МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

РАСІК МИКОЛА РОМАНОВИЧ

ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПІДБОРУ РЕЦЕПТІВ І МЕНЮ

Спеціальність: 122 «Комп'ютерні науки»
Освітньо-професійна програма Комп'ютерні науки та інформаційні технології
Робота на здобуття освітнього ступеня «магістр»

Науковий керівник:
ГЛИНЧУК ЛЮДМИЛА ЯРОСЛАВІВНА,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 20___ р.

Завідувач кафедри
(_____) Гришанович Т. О.
(підпис)

Луцьк – 2025

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій супроводжується активним поширенням мобільних застосунків, які допомагають користувачам вирішувати повсякденні завдання. Однією з таких сфер є організація харчування: пошук рецептів, планування раціону, формування меню на день чи тиждень. Багато користувачів витрачають значний час на підбір страв, узгодження рецептів з наявними інгредієнтами та власними вподобаннями.

Актуальність теми зумовлена потребою у зручних програмних засобах, що автоматизують процес підбору рецептів і формування меню, дозволяють систематизувати інформацію про страви та забезпечують швидкий доступ до необхідних даних. Поєднання мобільної платформи, зовнішніх сервісів із рецептами та локального збереження даних створює основу для створення сучасного прикладного програмного продукту у галузі комп’ютерних наук.

Мета роботи. Проєктування та розробка мобільного застосунку для підбору рецептів і формування меню, який забезпечує взаємодію із зовнішнім програмним інтерфейсом (API) та локальною базою даних і надає користувачу зручний інтерфейс для роботи з рецептами.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область та існуючі підходи до автоматизації підбору рецептів і формування меню;
- виконати огляд і порівняльний аналіз програмних засобів, які реалізують подібний функціонал;
- дослідити технології та інструменти розробки мобільних застосунків для Android (Kotlin, Jetpack Compose, робота з базами даних та вебсервісами);
- сформулювати вимоги до програмного забезпечення та виконати постановку задачі;
- розробити архітектуру мобільного застосунку та модель даних;
- реалізувати мобільний застосунок із використанням вибраних технологій та інтеграцією із зовнішнім API для отримання рецептів;

- провести тестування розробленого програмного забезпечення та проаналізувати отримані результати.

Об’єкт дослідження – процеси та підходи до розробки мобільного застосунку для підбору рецептів та формування меню для користувача.

Предмет дослідження – методи та програмні засоби розробки мобільних застосунків для підбору рецептів і меню з використанням зовнішніх вебсервісів та локальних баз даних.

Наукова новизна роботи полягає в поєднанні використання зовнішнього сервісу рецептів із локальною базою даних мобільного застосунку для реалізації пошуку страв за заданими параметрами та формування меню. Запропоновано структуру та архітектуру програмного забезпечення, орієнтовану на розширення функціоналу та подальшу інтеграцію додаткових сервісів.

Практичне значення роботи полягає в тому, що створений мобільний застосунок дозволяє користувачам ефективно знаходити рецепти, що відповідають їхнім вподобанням, спрощує процес планування харчування та може слугувати основою для подальшого розвитку систем із розширеним функціоналом, зокрема для автоматичного планування меню чи оцінювання характеристик страв.

Апробація. Результати цієї роботи представлено на науковій конференції-форумі «United Perspectives 24/7», що відбулася 10-11 грудня 2025 року в онлайн-форматі, яку було організовано Національною академією наук і мистецтв України. Тези доповіді «Архітектурні особливості та реалізація мобільного застосунку для підбору рецептів на платформі Android» опубліковано у науковому альманасі «Crosspoint» [1].

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

1.1 Аналіз предметної області та роль інформаційних технологій у сфері організації харчування

Сучасний етап розвитку суспільства характеризується глобальною цифровізацією всіх сфер людської діяльності, і побутова сфера не є винятком. Організація харчування, яка традиційно вважалася рутинним процесом, сьогодні трансформується під впливом інформаційних технологій. Мобільні пристрої стають невід'ємними помічниками на кухні, замінюючи паперові кулінарні книги, записники та хаотичні нотатки.

Предметна область даного дослідження охоплює сферу FoodTech (технології у сфері харчування) – екосистему, що поєднує агропродовольчий сектор з IT-рішеннями. Якщо на початкових етапах розвитку FoodTech асоціювався переважно з сервісами доставки їжі (e-grocery та delivery), то нині акцент зміщується в бік персоналізованих інструментів для приготування їжі вдома. Це зумовлено зростанням інтересу користувачів до здорового способу життя, раціонального споживання та економії часу [2; 3].

Роль інформаційних технологій у цій сфері полягає у вирішенні кількох ключових проблем, з якими стикається сучасна людина:

- Проблема надлишкового вибору (Decision Fatigue). Інтернет надає доступ до мільйонів рецептів, проте така кількість інформації часто не спрощує, а ускладнює вибір. Користувач витрачає значний час на перегляд контенту, намагаючись знайти страву, що відповідає його поточним бажанням та можливостям. Інформаційні системи дозволяють структурувати ці дані, застосовуючи алгоритми фільтрації та ранжування для надання релевантних результатів.
- Оптимізація використання ресурсів. Однією з глобальних проблем є нераціональне використання продуктів харчування (food waste) [4]. Часто користувачі не знають, що приготувати з наявних залишків продуктів, що

призводить до їх псування. Мобільні застосунки, оснащені алгоритмами пошуку за інгредієнтами, виступають інструментом менеджменту домашніх запасів, допомагаючи зменшити кількість відходів та зекономити кошти.

- Персоналізація харчування. Традиційні джерела рецептів зазвичай не враховують індивідуальні особливості споживача, зокрема наявність алергій, дієтичні обмеження (веганство, безглютенова дієта тощо) та потрібну калорійність страв. Сучасні програмні засоби дозволяють створювати динамічні профілі користувачів, адаптуючи видачу контенту під конкретні медичні або етичні вимоги.

З точки зору комп'ютерних наук, сучасний застосунок для підбору рецептів – це не просто база даних, а складна інформаційна система. Вона включає в себе:

- бази даних (DBMS) для зберігання структурованої інформації про рецепти, інгредієнти, нутрієнти та зв'язки між ними;
- пошукові алгоритми для швидкого знаходження інформації за ключовими словами, тегами або набором параметрів;
- інтерфейси взаємодії (UI/UX) для забезпечення зручного доступу до функціоналу на мобільних пристроях з урахуванням специфіки використання (наприклад, голосове керування або режим «вільні руки»).

Окремо варто виділити перехід від вебплатформ до мобільних застосунків. Смартфон є більш зручним інструментом у процесі приготування їжі завдяки мобільності, наявності камери (для сканування штрих-кодів продуктів) та можливості отримання push-повідомлень (нагадування про прийом їжі або необхідність докупити продукти).

Таким чином, інформаційні технології стають одним із ключових інструментів оптимізації процесів харчування. Розробка спеціалізованого програмного забезпечення, яке поєднує в собі функціонал кулінарної книги, планувальника меню та менеджера продуктів, є відповіддю на актуальні запити

користувачів. Це дозволяє перетворити процес приготування їжі з рутинного обов’язку на впорядкований та творчий процес.

1.2 Концепції, підходи та методи побудови рекомендаційних систем

Рекомендаційні системи є класом інформаційних систем, основним призначенням яких є підтримка користувача у виборі об’єктів із великої множини можливих варіантів. Такими об’єктами можуть бути фільми, музичні треки, товари в інтернет-магазинах, новини або, як у межах даної роботи, рецепти страв. На відміну від звичайного пошуку, рекомендаційна система не лише реагує на явний запит, а й намагається передбачити, які саме елементи будуть найбільш корисними чи цікавими конкретному користувачу [5].

У загальному вигляді рекомендаційна система оперує трьома основними компонентами – множиною користувачів, множиною об’єктів (рецептів) та інформацією про взаємодію між ними. Така взаємодія може бути представлена у вигляді явних оцінок (рейтинги, «подобається/не подобається»), неявних ознак (частота перегляду, додавання в обране, час взаємодії з елементом) або контекстних характеристик (час доби, пристрій, місце знаходження користувача тощо). Завдання рекомендаційної системи полягає в тому, щоб на основі цих даних побудувати модель переваг користувача і використати її для формування персоналізованих списків рекомендацій [6].

Однією з базових концепцій побудови рекомендаційних систем є контентно-орієнтований підхід. У цьому випадку кожен об’єкт описується набором ознак (наприклад, тип страви, набір інгредієнтів, час приготування, складність), а профіль користувача формується на основі тих об’єктів, які йому сподобалися раніше. Система намагається знайти нові об’єкти, максимально подібні до тих, з якими користувач уже взаємодіяв. Перевага підходу полягає в прозорості та можливості пояснення рекомендацій. Легко показати, що саме рецепт запропоновано через схожість інгредієнтів або типу страви. Водночас якість рекомендацій істотно залежить від повноти та якості опису об’єктів.

Іншим широко використовуваним підходом є фільтрація на основі спільних вподобань користувачів (collaborative filtering), яка ґрунтується не стільки на властивостях об’єктів, скільки на подібності поведінки користувачів [7]. Ідея полягає в тому, що якщо двоє користувачів мали схожі оцінки чи вибір у минулому, то одному з них можна рекомендувати ті об’єкти, які сподобались іншому. На практиці застосовуються два основні варіанти такого підходу: методи, орієнтовані на користувачів (user-based), де шукають «сусідів» серед користувачів, та методи, орієнтовані на об’єкти (item-based), де аналізують схожість між рецептами на основі їхніх спільних оцінок. Перевага методів на основі спільних вподобань полягає в тому, що вони здатні враховувати приховані закономірності, які важко формалізувати вручну, але водночас вони чутливі до проблеми «холодного старту», коли для нових користувачів або нових об’єктів бракує даних [8].

Подальшим розвитком такого підходу є моделі на основі факторизації матриць та інших методів машинного навчання, де взаємодія «користувач-об’єкт» подається як розріджена матриця, яка наближено розкладається на добуток матриць нижчої розмірності. Це дозволяє виділити латентні (приховані) фактори, що визначають уподобання користувачів і властивості об’єктів, і будувати більш точні прогнози оцінок. Такі підходи добре працюють для великих масивів даних, але потребують більш складної реалізації та налаштування, а також обчислювальних ресурсів.

Окремо виділяють гібридні рекомендаційні системи, які поєднують декілька підходів, зокрема контентно-орієнтований, підхід на основі спільних вподобань користувачів, урахування популярності об’єктів та фільтрацію за бізнес-правилами тощо. Метою гібридизації є компенсація недоліків окремих методів і підвищення якості рекомендацій, наприклад, шляхом поєднання персоналізованих рекомендацій із простим виведенням найпопулярніших рецептів. Це особливо актуально для систем, де частина користувачів має невелику історію взаємодії, а інша – значний накопичений досвід [9].

У сфері організації харчування важливу роль відіграє також концепція контекстно-залежних рекомендацій, коли при формуванні списку страв враховується не лише профіль користувача, але й поточний контекст, зокрема час доби (сніданок, обід, вечеря), тривалість, яку користувач готовий витратити на приготування, набір інгредієнтів, що є в наявності, або навіть пору року. Такий підхід дозволяє наблизити роботу системи до реальних потреб користувача, пропонуючи не просто «цікаві» рецепти, а ті, які є доречними в конкретний момент.

Методи побудови рекомендаційних систем тісно пов’язані з типами даних, які доступні розробнику. У випадку мобільного застосунку для підбору рецептів джерелом даних можуть виступати зовнішні API, локальна база даних із збереженими рецептами, дані про перегляди та вибір користувача. На практиці це часто приводить до використання спрощених, але надійних підходів, зокрема фільтрації та сортування рецептів за параметрами (тип страви, інгредієнти, час приготування), комбінування таких фільтрів із базовими рекомендаціями на основі популярності чи історії переглядів.

Таким чином, концепції та методи побудови рекомендаційних систем охоплюють широке коло підходів – від простих фільтрів до складних моделей машинного навчання. Конкретний вибір залежить від обсягу та якості даних, вимог до точності рекомендацій, обмежень платформи та складності програмної реалізації. У подальших підрозділах ці теоретичні підходи будуть враховані при проєктуванні мобільного застосунку.

1.3 Моделі та алгоритми підбору рецептів і формування меню

Задача підбору рецептів і формування меню з точки зору інформатики – це вибір потрібних об’єктів (рецептів) із великої множини можливих варіантів з урахуванням певних умов. У найпростішому випадку користувач хоче «знайти страви, які можна приготувати з наявних продуктів» або «підібрати меню на день, щоб страви відрізнялись і займали небагато часу на приготування».

Щоб реалізувати таку функціональність у програмному засобі, використовують різні моделі та алгоритми. Спільною рисою для них є наявність описів рецептів (інгредієнти, час приготування, тип страви) та набору правил, за якими система відбирає й комбінує ці рецепти [10].

Однією з базових моделей є фільтрація за параметрами. У ній кожен рецепт має набір атрибутів, серед яких перелік інгредієнтів, тип страви (сніданок, обід, вечеря), орієнтовний час приготування, складність, іноді калорійність. Користувач задає потрібні умови, а система відбирає лише ті рецепти, які цим умовам відповідають. Наприклад, можна обрати «страви до 30 хвилин» або «рецепти, що містять курку та овочі». Такий підхід легко реалізувати, він добре масштабується і зрозумілий для користувача.

Ще один поширений підхід – контентно-орієнтована модель, де система орієнтується не тільки на поточний запит, а й на те, які рецепти користувач переглядав або додавав у вибране раніше. Рецепти описуються як набір ознак (категорія страви, основний інгредієнт, час приготування тощо), а далі обчислюється схожість між новими рецептами та тими, що вже сподобалися користувачеві. У спрощеному варіанті це може бути навіть підбір «аналогічних» страв, якщо користувач часто обирає салати з овочами, система частіше пропонуватиме подібні рецепти.

У задачі формування меню важливо не лише знайти окремі страви, а й скласти з них певну послідовність. Меню на день зазвичай включає кілька різних позицій (наприклад, перша страва, друга страва, салат або десерт), тому алгоритм повинен враховувати різноманітність страв і обмеження по часу. На практиці тут зручно використовувати прості правила, коли спочатку формується список кандидатів для кожної категорії страв, а потім з нього обирається по одній позиції для включення в меню. Якщо додаються обмеження за часом приготування, алгоритм просто відкидає варіанти, які виходять за заданий ліміт [11].

У більш складних системах для підбору рецептів можуть застосовуватися елементи машинного навчання, коли модель навчається на історії вибору

багатьох користувачів і прогнозує, які страви з більшою ймовірністю сподобаються конкретній людині. Проте для побутового мобільного застосунку цілком достатньо поєднання фільтрації за параметрами, простих правил формування меню та, за потреби, базових рекомендацій «популярних» або «часто обираних» страв.

На (рис. 1.1) подано приклад загальної схеми роботи рекомендаційної системи. На вхід надходить профіль користувача та база об’єктів, далі система формує список рекомендацій. Подібний підхід застосовується і в задачі підбору рецептів і меню, де замість товарів чи фільмів використовуються кулінарні рецепти, а результатом є список страв або готове меню для користувача.

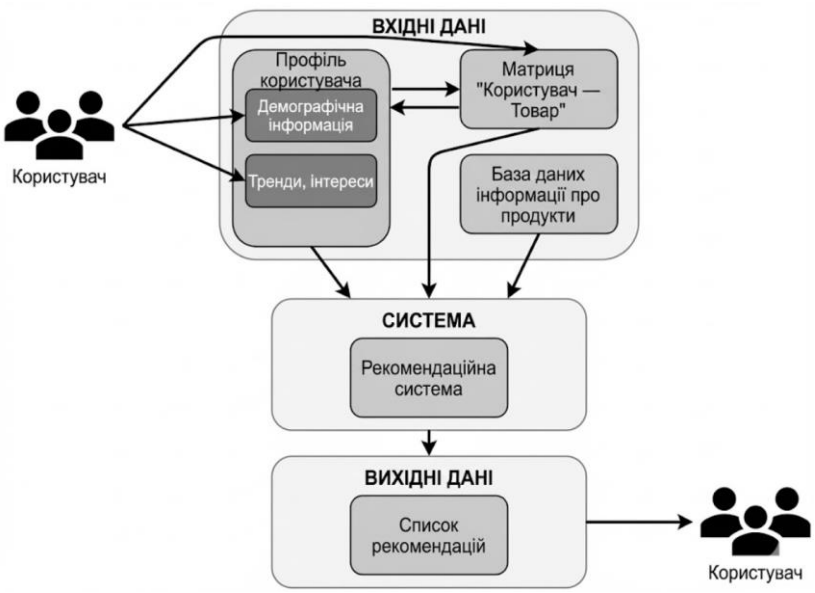

Рисунок 1.1. – Схема роботи рекомендаційних систем

1.4 Огляд та аналіз аналогічних програмних розробок

У сфері підбору рецептів та планування харчування вже існує низка мобільних застосунків. Для даної роботи розглянемо три типових представники: Recipe Keeper, Tasty та Mealime. Вони частково розв’язують подібні задачі, але мають різний акцент у функціональності, що дозволяє виділити як корисні ідеї, так і недоліки, яких бажано уникнути.

Recipe Keeper (рис. 1.2) позиціонується як «усе-в-одному» органайзер рецептів. Основний функціонал включає створення власної колекції рецептів, імпорт рецептів із вебсайтів, додавання фотографій, категорій та позначення улюблених страв. Також реалізовано планувальник харчування (тижневі та місячні плани) і список покупок, який формується автоматично на основі вибраних рецептів [12].

З погляду технологій і архітектури, Recipe Keeper реалізовано як кросплатформений мобільний застосунок з клієнт-серверною архітектурою: основна логіка зберігання та синхронізації даних розміщена на віддаленому сервері, а мобільний клієнт працює з локальною копією даних та періодично синхронізує зміни з хмарним сховищем. Такий підхід забезпечує доступ до однієї й тієї ж бази рецептів з різних пристроїв користувача.

Переваги Recipe Keeper:

- зручне зберігання власних рецептів і гнучка організація колекції (категорії, теги, фото);
- наявність вбудованого планувальника та списку покупок;
- підтримка синхронізації між декількома пристроями.

Недоліки:

- обмежена кількість готових рецептів «з коробки» – новому користувачу потрібно спочатку наповнити базу;
- акцент на ручному введенні, а не на автоматизованому підборі рецептів чи меню;
- відсутність розвинених рекомендаційних механізмів.

Сфера застосування – користувачі, які вже мають власну колекцію рецептів і хочуть перевести її в цифровий формат та поєднати з плануванням харчування. Для ситуацій, коли потрібно швидко підібрати нові страви за наявними інгредієнтами або згенерувати меню без попереднього заповнення бази, можливості Recipe Keeper є обмеженими. Саме ця особливість підкреслює потребу в застосунку, який поєднує зовнішнє джерело рецептів з простим сценарієм щоденного планування.

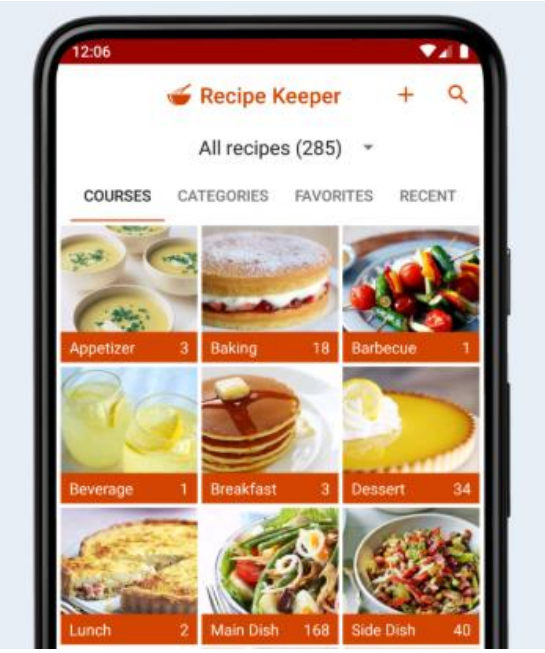

Рисунок 1.2. – Мобільний застосунок Recipe Keeper

Tasty (рис. 1.3) – застосунок від платформи BuzzFeed із великою колекцією покрокових рецептів. Основний функціонал включає перегляд детальних рецептів, відеоінструкції, покроковий режим приготування («cooking mode»), можливість зберігати рецепти в розділі «My Recipes» та фільтрувати їх за інгредієнтами, кухнею, нагодою, складністю тощо [13].

На рівні архітектури Tasty побудований як класичний клієнт-серверний застосунок: мобільний клієнт відповідає за відображення інтерфейсу та відправку запитів, а основні дані (рецепти, відео, фільтри) зберігаються та обробляються на стороні вебсервісу. Локальне збереження обмежене кешем останніх переглядів і списком збережених рецептів, тому більшість функцій залежить від інтернет-з’єднання.

Переваги Tasty:

- велика база готових рецептів;
- візуально привабливі відео та покрокові інструкції;
- гнучка система фільтрів за різними параметрами.

Недоліки:

- обмежений офлайн-режим, основна частина контенту потребує з’єднання з інтернетом;

- наявність реклами та внутрішніх покупок;
- слабкий акцент на системному формуванні повноцінного меню (застосунок більше орієнтований на окремі страви).

Сфера застосування – користувачі, які шукають ідеї нових страв, хочуть візуальні інструкції та не мають жорстких вимог до планування усього денного меню.

Рисунок 1.3. – Мобільний застосунок Tasty

Mealime (рис. 1.4) орієнтований насамперед на планування повсякденного харчування. При першому налаштуванні користувач обирає харчові вподобання, тип дієти та небажані інгредієнти, після чого застосунок генерує персоналізовані плани харчування і автоматично формує список покупок на основі обраних страв [14; 15]. Архітектурно Mealime реалізує клієнт-серверний підхід із сильним серверним ядром: генерація меню, зберігання профілю користувача та розрахунок харчової цінності виконуються на сервері, а мобільний застосунок отримує вже готові дані та зберігає їх локально для оперативного доступу. Доступна безкоштовна версія із базовим функціоналом та підписка Mealime Pro

з розширеними можливостями (додаткові рецепти, поглиблена інформація про харчову цінність тощо).

Переваги Mealime:

- зручне формування меню на кілька днів або тиждень;
- підтримка дієтичних профілів і харчових обмежень;
- автоматичний список покупок.

Недоліки:

- ключові розширені функції доступні лише за платною підпискою;
- логіка роботи жорстко прив’язана до певної структури планів харчування;
- менша гнучкість у випадку, коли користувач хоче просто швидко підібрати кілька страв на день без детального тижневого плану.

Сфера застосування – користувачі, які хочуть системно планувати свій раціон (особливо з дієтичними обмеженнями) та готові витрачати час на налаштування профілю і роботу з тижневими планами.

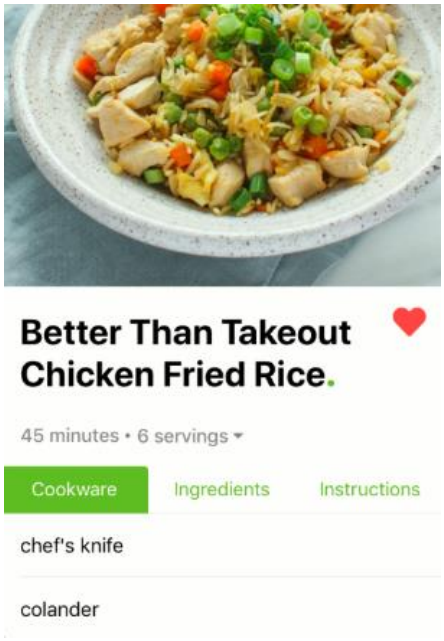

Рисунок 1.4. – Мобільний застосунок Mealime

Узагальнені результати порівняння розглянутих застосунків за основними критеріями (функціонал, джерела та зберігання даних, переваги, недоліки, орієнтація на планування меню) наведено в порівняльній таблиці (див. табл. В.1

у додатку В). З таблиці видно, що Recipe Keeper робить акцент на зберіганні та організації власних рецептів користувача, але менш придатний для швидкого старту без попередньо заповненої бази. Tasty забезпечує великий обсяг готового контенту та зручний візуальний формат, однак майже не підтримує повноцінне планування меню й сильно залежить від наявності інтернету. Mealime є найближчим до задачі планування меню, проте частина важливого функціоналу платна, а структура планів досить жорстко задана.

На основі аналізу можна сформулювати вимоги до програмної реалізації мобільного застосунку, що розробляється в межах даної кваліфікаційної роботи. На відміну від розглянутих аналогів, запропонований застосунок має:

- використовувати зовнішній API рецептів для швидкого старту без ручного наповнення бази та підтримувати пошук за інгредієнтами, категоріями;
- забезпечувати локальне збереження обраних рецептів та сформованих меню з можливістю роботи в офлайн-режимі без обов’язкової реєстрації чи платної підписки;
- реалізовувати просту модель формування меню на день, що дозволяє швидко підібрати кілька страв на день;
- мати мінімалістичний та інтуїтивно зрозумілий інтерфейс, орієнтований на швидкі повсякденні сценарії взаємодії.

Таким чином, проведений огляд і порівняльний аналіз показали, що жоден із розглянутих застосунків повністю не задовольняє вимоги до простого, безкоштовного та гнучкого інструменту для підбору рецептів і формування базового меню на день. Це обґрунтовує доцільність розробки власного мобільного застосунку з гібридним підходом до роботи з даними та спрощеною моделлю планування, що й буде реалізовано у подальших розділах роботи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Постановка задачі та вимоги до програмної реалізації

У попередньому розділі було показано, що процес підбору рецептів і формування меню зазвичай виконується користувачем вручну, зокрема через пошук інформації в інтернеті, перегляд кулінарних сайтів або використання окремих застосунків з обмеженим функціоналом. Такий підхід є малоефективним, вимагає багато часу та не дає можливості швидко сформувати набір страв на певний період з урахуванням базових умов, наприклад наявних інгредієнтів чи тривалості приготування.

Проблема, що розв’язується в межах кваліфікаційної роботи, полягає в необхідності створення простого у використанні мобільного застосунку, який дозволяє користувачеві підбирати рецепти за заданими параметрами, переглядати докладну інформацію про страви та формувати меню. При цьому застосунок повинен працювати на поширеній мобільній платформі, використовувати зовнішнє джерело даних із рецептами та забезпечувати збереження вибраної інформації в локальному сховищі для зручного повторного доступу.

Задача має прикладний характер і пов’язана з низкою важливих практичних аспектів, серед яких оптимізація побутових процесів, економія часу користувача та підтримка прийняття рішень у повсякденному плануванні харчування. З точки зору програмної інженерії це означає необхідність проєктування та реалізації системи, яка поєднує роботу з зовнішнім API, локальною базою даних і користувацьким інтерфейсом на мобільному пристрої.

Постановку задачі можна сформулювати таким чином. Необхідно розробити мобільний застосунок для платформи Android, який забезпечує:

- отримання рецептів із зовнішнього сервісу за допомогою програмного інтерфейсу (API);

- фільтрацію та пошук рецептів за базовими параметрами (інгредієнти, категорія страви, орієнтовний час приготування тощо);
- відображення структурованої інформації про рецепт (назва, зображення, список інгредієнтів, покроковий опис приготування);
- можливість додавання рецептів до локального «обраного» та формування простого меню з страв;
- збереження даних у локальній базі для подальшого перегляду без повторного звернення до зовнішнього сервісу.

Виходячи з поставленої задачі, визначимо основні функціональні вимоги до програмної реалізації:

- інтерфейс застосунку має надавати користувачеві головний екран із доступом до основних розділів (пошук рецептів, список обраних, меню);
- реалізується механізм пошуку та фільтрації рецептів (наприклад, за ключовим словом або за інгредієнтами), що працює поверх даних, отриманих із зовнішнього API;
- для кожного рецепта відображаються основні атрибути, зокрема назва, ілюстративне зображення (за наявності), перелік інгредієнтів, орієнтовний час приготування, опис кроків;
- користувач може додавати рецепти до списку обраних, видаляти їх із цього списку;
- формується меню (наприклад, набір страв на день), з можливістю перегляду та оновлення цього набору;
- дані про обрані рецепти та сформовані меню зберігаються в локальному сховищі мобільного пристрою.

Окрім функціональних, до мобільного застосунку висуваються й нефункціональні вимоги. Інтерфейс повинен бути простим, інтуїтивно зрозумілим та адаптованим до екранів мобільних пристроїв. Час відгуку на основні дії користувача (перехід між екранами, відкриття рецепта, додавання в обране) має бути прийнятним для комфортної роботи. Застосунок повинен коректно реагувати на відсутність або нестабільність мережевого з’єднання, і в

такому разі користувач має мати доступ до раніше збережених рецептів. Також важливо забезпечити коректну роботу застосунку на типових для сучасних пристроїв версіях операційної системи Android [16].

Цільовою платформою є смартфони під керуванням операційної системи Android із мінімальною версією, яка відповідає сучасним вимогам сумісності (конкретна версія визначається під час реалізації). Для роботи із зовнішнім сервісом обов'язковою є наявність доступу до мережі Інтернет, тоді як операції з уже збереженими рецептами та меню мають бути доступними також у режимі без підключення до мережі.

Сформульовані вище постановка задачі та вимоги до програмної реалізації є основою для розробки технічного завдання та подальшого проєктування архітектури мобільного застосунку, що буде розглянуто в наступних підрозділах.

2.2 Методологія дослідження

Методологія даної кваліфікаційної роботи поєднує підхід до вивчення предметної області та класичну каскадну модель життєвого циклу програмного забезпечення, адаптовану до розробки невеликого мобільного застосунку.

На етапі дослідження предметної області було виконано аналіз літературних джерел, інтернет-ресурсів та існуючих програмних продуктів, що стосуються підбору рецептів, рекомендаційних систем та мобільних застосунків у сфері організації харчування. Це дозволило визначити основні задачі, які має розв'язувати програмний засіб, а також окреслити типові підходи до їх реалізації. Окремо було проаналізовано кілька популярних застосунків для роботи з рецептами та планування меню, що дало змогу сформувати перелік функцій, які доцільно реалізувати в межах даного проєкту.

Для безпосередньої розробки мобільного застосунку було обрано каскадну модель життєвого циклу (Рис. 2.1). У рамках цієї моделі процес створення програмного забезпечення розбивається на послідовні етапи, де кожний

наступний базується на результатах попереднього [17]. У спрощеному вигляді послідовність етапів виглядає так:

аналіз вимог → проєктування → реалізація → тестування → впровадження (демонстрація результатів).

Рисунок 2.1. – Каскадна модель розробки

На етапі аналізу вимог було сформульовано мету роботи, окреслено функціональні та нефункціональні вимоги до мобільного застосунку, визначено цільову платформу, спосіб взаємодії із зовнішнім API та формат збереження локальних даних. Результатом цього етапу стала постановка задачі та початковий перелік функцій, які необхідно реалізувати.

На етапі проєктування розроблено логічну структуру застосунку, модель даних та архітектуру програмного забезпечення. Було визначено основні екрани мобільного застосунку, навігацію між ними, формат подання інформації про рецепт, структуру таблиць локальної бази даних, а також загальну схему

взаємодії між компонентами (інтерфейс, робота з Spoonacular API, збереження даних).

Етап реалізації передбачав створення програмного коду мобільного застосунку в середовищі Android Studio з використанням мови програмування Kotlin та обраних бібліотек (для роботи з інтерфейсом, мережею та базою даних). На цьому етапі були послідовно реалізовані основні модулі, зокрема отримання рецептів із зовнішнього API, відображення списку рецептів, перегляд деталей страви, додавання до обраного, формування меню та робота з локальною базою даних.

На етапі тестування було перевірено коректність роботи окремих екранів та функцій, а також застосунку в цілому. Проводилося функціональне тестування основних сценаріїв використання, до яких належать пошук рецептів, перегляд детальної інформації, додавання рецептів до обраного, формування меню, робота при відсутності мережевого з’єднання (для локально збережених даних). Виявлені помилки виправлялися, після чого відповідні сценарії перевірялися повторно.

Завершальний етап впровадження та демонстрації полягав у підготовці застосунку до представлення на захисті кваліфікаційної роботи, зокрема у формуванні демонстраційних сценаріїв, описі інтерфейсу та функціональних можливостей, а також підготовці супровідних матеріалів (скріншотів, інструкцій для користувача, пояснень щодо встановлення та запуску програми).

Хоча каскадна модель передбачає послідовне виконання етапів, у процесі розробки окремі кроки частково переглядалися. Після тестування вносилися зміни до проєктних рішень або реалізації окремих компонентів. Проте загальна логіка «від вимог до проєктування, далі до реалізації та тестування» залишалася незмінною, що відповідає формату навчального програмного проєкту та дозволяє чітко описати процес розробки мобільного застосунку в пояснювальній записці [18].

2.3 Теоретичні аспекти дослідження та архітектура програмного забезпечення

Під час розробки будь-якого прикладного програмного забезпечення важливо не лише реалізувати потрібний функціонал, а й організувати код так, щоб його було легко супроводжувати, розширювати та тестувати. Для цього застосовують архітектурні підходи, що передбачають поділ програми на логічні частини (шари) з чітко визначеними зонами відповідальності [19].

Для мобільних застосунків на Android, Google рекомендує використовувати багатошарову архітектуру, у якій виділяють щонайменше два основні шари – UI-шар (шар інтерфейсу користувача) та шар даних [20]. Додатково може використовуватися проміжний доменний шар для інкапсуляції бізнес-логіки.

У цій роботі для розробки мобільного застосунку було прийнято спрощений варіант такої архітектури з поділом на три основні рівні:

- шар представлення (UI) – екрани застосунку, які відображають дані користувачеві та реагують на його дії;
- шар логіки та керування даними – компоненти, що отримують дані з джерел, готують їх до відображення та передають у інтерфейс;
- шар даних – робота з зовнішнім API (мережеві запити) та локальною базою даних (збереження обраних рецептів і меню).

Як теоретичну основу було обрано підхід, близький до архітектурного шаблону MVVM (Model-View-ViewModel), який широко застосовується в Android-розробці для розділення інтерфейсу та бізнес-логіки. У типовій реалізації MVVM View відповідає за відображення, ViewModel – за підготовку даних та обробку подій, а Model – за доступ до даних із різних джерел . У даному проєкті принципи MVVM використовуються у спрощеному вигляді, з акцентом на розділенні коду інтерфейсу та роботи з даними [21].

На UI-шарі розташовані екрани застосунку, реалізовані засобами Jetpack Compose. Вони відповідають за відображення списку рецептів, детальної

інформації про страву, списку обраних рецептів та меню. UI-компоненти отримують уже підготовлені дані та не виконують складної обробки – основна логіка зосереджена на нижчих шарах. Це полегшує зміну оформлення та структури інтерфейсу без значних змін у логіці [22].

Середній шар (логіки й керування даними) відповідає за взаємодію між інтерфейсом і джерелами даних. У типовій MVVM-структурі його ролі відповідає ViewModel або набір класів-посередників, які:

- отримують дані з репозиторію (API або локальної бази);
- перетворюють їх у зручний для відображення вигляд (списки, об’єкти з потрібними полями);
- реагують на дії користувача (вибір рецепта, додавання до обраного, формування меню) та оновлюють стан інтерфейсу.

Такий поділ дозволяє відокремити бізнес-логіку від деталей реалізації інтерфейсу, що підвищує зрозумілість коду та спрощує модифікацію застосунку в майбутньому.

Шар даних включає два основні компоненти – клієнт для роботи із зовнішнім API та локальну базу даних. Для доступу до мережевого сервісу використовується бібліотека типу Retrofit, яка виконує HTTP-запити та перетворює відповіді сервера у зручні об’єкти. Локальне збереження обраних рецептів і меню реалізується за допомогою бібліотеки Room поверх вбудованої СУБД SQLite. Такий підхід відповідає рекомендованим практикам Androidархітектури, де для шарів даних виділяють окремі класи-джерела (локальні та віддалені), які інкапсулюють деталі доступу до даних .

Оскільки важливо не лише джерело даних, а й спосіб їхньої внутрішньої організації, структуру локальної бази даних було спроєктовано в максимально простому й водночас достатньо інформативному вигляді. На рис. 2.2 подано ER-діаграму структури таблиці RECIPES, яка використовується для зберігання інформації про страви. Таблиця містить локальний первинний ключ local_id, ідентифікатор рецепта у зовнішньому сервісі remote_id (за його наявності), текстові поля для назви (title), короткого опису (summary), інгредієнтів

(ingredients) та інструкцій (instructions), а також додаткові характеристики – посилання на зображення (image_url), орієнтовний час приготування (time_minutes), калорійність (calories) та тип прийому їжі (meal_type). Окремі логічні стани (is_favorite, is_custom) зберігаються у вигляді булевих полів, що дозволяє на рівні однієї таблиці виділяти підмножини «обраних» рецептів та рецептів, створених користувачем, без введення додаткових зв'язувальних таблиць.

RECIPES			
LONG	local_id	PK	Локальний ID запису
INT	remote_id		ID рецепта у API (null для власних)
STRING	title		Назва страви
STRING	summary		Короткий опис
STRING	ingredients		Інгредієнти (текст)
STRING	instructions		Кроки приготування
STRING	image_url		URL зображення (може бути null)
INT	time_minutes		Час приготування, хв
INT	calories		Калорійність
STRING	meal_type		Тип прийому їжі
BOOLEAN	is_favorite		Прапорець: обране
BOOLEAN	is_custom		Прапорець: власний рецепт

Рисунок 2.2. – ER-діаграма структури таблиці RECIPES

Таким чином, модель даних локальної БД органічно вписується в загальну архітектуру застосунку. Шар даних представляє рецепти у вигляді єдиної сутності, яка використовується як у мережевій, так і в локальній частині логіки. Це спрощує реалізацію репозиторію, зменшує кількість перетворень між різними форматами та дозволяє зручно фільтрувати й відбирати записи за потрібними критеріями.

Загалом архітектура розроблюваного мобільного застосунку базується на таких принципах:

- розділення інтерфейсу, бізнес-логіки та роботи з даними за допомогою шарів;
- мінімізація залежностей між шарами (UI не звертається безпосередньо до мережевого клієнта чи бази даних);
- можливість подальшого розширення (додавання нових екранів, джерел даних або логіки без кардинальної переробки всієї програми).

На рис. 2.3 наведено приклад загальної схеми MVVM-архітектури з використанням репозиторію та локальних/віддалених джерел даних, подібної до тієї, що використовується в даному застосунку. Така схема демонструє напрямки потоків даних між шарами і слугує основою для подальшого детального опису модулів у наступних підрозділах.

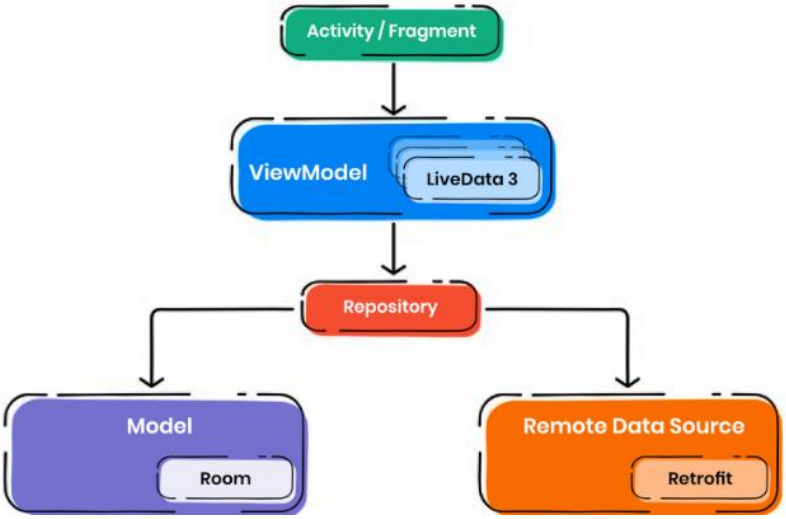

Рисунок 2.3. – Приклад загальної схеми MVVM-архітектури

Для кращого розуміння загальної структури системи на рис. 2.4 подано узагальнений вигляд архітектури. У цій схемі виділено три вузли – шар представлення (екрани, реалізовані на Jetpack Compose), шар логіки (ViewModel та пов’язана з нею бізнес-логіка) та шар даних (репозиторій зі своїми віддаленим і локальним джерелами даних, зовнішній веб-сервіс Spoonacular API та локальна база даних Room/SQLite). Стрілками показано основні напрями потоків даних, звернення від UI до шару логіки, подальше надсилання запитів до шару даних, а

також повернення обробленої інформації назад у вигляді стану інтерфейсу. Така узагальнена схема дозволяє наочно продемонструвати, що всі звернення до мережі та бази даних проходять виключно через шар логіки й репозиторій, а інтерфейс працює лише з підготовленими моделями, що сприяє низькій зв’язаності компонентів і спрощує подальший розвиток застосунку.

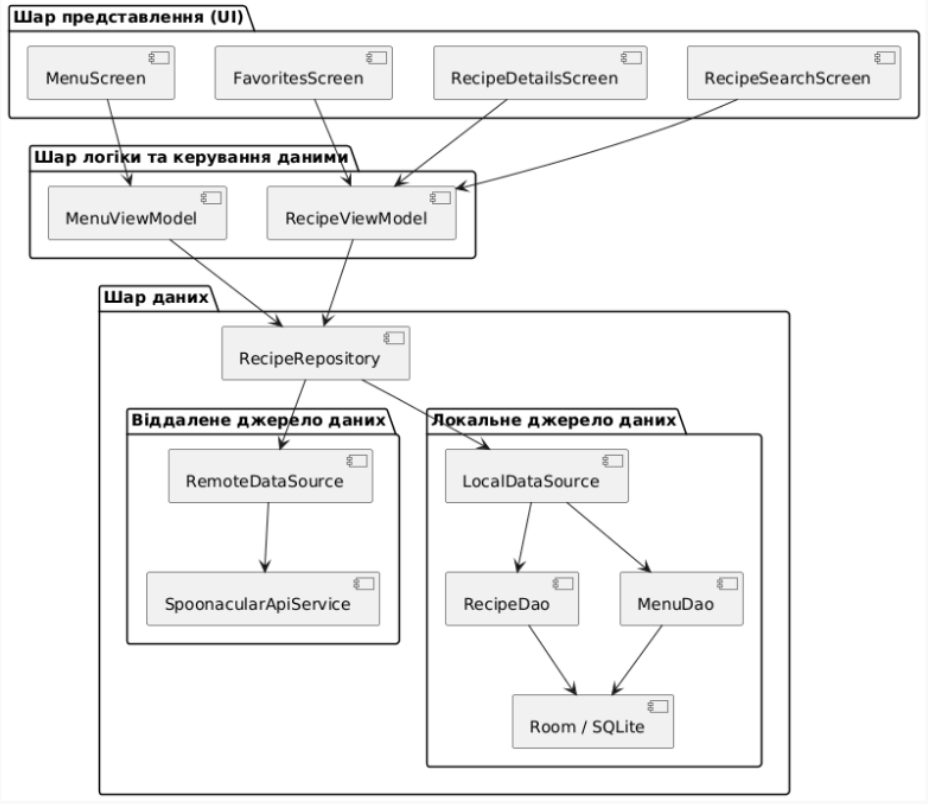

Рисунок 2.4. – Архітектура мобільного застосунку

2.4 Обґрунтування вибору інструментальних засобів розробки

Для розробки мобільного застосунку під Android існує багато фреймворків та мов програмування, проте при виконанні цієї кваліфікаційної роботи було обрано нативний підхід із використанням офіційних інструментів платформи. Це дозволяє спиратися на актуальні рекомендації розробників Android, отримати доступ до всіх можливостей системи та забезпечити стабільну роботу програми на реальних пристроях.

Основними інструментальними засобами, що використовуються в проєкті, є:

- Android Studio – офіційне середовище розробки для Android, яке містить редактор коду, засоби збірки, емулятори та налагодження. Воно надає шаблони проєктів, інтеграцію з системою збірки Gradle та зручні інструменти для роботи з інтерфейсом і ресурсами [23].
- Мова програмування Kotlin – сучасна мова, рекомендована Google як основна для Android-розробки. Вона підтримує лаконічний синтаксис, безпечну роботу з null-значеннями та добре інтегрується з наявними Java-бібліотеками [24].
- Jetpack Compose – сучасний декларативний фреймворк для побудови інтерфейсу користувача в Android-застосунках. Він дозволяє створювати гнучкі й адаптивні інтерфейси за допомогою функцій-компонентів, спрощує оновлення UI при зміні стану даних та рекомендований як основний підхід до розробки нового інтерфейсу на Android [25].
- Room – бібліотека рівня доступу до даних (ORM) з набору Android Jetpack, яка працює поверх SQLite і спрощує роботу з локальною базою даних. Room забезпечує компіляторну перевірку SQL-запитів і зручну роботу з об’єктами Kotlin, що знижує ймовірність помилок у запитах та спрощує збереження рецептів і меню [26].
- Retrofit – популярна бібліотека для виконання HTTP-запитів, яка дозволяє легко описувати методи доступу до зовнішнього API та перетворювати відповіді сервера в об’єкти Kotlin. У даному проєкті за допомогою Retrofit реалізовано інтеграцію з Spoonacular API – зовнішнім сервісом, що надає доступ до великої бази кулінарних рецептів, пошуку за інгредієнтами та пов’язаної інформації про страви [27].

Вибір саме цих інструментів обумовлений кількома факторами. По-перше, Android Studio, Kotlin, Jetpack Compose, Room і Retrofit фактично стали стандартом для сучасної нативної Android-розробки, мають розгорнуту документацію та активну спільноту. Це знижує ризики, пов’язані з

використанням застарілих або мало підтримуваних технологій. По-друге, такий стек добре підходить для навчального проєкту і дає змогу продемонструвати актуальні технічні підходи (декларативний UI, робота з локальною базою даних, взаємодія з вебсервісом) без надмірного ускладнення архітектури.

Альтернативами могли бути кросплатформені фреймворки (наприклад, Flutter або React Native) або використання класичного підходу до побудови інтерфейсу на основі XML-розмітки. Однак такі варіанти або потребують вивчення додаткових інструментів і мов програмування, або ґрунтуються на менш сучасних підходах до UI. З огляду на те, що даний проєкт орієнтований саме на платформу Android та має обмежений обсяг, використання нативного стеку з Jetpack Compose, Room і Retrofit та інтеграцією зі Spoonacular API є більш логічним та виправданим рішенням.

У подальших підрозділах на основі обраних інструментальних засобів буде розглянуто етапи програмної реалізації, організацію тестування та аналіз отриманих результатів.

2.5 Етапи програмної реалізації та структура компонентів мобільного застосунку

Розробка мобільного застосунку виконувалася в середовищі Android Studio з використанням мови програмування Kotlin та фреймворку Jetpack Compose для побудови інтерфейсу користувача. Структура проєкту організована таким чином, щоб розділити відповідальності між різними частинами програми, зокрема доступом до даних, логікою та відображенням інтерфейсу.

У корені пакету com.example.recipemenuapp розміщено основний клас MainActivity, з якого розпочинається робота застосунку. У ньому налаштовуються тема, базовий контейнер інтерфейсу та навігація між екранами. Решта коду згрупована в пакети, які відображають основні напрями функціональності застосунку. Умовно структуру можна подати так:

- data.local – компоненти локальної бази даних, зокрема класи AppDatabase, RecipeDao, RecipeEntity, а також допоміжний клас RecipeMappers для перетворення даних;
- data.remote – класи для взаємодії із зовнішнім сервісом рецептів SpoonacularApi та допоміжний компонент TranslationHelper;
- ui.screens – окремі екрани застосунку (HomeScreen, RecipeDetailScreen, FavoritesScreen, MenuScreen) та відповідні їм моделі представлення (HomeViewModel, RecipeDetailViewModel, FavoritesViewModel, MenuViewModel), а також допоміжні файли на кшталт MealTypeMapper та Screens;
- ui.theme – файли, що визначають візуальний стиль застосунку (Color.kt, Theme.kt, Type.kt).

Така організація дає змогу чітко відокремити частину, що відповідає за зберігання та отримання даних, частину, яка виконує перетворення і підготовку цих даних до відображення, та власне інтерфейс користувача. Шар даних охоплює локальну базу Room і мережеві запити до Spoonacular API, шар логіки складається з ViewModel-ів та пов’язаної з ними бізнес-логіки, а шар інтерфейсу включає екрани на Jetpack Compose та спільні налаштування теми. У підсумку структура пакунків відображає поділ застосунку на три основні рівні, описані в попередньому підрозділі, і забезпечує зрозумілу схему підтримки та розширення коду.

На рис. 2.5-2.6 (скріншот структури проєкту з Android Studio) наведено загальний вигляд пакунків і файлів застосунку. Ця структура є основою для подальшого опису значущих компонентів – моделей даних, роботи з API, локальної бази даних, екранів і логіки, що забезпечує підбір рецептів і формування меню.

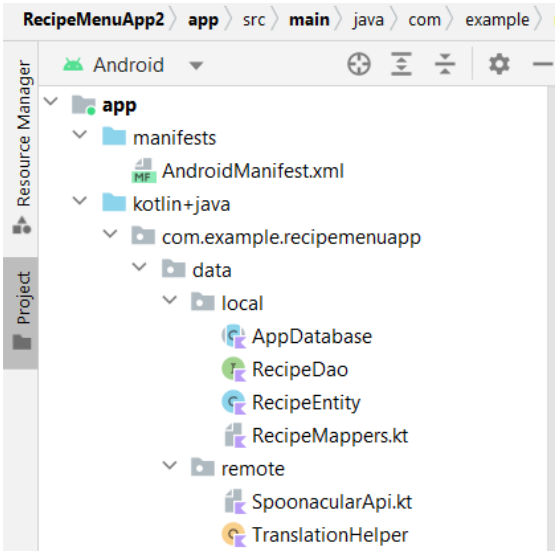

Рисунок 2.5. – Структура проєкту

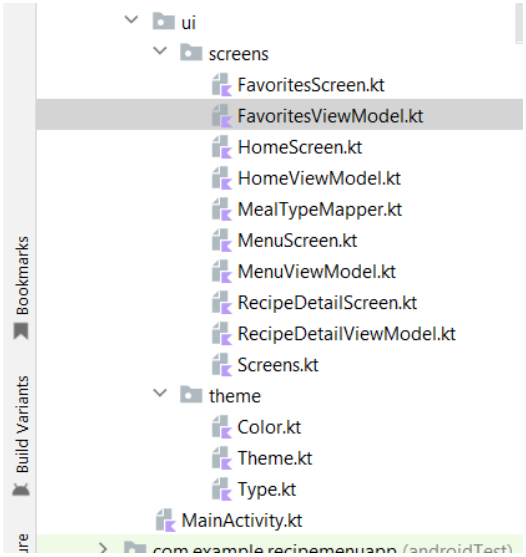

Рисунок 2.6. – Структура проєкту

Після визначення загальної структури пакунків та ролей основних компонентів наступним кроком стало проєктування та реалізація моделей даних, які використовуються в мобільному застосунку. Моделі мають бути зручними як для відображення інформації в інтерфейсі, так і для зберігання в локальній базі даних та обробки результатів від зовнішнього API. Важливо також забезпечити узгодженість структури даних між різними шарами застосунку, щоб одна й та сама інформація не дублювалася у різних форматах без необхідності.

У проєкті використано окремий пакет data.local для опису сутностей, що зберігаються в базі даних, та пакет data.remote для роботи зі структурами, які повертає вебсервіс. Такий поділ дозволяє не прив’язувати внутрішню модель застосунку жорстко до формату відповіді зовнішнього API й за потреби змінювати джерело даних без повної переробки програми. Для перетворення між мережевими моделями та локальними сутностями використовується спеціальний клас-мепер, що спрощує підтримку коду й зменшує ризик помилок під час змін у форматі даних.

Основною сутністю локальної бази є об’єкт рецепта. Для нього було створено клас RecipeEntity, який відображає таблицю в SQLite через бібліотеку Room. Оголошення сутності подано на рис. 2.7.

```kotlin
// Позначаємо клас як сутність Room, що зберігається в таблиці "recipes"
@Entity(tableName = "recipes")
data class RecipeEntity(
    @PrimaryKey(autoGenerate = true) // Первинний ключ рецепта, унікальний для кожного запису
    val localId: Long = 0L,          // локальний ID в БД

    val remoteId: Int?,              // ID з API (null - для своїх рецептів)
    val title: String,               // Назва страви
    val summary: String,
    val ingredients: String,         // Текстовий опис інгредієнтів, збережений у спрощеному форматі
    val instructions: String,        // Покроковий опис приготування страви
    val imageUrl: String?,           // Посилання на зображення страви
    val timeMinutes: Int,            // Орієнтовний час приготування у хвилинах
    val calories: Int,
    val mealType: String,            // зберігаємо enum як рядок

    val isFavorite: Boolean,         // true - у вкладці "Обране"
    val isCustom: Boolean            // true - власний рецепт користувача
)
```

Рисунок 2.7. – Реалізація класу RecipeEntity

Поле id використовується як первинний ключ і відповідає ідентифікатору рецепта у зовнішньому сервісі. Додаткові поля (title, imageUrl, timeMinutes, ingredients, instructions) зберігають інформацію, необхідну для відображення у списках та на екрані деталей рецепта. Інформація про інгредієнти та інструкції зберігається у вигляді рядків, що спрощує структуру таблиці й достатньо для цілей застосунку.

Для роботи з даними, які повертає Spoonacular API, у пакеті data.remote використовується окрема модель. Вона відображає поля відповіді сервера, причому не всі ці поля потрібні безпосередньо в інтерфейсі. Тому частина з них або ігнорується, або відразу при перетворенні відкидається. Модель для мережевої відповіді подано на рис. 2.8.

```kotlin
// Модель окремого рецепта у відповіді від Spoonacular API
data class SpoonacularRecipeDto(
    @SerializedName("id") val id: Int?,  // Ідентифікатор рецепта у зовнішньому сервісі
    @SerializedName("title") val title: String?,  // Назва страви у відповіді API
    @SerializedName("image") val image: String?,  // Повне посилання на зображення
    @SerializedName("readyInMinutes") val readyInMinutes: Int?,    // Час приготування у хвилинах
    @SerializedName("summary") val summary: String?,
    @SerializedName("dishTypes") val dishTypes: List<String>?,
    @SerializedName("nutrition") val nutrition: SpoonacularApi.NutritionDto?
)
```

Рисунок 2.8. – Реалізація класу SpoonacularRecipeDto

Оскільки формат даних у локальній базі та у відповіді API відрізняється, доцільно мати окремий шар перетворення. Для цього в пакеті data.local було створено допоміжний клас RecipeMappers, який містить функції конвертації між моделями. Функцію подано на рис. 2.9.

Тут використовується розширювальна функція для типу SpoonacularRecipeDto, яка створює відповідний екземпляр RecipeEntity. Поля ingredients та instructions можуть надходити з інших частин відповіді API або заповнюватися пізніше, тому вони передаються як додаткові параметри з типовими значеннями.

```
// Функція перетворює об'єкт, отриманий із мережі, у сутність для зберігання в БД
fun RecipeDetailUiState.toEntity(
    remoteId: Int?,
    isCustom: Boolean
): RecipeEntity {
    return RecipeEntity(
        localId = 0L,
        remoteId = remoteId,
        title = title,
        summary = summary,
        ingredients = ingredients,
        instructions = instructions,
        imageUrl = imageUrl,
        timeMinutes = timeMinutes,
        calories = calories,
        mealType = mealType.name,
        isFavorite = isFavorite,
        isCustom = isCustom
    )
}
```

Рисунок 2.9. – Функція конвертації між моделями

На основі визначених сутностей було створено компонент доступу до локальної бази даних – інтерфейс RecipeDao (рис. 2.10). У ньому описані основні операції читання й запису для роботи з обраними рецептами та меню.

```kotlin
@Dao
interface RecipeDao {

    // Усі обрані рецепти (для вкладки "Обране")
    @Query("SELECT * FROM recipes WHERE isFavorite = 1 ORDER BY localId DESC")
    fun getFavoriteRecipes(): Flow<List<RecipeEntity>>

    // Усі власні рецепти (для вкладки "Мої рецепти")
    @Query("SELECT * FROM recipes WHERE isCustom = 1 ORDER BY localId DESC")
    fun getMyRecipes(): Flow<List<RecipeEntity>>

    // Знайти рецепт по remoteId (ID з API)
    @Query("SELECT * FROM recipes WHERE remoteId = :remoteId LIMIT 1")
    suspend fun getByRemoteId(remoteId: Int): RecipeEntity?

    // Вставка / оновлення
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(recipe: RecipeEntity): Long

    // Видалити по remoteId (коли забираємо з обраних)
    @Query("DELETE FROM recipes WHERE remoteId = :remoteId")
    suspend fun deleteByRemoteId(remoteId: Int)
}
```

Рисунок 2.10. – Інтерфейс RecipeDao

Таким чином, RecipeDao інкапсулює всі необхідні операції з таблицею рецептів, а логіка застосунку працює вже з високорівневими методами, не формуючи SQL-запити вручну. Це зменшує кількість низькорівневого коду, знижує ризик помилок у запитах та дозволяє за потреби змінювати структуру таблиці, майже не торкаючись решти частин програми. Крім того, використання окремого DAO спрощує модульне тестування операцій із даними.

Над RecipeDao оголошується клас бази даних AppDatabase, який об’єднує всі сутності й надає єдиний доступ до DAO (рис. 2.11). Фактично він виступає центральною точкою налаштування локальної БД, де визначається список таблиць, версія схеми та правила міграції. У коді застосунку створення екземпляра AppDatabase виконується за допомогою бібліотеки Room, після чого метод recipeDao() використовується в репозиторії для збереження та отримання рецептів і слугує основним інтерфейсом роботи з локальними даними.

```
// Оголошення бази даних Room з єдиною сутністю RecipeEntity
@Database(
    entities = [RecipeEntity::class],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    // Метод для отримання DAO інтерфейсу для роботи з таблицею рецептів
    abstract fun recipeDao(): RecipeDao

    companion object {
        @Volatile
        // Посилання на єдиний екземпляр бази даних
        private var INSTANCE: AppDatabase? = null
        // Повертає існуючий екземпляр бази або створює новий, якщо його ще немає
        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "recipes.db"
                ).build().also { INSTANCE = it }  // зберігаємо створений екземпляр у INSTANCE
            }
        }
```

Рисунок 2.11. – Клас бази даних AppDatabase

Наступним важливим компонентом є мережевий модуль, що відповідає за отримання рецептів із зовнішнього сервісу Spoonacular. Для цього в пакеті data.remote оголошено інтерфейс SpoonacularApiService та допоміжний об’єкт SpoonacularApi, який налаштовує клієнт Retrofit і OkHttp.

В інтерфейсі SpoonacularApiService визначено три основні методи (рис. 2.12). Метод searchRecipes використовується на головному екрані для пошуку страв за ключовим словом або списком інгредієнтів. Поля number, addRecipeInformation та addRecipeNutrition дозволяють одразу отримувати обмежену кількість результатів разом із базовою інформацією та даними про харчову цінність. Метод getRandomRecipes дає можливість реалізувати функціонал «випадкового» підбору страв, а getRecipeInformation використовується на екрані деталей, коли потрібно завантажити розширені відомості про окремий рецепт.

```kotlin
interface SpoonacularApiService {
    // Пошук рецептів за текстовим запитом або інгредієнтами
    @GET("recipes/complexSearch")
    suspend fun searchRecipes(
        @Query("query") query: String? = null,
        @Query("includeIngredients") includeIngredients: String? = null,
        @Query("number") number: Int = 6,
        @Query("addRecipeInformation") addRecipeInformation: Boolean = true,
        @Query("sort") sort: String? = null,
        @Query("apiKey") apiKey: String = SPOONACULAR_API_KEY,
        @Query("addRecipeNutrition") addRecipeNutrition: Boolean = true,
    ): SpoonacularSearchResponse

    // Отримання випадкових рецептів
    @GET("recipes/random")
    suspend fun getRandomRecipes(
        @Query("number") number: Int,
        @Query("includeNutrition") includeNutrition: Boolean = true,
        @Query("apiKey") apiKey: String = SPOONACULAR_API_KEY
    ): RandomRecipesResponse

    // Отримання детальної інформації про конкретний рецепт
    @GET("recipes/{id}/information")
    suspend fun getRecipeInformation(
        @Path("id") id: Int,
        @Query("includeNutrition") includeNutrition: Boolean = true,
        @Query("apiKey") apiKey: String = SPOONACULAR_API_KEY
    ): SpoonacularApi.SpoonacularRecipeInfoDto
```

Рисунок 2.12. – Методи SpoonacularApiService

Для обробки відповіді сервісу визначено кілька моделей-DTO (рис. 2.13). У цих класах використовуються анотації @SerializedName, що дозволяють зіставити поля JSON-відповіді з полями Kotlin-класів. Не всі поля є обов’язковими, тому більшість із них описані як nullable (String?, Int?), що враховує можливі відсутні значення у відповіді. Клас RandomRecipesResponse представляє колекцію випадкових рецептів, SpoonacularSearchResponse відповідає за результат пошуку, а SpoonacularRecipeDto описує структуру окремого рецепта із зазначенням ідентифікатора, назви, посилання на зображення, часу приготування, типів страви та блоку з харчовою цінністю. Такий поділ на окремі DTO робить роботу з даними сервісу більш прозорою і спрощує подальше перетворення цих об’єктів у внутрішні моделі, що використовуються в логіці застосунку та в локальній базі даних.

```
// Колекція випадкових рецептів меню
data class RandomRecipesResponse(
    @SerializedName("recipes")
    val recipes: List<SpoonacularRecipeDto>?
)
// Результат пошуку рецептів
data class SpoonacularSearchResponse(
    @SerializedName("results") val results: List<SpoonacularRecipeDto>?
)

// Модель окремого рецепта у відповіді від Spoonacular API
data class SpoonacularRecipeDto(
    @SerializedName("id") val id: Int?,  // Ідентифікатор рецепта у зовнішньому сервісі
    @SerializedName("title") val title: String?,  // Назва страви у відповіді API
    @SerializedName("image") val image: String?,  // Повне посилання на зображення
    @SerializedName("readyInMinutes") val readyInMinutes: Int?,    // Час приготування у хвилинах
    @SerializedName("summary") val summary: String?,
    @SerializedName("dishTypes") val dishTypes: List<String>?,
    @SerializedName("nutrition") val nutrition: SpoonacularApi.NutritionDto?
)
```

Рисунок 2.13. – Класи для обробки даних

Більш деталізована інформація про рецепт (список інгредієнтів, покрокові інструкції, харчова цінність) описана у вкладених класах об’єкта SpoonacularApi (рис. 2.14).

```
data class SpoonacularRecipeInfoDto(
    @SerializedName("id") val id: Int?,
    @SerializedName("title") val title: String?,
    @SerializedName("image") val image: String?,
    @SerializedName("readyInMinutes") val readyInMinutes: Int?,
    @SerializedName("summary") val summary: String?,
    @SerializedName("extendedIngredients") val extendedIngredients: List<ExtendedIngredientDto>?,
    @SerializedName("analyzedInstructions") val analyzedInstructions: List<InstructionBlockDto>?,
    @SerializedName("dishTypes") val dishTypes: List<String>?,
    @SerializedName("nutrition") val nutrition: NutritionDto?
)

data class NutritionDto(
    @SerializedName("nutrients") val nutrients: List<NutrientDto>?
)
// Окремий нутрієнт (калорії)
data class NutrientDto(
    @SerializedName("name") val name: String?,
    @SerializedName("amount") val amount: Double?,
    @SerializedName("unit") val unit: String?
)
// Інгредієнт у текстовому вигляді
data class ExtendedIngredientDto(
    @SerializedName("original") val original: String?
)
```

Рисунок 2.14. – Класи об’єкта SpoonacularApi

Ці структури дозволяють отримати з API повний опис рецепта, включно з переліком інгредієнтів і кроків приготування, які згодом використовуються на екрані деталей.

Створення екземпляра клієнта Retrofit винесене в об’єкт SpoonacularApi (рис. 2.15). Там налаштовано базову URL-адресу сервісу, конвертер JSON (GsonConverterFactory) та HTTP-клієнт OkHttpClient із простим логуванням запитів.

```
object SpoonacularApi {
    val api: SpoonacularApiService by lazy {
        val logging = HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor
            level = HttpLoggingInterceptor.Level.BASIC  // Виводимо базову інформацію про запити
        }

        val client = OkHttpClient.Builder()
            .addInterceptor(logging)       // Додаємо логування до HTTP-клієнта
            .build()

        Retrofit.Builder() Retrofit.Builder
            .baseUrl(BASE_URL)  // Базова адреса сервісу Spoonacular
            .addConverterFactory(GsonConverterFactory.create()) // Конвертація JSON у Kotlin-об’єкти
            .client(client) Retrofit.Builder
            .build() Retrofit
            .create(SpoonacularApiService::class.java) ^lazy
    }
```

Рисунок 2.15. – Класи об’єкта SpoonacularApi

Використання by lazy забезпечує створення єдиного екземпляра клієнта при першому зверненні, що достатньо для мобільного застосунку. Наявність логера дає змогу зручно відстежувати виконувані запити під час налагодження.

Центральним елементом програмної реалізації є головний екран, який відповідає за пошук рецептів за інгредієнтами та початкове відображення результатів. Навігаційно він під’єднаний через MainActivity та RecipeNavHost: при старті застосунку створюється NavHostController, а початковим маршрутом визначено екран home, що відображає HomeScreen.

Головна сторінка реалізує три основні задачі:

- прийом та попередня обробка введених користувачем інгредієнтів;

- виконання мережевого запиту до Spoonacular API та формування внутрішньої моделі рецептів;
- відображення результатів пошуку з можливістю фільтрації за типом прийому їжі (сніданок, обід, вечеря, десерт).

Логіка пошуку інкапсульована в класі HomeViewModel, який зберігає стан екрана у вигляді HomeUiState (прапор завантаження, повідомлення про помилку, список рецептів) через MutableStateFlow.

Ключовим методом є searchRecipes, послідовність виконання якого подано на рис. 2.16-2.18.

```
class HomeViewModel : ViewModel() {

    private val _uiState = MutableStateFlow(HomeUiState())
    val uiState: StateFlow<HomeUiState> = _uiState

    fun searchRecipes(query: String) {
        // 1. Нормалізуємо введений рядок: ділимо за пробілами/комами,
        //    очищуємо зайві символи, залишаємо тільки "живі" слова
        val cleanedWords = query
            .lowercase() String
            .split(Regex("[,;\\s]+")) List<String>
            .map { it.trim() }
            .filter { it.isNotBlank() }
            .filter { word -> word.all { ch -> ch.isLetter() } }  // тільки букви

        // 2. Якщо після очищення немає слів - показуємо помилку і не викликаємо API
        if (cleanedWords.isEmpty()) {
            _uiState.update { it.copy(errorMessage = "Введіть інгредієнти!") }
            return
        }

        // 3. Формуємо параметр для API ("milk,egg,apple")
        val ingredientsParam = cleanedWords.joinToString(",")
        viewModelScope.launch { this: CoroutineScope
            _uiState.update { it.copy(isLoading = true, errorMessage = null) }
            try {
                // 4. Перекладаємо інгредієнти з української на англійську для API
                val englishQuery = TranslationHelper.translateUaToEn(ingredientsParam)
```

Рисунок 2.16. – Метод searchRecipes

```
// 5. Виконуємо пошук рецептів у Spoonacular
val response = api.searchRecipes(englishQuery)
val results = response.results.orEmpty()

// 6. Попередньо чистимо summary та рахуємо калорії
val baseList = results.map { dto ->
    val summaryClean = dto.summary
        ?.replace(Regex("<.*?>"), "")
        ?.trim()
        ?: ""
    val calories = dto.nutrition?.nutrients List<SpoonacularApi.NutrientDto>?
        ?.firstOrNull { it.name!!.contains("calories", ignoreCase = true) } SpoonacularApi.NutrientDto?
        ?.amount Double?
        ?.toInt() ?: 0
    RecipeUiModel(
        id = dto.id ?: 0,
        title = dto.title ?: "Без назви",
        description = summaryClean,
        timeMinutes = dto.readyInMinutes ?: 0,
        calories = calories,
        mealType = mapDishTypesToMealType(dto.dishTypes),
        imageUrl = dto.image
    ) ^map
}
```

Рисунок 2.17. – Метод searchRecipes

```
        // 7. Переклад назви та опису з англійської на українську
        val translated = baseList.map { recipe ->
            val titleUk = TranslationHelper.translateEnToUa(recipe.title)
            val descUk  = TranslationHelper.translateEnToUa(recipe.description)
            recipe.copy(
                title = titleUk,
                description = descUk
            ) ^map
        }

        // 8. Оновлюємо стан екрана: показуємо список рецептів
        _uiState.update { it: HomeUiState
            it.copy(
                isLoading = false,
                recipes = translated
            )
        }
    } catch (e: Exception) {
        // 9. У разі помилки – повідомлення для користувача
        _uiState.update { it: HomeUiState
            it.copy(
                isLoading = false,
                errorMessage = "Помилка завантаження рецептів"
            )
```

Рисунок 2.18. – Метод searchRecipes

Таким чином, головна сторінка не лише делегує роботу з API, а й виконує мінімальну «розумну» обробку введення, очищує текст, переводить інгредієнти на англійську, очищує HTML-розмітку в описах, витягує калорійність і автоматично класифікує страви за типом прийому їжі за допомогою допоміжної функції mapDishTypesToMealType (рис. 2.19).

```
fun mapDishTypesToMealType(dishTypes: List<String>?): MealType {
    val types = dishTypes?.map { it.lowercase() } ?: emptyList()

    return when {
        types.any { it.contains("breakfast") || it.contains("brunch") } ->
            MealType.BREAKFAST
        types.any { it.contains("lunch") || it.contains("main course") } ->
            MealType.LUNCH
        types.any { it.contains("dinner") || it.contains("supper") } ->
            MealType.DINNER
        types.any { it.contains("dessert") || it.contains("cake") } ->
            MealType.DESSERT
        else -> MealType.ALL
    }
}
```

Рисунок 2.19. – Функція mapDishTypesToMealType

Компонент HomeScreen отримує екземпляр HomeViewModel та підписується на зміни стану через collectAsState(). Уся верстка винесена в HomeScreenContent, куди передаються стан та обробники дій (рис. 2.20). Усередині цієї функції оголошуються локальні стани searchQuery та selectedFilter з використанням rememberSaveable, що дозволяє зберігати введений текст і обраний фільтр при повороті екрана. Додатково застосовується анімація масштабу іконки пошуку на основі rememberInfiniteTransition, яка активується під час завантаження даних і візуально підказує користувачеві, що запит виконується. Такий підхід поєднує керування станом, логіку обробки подій (натискання кнопки пошуку, вибір фільтра, відкриття рецепта) та відображення інтерфейсу в одному узгодженому компоненті, що робить код головного екрана компактним і зрозумілим.

```
@Composable
fun HomeScreenContent(
    state: HomeUiState,
    onSearch: (String) -> Unit,
    onRecipeClick: (Int) -> Unit
) {
    val focusManager = LocalFocusManager.current
    val keyboardController = LocalSoftwareKeyboardController.current

    val isLoading = state.isLoading
    var searchQuery by rememberSaveable { mutableStateOf("") }
    var selectedFilter by rememberSaveable { mutableStateOf(MealType.ALL) }

    // Анімація "пульсації" іконки пошуку, коли йде запит
    val infiniteTransition = rememberInfiniteTransition(label = "searchPulse")
    val animatedScale by infiniteTransition.animateFloat(
        initialValue = 1f,
        targetValue = 1.15f,
        animationSpec = infiniteRepeatable(
            animation = tween(durationMillis = 500, easing = FastOutSlowInEasing),
            repeatMode = RepeatMode.Reverse
```

Рисунок 2.20. – Функція HomeScreenContent

Нижче розташовані чіпи-фільтри типу прийому їжі (MealFilterChip), що дозволяють користувачеві обмежити список рецептів до сніданків, обідів, вечерь або десертів. Вибір фільтра не викликає новий запит до API, а лише відфільтровує вже завантажений список у пам’яті, як показано на рисунку 2.21.

```
Row(
    modifier = Modifier
        .fillMaxWidth()
        .horizontalScroll(rememberScrollState())
) { this: RowScope
    MealFilterChip(
        text = "Усе",
        isSelected = selectedFilter == MealType.ALL,
        onClick = { selectedFilter = MealType.ALL }
    )
    MealFilterChip(
        text = "Сніданок",
        isSelected = selectedFilter == MealType.BREAKFAST,
        onClick = { selectedFilter = MealType.BREAKFAST }
    )
    MealFilterChip(
        text = "Обід",
        isSelected = selectedFilter == MealType.LUNCH,
```

Рисунок 2.21. – Фільтрація рецептів

Коли state.isLoading == true, під фільтрами з'являється індикатор LinearProgressIndicator, що сигналізує про виконання запиту. У випадку помилки користувач бачить текстове повідомлення з червоним кольором.

Список знайдених рецептів відображається у вигляді послідовності карток RecipeCard. Попередньо колекція фільтрується за обраним типом прийому їжі. Картка рецепта містить зображення, назву, короткий опис, орієнтовний час приготування, калорійність та позначення типу страви.

Натискання на картку викликає onRecipeClick(recipe.id), який передається з MainActivity і здійснює навігацію до екрана деталей рецепта за маршрутом recipe_detail/{recipeId} (рис. 2.22).

```
val recipesToShow = state.recipes.filter { recipe ->
    selectedFilter == MealType.ALL || recipe.mealType == selectedFilter
}

recipesToShow.forEach { recipe ->
    RecipeCard(
        recipe = recipe,
        onClick = { onRecipeClick(recipe.id) } // перехід до екрана деталей
    )
    Spacer(modifier = Modifier.height(12.dp))
}
Spacer(modifier = Modifier.height(24.dp))
```

Рисунок 2.22. – Функція переходу на рецепт

Таким чином, головна сторінка поєднує алгоритм попередньої обробки введення та пошуку, логіку інтеграції з API та перекладу даних, а також інтерактивний інтерфейс з фільтрами та картками рецептів, забезпечуючи користувачеві зручну точку входу до функціоналу застосунку.

Далі важливим кроком програмної реалізації стала розробка екрана детальної інформації про рецепт. Його завдання – показати повний опис страви, інгредієнти, кроки приготування, базові характеристики (час, калорійність) та надати можливість додати або прибрати рецепт із «Обраного».

Логіка цього екрана зосереджена у класі RecipeDetailViewModel, який наслідує AndroidViewModel, оскільки йому потрібен доступ до контексту

додатка для створення екземпляра бази даних. У конструкторі ViewModel ініціалізується AppDatabase та RecipeDao, а також визначається стан екрана у вигляді StateFlow<RecipeDetailUiState>. Цей стан містить основні поля: назву страви, опис, інгредієнти, кроки приготування, калорійність, прапор «обране/не обране» та ознаку завантаження.

Ключовий метод ViewModel – це завантаження даних за ідентифікатором рецепта. Алгоритм працює у два кроки:

- спочатку ViewModel намагається отримати рецепт із локальної бази даних (якщо він уже був збережений і позначений як обраний);
- якщо в базі даних рецепт не знайдено, виконується запит до Spoonacular API для отримання детальної інформації, після чого дані нормалізуються (очищуються HTML-теги, формуються рядки інгредієнтів і кроків) та відображаються користувачеві.

Логіка додавання й видалення з обраного також реалізована в цій ViewModel. В окремому методі, умовно toggleFavorite(), спочатку визначається поточний стан прапора, потім або вставляється, або видаляється відповідний запис у локальній базі, як показано на рисунку 2.23.

```
// ------------ зберегти / прибрати з обраних ------------
fun toggleFavorite() {
    val id = currentRecipeId ?: return

    viewModelScope.launch { this: CoroutineScope
        val current = _uiState.value
        val isNowFavorite = !current.isFavorite  // Обчислюємо новий стан "обраного"

        if (isNowFavorite) {
            // Якщо тепер рецепт має бути в обраному -
            // створюємо сутність для БД з поточного стану
            val entity = current.copy(isFavorite = true)
                .toEntity(remoteId = id, isCustom = false)

            recipeDao.insert(entity) // зберігаємо в БД
        } else {
            // Якщо навпаки - прибираємо рецепт з обраного
            recipeDao.deleteByRemoteId(id)
        }  // Оновлюємо UI-стан, щоб кнопка/іконка змінила вигляд
        _uiState.update { it.copy(isFavorite = isNowFavorite) }
    }
```

Рисунок 2.23. – Функція toggleFavorite

Це дозволяє користувачеві однією дією керувати статусом рецепта, а застосунок автоматично синхронізує стан екрана з локальними даними.

Відповідний екран інтерфейсу реалізовано у файлі RecipeDetailScreen.kt у вигляді окремого @Composable. На вхід він отримує recipeId та екземпляр RecipeDetailViewModel, з якого через collectAsState() зчитується поточний стан. Верхня частина екрана містить зображення страви та назву, під ними блок із короткою інформацією (час приготування, калорії, тип страви). Нижче розташовано окремі секції:

- «Інгредієнти» – виводиться сформований список інгредієнтів у зручному текстовому вигляді;
- «Кроки приготування» – складається з послідовних кроків, сформованих на основі даних InstructionStepDto з API.

У правій верхній частині розміщено кнопку-іконку «серце» (Icons.Filled.Favorite), яка змінює заповненість залежно від поля isFavorite у стані. Натискання на цю іконку викликає метод toggleFavorite(), а інтерфейс миттєво відображає зміну, що робить взаємодію зручною та очевидною для користувача.

Наступним важливим компонентом є розділ «Обрані рецепти», який дозволяє користувачеві швидко повернутися до страв, що його зацікавили раніше. Тут головну роль відіграють два файли: FavoritesViewModel.kt та FavoritesScreen.kt.

У FavoritesViewModel описано стан FavoritesUiState, який складається з двох основних полів: isLoading та recipes (список об’єктів RecipeUiModel). На відміну від головного екрана, де дані беруться з мережі, у цьому випадку всі рецепти надходять з локальної бази даних через RecipeDao. У конструкторі ViewModel запускається корутина, яка підписується на потік обраних рецептів, як показано на рисунку 2.24. Список обраних у ViewModel автоматично оновлюється при будь-яких змінах у базі даних (додавання/видалення улюблених рецептів на інших екранах). Це спрощує логіку інтерфейсу – йому достатньо просто відображати поточний стан.

```
data class FavoritesUiState(
    val isLoading: Boolean = true,
    val recipes: List<RecipeUiModel> = emptyList()
)

class FavoritesViewModel(
    application: Application
) : AndroidViewModel(application) {

    private val recipeDao = AppDatabase.getInstance(application).recipeDao()
    private val _uiState = MutableStateFlow(FavoritesUiState())
    val uiState: StateFlow<FavoritesUiState> = _uiState

    init {
        viewModelScope.launch { this: CoroutineScope // Підписуємося на потік улюблених рецептів з БД
            recipeDao.getFavoriteRecipes().collect { list ->
                _uiState.update { it: FavoritesUiState
                    it.copy(
                        isLoading = false,
                        recipes = list.map { entity -> entity.toRecipeUiModel() }
                    )
```

Рисунок 2.24. – Клас FavoritesViewModel

Екран обраних рецептів (FavoritesScreen) працює за тим самим принципом, що й головна сторінка. У верхній частині зчитується стан ViewModel, далі в залежності від нього відображається один із варіантів:

- якщо isLoading == true, показується простий індикатор завантаження;
- якщо список recipes порожній, користувач бачить повідомлення про відсутність обраних рецептів;
- якщо в recipes є елементи – виводиться LazyColumn із картками RecipeCard, аналогічними тим, що використовуються на головному екрані.

Фрагмент композиції для відображення непорожнього списку подано на рисунку 2.25.

Таким чином, розділ «Обране» фактично є «вітриною» локальної бази даних застосунку. Він демонструє роботу з потоками Room (через Flow/collect), автоматичне оновлення інтерфейсу при зміні даних та повторне використання тих самих UI-компонентів (RecipeCard), що й на головному екрані. Для користувача це виглядає як простий і зрозумілий список «улюблених» рецептів, з яких за потреби можна миттєво перейти до повного опису страви.

```
fun FavoritesScreen(
    onRecipeClick: (Int) -> Unit
) {
    val viewModel: FavoritesViewModel = viewModel()
    val state by viewModel.uiState.collectAsState()

    when {
        state.isLoading -> {      // Показуємо, що дані завантажуються
            Box(
                modifier = Modifier.fillMaxSize(),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "Завантаження...",
                    style = MaterialTheme.typography.bodyLarge
                )
            }
        }

        state.recipes.isEmpty() -> {
            // Повідомлення, якщо користувач ще нічого не додав до обраного
            Box(
                modifier = Modifier.fillMaxSize(),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "Немає обраних рецептів",
                    style = MaterialTheme.typography.bodyLarge
                )
```

Рисунок 2.25. – Функція FavoritesScreen

Ще одним важливим компонентом застосунку є екран формування меню, який дозволяє користувачеві одним натисканням отримати набір страв на день, зокрема сніданок, обід, вечерю та десерт. Тут поєднуються мережеві запити до Spoonacular, простий алгоритм відбору страв за типом прийому їжі та збереження результату, щоб меню не змінювалося щоразу при відкритті екрана.

За логіку відповідає MenuViewModel (наслідує AndroidViewModel). Він зберігає стан у вигляді MenuUiState, який включає:

- поточну дату (рядок для відображення типу «Меню на сьогодні»);
- список рецептів для меню (3-4 страви у вигляді RecipeUiModel);
- прапори isLoading та текст помилки errorMessage.

У ViewModel використовуються:

- SpoonacularApi – для отримання списку випадкових рецептів (getRandomRecipes(number = …));
- Gson – для серіалізації/десеріалізації списку рецептів, щоб зберігати меню в SharedPreferences;
- TranslationHelper – для перекладу назв та описів з англійської на українську.

Загальна ідея алгоритму така:

- при відкритті екрана ViewModel намагається прочитати з SharedPreferences останнє збережене меню;
- якщо дата збігається з поточним днем – меню просто відновлюється в uiState, без нових запитів до API;
- якщо збереженого меню немає або воно не актуальне – користувач натискає кнопку «Згенерувати меню», після чого робиться запит до Spoonacular, результати обробляються й зберігаються.

Ключовий крок – розподіл знайдених рецептів за типами прийому їжі. Для цього використовується окремий допоміжний метод у файлі MealTypeMapper.kt, як показано на рисунку 2.26.

```
fun mapDishTypesToMealType(dishTypes: List<String>?): MealType {
    val types = dishTypes?.map { it.lowercase() } ?: emptyList()

    return when {
        types.any { it.contains("breakfast") || it.contains("brunch") } ->
            MealType.BREAKFAST
        types.any { it.contains("lunch") || it.contains("main course") } ->
            MealType.LUNCH
        types.any { it.contains("dinner") || it.contains("supper") } ->
            MealType.DINNER
        types.any { it.contains("dessert") || it.contains("cake") } ->
            MealType.DESSERT
        else -> MealType.ALL
    }
```

Рисунок 2.26. – Функція розподілу рецептів

Ця функція аналізує список «dishTypes» із API (наприклад, [main course, dinner]) і повертає одне з перерахованих значень MealType. На основі цього

ViewModel групує усі отримані SpoonacularRecipeDto та намагається вибрати по одній страві для кожної категорії (сніданок, обід, вечеря, десерт). Якщо для якогось типу нічого не знайшлося, використовуються резервні варіанти з інших категорій, щоб меню все одно було повним.

Далі алгоритм схожий на те, що використовується на головному екрані:

- очищується summary від HTML-тегів;
- із блоку nutrition витягується калорійність (перший нутрієнт із назвою, що містить «calories»);
- будується RecipeUiModel з полями id, title, description, timeMinutes, calories, mealType, imageUrl;
- назва та опис перекладаються на українську за допомогою TranslationHelper.

В результаті ViewModel отримує невеликий список рецептів (наприклад, 4 страви: сніданок, обід, вечеря, десерт), зберігає його у MenuUiState і паралельно серіалізує через Gson у JSON і записує в SharedPreferences разом із датою. Наступного разу, коли користувач відкриє екран у той самий день, програма просто прочитає це меню з локального сховища й відобразить без повторних мережевих запитів.

У разі будь-якої помилки (проблеми з мережею, некоректна відповідь API) ViewModel оновлює стан із errorMessage = «Не вдалося згенерувати меню» і скидає прапор isLoading, щоб користувач міг спробувати ще раз.

Інтерфейс екрану меню реалізовано у файлі MenuScreen.kt. Він отримує MenuViewModel та функцію onRecipeClick, яка використовується для переходу на екран деталей рецепта, як показано на рисунку 2.27. Як і на інших екранах, стан ViewModel зчитується через:

val state by viewModel.uiState.collectAsState()

```
fun MenuScreen(
    onRecipeClick: (Int) -> Unit
) {
    val viewModel: MenuViewModel = viewModel()
    val state by viewModel.uiState.collectAsState()

    val focusManager = LocalFocusManager.current
    val keyboardController = LocalSoftwareKeyboardController.current

    LaunchedEffect(Unit) { this: CoroutineScope
        focusManager.clearFocus(force = true)
        keyboardController?.hide()
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(horizontal = 16.dp, vertical = 8.dp)
    ) { this: ColumnScope
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) { this: RowScope
            Text(
                text = "Меню на сьогодні",
                style = MaterialTheme.typography.titleLarge
            )
```

Рисунок 2.27. – Функція MenuScreen

Структурно екран складається з трьох зон:

- верхній блок із заголовком («Меню на сьогодні»), поточною датою та коротким поясненням (наприклад, що меню складається з чотирьох страв різних типів);
- кнопка для генерації меню (умовно «Згенерувати меню»), яка викликає метод ViewModel для побудови нового набору страв і блокується під час завантаження (enabled = !state.isLoading);
- основна область вмісту, що містить або повідомлення про те, що меню ще не згенеровано, або список страв, які входять до меню.

Коли в state.recipes є елементи, вони відображаються у LazyColumn як набір карток RecipeCard, таких самих, як на головному екрані та в обраному.

Якщо state.isLoading == true, замість списку або над ним показується індикатор завантаження, щоб користувач розумів, що зараз іде формування меню. У випадку, коли state.errorMessage не порожнє, під кнопкою генерації виводиться коротке повідомлення про помилку (наприклад, проблеми з мережею).

Таким чином, екран меню демонструє:

- використання ViewModel для поєднання мережевої логіки, локального кешування в SharedPreferences та класифікації страв за типом прийому їжі;
- спрощений, але зрозумілий для користувача алгоритм побудови меню на день;
- повторне використання вже наявних UI-компонентів (RecipeCard) та єдиного підходу до керування станом через StateFlow і Jetpack Compose.

У сукупності з головною сторінкою та розділом «Обране» це завершує реалізацію основного функціоналу застосунку, до якого належать пошуку рецептів, детального перегляду, збереження улюблених страв і автоматичного формування меню на день.

2.6 Організація тестування та налагодження програмного засобу

Тестування мобільного застосунку проводилося з метою перевірки коректності роботи основних сценаріїв, стійкості до типових помилок та адекватної реакції на збої мережі. Тестування здійснювалося в середовищі Android Studio з використанням вбудованого емулятора та реального Android-пристрою. Для налагодження застосовувалися журнали подій (Logcat), точкові зупинки, а також HTTP-логування запитів до Spoonacular API за допомогою HttpLoggingInterceptor.

Основним був підхід функціонального (сценарного) тестування, коли програма перевірялася «очима користувача» [27].

Окремо тестувалися:

- головна сторінка (введення інгредієнтів, пошук рецептів, обробка порожнього та некоректного введення, відображення станів «завантаження» та «помилка»);
- перехід до детального перегляду рецепта, відображення повної інформації (інгредієнти, кроки приготування, час, калорійність);
- додавання/видалення рецепта з розділу «Обране» та перевірка збереження даних після перезапуску застосунку;
- формування меню на день, відсутність дублювання страв у меню та можливість переходу до деталей кожної страви;
- поведінка при відсутності інтернет-з’єднання, коли перевіряється наявність повідомлення про помилку та доступ до раніше збережених рецептів із локальної бази даних.

Критеріями успішності тестування були відсутність аварійного завершення програми, коректне відображення інформації на екранах, зрозумілі повідомлення про помилки та достатня швидкодія під час виконання основних операцій (пошук, відкриття рецепта, робота з обраним і меню).

Також у проєкті були реалізовані модульні тести на базі бібліотеки JUnit. Вони охоплювали передусім логіку, що не залежить від інтерфейсу:

- функції перетворення даних (mapper-и між моделями API та сутностями локальної бази), щоб перевірити правильність копіювання полів і обробки відсутніх значень;
- функцію визначення типу прийому їжі за списком dishTypes (клас MealTypeMapper), щоб переконатися, що страви коректно класифікуються як сніданок, обід, вечеря або десерт;

Приклади результатів виконання модульних тестів наведено в додатку Г (рис. Г.1 – Г.2).

Для частини логіки ViewModel-ів виконувалися спрощені тести із використанням фіктивних даних (наприклад, перевірка того, що при порожньому запиті на головній сторінці формується повідомлення про помилку,

а при успішному пошуку – змінюється прапор завантаження та список рецептів не є порожнім).

У процесі тестування було виявлено й усунуто кілька типових недоліків: некоректна обробка null-полів у відповіді API (призводила до помилок при відображенні), можливість багаторазового запуску пошуку при повторному натисканні на кнопку під час завантаження, а також відсутність зрозумілого повідомлення користувачеві при проблемах із мережею. Після внесення змін у код було додано перевірки на null, блокування кнопки пошуку під час виконання запиту, кешування згенерованого меню на день та відображення явного текстового повідомлення про помилку.

У цілому результати тестування показали, що застосунок коректно виконує свої основні функції, зокрема здійснює пошук рецептів за інгредієнтами, відображає детальну інформацію про страви, дозволяє працювати з «обраним» та формувати меню на день. Рекомендаціями для подальшого розвитку є розширення набору модульних тестів (зокрема, для більшої частини логіки ViewModel-ів) та впровадження автоматизованих UI-тестів для перевірки інтерфейсу в напівавтоматичному режимі.

2.7 Аналіз отриманих результатів дослідження, рекомендації щодо використання та впровадження

У результаті виконання роботи розроблено мобільний застосунок для Android, який дозволяє виконувати пошук рецептів за інгредієнтами, переглядати детальну інформацію про страви, зберігати їх у «Обране» та автоматично формувати меню на день. Застосунок поєднує роботу із зовнішнім API, локальне збереження даних і зручний інтерфейс на основі Jetpack Compose, що підтверджує досягнення поставленої мети та реалізацію основних завдань.

Для оцінки роботи програмного забезпечення було проведено прості вимірювання часу відгуку та споживання ресурсів на емуляторі та реальному смартфоні середнього класу. За стабільного підключення до Інтернету середній

час пошуку рецептів (від натискання кнопки «Пошук» до появи результатів) склав приблизно 2-4 секунди для 6-8 рецептів, а генерація меню на день близько 2-3 секунд. Такі значення є прийнятними для застосунку, який активно звертається до зовнішнього вебсервісу та додатково обробляє дані (очищення описів, переклад тексту тощо. Профілювання в Android Studio показало помірне споживання оперативної пам’яті й короткочасні піки завантаження процесора під час виконання мережевих запитів та зчитування даних із бази. Ці результати можна подати у вигляді простої стовпчикової діаграми (наприклад, порівняння часу пошуку та часу генерації меню), щоб наочно показати прийнятний рівень затримок.

Разом із позитивними результатами виявлено й обмеження розробленого рішення. Основне з них – залежність від якості інтернет-з’єднання та доступності Spoonacular API: без мережі нові рецепти й меню отримати неможливо, а користувач може працювати лише з уже збереженими даними. Також загальна швидкодія частково залежить від продуктивності конкретного пристрою, на якому запускається програма. Перспективним напрямом удосконалення є оптимізація роботи з перекладом і можливе часткове кешування перекладених описів, що зменшить час очікування користувача.

З практичної точки зору розроблений застосунок може використовуватися як щоденний помічник при плануванні харчування. Він спрощує пошук ідей страв, допомагає систематизувати рецепти та дозволяє швидко сформувати базове меню на день. Завдяки модульній архітектурі та чіткому поділу на шари даних, логіки та інтерфейсу застосунок може бути розширений у майбутньому, наприклад, за рахунок додавання дієтичних фільтрів, розширеної статистики харчування чи інтеграції з іншими сервісами.

Для коректної роботи програмного засобу достатньо стандартного сучасного Android-смартфона. Орієнтовні вимоги такі:

- операційна система Android не нижче версії 8.0;
- наявність підключення до Інтернету для пошуку рецептів і формування меню;

- кілька десятків мегабайт вільної пам’яті для встановлення застосунку та зберігання локальних даних;
- типова для сучасних пристроїв оперативна пам’ять (від 3 ГБ), що забезпечує плавну роботу інтерфейсу.

Отже, отримані результати свідчать, що розроблений мобільний застосунок є працездатним, демонструє прийнятну продуктивність у типових умовах використання й може бути впроваджений як практичний інструмент для підбору рецептів і формування меню з подальшою можливістю розвитку функціоналу.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано задачу проєктування та розробки мобільного застосунку для підбору рецептів і формування меню, що працює на платформі Android та використовує зовнішній програмний сервіс як джерело даних. Сформульована у вступі мета – створення прикладного програмного засобу, який поєднує функції пошуку, збереження рецептів та побудови меню на день – досягнута, а поставлені завдання послідовно реалізовані.

У теоретичній частині виконано аналіз предметної області організації харчування, розглянуто роль інформаційних технологій та можливості рекомендаційних систем у цьому контексті. Описано базові підходи до побудови рекомендаційних рішень, моделі та алгоритми підбору рецептів і меню. Проведено огляд і порівняльний аналіз існуючих програмних продуктів для роботи з рецептами, на підставі якого сформульовано вимоги до функціоналу й інтерфейсу власного застосунку. Це забезпечило цілісне розуміння задачі та обґрунтування вибраних проєктних рішень.

У практичній частині роботи розроблено архітектуру мобільного застосунку з чітким поділом на шари даних, логіки та інтерфейсу. Реалізовано роботу із зовнішнім API для пошуку та отримання детальної інформації про рецепти, проєктування та використання локальної бази даних для збереження обраних страв, а також екранну структуру застосунку (головна сторінка пошуку, перегляд рецепта, розділ «Обране», екран формування меню). Використання сучасних технологій Android (Kotlin, Jetpack Compose, Room, Retrofit) дало змогу побудувати гнучкий і розширюваний програмний продукт, який відповідає актуальним підходам до мобільної розробки.

Проведено тестування та налагодження програмного засобу. Перевірено основні користувацькі сценарії, враховано типові помилки введення, ситуації з відсутністю мережі та неповними відповідями від API. Реалізовано мінімальний набір модульних тестів для ключових функцій перетворення даних і допоміжної логіки. Отримані результати свідчать про працездатність застосунку,

прийнятний час відгуку та коректну роботу основних функцій у типових умовах експлуатації.

Разом з тим виявлено низку обмежень. Застосунок залежить від стабільності інтернет-з’єднання та доступності зовнішнього сервісу рецептів, а швидкодія частково прив’язана до продуктивності мобільного пристрою. Поточна логіка рекомендацій та формування меню є відносно простою й не враховує, наприклад, індивідуальні дієтичні обмеження, історію вибору страв чи довгострокове планування раціону. Ці недоліки не знижують практичної корисності розробки, але окреслюють можливі шляхи її удосконалення.

Подальший розвиток доцільно спрямувати на:

- ускладнення рекомендаційної складової (урахування дієт, алергій, історії вибору страв);
- розширення можливостей планування (меню на тиждень, згенеровані списки покупок);
- поглиблення тестування (розширений набір модульних та автоматизованих UI-тестів);
- адаптацію застосунку до інших платформ або інтеграцію з додатковими сервісами.

Загалом результати роботи мають як теоретичне, так і прикладне значення, оскільки узагальнюють підходи до побудови мобільних застосунків рекомендаційного типу та демонструють практичну реалізацію програмного засобу, який може використовуватися як основа для подальших досліджень і реальних користувацьких рішень у сфері організації харчування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Науковий альманах «Crosspoint», ISBN 978-617-7886-81-4, С. 254-256, DOI: 10.61718/cros2025. URL: https://www.newroute.org.ua/wp-content/uploads/cros_2025.pdf (дата звернення: 10.12.2025).
2. Інформаційно-аналітичні системи :: megalib.com.ua. *Електронна бібліотека онлайн MegaLib.com.ua*. URL: http://megalib.com.ua/content/1960_75_Informaciino_analitichni_sistemi.html (дата звернення: 02.11.2025).
3. Фудтех України: тренди, інвестори, новинки, стартапи, компанії. URL: https://newfood.ua/2022/02/06/fudtekh-ukrainy-2022-trendy-investory-novynky-startapy-kompanii (дата звернення: 02.11.2025).
4. Food waste. slowfood UA. *Slowfood UA*. URL: https://slowfood.in.ua/global-projects/food-waste (дата звернення: 02.11.2025).
5. SKALAR | Рекомендаційні системи. *SKALAR GLOBAL | More than a WEB development...* URL: https://skalar.ua/ua/expertise/recommender-systems (дата звернення: 02.11.2025).
6. Область застосування та види рекомендаційних систем. *Конференції Державного університету «Житомирська політехніка»*. URL: https://conf.ztu.edu.ua/wp-content/uploads/2021/01/41-2.pdf (дата звернення: 03.11.2025).
7. Моделі роботи рекомендаційних систем. *Конференції Державного університету «Житомирська політехніка»*. URL: https://conf.ztu.edu.ua/wp-content/uploads/2017/11/23.pdf (дата звернення: 03.11.2025).
8. Що таке колаборативна фільтрація - терміни та визначення в кібербезпеці. *VPN service for Security and Speed*. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/collaborative-filtering?srsltid=AfmBOoqaHlSDHoxBQ2hVd3D0xXWM6sYuolghmiw_fqU4znqe0bujNrmr (дата звернення: 04.11.2025).

9. What are recommender systems? Use cases, types, and techniques. *Coralogix*. URL: https://coralogix.com/ai-blog/what-are-recommender-systems-use-cases-types-and-techniques/ (дата звернення: 03.11.2025).
10. Intelligent food planning. *About - Shlomo Berkovsky*. URL: https://shlomo-berkovsky.github.io/files/pdf/IUI10b.pdf (дата звернення: 04.11.2025).
11. Konstantinos Zioutos. Article personalized recipe recommendations. URL: https://www.mdpi.com/2073-431X/13/1/1 (дата звернення: 04.11.2025).
12. Limited T. Recipe keeper - apps on google play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.tudorspan.recipekeeper (дата звернення: 08.11.2025).
13. BuzzFeed. Tasty - apps on google play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.buzzfeed.tasty (дата звернення: 08.11.2025).
14. Inc M. M. P. Mealime meal plans & recipes - apps on google play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.mealime (дата звернення: 08.11.2025).
15. All about mealime pro (how to upgrade, features) - mealime support docs. *Mealime Support Docs*. URL: https://support.mealime.com/article/79-mealime-pro (дата звернення: 08.11.2025).
16. Технології розроблення програмного забезпечення. *DSpace :: ELAKPI :: Репозитарій КПІ ім. Ігоря Сікорського*. URL: https://ela.kpi.ua/server/api/core/bitstreams/9521e5f9-421a-4874-a17d-f0853f942856/content (дата звернення: 09.11.2025).
17. Розробка ПЗ: моделі життєвого циклу, методи та принципи. *Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів*. URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html (дата звернення: 09.11.2025).

18. Моделі життєвого циклу. *ProgIngContrSystems*. URL: https://pupenasan.github.io/ProgIngContrSystems/Лекц/17_lyfecycle.html (дата звернення: 09.11.2025).
19. Просто про архітектуру програмного забезпечення або основи, які заощадять місяці роботи. *IT-компанія з розробки цифрових продуктів в Україні – Brander*. URL: https://brander.ua/blog/prosto-pro-arkhitekturu-prohramnoho-zabezpechennya (дата звернення: 11.11.2025).
20. Guide to app architecture | App architecture | Android Developers. *Android Developers*. URL: https://developer.android.com/topic/architecture (дата звернення: 11.11.2025).
21. MVVM (model view viewmodel) architecture pattern in android - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/android/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (дата звернення: 11.11.2025).
22. Get started with jetpack compose | android developers. *Android Developers*. URL: https://developer.android.com/develop/ui/compose/documentation (дата звернення: 15.11.2025).
23. Meet android studio | android developers. *Android Developers*. URL: https://developer.android.com/studio/intro (дата звернення: 16.11.2025).
24. Kotlin and android | android developers. *Android Developers*. URL: https://developer.android.com/kotlin (дата звернення: 16.11.2025).
25. Використання jetpack compose у сучасній android-розробці. URL: https://careers.epam.ua/blog/jetpack-compose-in-android-development (дата звернення: 16.11.2025).
26. Save data in a local database using Room | App data and files | Android Developers. *Android Developers*. URL: https://developer.android.com/training/data-storage/room (дата звернення: 16.11.2025).
27. Spoonacular recipe and food API. *Free meal planner, food tracker, and recipe saver*. URL: https://spoonacular.com/food-api (дата звернення: 16.11.2025).

ДОДАТКИ

Додаток А. Технічне завдання

Вступ

Мобільні пристрої стали основним інструментом для вирішення повсякденних задач, у тому числі пов’язаних з організацією харчування. Користувачі часто шукають рецепти, намагаються раціонально використати наявні інгредієнти та планувати меню. Мобільний застосунок «Cookify» призначений для спрощення цих процесів на платформі Android.

Підстави для розробки

Розробка виконується в межах кваліфікаційної роботи здобувача ступеня магістра спеціальності «Комп’ютерні науки» Волинського національного університету імені Лесі Українки.

Підставами є:

- навчальний план та програма підготовки магістрів;
- завдання на кваліфікаційну роботу, затверджене кафедрою;
- тема роботи «Проєктування та розробка мобільного застосунку для підбору рецептів і меню».

Призначення розробки

Функціональне призначення – надання користувачеві інструменту для пошуку рецептів за інгредієнтами, перегляду детальної інформації про страви, збереження улюблених рецептів і автоматичного формування меню на день.

Експлуатаційне призначення – використання застосунку на персональних смартфонах під керуванням Android у побутових умовах одним користувачем без спеціальної підготовки.

Вимоги до програми чи програмного продукту

Вимоги до функціональних характеристик

Програма повинна забезпечувати:

- введення текстового запиту з інгредієнтами та отримання списку відповідних рецептів із зовнішнього API;

- відображення для рецепта назви, зображення, короткого опису, часу приготування та калорійності;
- перегляд детальної інформації про рецепт (інгредієнти, кроки приготування, додаткові характеристики);
- додавання й вилучення рецептів зі списку «Обране» з локальним збереженням даних;
- формування меню на день із кількох страв різних типів (сніданок, обід, вечеря, десерт) та збереження меню на поточну дату;
- показ повідомлень про помилки (порожній запит, відсутність результатів, проблеми з мережею) та індикацію процесів завантаження;

Вимоги до надійності

Мобільний застосунок повинен відповідати призначенню та стабільно працювати.

Умови експлуатації

Експлуатація передбачена в умовах звичайного використання смартфона. Необхідні:

- доступ до мережі Інтернет для пошуку рецептів і формування меню;

Спеціальне обслуговування не потрібне. Щоб користуватися мобільним застосунком, на мобільний пристрій потрібно встановити apk-файл та запустити його.

Вимоги до складу і параметрів технічних засобів

Необхідний мобільний пристрій з операційною системою Android версії 8.0 або вище, процесором з тактовою частотою не менше 1.5 ГГц, оперативною пам'яттю не менше 2 ГБ, наявність мобільного Інтернету або Wi-Fi та не менше 100 МБ вільної постійної пам’яті.

Вимоги до інформаційної і програмної сумісності

Застосунок повинен:

- працювати з API, що використовує формат JSON і протокол HTTPS;
- бути сумісним із підтримуваними версіями Android SDK;

- коректно взаємодіяти зі стандартними службами ОС (мережа, файлове сховище, система дозволів).

Спеціальні вимоги до захисту персональних даних мінімальні, оскільки застосунок не використовує облікові записи користувача.

Вимоги до маркування і упаковки

Маркування додатку повинно включати назву програми та логотип. Упаковка – цифрова, у вигляді apk-файлу.

Вимоги до транспортування і збереження

Вимоги до транспортування і збереження відсутні.

Вимоги до програмної документації

Документація повинна включати:

- технічне завдання;
- інструкція користувачу.

Техніко-економічні показники

Використання мобільного застосунку не передбачає з боку користувача будь-яких витратних операцій.

Стадії і етапи розробки

Розробка включає такі основні стадії:

- аналіз предметної області та вимог, огляд існуючих рішень;
- проєктування архітектури, моделі даних, структури екранів і навігації;
- реалізація логіки роботи з API, базою даних і інтерфейсом користувача;
- тестування, налагодження й оптимізація;
- підготовка та оформлення документації, підготовка до захисту.

Порядок контролю і приймання

Прийняття роботи включає функціональні випробування, тестування на стабільність та відповідність вимогам технічного завдання. Програма вважається прийнятою після успішного проходження всіх видів випробувань.

Додаток Б. Інструкція користувачу

Загальні відомості

Позначення програмного засобу: мобільний застосунок «Cookify». Призначення: підбір кулінарних рецептів за інгредієнтами, збереження обраних страв та автоматичне формування меню на день на мобільних пристроях з ОС Android.

Функціональне призначення

Застосунок «Cookify» призначений для індивідуального використання користувачами, які планують своє харчування та хочуть швидко знаходити ідеї страв із наявних продуктів.

За допомогою програми можна:

- виконувати пошук рецептів за введеними інгредієнтами;
- переглядати детальну інформацію про страву (інгредієнти, кроки приготування, час, калорійність);
- додавати рецепти до списку «Обране» і переглядати їх окремо;
- автоматично формувати меню на день (сніданок, обід, вечеря, десерт).

Умови застосування програми

Для коректної роботи програми необхідні такі умови:

- Тип пристрою: смартфон або планшет на базі ОС Android.
- Операційна система: Android версії 8.0 або новішої.
- Оперативна пам’ять: не менше 2 ГБ (рекомендовано для комфортної роботи).
- Постійна пам’ять: не менше 100 МБ вільного місця для встановлення та збереження локальних даних.
- Підключення до мережі: стабільний доступ до Інтернету (мобільна мережа або Wi-Fi) для пошуку рецептів та формування меню.

Встановлення та оновлення застосунку виконується стандартними засобами Android (через інсталяційний Apk-файл).

Повідомлення оператору

У процесі роботи програма може виводити службові повідомлення. Нижче наведені типові повідомлення та рекомендовані дії користувача.

- «Введіть інгредієнти для пошуку»

  Зміст: користувач натиснув кнопку пошуку, не ввівши жодного слова.

  Дія: ввести хоча б один інгредієнт (наприклад, «курка», «рис», «яблуко») і повторити пошук.

- «Не вдалося завантажити рецепти. Перевірте підключення до Інтернету.»

  Зміст: програма не змогла отримати дані з серверу (відсутнє підключення до мережі або тимчасова помилка сервісу).

  Дія: перевірити підключення до Інтернету (активність Wi-Fi або мобільних даних), за потреби спробувати повторити пошук пізніше.

- «Меню ще не згенеровано. Натисніть кнопку, щоб отримати пропозицію на день.»

  Зміст: екран меню відкрито вперше або меню на поточний день ще не створено.

  Дія: натиснути кнопку генерації меню, після чого програма підбере страви на день.

- «Список обраних порожній»

  Зміст: користувач ще не додав жодного рецепта до «Обраного».

  Дія: перейти на головну сторінку, знайти цікаві рецепти і додати їх до обраних через кнопку/іконку на екрані рецепта.

- «Сталася помилка. Спробуйте ще раз»

  Зміст: внутрішня помилка або нештатна ситуація.

  Дія: повторити дію (пошук, відкриття рецепта, генерацію меню). Якщо помилка повторюється – закрити та знову відкрити застосунок, перевірити Інтернет.

Опис роботи програми

Робота з програмою складається з кількох основних кроків.

Запуск програмного засобу

Користувач знаходить іконку «Cookify» на екрані пристрою та натискає на неї (Рис. Б.1). Після запуску відкривається головний екран (Рис. Б.2).

Рисунок Б.1. – Іконка застосунку

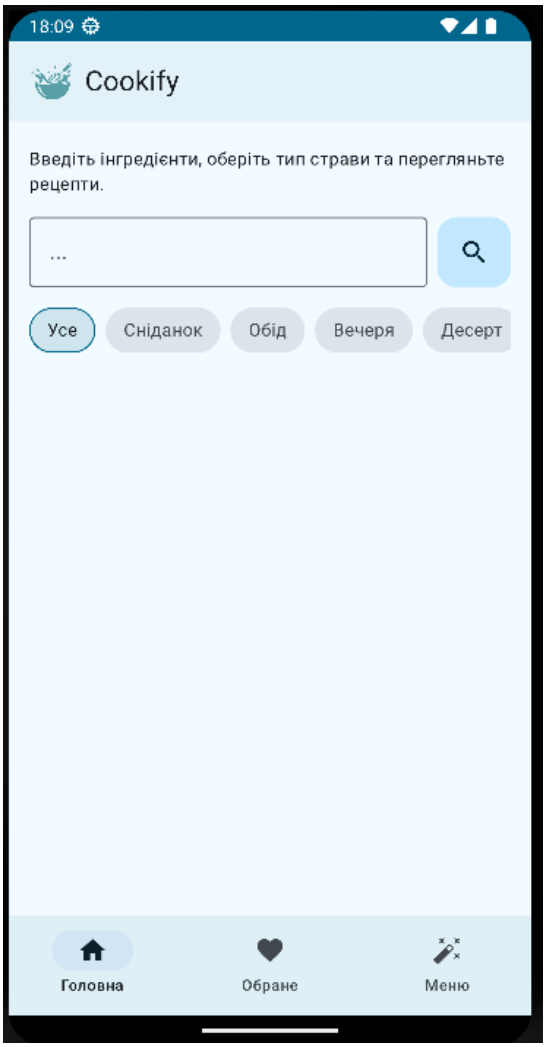

Рисунок Б.2. – Головний екран

Головний екран

У верхній частині розміщено поле введення інгредієнтів та кнопка «Пошук». Користувач вводить один або кілька інгредієнтів (через пробіл або кому) і натискає кнопку. Програма надсилає запит до сервісу рецептів, отримує результати та відображає список страв у вигляді карток із назвою, зображенням, коротким описом, часом приготування та калорійністю. За допомогою фільтрів користувач може обмежити список типом прийому їжі. (Рис. Б.3-Б.4).

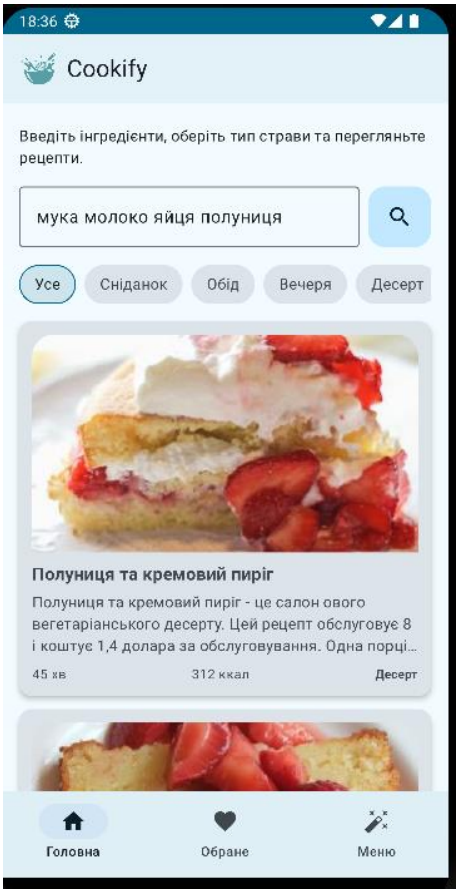

Рисунок Б.3. – Пошук рецептів

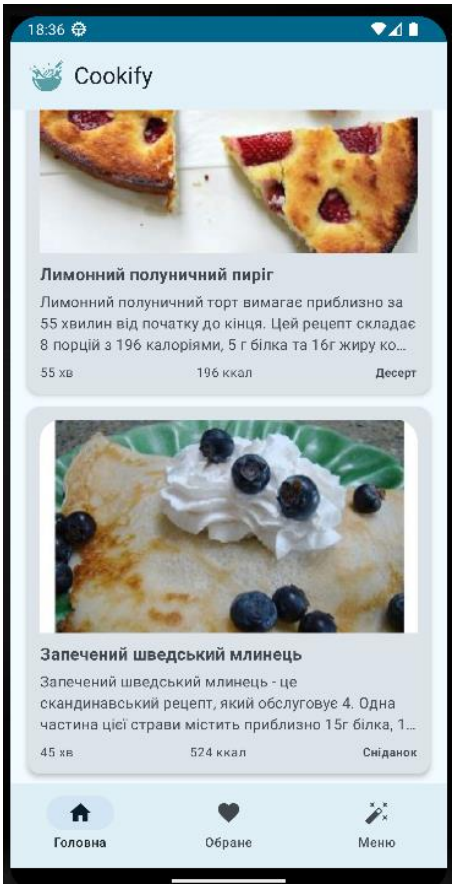

Рисунок Б.4. – Пошук рецептів

Перегляд детальної інформації про рецепт

Натиснувши на будь-яку картку зі списку, користувач переходить на екран детального перегляду. Тут відображаються: фото страви, назва, час приготування, калорійність, список інгредієнтів та покрокова інструкція приготування (Рис. Б.5-Б.6).

На цьому екрані розміщена кнопка/іконка додавання до «Обраного» (серце): повторне натискання її видаляє рецепт із «Обраного».

Рисунок Б.5. – Перегляд рецепта

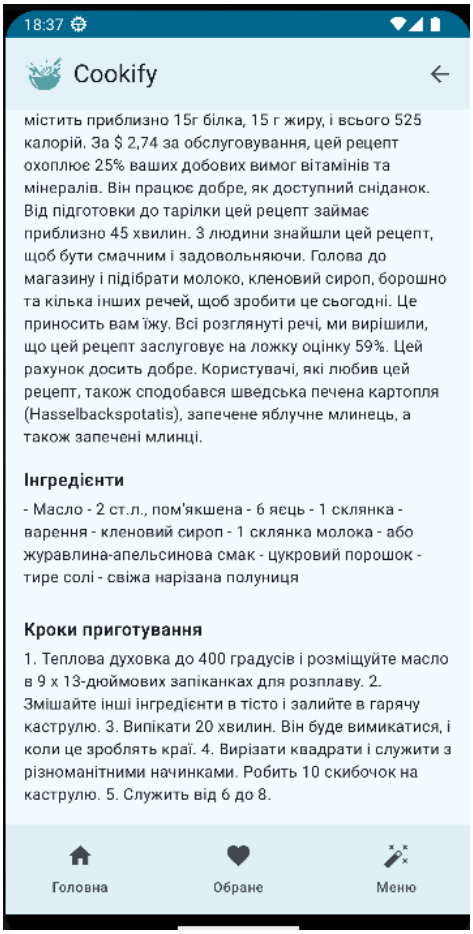

Рисунок Б.6. – Перегляд рецепта

Розділ «Обране»

На нижній панелі навігації користувач обирає пункт «Обране». Відкривається список раніше збережених рецептів. Тап по картці відкриває детальну інформацію про рецепт. Якщо обраних рецептів немає, відображається відповідне повідомлення (Рис. Б.7-Б.8).

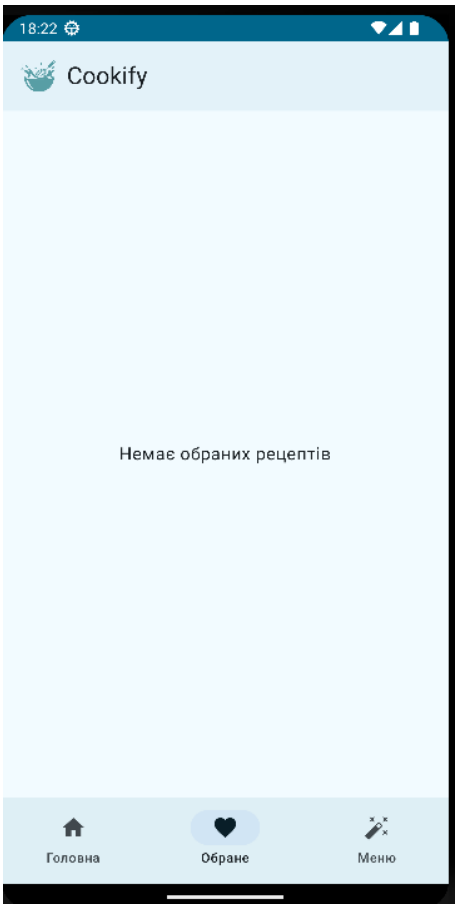

Рисунок Б.7. – Сторінка обраних рецептів

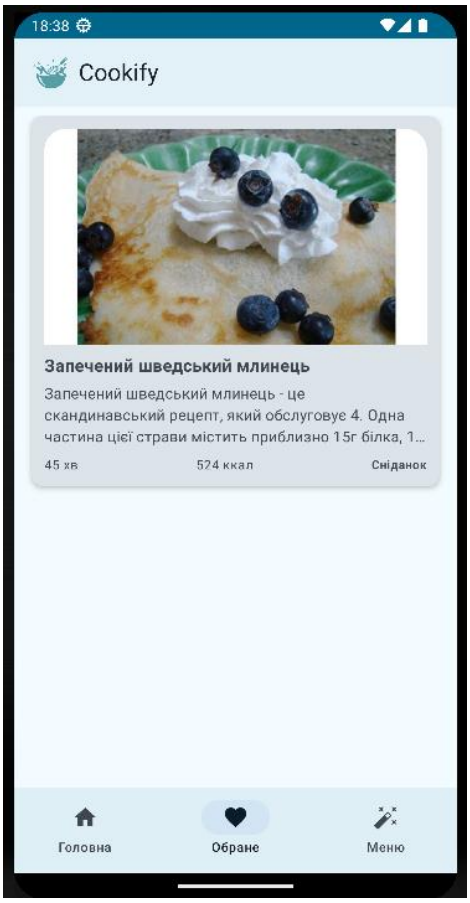

Рисунок Б.8. – Сторінка обраних рецептів

Екран меню на день

На нижній панелі обирається пункт «Меню». При першому відкритті користувач натискає кнопку генерації меню. Програма автоматично підбирає кілька страв різних типів (сніданок, обід, вечеря, десерт) і показує їх у вигляді списку карток. Натискання на картку відкриває деталі рецепта. Згенероване меню зберігається для поточної дати й відображається при повторному відкритті екрана без нового запиту (Рис. Б.9-Б.10).

Рисунок Б.9. – Сторінка меню

Рисунок Б.9. – Сторінка меню

Після завершення роботи користувач може закрити застосунок стандартними засобами Android (кнопка «Назад», «Дім», перегляд останніх програм). Локально збережені дані (обране, меню) залишаються доступними при наступному запуску.

Додаток В

Таблиця В.1

Характеристика	Recipe Keeper	Tasty	Mealime
Основний фокус	Зберігання власних рецептів, органайзер	Готові рецепти та відеоінструкції	Планування меню і список покупок
Джерело рецептів	Переважно користувач (власні рецепти, імпорт з вебу)	Велика внутрішня база рецептів	Внутрішня база рецептів і плани харчування
Планування меню	Є тижневий/місячний планувальник	Обмежене, без акценту на повноцінне меню	Основна функція (плани на тиждень)
Список покупок	Є, формується з обраних рецептів	Є базові можливості	Автоматично генерується з плану
Робота з уподобаннями користувача	Категорії, улюблені рецепти	Фільтри за інгредієнтами, типом кухні	Харчові профілі та дієтичні обмеження
Модель монетизації	Одноразова покупка / розширений функціонал	Безкоштовно з рекламою й додатковими опціями	Безкоштовна версія + платна підписка
Типові обмеження	Менше готового контенту «з коробки»	Обмежений офлайн-режим, внутрішні покупки	Частина функцій доступна лише за підпискою

Додаток Г

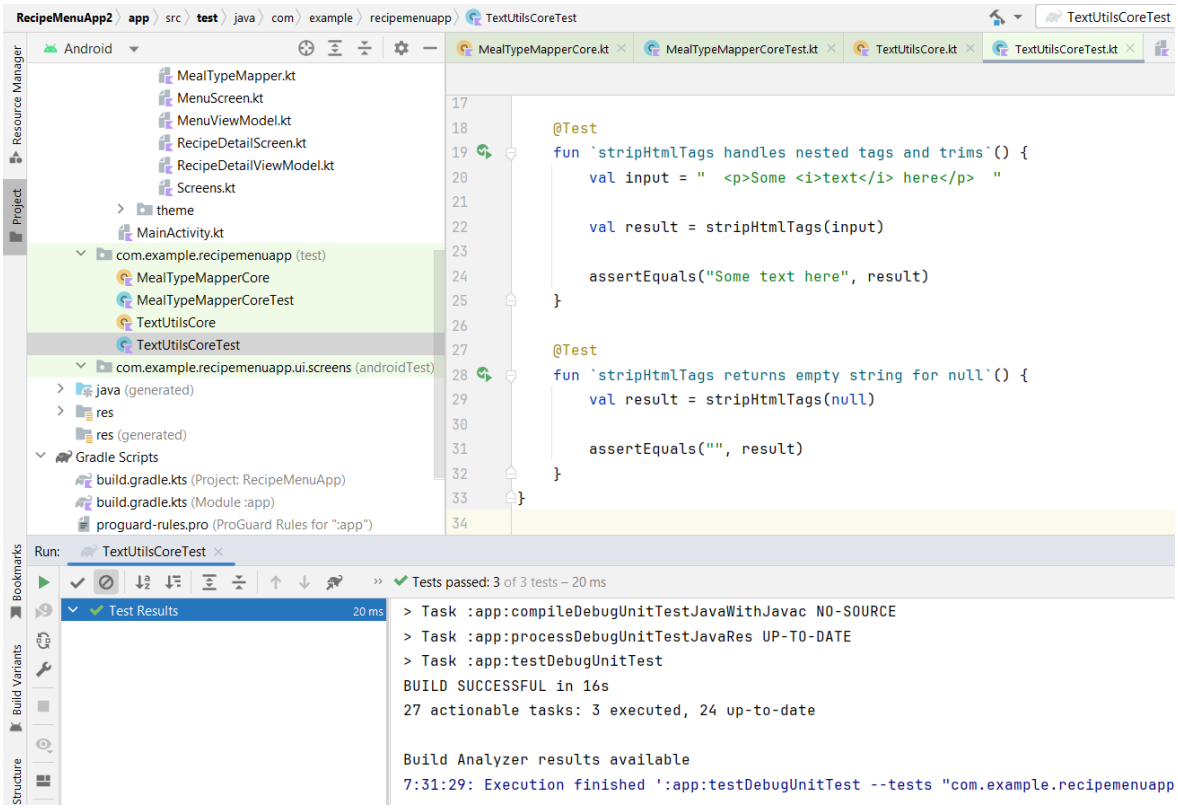

Рисунок Г.1. – Результати тестів TextUtilsCoreTest

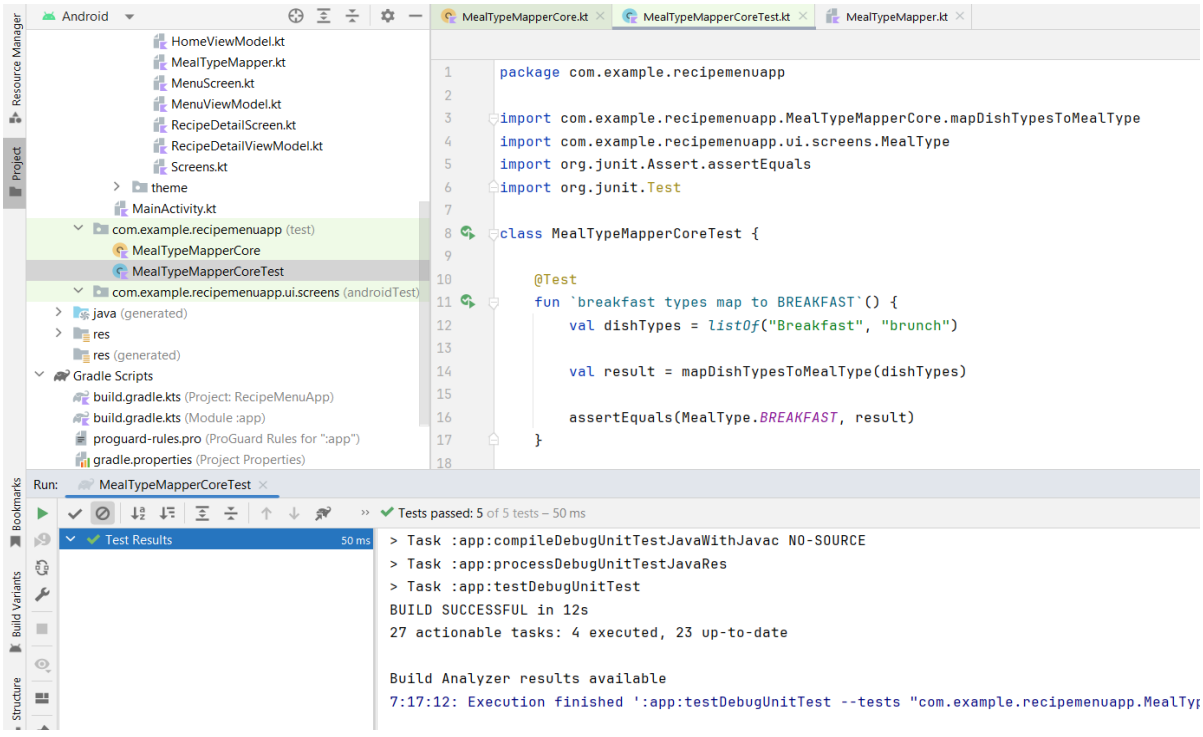

Рисунок Г.2. – Результати тестів MealTypeMapperCoreTest

АНОТАЦІЯ

Расік М. Р. Проєктування та розробка мобільного застосунку для підбору рецептів і меню. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025 рік.

Кваліфікаційна робота присвячена проєктуванню та реалізації мобільного застосунку для платформи Android, який призначений для підбору кулінарних рецептів за інгредієнтами та формування меню на день. У роботі проаналізовано предметну область організації харчування, розглянуто роль інформаційних технологій та рекомендаційних систем, виконано огляд наявних програмних засобів для роботи з рецептами та сформульовано вимоги до функціональності й зручності використання розроблюваного застосунку.

Запропоновано архітектуру застосунку з поділом на рівні даних, логіки та інтерфейсу. Реалізацію виконано з використанням Kotlin, Jetpack Compose, Room, Retrofit та зовнішнього REST API сервісу Spoonacular як основного джерела даних про рецепти. Застосунок забезпечує пошук рецептів за інгредієнтами, перегляд детальної інформації про страви, збереження улюблених рецептів і автоматичне формування денного меню з урахуванням типу прийому їжі. Передбачено кешування частини даних і використання простих правил відбору страв.

Результати роботи узагальнюють вимоги до мобільних застосунків для підбору рецептів і демонструють послідовність етапів розробки програмного забезпечення, що інтегрується з зовнішнім вебсервісом. Практичне значення полягає у створенні працездатного програмного продукту, який може бути використаний для щоденного планування харчування та як основа для подальшого розширення функціональності й навчальних прикладів з мобільної розробки.

Ключові слова: мобільний застосунок, рецепти, меню, рекомендаційна система, Android, Kotlin, Jetpack Compose, REST API, Spoonacular.

ANNOTATION

Rasik M. R. Design and development of a mobile application for recipe and menu selection. Manuscript.

Qualification thesis for obtaining the Master's degree in speciality 122 Computer Science. Lesya Ukrainka Volyn National University, Lutsk, 2025.

The thesis is devoted to the design and implementation of a mobile application for the Android platform intended for selecting cooking recipes based on ingredients and generating a daily menu. The work analyses the problem domain of meal planning, the role of information technologies and recommender systems, reviews existing applications for working with recipes and formulates requirements for the functionality and usability of the developed solution.

The proposed architecture of the application is based on a clear separation into data, business logic and presentation layers. The implementation uses Kotlin, Jetpack Compose, Room, Retrofit and the external Spoonacular REST API as the main data source for recipes. The application supports recipe search by ingredients, detailed recipe view, saving favourite recipes and automatic generation of a daily menu that includes meals of different types (breakfast, lunch, dinner, dessert). Partial data caching and simple rules for selecting dishes are applied.

The results of the work summarise requirements for mobile applications for recipe selection and demonstrate a sequence of development stages for software that integrates with an external web service. The practical value lies in a working software product that can be used for everyday meal planning and as a basis for further functionality extensions and teaching examples in mobile development courses.

Keywords: mobile application, recipes, menu, recommender system, Android, Kotlin, Jetpack Compose, REST API, Spoonacular.