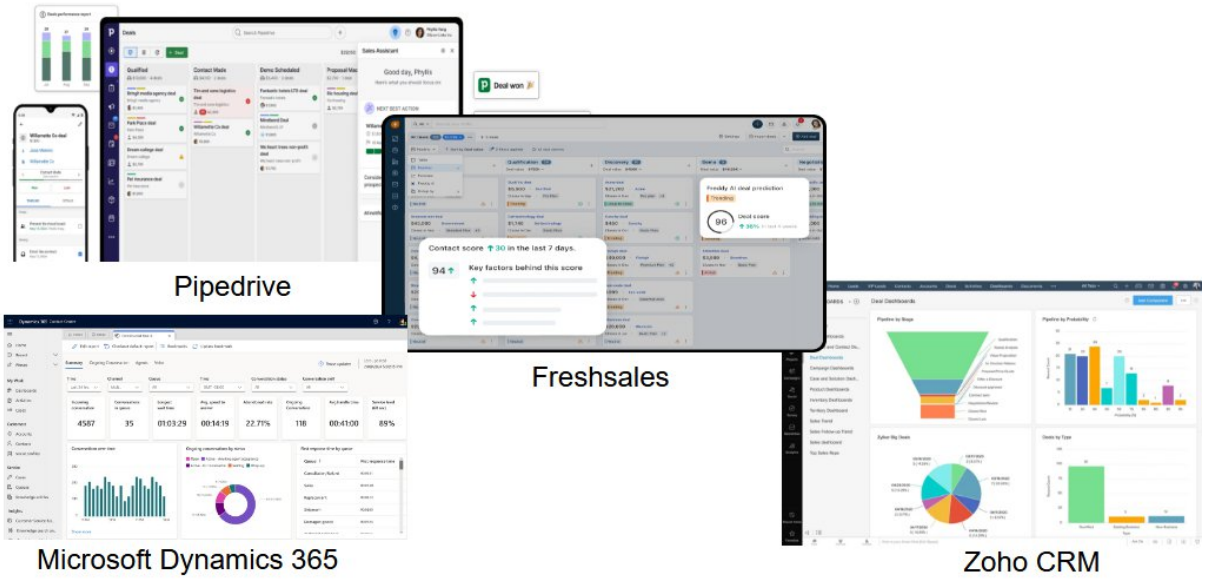

Рисунок Г.7 — Аналоги

Порівняльний аналіз аналогів

Характеристика	Dynamics 365	Freshsales	Zoho CRM	Pipedrive	Власна розробка
Управління контактами	+	+	+	+	+
Управління угодами/продажами	+	+	+	+	+
Розширена аналітика та звітність	+	-	+	-	+
Мобільний додаток	+	+	+	+	+
Можливість налаштування бізнес-процесів	+	+	+	+	+
Функціонал для малого бізнесу	-	+	+	+	+
Автоматичне формування пріоритетів завдань	-	-	-	-	+
Сумарний бал	5	5	6	5	7

Рисунок Г.8 — Порівняльний аналіз аналогів

Використані технології

Рисунок Г.9 — Використані технології

Розробка методу офлайн-менеджера валют

1. Менеджер зчитує з DataStore два значення: JSON-рядок з раніше закешованими курсами та мітку часу останнього оновлення.
2. Обчислюється, чи минуло більше 24 годин від останнього запису, або чи сам JSON відсутній.
3. Якщо кеш існує і він актуальний, відбувається перехід до кроку 4. Інакше виконується асинхронний запит до віддаленого сервісу через Retrofit у корутині, результат перетворюється в ExchangeRateResponse.
4. Менеджер намагається обробити JSON за допомогою Gson. У разі успішної обробки, отримані курси валют повертаються відразу в інтерфейс користувача. У разі помилки відбувається автоматичний повторний запит до API.
5. Якщо викликано API (у кроці 3 або через помилку парсингу), менеджер записує свіжий JSON і нову мітку часу в DataStore.
6. Отриманий результат (карта валют і курсів) повертається для відображення користувачеві.

Рисунок Г.10 — Розробка методу офлайн-менеджера валют

Розробка методу офлайн-менеджера валют

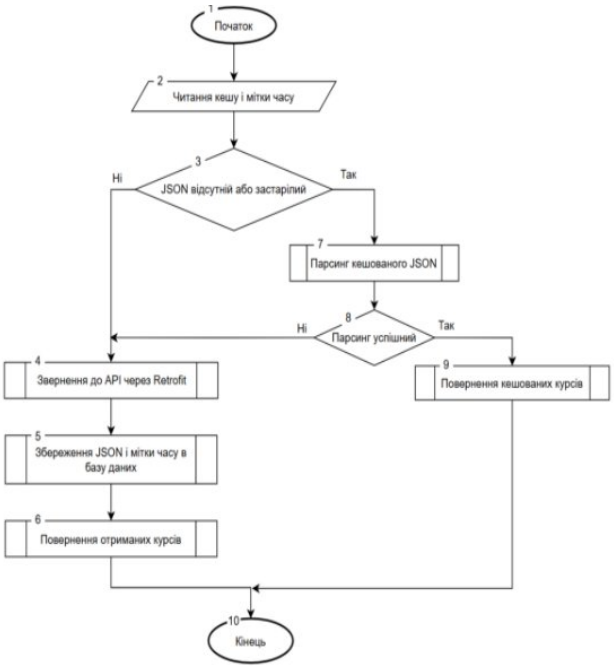

Рисунок Г.11 — Блок-схема методу офлайн-менеджера валют

Розробка методу формування пріоритетів завдань

1) завдання аналізується за допомогою простих лінгвістичних правил (наприклад, розбиття на фрази по пробілах та пунктуації).
2) для знайдених слів виконується запит до бази даних, де зберігається інформація про попередній досвід взаємодії з подібними завданнями.
3) на основі кількості відкладань, виконань та часу останнього використання визначається ваговий коефіцієнт кожного слова.
4) всі коефіцієнти усереднюються, після чого значення порівнюється з встановленими порогами для визначення остаточного пріоритету.

Рисунок Г.12 — Розробка методу формування пріоритетів завдань

Розробка методу формування пріоритетів завдань

Рисунок Г.13 — Розробка методу формування пріоритетів завдань

Розробка моделі системи

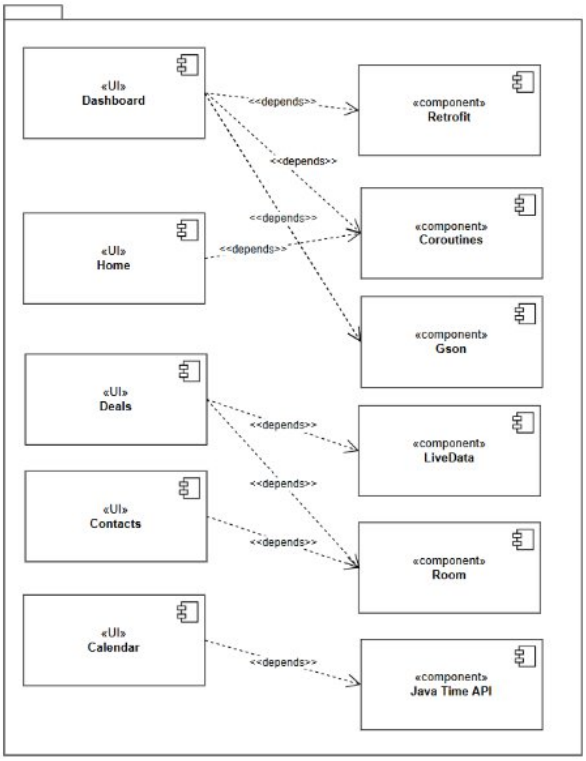

Рисунок Г.14 — Розробка моделі системи

Тестування системи

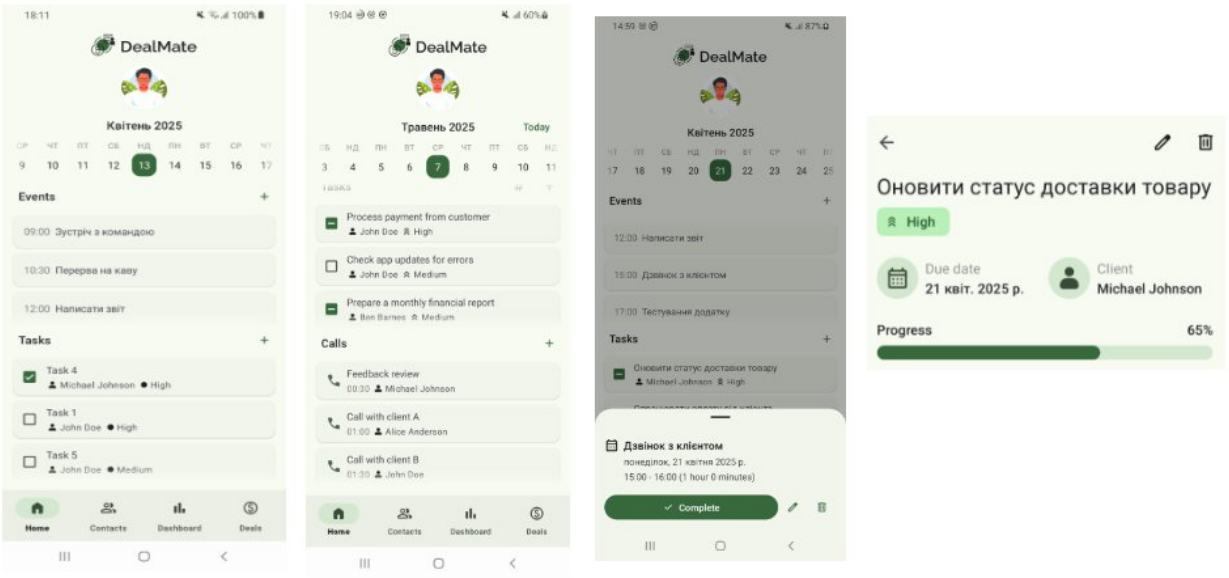

Рисунок Г.15 — Тестування головного екрану системи

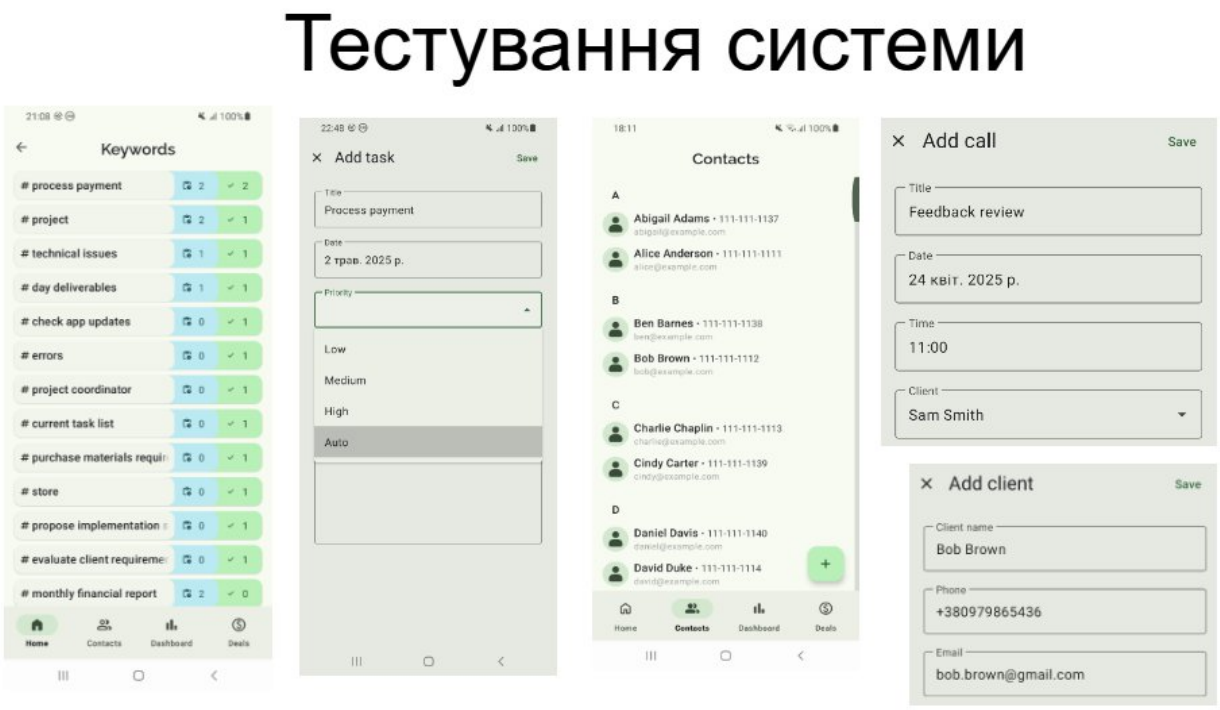

Рисунок Г.16 — Тестування модуля формування пріоритетів завдань

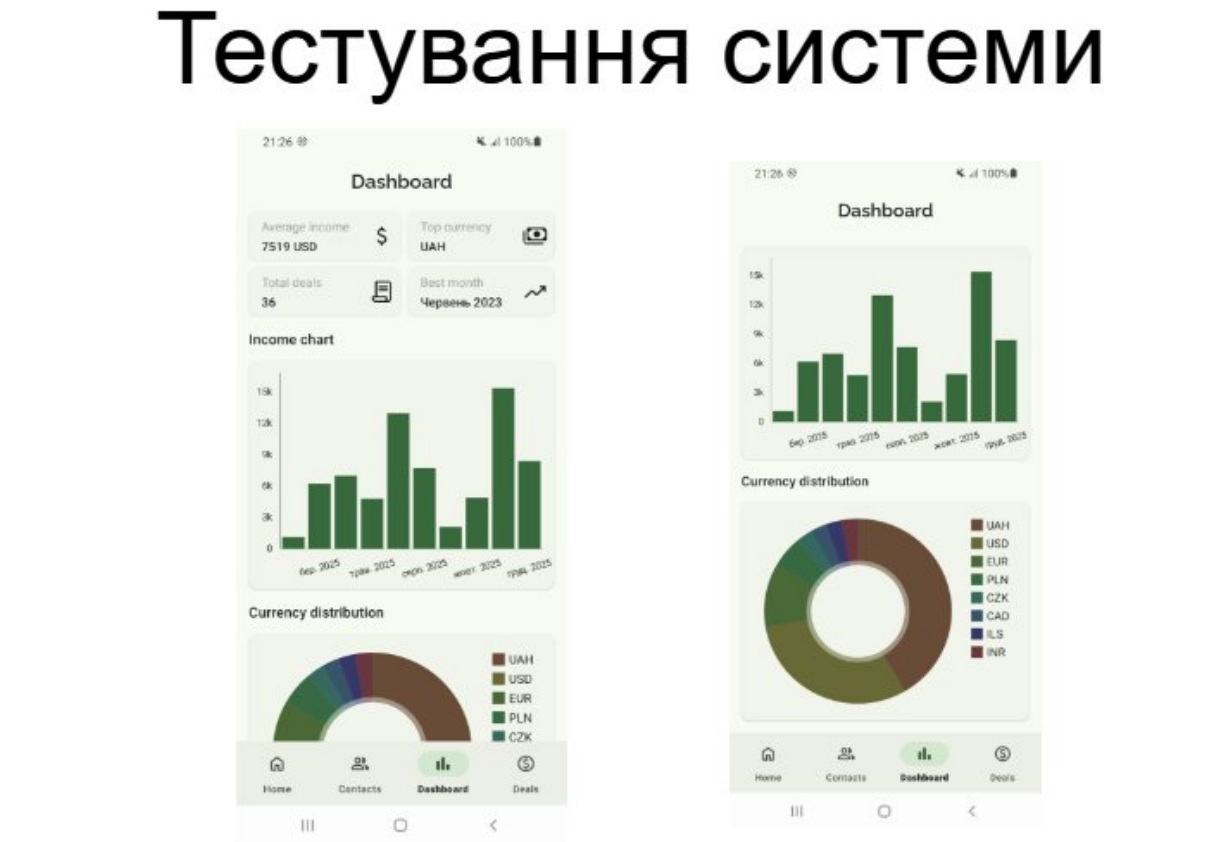

Рисунок Г.17 — Тестування модуля менеджера валют

Висновки

У бакалаврській кваліфікаційній роботі було розроблено засоби CRM-системи управління взаємовідносинами з клієнтами для малого бізнесу. Для розробки використовувалося середовище розробки Android Studio. Розроблене ПЗ має на меті удосконалення бізнес-процесів малого бізнесу шляхом розробки спеціалізованої CRM-системи, що дозволяє вести облік угод, зустрічей з клієнтами та справ.

Було здійснено формулювання цілей і постановку задач дослідження. Для цього проаналізовано стан розвитку CRM-систем для малого бізнесу, проведено порівняння з аналогами, розглянуто сильні та слабкі сторони актуальних рішень. На основі цього сформовано перелік задач, які відповідають реальним потребам малого бізнесу.

Для вирішення задачі розробки структури CRM-системи було реалізовано архітектуру застосунку за шаблоном MVVM з чітким поділом на модулі, що відповідають за дані, логіку та інтерфейс. Модель системи розроблено на основі кількох ключових сутностей з урахуванням їх зв'язків. Розроблені алгоритми роботи системи включають метод формування пріоритетів завдань на основі ключових слів і статистики виконання, а також метод офлайн-менеджера валют, який гарантує стійкий доступ до курсів навіть за відсутності мережі. Це забезпечило логічну і функціональну базу для реалізації системи.

Задачу розробки інтерфейсу користувача було вирішено з урахуванням потреб малого бізнесу, що дозволяє користувачеві отримати доступ до списку завдань, дзвінків, подій та угод у зручному форматі. Було розроблено функціонал системи: керування записами, адаптивне визначення пріоритетів завдань, стійкий до помилок обробник курсів валют. Усі компоненти працюють узгоджено, забезпечуючи користувачеві повноцінну CRM-систему.

Для тестування розробленої системи було здійснено перевірку роботи інтерфейсу, бази даних, алгоритмів формування пріоритетів і роботи з курсами валют. За результатами тестування розроблена система виконує всі поставлені задачі.

Рисунок Г.18 — Висновки

Апробація результатів роботи і публікації

Апробація результатів роботи. Результати роботи доповідалися на: XXV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів «Стан, досягнення і перспективи інформаційних систем і технологій» (Одеса, 2025 р.).

Публікації. Богач І.І., Ліщинська Л.Б. Роль CRM-систем у цифровій трансформації малого бізнесу : збірник матеріалів XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса : Видавництво ОНТУ, 2025. с. 94-96.

Рисунок Г.19 — Апробація результатів роботи та публікації