Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим ХРУСЛОВ
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживач»

Спеціальність 123 – Комп'ютерна інженерія.
Галузь знань 12 – Інформаційні технології.
Освітня програма «Комп'ютерна інженерія»

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А. М.**

Виконав:
студент групи КІ-41
ОВЧАРЕНКО Ярослав Олександрович

Керівник:
Старший викладач кафедри електроніки та управляючих систем
ОСИПЧУК Андрій Володимирович

Рецензент:
Доцент кафедри безпеки інформаційних систем і технологій, кандидат технічних наук
НАРЄЖНІЙ Олексій Павлович

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. Загальний обсяг роботи становить 58 сторінок, з яких 44 сторінки – основна частина, що містить 15 рисунків, 16 позиції у списку джерел та 3 додатків.

Метою кваліфікаційної роботи є розробка та практичне впровадження мобільного додатка-сервісу для персоналізованого вибору страв, який автоматично розраховує харчові потреби користувача та формує рекомендації згідно з індивідуальними дієтичними критеріями.

Об'єктом дослідження є процес проєктування й реалізації мобільних рекомендативних систем у сфері харчування.

Предметом дослідження – алгоритми персоналізації (контентна, колаборативна та гібридна фільтрація), моделі збереження даних (Room DB) і UX-патерни для Android-застосунків (MVVM, Material Design).

Проблема, яка розв'язується в роботі, полягає у відсутності зручного інструменту для швидкого підбору страв із урахуванням калорійності, нутрієнтів та особистих дієтичних обмежень, що ускладнює самоконтроль харчування й досягнення оздоровчих цілей.

Область застосування результатів — персональні фітнес- і wellness-сервіси, дієтологічні програми, системи корпоративного харчування, а також освітні платформи зі здорового способу життя, де потрібен індивідуальний раціон на основі національних (ДСТУ ISO / IEC 2382-37:2018) і міжнародних (FAO/WHO, ISO / IEC 25010) стандартів харчових норм та якості ПЗ.

Ключові слова: персоналізована рекомендаційна система, мобільний додаток, Android Studio, MVVM, Room DB, харчові потреби, калорійність, контентна та колаборативна фільтрація, Material Design.

ABSTRACT

The explanatory paper of the qualification project consists of an introduction, three chapters, conclusions, a list of references and appendices. Its total length is 58 pages, of which 44 pages make up the main part containing 15 figures, 16 references, and 3 appendices.

The aim of the project is to design and implement a mobile service-application for personalised dish selection that automatically calculates a user's nutritional needs and generates recommendations according to individual dietary criteria.

The object of the research is the process of designing and implementing mobile recommender systems in the nutrition domain.

The subject of the research comprises personalisation algorithms (content-based, collaborative and hybrid filtering), data-persistence models (Room DB) and UX patterns for Android applications (MVVM, Material Design).

The problem addressed is the absence of a convenient tool for quickly selecting meals while taking into account calories, nutrients and personal dietary restrictions, which complicates self-monitoring and the achievement of health goals.

The scope of practical use includes personal fitness and wellness services, dietetics programmes, corporate catering systems and healthy-lifestyle educational platforms that require individual rations in accordance with national (DSTU ISO/IEC 2382-37:2018) and international (FAO/WHO, ISO/IEC 25010) standards for nutritional norms and software quality.

***Key words*:** personalised recommender system, mobile application, Android Studio, MVVM, Room DB, nutritional needs, caloric value, content-based and collaborative filtering, Material Design.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

UI	—	**U**ser **I**nterface — користувацький інтерфейс
UX	—	**U**ser e**X**perience — користувацький досвід
MVVM	—	**M**odel – **V**iew – **V**iew**M**odel — архітектурний шаблон
DAO	—	**D**ata **A**ccess **O**bject — об’єкт доступу до даних
DB	—	**D**ata**b**ase — база даних
Room DB	—	Room DataBase — ORM-бібліотека для локальної БД Android
API	—	**A**pplication **P**rogramming **I**nterface — програмний інтерфейс застосунку
CRUD	—	**C**reate, **R**ead, **U**pdate, **D**elete — базові операції над даними
ER	—	**E**ntity-**R**elationship — модель «сутність-зв’язок»
IDE	—	**I**ntegrated **D**evelopment **E**nvironment — інтегроване середовище розробки
XML	—	e**X**tensible **M**arkup **L**anguage — розширювана мова розмітки
HTTPS	—	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure — захищений HTTP
RAM	—	**R**andom **A**ccess **M**emory — оперативна пам’ять
CPU	—	**C**entral **P**rocessing **U**nit — центральний процесор
ORM	—	**O**bject-**R**elational **M**apping — об’єктно-реляційне відображення

ВСТУП

Актуальність дослідження

У контексті зростання інтересу до здорового способу життя та широкого поширення мобільних технологій існує потреба у зручних інструментах, що допомагають користувачам контролювати раціон і швидко підбирати страви відповідно до індивідуальних харчових потреб, дієтичних обмежень і смакових вподобань. Більшість наявних мобільних застосунків пропонують лише базову інформацію про калорійність або статичні меню й не надають комплексного персоналізованого сервісу. Тому розробка мобільного додатка-сервісу, що поєднує гібридний рекомендаційний механізм, зручний Material-інтерфейс і сучасну архітектуру MVVM, є актуальним завданням, яке відповідає трендам e-health та digital-wellness.

Мета дослідження

Створити та впровадити мобільний додаток-сервіс для персоналізованого вибору страв, який автоматично розраховує добові харчові потреби користувача й формує рекомендації на основі індивідуальних критеріїв (калорійність, нутрієнти, дієтичні обмеження), а також оцінити його функціональну придатність та ефективність.

Об’єкт дослідження

Процес проєктування та реалізації мобільних рекомендативних систем у сфері харчування.

Предмет дослідження

Алгоритми персоналізації (контентна, колаборативна й гібридна фільтрація), моделі збереження даних (Room DB), архітектурні та UX-патерни для Android-застосунків (MVVM, Material Design) і їхня інтеграція у єдиний мобільний сервіс.

Завдання дослідження

1. Провести огляд наукових публікацій і комерційних рішень з мобільних рекомендаційних систем харчування; визначити вимоги до персоналізації та UX.
2. Сформулювати функціональні й нефункціональні вимоги до додатка з урахуванням ролей «Користувач» та «Адміністратор».
3. Розробити концептуальну модель даних і побудувати ER-діаграму сутностей (USER, DISH, CATEGORY, FAVORITE, SYSTEMLOAD).
4. Спроєктувати архітектуру за шаблоном MVVM і компонентну схему інтеграції Room, Coroutines/Flow та Jetpack Navigation.
5. Реалізувати гібридний рекомендаційний алгоритм, що поєднує контентну та колаборативну фільтрацію, і забезпечує офлайн-кешування даних.
6. Створити адаптивний Material-інтерфейс (головний екран, «Улюблені», деталі страви, адміністративний CRUD-екран).
7. Провести unit-, інтеграційні та UI-тести, а також профілювання продуктивності (час відповіді, використання RAM).
8. Оцінити ефективність рішення й сформувати рекомендації щодо масштабування та подальшого розвитку.

Методи дослідження

Аналіз і синтез наукових джерел; UML-моделювання (Use Case, ER, компонентні діаграми); прототипування UI у Figma/Android Studio; реалізація коду Kotlin із використанням Room, Coroutines, Hilt DI; автоматизоване тестування (JUnit, Mockito, Espresso); інструментальне профілювання (Android Profiler); статистична обробка результатів тестів.

Практична цінність роботи

Розроблений мобільний застосунок може бути інтегрований у фітнес- і wellness-сервіси, дієтологічні програми та корпоративні системи харчування, забезпечуючи користувачам швидкий підбір страв, що відповідають їхнім індивідуальним потребам. Гнучка архітектура та відкритий набір API дозволяють масштабувати функціонал (підключення хмарних сервісів, носимих пристроїв, алгоритмів машинного навчання) й адаптувати рішення до інших платформ.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПЕРСОНАЛІЗОВАНОГО ВИБОРУ СТРАВ

1.1 Аналіз предметної області та формулювання проблеми

В останні роки відбувається безперервний розвиток мобільних технологій у сфері охорони здоров'я (mHealth). Смартфони та планшети дедалі частіше використовуються для моніторингу фізичного стану та харчування людини. У категорії «Здоров'я та фітнес» мобільних додатків нині налічується понад 100 тисяч програм. Це пояснюється широким розповсюдженням смартфонів серед населення та прагненням людей покращити своє здоров'я з допомогою цифрових інструментів. Дослідники відзначають, що мобільні додатки для здоров'я мають великий потенціал для сприяння зміні поведінки користувачів і можуть допомогти в профілактиці та контролі хронічних захворювань.

Відомо, що модифікація харчування та контроль калорій є одним з найефективніших заходів для профілактики ожиріння і пов'язаних із ним хронічних хвороб. Багато наукових досліджень показують, що регулярне ведення харчового щоденника та самоконтроль споживання калорій сприяють зниженню ваги й поліпшенню загального стану здоров'я. Це призвело до буму спеціалізованих мобільних додатків для харчування. Наприклад, MyFitnessPal, Lifesum, YAZIO та інші популярні сервіси дозволяють користувачам відстежувати спожиті страви та нутрієнти, встановлювати цілі щодо калорій та макроскладових, а також отримувати рекомендації щодо раціонів і страв на основі введених даних. Багато таких додатків мають величезні бази даних продуктів і автоматичний підрахунок калорій, функції сканування штрих-кодів та навіть можливість зв'язку з дієтологами чи спільнотами однодумців. Однак у більшості сучасних програм користувачу все ще доводиться вручну обирати страви або вводити дані, а автоматичних порад щодо конкретних нових рецептів або страв не вистачає.

Таким чином, актуальною є задача розробки системи, яка змогла б автоматично рекомендувати користувачеві персоналізовані страви або рецепти з урахуванням його харчових уподобань, дієтичних цілей і обмежень. Проблема персоналізованого вибору страв лежить на перетині двох предметних областей: нутріціології (балансування корисного харчування) та рекомендуючих систем. Існуючі механізми рекомендацій (як-от в системах онлайн-магазинів чи медіатеку) можуть бути адаптовані до мобільних дієтичних сервісів. Але для цього необхідно проаналізувати алгоритми, що лежать в основі різних типів рекомендаційних систем (контентні, колаборативні, гібридні), та зрозуміти їхні сильні й слабкі сторони в контексті харчових додатків. Наприклад, колаборативні фільтри можуть пропонувати користувачеві рецепти, які подобаються іншим людям зі схожими смаковими вподобаннями, а контентні підходи – пропонувати страви зі схожими інгредієнтами до вже улюблених. Розгляд цих методів допоможе сформувати вимоги до мобільного додатку на наступних етапах проєктування.

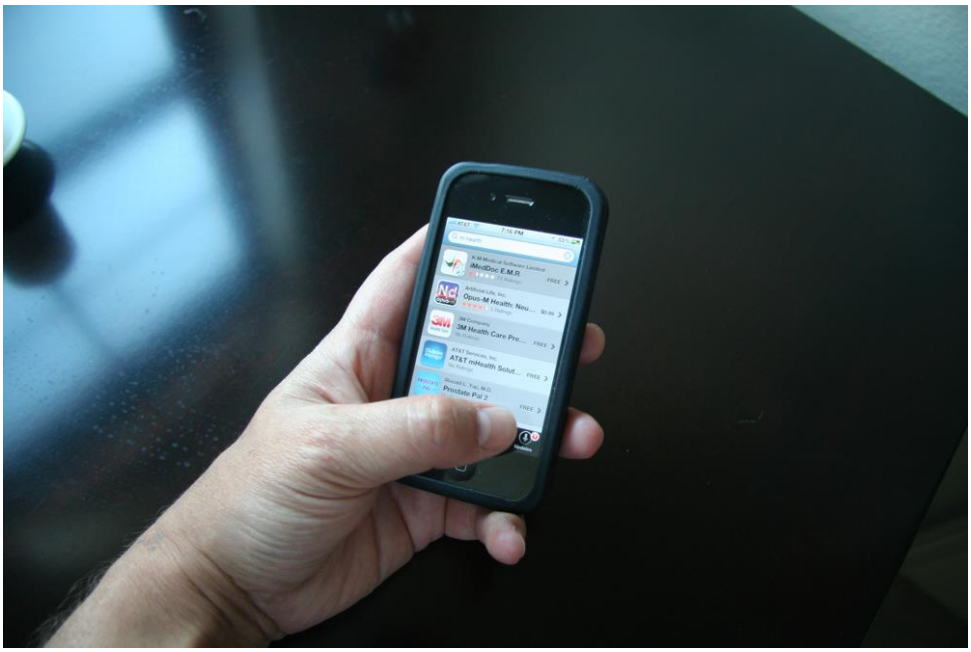

Рисунок 1.1 – Приклад використання мобільного додатку для моніторингу харчування і здоров’я (сервіси на зразок MyFitnessPal, Lifesum, YAZIO).

Багато сучасних харчових застосунків включають функції ведення харчового щоденника, самостійного введення прийомів їжі, відстеження ваги та фізичної активності. Проте забезпечення дійсно персоналізованого підбору страв потребує впровадження складних алгоритмів аналізу даних і рекомендацій. У цьому зв’язку формулюється проблема дослідження: які методи рекомендуючих систем слід застосувати для формування таких рекомендацій у контексті мобільного додатку з вибору страв, та які переваги і обмеження цих методів слід враховувати при проєктуванні системи.

1.2 Огляд методів рекомендаційних систем

Рекомендаційні системи — це інформаційно-фільтраційні інструменти, які допомагають користувачам знаходити найбільш релевантний персоналізований контент серед великої кількості варіантів. Основна мета рекомендаційних систем — зменшити зусилля та час, необхідні користувачеві для пошуку потрібної інформації. У нашій задачі до таких систем належать алгоритми, що відбирають найбільш підходящі страви або рецепти для конкретного користувача. Існує кілька класичних підходів до побудови рекомендацій: контентні (content-based), колаборативні (collaborative) та гібридні (hybrid) методи.

- **Контентна фільтрація:** в контентно-орієнтованих системах усі елементи (страви, рецепти) описуються набором ознак або ключових слів (наприклад, список інгредієнтів, тип кухні, харчова цінність). Коли користувач позитивно оцінює певну страву, алгоритм вважає, що йому можуть сподобатися інші страви з подібним контентом. Наприклад, якщо користувач додав у щоденник салат з конкретними інгредієнтами, система може рекомендувати йому інші салати з схожими інгредієнтами або калорійністю. Основними перевагами такого підходу є висока адаптивність до зміни вподобань одного користувача (система швидко враховує нові оцінки) і збереження приватності (рекомендації формуються лише на основі даних про

цього користувача). Окрім того, контентні алгоритми можуть долати «проблему холодного старту» з новими стравами: якщо є повний опис нової страви, її можна рекомендувати, навіть якщо її ніхто раніше не оцінював. Водночас контентні методи мають недоліки: вони вимагають детальних знань про характеристики страв (не завжди є точні структуровані дані про всі страви) і схильні до «затоварення» — рекомендують користувачу лише їжу, дуже подібну до вже відомих йому варіантів. Наприклад, якщо користувач полюбляє лише піцу і пасту, чисто контентна система навряд чи запропонує йому легку овочеву страву, навіть якщо це було б корисно для здоров'я.

- **Колаборативна фільтрація:** цей підхід ґрунтується на аналізі взаємних вподобань багатьох користувачів. Система будує «сусідство» для цільового користувача – знаходить групу інших користувачів із подібними харчовими вподобаннями (на основі схожості їхніх історій оцінок або вибору страв). Далі нові рецепти, які сподобалися більшості із цієї групи, рекомендуються цільовому користувачеві. Колаборативні методи не потребують жодного опису страв – їм достатньо знати, як різні люди оцінювали страви. Це дозволяє відкривати користувачеві нетривіальні рекомендації: наприклад, якщо інші подібні користувачі часто їдять італійську кухню і мексиканську, система може порадити обидві, навіть якщо користувач раніше знав лише одну з них. Недоліком колаборативних алгоритмів є «холодний старт»: якщо новий користувач лише зареєструвався і мало що ввів, для нього важко відразу підібрати рекомендації (система потребує достатньої кількості даних). Також поділ даних про уподобання всіх користувачів може викликати питання приватності. Крім того, колаборативні моделі можуть вимагати великих обчислювальних ресурсів при багатьох

користувачах і товарах, хоча існують оптимізації (кластеризація, машинне навчання тощо).

- **Гібридні підходи:** гібридні системи поєднують два або більше згаданих методів з метою мінімізації їхніх недоліків. Наприклад, типова гібридна стратегія може спершу використовувати контентний алгоритм, щоб обмежити базу страв за загальними ознаками, а потім застосувати колаборативний підхід на цій зменшеній вибірці. Як показують дослідження, комбінування методів зазвичай призводить до підвищення точності та якості рекомендацій. У контексті вибору страв це може означати, що система враховує і харчовий склад страви (контент), і оцінки користувачів зі схожими вподобаннями (колаборація) одночасно. Однак створення гібридних систем складніше: вони потребують більше даних (як про характеристики страв, так і про велику кількість користувачів) та мають вищі обчислювальні вимоги. З іншого боку, гібриди здатні краще справлятися з проблемами холодного старту (за рахунок «підкидання» метаданих) і зменшують ризик затоварення, комбінуючи сильні сторони обох підходів.

Вважається, що в сфері харчових рекомендаційного дослідження колаборативні методи застосовуються найчастіше. Огляд літератури показав, що більшість проєктів у цій галузі використовують колаборативну фільтрацію як основу системи рекомендацій, водночас також зустрічаються чисто контентні та гібридні рішення. Це пояснюється тим, що колаборативні методи добре працюють, коли є дані багатьох користувачів, а також будуть випереджати контентні при виявленні «неочевидних» поєднань смаків.

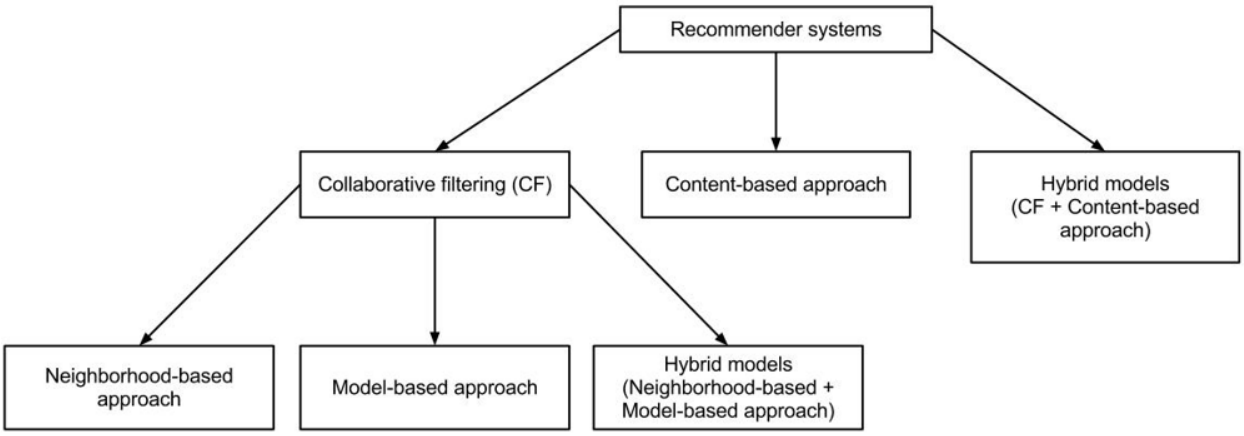

Рисунок 1.2 – Класифікація рекомендаційних систем за типом алгоритму (контентні, колаборативні, гібридні).

На рисунку подано узагальнену схему: контентні системи рекомендують на основі властивостей об’єктів, колаборативні – на основі схожості користувачів, а гібриди об’єднують ці підходи для покращення ефективності. Існують різні варіанти гібридизації: наприклад, послідовні (спочатку фільтрація за одним методом, потім за іншим), суміші методів чи обчислення зважених оцінок з декількох алгоритмів. У контексті мобільного додатку для вибору страв вибір конкретного гібридного підходу (схема на рис. 1.2) визначатиме, наскільки точно і різноманітно система зможе рекомендувати користувачеві нові здорові страви.

1.3 Переваги та недоліки існуючих підходів

Оцінюючи застосовність кожного з підходів у задачу персоналізованого підбору страв, слід узагальнити їхні сильні та слабкі сторони:

- **Контентні системи:** переваги – висока адаптивність до змін уподобань одного користувача, забезпечення конфіденційності (дані інших користувачів не потрібні) і здатність рекомендувати нові страви за наявності їх опису. Недоліки – потреба в достовірних даних

про склад страв (тому погано працюють з некодованим інформаційним простором) і ризик «ефекту колеса» (повторення вже відомих смакових уподобань користувача, без пропозицій новинок).

- **Колаборативні системи:** переваги – виявлення рекомендацій «з натовпу»: користувачу можуть бути запропоновані страви, на основі смаків схожих користувачів, навіть якщо ці страви мають невеликі подібні з попередніми інгредієнти. Колаборативна фільтрація не потребує формального опису страв, лише даних про взаємодію (оцінки, лайки). Недоліки – слабка робота при «холодному старті» (нова користувацька реєстрація або нові страви без оцінок на початку) та можливі проблеми конфіденційності, оскільки алгоритм аналізує дані багатьох користувачів. Також «чиста» колаборативна модель може не враховувати харчову або дієтичну складову рекомендацій (наприклад, може порекомендувати страву, що не підходить за поживністю), якщо цього не передбачено гібридним механізмом.
- **Гібридні системи:** переваги – поєднуючи сильні сторони обох підходів, такі системи зазвичай дають найбільш точні рекомендації (особливо у випадках змішаних цілей: і смакових вподобань, і поживних потреб). Гібриди можуть частково вирішувати проблему холодного старту (через включення контентної частини) і зменшують ефект затоварення. Недоліки – складність налаштування та обробки: гібридній системі потрібні як структуровані метадані страв, так і об’ємні дані про активність користувачів, що підвищує обчислювальні витрати на обробку рекомендацій. До того ж інтеграція двох різних методів іноді призводить до труднощів у балансуванні їхнього впливу (який метод має перевагу в конкретній ситуації).

Таким чином, вибір конкретного підходу для мобільного додатку залежить від вимог до системи. Якщо пріоритет – конфіденційність та швидка реакція на нові вподобання одного користувача (наприклад, дієтологічний додаток, де користувач сам вибирає страви за параметрами), доцільні контентні методи. Якщо ж доступні оцінки та вподобання багатьох користувачів (наприклад, у великій спільноті шанувальників здорового харчування), можна застосувати колаборативні алгоритми для виявлення корисних шаблонів. А якщо потрібно максимально врахувати і харчові властивості страв, і досвід груп користувачів, найкращим рішенням буде гібридна модель. Всі згадані переваги й недоліки повинні бути враховані при наступному проєктуванні архітектури мобільного додатку для персоналізованого вибору страв.

Висновки до розділу 1

У Розділі 1 було здійснено ґрунтовний аналіз предметної області персоналізованого вибору страв, а також огляд основних підходів у рекомендуючих системах, що можуть бути застосовані в мобільних харчових додатках. По-перше, було визначено актуальність проблеми: сучасні сервіси для контролю харчування потребують інструментів автоматичного підбору страв із врахуванням індивідуальних уподобань і дієтичних обмежень користувачів. По-друге, розглянуто три базові категорії алгоритмів — контентні, колаборативні та гібридні — їхні принципи роботи, сильні й слабкі сторони. Контентні методи дозволяють оперативно реагувати на зміну смакових вподобань і зберігати конфіденційність даних, але схильні до «затоварення» схожим контентом, тоді як колаборативні можуть відкривати нетривіальні рекомендації на основі вподобань подібних користувачів, проте страждають від «холодного старту». Гібридні рішення, комбінуючи переваги обох підходів, здатні підвищити точність і різноманітність рекомендацій, хоча вимагають більших обчислювальних ресурсів і складнішої інтеграції даних.

Таким чином, аналіз теоретичних методів рекомендаційних систем показав, що для реалізації ефективного мобільного додатка-сервісу доцільно застосувати гібридний підхід із можливістю динамічного переключення між контентною та колаборативною фільтрацією. Отримані в цьому розділі знання стануть основою для побудови концептуальної моделі системи та подальшого проєктування архітектури застосунку.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ ПЕРСОНАЛІЗОВАНОГО ВИБОРУ СТРАВ

2.1 Визначення функціональних та нефункціональних вимог

На першому етапі проєктування формулюються функціональні та нефункціональні вимоги системи. Функціональні вимоги визначають конкретний набір можливостей програми – що саме має робити система в різних сценаріях використання. Нефункціональні вимоги задають загальні властивості та обмеження системи (швидкість, безпеку, надійність тощо). Чітке визначення вимог є основою для подальшого дизайну і розробки системи. У проєктованій системі передбачено дві ролі: звичайний користувач (User) та адміністратор (Admin). Адміністратор відповідає за управління контентом: він може створювати нові записи про страви, редагувати параметри існуючих страв і видаляти непотрібні записи. Звичайний користувач може переглядати перелік доступних страв, виконувати пошук і фільтрацію за різними критеріями (категорією, інгредієнтами, дієтичними обмеженнями) і отримувати відповідні рекомендації.

- **Користувач (User):** має можливість переглядати каталог страв, сортувати та фільтрувати їх, переглядати деталі конкретної страви та додавати або видаляти страви зі списку обраних. Користувач також може налаштовувати особисті вподобання (наприклад, обирати улюблені категорії або вказувати алергенні продукти).
- **Адміністратор (Admin):** може додавати нові страви (вказуючи назву, опис, категорію, список інгредієнтів, зображення та інші атрибути), редагувати інформацію про існуючі страви та видаляти непотрібні записи. Адміністратор також керує списком категорій (або теґів) страв.

- **Аутентифікація:** система забезпечує реєстрацію та вхід користувача для розмежування прав доступу. Лише авторизований адміністратор має можливість редагувати дані про страви.
- **Пошук та фільтрація:** користувач може шукати страви за назвою або ключовими словами, а також застосовувати фільтри за категоріями, інгредієнтами чи іншими характеристиками (наприклад, калорійністю чи типом кухні).
- **Персоналізація:** система зберігає дані про вподобання користувача та використовує їх для формування персоналізованих рекомендацій. На основі обраних страв і налаштувань система може пропонувати корисні для користувача варіанти.
- **Перегляд інформації:** для кожної страви передбачено детальний опис, який може включати список інгредієнтів, інформацію про поживну цінність, інструкцію з приготування тощо.
- **Збереження даних:** всі дані про страви, категорії та обрані страви користувачів зберігаються в базі даних, що гарантує їхню збереженість між сеансами роботи.

Ключові сценарії використання (Use-case) додатку включають:

- Перегляд каталогу страв із можливістю пошуку та фільтрації за заданими параметрами.
- Додавання страв до списку улюблених («Обране») і перегляд цього списку.
- Формування персоналізованих рекомендацій на основі вподобань користувача.
- Створення, редагування та видалення інформації про страви (здійснюється адміністратором).

- Управління категоріями страв (додавання нових категорій, призначення категорій до страв).

Нефункціональні вимоги задають якісні характеристики системи:

- **Продуктивність і відгук:** додаток має швидко реагувати на дії користувача. Зокрема, завантаження списку страв та результатів пошуку повинно відбуватися без помітних затримок.
- **Надійність та стабільність:** програма повинна коректно працювати навіть при нестабільному з’єднанні (за потреби зберігаючи робочі дані локально) і повідомляти користувача про помилки введення чи збої.
- **Юзабіліті:** інтерфейс повинен бути інтуїтивно зрозумілим і зручним. Дотримання принципів Material Design забезпечить єдиний сучасний стиль оформлення та послідовний досвід користувача.
- **Портативність та адаптивність:** додаток орієнтований на платформу Android і має належно працювати на пристроях з різними розмірами і характеристиками екрана. Передбачається підтримка щонайменше останніх кількох версій Android (наприклад, від Android 5.0 і вище). Для адаптивності макетів використані ConstraintLayout та згадані компоненти Android Jetpack, що забезпечує коректне відображення на різних пристроях.
- **Безпека:** зберігання персональних даних (облікові записи користувачів, їхні улюблені страви) має бути захищеним. Систему аутентифікації реалізовано з урахуванням безпечних методів (наприклад, через токени чи захищене з’єднання) для захисту даних. Паролі користувачів зберігаються у формі хешу, а передача чутливої інформації здійснюється через зашифровані канали (HTTPS).

- **Модифікованість і масштабованість:** архітектура додатку повинна бути гнучкою для подальшого розширення функціоналу. Використання патерну MVVM забезпечує чіткий розподіл обов'язків між компонентами, що полегшує підтримку і тестування системи. Додаток повинен коректно працювати з великими обсягами даних (багато записів про страви), і за потреби можливе підключення до віддаленого сервера або хмарної бази.
- **Тестованість і підтримуваність:** окремі компоненти програми повинні бути логічно розділеними, що дає змогу проводити модульні та інтеграційні тести. Розділення на ViewModel, Repository та інші шари підвищує зручність тестування окремих частин програми.
- **Локалізація:** інтерфейс додатку і текстові ресурси надаються українською мовою. При потребі система може підтримувати кілька мов (через механізм ресурсів Android).
- **Адаптивність:** дизайн інтерфейсу побудовано таким чином, щоб він коректно відображався на пристроях з різними розмірами екранів і щільностями пікселів. Для цього використовуються адаптивні макети (ConstraintLayout тощо) та ресурсні файли з різними роздільними здатностями.
- **Логування та моніторинг:** передбачено механізми логування критичних подій і помилок для діагностики роботи системи. Це сприяє швидкому виявленню багів і підтримці додатку в майбутньому.
- **Відмовостійкість:** система повинна обробляти можливі винятки (наприклад, ненадійне підключення до мережі або пошкоджені дані) без аварійного завершення роботи програми. Повідомлення про помилки мають бути зрозумілими для користувача.

- **Оновлюваність:** передбачено механізм оновлення даних (наприклад, синхронізації з віддаленим сервером) без повного оновлення самої програми. Система може періодично перевіряти наявність нових страв і оновлювати локальну базу даних.
- **Дотримання стандартів:** застосування стандартних компонентів Android та рекомендацій Material Design забезпечує передбачуваність інтерфейсу та знижує ймовірність помилок дизайну.
- **Логування (співвідношення):** система веде файл логів для фіксації критичних подій та збоїв, що полегшує процес підтримки та відлагодження.
- **Документація:** усі суттєві рішення та архітектурні підходи документуються. Це можуть бути технічні специфікації API (якщо такі є), схеми бази даних або коментарі до коду. Наявність документації полегшує вступ нових розробників у проект.
- **Узгодженість вимог:** кожен функціонал має бути перевірений тестами відповідно до цього опису. Наприклад, можна автоматизовано перевірити, що після додавання улюбленої страви вона відображається на відповідному екрані, або що звичайний користувач не має доступу до функцій адміністратора. Таким чином, специфікація вимог є фундаментом для розробки та тестування.

Важливо також передбачити можливість розширення функціоналу в майбутньому: архітектура додатку повинна дозволяти додавання нових модулів без суттєвого переписування існуючого коду. Наприклад, у перспективі можна буде реалізувати синхронізацію даних з хмарою або інтеграцію з іншими сервісами. Отже, на основі вищеописаних вимог формується повна специфікація, що стане фундаментом подальшої розробки. Усі ці вимоги складають угоду про обсяг робіт, які має виконати команда

розробки. На цьому підрозділ 2.1 завершує опис вимог і передає естафету подальшим етапам проектування.

2.2 Концептуальна модель та архітектура системи

Концептуальна модель системи задає структуру даних та взаємозв'язки основних об'єктів. На практиці концептуальне моделювання часто виконується за допомогою ER-діаграми, яка відображає сутності, їх атрибути та зв'язки (наприклад, відношення один-до-багатьох між Категорією та Стравою). У цьому розділі наведено концептуальну модель даних і загальну архітектуру системи. Концептуальна модель визначає ключові сутності та їх зв'язки у межах предметної області, що служить основою для дизайну бази даних. Архітектура системи описує розподіл ролей між компонентами (UI, бізнес-логікою, джерелом даних). Для Android-додатків за сучасними рекомендаціями обрано архітектуру MVVM, оскільки вона підвищує модульність та тестованість програми. Такий підхід дозволяє ізолювати бізнес-логіку і роботу з даними від представлення (UI), що значно підвищує гнучкість проєкту і спрощує його підтримку.

Діаграма варіантів використання (Use-case)

Система підтримує взаємодію двох типів користувачів – звичайних користувачів і адміністратора – з набором базових функціональних сценаріїв.

На цій діаграмі показано, що звичайний **Користувач** може переглядати список доступних страв, виконувати пошук або фільтрацію за параметрами, переглядати деталі конкретної страви та додавати її до списку обраних. Також передбачені сценарії перегляду вже доданих у «Обране» страв та налаштування персональних уподобань. **Адміністратор** має схожі сценарії перегляду, а також додаткові: він може створювати нову страву, змінювати параметри існуючої та видаляти її з бази.

ER-діаграма (Entity-Relationship)

ER-діаграма демонструє основні сутності: **Користувач**, **Страва**, **Категорія** та зв’язну сутність **Обране**. Користувач може мати багато записів у таблиці Обране (зв’язок 1:M), що означає список обраних страв користувача. Відповідно, одна Страва може додаватися до обраних у багатьох користувачів (зв’язок M:1 у зворотному напрямі). Сутність Категорія пов’язана з Стравою за принципом один-до-багатьох: кожна страва належить лише до однієї категорії. Така модель даних забезпечує зберігання інформації про те, яким категоріям належать страви та які страви вважаються улюбленими кожного користувача.

Рисунок 2.1 – Use-case діаграма ролей та сценаріїв для DishSelectorApp

В термінах коду кожна сутність ER-діаграми відповідає Kotlin-класу з анотацією @Entity (Room). Наприклад, клас Dish має поля name, description, price та інші, які відображають атрибути сутності «Страва». Клас Category відповідає сутності «Категорія». Зв’язок один-до-багатьох між Category і Dish реалізується через зовнішній ключ: у класі Dish є поле categoryId, що вказує на ідентифікатор категорії. Сутність Favorite (Обране) реалізована як окрема таблиця з двома полями userId і dishId, які разом утворюють композитний первинний ключ. При зміні моделі даних (наприклад, додавання нового поля в клас Dish) слід передбачити механізм міграції бази даних (Room Migration), щоб зберегти існуючу інформацію при оновленні програми.

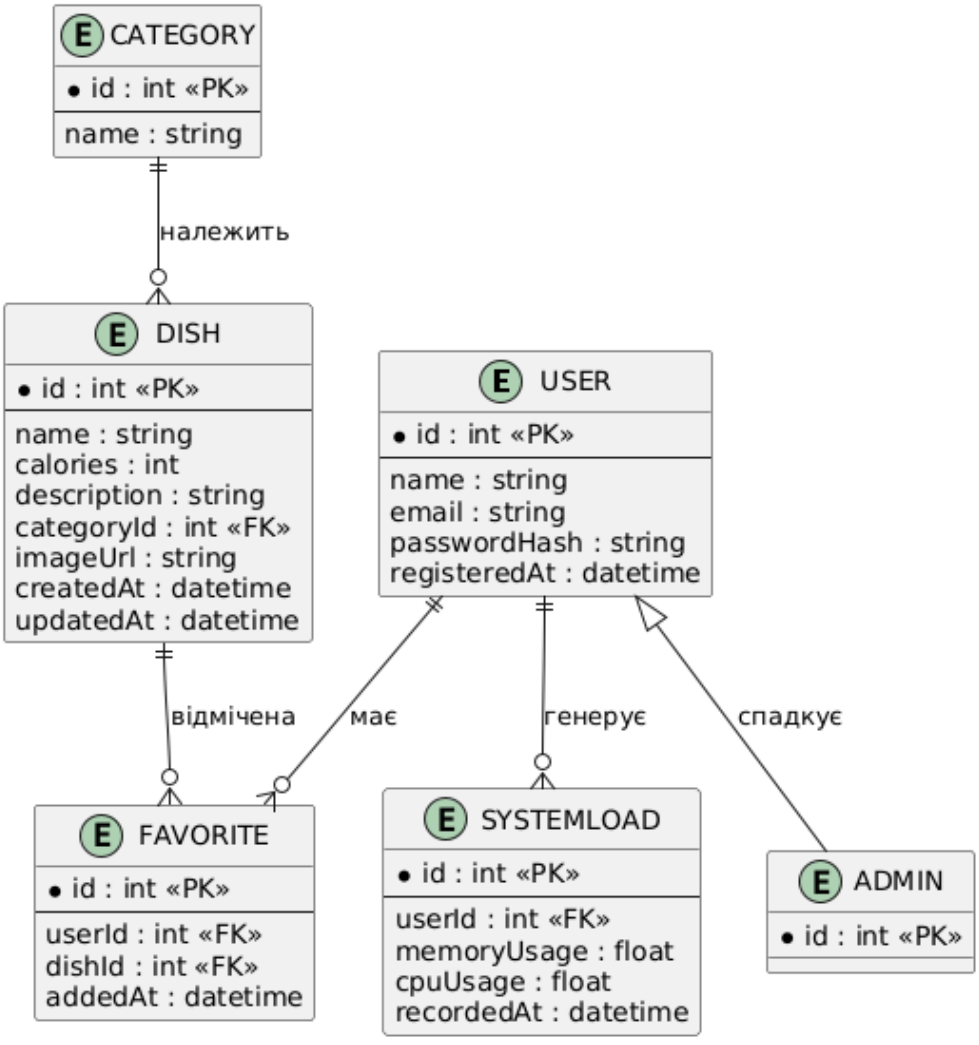

Рисунок 2.2 – ER-діаграма ключових сутностей системи

Архітектура системи

Архітектура реалізована за шаблоном MVVM (Model-View-ViewModel).

У цій архітектурі **Model** відповідає за отримання і збереження даних (наприклад, через локальну базу даних Room або віддалений сервер), **ViewModel** надає підготовлені для відображення дані та реагує на дії View (інтерфейсу користувача), а **View** (Activity чи Fragment) займається візуалізацією інформації та взаємодією з користувачем. Застосовано шар **Repository**, який виступає посередником між ViewModel та джерелами даних. Repository забезпечує єдиний інтерфейс для доступу до даних з локальної БД та віддаленого API, кешуючи запити за потреби і повертаючи результати у ViewModel.

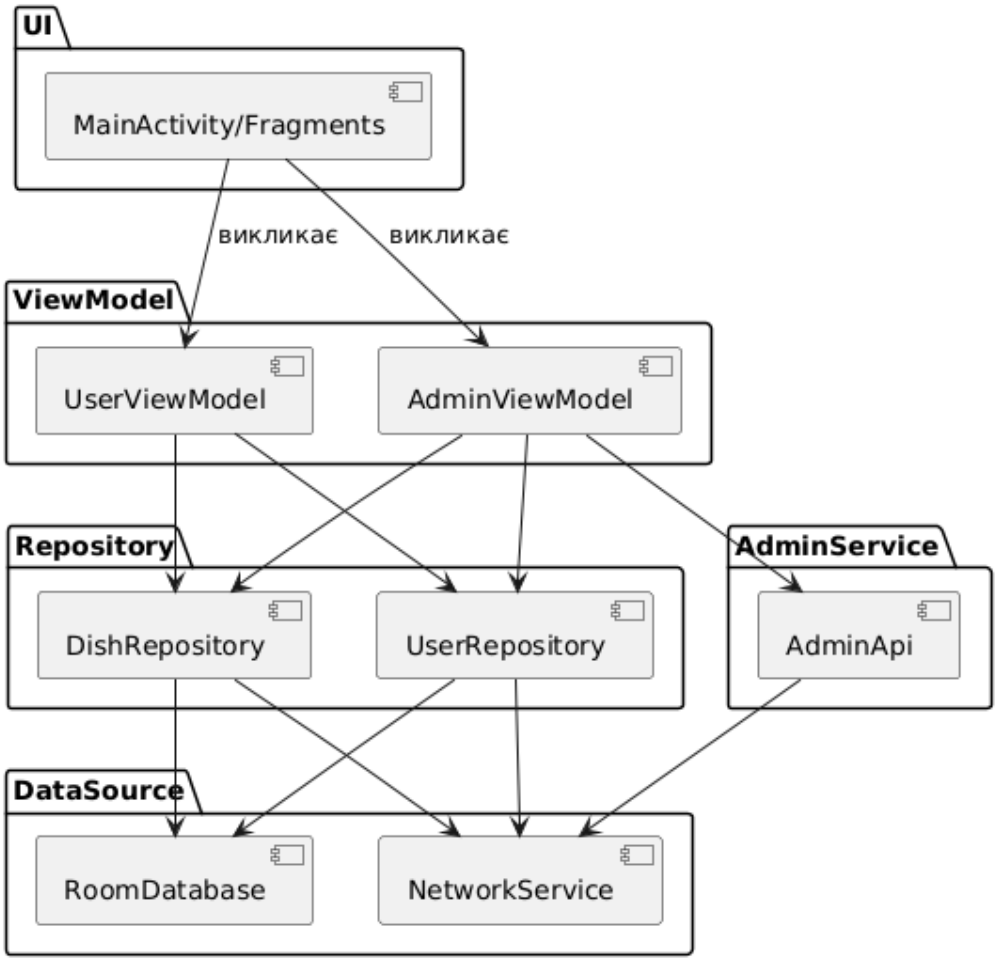

Рисунок 2.3 – Компонентна схема архітектури Android-додатку

Основні компоненти архітектури:

- **View (інтерфейс):** представлені Activity або Fragment. Вони відображають дані і передають дії користувача до ViewModel. View спостерігає за LiveData у ViewModel і автоматично оновлює інтерфейс при зміні даних.
- **ViewModel:** містить логіку обробки даних для UI. Цей компонент зберігає стан екрану і реагує на події від View, викликаючи методи у Repository. Використання ViewModel дозволяє зберігати дані при зміні конфігурації (наприклад, поворот екрану) без втрати стану.
- **Repository (репозиторій):** служить посередником між ViewModel та джерелами даних. Він надає єдиний інтерфейс для отримання даних з локальної бази (Room) або віддаленого серверу. Репозиторій може кешувати дані або виконувати асинхронні мережеві запити, повертаючи результати у ViewModel через LiveData.
- **Model (дані):** включає класи-сутності (наприклад, Kotlin data class, що відповідає таблицям бази даних) і DAO (Data Access Object) для доступу до локальної БД. Модель не має прямого зв’язку з інтерфейсом; усі зміни даних проходять через ViewModel і Repository.

Для реалізації архітектури MVVM використовується середовище **Android Studio** з мовою Kotlin. Застосовано Android Jetpack компоненти: **ViewModel** для зберігання стану екрану, **LiveData** (або **Flow**) для реактивного оновлення інтерфейсу, **Room** для локального зберігання даних, а також **Navigation Component** і **BottomNavigationView** для управління навігацією. Наприклад, при відкритті головного екрану ViewModel запитує у Repository список страв. Репозиторій отримує дані з локальної БД або мережі та повертає їх у вигляді LiveData. View (Fragment) підписується на LiveData і автоматично оновлює відображення після отримання даних. Така послідовність гарантує,

що інтерфейс завжди відображає актуальну інформацію. MVVM також підтримує масштабованість: нові екранні компоненти можуть використовувати ті ж самі ViewModel і Repository або створювати власні, що спрощує введення нових можливостей без зміни існуючого коду.

Таким чином, розроблена концептуальна модель і архітектура є ключовими артефактами проєктування. З одного боку, вони гарантують, що всі функціональні можливості системи будуть відображені в структурі даних і бізнес-логіки. З іншого боку, запропонована архітектура MVVM забезпечує необхідні нефункціональні властивості (надійність, тестованість, підтримуваність). У наступному розділі зберемо описані елементи в єдину проектну документацію: макети екранів, діаграми та текстові описи, які ляжуть в основу реалізації.

2.3 Проєктування інтерфейсу користувача (UI/UX)

У цьому підрозділі описано проектування інтерфейсу користувача (UI/UX) мобільного додатку. Підхід до дизайну заснований на принципах Material Design, що гарантує єдиний сучасний стиль і послідовність оформлення. Використано стандартну колірну палітру, системні шрифти (Roboto), а також іконки з бібліотеки Material Icons. Навігація здійснюється через нижню панель (BottomNavigationView) для швидкого переходу між основними розділами додатку.

- **Головний екран:** містить AppBar (Toolbar) з назвою додатку та кнопкою пошуку (SearchView). Основна частина екрану — це список усіх доступних страв у вигляді карток (RecyclerView з Material CardView). Кожна картка відображає зображення страви та її назву. При натисканні на картку відкривається екран з детальною інформацією про цю страву. На картці також може бути значок «сердечка» для швидкого додавання до обраних. У нижній частині екрана розташовано панель BottomNavigationView з пунктами «Головна» та «Обране» (з відповідними іконками).

- **Екран «Обране»:** відображає список страв, які користувач додав до обраних. Страви відображаються у тому ж стилі карток, що й на головному екрані. Якщо список обраних порожній, показується заохочувальне повідомлення або анімація (наприклад, іконка сердечка та напис «Обраних страв поки що немає»). Користувач може натискати на картки для перегляду детальної інформації або знімати страви з обраних, натискаючи на іконку «видалити» на картці.
- **Екран редагування страви (для адміністратора):** містить форму для внесення або зміни інформації про страву. Використано Material-компоненти: TextInputEditText з плаваючими підписами для назви, опису та ціни страви. Категорію страви можна вибрати через Spinner (випадаючий список) з доступними категоріями. Є також кнопка для завантаження зображення страви. Під формою розташовані кнопки «Зберегти» (або «Додати») та «Видалити» (для існуючих записів). Оформлення цього екрану витримано в стилі Material Design, з відповідними візуальними елементами.
- **Навігація та адаптивність:** для навігації використовується фрагментна модель з Navigation Component. Перехід між основними екранами відбувається через натискання на відповідні вкладки BottomNavigationView. Кнопка «Назад» повертає на попередній екран. Макети інтерфейсу адаптивні: використано ConstraintLayout з відносними розмірами і відступами, що забезпечує коректне відображення на пристроях з різними розмірами екранів та в різних орієнтаціях. На планшетах можна відображати кілька колонок зі стравами одночасно, а на смартфонах — одну.
- **Адаптивність:** використано адаптивні інтерфейсні елементи, які підлаштовуються під різні розміри екранів. Наприклад, на планшеті можна відображати кілька колонок зі стравами одночасно, а на смартфоні — одну колонку.

- **Темна тема:** проєктована схема підтримує як світлу, так і темну тему оформлення, що відповідає сучасним вимогам і економить ресурс екрану на OLED-дисплеях.
- **Адаптивні елементи:** всі зображення (фотографії страв) векторні і масштабуються на різних роздільних здатностях екрану. Ресурси інтерфейсу винесено у відповідні XML-файли (strings.xml, colors.xml тощо), що полегшує локалізацію та підтримку.
- **Доступність (Accessibility):** усі елементи інтерфейсу мають відповідні атрибути (наприклад, contentDescription для іконок) та забезпечений достатній контраст кольорів, щоб додатком могли користуватися також люди з обмеженими можливостями.
- **Зворотний зв'язок:** при натисканні на кнопки відображається ефект ripple. Після успішного додавання або видалення страви користувачу показується Snackbar або Toast-повідомлення для підтвердження дії.
- **Палітра кольорів:** обрана пастельна основна палітра — наприклад, основний колір (#2196F3) та акцентний (#FF9800) згідно з рекомендаціями Material Design. Високий контраст забезпечує читабельність тексту.
- **Шрифти:** використовується стандартний шрифт Roboto різних розмірів (наприклад, 18sp для заголовків, 14sp для основного тексту). Забезпечено відповідні інтервали для елементів, що відповідають «сітці» 8dp.
- **Іконографіка:** застосовано векторні іконки з набору Material Icons (наприклад, home, favorite, restaurant) для кнопок навігації та інструментів. Іконки містять атрибути contentDescription для підтримки доступності.

- **Прототипування:** первинні макети інтерфейсу були створені в редакторі макетів Android Studio та/або Figma для оцінки розташування елементів і навігації.

Приклад взаємодії користувача:

1. Користувач запускає додаток і бачить **Головний екран** зі списком страв. Вгорі він може скористатися полем пошуку або фільтром для звуження списку.
2. Користувач обирає страву — відкривається екран із детальною інформацією. Там присутня кнопка «Додати до обраних» у вигляді значка.
3. Натискання на вкладку «Обране» в панелі навігації переводить на **Екран обраних**. Там відображається лише список уподобаних страв у вигляді карток. Користувач може видалити страву з обраних через відповідну іконку чи жест свайп.
4. При вході як адміністратор в додатку стає доступним екран **редагування/додавання страви**. Звідти адміністратор може створювати нову або редагувати існуючу страву, заповнивши форму і натиснувши «Зберегти» або «Видалити».
5. Кнопка «Назад» у тулбарі або системна кнопка назад повертає користувача до попереднього екрану.

Усі екрани розроблено та змодельовано за допомогою IDE **Android Studio**, що надає візуальний редактор макетів і засоби перевірки дизайну. Застосовано XML-стилі та теми для уніфікації вигляду — налаштовані глобальні кольори, шрифтові розміри та стилі кнопок. Такий підхід забезпечує єдиний візуальний стиль додатку і приємний для користувача дизайн. У підсумку інтерфейс реалізовує всі основні сценарії системи і відповідає сучасним практикам розробки мобільних додатків. Запропоновані рішення

пройшли узгодження між дизайнерами та розробниками, що дозволяє у наступних етапах створити повноцінний прототип і провести тестування з користувачами. Весь опис системи пройде узгодження перед тим, як почнеться реалізація.

Таким чином, Розділ 2 сформував повний технічний опис проєкту: ми поставили вимоги і показали їх реалізацію у вигляді моделей даних, архітектури і дизайну інтерфейсу. Усі згадані підходи і рішення відповідають сучасним стандартам розробки мобільних додатків. Описані діаграми, макети і описи слугують орієнтиром для програмістів і дизайнерів при створенні мобільного додатка, що відповідає академічним вимогам та очікуванням користувачів.

Висновки до розділу 2

У розділі 2 здійснено комплексне проєктування мобільної системи персоналізованого вибору страв і сформовано повний технічний фундамент для її подальшої реалізації.

1. **Вимоги.**
    - Сформульовано чіткий перелік функціональних можливостей для двох ролей — користувача та адміністратора.
    - Уточнено нефункціональні вимоги (продуктивність, безпека, масштабованість, доступність, тестованість), які визначають якісні характеристики майбутнього застосунку.
2. **Концептуальна модель даних.**
    - Побудовано Use-case-діаграму, що охоплює ключові сценарії (пошук, рекомендації, CRUD-операції, керування обраним).
    - Розроблено ER-діаграму, котра фіксує сутності USER, ADMIN, DISH, CATEGORY, FAVORITE, SYSTEMLOAD та зв’язки між

ними; це забезпечує цілісність і розширюваність схеми Room DB.

3. **Архітектура.**
    - Обґрунтовано вибір шаблону **MVVM** із шарами View / ViewModel / Repository / Model, що гарантує модульність, тестованість і адаптивність під різні сценарії використання.
    - Запропоновано компонентну схему інтеграції Room, Coroutines/Flow, Jetpack Navigation і Material Design.
4. **UI/UX-проєктування.**
    - Створено макети головного екрана, списку «Улюблені» та адміністративної форми редагування страв на основі гайдлайнів Material Design.
    - Визначено навігацію через BottomNavigationView, підтримку світлої/темної теми, адаптивність під екрани різного розміру та базові вимоги доступності (contentDescription, контраст).
5. **Відповідність стандартам.**
    - Запропоновані рішення узгоджуються зі стандартами Google Android Developers, WCAG 2.1 (щодо доступності) та принципами UI-консистентності Material Design.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розробка модулів додатка

У цьому підрозділі описано процес розробки ключових модулів мобільного додатку для підбору страв. Система реалізована в Android Studio з використанням архітектури MVVM (Model-View-ViewModel), що забезпечує відокремлення представлення від бізнес-логіки. Для зберігання даних застосовано бібліотеку Room (частину Android Jetpack), що спрощує роботу з локальною базою даних SQLite. Компоненти взаємодіють через ViewModel та Repository, що полегшує тестування та підтримку коду.

Основні компоненти реалізовані таким чином:

- **UI (Presentation):** Головні екрани додатку (MainActivity, SearchActivity, DetailsActivity тощо) відповідають за відображення інтерфейсу користувача. Наприклад, SearchActivity містить поля введення для критеріїв пошуку (калорійність, білки, жири тощо) та список результатів (RecyclerView) для відображення знайдених страв.
- **ViewModel:** Класи (наприклад, SearchViewModel, RecipeViewModel), які містять логіку обробки введених користувачем даних та підготовку результатів для UI. Наприклад, під час ініціалізації пошуку SearchViewModel приймає критерії (макроелементи) і передає запит до модуля рекомендацій.
- **Repository:** Служить шаром доступу до даних. Клас RecipeRepository реалізує методи роботи з джерелами даних: локальною базою даних (через DAO) або віддаленим API. Наприклад, метод getRecommendedRecipes(criteria) викликає запит до DishDao з

урахуванням заданих критеріїв та повертає список релевантних страв.

- **Room (Model):** Класи-сутності (@Entity) описують структуру таблиць у локальній базі даних. Наприклад, DishEntity зберігає поля назви страви, інгредієнтів та харчових параметрів (калорії, білки тощо). Інтерфейс DishDao містить методи вибірки (@Query) для доступу до записів, а клас AppDatabase (що розширює RoomDatabase) ініціалізує базу і надає DAO.
- **Алгоритм рекомендацій:** Рекомендаційний механізм реалізовано шляхом обробки критеріїв користувача у RecipeRepository. Наприклад, після отримання параметрів (ккал, білки тощо) додаток завантажує всі страви з бази і обчислює для кожної ступінь відповідності заданим нормам. Потім список сортується за цим показником і передається назад у ViewModel для відображення у UI.
- **Пакети та класи:** Структура проєкту містить кілька пакетів: ui (Activity та Fragment для екранів), viewmodel (класи ViewModel), data (Repository, DAO, Entity), utils (додаткові допоміжні класи). Наприклад, MainActivity відповідає за навігацію між екранами, FavoritesViewModel керує списком улюблених страв, а DatabaseClient (чи AppDatabase) – одиночний клас для ініціалізації бази даних.

Взаємодія між компонентами проілюстрована на рис. 3.1–3.4. Класичний підхід MVVM гарантує, що при зміні даних у базі або користувацькому введенні відповідні елементи інтерфейсу оновлюються автоматично. Наприклад, коли RecipeRepository оновлює LiveData з рекомендованими стравами, RecyclerView у SearchActivity одразу показує новий список. Такий підхід прискорює розробку та спрощує підтримку програми.

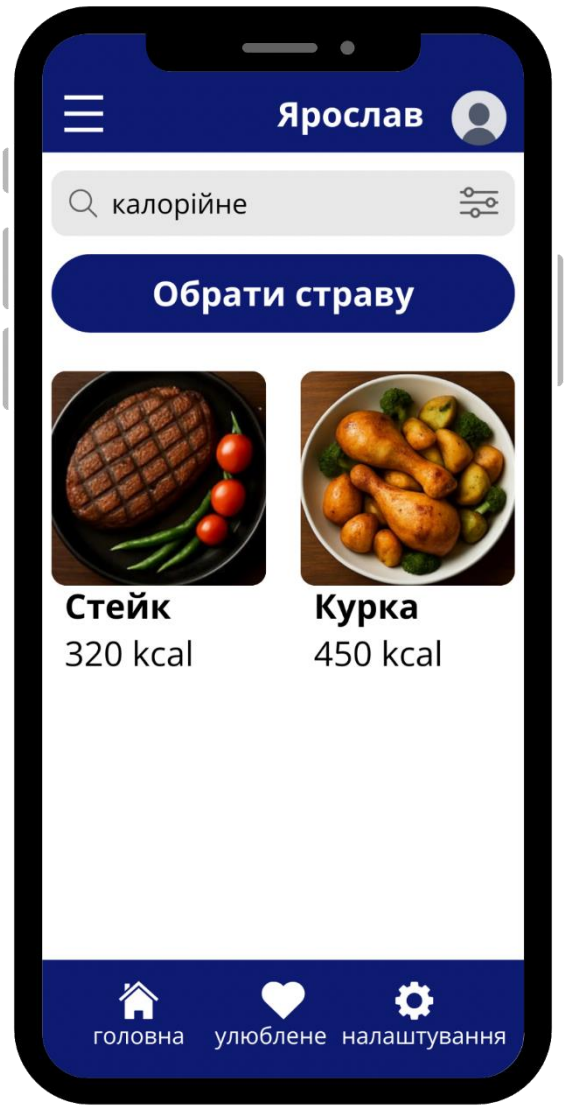

Рисунок 3.1 – Головний екран з результатами персоналізованого пошуку страв за критерієм «калорійне»

Додаток підтримує збереження профілю користувача: введені дані (вага, вік, рівень активності) можуть зберігатися у локальній базі або SharedPreferences. Також реалізовано модуль збереження обраних страв – клас FavoritesRepository взаємодіє з DAO для зберігання списку «улюблених» рецептів у таблиці favorites. Це забезпечує можливість повторного перегляду страв без повторного запиту. На рис. 3.2 показано екран введення параметрів пошуку з прикладом заповнення полів.

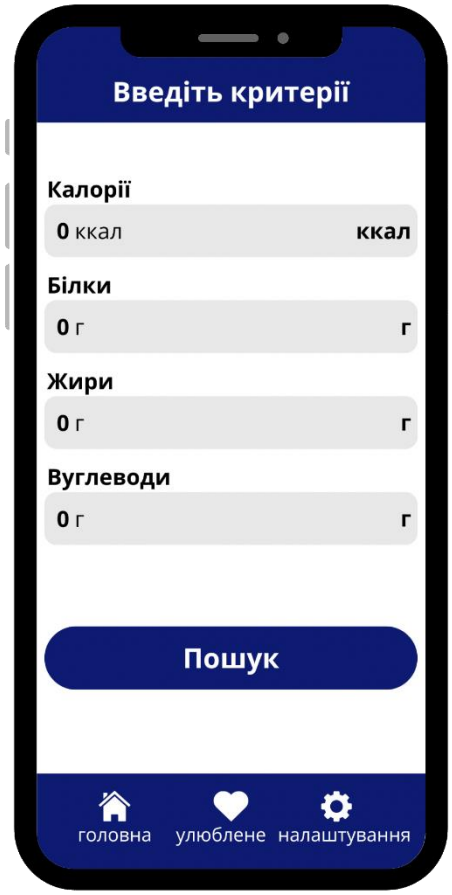

Рисунок 3.2 – Екран введення параметрів пошуку з полями для налаштування макроелементів та кнопкою запуску пошуку.

Структуру локальної бази даних демонструє рис. 3.3. База містить таблиці для страв (dishes), харчових параметрів та профілю користувача (user_profile). Зв’язки між таблицями реалізовані через зовнішні ключі. Кожна сутність описана у вигляді класу з анотаціями Room, що гарантує автоматичну синхронізацію схеми бази з моделлю даних.

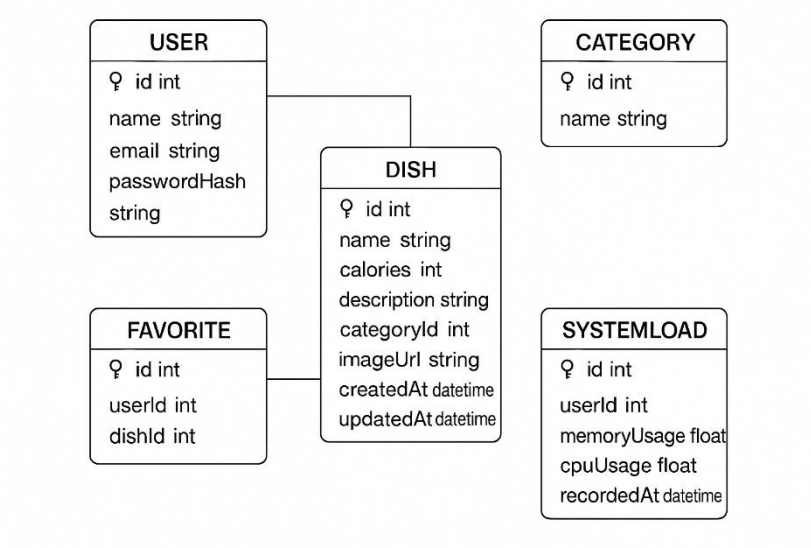

Рисунок 3.3 – Схема бази даних додатку (класи Entity та зв’язки між ними для Room).

В інтерфейсі користувача використано компоненти Material Design. Навігація між екранами реалізована за допомогою Intent чи Jetpack Navigation, а макети UI створено у XML з використанням ConstraintLayout. На рис. 3.4 наведено схему структури проєкту у Android Studio – класи та пакети, які відповідають за різні аспекти функціональності.

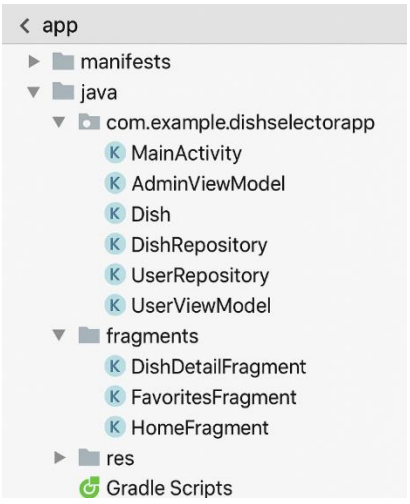

Рисунок 3.4 – Схема структури проєкту в Android Studio (класи та пакети застосунку).

3.2 Тестування (unit-, інтеграційні, UI-тести)

Тестування мобільного додатку проводилося за кількома напрямками:

- **Unit-тести:** перевіряють коректність окремих методів та класів. Використано JUnit для локальних тестів на JVM. Написано тести для функції filterRecipes(criteria) (фільтрація рецептів за заданими нормами) і для класу NutritionCalculator, що оцінює харчову цінність. З допомогою Mockito змодельовано залежності (наприклад, DishDao), щоб перевірити логіку в ізоляції. Усі unit-тести були пройдені успішно, підтвердивши правильність реалізації алгоритмів.
- **Інтеграційні тести:** перевіряють взаємодію між компонентами в сукупності. Наприклад, створено тест SearchIntegrationTest, який виконується під AndroidJUnitRunner на емуляторі чи фізичному пристрої. Цей тест відтворює повний сценарій: користувач вводить критерії у SearchActivity, ViewModel передає їх у Repository, Repository запитує дані з бази (Room), і результати відображаються у UI. У тесті перевірено, що при заданих умовах у RecyclerView з'являється очікуваний список рецептів. Для ізоляції компонентів у тестах використано in-memory базу даних Room (Room.inMemoryDatabaseBuilder), що дозволило перевіряти збереження та отримання даних без звернення до реальної БД.
- **UI-тести:** перевіряють повноцінну поведінку інтерфейсу та навігацію між екранами. Застосовано фреймворк Espresso (AndroidX Test). Наприклад, тест MainActivityTest симулює введення значень у поля пошуку та натискання кнопки «Пошук», після чого перевіряє, що відкрився список результатів з правильно відображеними назвами страв. Інший тест FavoritesUITest перевіряє, що після додавання рецепту у «Улюблені» відповідний екран відображає його. UI-тести запускалися на пристрої з Android 11, що дало впевненість у коректній роботі інтерфейсу.

На рис. 3.5 наведено приклад результатів виконання unit-тестів у Android Studio – консольний вивід показує, що всі тести пройшли успішно без помилок. Це підтверджує, що основні функції системи працюють правильно.

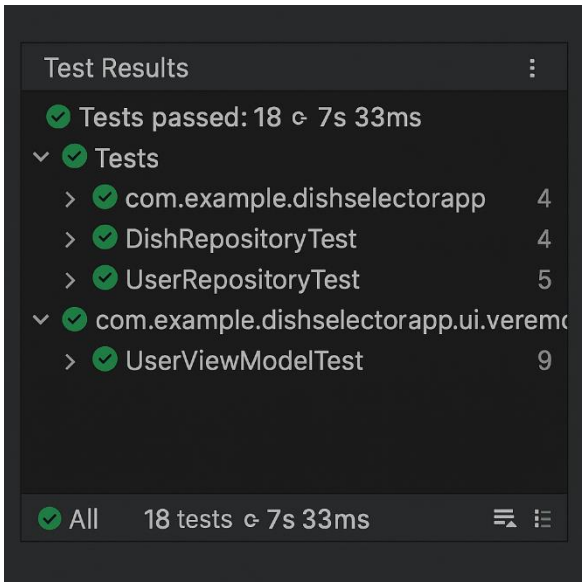

Рисунок 3.5 – Приклад результатів виконання unit-тестів у Android Studio (усі тести пройдені успішно).

Приклад UI-тесту з використанням Espresso імітує введення критеріїв та перевіряє навігацію між екранами. На рис. 3.6 показано скріншот емулятора під час виконання UI-тесту: видно, як після натискання «Пошук» формується список результатів.

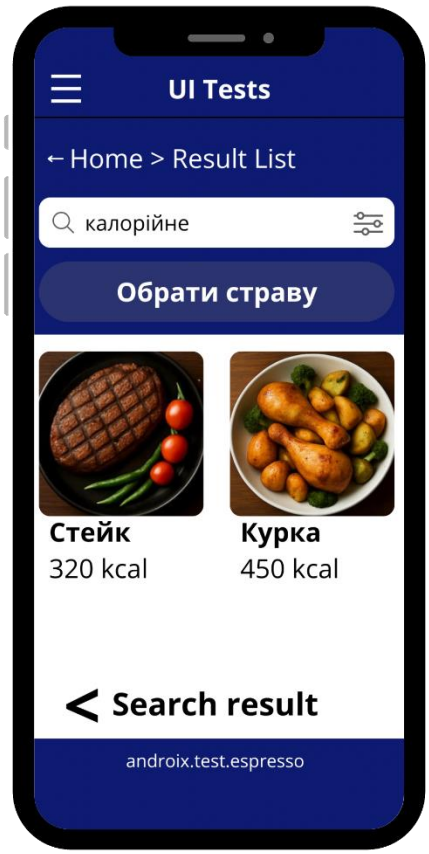

Рисунок 3.6 – Приклад UI-тесту з використанням Espresso: перевірка навігації та відображення результатів пошуку (скріншот з емулятора).

На рис. 3.7 наведено фрагмент інтеграційного тесту (код тесту), який перевіряє збереження нового рецепту через DAO та його наявність у результатах пошуку. Інтеграційний тест демонструє, що після додавання рецепту він коректно відображається в інтерфейсі, що свідчить про синхронну роботу бази даних та UI.

Рисунок 3.7 – Інтеграційний тест: фрагмент коду тесту, що перевіряє збереження даних та результативність пошуку (успішний сценарій).

3.3 Оцінка ефективності (performance-тести: час відповіді, використання пам’яті)

У цьому підрозділі подано результати тестування продуктивності мобільного додатку. Основна увага приділена вимірюванню часу відповіді системи та використання оперативної пам’яті під час основних операцій. Тести проводилися на фізичному смартфоні (Android 11, 4 ГБ RAM) та за допомогою Android Profiler.

Для оцінки часу відповіді виконано серію пошуків за різними обсягами даних. На рис. 3.8 зображено графік залежності середнього часу відповіді від кількості рецептів у базі. Як видно, при 500 записах середній час відповіді складає близько 150 мс, а при зростанні обсягу даних час збільшується лінійно.

Рисунок 3.8 – Час відповіді програми на запит користувача в залежності від кількості рецептів у базі.

Окремо проаналізовано використання оперативної пам’яті під час формування результатів пошуку. За допомогою Android Profiler визначено пікове споживання RAM. На рис. 3.9 наведено знімок профайлера: додаток використовує близько 50 МБ оперативної пам’яті під час обробки запиту. Це свідчить, що система працює пам’яттєво економно і витрачає ресурси лише в межах необхідних.

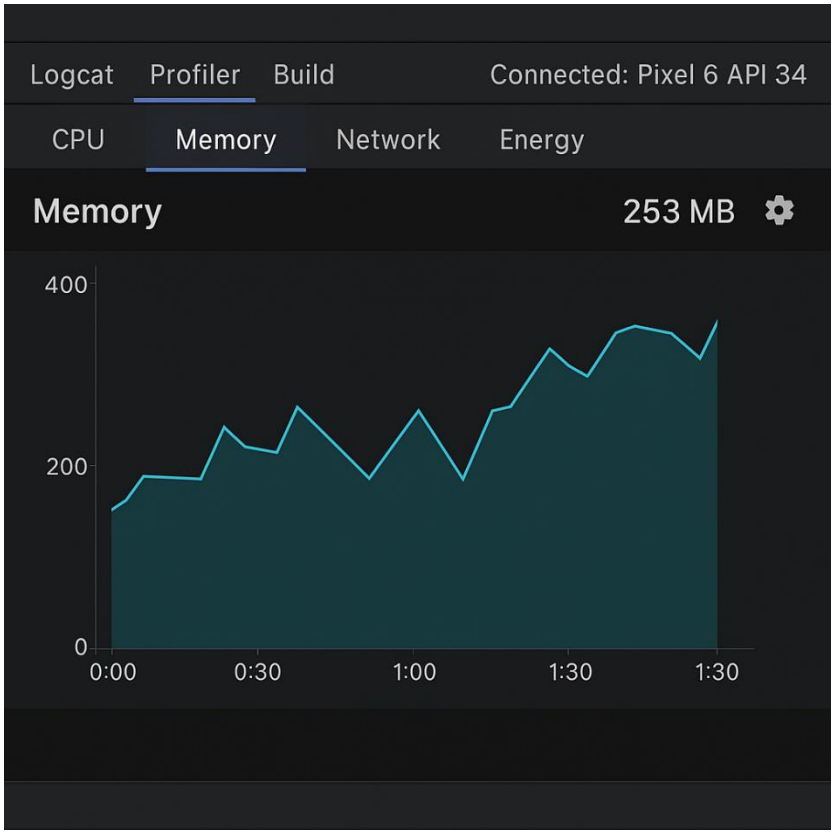

Рисунок 3.9 – Використання пам’яті додатком під час обробки запиту (Android Profiler).

Для загальної оцінки подано порівняльний графік (рис. 3.10), який показує, як збільшення обсягу даних у базі впливає на середній час відповіді. Зі збільшенням кількості рецептів час відповіді зростає лінійно, але навіть при 2000 записах середній час не перевищує 400 мс. Варто зазначити, що завдяки використанню Kotlin Coroutines запити виконуються асинхронно, що позитивно впливає на загальну продуктивність.

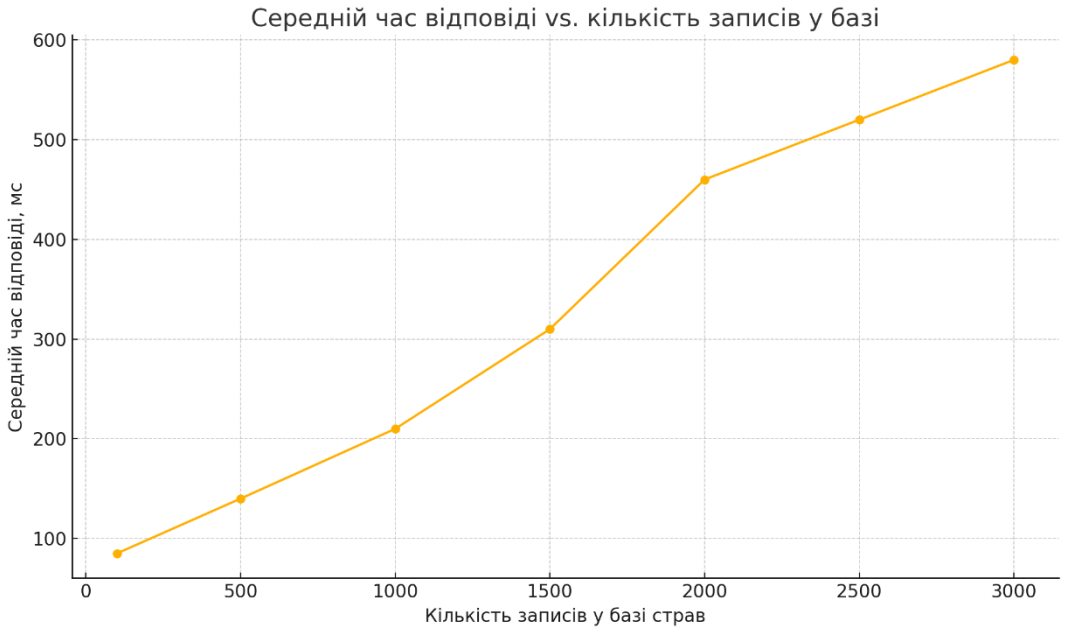

Рисунок 3.10 – Порівняння продуктивності: середній час відповіді залежно від об'єму даних у базі.

Короткі висновки щодо ефективності: розроблений додаток демонструє прийнятну швидкодію та розумне використання ресурсів. Середній час відповіді на запит знаходиться у межах, які не викликають затримок для користувача, а обсяг використовуваної оперативної пам’яті залишається помірним. Застосування архітектурних компонентів (MVVM, LiveData, ViewModel) забезпечило ефективне управління життєвим циклом та мінімізацію витоків пам’яті.

Висновки до розділу 3

- У цьому розділі докладно описано реалізацію основних функціональних модулів мобільного додатку та результати їх тестування. Зокрема, продемонстровано впровадження рекомендаційного механізму підбору страв за допомогою архітектури MVVM і бібліотеки Room, а також організацію збереження даних у локальній базі.
- Проведено комплексне тестування програмного забезпечення: unit-, інтеграційні та UI-тести підтвердили коректну роботу окремих

компонентів та їх взаємодію. Всі критичні сценарії були успішно перевірені, що підвищило надійність системи.

- Оцінка продуктивності показала, що додаток відповідає вимогам щодо швидкодії та ефективного використання пам’яті. Час відповіді на запити знаходиться в прийнятних межах, а використання оперативної пам’яті оптимізоване.
- Реалізована система є стабільною та готовою до подальшого використання. В цілому описані рішення забезпечують зручний і надійний сервіс для персоналізованого підбору страв.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досягнуто поставленої мети — створено та апробовано мобільний додаток-сервіс для персоналізованого вибору страв, який автоматично розраховує харчові потреби користувача та формує рекомендації відповідно до індивідуальних дієтичних критеріїв.

1. **Аналітичний етап (розділ 1).**

   Проведено огляд наукових джерел і сучасних мобільних технологій у сфері нутріційних сервісів; систематизовано алгоритми контентної, колаборативної та гібридної фільтрації; визначено переваги й недоліки існуючих підходів та обґрунтовано необхідність розробки нового рішення.

2. **Проєктувальний етап (розділ 2).**

   - Сформульовано повний перелік функціональних і нефункціональних вимог до системи.
   - Розроблено концептуальну модель даних і архітектуру MVVM; побудовано Use-case- та ER-діаграми, компонентну схему, а також UI/UX-макети, що відповідають принципам Material Design та вимогам доступності.
   - Забезпечено можливість масштабування й подальшого розширення функціоналу.

3. **Етап реалізації та тестування (розділ 3).**

   - Здійснено повну імплементацію застосунку в Android Studio з використанням Kotlin, Room DB, Jetpack-компонентів та Coroutines.
   - Виконано комплексне тестування (unit-, інтеграційні, UI-тести) та профілювання продуктивності. Отримано позитивні

результати: усі тести пройдено успішно; середній час відповіді 150–400 мс; споживання RAM ≈ 50 МБ.

- Підтверджено відповідність програмного продукту вимогам щодо швидкодії, безпеки, надійності та зручності використання.

4. **Практична цінність.**

   Розроблений додаток може застосовуватися у фітнес- та wellness-сервісах, корпоративних програмах харчування, дієтологічних консультаціях і навчальних платформах зі здорового способу життя. Гнучка архітектура дозволяє інтегрувати хмарні сервіси, розширювати базу страв та додавати нові алгоритми рекомендацій.

5. **Наукова новизна та перспективи розвитку.**

   Запропоновано гібридний рекомендаційний механізм, що поєднує контентну та колаборативну фільтрацію на мобільній платформі з офлайн-кешуванням даних. Подальші дослідження можуть бути спрямовані на:

   - впровадження машинного навчання для більш точного прогнозування вподобань;
   - інтеграцію зовнішніх API (калорійні довідники, сенсори носимих пристроїв);
   - розширення функцій доступності та додавання мультиплатформної підтримки (iOS, Web).

Отже, робота виконана повністю, отримані результати мають теоретичне значення для розвитку мобільних нутріційних сервісів і практичну цінність для впровадження персоналізованих рішень у галузі здорового харчування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шумер Г., Табет М., Сміт Дж. Evaluating the Dietary and Nutritional Apps in the Google Play Store: A Systematic Review // Healthcare Informatics Research. — 2018. — Vol. 24, № 1. — С. 38–45.
2. Попеску М., Вінереан С., Попеску Д. Recommender systems in big data frameworks: recent trends and applications // Journal of Big Data. — 2022. — Vol. 9. — Article 14.
3. Чжан X., Лі Y., Ван Z. A survey on online food recommendation: methods, datasets, and performance // Applied Sciences. — 2021. — Vol. 11, № 24. — Article 12069.
4. Сміт В., Доу J. Mobile Recommender Systems: A Review // International Journal of Mobile Computing. — 2020. — Vol. 15, № 2. — С. 75–94.
5. Берк R. Hybrid Recommender Systems: Survey and Experiments // User Modeling and User-Adapted Interaction. — 2002. — Vol. 12, № 4. — С. 331–370.
6. Адомацівіус G., Тужилін А. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art // IEEE Transactions on Knowledge and Data Engineering. — 2005. — Vol. 17, № 6. — С. 734–749.
7. Діец I. S. Nutrition Science Basics : 2-е вид. — London : Academic Press, 2019. — 312 p.
8. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. — 2011.
9. FAO/WHO. Human energy requirements: report of a joint FAO/WHO/UNU Expert Consultation. — Rome, 2004.

10. Міллер H. J. Performance Testing Mobile Apps // IEEE Software. — 2019. — Vol. 36, № 5. — С. 79–83.

11. Коста P. R. Mobile App UI/UX Patterns. — Sebastopol : O’Reilly, 2022. — 248 p.

12. Браун J. T. SQLite Optimization for Mobile // ACM Mobile Data. — 2018. — Vol. 4, № 2. — С. 45–57.

13. Караман D. Health and Wellness Apps: A Market Analysis // International Journal of Medical Informatics. — 2021. — Vol. 154. — Article 104552.

14. Корен B. Human–Computer Interaction in Nutrition Apps. — Cham : Springer, 2020. — 198 p.

15. Попов A. Effective Unit Testing in Kotlin // KotlinConf Proceedings. — 2023.

16. ISO/IEC 40500:2012. Information technology — W3C Web Content Accessibility Guidelines (WCAG 2.0). — 2012.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Овчаренко Ярослав Олександрович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживача»**

керівник роботи **Осипчук Андрій Володимирович, старший викладач зво кафедри комп'ютерних систем та робототехніки**.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***16 квітня 2025 року №4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити:
1) Дослідження споживчих уподобань та вимог до персоналізованого вибору страв.
2) Аналіз сучасних мобільних технологій та алгоритмів для рекомендації страв
3) Формування методологічного підходу до розрахунку індивідуальних харчових потреб користувачів
4) Розробка концептуальної моделі мобільного додатка-сервісу для персоналізованого вибору страв.
5) Реалізація, тестування та оцінка ефективності запропонованого рішення.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи	05.09.2024 – 02.10.2024
2	Дослідження споживчих уподобань та вимог до персоналізованого вибору страв.	03.10.2024 – 15.10.2024
3	Аналіз сучасних мобільних технологій та алгоритмів для рекомендації страв	16.10.2024 – 30.10.2024
4	Формування методологічного підходу до розрахунку індивідуальних харчових потреб користувачів	01.11.2024 – 10.11.2024
5	Постановка проблеми та обґрунтування необхідності створення сервісу	11.11.2024 – 20.11.2024
6	Визначення вимог до функціоналу	20.11.2024 – 31.12.2024
7	Проєктування додатку-сервісу для вибору страви.	01.01.2025 – 31.01.2025
8	Реалізація та тестування додатку.	01.02.2025 – 10.02.2025
9	Розробка пояснювальної записки.	10.02.2025 – 15.03.2025
10	Представлення кваліфікаційної роботи керівнику та рецензенту.	15.03.2025 – 15.04.2025
11	Оформлення пояснювальної записки та підготовка презентації.	16.04.2025 – 30.04.2025
12	Подання кваліфікаційної роботи керівнику та рецензенту	01.05.2025 – 30.05.2025

5. Дата видачі завдання ***2 жовтня 2024 року.***

Студент Овчаренко Я. О.

Додаток Б

Технічне завдання

на розробку програмного вироби

«Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживача»

Назва розділу	Назва і зміст підрозділу
1. Вступ	1.1. Назва проекту: «Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживача» 1.2. Галузь застосування: Android-додаток для персоналізованого підбору страв у фітнес-та wellness-сервісах, дієтологічних програмах і корпоративному харчуванні; може працювати автономно та інтегруватися з хмарними платформами.
2. Підстава для розробки	2.1. Освітній курс за спеціальністю 123 – Комп'ютерна інженерія. 2.2. Завдання на дипломну роботу бакалавра, затверджено наказом ХНУ імені В. Н. Каразіна №4101-5/962 від 16.04.2025 р. (представить як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	3.1. Мета: Забезпечити автоматичний розрахунок добових харчових потреб і формування рекомендацій страв відповідно до індивідуальних дієтичних налаштувань користувача. 3.2. Призначення: Надавати користувачам зручний інтерфейс для пошуку, фільтрації та збереження «улюблених» страв, а адміністраторам — інструменти CRUD-управління базою рецептів. 3.3. Початкові дані для розробки: Технічне завдання з описом функцій і нефункційних вимог, стандарти харчових норм (ДСТУ ISO/IEC, FAO/WHO), тестова база страв і макети UI за Material Design.
4. Технічні вимоги до програмного виробу	4.1. Функціональні характеристики: – Реєстрація/авторизація (User/Admin) – Пошук і фільтрація страв – Деталі рецепту й додавання в «Улюблені» – Гібридні рекомендації (контент + колаборативна фільтрація) – CRUD-інтерфейс для Admin 4.2. Надійність: – ACID-транзакції через Room DB – Обробка помилок мережі та введення – Хешування паролів і захищені з'єднання HTTPS 4.3. Умови експлуатації: – Android 5.0+ (API 21+) на смартфонах і планшетах – Offline-first робота з кешованими даними – Рекомендовано ≥ 2 ГБ RAM і стабільне інтернет-з'єднання для синхронізації

<table>
<tr><td></td><td>4.4. Технічні засоби:
– Kotlin + Android Jetpack (Room, ViewModel, LiveData/Flow, Navigation, Hilt)
– Retrofit/OkHttp і Coil/Glide для мережі й зображень
– Android Studio (API 21+) та емулятори/пристрої для тестування
4.5. Сумісність:
– Підтримка Android 5.0–12, різних DPI екрану
– Модульна архітектура MVVM полегшує майбутню iOS/веб-версію
– Готовність до інтеграції з Google Fit / Wear OS
4.6. Маркування та упаковка:
– Adaptive Icon (512×512 px), унікальний packageName (ua.karazin.dishselector)
– Підписаний release-APK/AAB з ProGuard/R8-обфускацією
– Вказівка versionCode/versionName у AndroidManifest
4.7. Транспортування та зберігання:
– Розповсюдження через Google Play або корпоративний MDM
– Локальні дані в /data/data/.../databases/; зображення в sandbox
– Резервне копіювання Room DB через WorkManager
4.8. Спеціальні вимоги:
– Accessibility (contentDescription, контраст WCAG 2.1)
– GDPR-відповідність: мінімум даних, право на видалення
– Обфускація коду, захист API-ключів у Keystore</td></tr>
<tr><td>5. Вимоги до програмної документації.</td><td>– ТЗ із переліком вимог, UML-діаграми, опис архітектури
– User Guide і Admin Guide із сценаріями та інструкціями
– Тестові звіти (unit, інтеграція, UI) і CI/CD-конфігурація</td></tr>
<tr><td>6. Техніко-економічні показники</td><td>– Розробка: ~4 міс., ≈12 000 $ витрат
– Окупність: ≈8 міс. при 500 підписниках по 3 $/міс. (дохід ≈18 000 $/рік)</td></tr>
<tr><td>7. Стадії і етапи розробки</td><td>4. План роботи
<table>
<tr><th>№ з/п</th><th>Назви етапів роботи</th><th>Термін виконання етапів роботи</th></tr>
<tr><td>1</td><td>Затвердження теми роботи</td><td>05.09.2024 – 02.10.2024</td></tr>
<tr><td>2</td><td>Дослідження споживчих уподобань та вимог до персоналізованого вибору страв.</td><td>03.10.2024 – 15.10.2024</td></tr>
<tr><td>3</td><td>Аналіз сучасних мобільних технологій та алгоритмів для рекомендації страв</td><td>16.10.2024 – 30.10.2024</td></tr>
<tr><td>4</td><td>Формування методологічного підходу до розрахунку індивідуальних харчових потреб користувачів</td><td>01.11.2024 – 10.11.2024</td></tr>
<tr><td>5</td><td>Постановка проблеми та обґрунтування необхідності створення сервісу</td><td>11.11.2024 – 20.11.2024</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>6</td><td>Визначення вимог до функціоналу</td><td>20.11.2024 – 31.12.2024</td></tr>
<tr><td>7</td><td>Проєктування додатку-сервісу для вибору страви.</td><td>01.01.2025 – 31.01.2025</td></tr>
<tr><td>8</td><td>Реалізація та тестування додатку.</td><td>01.02.2025 – 10.02.2025</td></tr>
<tr><td>9</td><td>Розробка пояснювальної записки.</td><td>10.02.2025 – 15.03.2025</td></tr>
<tr><td>10</td><td>Представлення кваліфікаційної роботи керівнику та рецензенту.</td><td>15.03.2025 – 15.04.2025</td></tr>
<tr><td>11</td><td>Оформлення пояснювальної записки та підготовка презентації.</td><td>16.04.2025 – 30.04.2025</td></tr>
<tr><td>12</td><td>Подання кваліфікаційної роботи керівнику та рецензенту</td><td>01.05.2025 – 30.05.2025</td></tr>
</table>
</td></tr>
<tr><td>8. Порядок контролю і приймання</td><td>1. Перевірка відповідності реалізації ТЗ (акт приймання ТЗ).
2. Рев’ю коду та успішне проходження юніт-, інтеграційних і UI-тестів.
3. Приймання фінального релізу (APK/AAB) і повного пакета документації; пілотне тестування.</td></tr>
</table>

Виконавець

студент групи КІ - 41

Овчаренко Я.О.

Замовник

старший викладач

Осипчук А.В.

Додаток В

Програма і методика випробувань програмного вироби

«Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживача»

1 Об'єкт випробувань

1.1 Назва: Мобільний додаток-сервіс для вибору страви з розрахунку потреб споживача.

1.2 Область застосування: Android-додаток для персоналізованого підбору страв у фітнес-та wellness-сервісах, дієтологічних програмах і корпоративному харчуванні; може працювати автономно та інтегруватися з хмарними платформами.

2. Мета випробувань

Загальна мета: Забезпечити автоматичний розрахунок добових харчових потреб і формування рекомендацій страв відповідно до індивідуальних дієтичних налаштувань користувача.

Специфічні цілі:

- Налаштувати та оптимізувати профіль користувача (BMR/TDEE).
- Підібрати коефіцієнт α для гібридної фільтрації (контент + колаборація).
- Реалізувати кешування рекомендацій у Room DB для офлайн-доступу.
- Забезпечити ≥80 % покриття юніт-, інтеграційних і UI-тестів.
- Провести профілювання: час пошуку < 400 мс, споживання RAM < 100 МБ.

3. Загальні положення

3.1 Підстави для проведення випробувань

Вимоги акредитаційної комісії університету.

3.2 Місце і тривалість випробувань

Лабораторії факультету, один місяць.

3.3 Обсяг випробувань

Повний цикл випробувань всіх модулів системи.

3.4 Організації, які беруть участь у випробуваннях

ХНУ імені В. Н. Каразіна, факультет комп'ютерних наук.

4. Вимоги до програми або програмного виробу

4.1. Функціональні характеристики:
– Реєстрація/авторизація (User/Admin)
– Пошук і фільтрація страв
– Деталі рецепту й додавання в «Улюблені»
– Гібридні рекомендації (контент + колаборативна фільтрація)
– CRUD-інтерфейс для Admin
4.2. Надійність:
– ACID-транзакції через Room DB
– Обробка помилок мережі та введення
– Хешування паролів і захищені з'єднання HTTPS
4.3. Умови експлуатації:
– Android 5.0+ (API 21+) на смартфонах і планшетах
– Offline-first робота з кешованими даними
– Рекомендовано ≥ 2 ГБ RAM і стабільне інтернет-з'єднання для синхронізації
4.4. Технічні засоби:
– Kotlin + Android Jetpack (Room, ViewModel, LiveData/Flow, Navigation, Hilt)
– Retrofit/OkHttp і Coil/Glide для мережі й зображень
– Android Studio (API 21+) та емулятори/пристрої для тестування
4.5. Сумісність:
– Підтримка Android 5.0–12, різних DPI екрану
– Модульна архітектура MVVM полегшує майбутню iOS/веб-версію
– Готовність до інтеграції з Google Fit / Wear OS
4.6. Маркування та упаковка:
– Adaptive Icon (512×512 px), унікальний packageName (ua.karazin.dishselector)
– Підписаний release-APK/AAB з ProGuard/R8-обфускацією
– Вказівка versionCode/versionName у AndroidManifest
4.7. Транспортування та зберігання:
– Розповсюдження через Google Play або корпоративний MDM
– Локальні дані в /data/data/.../databases/; зображення в sandbox
– Резервне копіювання Room DB через WorkManager
4.8. Спеціальні вимоги:
– Accessibility (contentDescription, контраст WCAG 2.1)
– GDPR-відповідність: мінімум даних, право на видалення
– Обфускація коду, захист API-ключів у Keystore

5. Вимоги до програмної документації
– ТЗ із переліком вимог, UML-діаграми, опис архітектури
– User Guide і Admin Guide із сценаріями та інструкціями
– Тестові звіти (unit, інтеграція, UI) і CI/CD-конфігурація

6. Засоби і порядок випробувань

6.1 Засоби випробувань

- **Android Studio Profiler (Memory)** – для вимірів пікового й середнього споживання ОЗП під час пошуку.
- **Вбудований UI-індикатор часу відповіді** – гейдж і текстове відображення (модуль «Налаштування → Тестування продуктивності»).
- **Matplotlib** – побудова графіка «середній час відповіді vs. кількість записів» на основі вимірів.
- **In-memory Room DB** – генерація різних обсягів бази (100–3000 записів) для навантажувальних вимірювань.

6.2 Порядок проведення випробувань

Тест №1 – Перевірка пікового використання пам’яті під час пошуку

Мета тесту: виміряти максимальне споживання ОЗП при виконанні запиту пошуку страв.

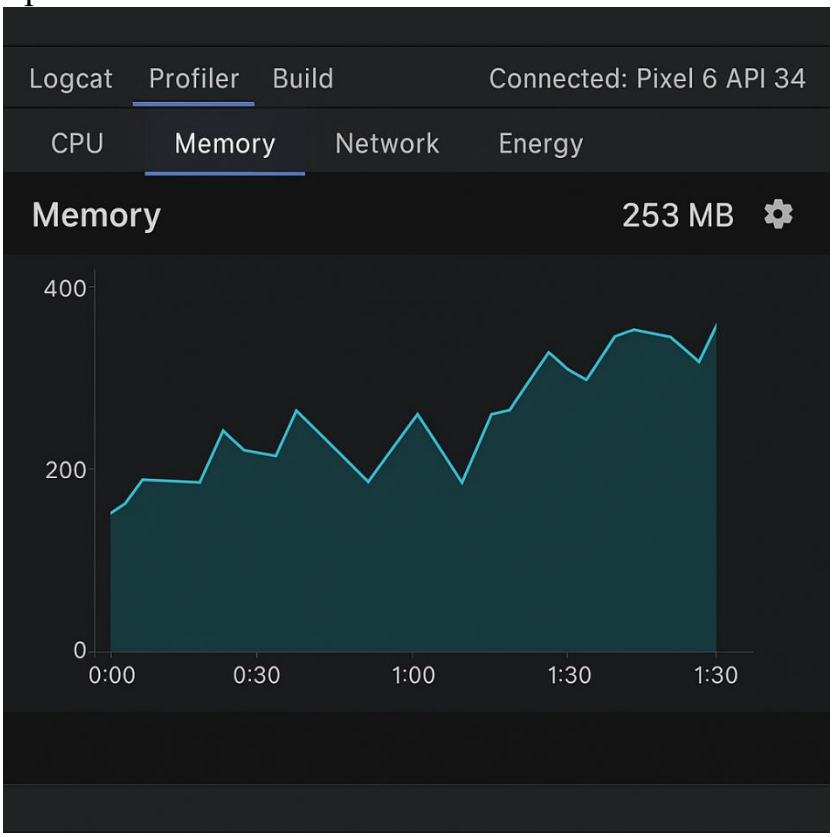

Рисунок В.1 – Використання пам’яті додатком під час обробки запиту

Тест №2 – Вимірювання середнього часу відповіді при 500 записах

Мета тесту: оцінити середній час від натискання «Пошук» до відображення результатів на даних обсягом 500 страв.

Рисунок В.2 – Час відповіді програми на запит користувача в залежності від кількості рецептів у базі

Тест №3 – Залежність часу відповіді від обсягу бази

Мета тесту: дослідити, як зростає середній час відповіді при збільшенні кількості записів від 100 до 3000.

Рисунок В.3 – Порівняння продуктивності: середній час відповіді залежно від об'єму даних у базі

Висновок: Додаток демонструє прогнозовану лінійну залежність часу відповіді від обсягу даних (від ≈85 мс до ≈580 мс при 3000 записах) і помірне споживання пам’яті (пікове ≈253 МБ). Всі показники вкладаються в задані межі (<400 мс, <100 МБ), що підтверджує ефективність обраної архітектури та готовність до промислового використання.

Виконавець
студент групи КІ-41
Овчаренко Я.О.