МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
« » травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Мобільний застосунок для створення користувацького кулінарного довідника»
здобувача групи ІНз-01с Савицького Артема Олеговича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Савицький Артем
(підпис)

Керівник,
старший викладач кафедри комп'ютерних наук, к.т.н., доцент
Борис КУЗІКОВ _____
(підпис)

Суми – 2024

Сумський державний університет
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика» здобувача групи ІНз-01с Савицького Артема Олеговича

1. Тема роботи: «Мобільний застосунок для створення користувацького кулінарного довідника» затверджую наказом по СумДУ від *«26» квітня 2024 р. № 0438-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до «05» червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз особливостей кулінарних додатків та операційної систами Android 2) Вибір інструментів та технологій розробки 3) Розробка програмного рішення та тестування програми.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз предметної області, вивчення особливостей кулінарних додатків, вибір операційної системи*		
2	*Вибір інструменів та технологій розробки*		
3	*Розробка програмного рішення, проектування програми, розробка кодової програми та тестування*		
4	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 73 ст., 1 додаток, 18 літературних джерел.

Обґрунтування актуальності теми роботи - Мобільні додатки стають невід'ємною частиною повсякденного життя. Розробка додатку з рецептами сприяє полегшенню процесу приготування їжі, забезпечуючи швидкий доступ до різноманітних кулінарних ідей, порад та рецептів. Зростання популярності різноманітних кухонь та дієтичних обмежень також підсилює необхідність у такому додатку, який би надавав інформацію та підтримку для різних кулінарних вподобань та потреб.

Об’єкт дослідження - додатки для способу життя (lifestyle apps).

Мета роботи - розробити мобільний додаток, який дозволить користувачам швидко знаходити, переглядати та зберігати рецепти кулінарних страв.

Методи дослідження – практичні та теоретичні аспекти розробки кулінарного мобільного додатку.

Результати - розроблено мобільний застосунок для створення користувацького кулінарного довідника

Ключові слова: android, lifestyle apps, збірник рецептів, sqlite, кулінарний довідник

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI - Користувацький інтерфейс.

SDK - Набір інструментів розробки програмного забезпечення.

API - Інтерфейс програмування застосунків.

IDE - Інтегроване середовище розробки.

DB - База даних.

HTTP - Протокол передачі гіпертексту.

URL - Загальний ресурсний локатор.

XML - Розширена мова розмітки.

JSON - Об'єктна нотація JavaScript.

MVC - Модель-представлення-контролер.

MVVM - Модель-представлення-в'юмодель.

DAO - Об'єкт доступу до даних.

ORM - Об'єктно-реляційне відображення.

REST - Представлення стану переносу.

CRUD - Створення, читання, оновлення, видалення.

URL - Загальний ресурсний локатор.

SSL - Протокол захищеної сокетної шару.

UX - Враження користувача.

DIP - Принцип інверсії залежностей.

ВСТУП

Актуальність теми. Мобільні додатки стають невід'ємною частиною повсякденного життя, і розробка додатку з рецептами сприяє полегшенню процесу приготування їжі, забезпечуючи швидкий доступ до різноманітних кулінарних ідей, порад та рецептів. Зростання популярності різноманітних кухонь та дієтичних обмежень також підсилює необхідність у такому додатку, який би надавав інформацію та підтримку для різних кулінарних вподобань та потреб.

Об’єкт дослідження - додатки для способу життя (lifestyle apps).

Предмет дослідження - практичні та теоретичні аспекти розробки кулінарного мобільного додатку.

Мета дослідження - розробити мобільний додаток, який дозволить користувачам швидко знаходити, переглядати та зберігати рецепти кулінарних страв.

Відповідно до мети було поставлено наступні **завдання:**

- Розглянути особливості розробки кулінарних додатків.
- Проаналізувати мови програмування для розробки Android-додатків.
- Проаналізувати системи керування бази даних.
- Спроектувати програмне рішення.
- Реалізувати програмне рішення.
- Протестувати розроблену програму.

Структура роботи. Робота складається з переліку позначень, трьох розділів, восьми підрозділів, висновків та списку використаних джерел. Загальний обсяг роботи - 73 сторінки.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості розробки кулінарних додатків

Додатки з рецептами - мобільні додатки, завдяки яким готувати їжу легше, швидше та зручніше. По суті, такий додаток – вірний помічник на кухні, такий собі цифровий повар. Це міні-кухар, готовий допомогти вам порадою та дати можливість порадитися з іншими користувачами; і ви можете поділитися з ними своїм кулінарним досвідом.

Звичайно, додатки, про які йде мова, так чи інакше відрізняються за своїм функціоналом, але їх об’єднує одне: вони пропонують вам корисний кулінарний контент і допомагають покращити ваші кулінарні навички.

Спочатку, потенційний користувач завантажує та встановлює додаток, потім він відкриває програму і вибирає потрібний рецепт зі списку топових або нових. Крім того, він має можливість скористатися пошуковою системою і уточнити свої вимоги.

Користувач переглядає вподобаний рецепт і дізнається, які інгредієнти потрібні і наскільки складний процес приготування. Якщо його все влаштовує, він починає готувати, після чого може викласти фотографію готової страви та поділитися нею з друзями в соціальних мережах.

Попит на подібні програми сьогодні зростає, що підтверджується статистикою. Згідно з останніми дослідженнями, ринок додатків для приготування їжі та рецептів досяг понад 2,15 мільйона доларів і вже залучив щонайменше 21 мільйон користувачів.

Причини популярності кулінарних додатків:

Задоволення від кулінарних експериментів. Багато представників сучасного покоління (яке іноді називають поколінням Z) люблять приділяти частину свого дозвілля кулінарії, і віддають перевагу новим, незнайомим стравам. А їхніми вірними помічниками в цих кулінарних експериментах можуть стати різні додатки з рецептами.

Зросла популярність моделі під замовлення. Сьогодні ми всі постійно кудись поспішаємо. І тому ми хочемо мати можливість замовляти будь-яку послугу онлайн і отримувати виконану роботу тут і зараз (на вимогу). Таксі Uber є яскравим прикладом такої моделі: нам потрібна машина, ми робимо пару натискань і знаходимо найближче таксі. Те ж саме стосується кулінарії: ми хочемо щось приготувати, не витрачаючи зайвого часу на пошук потрібного рецепту. Тому ми повинні мати відповідну програму для досягнення поставленої мети… це ще одна причина створити програму рецептів.

Глобальна цифровізація. Більшість компаній намагаються надати свої послуги онлайн, щоб задовольнити вимоги своїх клієнтів. І кулінарія не виняток.

Створення додатку з рецептами може включати в себе наступні кроки:

- Вибір стека технологій для розробки додатків для приготування їжі
- Вибір мобільної ОС. Іншими словами, перш за все, ви повинні вирішити, на яких платформах буде працювати ваш додаток, iOS чи Android.

Найкраще проаналізувати своїх майбутніх користувачів і визначити, які пристрої вони зазвичай вибирають. Якщо більшість із них віддають перевагу продуктам Apple, почніть зі створення програми рецептів для платформи iOS.

Нативна та кросплатформна розробка. Якщо ви хочете досягти успіху, вам потрібно, щоб ваша мобільна програма працювала на всіх платформах, як Apple, так і Android, тому наша попередня порада корисна лише на ранніх етапах життя програми. Згодом, коли додаток почне приносити прибуток, варто розширити його можливості. Однак визначитися з підходом до розвитку потрібно заздалегідь:

Нативна підтримка. Іншими словами, ви створюєте додаток, використовуючи рідну мову програмування певної ОС (окремо для платформ Android та iOS). Цей підхід потребує більше часу, а отже, і грошей.

Кроссплатформний підхід, що дозволяє спростити процес розробки та створити додаток, що підтримується різними операційними системами. Вартість розробки додатка з рецептами знизиться, оскільки ви створюєте свій сервіс лише один раз.

Вибір архітектури програми.

- монолітна архітектура, що означає, що ми створюємо єдину взаємопов'язану систему. Така система досить складна і іноді менш зручна у використанні.
- мікросервісний підхід. У 2-му випадку система складається з численних служб, які спілкуються один з одним за допомогою деяких протоколів. Метод більш гнучкий і легший в управлінні, оскільки ці служби менш залежать одна від одної.

Вибір мови програмування для написання програми. Варіантів дуже багато, і всі перерахувати практично неможливо.

Звичайно, ми описали лише невелику частину проблем, з якими доведеться зіткнутися розробнику в процесі розробки програми кулінарних рецептів; ви просто бачите приклад основних можливих проблем.

Вибір технологічного стеку може впливати на:

- Ефективність застосування. Почнемо з того, що обраний вами метод розробки впливає на швидкість і продуктивність вашої програми.
- Масштабованість програми, що передбачає можливість її подальшого розвитку [2].
- Вартість програми. Нарешті, вартість розробки також пов'язана з вибраним стеком технологій.

Корисні API для легкого створення програми рецептів

Розробка рецептурного додатку – процес, м'яко кажучи, складний, тривалий і, відповідно, дорогий. Однак існують різні способи спростити це, і один із них – використання API.

API (інтерфейс прикладного програмування) — це набір певних процедур, функцій і констант для взаємодії між сайтом або додатком і сторонніми серверами. Розробник може скористатися перевагами API, щоб отримати доступ до функціональності сторонньої програми (скажімо, для впровадження платіжної системи).

Хорошим прикладом є вбудовування відео YouTube на ваш сайт, що можливо завдяки API YouTube. Спрощений процес реєстрації в соціальних мережах (в цьому на допомогу прийде Facebook API); Доступ до джерела різноманітних рецептів; Розширений пошук рецептів (точніше, йдеться про інтеграцію спеціального сервісу, щоб зробити весь процес пошуку зручним та інтуїтивно зрозумілим); Обробка та управління інгредієнтами конкретного рецепту; Пошук найближчих продуктових магазинів;

І це лише короткий перелік того, на що здатні API. Звісно, деякі з них є платними, що може збільшити ваші витрати на створення власного додатка для рецептів, але іноді це варто. Особливо коли ви також можете знайти безкоштовні рішення.

Функції користувача

Реєстрація. Давайте почнемо з найпростішої речі - реєстрації. Найважливіше - не змушуйте користувача робити непотрібні кроки, інакше він відмовиться від ідеї встановлення вашого додатка і вибере конкурентний сервіс. Найпростіший спосіб зареєструватися - через соціальні мережі.

Створення облікового запису. Користувач успішно зареєструвався. Тепер йому потрібно розповісти іншим користувачам про себе. Запросіть його заповнити коротку анкету з основними питаннями, такими як вік, стать, місцезнаходження, заняття, хобі і т. д.

До речі, якщо користувач зареєструвався через соціальні мережі, ключова інформація про нього буде отримана з його акаунта в FB, Instagram, Twitter (або інших соціальних мережах).

Кулінарний контент. Основний екран додатка повинен пропонувати користувачеві список різних видів рецептів, які можна відсортувати за новизною, рейтингом та іншими параметрами; такий варіант був би зручний, якщо користувач не збирається вдаватися до іншої функції, а саме…

Пошук з фільтрами. Також всі типи додатків для рецептів повинні мати функцію пошуку, щоб допомогти користувачам знайти потрібні рецепти. Крім того, користувачі можуть скористатися різними фільтрами, щоб уточнити, який рецепт їх цікавить (наприклад, здорова дієта, італійська кухня і т. д.).

Список доступних рецептів. Користувач вказав свої побажання і клацнув кнопку Пошук. Тепер ваш додаток повинен вивести всі відповідні результати.

Опис рецепту. Допустимо, користувач переглянув список доступних рецептів і вибрав ті, які йому здалися найбільш відповідними. Що він повинен робити на наступному етапі? Звісно, користувач повинен перейти за посиланням і прочитати детальний опис рецепту, який йому подобається найбільше. І ви, як власник додатка (тобто той, хто вирішив створити додаток для рецептів), повинні переконатися, що він може це зробити. Ідеальний рецепт повинен містити список інгредієнтів і опис методу приготування; бажано також ілюструвати рецепт зображеннями.

Відеоінструкція. Це добре, коли рецепт описаний і проілюстрований детально. Але краще, якщо користувач може побачити процес приготування своїми власними очима. Саме тоді всі види відеоінструкцій будуть дуже корисними.

Харчова цінність. Сьогодні модно вести здоровий спосіб життя і дотримуватися дієти. І люди, які дотримуються такого тренду, використовують додаток для рецептів для здорового приготування (поміж інших речей). Це означає, що ваш сервіс повинен надавати інформацію про харчову цінність кожного рецепту.

Обране. Допустимо, користувачу подобається рецепт, але він не готовий приготувати страву зараз. Тоді дайте йому можливість зберегти рецепт і додати його до розділу Обране.

Соціальна інтеграція. Користувачі люблять ділитися контентом, який їм подобається, на своїх сторінках у соціальних мережах. Це також стосується готування. Тому кнопка Поширення дозволить вашим користувачам публікувати цікаві рецепти на Facebook або інших соціальних мережах та таким чином просувати ваш мобільний сервіс [5]..

Додавання рецептів. Під час створення додатка для рецептів ви можете використовувати модель агрегатора, що може значно спростити все. У цьому випадку додаток збирає (агрегує) і класифікує рецепти з різних ресурсів. Проте ми рекомендуємо дозволити користувачам додавати рецепти самостійно і супроводжувати їх зображеннями та відеоінструкціями (якщо це можливо).

Планування меню. Крім того, користувач оцінить можливість планувати своє меню, складене з рецептів вашого додатка.

Відгуки та оцінки. Витрати на створення мобільного додатка для рецептів не підвищаться занадто сильно, якщо ви додасте відгуки та оцінки, але користувачі точно оцінять таку функцію. Вони зможуть оцінювати рецепти, які спробували, і читати відгуки інших користувачів про певну страву перед тим, як почати готувати її самостійно.

Керування налаштуваннями. Дайте користувачу можливість керувати своїм обліковим записом і змінювати налаштування за потреби.

Повідомлення push. Звісно, ви не можете обійтися без сповіщень у процесі розробки додатка для рецептів. Ця функція є важливою для більшості цифрових продуктів [6]..

Особливості адміністратора: Другий блок функцій спрямований на адміністратора вашого додатка. Завдання адміністратора - управляти системою: платежами, користувачами, контентом та іншими аспектами.

5 Додаткових функцій для кулінарного додатка:

1. Перерахунок інгредієнтів. Звичайно, будь-який рецепт вказує правильну кількість кожного інгредієнту на порцію (або на певну кількість порцій). Але що, якщо потрібно приготувати більше або менше порцій (порівняно з рецептом)? Для спрощення перерахунку інгредієнтів додайте відповідну функцію.
2. Список покупок. Припустимо, користувачу подобається певний рецепт, але деяких інгредієнтів немає в його холодильнику. Що він має зробити? Правильно, додайте їх до свого списку покупок за допомогою вашого додатка. Це одна з багатьох переваг додатків для кулінарії.
3. Калькулятор вартості рецепту. Попередня опція логічно призводить до цього, оскільки складання списку покупок - це лише половина бою. Важливо також визначити, скільки грошей потрібно витратити на покупку всіх інгредієнтів. Калькулятор вартості рецепту допоможе користувачу скласти свій бюджет на приготування страви.
4. Пошук найближчих продуктових магазинів. У користувача вже є список покупок, тепер йому лише потрібно дізнатися, в який магазин йому йти з ним. Ви можете дати йому підказку, знаходячи і показуючи найближчі продуктові магазини.
5. Голосове управління. Наш невеликий список додаткових функцій додатка для рецептів завершується такою чудовою опцією, як голосове управління. Користувач готує страву, тому він не може тримати смартфон у руках, оскільки вони зараз зайняті; і немає потреби робити це! Все, що користувач повинен зробити - це вимовити команду вслух (звісно, це стосується простих команд, таких як "Повернутися Назад").

1.2 Оперційна система Android

Android – це технологія. Це операційна система з відкритим кодом, що означає, що кожен, хто хоче використовувати Android, [8]. Може зробити це, завантаживши повний вихідний код Android. Це було використано для розробки мобільних додатків і настільних додатків. Це повний набір програмного забезпечення для мобільних пристроїв, наприклад, планшетних комп'ютерів, смартфонів, ноутбуків, годинників і пристроїв для читання книг, телеприставок тощо. Він містить три речі: операційну систему, проміжне програмне забезпечення, ключові програми. Android — це операційна система на базі Linux.

ФУНКЦІЇ ANDROID

- Зберігання: легка реляційна база даних SQLite, використовується для зберігання даних.
- Підключення: він підтримує багато технологій підключення, як-от WIFI, BLUETOOTH, UMTS, WIXMAX, CDMA та GSM/EDGH.
- Обмін повідомленнями: Android підтримує функцію як MMS, так і SMS.
- Веб-браузер: у поєднанні з двигуном Chrome V8 JavaScript, який підтримує CSS3 і HTML5, на основі механізму макета Web Kit з відкритим кодом.
- Multi-touch: Android має функцію Multi-touch, яка вперше була створена в телефоні HTC Hero.
- Багатозадачність: різні програми можуть працювати одночасно. Користувач може переходити від одного завдання до іншого одночасно.
- Модем: андроїд підтримує спільний доступ до Інтернету як дротову/бездротову точку доступу.

АРХІТЕКТУРА ANDROID

Стек програмного забезпечення складається з п'яти рівнів:

- Прикладний рівень
- Рамка програми працює

- Бібліотеки
- Час виконання
- Ядро

Андроїд базується на архітектурі ядра Linux 2.6. Цей рівень є ядром архітектури Android. Він надає такі послуги, як керування процесором і живленням, безпека, керування пам’яттю тощо [11]..

Час виконання Android

Android Runtime – це програма. Це середовище виконання. Використовується операційною системою Android. Де у вас обмежений акумулятор, обмежений процесор, обмежена пам’ять. Android має власну віртуальну машину під назвою Dalvik. Запускається в програмі Android. Dalvik використовується в мобільних пристроях, таких як планшети, телефони тощо. Програми зазвичай пишуться на java та компілюються в байт-код.

Android Run Time

Це синім кольором означає, що воно написано мовою програмування Java. Внутрішня бібліотека містить усі утиліти колекції, клас, IO та всі утиліти.

Framework програми

Фреймворк програми – це набір інструментів, який використовують усі програми, і все це написано мовою програмування Java. До цих програм належать програми, які постачаються разом із телефоном, наприклад програма для телефону, або домашні програми, .Включає програми, написані Google, і включає програми, які будуть написані вами. Кожна програма використовує однакові API та структуру.

ПЕРЕВАГИ І НЕДОЛІКИ ANDROID

Багато переваг або недоліків програми для Android.

Переваги Android

- Телефон Android легко отримати.

- Його легко носити з собою.
- Це безпечніше.
- Завдяки підтримці багатьох програм користувач може змінювати відображення екрана.
- Це не залежить від платформи.
- Головною перевагою Android є багатозадачність.
- Це підтримка сервісу Google.
- Це дало вам краще сповіщення.
- Більш зріла платформа.

Недолік Android

- Деякий час зловмисники можуть атакувати android.
- Операційна система Android має багато процесів через це зависання ОС Android.
- Якщо пам'ять заповнена, телефон Android працює повільно.
- Надзвичайна невідповідність дизайну додатків.
- Він нестабільний і деякий час виходить з ладу.

ВИСНОВОК ТА МАЙБУТНІ СФЕРИ ЗАСТОСУВАННЯ

Android є платформою з відкритим вихідним кодом, на ній можна легко розробити мобільну програму. Усі API доступні для розробки цих програм. Він дуже безпечний і допомагає захистити від зловмисників. У сфері штучного інтелекту можна створити багато типів корисних роботів, які знадобляться в майбутньому. Розробляється для використання в автомобілях, годинниках, а також Android TV [16].

Таким чином, було проаналізовано мобільні додатки для рецептів, а також охарактеризовано операційну систему Android.

2. ВИБІР ІНСТРУМЕНТІВ І ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Аналіз мов програмування

На сьогоднішній день Java залишається однією з найпопулярніших мов програмування у світі, широко використовується в корпоративному середовищі, веб-розробці, мобільній розробці та багатьох інших сферах програмування. Її поширення та відкритість дозволяють розробникам створювати різноманітні програмні рішення для різних платформ та задач.

Java також відома своєю простотою в освоєнні для новачків. Синтаксис Java досить легкий для розуміння, що робить її привабливою для тих, хто тільки починає свій шлях в програмуванні. Крім того, Java - це об'єктно-орієнтована мова програмування, що означає, що код структурований навколо об'єктів, що спрощує розробку та підтримку складних програм.

Переваги:

Широке використання: велика кількість розробників використовує Java, тому є велика спільнота, багато документації та підтримка.

Об'єктно-орієнтована: Java є повністю об'єктно-орієнтованою мовою, що дозволяє структурувати код у вигляді об'єктів.

Крос-платформенність: Код, написаний на Java, може бути використаний на різних платформах з деякими або з жодними змінами. Це дозволяє розробникам використовувати свій досвід і знання для різних проектів.

Недоліки:

У порівнянні з Kotlin, Java може бути менш продуктивною мовою через своє старіння та менш сучасні можливості.

В Java потрібно більше коду, ніж у Kotlin, для досягнення тих самих результатів, що може призвести до більшої кількості робочих годин та можливих помилок.

До версії 8 Java не мала нативної підтримки нульових посилань, що може викликати проблеми з безпекою та викликати помилки в процесі виконання.

Додатки, написані на Java, можуть вимагати більшої кількості пам'яті та обчислювальних ресурсів, порівняно з додатками, написаними на більш нових мовах, таких як Kotlin.

Kotlin - це мова програмування, яка призначалася як альтернатива Java. Одним з основних цілей розробників Kotlin було зробити мову, яка була би 100% сумісною з Java, щоб розробники могли поступово переходити на нову мову без проблем з більш сучасним та ефективним підходом.

Переваги Kotlin:

Сучасний синтаксис: Kotlin має чистий і сучасний синтаксис, який полегшує читання та написання коду. Це робить процес розробки більш приємним і продуктивним.

Безпека нульових посилань: Kotlin вбудовує безпеку нульових посилань, що допомагає уникнути багатьох типових помилок, пов'язаних з нульовими значеннями, які можуть виникати в Java.

Інтероперабельність з Java: Kotlin повністю сумісний з Java, що дозволяє використовувати код Kotlin поряд з існуючим Java-кодом без проблем.

Розширена функціональність: Kotlin має багато нововведень, які спрощують розробку, такі як лямбда-функції, властивості, розширення функціональності та інші.

Підтримка Android: Kotlin офіційно підтримується Google для розробки Android-додатків, що робить його привабливим вибором для розробників мобільних додатків.

Недоліки Kotlin:

Великий розмір файлів APK: Додатки, написані на Kotlin, можуть мати більший розмір APK файлів порівняно з додатками, написаними на Java, через додаткові бібліотеки Kotlin, які включаються в компіляцію.

Відсутність повної сумісності з JavaFX та Android SDK (в деяких випадках): Хоча Kotlin повністю сумісний з Java, у деяких випадках можуть

виникати проблеми з сумісністю з JavaFX та Android SDK через різниці в підходах та деяких особливостях мови.

XML (eXtensible Markup Language) - це розширювана мова розмітки, яка використовується для представлення та обміну даними у текстовому форматі.

У розробці Android-додатків XML використовується для опису інтерфейсу користувача (UI) за допомогою файлів розмітки ресурсів. Файли розмітки XML містять опис вигляду екранів додатка, включаючи розташування та параметри різних елементів, таких як кнопки, текстові поля, зображення тощо.

Один з найбільш поширених форматів файлів розмітки у Android-розробці - це файл макету (layout file), який зазвичай має розширення .xml та знаходиться у папці res/layout вашого проекту. У цьому файлі ви можете описати розміщення та вигляд всіх елементів UI за допомогою різних тегів XML, таких як <LinearLayout>, <RelativeLayout>, <TextView>, <Button> та інші.

Загалом, XML грає важливу роль у розробці Android-додатків, дозволяючи розробникам легко та ефективно створювати та налаштовувати інтерфейси користувача своїх додатків.

2.2 Аналіз системи управління бази даних

SQLite є вбудованою, легкою та надійною реляційною базою даних, яка широко використовується в розробці Android-додатків. Її основні переваги полягають в простоті використання, ефективності та низькому рівні споживання ресурсів. Одним з ключових аспектів SQLite є те, що вона зберігає всю базу даних у вигляді одного файла на пристрої, що спрощує розгортання та управління даними.

Однією з головних особливостей SQLite є його підтримка SQL, що робить його знайомим та зрозумілим для багатьох розробників, які вже мають досвід з реляційними базами даних. Вона підтримує стандартні операції SQL, такі як

SELECT, INSERT, UPDATE та DELETE, а також ряд розширених можливостей, таких як транзакції, індексація та обмеження цілісності даних.

Крім того, SQLite є добре оптимізованою базою даних, яка забезпечує високу швидкість операцій навіть на пристроях з обмеженими ресурсами. Вона має низький рівень споживання пам'яті та процесорних ресурсів, що робить її ідеальним вибором для мобільних пристроїв.

Незважаючи на свої переваги, SQLite має деякі обмеження. Наприклад, вона не має вбудованої підтримки шифрування даних, тому для збереження конфіденційної інформації може знадобитися додатковий код або використання сторонніх бібліотек. Крім того, вона має обмежену підтримку для операцій, таких як JOIN, які можуть бути складнішими для виконання порівняно з більш потужними базами даних, такими як MySQL або PostgreSQL.

Однак, враховуючи його простоту використання, ефективність та низький рівень споживання ресурсів, SQLite залишається популярним вибором для розробки Android-додатків, особливо для додатків з невеликим обсягом даних або для простих використань бази даних.

Firebase - це платформа для розробки мобільних та веб-додатків, яка надає різноманітні сервіси, такі як аналітика, аутентифікація, зберігання файлів та, зокрема, бази даних. Одним із ключових компонентів Firebase є Firebase Realtime Database та Firestore, які надають можливості зберігання та синхронізації даних у реальному часі.

Firebase Realtime Database - це база даних реального часу, яка дозволяє зберігати та синхронізувати дані між пристроями користувачів та сервером у реальному часі. Вона працює на основі JSON і забезпечує миттєву синхронізацію даних, тобто зміни, які внесені на одному пристрої, автоматично відображаються на всіх інших пристроях, підключених до бази даних.

Firestore - це NoSQL база даних, яка пропонує розширені можливості синхронізації даних, масштабованості та гнучкості. Firestore працює на основі

колекцій та документів, що дозволяє зберігати дані у вигляді деревовидної структури. Вона підтримує багатофільтровані запити, а також підтримує офлайн-режим, що дозволяє працювати з даними навіть без підключення до Інтернету.

Обидва варіанти баз даних Firebase мають деякі спільні переваги. Вони легко інтегруються з іншими сервісами Firebase, такими як аутентифікація користувачів, зберігання файлів, аналітика та інші. Крім того, Firebase має простий та зрозумілий для використання SDK, який дозволяє швидко почати працювати з базою даних.

Одним з недоліків Firebase є те, що він може виявитися вартим для проектів, які вимагають багато операцій читання/запису даних, оскільки це може призвести до високих витрат на серверні операції у разі частого доступу до бази даних.

У кінцевому підсумку, Firebase - це потужний інструмент для зберігання та синхронізації даних у реальному часі, який забезпечує зручний та ефективний спосіб управління даними у вашому Android-додатку.

MySQL - це одна з найпопулярніших реляційних баз даних у світі, що відкрита для використання, безкоштовна для більшості випадків та широко використовується для розробки різноманітних веб-додатків. Ця база даних є частиною стеку LAMP (Linux, Apache, MySQL, PHP/Python/Perl), що зробило її особливо популярною у веб-розробці. MySQL відома своєю швидкістю, надійністю та простотою використання.

Перша версія MySQL була випущена в 1995 році компанією MySQL AB, яка пізніше була придбана корпорацією Sun Microsystems, а потім Oracle. Однак, вона залишилася відкритою для використання і розвитку за допомогою спільноти та різних фірм-партнерів.

Однією з ключових переваг MySQL є його швидкодія, яка робить його ідеальним вибором для веб-додатків з високим навантаженням. Вона має вбудовані оптимізації запитів, індексацію даних та кешування, що дозволяє розробникам створювати продуктивні та швидкодіючі додатки.

MySQL підтримує стандартні SQL-запити, що робить його зрозумілим для більшості розробників, які мають досвід з реляційними базами даних. Вона також має розширені можливості, такі як транзакції, тригери, збережені процедури, що дозволяє створювати складні та потужні рішення для управління даними.

Однією з переваг MySQL є його велика спільнота користувачів та розробників, що забезпечує доступ до багатьох ресурсів, документації та підтримки. Це робить MySQL привабливим вибором для багатьох компаній та розробників, які шукають надійне та ефективне рішення для зберігання даних.

MongoDB - це документ-орієнтована, NoSQL база даних, яка широко використовується для зберігання та управління даними у веб-додатках, мобільних додатках та інших проектах з великим обсягом даних. MongoDB відома своєю гнучкістю, масштабованістю та простотою використання.

Однією з ключових особливостей MongoDB є модель даних, яка базується на документах у форматі BSON (Binary JSON). У MongoDB дані зберігаються у вигляді колекцій документів, де кожен документ може мати власну структуру. Це робить MongoDB дуже гнучкою базою даних, оскільки дозволяє зберігати дані без явної схеми та легко модифікувати їх структуру.

Ще однією важливою особливістю MongoDB є його підтримка розподіленого зберігання даних. MongoDB може легко масштабуватися горизонтально, дозволяючи розподілити дані між різними серверами або кластерами серверів. Це дозволяє обробляти великі обсяги даних та навантаження, забезпечуючи високу доступність та надійність системи.

MongoDB також відома своєю простотою використання та наявністю розширених можливостей, таких як індексація, агрегація, текстовий пошук та багато іншого. Вона має потужний мовний драйвер, який дозволяє взаємодіяти з базою даних з багатьох мов програмування, включаючи JavaScript, Python, Java, PHP та інші.

Однією з недоліків MongoDB є те, що вона може бути менш ефективною для додатків з великим обсягом транзакцій або складних операцій, які потребують використання транзакційної моделі. Також важливо враховувати, що MongoDB, у порівнянні з реляційними базами даних, може вимагати більше додаткового зусилля для забезпечення цілісності та унікальності даних.

У кінцевому підсумку, MongoDB - це потужна та гнучка база даних, яка надає широкий спектр можливостей для зберігання та управління даними. Вона ідеально підходить для великих проектів з великим обсягом даних, які вимагають гнучкої та масштабованої моделі даних.

Таким чином, для розробки програмного рішення буде використано мову програмування Kotlin, оскільки вона є найбільш актуальною на сьогоднішний день для розробки Android-додатків, а також СУБД SQLite, що дозволить легко і зручно зберігати дані користувача локально на його пристрої.

3. РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

3.1 Проектування програми

MVVM, або Model-View-ViewModel, є архітектурним шаблоном проектування програмного забезпечення, який використовується для розробки користувацьких інтерфейсів. У цій архітектурі компоненти програми розділяються на три основні частини: модель (Model), представлення (View) та модель представлення (ViewModel). Кожна з цих частин виконує певні функції, що сприяють відокремленню логіки програми та полегшують тестування та розвиток програмного забезпечення.

У MVVM модель (Model) відповідає за управління даними та бізнес-логікою програми. Це може бути база даних, віддалені джерела даних або будь-яка інша логіка, яка працює з даними. Модель не залежить від інших компонентів програми і може бути перевикористана у різних контекстах.

Представлення (View) відображає дані користувачу і обробляє взаємодію з користувачем. Вона відповідає за створення і відображення графічного інтерфейсу користувача, а також за передачу користувачських дій до моделі представлення. У контексті Android, це може бути активність (Activity), фрагмент (Fragment) або інший компонент інтерфейсу користувача.

Модель представлення (ViewModel) відповідає за взаємодію між моделлю і представленням. Вона отримує дані від моделі, обробляє їх і підготовлює для відображення в представленні. ViewModel також відстежує стан представлення та реагує на зміни в ньому, сповіщаючи представлення про оновлення даних.

Одна з основних переваг MVVM полягає в тому, що вона забезпечує відокремлення бізнес-логіки від логіки інтерфейсу користувача, що полегшує тестування і підтримку програми. Крім того, MVVM сприяє відокремленню розробки між розробниками, які працюють над моделлю, представленням і моделлю представлення.

При проектуванні майбутньої програми було розроблено наступну діаграму використання:

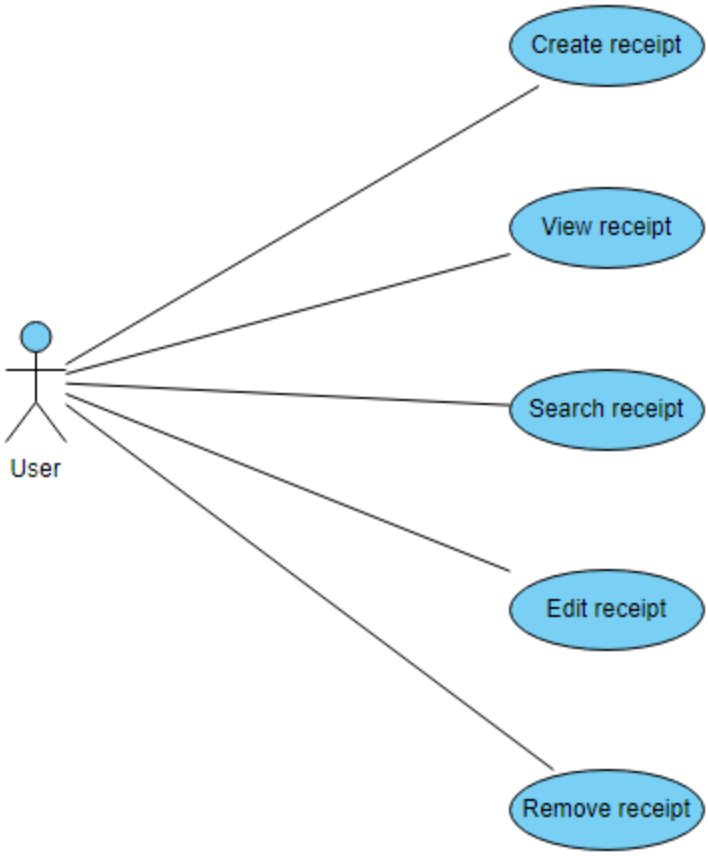

Рисунок 3.1 - Діаграма використання

Також, було розроблено базу даних, яка має наступну структуру:

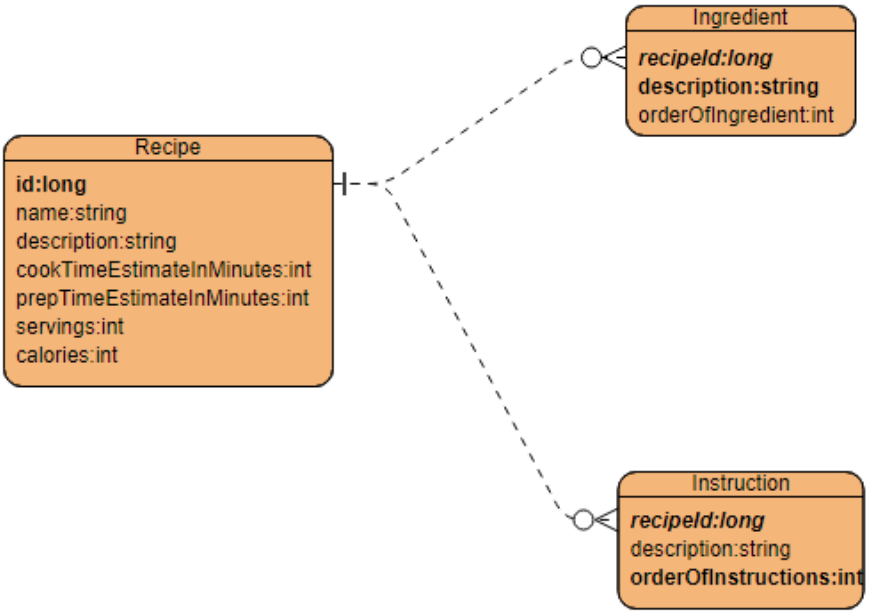

Рисунок 3.2 - Діаграма відношення сутностей БД

3.2 Розробка кодової бази

```
class RecipeFragment : Fragment() {
    private lateinit var recipeAdapter: RecipeAdapter
    private lateinit var viewPager: ViewPager2

    private var recipeViewModel: RecipeViewModel? = null

    private var _binding: FragmentRecipeBinding? = null

    private val binding get() = _binding!!
```

Лістинг 3.1- Об’явлення класу фрагементу рецепту

Код створює фрагмент для відображення рецептів у додатку для Android. Давайте розглянемо його крок за кроком:

RecipeFragment наслідує клас Fragment, що вказує на те, що цей клас буде використовуватися як фрагмент у додатку Android.

У класі RecipeFragment є приватні властивості:

recipeAdapter: це адаптер для відображення рецептів.

viewPager: це ViewPager2, який використовується для прокрутки списку рецептів.

recipeViewModel: це екземпляр RecipeViewModel, який використовується для отримання даних про рецепти.

_binding: це приватна змінна для збереження зв'язування фрагменту.

Властивість binding є відкладеною (lazy) і забезпечує доступ до зв'язування фрагменту. Вона повертає _binding, який має значення лише після того, як фрагмент був прикріплений до інтерфейсу користувача.

В цьому коді фрагмент налаштовує зв'язування, адаптер і ViewPager2, а також отримує доступ до моделі даних рецептів через RecipeViewModel. Такий фрагмент може використовуватися для відображення списку рецептів у додатку для Android.

У цьому методі onCreateView фрагмент налаштовує свій інтерфейс користувача. Розглянемо його детально:

setHasOptionsMenu(true): Метод вказує, що у фрагмента є меню опцій.

Отримання аргументів: Фрагмент отримує аргументи через arguments, які були передані під час створення фрагмента.

Зв'язування даних: Фрагмент використовує DataBindingUtil.inflate для зв'язування макету fragment_recipe з об'єктом зв'язування FragmentRecipeBinding. Після цього _binding отримує значення xBinding.

Встановлення життєвого циклу для LiveData: Властивість lifecycleOwner встановлює фрагмент як власника для спостереження за LiveData, що дозволяє автоматично відключати спостереження при знищенні фрагмента.

Створення ViewModel: Фрагмент створює RecipeViewModel з допомогою ViewModelProvider. Для цього використовується фабрика RecipeViewModelFactory, якій передається джерело даних (dataSource) та ідентифікатор рецепту (recipeId). ViewModel зберігається в recipeViewModel.

Спостереження за LiveData: Фрагмент створює спостереження за navigateBackToRecipes у recipeViewModel. Коли значення цього LiveData змінюється на true, фрагмент викликає метод popBackStack() для повернення до попереднього фрагмента та викликає метод doneDeleting() у recipeViewModel.

Повернення кореневого елемента: Метод повертає кореневий елемент зв'язаного макету, який буде відображатися у фрагменті.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
        val args = this.arguments
        val recipeId = requireNotNull(args?.getLong("recipeId"))
        recipeAdapter = RecipeAdapter(this, recipeId)
        viewPager = binding.recipePager
        viewPager.adapter = recipeAdapter
        val tabLayout = binding.recipeTabLayout
        TabLayoutMediator(tabLayout, viewPager) { tab, position ->
            tab.text = when (position) {
                0 -> getString(R.string.information)
                1 -> getString(R.string.ingredients)
                2 -> getString(R.string.instructions)
                else -> error("Recipe tab position > 2")
            }
        }.attach()
    }
```

Лістинг 3.3 - Метод onViewCreated

У методі onViewCreated, який викликається після того, як фрагмент прикріпився до свого кореневого елемента і після того, як у фрагмента було викликано метод onCreateView, налаштовуються адаптер та ViewPager для перегляду рецепту разом з вкладками TabLayout. Розглянемо кожен рядок коду:

Отримання аргументів: Фрагмент отримує аргументи через arguments, які були передані під час створення фрагмента.

Створення адаптера: Створюється об'єкт RecipeAdapter з переданим контекстом фрагменту та recipeId.

Налаштування ViewPager: Встановлюється адаптер для viewPager, який відображає рецепт.

Налаштування TabLayout: Використовуючи TabLayoutMediator, налаштовується tabLayout для відображення вкладок разом з ViewPager. Кожній вкладці присвоюється відповідний текст з ресурсу рядка з вказаним індексом позиції.

Метод дозволяє створити віджет ViewPager з вкладками для перегляду різних частин рецепту (інформація, інгредієнти, інструкції).

```
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater)
{
        inflater.inflate(R.menu.fragment_recipe, menu)
        super.onCreateOptionsMenu(menu, inflater)
    }
```

Лістинг 3.4 - Метод onCreateOptionsMenu

У методі onCreateOptionsMenu, який викликається для створення меню фрагмента, використовується MenuInflater для надування ресурсу меню R.menu.fragment_recipe. Після того як меню надуто, викликається суперклас onCreateOptionsMenu, щоб забезпечити правильну роботу інших методів, які можуть бути перевизначені в майбутньому.

Метод дозволяє додати пункти меню до віджета фрагмента, які можуть виконувати різні дії, наприклад, видалення рецепту, відкриття діалогу для редагування тощо.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
        val recipeId =
requireNotNull(this.arguments?.getLong("recipeId"))
        val args = Bundle()
        args.putLong("recipeId", recipeId)
```

```
        when (item.itemId) {
            R.id.fragment_recipe_edit -> {
                this.findNavController().navigate(

R.id.action_recipeFragment_to_editRecipeFragment,
                    this.arguments,
                )
            }

            R.id.fragment_recipe_delete -> {
                getConfirmDeleteAlertDialog()?.show()
            }
        }
        return super.onOptionsItemSelected(item)
    }
```

Лістинг 3.5 - Метод onOptionsItemSelected

У методі onOptionsItemSelected оброблюються події вибору пунктів меню. Здебільшого Метод використовується для визначення реакції на клік по певному пункту меню.

Отримується ідентифікатор рецепту з аргументів фрагмента.

Створюється Bundle з ідентифікатором рецепту.

Здійснюється перевірка item.itemId на вибраний пункт меню.

В залежності від вибору користувача виконуються відповідні дії:

Якщо вибраний пункт меню R.id.fragment_recipe_edit, то навігація до фрагмента редагування рецепту (EditRecipeFragment) з передачею аргументів.

Якщо вибраний пункт меню R.id.fragment_recipe_delete, то показується діалогове вікно підтвердження видалення рецепту.

На завершення метод повертає super.onOptionsItemSelected(item), щоб забезпечити коректну обробку залишкових пунктів меню.

```
private fun getConfirmDeleteAlertDialog(): AlertDialog? {
        return activity?.let {
            val builder = AlertDialog.Builder(it)
            builder.apply {
                setPositiveButton(R.string.delete) { _, _ ->
                    recipeViewModel?.delete()
                }
                setNegativeButton(R.string.cancel, null)
```

```
                setTitle(R.string.delete_recipe)
                setMessage(R.string.delete_recipe_message)

setIcon(R.drawable.ic_delete_on_surface_high_emphasis)
            }
            builder.create()
        }
    }
```

Лістинг 3.6 - Метод getConfirmDeleteAlertDialog

Приватний метод getConfirmDeleteAlertDialog() створює діалогове вікно підтвердження видалення рецепту (AlertDialog). Ось кроки, які він виконує:

Використовуючи функцію let для безпечного доступу до поточної активності (якщо активність доступна).

Створюється об'єкт AlertDialog.Builder, який використовує поточну активність для побудови діалогового вікна.

Додаються налаштування для діалогового вікна:

1. Позитивний кнопка для видалення рецепту з викликом методу delete() у recipeViewModel.
2. Негативний кнопка для скасування видалення.
3. Назва і повідомлення про видалення рецепту.
4. Іконка, що підтверджує видалення.
5. Створюється діалогове вікно за допомогою методу create() об'єкта AlertDialog.Builder.
6. Повертається створене діалогове вікно.
7. Метод повертає null, якщо активність недоступна або відсутня.

```
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Лістинг 3.7 - Метод onDestroyView

Метод onDestroyView() відновлює інтерфейсний зв'язок (binding) на null при завершенні відображення фрагмента. Ось що він робить:

Викликає метод onDestroyView() батьківського класу за допомогою super.onDestroyView(), щоб забезпечити коректне функціонування внутрішніх механізмів фрагмента.

Призначає змінній _binding значення null, тим самим вивільняючи ресурси, пов'язані з інтерфейсним зв'язком фрагмента.

Це важливий етап життєвого циклу фрагмента, оскільки це дозволяє вивільнити ресурси, які використовуються для прив'язки до макету фрагмента, коли він більше не відображається на екрані.

```
class RecipeViewModel (val database: RecipeDao, val recipeId:
Long) : ViewModel() {
    /** Coroutine setup from Udacity's "Developing Android Apps with
Kotlin" */

    // viewModelJob allows us to cancel all coroutines started by
this ViewModel.
    private val viewModelJob = Job()

    // A CoroutineScope keeps track of all coroutines started by
this ViewModel.
    private val uiScope = CoroutineScope(Dispatchers.Main +
viewModelJob)
```

Лістинг 3.8 - Об’явлення класу RecipeViewModel

Цей клас RecipeViewModel використовується для управління даними та бізнес-логікою, пов'язаною з рецептами. Ось опис його функціональності:

database - це інтерфейс доступу до бази даних RecipeDao, який надає методи для взаємодії з даними рецептів.

recipeId - це ідентифікатор конкретного рецепту, з яким працює RecipeViewModel.

viewModelJob - це об'єкт типу Job, який використовується для скасування всіх корутин, які створюються в цьому ViewModel.

uiScope - це область корутин, яка відповідає за запуск корутин у головному потоці. Вона використовується для забезпечення безпеки в потоці користувацького інтерфейсу.

Отже, RecipeViewModel відповідає за взаємодію з базою даних, обробку запитів та оновлення даних рецептів у відповідь на дії користувача або зміни в додатку.

```
// Cancels all coroutines when the ViewModel is cleared.
    override fun onCleared() {
        super.onCleared()
        viewModelJob.cancel()
    }
```

Лістинг 3.9 - Метод onCleared

Метод onCleared() викликається, коли ViewModel вже не потрібна і буде очищена. У цьому методі всі корутини, які були запущені в області uiScope, скасовуються за допомогою viewModelJob.cancel(). Це важливо для запобігання витоку ресурсів і забезпечення коректного життєвого циклу ViewModel.

```
    // The ID to the recipe that we are navigating to
    private val _navigateBackToRecipes = MutableLiveData<Boolean>()
    val navigateBackToRecipes
        get() = _navigateBackToRecipes
```

Лістинг 3.10 – Поле _navigateBackToRecipes

Код створює приватне поле _navigateBackToRecipes, яке є об'єктом типу MutableLiveData<Boolean>. Це означає, що це змінна, яку можна змінювати та на яку можна підписуватися для слідкування за змінами її значення. Публічний властивий зчитувач navigateBackToRecipes надає доступ до цього поля, але не дозволяє його змінювати ззовні. Він повертає те саме значення, що і _navigateBackToRecipes, але через обгортку, що дозволяє тільки читати дані.

```
    // Signify that navigation is complete
     fun doneDeleting() {
         _navigateBackToRecipes.value = null
     }
```

Лістинг 3.11 – Метод doneDeleting

Метод doneDeleting() встановлює значення _navigateBackToRecipes на null, тим самим сигналізуючи про завершення навігації. Коли значення _navigateBackToRecipes змінюється, підписані на нього об'єкти будуть повідомлені про цю зміну. У цьому випадку, якщо значення встановлене на null, це може служити як підтвердження того, що операція завершилася.

```
    fun delete() {
        uiScope.launch {
            withContext(Dispatchers.IO) {
                database.deleteRecipe(recipeId)
            }
            _navigateBackToRecipes.value = true
        }
    }
```

Лістинг 3.12 – Метод delete

Метод delete() видаляє рецепт з бази даних у відокремленому потоці, використовуючи Dispatchers.IO, щоб виконати операцію вводу/виводу без блокування основного потоку. Після успішного видалення рецепту він встановлює значення _navigateBackToRecipes на true, щоб сигналізувати про завершення навігації назад до списку рецептів.

```
class RecipeViewModelFactory(private val dataSource: RecipeDao,
                                        private val recipeId: Long)
    : ViewModelProvider.Factory {
    @Suppress("unchecked_cast")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if
(modelClass.isAssignableFrom(RecipeViewModel::class.java)) {
            return RecipeViewModel(dataSource, recipeId) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
```

```
}
```

Лістинг 3.13 – фабрика RecipeViewModelFactory

RecipeViewModelFactory - це фабрика, яка використовується для створення екземплярів RecipeViewModel. У методі create вона перевіряє, чи переданий клас є RecipeViewModel, і якщо так, то створює новий екземпляр RecipeViewModel, передаючи йому dataSource та recipeId, які використовуються для доступу до бази даних та ідентифікатора конкретного рецепту відповідно.

3.3 Тестування програми

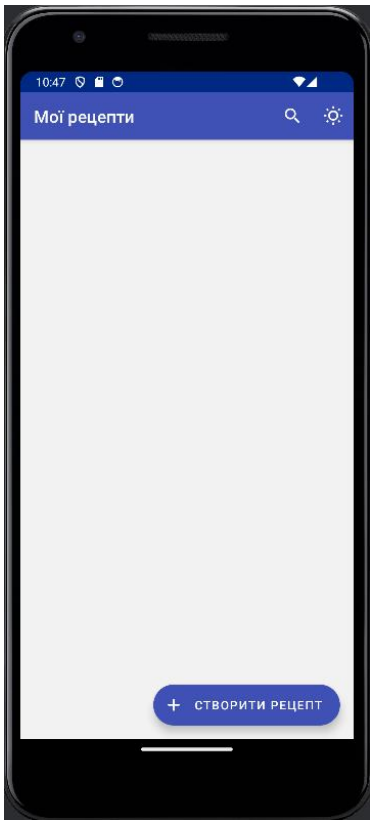

Рисунок 3.3 - Головний екран додатку

Запускаючи додаток, потрапляємо на головний екран. Він містить в собі верхню панель, список наявних рецептів (наразі порожній), а також кнопку «Створити рецепт».

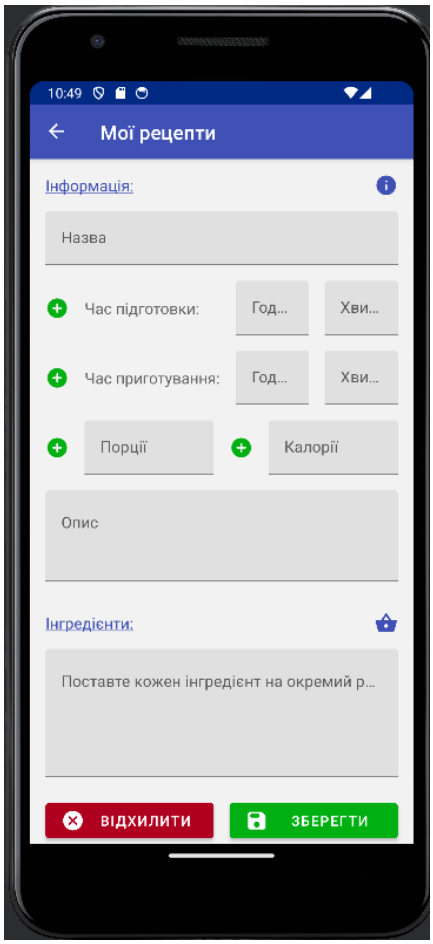

Рисунок 3.4 - Екран створення рецепту

На екрані створення рецепту наведені поля, які потрібно заповнити, для того, щоб створити рецепт, зокрема:

- Назва рецепту
- Час підготовки/час приготування
- Порції
- Калорії
- Опис

- Інгредієнти
- Інструкції
- Кнопки «Зберегти» та «Відхилити»

Додамо кілька рецептів:

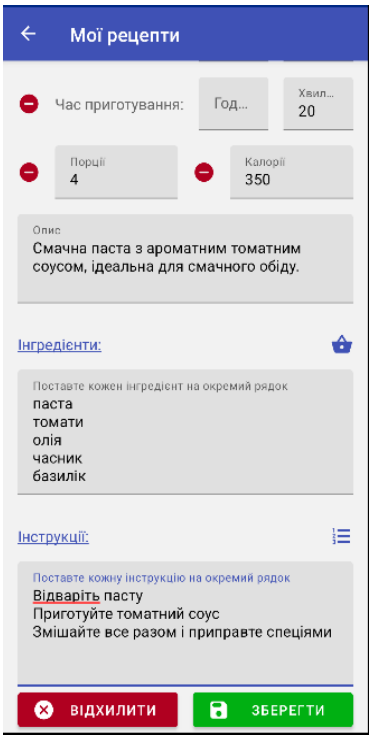

Рисунок 3.5 – Додавання рецепту

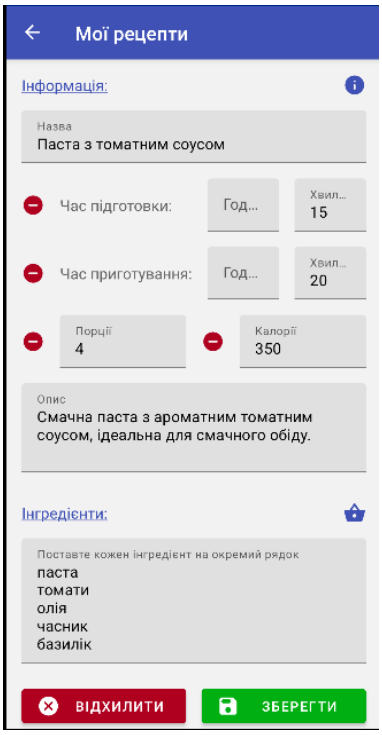

Рисунок 3.6 – Додавання рецепту

Після створення нового рецепту можемо побачити екран з переглядом нашого рецепту, який має 3 вкладки: Інформація, Інгредієнти та Іструкції.

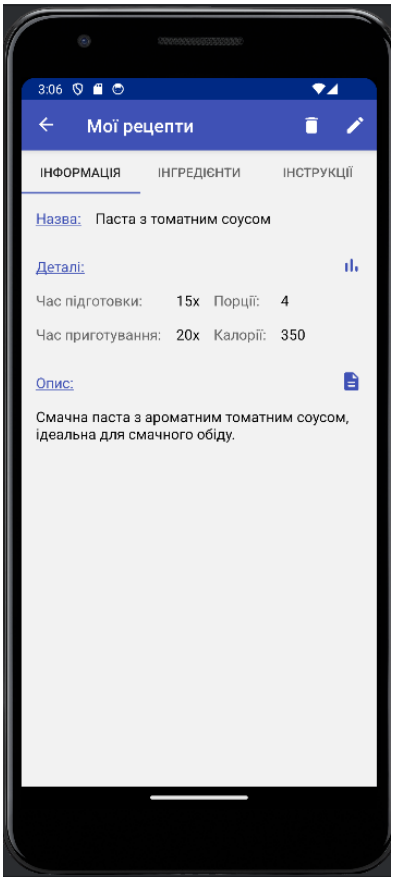

Рисунок 3.7 – Перегляд рецепту

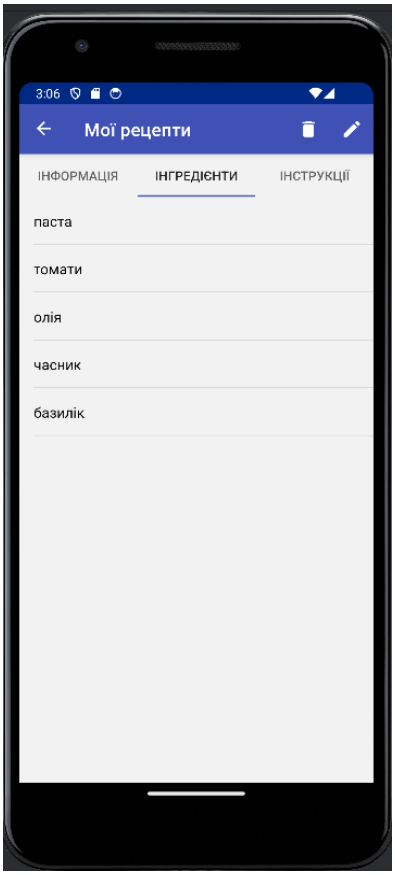

Рисунок 3.6 – Додавання рецепту

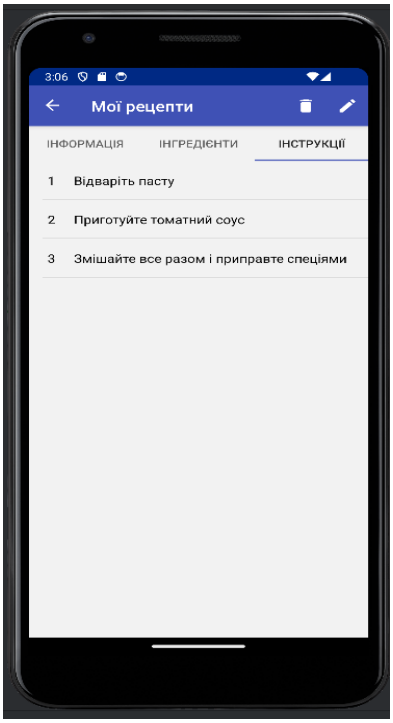

Рисунок 3.9 – Перегляд рецепту

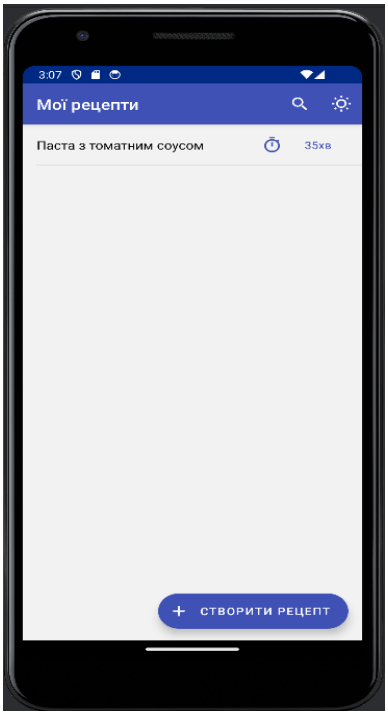

Рисунок 3.10 – Список рецептів

Далі, додамо ще кілька рецептів:

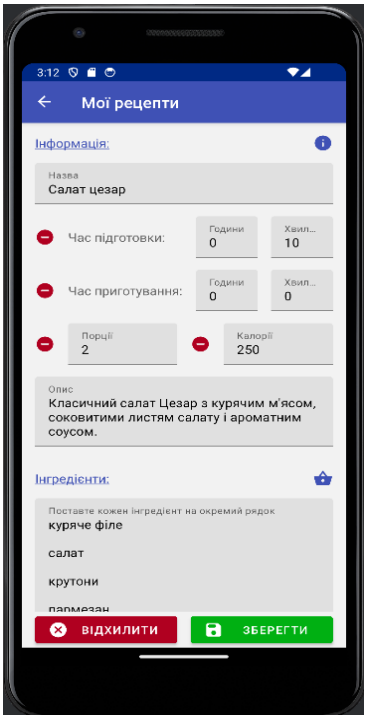

Рисунок 3.11 – Додавання рецепту

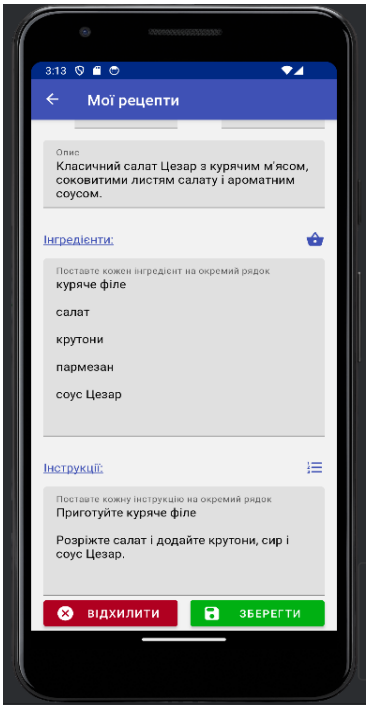

Рисунок 3.12 – Додавання рецепту

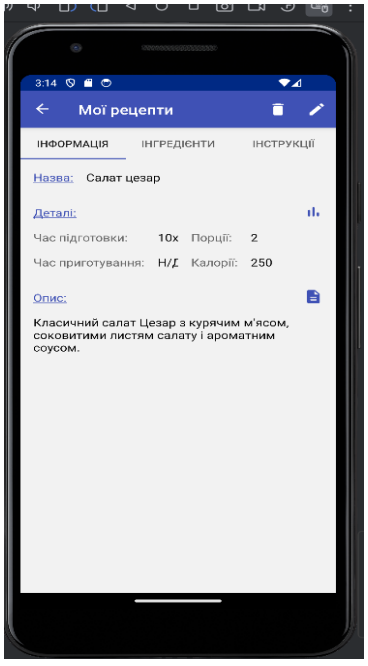

Рисунок 3.13 – Перегляд рецепту

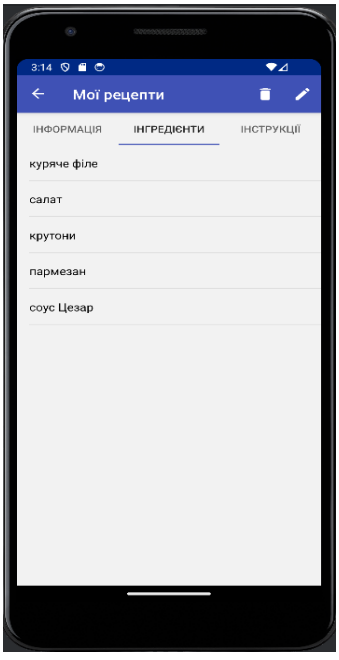

Рисунок 3.14 – Перегляд рецепту

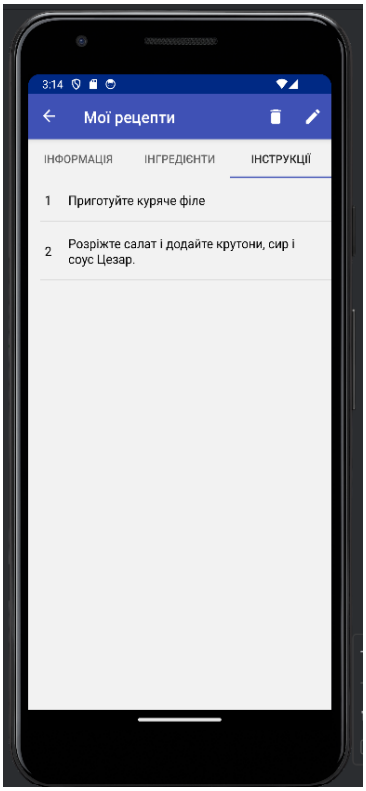

Рисунок 3.15 – Перегляд рецепту

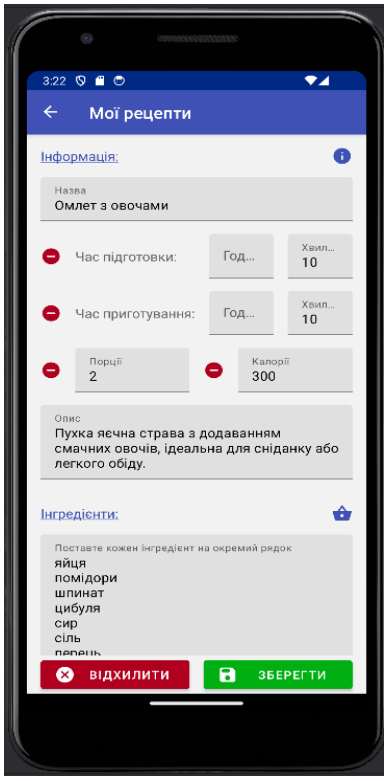

Рисунок 3.16 – Додавання рецепту

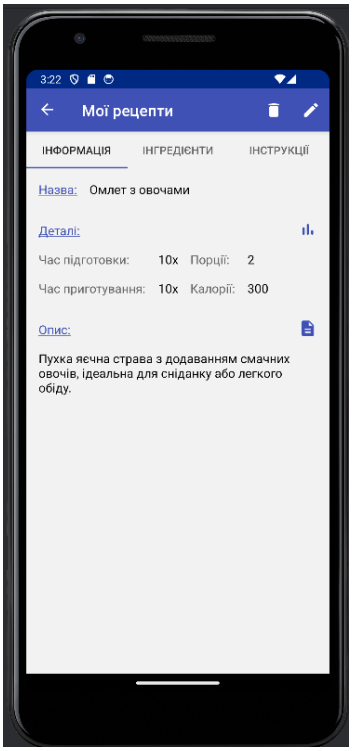

Рисунок 3.17 – Перегляд рецепту

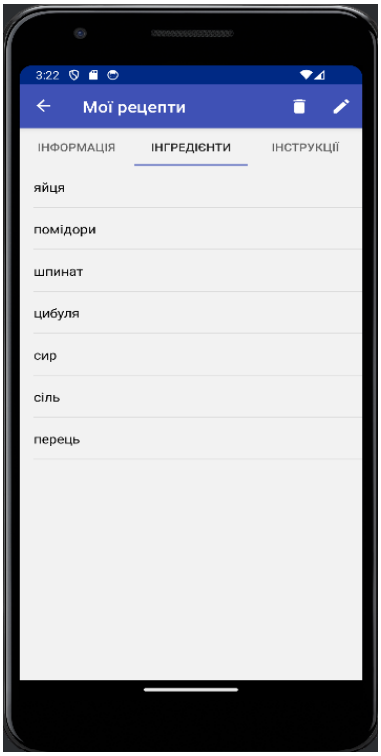

Рисунок 3.18 – Перегляд рецепту

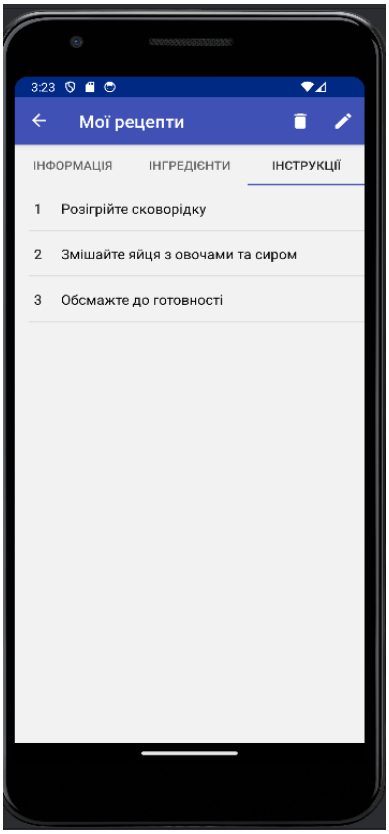

Рисунок 3.19 – Перегляд рецепту

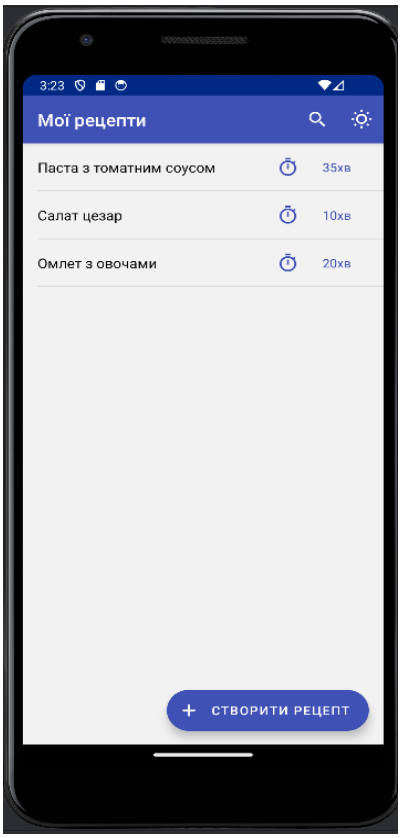

Рисунок 3.20 – Перегляд рецепту

Також, додаток підтримує переключення денного і нічного режимів, для цього, на головному екрані слід натиснути іконку:

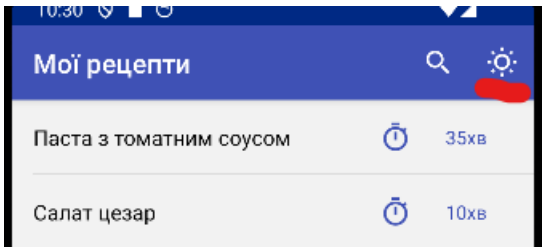

Рисунок 3.21 – Кнопка переключення теми

Після, переключення, можемо побачити «нічну» тему:

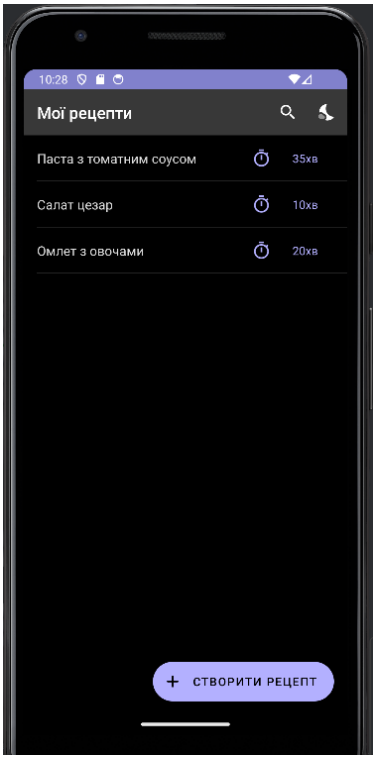

Рисунок 3.21– Темна тема користувацького інтерфейсу

Таким чином, було розроблено додаток, який дозволяє користувачам зручо додавати, зберігати, шукати і переглядати кулінарні рецепти.

ВИСНОВКИ

Народження та розвиток інтернет-технологій різноманітних мобільних пристроїв викликали зміни у способах, якими ми шукаємо, зберігаємо та спільно використовуємо інформацію.

Сучасне суспільство характеризується швидким темпом життя та бажанням мати доступ до інформації в будь-який час і в будь-якому місці. У такому контексті мобільні додатки, а зокрема додатки з рецептами, стають невід'ємною складовою нашого повсякдення, які не лише забезпечують доступ до кулінарних ідей, а й допомагають організувати процес приготування їжі, використовуючи різноманітні функції, такі як списки покупок, обчислення вартості складників та готові шаблони планування страв. Крім того, зростання популярності здорового способу життя, дієтичних обмежень та веганізму спонукає до пошуку нових та цікавих рецептів, які відповідали б унікальним потребам користувачів.

У першому розділі даної бакалаврської роботи було проаналізовано основні риси і особливості розробки програмних кулінарних довідників, охарактеризовано операційну систему Android, а також розглянуто роботу з Android Studio.

Другий розділ був присвячений аналізу і вибору технологій і інструментів розробки додатків для Android, зокрема розглянуто мови програмування, а також системи баз даних.

У третьому розділі описано проектування, розробку, а також тестування розробленого програмного рішеня. Розроблений додаток дозволяє зручно записувати, зберігати, знаходити та переглядати кулінрані рецепти користувача.

Отже, актуальність та важливість розробки додатку з рецептами полягає не лише у полегшенні процесу готування їжі, а й у забезпеченні користувачам доступу до різноманітних кулінарних можливостей, що відповідають сучасним тенденціям та потребам споживачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка мобільних додатків від А до Я: повний гайд [Електронний ресурс] – Режим доступу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-japovnij-gajd/

2. Типи мобільних додатків [Електронний ресурс] – Режим доступу: https://smileukraine.com/ua/mobile-apps/mobile-apps-types

3. Основи програмування на Java [Електронний ресурс] – Режим доступу: https://promoter.net.ua/articles/osnovi-programuvannya-na-java.html

4. Копитко М. Ф. Основи програмування мовою Java / М. Ф. Копитко, К. С. Іванків; Нац. ун-т «ЛНУ ім. Івана Франка». − Львів: Вид-во Нац. ун-ту «ЛНУ ім. Івана Франка», 2016.− 83 с.

5. Рейтинг мов програмування 2023. [Електронний ресурс] – Режим доступу: https://dou.ua/lenta/articles/language-rating-2023/

6. Інструменти і середовища розробки мобільних додатків [Електронний ресурс] – Режим доступу: https://ppt-online.org/341525

7. Розробка мобільних додатків [Електронний ресурс] – Режим доступу: https://webcase.com.ua/uk/razrabotka-mobilnyh-prilozhenij/

8. Figma з нуля [Електронний ресурс] – Режим доступу: https://ux.pub/designis/figma-z-nulia-vchimos-pratsiuvati-u-fighma-55pl

9. Dao…[Електронний ресурс] – Режим доступу: https://developer.android.com/reference/android/arch/persistence/room/Dao

10. Database [Електронний ресурс] – Режим доступу: https://developer.android.com/reference/android/arh/persistence/room/Database..

11. Delete…[Електронний ресурс] – Режим доступу: https://developer.android.com/reference/android/arh/persistence/room/Database

12. Deprecated [Електронний ресурс] – Режим доступу: https://developer.android.com/reference/android/arch/persistence/room/Insert.

13. "HTML та CSS. Розробка та дизайн веб-сайтів" — Джон Дакетт – Режим доступу: http://vk.academy.lv/file/Dakett_J_HTML_i_CSS_Razrabotka_i_dizayn_web_stranic.pdf

14. “Нова велика книга CSS” - Девід Макфарланд – Режим доступу: https://codelibrary.info/online?url=L2Rvd25sb2FkLzE0Mzhfbm92YXlhLWJvbHNoYXlhLWtuaWdhLWNzcy5wZGY=

15. «РОЗРОБЛЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ, КОНСПЕКТ ЛЕКЦІЙ» КПІ ім. І. Сікорського, навчальний посібник. Лекція 1 «Архітектура та базові відомості про платфотму Android» – Режим доступу: https://dspace.chmnu.edu.ua/jspui/bitstream/123456789/423/1/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0%20%D0%BC%D0%BE%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D1%85%20%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BA%D1%96%D0%B2%20%D0%B4%D0%BB%D1%8F.pdf

16. “РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ OS Android” - М.Л. Дворецький, Ю.О. Нездолій, С.В. Дворецька, І.О. Кандиба – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/60e0e698-645d-409b-84b0-b1f49d85d6bc/content#page7

17. “Програмування мовою Java” - Васильєв Олексій Миколайович – Режим доступу: https://books.google.ch/books/about/Програмування_мовою_Ja.html?id=D1lcEAAAQBAJ&redir_esc=y

18. «ОСНОВНІ АСПЕКТИ СТВОРЕННЯ МОБІЛЬНИХ ДОДАТКІВ ТА ВИБІР ІНСТРУМЕНТІВ ЇХ РОЗРОБКИ» - В. Ічанська, С. І. Улько – Режим доступу: https://doi.org/10.26906/SUNZ.2020.1.074

Додаток А

MainActivity.kt

```
package com.unoknowbo.recime

import android.content.Context
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.findNavController
import androidx.navigation.ui.NavigationUI
import com.unoknowbo.recime.ui.edit.EditRecipeFragment
import com.unoknowbo.recime.util.dayNightPrefListener
import com.unoknowbo.recime.util.setDayNightMode

interface MyOnBackPressed {
    fun onBackPressed()
}

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        setTheme(R.style.AppTheme)
        super.onCreate(savedInstanceState)

        // Register the day/night listener and set the day/night mode to what was previously saved.
```

```
        val sharedPrefs = this.getPreferences(Context.MODE_PRIVATE)

sharedPrefs?.registerOnSharedPreferenceChangeListener(dayNightPrefListener)
        setDayNightMode(sharedPrefs)

        setContentView(R.layout.activity_main)

        val navController = this.findNavController(R.id.nav_host_fragment)
        NavigationUI.setupActionBarWithNavController(this, navController)
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = this.findNavController(R.id.nav_host_fragment)
        return if (!calledMyOnBackPressed()) {
            navController.navigateUp()
        } else {
            true
        }
    }

    override fun onBackPressed() {
        if (!calledMyOnBackPressed()) {
            super.onBackPressed()
        }
    }

    private fun calledMyOnBackPressed(): Boolean {
```

```
            val navHostFragment =
this.supportFragmentManager.findFragmentById(R.id.nav_host_fragment)
            val fragments = navHostFragment?.childFragmentManager?.fragments
            for (f in fragments ?: listOf()) {
                if (f is EditRecipeFragment) {
                    (f as MyOnBackPressed).onBackPressed()
                    return true
                }
            }
            return false
        }
    }
```

EditRecipeFragment.kt

```
package com.unoknowbo.recime.ui.edit

import android.app.Activity
import android.app.AlertDialog
import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.view.inputmethod.InputMethodManager
import android.widget.ImageView
```

```
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.databinding.DataBindingUtil
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.fragment.findNavController
import com.google.android.material.textfield.TextInputEditText
import com.unoknowbo.recime.MyOnBackPressed
import com.unoknowbo.recime.R
import com.unoknowbo.recime.database.RecimeDatabase
import com.unoknowbo.recime.databinding.FragmentEditRecipeBinding

class EditRecipeFragment : Fragment(), MyOnBackPressed {

    var recipeId: Long = -1L

    private var _binding: FragmentEditRecipeBinding? = null

    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {

        // Get a reference to the binding object and inflate the fragment views
        val xBinding: FragmentEditRecipeBinding = DataBindingUtil.inflate(
            inflater, R.layout.fragment_edit_recipe, container, false)
```

```
        _binding = xBinding

        val args = this.arguments
        val recipeId = requireNotNull(args?.getLong("recipeId"))

        // Set the lifecycle of the LiveData to this fragment's lifecycle
        binding.lifecycleOwner = this

        // Set the editRecipeViewModel
        val application = requireNotNull(this.activity).application
        val recipeDao = RecimeDatabase.getInstance(application).recipeDao
        val ingredientDao =
RecimeDatabase.getInstance(application).ingredientDao
        val instructionDao =
RecimeDatabase.getInstance(application).instructionDao
        val viewModelFactory =
          EditRecipeViewModelFactory(
            recipeDao,
            ingredientDao,
            instructionDao,
            recipeId
          )
        val editRecipeViewModel =
          ViewModelProvider(this,
viewModelFactory)[EditRecipeViewModel::class.java]

        // Data binding variables: recipe, ingredients, instructions from database
```

```
            editRecipeViewModel.recipe.observe(viewLifecycleOwner, Observer {
                it?.let {
                    binding.recipe = it
                }
            })
            editRecipeViewModel.ingredients.observe(viewLifecycleOwner, Observer
{
                it?.let {
                    binding.ingredients = it
                }
            })
            editRecipeViewModel.instructions.observe(viewLifecycleOwner,
Observer {
                it?.let {
                    binding.instructions = it
                }
            })

            // Navigate back to the recipe fragment on save

editRecipeViewModel.navigateBackToRecipeAfterSave.observe(viewLifecycleOwne
r, Observer {
                if (it == true) {
                    navigateBack(editRecipeViewModel.recipeId)
                    editRecipeViewModel.doneNavigating()
                }
            })
```

```
binding.editRecipeCancelButton.setOnClickListener {
    getCancelAlertDialog(recipeId)?.show()
}

// Execute view model's save when save button is clicked
binding.editRecipeSaveButton.setOnClickListener {
    editRecipeViewModel.save(
        binding.editRecipeNameEditText.text.toString(),
        binding.editRecipePrepTimeHoursEditText.text.toString(),
        binding.editRecipePrepTimeMinutesEditText.text.toString(),
        binding.editRecipeCookTimeHoursEditText.text.toString(),
        binding.editRecipeCookTimeMinutesEditText.text.toString(),
        binding.editRecipeServingsEditText.text.toString(),
        binding.editRecipeCaloriesEditText.text.toString(),
        binding.editRecipeDescriptionEditText.text.toString(),
        binding.editRecipeIngredientsEditText.text.toString(),
        binding.editRecipeInstructionsEditText.text.toString()
    )
}

// Add click listeners to each add/remove button.
binding.editRecipePrepTimeAddRemove.setOnClickListener {
    addRemoveOnClickListener(
        binding.editRecipePrepTimeAddRemove,
        listOf(
            binding.editRecipePrepTimeHoursEditText,
            binding.editRecipePrepTimeMinutesEditText
        ),
```

```
        binding.editRecipeConstraintLayout,
        activity
    )
}
binding.editRecipeCookTimeAddRemove.setOnClickListener {
    addRemoveOnClickListener(
        binding.editRecipeCookTimeAddRemove,
        listOf(
            binding.editRecipeCookTimeHoursEditText,
            binding.editRecipeCookTimeMinutesEditText
        ),
        binding.editRecipeConstraintLayout,
        activity
    )
}
binding.editRecipeServingsAddRemove.setOnClickListener {
    addRemoveOnClickListener(
        binding.editRecipeServingsAddRemove,
        listOf(binding.editRecipeServingsEditText),
        binding.editRecipeConstraintLayout,
        activity
    )
}
binding.editRecipeCaloriesAddRemove.setOnClickListener {
    addRemoveOnClickListener(
        binding.editRecipeCaloriesAddRemove,
        listOf(binding.editRecipeCaloriesEditText),
        binding.editRecipeConstraintLayout,
```

```
                activity
            )
        }

        // Add on focus change listeners to each specific (i.e. prep time, cook time,
servings,
        // calories).
        binding.editRecipePrepTimeMinutesEditText.setOnFocusChangeListener
{ _, hasFocus ->
            specificOnFocusChange(
                binding.editRecipePrepTimeAddRemove,
                listOf(
                    binding.editRecipePrepTimeHoursEditText,
                    binding.editRecipePrepTimeMinutesEditText
                ),
                hasFocus
            )
        }
        binding.editRecipePrepTimeHoursEditText.setOnFocusChangeListener    {
_, hasFocus ->
            specificOnFocusChange(
                binding.editRecipePrepTimeAddRemove,
                listOf(
                    binding.editRecipePrepTimeHoursEditText,
                    binding.editRecipePrepTimeMinutesEditText
                ),
                hasFocus
            )
```

```
            }
            binding.editRecipeCookTimeMinutesEditText.setOnFocusChangeListener
{ _, hasFocus ->
                specificOnFocusChange(
                    binding.editRecipeCookTimeAddRemove,
                    listOf(
                        binding.editRecipeCookTimeHoursEditText,
                        binding.editRecipeCookTimeMinutesEditText
                    ),
                    hasFocus
                )
            }
            binding.editRecipeCookTimeHoursEditText.setOnFocusChangeListener  {
_, hasFocus ->
                specificOnFocusChange(
                    binding.editRecipeCookTimeAddRemove,
                    listOf(
                        binding.editRecipeCookTimeHoursEditText,
                        binding.editRecipeCookTimeMinutesEditText
                    ),
                    hasFocus
                )
            }
            binding.editRecipeServingsEditText.setOnFocusChangeListener     {     _,
hasFocus ->
                specificOnFocusChange(
                    binding.editRecipeServingsAddRemove,
                    listOf(binding.editRecipeServingsEditText),
```

```
                hasFocus
            )
        }
        binding.editRecipeCaloriesEditText.setOnFocusChangeListener { _,
hasFocus ->
            specificOnFocusChange(
                binding.editRecipeCaloriesAddRemove,
                listOf(binding.editRecipeCaloriesEditText),
                hasFocus
            )
        }

        return binding.root
    }

    override fun onBackPressed() {
        getCancelAlertDialog(recipeId)?.show()
    }

    private fun isAdd(imageView: ImageView): Boolean {
        return imageView.tag == R.drawable.ic_add
    }

    // Handle pushing an add/remove button.
    private fun addRemoveOnClickListener(
        addRemoveImageView: ImageView,
        listOfEditTexts: List<TextInputEditText>,
        layout: ConstraintLayout,
```

```
        activity: Activity?
    ) {
        if (isAdd(addRemoveImageView)) {
            addRemoveImageView.setImageResource(R.drawable.ic_remove)
            addRemoveImageView.tag = R.drawable.ic_remove
            showKeyboard(activity, listOfEditTexts.first())

        } else {
            addRemoveImageView.setImageResource(R.drawable.ic_add)
            addRemoveImageView.tag = R.drawable.ic_add
            listOfEditTexts.map { it.setText("") }

            // If the focus is on one of the TextViews that just got cleared by the user
clicking
            // the remove button, then hide the keyboard and clear the focus.
            val currentFocusedView = activity?.currentFocus
            currentFocusedView?.let {
                for (editText in listOfEditTexts) {
                    if (it == editText) {
                        layout.requestFocus()
                        hideKeyboard(activity)
                        break
                    }
                }
            }
        }
    }
```

```
        // Handle the focus change of an EditText that is associated with an add/remove
button.
        private fun specificOnFocusChange(
            addRemoveImageView: ImageView,
            listOfEditTexts: List<TextInputEditText>,
            hasFocus: Boolean

        ) {
            if (isAdd(addRemoveImageView) && hasFocus) {
                addRemoveImageView.setImageResource(R.drawable.ic_remove)
                addRemoveImageView.tag = R.drawable.ic_remove
            } else if (
                !isAdd(addRemoveImageView) &&
                !hasFocus &&
                listOfEditTexts.all { it.text.toString() == "" }
            ) {
                addRemoveImageView.setImageResource(R.drawable.ic_add)
                addRemoveImageView.tag = R.drawable.ic_add
            }
        }

        private fun getCancelAlertDialog(recipeId: Long): AlertDialog? {
            return activity?.let {
                val builder = AlertDialog.Builder(it)
                builder.apply {
                    setPositiveButton(R.string.discard) { _, _ ->
                        navigateBack(recipeId)
                    }
```

```
                setNegativeButton(R.string.cancel, null)
                setTitle(
                    if (recipeId == (-1).toLong()) R.string.discard_recipe
                    else R.string.discard_changes
                )
                setMessage(
                    if (recipeId == (-1).toLong()) R.string.discard_recipe_message
                    else R.string.discard_changes_message
                )
                setIcon(R.drawable.ic_discard)
            }
            builder.create()
        }
    }

    // Navigate back to RecipeFragment if a recipe was created or one was updated; navigate back to
    // RecipesFragment if a recipe under creation is discarded. Hide the keyboard if it is visible.
    private fun navigateBack(recipeId: Long) {
        val navController = this.findNavController()
        if (recipeId == (-1).toLong()) {
            navController.popBackStack()
        } else {
            val args = Bundle()
            args.putLong("recipeId", recipeId)
            navController.navigate(
                R.id.action_editRecipeFragment_to_recipeFragment,
```

```
                args,
            )
        }
        hideKeyboard(activity)
    }

    private fun showKeyboard(activity: Activity?, editText: TextInputEditText) {
        activity?.let {
            val inputMethodManager =
                activity.getSystemService(Context.INPUT_METHOD_SERVICE) as
InputMethodManager

            editText.requestFocus()
            inputMethodManager.showSoftInput(editText,
InputMethodManager.SHOW_IMPLICIT)
        }
    }

    private fun hideKeyboard(activity: Activity?) {
        activity?.let {
            val inputMethodManager =
                activity.getSystemService(Context.INPUT_METHOD_SERVICE) as
InputMethodManager

            // Check if a view has focus
            val currentFocusedView = activity.currentFocus
            currentFocusedView?.let {
                inputMethodManager.hideSoftInputFromWindow(
```

```
                    currentFocusedView.windowToken,
InputMethodManager.HIDE_NOT_ALWAYS
                )
            }
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

EditRecipeViewModel.kt

```
package com.unoknowbo.recime.ui.edit

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.unoknowbo.recime.database.ingredient.Ingredient
import com.unoknowbo.recime.database.ingredient.IngredientDao
import com.unoknowbo.recime.database.instruction.Instruction
import com.unoknowbo.recime.database.instruction.InstructionDao
import com.unoknowbo.recime.database.recipe.Recipe
import com.unoknowbo.recime.database.recipe.RecipeDao
import kotlinx.coroutines.*
```

```
class EditRecipeViewModel (
    private val recipeDao: RecipeDao,
    private val ingredientDao: IngredientDao,
    private val instructionDao: InstructionDao,
    var recipeId: Long
) : ViewModel() {

    /** Data binding values */
    val recipe = recipeDao.getRecipe(recipeId)
    val ingredients = ingredientDao.getRecipeIngredients(recipeId)
    val instructions = instructionDao.getRecipeInstructions(recipeId)


    /** Coroutine setup from Udacity's "Developing Android Apps with Kotlin"
*/

    // viewModelJob allows us to cancel all coroutines started by this ViewModel.
    private val viewModelJob = Job()

    // A CoroutineScope keeps track of all coroutines started by this ViewModel.
    private val uiScope = CoroutineScope(Dispatchers.Main + viewModelJob)

    // Cancels all coroutines when the ViewModel is cleared.
    override fun onCleared() {
        super.onCleared()
        viewModelJob.cancel()
    }
```

```
        /** Navigation handling after save or cancel */

        // Variable to signify that saving is done and we should navigate out of
EditRecipeFragment
        private val _navigateBackToRecipeAfterSave =
MutableLiveData<Boolean>()
        val navigateBackToRecipeAfterSave
            get() = _navigateBackToRecipeAfterSave


        // Signify that navigation is complete
        fun doneNavigating() {
            _navigateBackToRecipeAfterSave.value = null
        }


        /** Save changes */

        fun save(
            nameString: String,
            prepTimeHoursString: String,
            prepTimeMinutesString: String,
            cookTimeHoursString: String,
            cookTimeMinutesString: String,
            servingsString: String,
            caloriesString: String,
```

```
            descriptionString: String,
            ingredientsString: String,
            instructionsString: String
        ) {
            uiScope.launch {
                withContext(Dispatchers.IO) {
                    val prepTimeEstimateInMinutes =
                        if (prepTimeHoursString == "" && prepTimeMinutesString == "")
-1 else
                            (if (prepTimeHoursString == "") 0 else
prepTimeHoursString.toInt() * 60) +
                                    (if (prepTimeMinutesString == "") 0 else
prepTimeMinutesString.toInt())
                    val cookTimeEstimateInMinutes =
                        if (cookTimeHoursString == "" && cookTimeMinutesString == "")
-1 else
                            (if (cookTimeHoursString == "") 0 else
cookTimeHoursString.toInt() * 60) +
                                    (if (cookTimeMinutesString == "") 0 else
cookTimeMinutesString.toInt())
                    val servings = if (servingsString == "") -1 else servingsString.toInt()
                    val calories = if (caloriesString == "") -1 else caloriesString.toInt()

                    if (recipeId == (-1).toLong()){
                        recipeId = recipeDao.insert(
                            Recipe(
                                nameString,
                                cookTimeEstimateInMinutes,
```

```
                    prepTimeEstimateInMinutes,
                    descriptionString,
                    servings,
                    calories
                )
            )
        } else {
            recipeDao.updateName(nameString, recipeId)
            recipeDao.updatePrepTime(prepTimeEstimateInMinutes, recipeId)
            recipeDao.updateCookTime(cookTimeEstimateInMinutes,
recipeId)
            recipeDao.updateServings(servings, recipeId)
            recipeDao.updateCalories(calories, recipeId)
            recipeDao.updateDescription(descriptionString, recipeId)
        }

        val ingredients = getIngredientsFromString(ingredientsString)
        // Remove old ingredients
        ingredientDao.deleteIngredients(recipeId)
        for (ingredient in ingredients) {
            ingredientDao.insert(ingredient)
        }

        val instructions = getInstructionsFromString(instructionsString)
        // Remove old instructions
        instructionDao.deleteInstructions(recipeId)
        for (instruction in instructions) {
            instructionDao.insert(instruction)
```

```
                }
            }
            _navigateBackToRecipeAfterSave.value = true
        }
    }

    private fun getIngredientsFromString(ingredientsString: String):
List<Ingredient> {
        val ingredientEntries = ingredientsString
            .split("\n").map {s: String -> s.trim()}.filterNot {s: String -> s == ""}
        val ingredients = mutableListOf<Ingredient>()
        for ((order, entry) in ingredientEntries.withIndex()) {
            ingredients.add(Ingredient(recipeId, entry, order))
        }
        return ingredients
    }

    private fun getInstructionsFromString(instructionsString: String):
List<Instruction> {
        val instructionEntries = instructionsString
            .split("\n").map {s: String -> s.trim()}.filterNot {s: String -> s == ""}
        val instructions = mutableListOf<Instruction>()
        for ((order, entry) in instructionEntries.withIndex()) {
            instructions.add(Instruction(recipeId, entry, order)
            )
        }
        return instructions
    }
```

```
}
```

EditRecipeViewModelFactory.kt

```
package com.unoknowbo.recime.ui.edit

/**
 * Based on the implementation provided in Udacity's "Developing Android Apps with Kotlin"
 */
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import com.unoknowbo.recime.database.ingredient.IngredientDao
import com.unoknowbo.recime.database.instruction.InstructionDao
import com.unoknowbo.recime.database.recipe.RecipeDao

/**
 * Provides the RecipeDao and recipeId to the ViewModel.
 */
class EditRecipeViewModelFactory(
    private val recipeDao: RecipeDao,
    private val ingredientDao: IngredientDao,
    private val instructionDao: InstructionDao,
    private val recipeId: Long
) : ViewModelProvider.Factory {
    @Suppress("unchecked_cast")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(EditRecipeViewModel::class.java)) {
```

```
            return EditRecipeViewModel(
                recipeDao,
                ingredientDao,
                instructionDao,
                recipeId
            ) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

RecipeFragment.kt

```
package com.unoknowbo.recime.ui.recipe

import android.app.AlertDialog
import android.os.Bundle
import android.view.*
import androidx.databinding.DataBindingUtil
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.fragment.findNavController
import androidx.viewpager2.widget.ViewPager2
import com.google.android.material.tabs.TabLayoutMediator
import com.unoknowbo.recime.R
import com.unoknowbo.recime.database.RecimeDatabase
import com.unoknowbo.recime.databinding.FragmentRecipeBinding
```

```
class RecipeFragment : Fragment() {
    private lateinit var recipeAdapter: RecipeAdapter
    private lateinit var viewPager: ViewPager2

    private var recipeViewModel: RecipeViewModel? = null

    private var _binding: FragmentRecipeBinding? = null

    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {

        // This fragment has an options menu
        setHasOptionsMenu(true)

        // Retrieve the recipeId from the arguments bundle
        val args = this.arguments
        val recipeId = requireNotNull(args?.getLong("recipeId"))

        // Get a reference to the binding object and inflate the fragment views
        val xBinding: FragmentRecipeBinding = DataBindingUtil.inflate(
            inflater, R.layout.fragment_recipe, container, false
        )
```

```
            _binding = xBinding

            // Set the lifecycle of the LiveData to this fragment's lifecycle
            binding.lifecycleOwner = this

            // Set the recipeViewModel
            val application = requireNotNull(this.activity).application
            val dataSource = RecimeDatabase.getInstance(application).recipeDao
            val viewModelFactory = RecipeViewModelFactory(dataSource, recipeId)
            recipeViewModel =
                ViewModelProvider(this,
viewModelFactory)[RecipeViewModel::class.java]


recipeViewModel?.navigateBackToRecipes?.observe(viewLifecycleOwner,  Observer
{
                if (it == true) {
                    this.findNavController().popBackStack()
                    recipeViewModel?.doneDeleting()
                }
            })

            return binding.root
        }

        override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
            val args = this.arguments
```

```
    val recipeId = requireNotNull(args?.getLong("recipeId"))
    recipeAdapter = RecipeAdapter(this, recipeId)
    viewPager = binding.recipePager
    viewPager.adapter = recipeAdapter
    val tabLayout = binding.recipeTabLayout
    TabLayoutMediator(tabLayout, viewPager) { tab, position ->
        tab.text = when (position) {
            0 -> getString(R.string.information)
            1 -> getString(R.string.ingredients)
            2 -> getString(R.string.instructions)
            else -> error("Recipe tab position > 2")
        }
    }.attach()
}

override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
    inflater.inflate(R.menu.fragment_recipe, menu)
    super.onCreateOptionsMenu(menu, inflater)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    val recipeId = requireNotNull(this.arguments?.getLong("recipeId"))
    val args = Bundle()
    args.putLong("recipeId", recipeId)

    when (item.itemId) {
        R.id.fragment_recipe_edit -> {
            this.findNavController().navigate(
```

```
                R.id.action_recipeFragment_to_editRecipeFragment,
                this.arguments,
            )
        }

        R.id.fragment_recipe_delete -> {
            getConfirmDeleteAlertDialog()?.show()
        }
    }
    return super.onOptionsItemSelected(item)
}

private fun getConfirmDeleteAlertDialog(): AlertDialog? {
    return activity?.let {
        val builder = AlertDialog.Builder(it)
        builder.apply {
            setPositiveButton(R.string.delete) { _, _ ->
                recipeViewModel?.delete()
            }
            setNegativeButton(R.string.cancel, null)
            setTitle(R.string.delete_recipe)
            setMessage(R.string.delete_recipe_message)
            setIcon(R.drawable.ic_delete_on_surface_high_emphasis)
        }
        builder.create()
    }
}
```

```
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```