Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технологій Java

Виконав: студент ___IV___ курсу
групи ____1ІП-20б_____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Щербацький Б.І.
(прізвище та ініціали)
Керівник: к.т.н., доц. Катєльніков Д.І
(прізвище та ініціали)
Рецензент: к.т.н., доц. Снігур А. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«__10__» ___червня____ 2024 р.

ВНТУ – 2024

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

Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк _____
«12» березня 2024 р.

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

Щербацькому Борису Ігоровичу

1. Тема роботи – розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технологій Java

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80

2. Строк подання студентом роботи 3 червня 2024 р.

3. Вихідні дані до роботи: інструмент для розробки мобільного додатку на платформі Android Android Studio; мова розробки Java.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка архітектури та алгоритмів програмного продукту; розробка програмного продукту; тестування програми, висновки; список використаних джерел , додатки.

5. Перелік графічного матеріалу: мета; об'єкт та предмет дослідження; діаграми варіантів використання; структурна схема компонентів додатку; загальний алгоритм роботи; приклади аналогів; тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	к.т.н., доц. кафедри ПЗ Катєльніков Д.І	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ п\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз та обґрунтування вибору методу розробки та постановка задачі дослідження	13.03.24 – 18.03.24	Виконано
2	Розробка архітектури та алгоритмів програмного продукту	18.03.24 – 26.03.24	Виконано
3	Розробка програмного продукту	27.03.24 – 21.04.24	Виконано
4	Тестування програми	22.04.24 – 16.05.24	Виконано
5	Оформлення матеріалів до захисту БДР	17.05.24 – 30.05.24	Виконано

Студент _____ (підпис) **Б.І. Щербацький** (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ (підпис) **Д.І Катєльнік** (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 93 сторінок формату А4, на яких є 25 рисунків, 16 таблиць, список використаних джерел містить 20 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз мобільного додатку по пошуку рецептів. Встановлено об’єкт, предмет, завдання та методи дослідження. Робота спрямована на скорочення часу пошуку рецептів за вибраними. Для реалізації цілей було розроблено мобільний додаток “Pro100 Cooked ”.

Запропоновано метод сортування та фільтрації рецептів по їх інгрідієнтам без строгої прив’язки до них та модель роботи мобільного додатку.

Розроблено алгоритми та програми для мобільного додатку для оцінювання характеристик рецептів на етапі сортування.

Мобільний додаток розроблений з використанням мови програмування Java, для створення його базових механік, та використано середовище Android Studio для створення його каркасу.

Отримані в бакалаврській дипломній роботі результати можна використати для пошуку рецептів у мобільному додатку .

Ключові слова: мобільний додаток, користувач, сортування та фільтрація, рецепти.

Annotation

The bachelor thesis consists of 93 pages of A4 format, on which there are 25 figures, 16 tables, the list of used sources contains 20 names.

A detailed analysis of the recipe search mobile application was Reciperied out in the bachelor thesis. The object, subject, tasks and research methods are established. The work is aimed at reducing the time of searching for recipes by favorites. To realize the goals, the mobile application "Pro100 Cooked" was developed.

A method of sorting and filtering recipes by their ingredients without strict binding to them and a model of mobile application operation are proposed.

Algorithms and programs for a mobile application have been developed for evaluating the characteristics of recipes at the sorting stage.

The mobile application is developed using the Java programming language to create its basic mechanics, and the Android Studio environment is used to create its framework.

The results obtained in the bachelor thesis can be used to search for recipes in the mobile application.

Keywords: mobile application, user, sorting and filtering, recipes.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. "Сортування та фільтрація даних" має суттєве значення, оскільки ці концепції є основою для ефективної обробки даних у різних галузях, включаючи комп'ютерні науки, бази даних, аналітику даних та багато інших. Розуміння та використання методів сортування та фільтрації даних дозволяє покращити продуктивність та ефективність обробки інформації, зокрема в умовах зростаючого обсягу даних, що стає все більшою проблемою у сучасному світі. Знання про різноманітні алгоритми сортування та методи фільтрації даних має важливе практичне застосування в розв'язанні реальних завдань, від оптимізації роботи баз даних до підтримки розумних рішень у сферах фінансів, медицини, торгівлі тощо. Крім того, ця тема залишається актуальною в контексті наукових досліджень, оскільки вона стимулює пошук нових алгоритмів, оптимізацію існуючих підходів та виявлення тенденцій у розвитку обробки даних. Таким чином, вибір даної теми дослідження обумовлений її широким застосуванням, практичною значимістю та потенціалом для досягнення наукового прогресу. Створення мобільного додатка для консультування з приготування їжі в контексті сортування та фільтрація даних дозволить зберегти час, отримати здорові рецепти та навчитися приготовлювати смачну та поживну їжу вдома, створюючи таким чином для них зручне і приємне харчування.

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

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є скорочення часу для отримання рецептів смачної та поживної їжі за рахунок реалізації сортування та фільтрації даних при розробці мобільного програмного додатку.

Основними задачами дослідження є:

– розробити за допомогою Android Studio та Java мобільний додаток з пошуку потрібного рецепту приготування їжі;

– розробити метод та алгоритм фільтрування та сортування рецептів за категоріями та інгредієнтам;

– розробити модель мобільного додатку;

– розробити дизайн мобільного додатку, що буде зручним у використанні непідготовленим користувачем;

– програмно реалізувати мобільний додаток з пошуку та інструкцій по приготуванні їжі ;

– провести тестування програмного продукту.

Об’єкт дослідження: процес розробки мобільних застосунків для персонального консультування з приготування їжі на платформі Android.

Предмет дослідження: методи та засоби розробки модулів сортування та фільтрації даних для персонального консультування з приготування їжі.

Методи дослідження. Протягом дослідження було використано: теорія людино-машинної взаємодії для побудови інтерфейсу, теорія баз даних для організації зберігання інформації, методи об’єктно-орієнтованого програмування для розробки архітектури класів та модулів програмного забезпечення.

Новизна отриманих результатів.

Подальшого розвитку отримала технологія фільтрації та сортування первинних даних, у якій на відміну від існуючих підходів до обробки даних, застосовуються сучасні засоби мови Java: Java Stream API та лямбда-виразів, що дозволило ефективно обробляти дані, виконуючи операції фільтрації, трансформації, згортання та сортування. Окрім того, це поєднання робить вихідні тексти програми такими, що легко читаються, що значно полегшить масштабування в майбутньому.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень

запропоновано алгоритми та розроблено програмні засоби модуля сортування та фільтрації даних для пошуку рецептів в мобільному додатку.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій праці [1], опублікованій у співавторстві, автору належать такі результати: розробка мобільного застосунку-персонального консультанта з приготування їжі на платформі android з використанням технологій java.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів «СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ» (ВНТУ, 2024).

Публікації. Основні результати досліджень опубліковано в наукових праці у матеріалах конференцій.

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

Зі стрімким розвитком технологій мобільні додатки стають дедалі популярнішими в різних галузях. Пошук рецептів за продуктами є перспективним у цій галузі, що надає можливість дозволяючи їм ефективно використовувати наявні продукти, що зменшує зайві витрати та сприяє уникненню марної їжі. Розробка мобільного застосунку - персонального консультанта з приготування їжі є важливим завданням у цьому плані, оскільки дозволяє створювати інноваційні рішення, які покращують користувацький досвід та надають конкурентні переваги проекту.

Однією з основних переваг використання фільтрації даних у мобільних застосунку є можливість обирати саме наявні у вас продукту для пошуку рецептів за ними. Ця технологія може бути використана в різних сферах, таких як роздрібна торгівля, реклама, освіта, розваги тощо. Однак розробка програмних засобів для впровадження фільтрації даних в мобільному додатку є складним завданням, що вимагає спеціалізованих навичок і знань.

Щоб вирішити цю проблему, багато компаній звертаються до компаній-розробників програмного забезпечення, які спеціалізуються на розробці мобільних застосунків. Ці компанії роблять детальний аналіз, що дозволяє розробити стратегію для подальшого розвитку та покращення продукту. [2].

Ще однією перевагою використання програмних засобів для впровадження фільтрації даних в мобільний додаток є можливість користувачам зручний доступ до інформації про рецепти навіть під час перебування в дорозі або в магазині. Мобільний формат дозволяє людям швидко знаходити страви, використовуючи наявні продукти, що допомагає уникнути марної їжі та економить час на шукання рецептів.

Отже, розробка програмних засобів для мобільного застосунку з використанням фільтрації даних є критично важливим завданням у сучасному цифровому ландшафті. Використовуючи досвід спеціалізованих компаній з розробки програмного забезпечення, компанії можуть створювати інноваційні рішення, які покращують користувацький досвід та надають конкурентні переваги. Зважаючи на постійний розвиток технології фільтрації даних, важливо бути в курсі останніх розробок і тенденцій у цій галузі, щоб залишатися конкурентоспроможними та актуальними .

1.2 Порівняльний аналіз аналогів

Використання технології фільтрації даних є популярним на проміжку всього часу і ця технологія інтегрується в різні додатки і системи, включаючи мобільні застосунки. Розробка програмних засобів для реалізації мобільної застосунку з використанням фільтрації даних може допомогти зменшити час пошуку потрібної вам інформації, підвищити залученість та утримання користувачів. До найбільш відомих рішень відносять:

- Cookpad;
- Прості рецепти від DIL Studio;
- SuperCook;
- My Recipe Box

Одним із аналогів по пошуку рецептів в мобільній застосунках є популярна застосунок Cookpad (рис. 1.1). інтернет-корпорація, що спеціалізується на розміщенні кулінарних рецептів [3]. Місія Cookpad — зробити щоденне приготування їжі захоплюючим. Ми віримо, що домашня кухня - ключ до щасливого та здорового життя людей, суспільства і планети. Ми пропонуємо домашнім кулінарам у всьому світі допомагати один одному, поділившись рецептами та кулінарними порадами[4].

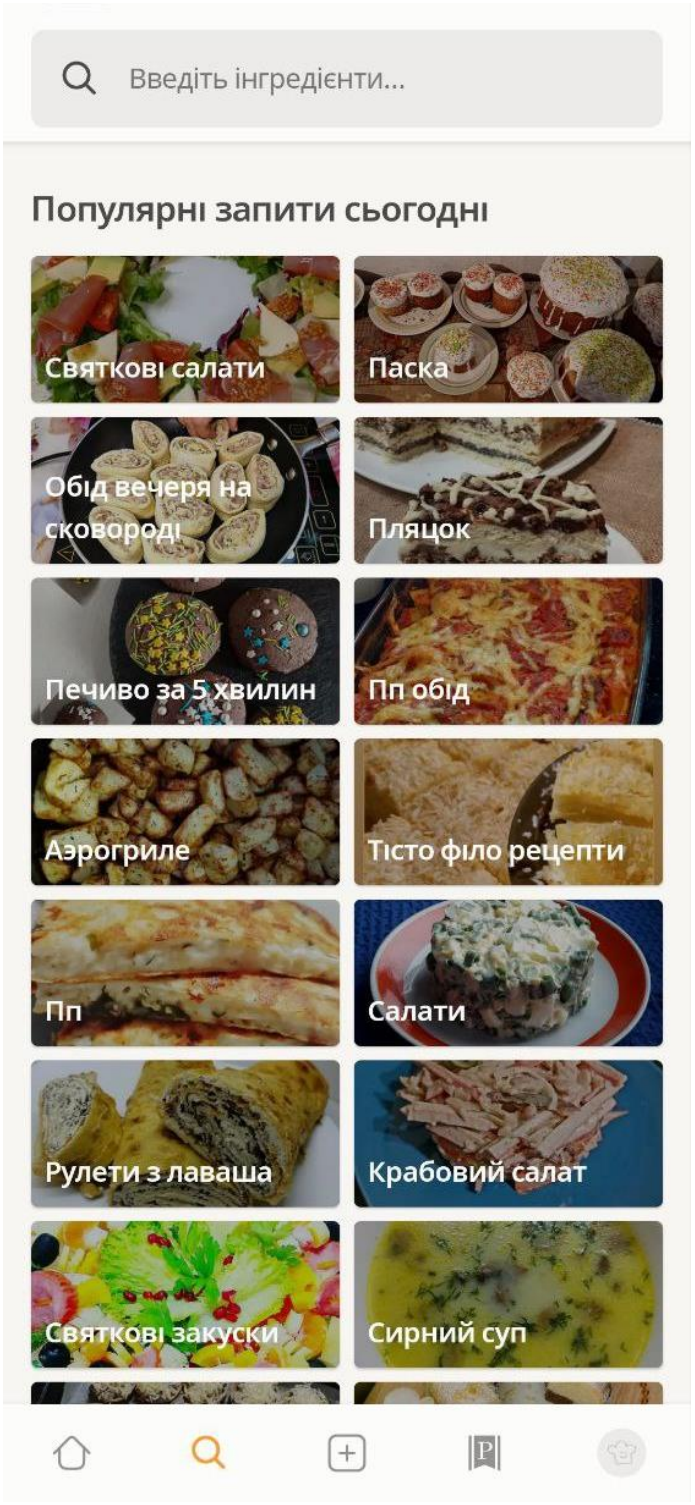


Рисунок 1.1 – Приклад роботи додатку Cookpad

Інший приклад Прості рецепти від DIL Studio – Основними функціями якого є пошук рецептів за прийомом їжі Сніданок, Перекус, Обід, Вечеря. Ось що вони говорять про себе – Прості рецепти на кожен день. Книга рецептів без Інтернету. Готувати легко! Перед вами безкоштовна книга рецептів з простими

стравами: прості рецепти супів, овочеві страви, м'ясні, птиця і риба, смачні й прості гарніри, холодні і гарячі закуски, рибні салати, проста солодка випічка і легкі десерти, рецепти напоїв, страви з кабачків та багато іншого без Інтернету.. [5](рис. 1.2).

Рисунок 1.2 – Приклад роботи додатку Прості рецепти від DIL Studio

І останнім з обраних аналогів є мобільний додаток Supercook (рис. 1.3), створений для іноземного ринку та все рівно користується популярністю в

Україні із за різноманітності за пошуком рецептів. . Про себе вони пишуть так – Скільки разів ви шукали ідеальний рецепт, але виявляли, що вам не вистачає одного чи кількох інгредієнтів? Скільки разів ви відкривали холодильник і думали про себе - що я можу зробити? Скільки разів ви викидали інгредієнт, тому що не могли зрозуміти, як його використовувати до закінчення терміну придатності? СуперКухар на допомогу! [6].

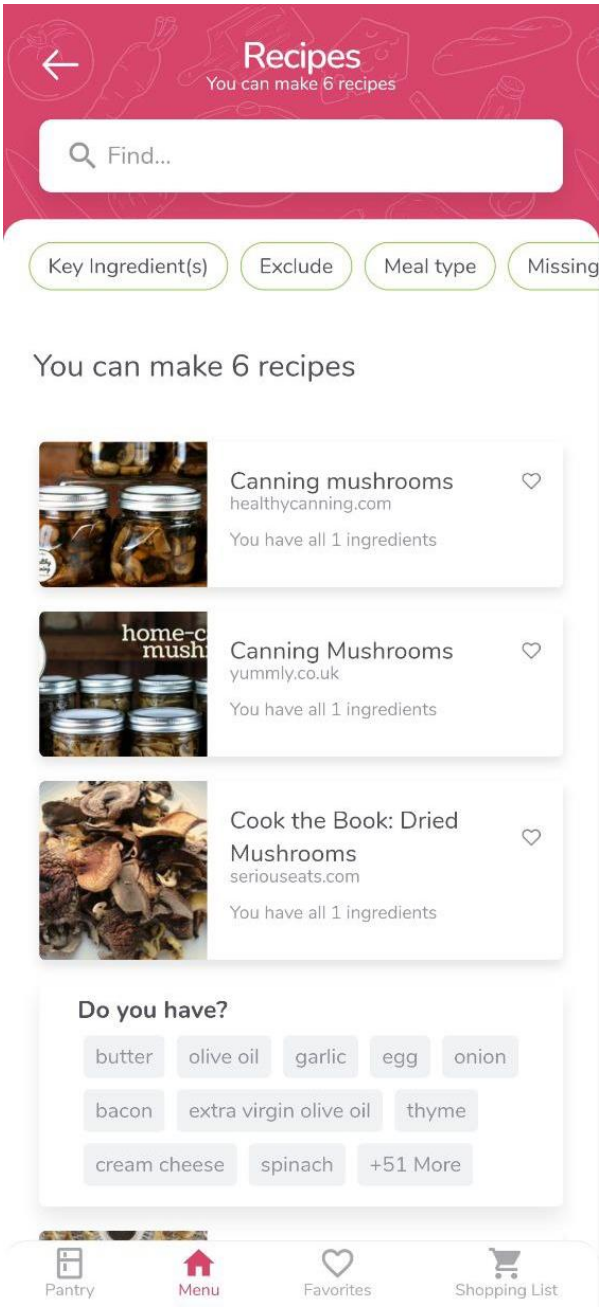


Рисунок 1.3 – Приклад роботи додатку Supercook

Останнім серед обраних аналогів будо вирішено взяти один із най популярніших мобільний додаток My Recipe Box(рис. 1.4). У якому рецепт знаходиться за допомогою пошуку в інтернеті та сканування на рахунок рецепту. Тепер знайти улюблений рецепт стало ще проще! My Recipe Box - це повноцінна кулінарна книга для всіх любителів готовок. За допомогою даного додатка Ви можете створити свою власну бібліотеку рецептів і з легкістю ними керувати Тепер Ви зможете зібрати всі свої улюблені рецепти в одному додатку [7]. Це те, що вони пишуть про себе.

Рисунок 1.4 – Приклад роботи додатку Supercook

Після аналізу усіх аналогів визначено їхні переваги та недоліки і проведено порівняння із розроблюваним мобільним додатком «Pro100 Cooked».

Результати порівняння аналогів зведено в табл. 1.1.

Критерії	Cookpad	Прості рецепти від DIL Studio	Supercook	My recipe Box	Pro100 Cooked
Робота без інтернет з’єднання	-	+	-	+	+
Фільтри за категорією	+	+	+	-	+
Функція не строгого пошуку	-	-	+	-	+
Змога додавання власних рецептів	+	+	-	+	+
Наявність української мови	+	+	-	+	+
Загальна оцінка	60%	80%	40%	60%	100%

Таблиця 1.1 – Порівняльні характеристика онлайн платформ

Відповідно до таблиці порівняльних характеристик розробка власного мобільного застосунку з персонального консультанта з їжі є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити новий досвід та можливості застосування технології фільтрації даних.

Отже, розробка програмних засобів для впровадження в мобільний застосунок з використанням фільтрації даних відкриває можливість швидкого та зручного пошуку з широкими можливостями для взаємодії зі даними, а також підвищення рівня залученості та утримання користувачів. Розуміючи потенційні можливості застосування технології фільтрації даних та вимоги до розробки

програмного забезпечення, компанії можуть створювати інноваційні та привабливі мобільні застосунки, які використовують технологію фільтрації для зростання та успіху.

1.3 Аналіз методів розв’язання поставленої задачі

Розробка програмних засобів мобільного застосунку - персонального консультанта з приготування їжі вимагає ретельного розгляду найкращих підходів до вирішення проблеми. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних засобів.

Одним з варіантів вирішення проблеми розробки програмних засобів для реалізації мобільного застосунку персонального консультанта з приготування їжі з використанням фільтрації даних є використання окремої бібліотеки Java Stream API, яка дозволяє інтегрувати фільтрацію в існуючі рішення. Однак такий підхід має певні недоліки. Хоча це швидкий спосіб розробки модуля, він є лінивою, тому фільтрація не виконується безпосередньо після виклику filter(), а лише при виклику термінальної операції, яка призводить до виконання проміжних операцій[8]. Цей підхід також може призвести до значного обсягу роботи, покладеного на команду розробників, яка хоче інтегрувати це рішення. Тому такий підхід не варто застосовувати.

Серед нововведень, які були привнесені в мову Java з виходом JDK 8, окремо стоять лямбда-вирази. Лямбда представляє набір інструкцій, які можна виділити в окрему змінну і багаторазово викликати в різних місцях програми. Основу лямбда-виразу становить лямбда-оператор, який представляє стрілку ->. Цей оператор поділяє лямбда-вираз на дві частини: ліва частина містить список параметрів виразу, а права власне представляє тіло лямбда-виразу, де виконуються всі дії. їх основним призначенням є підтримка функціонального програмування в мові Java, яка дозволяє писати більш чистий і компактний код [9]. Фільтрування в лямбда-виразів працює так після створення потоку даних (наприклад, списку або потоку), до нього можна застосувати метод filter() як і

для Stream API . Лямбда-вираз виступає у ролі предикату, який приймає кожен елемент потоку та повертає true, якщо елемент відповідає умові, або false, якщо ні. Таким чином, використання лямбда-виразів для фільтрації дозволяє коротко та зрозуміло визначати умови відбору елементів зі списку або потоку даних, що робить код більш зрозумілим та зручним для читання та надає змогу зручне задання умов для фільтрації даних. Однак цей засіб ускладнює читабельність коду та значно збільшує Важкість відлагодження

Використання рекурсії – це метод програмування, який дає змогу функції багаторазово викликати себе доти, доки не буде виконано умову завершення [10]. Працює він наступним чином за допомогою вхідного списоку (або масива) та допоміжних параметрів (наприклад, індекс поточного елемента або додатковий результат). Під час кожного виклику перевіряється, чи виконується умова для поточного елемента масива. Якщо умова виконується, елемент додається до результату, і метод викликає сам себе для наступного елемента. Рекурсія продовжується, доки всі елементи не будуть перевірені, після чого повертається фінальний результат. Хоча рекурсивний підхід може бути потужним і досить зручним для деяких завдань він має масу недоліків, а саме, кожен рекурсивний виклик вимагає додавання контексту до стеку викликів та є не є ефективним з боку використання даних та часу роботи.

Усі вище перераховані методи мають свої сильні та слабкі сторони , тому проект передбачає комбінування в Java 8, що забезпечує підтримку лямбда-виразів так і Java Stream API [11].

Використання даного методу дозволить при створенні програмного забезпечення відкриває широкий простір для ефективної обробки та аналізу даних. Використання Stream API дозволяє легко та зручно виконувати різноманітні операції над колекціями даних, такі як фільтрація, трансформація, згортання та сортування. Лямбда-вирази забезпечують зручний спосіб визначення коротких та зрозумілих функцій, що робить код більш компактним та читабельним. Крім того, можливість паралельної обробки даних в Stream API сприяє підвищенню продуктивності за рахунок використання багатоядерних

систем. Загалом, це поєднання дозволяє розробникам створювати ефективне та легко читабельне програмне забезпечення для обробки великих обсягів даних.

1.4 Постановка задачі розробки

Проаналізувавши переваги та недоліки існуючих застосунків з використанням засобів фільтрації було визначено наступні завдання, які необхідно виконати для розробки мобільного додатку:

- визначити найбільш ефективний підхід для фільтрації даних;
- розробити алгоритм фільтрації даних;
- розробити зручний та адаптивний графічний інтерфейс мобільного застосунку;
- розробити мобільний застосунок персонального консультанта з приготування їжі з фільтрацією даних.
- провести тестування мобільного застосунку згідно поставлених задач.

1.5 Висновки

У цьому розділі було проведено порівняння відомих платформ та їхні способи фільтрації даних, таких як: Cookpad; Прості рецепти від DIL Studio; SuperCook; My Recipe Box;.

У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності серед учасників ринку завдяки своїй зручності та доступності. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що вони не надають можливості працювати без інтернет з’єднання та працюють з строгим пошуком. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного мобільного додатку.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Вибір архітектури програмного компоненту

Архітектура додатка – це план його структури та організації. Вона визначає, як різні компоненти додатка взаємодіятимуть один з одним, і відіграє ключову роль у його продуктивності, масштабованості та зручності обслуговування[12].

Почнемо з патерну MVC зображеного на рисунку 2.1 — він був створений в 1970-х роках і його метою було розбиття будь-якої програми, з якою взаємодіє користувач, на три логічні блоки:

Model — це бізнес-логіка додатку, в ньому відбуваються найважчі процеси (обробка, сортування, запис даних, звернення до бази даних, генерація вебсторінки тощо), і після виконаної події відправляє результат до View.

View — це інтерфейс, з яким взаємодіє користувач, візуальна частина (наприклад, сторінка сайту). Задача View — зчитати інформацію, введену користувачем, і передати її в конкретний метод або функцію контролера, а також зчитувати дії користувача (натискання кнопки або оновлення сторінки, що теж викликає метод або функцію з контролера).

Controller — приймає дані або подію від View і може діяти таким чином, наприклад: за допомогою виключень обирає конкретний метод Model, який треба викликати, спираючись на дані або подію від View; формує об’єкт з отриманих параметрів, який передаємо до Model за допомогою метода, функції Model або створюємо http–запит до model.[13]

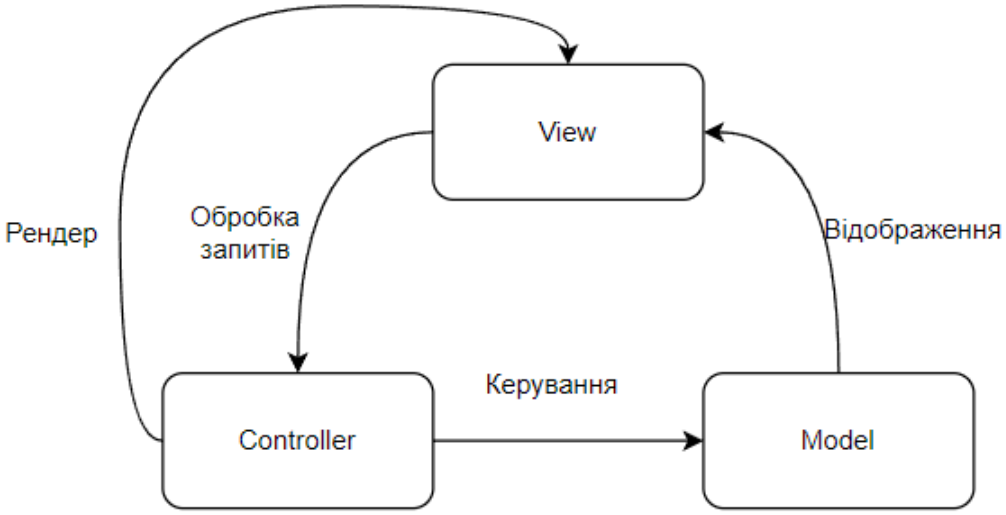


Рисунок 2.1 – Архітектура MVC патерну

Ми розглянули з чого все почалося і тепер готові піти далі до патерну який обрав я. MVP (Model-View-Presenter) — це патерн проектування, який покликаний відокремити логіку представлення від бізнес-логіки і полегшити тестування і підтримку застосунку. Основна відмінність MVP від MVC (Model-View-Controller) полягає у ролі посередника, який виконує модуль Presenter. Нижче наведено детальний опис патерну MVP, а також його блок-схема на рисунку 2.2 .

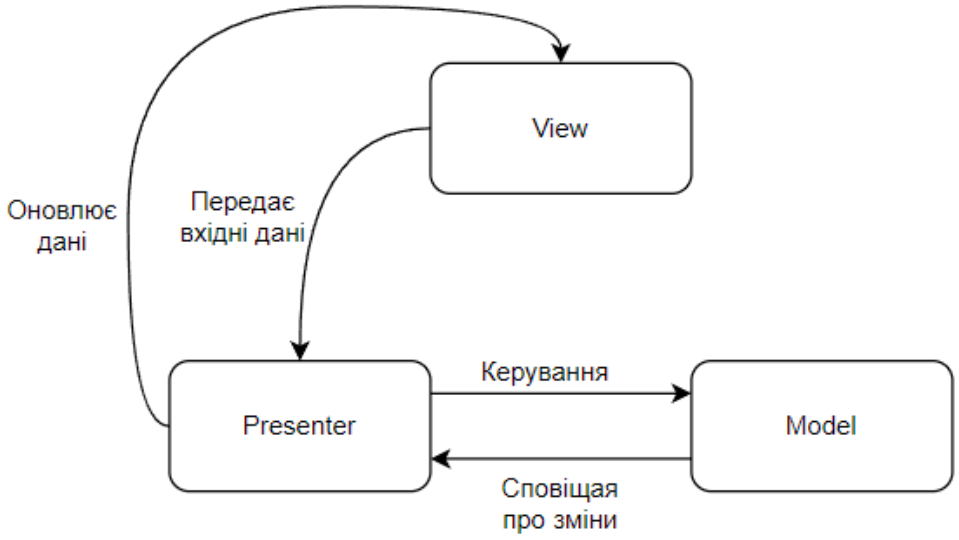


Рисунок 2.2 – Архітектура MVP патерну

Model — відповідає за дані та бізнес-логіку. Вона надає дані Presenter'у, але не має жодних знань про View або інтерфейс користувача.

View — відповідає за відображення даних і взаємодію з користувачем. Вона не має прямого доступу до Model, а лише взаємодіє з Presenter'ом. Приймає користувацькі події (наприклад, натискання кнопок, введення тексту). Передає ці події Presenter'у.

Presenter — посередник між View та Model. Він отримує дані від Model, обробляє їх (якщо потрібно) і передає їх View для відображення. Також він обробляє події від View, викликає відповідні методи в Model, і отримує оновлені дані.

Взаємодія між модулями:

View -> Presenter: View передає вхідні дані користувача Presenter'у.

Presenter -> Model: Presenter звертається до Model для отримання даних або виконання дій.

Model -> Presenter: Model повертає дані або результати дій Presenter'у.

Presenter -> View: Presenter передає оброблені дані або результати дій View для відображення.

Також для більш детального опису взаємодії між основними модулями розроблюваного додатку було використано структурну карту Константайна. Структурні карти Константайна є моделлю зв’зків між модулями програми. Вузли структурних карт відповідають модулям і областям даних, потоки зображають міжмодульні зв'язки. На діаграмі спеціальними вузлами зображають циклічні і умовні виклики модулів, а потоки проходять через ці спеціальні вузли. Потоки, що зображають міжмодульні зв'язки між даними і функціями також зображають на діаграмі спеціальними вузлами, а стрілками вказують напрями потоків. [14]. Розроблену структурну карту Константайна зображено на рисунку 2.3.

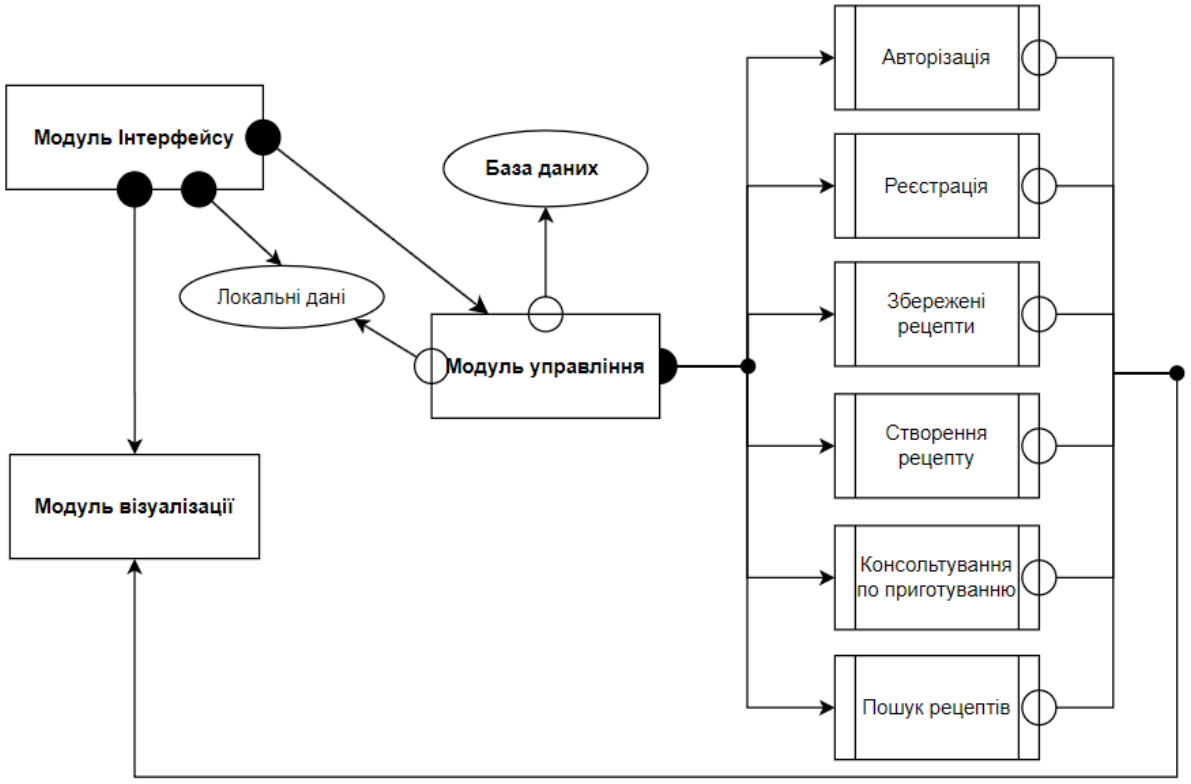


Рисунок 2.3 – Структурна карта Константайна додатку

Переваги через що було обрано MVP над MVC — це чітке розділення обов'язків адже кожен модуль має свою чітко визначену роль та оскільки Presenter містить логіку, його можна легко тестувати без необхідності створення складних UI-тестів, його легко змінювати або оновлювати Model без змін у View або Presenter. Це дозволяє використовувати різні реалізації Model (наприклад, локальні або серверні дані). Саме через ці особливості роботи MVP вирішено було обрати самий цей патерн адже він є потужним інструментом для розробки застосунків з багатою логікою, забезпечуючи гнучкість та легкість підтримки коду.

2.2 Обґрунтування вибору інтерфейсу

Вибір інтерфейсу Android Studio обгрунтований кількома факторами:

1. Інтеграція з Android SDK: Android Studio розроблена спеціально для роботи з платформою Android. Вона повністю інтегрована з Android SDK, що

забезпечує максимальну сумісність та підтримку для розробки на Android.

2. Оновлення та підтримка: Android Studio постійно оновлюється та підтримується Google. Це означає, що користувачі отримують доступ до нових функцій, інструментів та виправлень помилок.

3. Багатофункціональність: Android Studio має широкий набір інструментів для розробки, таких як емулятори, дебагери, аналізатори профілювання, що полегшують розробку та тестування додатків на платформі Android.

4. Підтримка мов програмування: Android Studio підтримує кілька мов програмування, включаючи Java та Kotlin, офіційно рекомендований Google для розробки Android додатків.

5. Спільнота користувачів: Android Studio має велику та активну спільноту користувачів, яка надає підтримку, допомогу та різноманітні ресурси для розробників Android додатків.

6. Наявність інструментів для Google Play**: Android Studio має інтегровані інструменти для розгортання та публікації додатків на Google Play, що полегшує процес розгортання та оновлення додатків.

Враховуючи ці фактори, Android Studio є найоптимальнішим вибором для розробки Android додатків, оскільки вона надає повний набір інструментів та підтримку від компанії Google.

2.3 Проектування бази даних

Для розроблюваного додатка, що зберігатиме інформацію про користувачів та рецепти, доцільно використовувати базу даних через великий обсяг інформації. При проектуванні бази даних важливо створити універсальне відношення, яке містить всі атрибути, що використовуються в базі даних. Універсальне відношення є відправною точкою для проектування невеликих баз даних. Атрибути – це поля або комбінації полів, що містять унікальні значення для кожного запису в таблиці. Їх основна мета – ідентифікувати та встановлювати зв’язок між різними таблицями в базі даних[15]. Кожен атрибут характеризує певну сутність бази даних,

відображаючи окремі характеристики об'єктів.

Розроблювана база даних повинна включати в себе атрибути, які представлені в таблиці 2.1.

Таблиця 2.1 – Перелік атрибутів універсального відношення

№	Назва атрибута	Ім’я поля	Коментар
1	ID користувача	UserID	Ідентифікаційний номер зареєстрованого користувача
2	Логін користувача	UserLogin	Логін користувача
3	Пароль користувача	UserPassword	Пароль користувача
4	Ім’я користувача	UserName	Ім’я користувача
5	Статус користувача	UserStatus	Статус користувача
6	ID рецепту	RecipeID	Ідентифікаційний номер рецепту
7	Назва рецепту	RecipeName	Назва рецепту
8	Інгредієнти рецепту	RecipeIngredients	Інгредієнти рецепту
9	Тип кухні	RecipeCusinePreference	Тип кухні рецепту
10	Витрати часу на приготування	RecipeCookingtime	Час на приготування
11	Складність	RecipeDifficulty	Складність приготування рецепту
12	Консультування для приготування	RecipeInfo	Інструкція по приготуванню

Відповідно до таблиці 2.1 універсальне відношення має наступний вигляд: R(UserID, UserLogin, UserPassword, UserName, UserStatus , RecipeID, RecipeName, RecipeIngredients, RecipeCusinePreference, RecipeCookingtime, RecipeDifficulty, RecipeInfo). Потужність універсальної множини – 12.

На основі інформаційних об’єктів бази даних,виділяються такі сутності та їх атрибути:

1. Users (<UserID>, UserLogin, UserPassword, UserName).
2. Recipe (<RecipeID >, RecipeName, RecipeIngredients, RecipeCusinePreference, RecipeCookingtime, RecipeDifficulty, RecipeInfo).

Відповідно до отриманих сутностей, було розроблено структуру бази даних, яка зображена на рисунку 2.4.

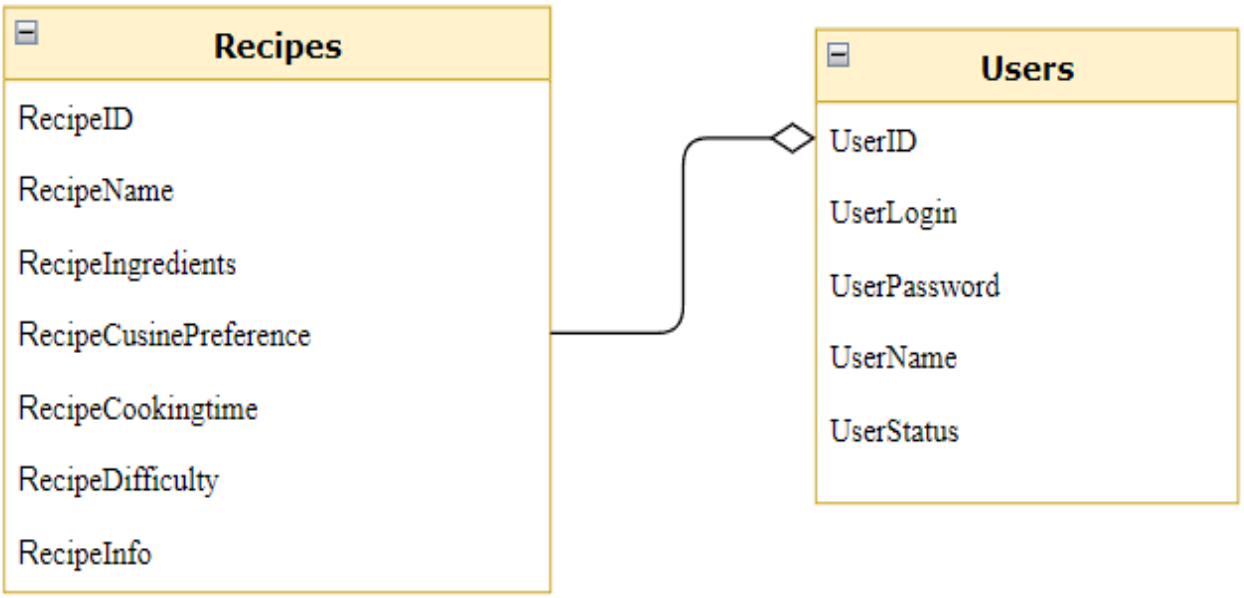


Рисунок 2.4 – Структура бази даних додатку

Сутність Users знаходяться у відношенні 1:N до сутності Recipes оскільки у одного користувача може бути декілька записів про рецепти. Тому, в сутності Recipes. Функціональна залежність пов'язує атрибути в одному відношенні з єдиним значенням в іншому[16].

2.4 Розробка блок-схеми роботи програми

Розроблюваний додаток буде складатись з таких вікон що представлена на рисунку 2.5 :

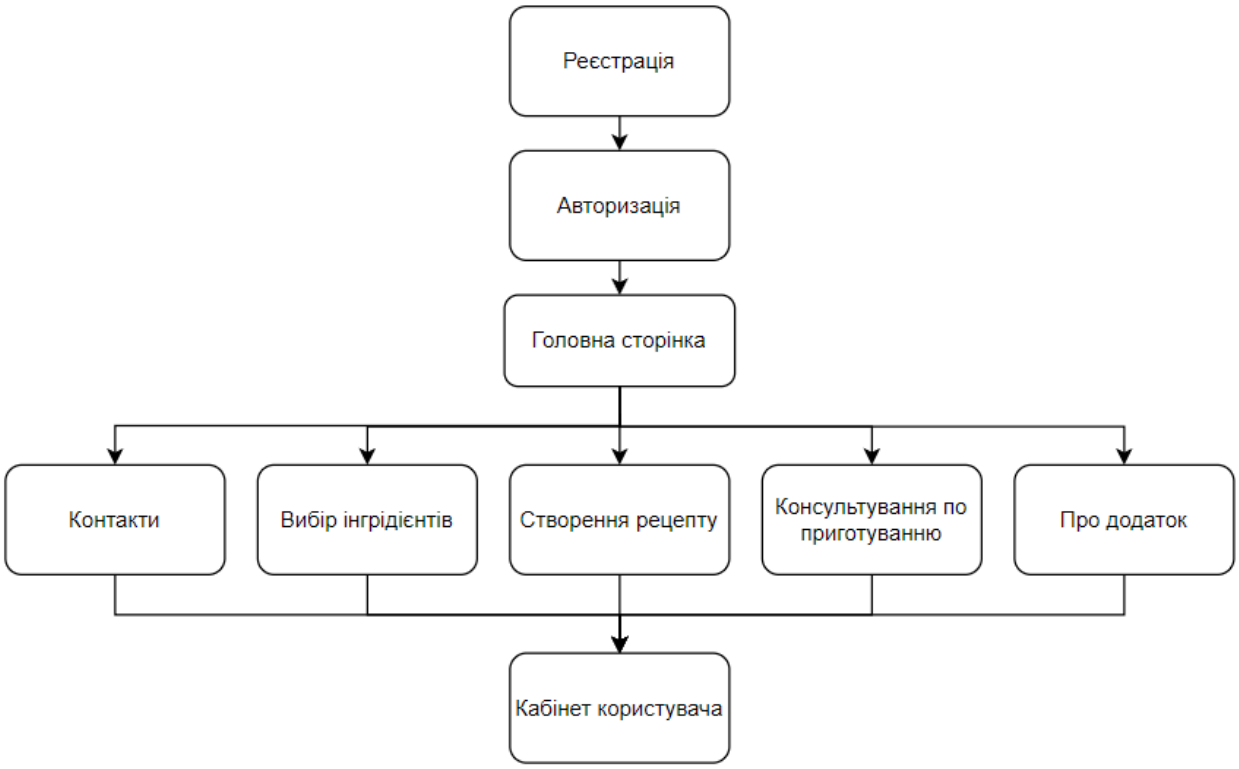


Рисунок 2.5 – Структурна схема додатку

1. «Реєстрація» – вікно реєстрації користувача.

2. «Авторизація» – вікно авторизації користувача.

3. «Головна сторінка» – головне вікно програми з пошуком потрібних рецептів.

4. «Контакти » – вікно для зв’язку з розробником та змогою повідомити про помилку.

5. Вибір інгредієнтів – вікно вибору інгредієнтів.

6. «Створення рецепту» – вікно для створення рецепту.

7. «Консультування по приготуванню» – вікно з інформацією та інструкцією до приготування рецепту.

8. «Про додаток» – вікно з інформацією про додаток та змогою зв’язатися з розробником для .

9. «Кабінет користувача » – вікно кабінету користувача де відображено персональні дані про користувача .

2.5 Розробка блок-схеми алгоритму фільтрування та сортування

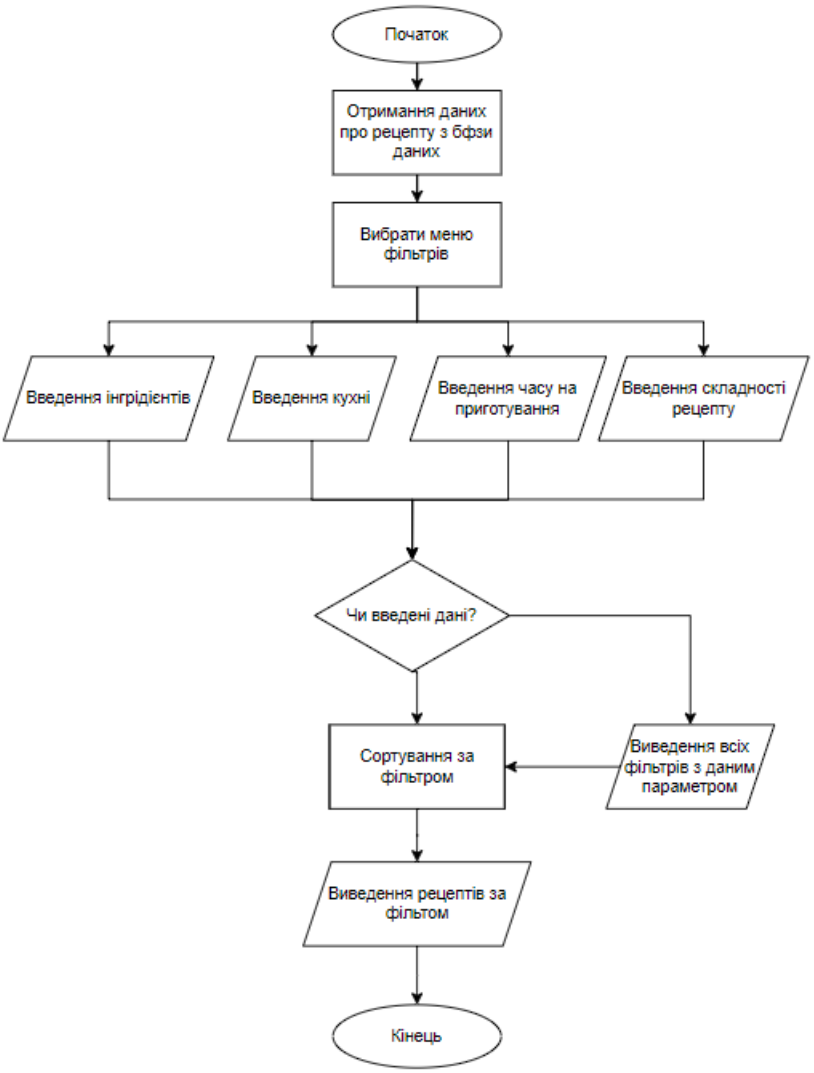


Рисунок 2.6 – Блок-схема алгоритму фільтрації та сортування рецептів

Опис алгоритму фільтрації та сортування рецептів зображених на рисунку 2.6:

Для початку користувач має змогу обрати всі рецепти з бази даних

Далі він має змогу обрати потрібний йому фільтри:

1. Фільтрація за інгредієнтами: Зі списку відбираються тільки ті рецепти, які містять всі зазначені користувачем інгредієнти.
2. Фільтрація за вибором кухні: Зі списку відбираються тільки ті рецепти, які містять всі зазначені уподобання солодкі, гострі, екзотичні та інші.
3. Фільтрація за часом приготування: Зі списку відбираються тільки ті рецепти, час приготування яких не перевищує вказаного значення.

4. Фільтрація за складністю у приготувані: Зі списку відбираються тільки ті рецепти які може приготувати людина з вказаним діапазоном навичок.

Після чого відбувається збір відфільтрованих рецептів: Всі рецепти, які відповідають всім заданим критеріям, збираються в окремий список .

Далі сортування відфільтрованих рецептів: Список відфільтрованих рецептів сортується за критеріями вказаним користувачем.

Та нарешті виведення результатів: Відсортований і відфільтрований список рецептів виводиться на екран для перегляду користувачем, якщо не було обрано фільтр то буде показано всі рецепти за даним фільтром..

Цей алгоритм дозволяє користувачу легко знайти рецепти, що відповідають його вимогам, та відсортувати їх за бажаним параметром для зручного вибору.

2.6 Розробка кольорової гамми інтерфейса

Необхідно підібрати кольорову гаму інтерфейсу, адже коли справа доходить до візуалізації даних, колір особливо важливий. Кольорова схема задає тон образів і кожен колір служить для подання унікальних фрагментів інформації.

Кольори, що ви використовуєте у вашій візуалізації даних – це більше, ніж просто одна ідея. Кольорова схема має силу відображати тип даних, які ви показуєте, їх відносини, відмінності між категоріями, і багато іншого[17].

Для вибору кольорової гами інтерфейсу було обрано круг кольорів Ітена, який зображено на рисунку 2.7 .

Рисунок 2.7 – Круг кольорів Ітена

Кольорове коло Іттена складається з кількох основних елементів, які допомагають зрозуміти взаємодію кольорів та їх гармонійне поєднання. Ось основні елементи кольорового кола Іттена:

1. Три первинні кольори: червоний, жовтий і синій. Вони є основними кольорами, які не можна отримати шляхом змішування інших кольорів.

2. Три вторинні кольори: оранжевий, зелений і фіолетовий. Вони утворюються шляхом змішування первинних кольорів:

- Оранжевий = червоний + жовтий
- Зелений = жовтий + синій
- Фіолетовий = синій + червоний

3. Шість третинних кольорів: ці кольори отримуються шляхом змішування первинного кольору з сусіднім вторинним кольором, наприклад:

- Жовто-оранжевий
- Червонувато-оранжевий
- Червонувато-фіолетовий
- Синьо-фіолетовий
- Синьо-зелений
- Жовто-зелений

Застосування кольорового кола Іттена широко використовується в різних сферах мистецтва та дизайну для досягнення гармонії кольорів та створення візуально привабливих композицій. Ось основні способи застосування кольорового кола Іттена:

1. Гармонія кольорів: Кольорове коло допомагає в підборі гармонійних колірних схем. Наприклад:

- Комплементарні кольори: кольори, розташовані один проти одного на колі (наприклад, червоний і зелений).
- Аналогові кольори: кольори, що знаходяться поруч на колі (наприклад, синій, синьо-зелений, зелений).
- Триадні кольори: три кольори, рівномірно розташовані навколо кола

(наприклад, червоний, жовтий, синій).

2. Контрасти кольорів: Коло допомагає зрозуміти, як різні кольори можуть створювати контрасти або взаємодіяти один з одним.

3. Психологія кольору: Іттен також досліджував, як кольори впливають на настрій і емоції людини, що є важливим аспектом у мистецтві та дизайні.

За основу кольорової гами було обрано зелений та білий кольори, оскільки ці кольори символізують свіжість, чистоту та природність. Зелений колір асоціюється з природою, здоров'ям і спокоєм, що робить його ідеальним вибором для додатків, пов'язаних з їжею, здоров'ям та благополуччям. Білий колір, в свою чергу, символізує чистоту, простоту і прозорість, сприяючи легкому сприйняттю інформації та створюючи враження простору і свіжості.

Таке поєднання кольорів допомагає створити приємну і комфортну візуальну атмосферу для користувачів, сприяє їх позитивному настрою та забезпечує зручність використання додатку. Крім того, зелений і білий кольори добре контрастують між собою, що забезпечує читабельність тексту та виразність графічних елементів, роблячи інтерфейс більш зручним і інтуїтивно зрозумілим.

2.7 Висновки

Другий розділ містить аналіз вибору патерна архітектури, де розглядаються різні підходи до структурування програмного забезпечення та обґрунтовується вибір конкретного патерна для даного додатку. Виносяться на розгляд переваги обраного патерна та пояснюється, як саме він допоможе у вирішенні конкретних завдань додатку.

Після аналізу архітектурного патерна розробляється структурна карта додатку та описується створення бази даних для додатку. Це включає визначення сутностей та їх атрибутів, а також взаємозв'язки між ними.

Останнім етапом у цьому розділі є опис основного алгоритму роботи мобільного додатку зі структурною схемою переходів та обґрунтування кольорової гами.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору мови програмування

Розробляючи програмні засоби для мобільного застосунку, яка використовує інструменти фільтрації даних, важливо ретельно обирати технології, що використовуються. Мова Java об'єктно-орієнтована, існує вже понад 25 років і завоювала велику довіру серед розробників. Не один рік в останньому десятилітті ця мова визнавалася першою за популярністю серед усіх мов програмування. Обираючи Java для створення мобільного додатку, ми враховуємо різноманітні аспекти, які визначають його успіх та конкурентоспроможність.

По-перше, Java надає можливість розробки крос-платформених додатків, що дозволяє їх запускати на різних пристроях, що розширює аудиторію користувачів та сприяє швидшому поширенню додатку.

По-друге, велика та розгалужена екосистема Java забезпечує доступ до безлічі рішень, бібліотек та фреймворків, що полегшують розробку та роблять її більш ефективною. Крім того, висока продуктивність Java, яка виявляється у швидкості реалізації та оптимізації роботи додатку, робить її привабливим вибором для великих та складних проєктів. Надійність та безпека, що характеризують Java, є ключовими факторами, особливо у сфері мобільних додатків, які обробляють конфіденційну інформацію користувачів.

Нарешті, досвід розробки на Java для багатьох фахівців є добре знайомим, що зменшує час вивчення та сприяє швидшому запуску проєкту. Таким чином, обрання Java виявляється стратегічно обґрунтованим рішенням, що дозволяє забезпечити успішну та ефективну розробку мобільного додатку, готового конкурувати на ринку та задовольняти потреби користувачів [18].

Починаючи з JDK 8 в Java з’явився новий API – Stream API. Його завдання – спростити роботу з наборами даних, зокрема спростити операції фільтрації, сортування та інші маніпуляції з даними. Вся основна функціональність API зосереджена в пакеті java.util.stream.. Завдяки стриму більше не потрібно писати стереотипний код щоразу, коли доводиться щось робити з даними: сортувати,

фільтрувати, перетворювати. Розробники менше думають про стандартну реалізацію і більше часу приділяють складнішим речам[19].

Поєднання цих технологій дозволяє розробити мобільний застосунок, який забезпечує швидкий та надійний спосіб сортування та фальтрації даних.

Отже, вибір технологій для розробки програмних засобів для мобільного застосунку, що використовує інструменти фільтрації даних в Java, має вирішальне значення для створення надійного та швидкого моделі. Поєднання цих технологій надає необхідні інструменти та фреймворки для розробки ефективної, надійної та простої в обслуговуванні мобільного застосунку.

3.2 Вибір середовища розробки

Android Studio (рис.2.2) – офіційне середовище розробки для Android-додатків, що надає безліч інструментів і функцій[20]. Також Android Studio підтримує кілька мов програмування, включаючи C/C++ і Java, Kotlin, має вбудований емулятор і велику бібліотеку з усілякими шаблонами та компонентами, які суттєво спрощують та прискорюють процес розробки. В ній можна створювати додатки для останньої версії Android , а створений додаток можна відразу ж перевірити на наявність помилок, протестувати різними інструментами всі елементи програми та заздалегідь виявити всі можливі помилки в роботі. Якщо ви робите тільки перші кроки у непростій справі розробки мобільних додатків на Android, то ви можете завантажити досить зручний посібник із використання середовища з офіційного сайту. Завдяки вбудованому емулятору можна проводити тести продуктивності та коректності роботи програм, що розробляються на різних системах, і в разі необхідності проводити оптимізацію. З 2014 року Android Studio є офіційним середовищем розробки продуктів під Android та постійно оновлюється та покращується, забезпечуючи розробникам актуальні інструменти та можливості.

.

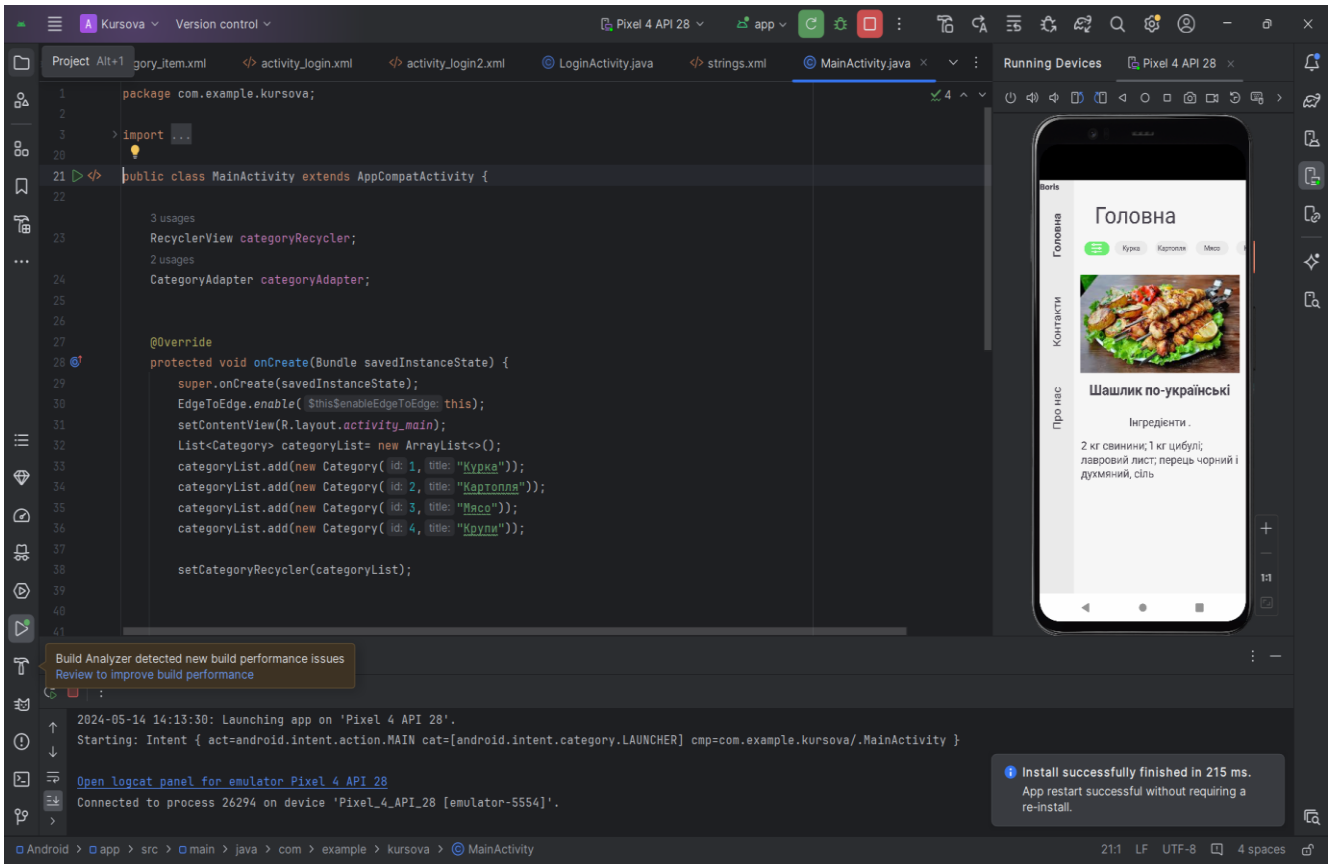


Рисунок 3.1 – Програмний інтерфейс Android Studio

Отже, для реалізації мобільного додатку «Pro100 Cooked» було обрано інтегроване середовище розробки Android Studio. Це середовище забезпечує всі необхідні інструменти для створення, тестування та налагодження Android-додатків, що робить його оптимальним вибором для розробки цього проекту.

3.3 Програмна реалізація

Існує інтерфейс IRecipeDataSource, який призначений для взаємодії з рецептами. Цей інтерфейс реалізується єдиним класом під назвою RecipeDataSource. У цьому класі міститься весь необхідний функціонал для роботи з рецептами. Давайте детально розглянемо, які можливості та методи надає цей клас, і як вони використовуються для управління рецептами та їх доступністю для звичайних користувачів і модераторів.

Таблиця 3.1 – Опис функції класу RecipesDataSource:

Назва функції	**Опис функції**
Observable<List<RecipeDto>> getRecipe (String header, String ingridients ,int cookingtime, String difficulty, String CuisinePreference)	Отримати список доступних рецептів за вибраними параметрами. Якщо параметри не будуть передані серверу, отримати всі доступні авто
Observable<Void> acceptRecipe (String header, int id)	Підтвердження рецепту модератором для того, щоб воно попало в список всіх рецептів.
Observable<Void> removeRecipe (String header, int id)	Видалення рецепту модератором для того, щоб воно не попало в список доступних рецептів звичайним користувачам. Доступно тільки модератору.
Observable<RecipeDto> addRecipe (String header, CreateRecipeDto createRecipeDto)	Додати рецепт для перевірки його модератором сервісу. Доступно всім користувачам.
Observable<List<RecipeDto>> getPendings (String header)	Отримати список оголошень, які необхідно переглянути модераторові для підтвердження. Доступно тільки модератору.
Observable<Void> uploadImage(String header, int id, String filePath)	Завантажити фотографію рецепту.

Розглянемо моделі даних, присутні в додатку:

Таблиця 3.2 – Моделі даних програмного додатку:

Назва моделі	**Призначення моделі**
BaseEntity	Базова модель
AccountRole	Роль користувача. ADMIN – модератор, USER – звичайний користувач
MessageDto	Модель обміну провідомленнями для виводу повідомлень з серверу на екран
ErrorDto	Модель помилки, яка використовуєьться при виникненні будь- якої помилки на сервері при роботі з ним
RecipeDto	Модель рецепту.
LoginResponseDto	Модель, яка отримується при успішному вході користувача в систему.
EmailDto	Модель передачі email користувача на сервер
CreateAccountResultDto	Модель, яка використовується як результат створення облікового запису користувача
CreateAccountDto	Модель, яка використовується для створення облікового запису користувача
AccountDto	Модель облікового запису користувача

Для кожного окремого розділу бізнес-логіки додатку створено свій клас-презентер. Розглянемо ці класи окремо. Цей клас створено для роботи з сервером для додання власного оголошення для подання рецепту.

Таблиця 3.3 – Опис функцій класу AddRecipePresenter:

Назва функції	**Опис функції**
void addRecipe(String header CreateRecipeDto createRecipeDto)	Додати оголошення власного рецепту
void uploadPhoto (String header, int RecipeId, String filePath)	Завантажити фото

Цей клас створено для отримання списку доступних рецептів.
Використовується як для звичайних користувачів, так і для модераторів.

Таблиця 3.4 – Опис функцій класу RecipeListPresenter:

Назва функції	**Опис функції**
void getRecipes(String header, String ingridients , int cookingtime, String difficulty, String CuisinePreference)	Отримати список рецептів за заданими параметрами. Якщо параметри не будуть передані, повернеться список з усіма доступними користувачеві автомобілями
void getMyRecipes(String header, String userLogin)	Отримати список рецептів, створених поточним користувачем
Void getPendings(Stringheader)	Отримати список оголошень, які потребують перевірки модератором. Доступно лише модераторові.

Даний клас створено для обробки бізнес-логіки, необхідної у випадку, якщо користувач втратив свій пароль для його відновлення.

Таблиця 3.5 – Опис функцій класу ForgotPasswordPresenter:

Назва функції	**Опис функції**
void onForgotPassword(EmailDto emailDto)	Отримати лист на почту для відновлення паролю.

Клас вище створено для обробки бізнес-логіки, необхідної для підтвердження оголошення модератором перед тим, як воно потрапить у список актуальних оголошень, або видалення оголошення модератором або користувачем.

Таблиця 3.6 – Опис функцій класу FullRecipeInfoPresenter:

Назва функції	**Опис функції**
void acceptRecipe(String header,int RecipeId)	Підтвердити оголошення для того, щоб воно потрапило у актуальні оголошення і стало доступним всім користувачам. Цей функціонал доступний лише модератору.
void removeRecipe(String header,int RecipeId)	Видалити оголошення. Цей метод доступний модератору для всіх оголошеньі звичайному користувачеві для оголошень, створених ним самим.

Цей клас створено для обробки бізнес-логіки, необхідної для входу користувача в обліковий запис, а також для повторного запиту на активацію облікового запису, якщо він не був активований впродовж 24 годин після реєстрації

.

Таблиця 3.7 – Опис функцій класу LoginPresenter:

Назва функції	**Опис функції**
void resendValidation(EmailDto emailDto)	Надіслати лист для повторної активації облікового запису.
void login(LoginRequestDto loginRequestDto)	Увійти в систему

Цей клас створено для обробки бізнес-логіки, яка необхідна для створення облікового запису користувача. Він відповідає за всі аспекти цього процесу, включаючи валідацію введених даних, перевірку унікальності користувацького імені та електронної пошти, хешування паролів для забезпечення безпеки, а також збереження інформації про користувача в базі даних.

Таблиця 3.8 – Опис функцій класу SignUpPresenter:

Назва функції	**Опис функції**
void signUp(CreateAccountDto createAccountDto)	Створити обліковий запис користувача.

В програмного додатку до цієї роботи доданий базовий клас View, який наслідується кожним класом, який відповідає за відображення графічного інтерфейсу користувача. Розглянемо, який функціонал закладений в базовий клас BaseView:

Таблиця 3.9 – Опис функцій інтерфейсу BaseView:

Назва функції	**Опис функції**
void showError(String error)	Відобразити помилку
Context getContext()	Отримати контекст додатку для отримання його ресурсів у місці, це використовується поточний об’єкт View

Цей клас був створений для відображення інтерфейсу, який відображається користувачу коли успішно доданні рецепти .

Таблиця 3.10 – Опис функцій інтерфейсу AddRecipeView:

Назва функції	**Опис функції**
void onAdded(RecipeDto RecipeDto)	Відобразити графічний інтерфейс при успішному додаванні оголошення в базу даних сервера.

Цей клас був створений для відображення інтерфейсу, який відображається користувачу при перегляді всіх доступних йому рецептів.

Таблиця 3.11 – Опис функцій інтерфейсу RecipeListView:

Назва функції	**Опис функції**
void showRecipeList(List<RecipeDto> Recipes)	Відобразити графічний інтерфейс доступних користувачеві рецептів

Цей клас був створений для відображення інтерфейсу, який відображається користувачу у випадку відновлення паролю.

Таблиця 3.12 – Опис функцій інтерфейсу ForgotPasswordView:

Назва функції	**Опис функції**
void showSuccessDialog(MessageDto messageDto)	Показати діалог про успішне відправлення листа для відновлення паролю на електронну пошту користувача.

Цей клас був створений для відображення інтерфейсу, який відображається користувачу на екрані з повною інформацією про рецепт.

Таблиця 3.13 – Опис функцій інтерфейсу FullRecipeInfoView:

Назва функції	**Опис функції**
void onRemoved()	Обробити подію про видалення рецепту у графічному інтерфейсі користувача.
void onAccepted()	Обробити подію про підтвердження рецепту модератором у графічному інтерфейсі користувача.

Цей клас був створений для відображення інтерфейсу, який відображається користувачу на екрані для входу в обліковий запис користувача.

Таблиця 3.14 – Опис функцій інтерфейсу LogInView:

Назва функції	**Опис функції**
void processLogginIn(LoginResponseDto loginresponse)	Показати користувачу екран успішного входу в систему.

Цей клас був створений для відображення інтерфейсу, який відображається користувачу на екрані для створення облікового запису користувача.

3.5 Висновки

У цьому розділі було обґрунтовано вибір мови програмування та технологій для розробки мобільного застосунку. Проаналізувавши потреби програмного продукту було обрано мову програмування Java. Для зручності розробки графічного інтерфейсу було обрано Android Studio.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування мобільної системи на Android – це процес перевірки функціональності, безпеки та коректності роботи програм на мобільній платформі від Google. Одним з найважливіших критеріїв успішного тестування є перевірка сумісності програми з різними версіями операційної системи.

Перш ніж почати тестування, необхідно встановити тестову версію програми на мобільний пристрій з операційною системою Android, яка буде використовуватися під час тестування. Це може бути здійснене через різні механізми, такі як встановлення з Google Play або надсилання тестової версії через електронну пошту або месенджери.

Після встановлення тестової версії програми проводяться функціональні тести для перевірки правильності роботи програми. Це включає тести на введення та виведення даних, перевірку реакції програми на негативні вхідні дані, а також перевірку відповідності бізнес-логіці програми. Наприклад, якщо програма призначена для введення користувача, тести можуть перевіряти, чи правильно обробляються дані, введені користувачем, та чи правильно відображається реакція програми на ці дані.

Після функціональних тестів проводяться тести на безпеку програми. Це може включати перевірку на проникнення, тести на збереження конфіденційної інформації та тести на злам програми. Для цього використовуються спеціальні програмні засоби, які дозволяють моделювати атаки на програму та виявляти її вразливості.

Після успішного проходження всіх тестів проводиться оцінка продуктивності програми. Це включає тести на швидкість роботи програми, тести на використання ресурсів мобільного пристрою та тести на життєвий цикл програми. Наприклад, можуть бути проведені тести на швидкість завантаження програми, швидкість виконання певних операцій та використання оперативної пам'яті.

В цій роботі основним направленням тестування буде тестування програмного запозичення .

Існує багато методик для тестування ПЗ. Для аналізу було обрано такі методики функціонального тестування:

- тестування методом білої скриньки ;
- тестування методом чорної скриньки;
- тестування методом сірої скриньки;

Тестування "чорної скриньки" ґрунтується на тому, щоб розглядати програму як чорний ящик, де тестувальник не має доступу до внутрішніх деталей коду. Вміст інструкцій та реалізація програми не важливі, тільки вхідні дані та очікувані результати. Цей метод дозволяє виявити проблеми функціональності без звернення до коду.

"Біла скринька" використовується для аналізу внутрішніх структур програми. Тестувальник має доступ до коду та використовує це, щоб створити тести, орієнтовані на структуру програми. Це дозволяє виявити проблеми, пов'язані з логікою програми та шляхами виконання коду.

"Сіра скринька" поєднує обидва підходи. Тестувальник має обмежений доступ до внутрішніх деталей програми, що дозволяє йому створювати тести, які використовують як знання про внутрішню структуру, так і вхідні та вихідні дані програми.

Кожен з цих підходів має свої переваги та недоліки і може бути використаний у відповідності з конкретними потребами проекту та доступними ресурсами для тестування. Але вирішено було обрати «чорна скринька», саме використання такого тестування дозволить визначити помилки в логіці використання користувачем додатку для подальшого їх усуннення на ранній стадії

4.2 Тестування розробленого програмного продукту

Першим кроком буде перевірка екрану реєстрації користувача шляхом спроби зареєструвати користувача без введення інформації. У разі спроби

реєстрації без обов'язкової інформації про ім'я, система повідомить про помилку та запропонує ввести дані, що є правильною поведінкою системи, оскільки інформація про ім'я є обов'язковою для подальшої роботи користувача. Результат тестування зображено на рисунку 4.1.

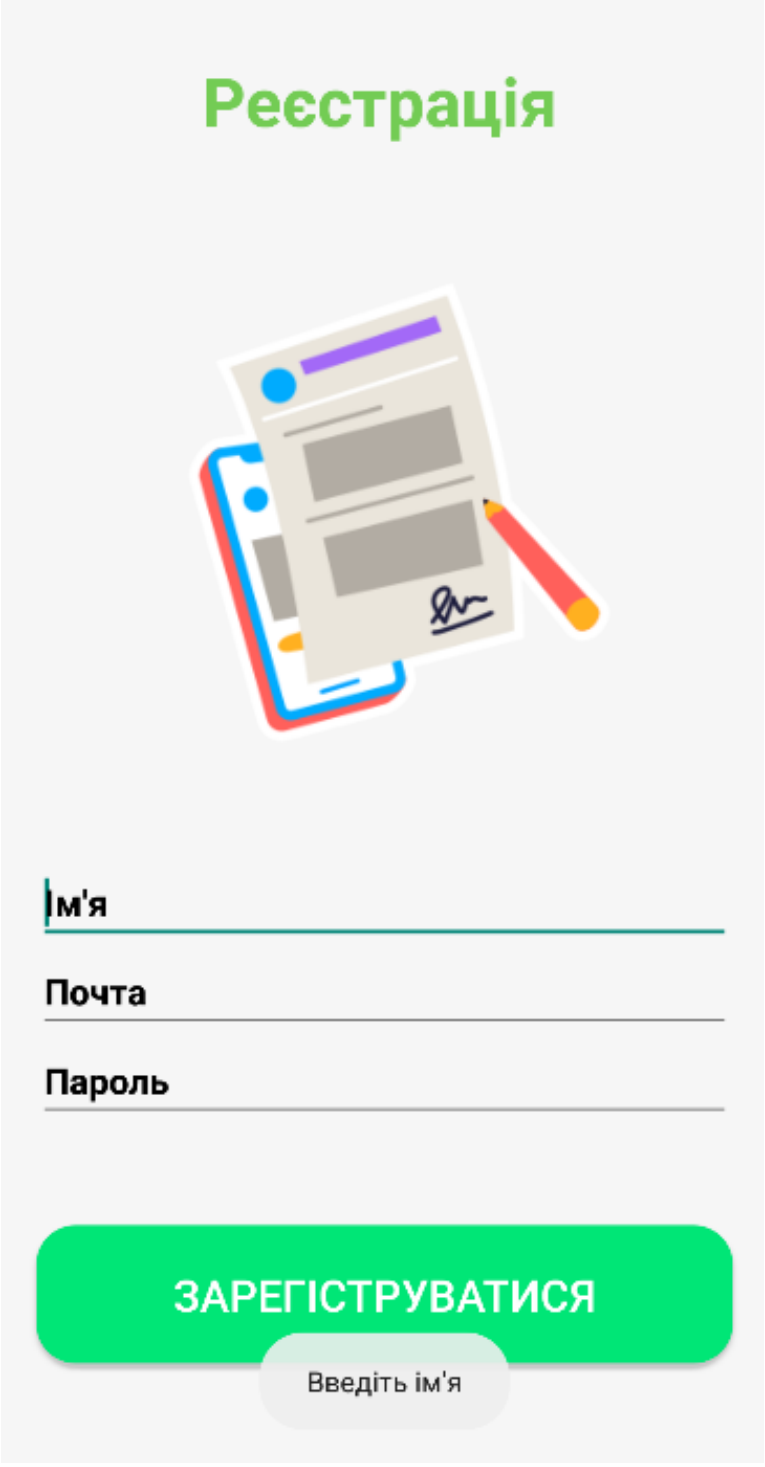


Рисунок 4.1 – Реєстрація користувача без імені

Переконавшись, що система не дозволяє увійти в застосунок користувача без необхідних полів, спробуємо ввести неправильне значення в поле електронної адреси. У результаті тестування отримаємо сповіщення про те, що такого аккаунта не існує та користувача буде переведено на сторінку реєстрації, що можна побачити на рисунку 4.2.

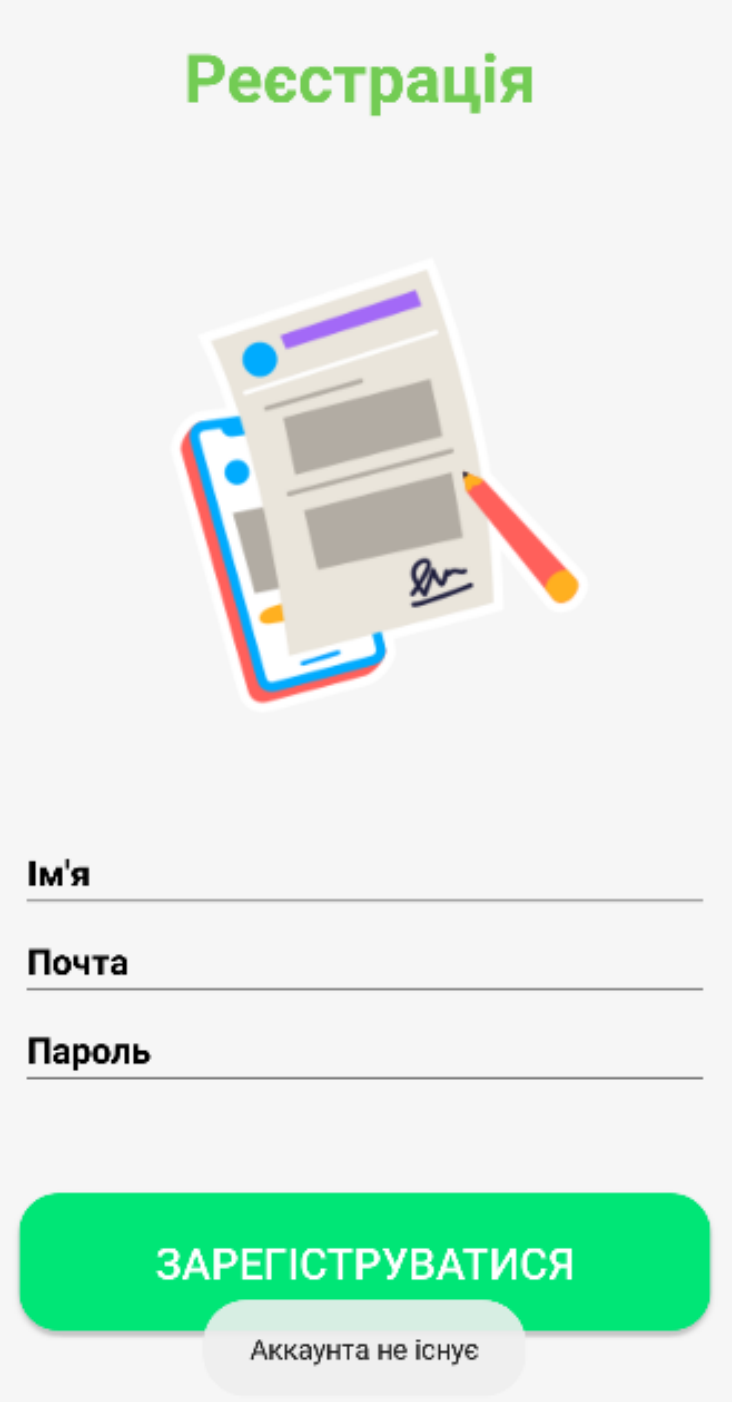


Рисунок 4.2 – Вхід користувача з неправильними даними

Перевіривши введення не дійсних даних, спробуємо заповнити всі поля вірно і перевірити чи зареєструє система користувача. У результаті перевірки система перенаправила користувача на сторінку авторизації, отже користувача було успішно зареєстровано. Результат тестування зображено на рисунку 4.3.

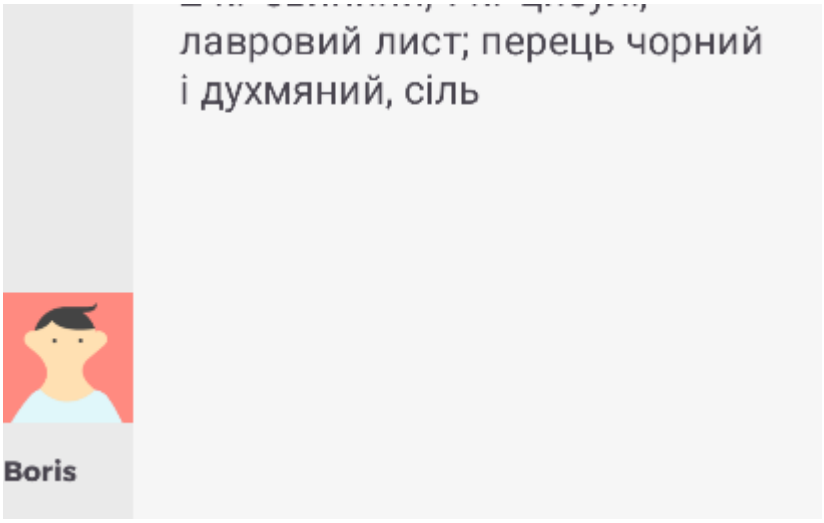


Рисунок 4.3 – Результат успішної реєстрації користувача

Важливим аспектом мобільної платформи є валідація вхідних даних поля паролю. На рисунку 4.4 повідомлення про невірну розкладку клавіатури при вводу паролю.

Рисунок 4.4 – Валідація поля паролю

4.3 Інструкція користувача

При відкритті додатку, користувача вітає екран (рис. 4.5), з логотипом, який привертає увагу своїми чіткими контурами та виразними кольорами. Логотип створений з урахуванням сучасних тенденцій та відображає концепцію та характер додатку. Під ним розташована назва застосунку, виконана в стильному шрифті, що доповнює загальний дизайн екрану та створює враження професіоналізму та сучасності. Цей екран відтворює перше враження, з яким користувач буде асоціювати свій досвід користування додатком, підкреслюючи його унікальність та привабливість.

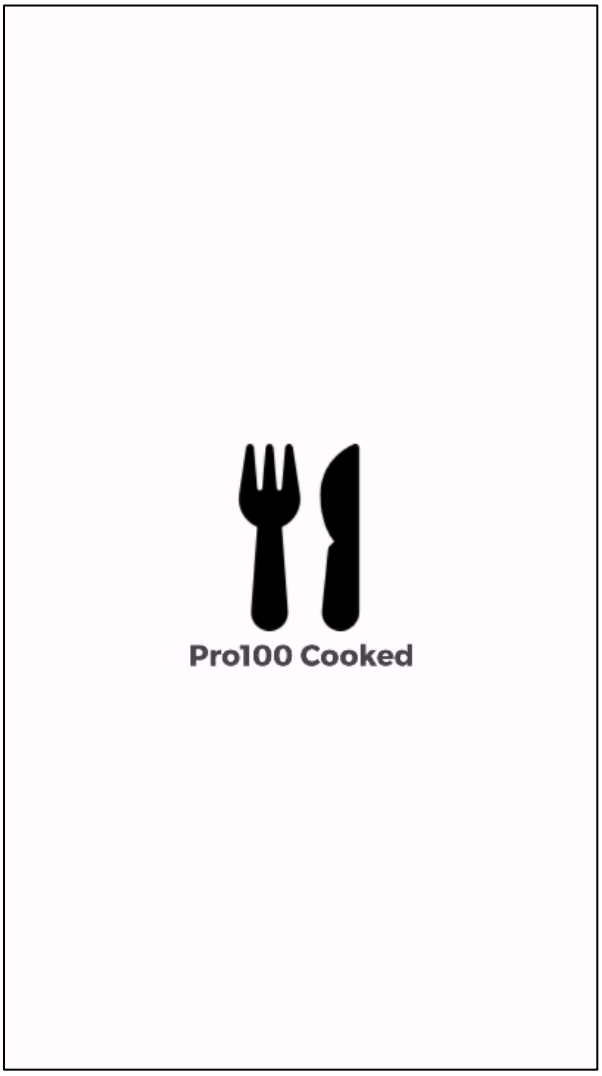


Рисунок 4.5 – Екран завантаження

На екрані реєстрації користувача (рис. 4.6) кожен користувач системи запрошується ввести свої персональні дані для створення облікового запису. Чистий та зрозумілий інтерфейс надає можливість користувачам легко та зручно вводити інформацію. Поле для введення електронної пошти та пароля розташовані у відповідних секціях екрану, а кнопка "Регістрація" чітко виділена, щоб забезпечити простоту взаємодії.

Рисунок 4.6 – Екран реєстрації користувача

Після успішної реєстрації особистого кабінету, користувач переходить на екран авторизації (рис. 4.7), де йому пропонується ввести свої аутентифікаційні дані для входу в систему додатка. Цей екран відображається з метою забезпечення безпеки та захисту особистих даних користувача. Великі та чіткі поля для введення електронної пошти та пароля забезпечують зручну інтерактивність, а кнопка "Увійти" проста та доступна для використання. Додаткові функції, такі як відновлення паролю або довідкова інформація, можуть також бути доступні на цьому екрані для полегшення процесу авторизації користувача.

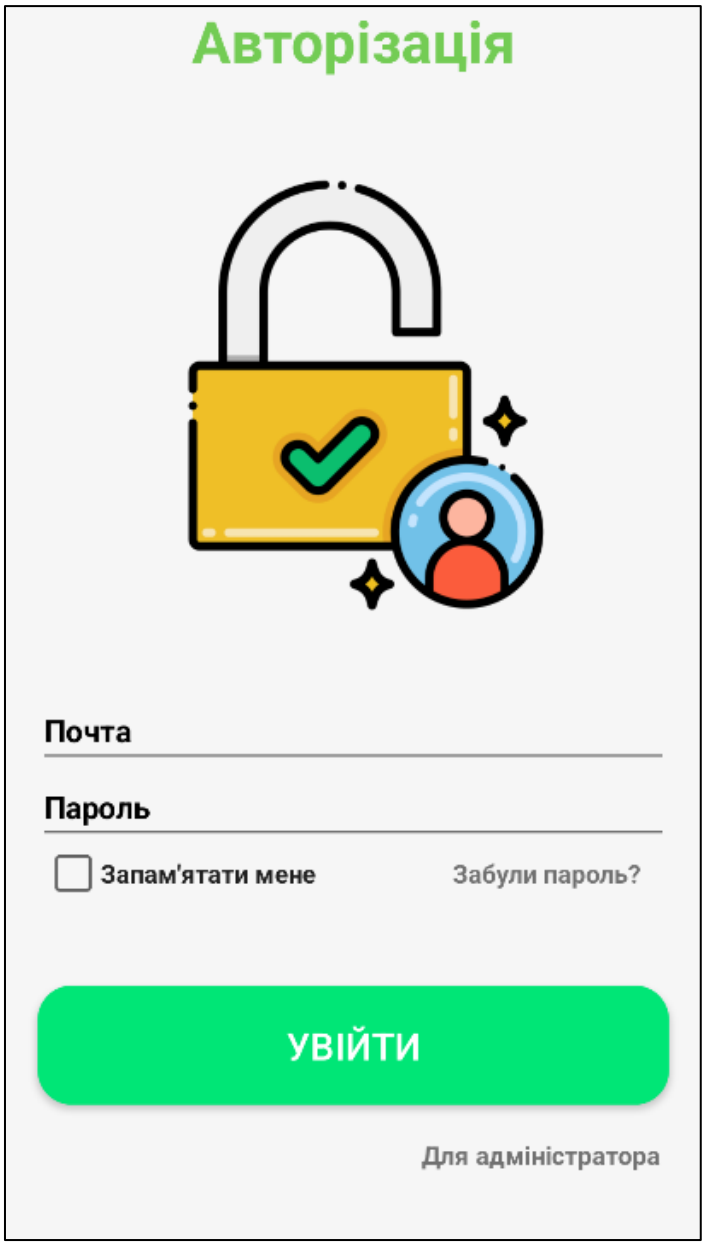


Рисунок 4.7 – Екран авторизації користувача

Після успішної авторизації користувач може продовжити користуватися застосунком. На екрані з вибором рецепту (рис. 4.8) користувач має можливість обрати різні потрібні йому інгредієнти для пошуку рецептів за ними. Цей екран відображається з метою полегшення користувачам пошуку необхідних рецептів та забезпечення їхнього комфорту та задоволення використання застосунку. Інтерактивні елементи інтерфейсу дозволяють користувачам зручно та швидко вибрати необхідні інгредієнти, після чого вони отримують список рецептів, що відповідають їхнім вимогам.

Рисунок 4.8 – Екран з вибором рецепту

Натиснувши на картинку з фільтром користувач переходить на екран з вибором типом інгредієнта (рис. 4.9) на якому зображено типи інгредієнтів при нажаті на яких є змога обрати інгредієнт.

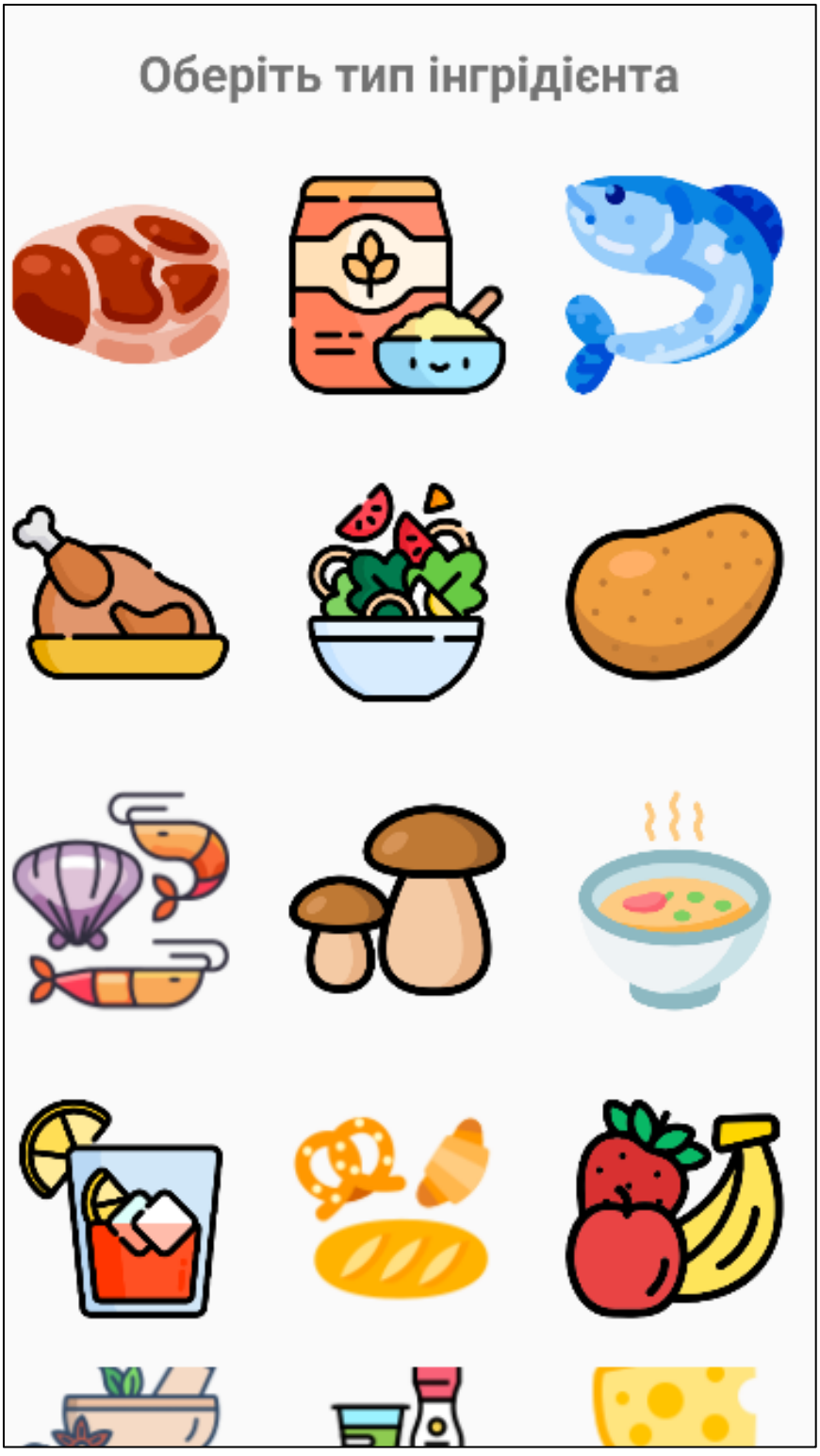


Рисунок 4.9 – Екран з консультуванням по рецепту

Натиснувши на картинку рецепту, користувач переходить на екран з більш детальною інформацією про приготування цього рецепту (рис. 4.10). На цьому екрані представлена повна інформація про склад і процес приготування страви, включаючи список інгредієнтів, інструкції та порційні розміри. Інтуїтивний та зручний дизайн дозволяє користувачам легко зорієнтуватися та виконати всі необхідні дії для приготування обраної страви.

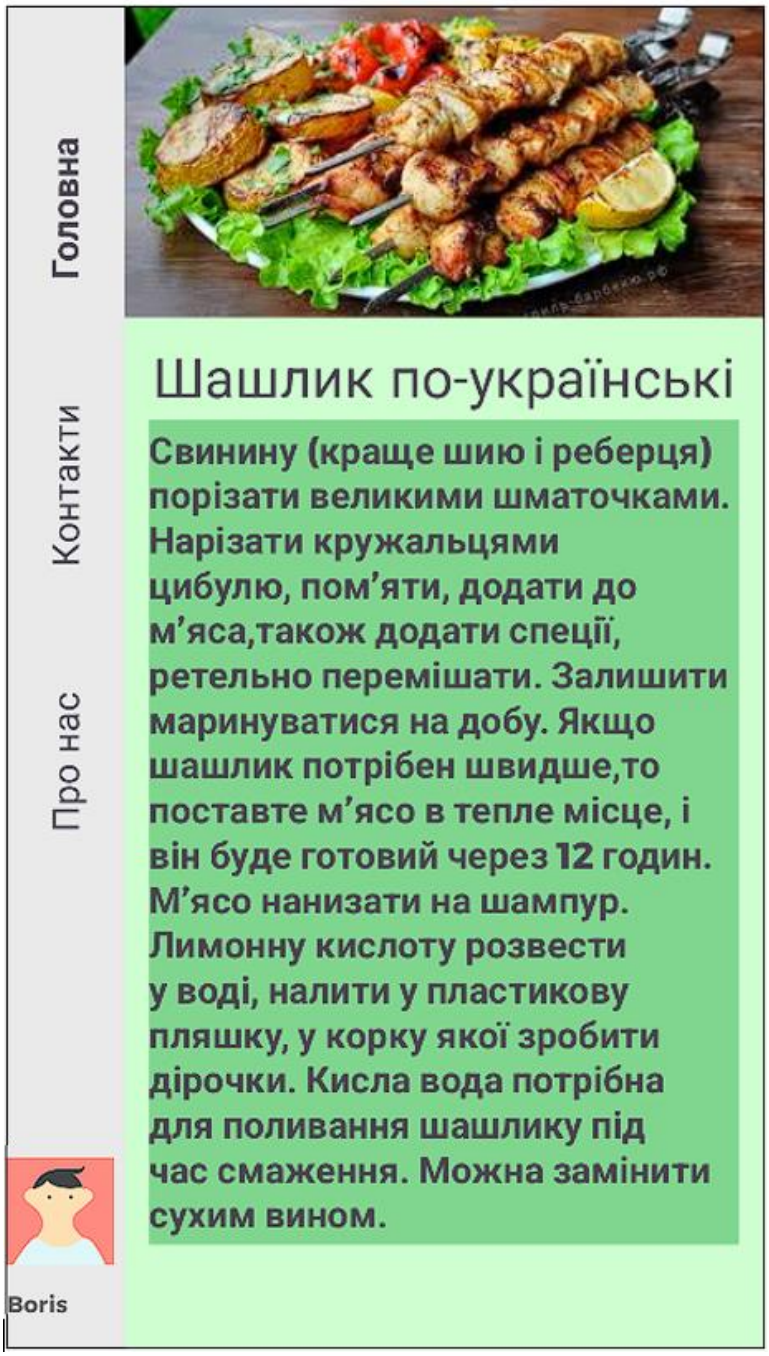


Рисунок 4.10 – Екран з консультуванням по рецепту

Рисунок 4.10 – Екран з додаванням рецепту

Ця сторінка(рис.4.10) дозволяє вам додати новий рецепт до нашої колекції. Ви можете ввести всю необхідну інформацію про рецепт, включаючи ім'я страви, список інгредієнтів, час приготування, складність, тип кухні та детальні інструкції щодо приготування.

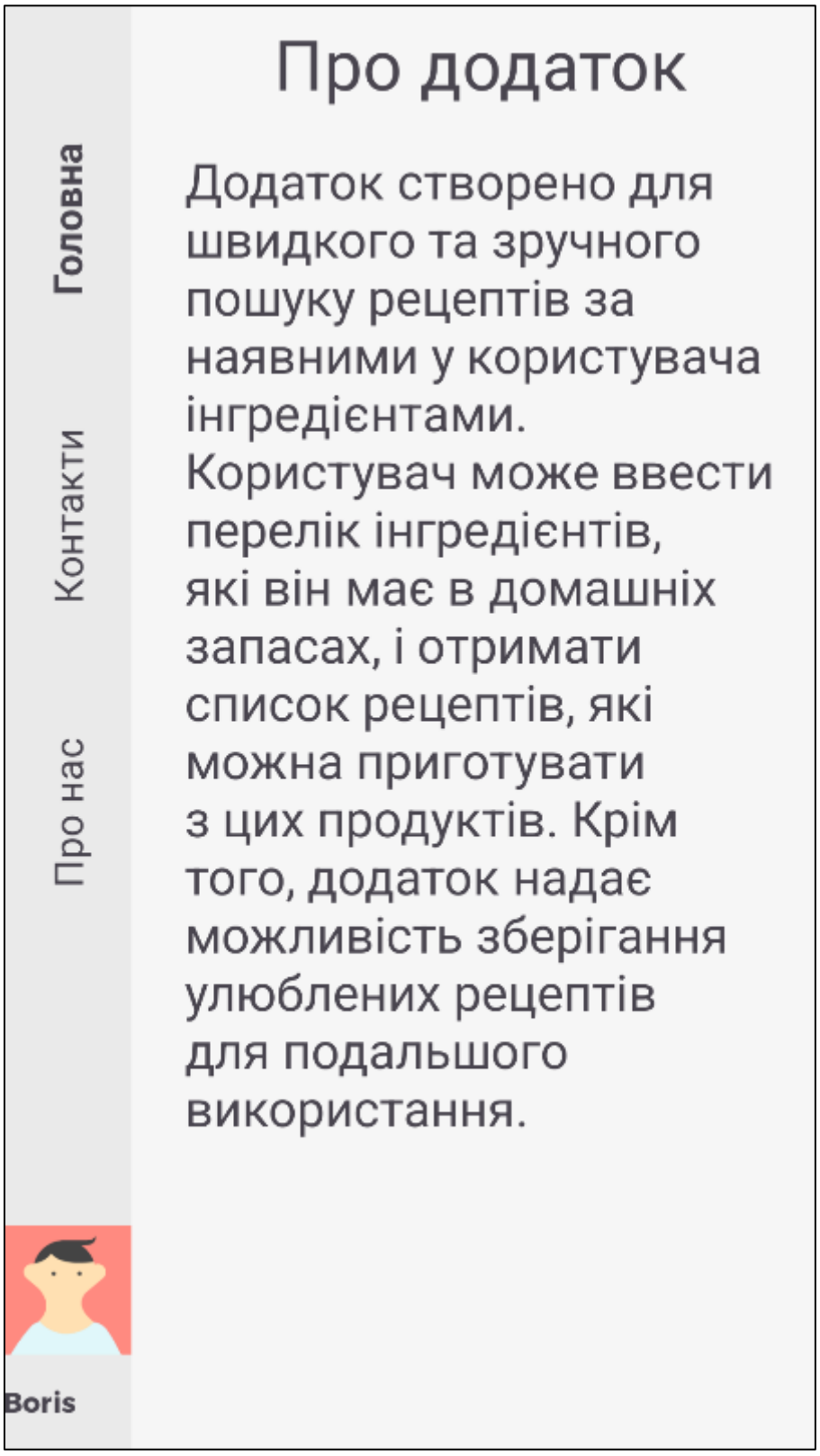


Рисунок 4.11 – Екран з інформацією про додаток

На цій сторінці(рис.4.11) представлена інформація про наш додаток та його функції.

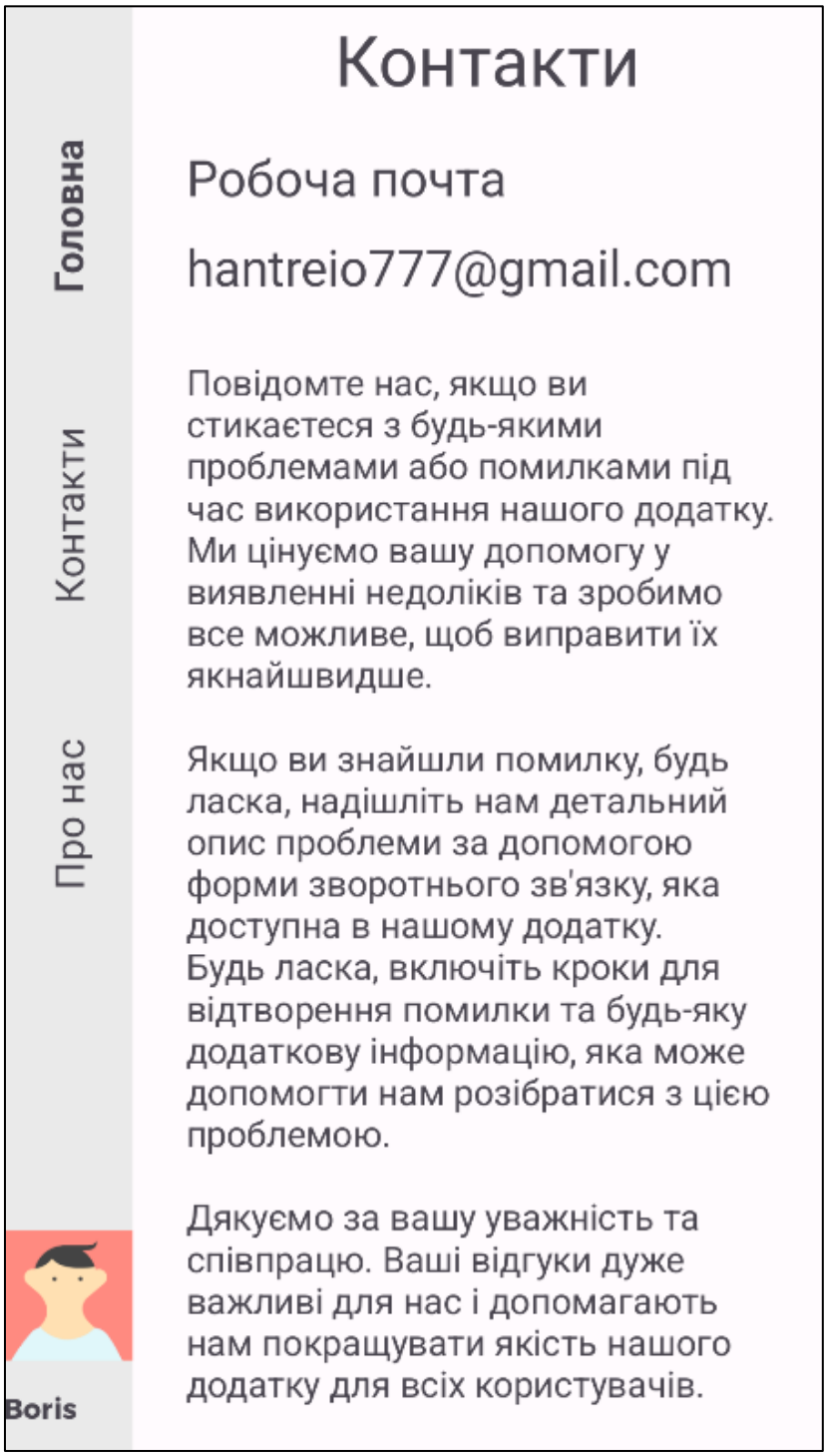


Рисунок 4.12 – Екран з контактною інформацією розробника

На цій сторінці (рис.4.12) представлена інформація для зв’язку з розробником та надає можливість користувачу звернутись у разі виникнення помилку у роботі додатку.

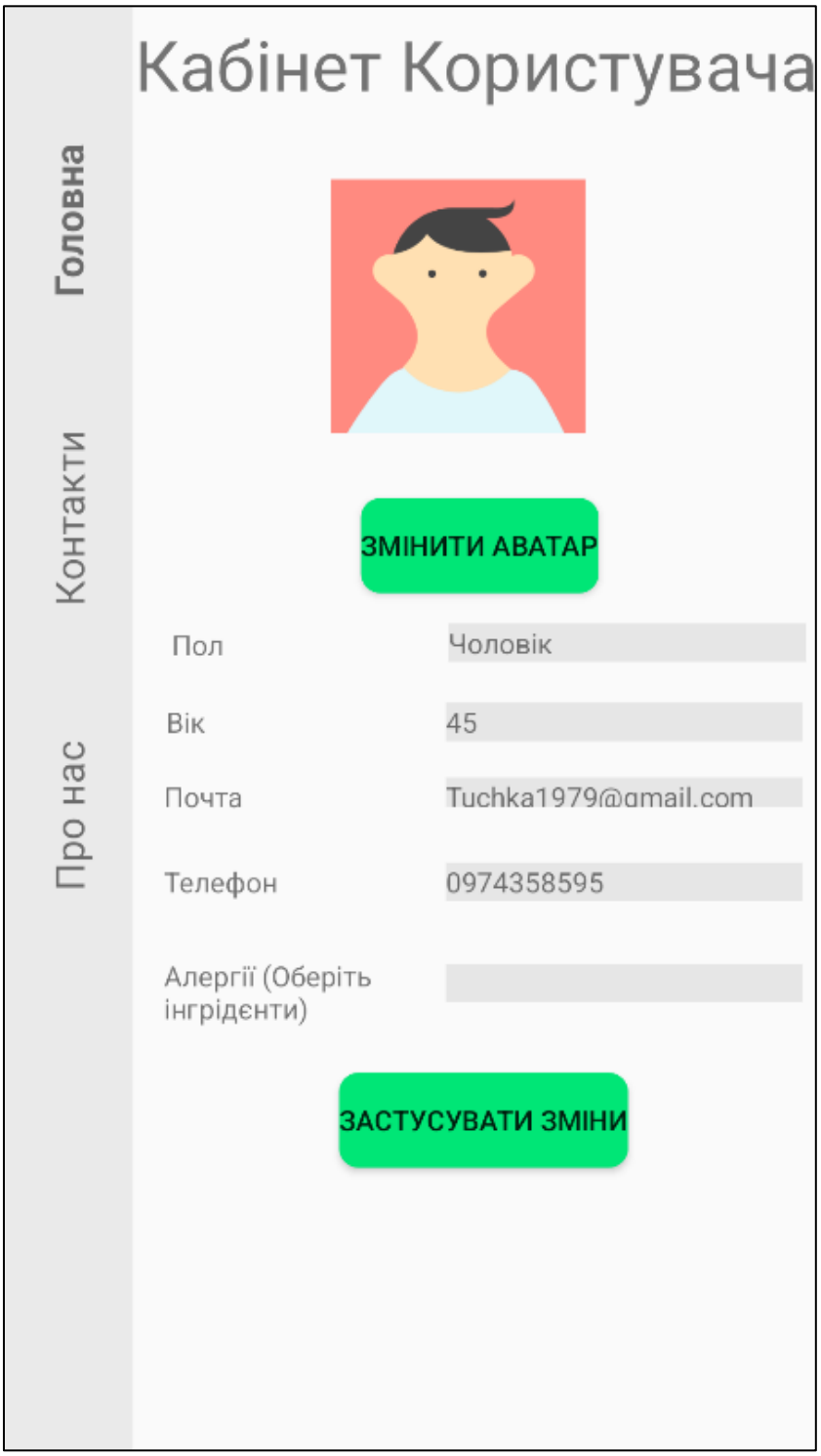


Рисунок 4.13 – Екран з додаванням рецепту

На цій сторінці (рис.4.13) користувач буде мати змогу подивитися інформацію про себе, змінити аватар та обрати інгредієнт на який у нього алергія, що запобігає потрапляння вказаних алергенів в страви

4.4 Висновки

У цьому розділі було проведено зведення кількох методик тестування з метою вибору найбільш підходящого для розроблюваного програмного додатку..

Після проведення тестування прийшли до висновку, що програмний продукт є повністю працездатним і відповідає всім поставленим вимогам згідно технічного завдання. Результати тестування показали, що додаток відповідає вимогам щодо функціональності, продуктивності та надійності.

Додатково, ми розробили інструкцію користувача, яка детально описує всі можливості та функціонал додатку, а також надає користувачам необхідні вказівки щодо його використання. Що забезпечило користувачам чітке розуміння того, як використовувати наш додаток та отримати максимальну користь від його використання.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено мобільний додаток «Pro100 Cooked». Під час роботи було ознайомлена зі станом проблеми, порівняно розроблюваний програмний додаток з існуючими аналогами та описано алгоритми для вирішення поставлених проблем засобів мобільного застосунку - персонального консультанта з приготування їжі з використанням фільтрації даних.

Було обрано графічний інтерфейс. Описані графічні інтерфейси програмного додатку. Розроблено блок-схеми алгоритмів програмного додатку. Розроблено і описано роботу штучного інтелекту.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Java та бібліотеки Stream API.

Було розроблено схеми загального алгоритму роботи мобільного застосунку. А також розроблено модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка мобільного застосунку-персонального консультанта з приготування їжі на платформі Android з використанням технологій JAVA. Щербацький Б.І., Катєльніков Д.І. (Вінницький національний технічний університет) // Матеріали Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Стан, досягнення і перспективи інформаційних систем і технологій» 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ontu.edu.ua/download/konfi/2024/Conference_abstract-IT-2024.pdf
2. Аналітика мобільних додатків та MMP [Електронний ресурс] – Режим доступу до ресурсу: https://netpeak.net/uk/blog/povniy-posibnik-z-mobil-noi-analitiki-ta-mmp/
3. Wikipedia Cookpad [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Cookpad#:~:text=Cookpad%20Inc.,%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D1%96%20%D1%82%D0%B0%D0%BA%D0%BE%D0%B6%20%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0%D0%BC%D0%B8%20%D1%81%D0%B0%D0%B9%D1%82%D1%83%2C%20%D1%80%D0%B5%D1%86%D0%B5%D0%BF%D1%82%D0%B8.
4. Cookpad homepage [Електронний ресурс] – Режим доступу до ресурсу: https://cookpad.com/ua/homepage.
5. Прості рецепти [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.dilstudio.easyrecipes&hl=ua
6. Supercook [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/search?q=supercook&c=apps&hl=ua
7. My Recipe Box: Мої рецепти [Електронний ресурс] – Режим доступу до ресурсу:https://play.google.com/store/apps/details?id=fr.recettetek&hl=ua&gl=US

8. Фільтрування, перебір елементів та відображення [Електронний ресурс]. Режим доступу до ресурсу: - https://abitap.com/10-3-filtruvannya-perebir-elementiv-ta-vidobrazhennya/
9. Введення в лямбда-вирази [Електронний ресурс]. Режим доступу: - https://abitap.com/8-1-vvedennya-v-lyambda-vyrazy/
10. Рекурсія в програмуванні та як її застосовувати [Електронний ресурс]. Режим доступу: - https://foxminded.ua/rekursiia-v-prohramuvanni/
11. Факультет прикладної математики [Електронний ресурс] – Режим доступу до ресурсу: http://fpm.dnu.dp.ua/2019/12/04/lambda/
12. Про архітектуру додатків [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/arkhitektura-zastosunku/#:~:text=MVC%20(Model%2DView%2DController,%D0%B2%D0%B7%D0%B0%D1%94%D0%BC%D0%BE%D0%B4%D1%96%D1%94%D1%8E%20%D0%BC%D1%96%D0%B6%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%BB%D1%8E%20%D1%82%D0%B0%20%D0%BF%D0%BE%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC).
13. Розділяй та володарюй: що таке патерни MVC і MVP, та як їх використовувати [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/uk/blogs/shho-take-mvc-ta-mvp-patterni/
14. Структурні карти Константайна [Електронний ресурс]. – Режим доступу до ресурсу: https://studfile.net/preview/9445425/page:36/
15. Навіщо при створенні баз даних використовують ключові атрибути [Електронний ресурс]. – Режим доступу до ресурсу: https://reporter.zp.ua/navishho-pry-stvorenni-baz-danyh-vykorystovuyut-klyuchovi-atrybuty.html
16. Нормалізація відношень при проектуванні БД [Електронний ресурс]. – Режим доступу до ресурсу: https://rdb.dp.ua/uk/chapter_03#:~:text=4%D0%9D%D0%A4%20%D1%83%D1%81%D1%83%D0%B2%D0%B0%D1%94%20%D0%BD%D0%B5%D1%82%D1%8

0%D0%B8%D0%B2%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D1%83%20%D0%B1%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%B7%D0%BD%D0%B0%D1%87%D0%BD%D1%83%20%D0%B7%D0%B0%D0%BB%D0%B5%D0%B6%D0%BD%D1%96%D1%81%D1%82%D1%8C,%D0%B7%20%D0%BC%D0%B5%D1%82%D0%BE%D1%8E%20%D0%B7%D0%BC%D0%B5%D0%BD%D1%88%D0%B5%D0%BD%D0%BD%D1%8F%20%D0%BD%D0%B0%D0%B4%D0%BC%D1%96%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%96%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.

17. Навіщо колірні схеми? [Електронний ресурс]. – Режим доступу до ресурсу: https://naciya.home.cx.ua/navishho-kolirni-skhemi/

18. Go it Що таке Java і де вона використовується? [Електронний ресурс] – Режим доступу до ресурсу: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/

19. Вступ в Stream API. [Електронний ресурс] – Режим доступу до ресурсу: https://abitap.com/10-1-vstup-v-stream-api/

20. QualityAsuranceGroup Android Studio: переваги та особливості [Електронний ресурс] – Режим доступу до ресурсу: https://qagroup.com.ua/publications/android-studio-perevagy-ta-osoblyvosti/

ДОДАТКИ

Додаток А – Лістинг програми

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технологій Java» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Катєльніков Д.І
«12» березня 2024 р.
Виконав:
_____ студент гр. 1ПІ-20б Б. І. Щербацький
«12» березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технологій Java».

Галузь застосування – швидкий пошук рецептів за наявними інгредієнтами і використання в харчові промисловості.

2. Підстава для розробки

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою є скорочення часу для отримання рецептів смачної та поживної їжі за рахунок реалізації сортування та фільтрації даних при розробці мобільного програмного додатку.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Факультет прикладної математики [Електронний ресурс] – Режим доступу до ресурсу: http://fpm.dnu.dp.ua/2019/12/04/lambda/
2. Про Програмування. Категорія: Java [Електронний ресурс]. Режим доступу: - https://abitap.com/category/java/

5. Технічні вимоги

Вихідні дані до роботи: інструмент для розробки мобільного додатку на платформі Android, Android Studio; мова розробки Java, база даних Firebase.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

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

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ п\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз та обґрунтування вибору методу розробки та постановка задачі дослідження	13.03.24 –18.03.24
2	Розробка архітектури та алгоритмів програмного продукту	18.03.24 –26.03.24
3	Розробка програмного продукту	27.03.24 –21.04.24
4	Тестування програми	22.04.24 –16.05.24
5	Оформлення матеріалів до захисту БДР	17.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технологій Java

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Катєльніков Д.І.

Оригінальність	89,2%
Схожість	10,8%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Щербацький Б.І.

Керівник роботи Катєльніков Д.І.

Додаток В – Лістинг програми

//MainActivity.java

```
package com.example.mybpd;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.QuickContactBadge;
import android.widget.Toast;
import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import com.example.mybpd.Model.Users;
import com.example.mybpd.Prevalent.Prevalent;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import io.paperdb.Paper;
public class MainActivity extends AppCompatActivity {
    private Button joinButton;
    private Button loginButton;
    private ProgressDialog loadingBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
```

```
setContentView(R.layout.activity_main);
joinButton=(Button) findViewById(R.id.main_join_btn);
loginButton=(Button) findViewById(R.id.main_login_btn);
loadingBar = new ProgressDialog(this);
Paper.init(this);
loginButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      Intent loginIntent = new Intent(MainActivity.this, LoginActivity.class);
      startActivity(loginIntent);
   }
});
joinButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      Intent registerIntent = new Intent(MainActivity.this, RegisterActivity.class);
      startActivity(registerIntent);
   }
});
String UserMailKey = Paper.book().read(Prevalent.UserMailKey);
String UserPasswordKey = Paper.book().read(Prevalent.UserPasswordKey);
if(UserMailKey!=""&& UserPasswordKey!="")
{
   if(!TextUtils.isEmpty(UserMailKey) && !TextUtils.isEmpty(UserPasswordKey) )
   {
      ValidateUser(UserMailKey,UserPasswordKey);
      loadingBar.setTitle("Вхід...");
      loadingBar.setMessage("Будь ласка, зачекайте...");
      loadingBar.setCanceledOnTouchOutside(false);
      loadingBar.show();
   }
}
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
```

```
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}
private void ValidateUser(final String mail,  final String password)
{
    final DatabaseReference RootRef;
    RootRef = FirebaseDatabase.getInstance().getReference();
    RootRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (snapshot.child("Users").child(mail).exists())
            {
                Users usersData = snapshot.child("Users").child(mail).getValue(Users.class);
                if (usersData.getMail().equals(mail))
                {
                    if (usersData.getPassword().equals(password))
                    {
                        loadingBar.dismiss();
                        Toast.makeText(MainActivity.this, "Вхід...", Toast.LENGTH_SHORT).show();
                        Intent homeIntent = new Intent (MainActivity.this,HomeActivity.class);
                        startActivity(homeIntent);
                    }
                    else
                    {
                        loadingBar.dismiss();
                    }
                }
            }
            else
            {
                loadingBar.dismiss();
```

```
                Toast.makeText(MainActivity.this, "Аккаунта не існує", Toast.LENGTH_SHORT).show();
                Intent registerIntent = new Intent (MainActivity.this,RegisterActivity.class);
                startActivity(registerIntent);
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
        }
    });
  }
 }
}
```

//LoginActivity.java

```
package com.example.mybpd;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import com.example.mybpd.Model.Users;
import com.example.mybpd.Prevalent.Prevalent;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
```

```
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.rey.material.widget.CheckBox;
import io.paperdb.Paper;
public class LoginActivity extends AppCompatActivity {
    private Button loginBtn;
    private EditText mailInput;
    private EditText passwordInput;
    private ProgressDialog loadingBar;
    private String parentDbName= "Users";
    private CheckBox checkBoxRememberMe;
    private TextView AdminLink;
    private TextView NotAdminLink;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_login);
        loginBtn=(Button)findViewById(R.id.login_btn);
        mailInput=(EditText) findViewById(R.id.login_mail_input);
        passwordInput=(EditText) findViewById(R.id.login_password_input);
        loadingBar = new ProgressDialog(this);
        checkBoxRememberMe = (CheckBox) findViewById(R.id.login_check_box);
        Paper.init(this);
        AdminLink = (TextView) findViewById(R.id.admin_panel_link);
        NotAdminLink = (TextView) findViewById(R.id.not_admin_panel_link);
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                loginUser();
            }
        });
```

```
    AdminLink.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            AdminLink.setVisibility(View.INVISIBLE);
            NotAdminLink.setVisibility(View.VISIBLE);
            loginBtn.setText("Вхід для Адміна");
            parentDbName= "Admins";
        }
    });
    NotAdminLink.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            AdminLink.setVisibility(View.VISIBLE);
            NotAdminLink.setVisibility(View.INVISIBLE);
            loginBtn.setText("Вхід");
            parentDbName= "Users";
        }
    });
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}
private void loginUser()
{
    String mail = mailInput.getText().toString();
    String password = passwordInput.getText().toString();
    if (TextUtils.isEmpty(mail))
    {
        Toast.makeText(this, "Введіть почту", Toast.LENGTH_SHORT).show();
    }
    else if (TextUtils.isEmpty(password))
```

```
    {
        Toast.makeText(this, "Введіть пароль", Toast.LENGTH_SHORT).show();
    }
    else
    {
        loadingBar.setTitle("Вхід...");
        loadingBar.setMessage("Будь ласка, зачекайте...");
        loadingBar.setCanceledOnTouchOutside(false);
        loadingBar.show();
        ValidateUser(mail,password);
    }
}
private void ValidateUser(String mail, String password)
{
    if(checkBoxRememberMe.isChecked())
    {
        Paper.book().write(Prevalent.UserMailKey,mail);
        Paper.book().write(Prevalent.UserPasswordKey,password);
    }
    final DatabaseReference RootRef;
    RootRef = FirebaseDatabase.getInstance().getReference();
    RootRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (snapshot.child(parentDbName).child(mail).exists())
            {
                Users usersData = snapshot.child(parentDbName).child(mail).getValue(Users.class);
                if (usersData.getMail().equals(mail))
                {
                    if (usersData.getPassword().equals(password))
                    {
                        if (parentDbName.equals("Users")){
                            loadingBar.dismiss();
```

```
                    Toast.makeText(LoginActivity.this, "Вхід...", Toast.LENGTH_SHORT).show();
                    Intent homeIntent = new Intent (LoginActivity.this,HomeActivity.class);
                    startActivity(homeIntent);
                }
                else if (parentDbName.equals("Admins")){
                    loadingBar.dismiss();
                    Toast.makeText(LoginActivity.this, "Вхід...", Toast.LENGTH_SHORT).show();
                    Intent homeIntent = new Intent (LoginActivity.this,CategoryActivity.class);
                    startActivity(homeIntent);
                }
            }
            else
            {
                loadingBar.dismiss();
                Toast.makeText(LoginActivity.this, "Не правильний пароль", Toast.LENGTH_SHORT).show();
            }
        }
    }
    else
    {
        loadingBar.dismiss();
        Toast.makeText(LoginActivity.this, "Аккаунта не існує", Toast.LENGTH_SHORT).show();
        Intent registerIntent = new Intent (LoginActivity.this,RegisterActivity.class);
        startActivity(registerIntent);
    }
}
@Override
public void onCancelled(@NonNull DatabaseError error) {
}
});
}
}
```

//RegisterActivity.java

```
package com.example.mybpd;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.HashMap;
public class RegisterActivity extends AppCompatActivity {
    private Button registerBtn;
    private EditText usernameInput;
    private EditText mailInput;
    private EditText passwordInput;
    private ProgressDialog loadingBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
```

```
    setContentView(R.layout.activity_register);
    registerBtn=(Button)findViewById(R.id.register_btn);
    usernameInput=(EditText) findViewById(R.id.register_username_input);
    mailInput=(EditText) findViewById(R.id.register_mail_input);
    passwordInput=(EditText) findViewById(R.id.register_password_input);
    loadingBar = new ProgressDialog(this);
    registerBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            CreateAccount();
        }
    });
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}
private void CreateAccount() {
    String username = usernameInput.getText().toString();
    String mail = mailInput.getText().toString();
    String password = passwordInput.getText().toString();
    if (TextUtils.isEmpty(username))
    {
        Toast.makeText(this, "Введіть ім'я", Toast.LENGTH_SHORT).show();
    }
    else if (TextUtils.isEmpty(mail))
    {
        Toast.makeText(this, "Введіть почту", Toast.LENGTH_SHORT).show();
    }
    else if (TextUtils.isEmpty(password))
    {
        Toast.makeText(this, "Введіть пароль", Toast.LENGTH_SHORT).show();
```

```
    }
    else
    {
        loadingBar.setTitle("Створення аккаунта");
        loadingBar.setMessage("Будь ласка, зачекайте...");
        loadingBar.setCanceledOnTouchOutside(false);
        loadingBar.show();
        ValidateMail(username,mail,password);
    }
    }
private void ValidateMail(String username, String mail, String password) {
    final DatabaseReference RootRef;
    RootRef = FirebaseDatabase.getInstance().getReference();
    RootRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (!(snapshot.child("Users").child(mail).exists()))
            {
                HashMap<String,Object> userDataMap = new HashMap<>();
                userDataMap.put("mail",mail);
                userDataMap.put("name",username);
                userDataMap.put("password",password);
                RootRef.child("Users").child(mail).updateChildren(userDataMap)
                        .addOnCompleteListener(new OnCompleteListener<Void>() {
                            @Override
                            public void onComplete(@NonNull Task<Void> task) {
                                if (task.isSuccessful()) {
                                    loadingBar.dismiss();
                                    Toast.makeText(RegisterActivity.this, "Реєстрація успішна", Toast.LENGTH_SHORT).show();
                                    Intent loginIntent = new Intent(RegisterActivity.this, LoginActivity.class);
                                    startActivity(loginIntent);
                                }
                                else {
```

```
                        loadingBar.dismiss();
                        Toast.makeText(RegisterActivity.this, "Ошибка!", Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
        else {
            loadingBar.dismiss();
            Toast.makeText(RegisterActivity.this, "Почта " + mail + " уже існує", Toast.LENGTH_SHORT).show();
            Intent loginIntent = new Intent(RegisterActivity.this, LoginActivity.class);
            startActivity(loginIntent);
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
  });
 }
}
```

//CategoryActivity.java

```
package com.example.mybpd;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
public class CategoryActivity extends AppCompatActivity {
   private ImageView meat, oatmeal,fish;
```

```
private ImageView chicken, salad,potato;
private ImageView seat, mushrooms,soup;
private ImageView alcohol, vipechcka,fruits;
private ImageView spices, dairy,chees;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_category);
    init();
    meat.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
            intent.putExtra("category","meat");
            startActivity(intent);
        }
    });
    oatmeal.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
            intent.putExtra("category","oatmeal");
            startActivity(intent);
        }
    });
    fish.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
            intent.putExtra("category","fish");
            startActivity(intent);
        }
```

```
});
chicken.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","chicken");
        startActivity(intent);
    }
});
salad.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","salad");
        startActivity(intent);
    }
});
potato.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","potato");
        startActivity(intent);
    }
});
seat.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","seat");
        startActivity(intent);
    }
});
```

```
mushrooms.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","mushrooms");
        startActivity(intent);
    }
});
soup.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","soup");
        startActivity(intent);
    }
});
alcohol.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","alcohol");
        startActivity(intent);
    }
});
vipechcka.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","vipechcka");
        startActivity(intent);
    }
});
fruits.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","fruits");
        startActivity(intent);
    }
});
spices.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","spices");
        startActivity(intent);
    }
});
dairy.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","dairy");
        startActivity(intent);
    }
});
chees.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(CategoryActivity.this,AdminAddNewProductActivity.class);
        intent.putExtra("category","chees");
        startActivity(intent);
    }
});
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
    Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
```

```
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
    private void init(){
        meat = findViewById(R.id.meat);
        oatmeal = findViewById(R.id.oatmeal);
        fish = findViewById(R.id.fish);
        chicken = findViewById(R.id.chicken);
        salad = findViewById(R.id.salad);
        potato = findViewById(R.id.potato);
        seat = findViewById(R.id.seat);
        mushrooms = findViewById(R.id.mushrooms);
        alcohol = findViewById(R.id.alcohol);
        vipechcka = findViewById(R.id.vipechcka);
        fruits = findViewById(R.id.fruits);
        spices = findViewById(R.id.spices);
        dairy = findViewById(R.id.dairy);
        chees = findViewById(R.id.chees);
    }
}
```

//Users.java

```
package com.example.mybpd.Model;
public class Users {
    private String name;
    private String mail;
    private String password;
    public Users()
    {
    }
    public Users(String name, String mail, String password) {
        this.name = name;
```

```
        this.mail = mail;
        this.password = password;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

//HomeActivity.java

```
package com.example.mybpd;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
```

```
import androidx.core.view.WindowInsetsCompat;
import io.paperdb.Paper;
public class HomeActivity extends AppCompatActivity {
    private Button logutBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);
        logutBtn =(Button) findViewById(R.id.button);
        logutBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Paper.book().destroy();
                Intent logoutIntent = new Intent(HomeActivity.this,MainActivity.class);
                startActivity(logoutIntent);
            }
        });
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
}
```

Додаток Г. Ілюстративний матеріал

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

Бакалаврська дипломна робота на тему
Розробка програмного мобільного застосунку для персонального консультування з приготування їжі на платформі Android з використанням технології Java

ВИКОНАВ : СТУДЕНТ IV КУРСУ ГРУПИ 1ПІ-20Б
СПЕЦІАЛЬНОСТІ 121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЩЕРБАЦЬКИЙ Б.І.
КЕРІВНИК : К.Т.Н ., ДОЦ . КАТЄЛЬНІКОВ Д.І
РЕЦЕНЗЕНТ :

Мета, предмет та об'єкт дослідження

Мета - скорочення часу для отримання рецептів смачної та поживної їжі в мобільному додатку.

Предмет дослідження – є методи і засоби розробки програмного додатку «Pro100 Cooked»

Об'єкт дослідження– є процес розробки мобільних додатків на платформі Android.

Актуальність

ЗІ СТРІМКИМ РОЗВИТКОМ ТЕХНОЛОГІЙ МОБІЛЬНІ ДОДАТКИ СТАЮТЬ ДЕДАЛІ ПОПУЛЯРНІШИМИ В РІЗНИХ ГАЛУЗЯХ . ПОШУК РЕЦЕПТІВ ЗА ПРОДУКТАМИ Є ПЕРСПЕКТИВНИМ У ЦІЙ ГАЛУЗІ, ЩО НАДАЄ МОЖЛИВІСТЬ ДОЗВОЛЯЮЧИ ЇМ ЕФЕКТИВНО ВИКОРИСТОВУВАТИ НАЯВНІ ПРОДУКТИ, ЩО ЗМЕНШУЄ ЗАЙВІ ВИТРАТИ ТА СПРИЯЄ УНИКНЕННЮ МАРНОЇ ЇЖІ . РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ - ПЕРСОНАЛЬНОГО КОНСУЛЬТАНТА З ПРИГОТУВАННЯ ЇЖІ Є ВАЖЛИВИМ ЗАВДАННЯМ У ЦЬОМУ ПЛАНІ, ОСКІЛЬКИ ДОЗВОЛЯЄ СТВОРЮВАТИ ІННОВАЦІЙНІ РІШЕННЯ, ЯКІ ПОКРАЩУЮТЬ КОРИСТУВАЦЬКИЙ ДОСВІД ТА НАДАЮТЬ КОНКУРЕНТНІ ПЕРЕВАГИ ПРОЕКТУ .

Практичнацінність

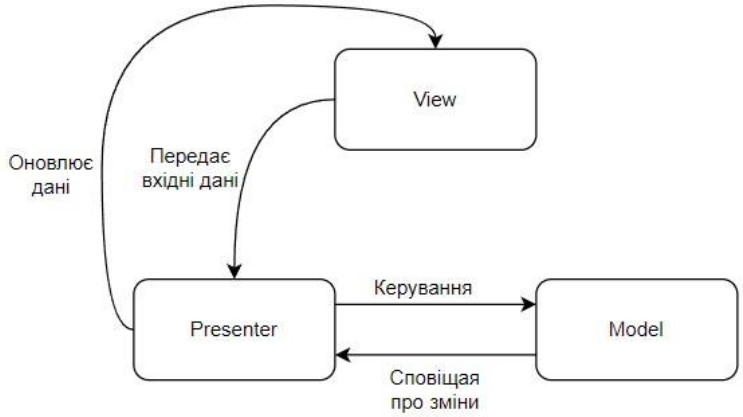
- Покращення пошуку потрібного рецепту;
- Оптимізація фільтрації та сортування;
- Запобігання потрапляння алергенів в страви

Новизна

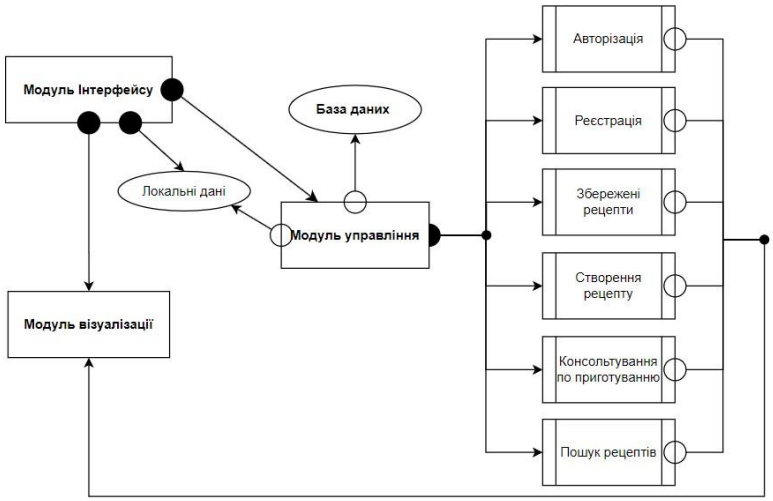
ПОДАЛЬШОГО РОЗВИТКУ ОТРИМАЛА ТЕХНОЛОГІЯ ФІЛЬТРАЦІЇ ТА СОРТУВАННЯ ПЕРВИННИХ ДАНИХ, У ЯКІЙ НА ВІДМІНУ ВІД ІСНУЮЧИХ ПІДХОДІВ ДО ОБРОБКИ ДАНИХ, ЗАСТОСОВУЮТЬСЯ СУЧАСНІ ЗАСОБИ МОВИ JAVA : JAVA STREAM API ТА ЛЯМБДА - ВИРАЗІВ, ЩО ДОЗВОЛИЛО ЕФЕКТИВНО ОБРОБЛЯТИ ДАНІ, ВИКОНУЮЧИ ОПЕРАЦІЇ ФІЛЬТРАЦІЇ, ТРАНСФОРМАЦІЇ, ЗГОРТАННЯ ТА СОРТУВАННЯ . ОКРІМ ТОГО, ЦЕ ПОЄДНАННЯ РОБИТЬ ВИХІДНІ ТЕКСТИ ПРОГРАМИ ТАКИМИ, ЩО ЛЕГКО ЧИТАЮТЬСЯ , ЩО ЗНАЧНО ПОЛЕГШИТЬ МАСШТАБУВАННЯ В МАЙБУТНЬОМУ .

Аналіз існуючих аналогів:

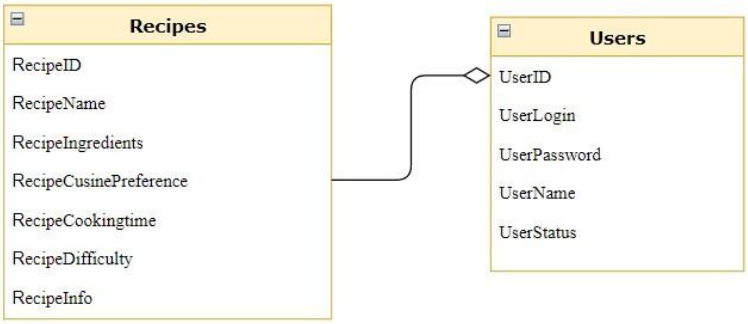
Критерії¤	Cookpad¤	Прості·рецепти· від·DIL·Studio¤	Supercook¤	My· recipe· Box¤	Pro100¶ Cooked¤
Робота·без·інтернет· з'єднання·¤	-¤	+¤	-¤	+¤	+¤
Фільтри·за·категорією¤	+¤	+¤	+¤	-¤	+¤
Функція·не·строгого· пошуку¤	-¤	-¤	+¤	-¤	+¤
Змога·додавання· власних·рецептів¤	+¤	+¤	-¤	+¤	+¤
Наявність·української· мови¤	+¤	+¤	-¤	+¤	+¤
Загальна·оцінка¤	60%¤	80%¤	40%¤	60%¤	100%¤



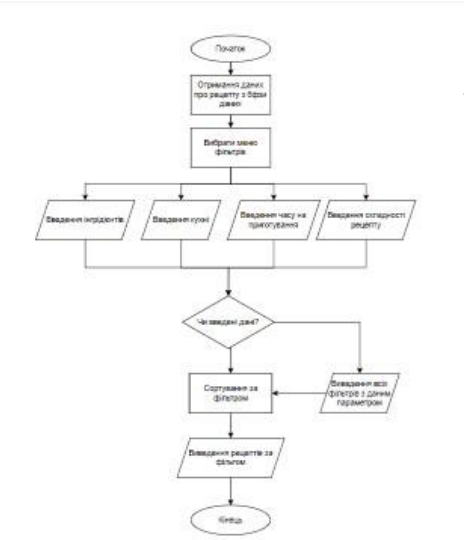
Архітектура патерну проектування MVP



Структурна карта Константайна додатку



База даних додатку

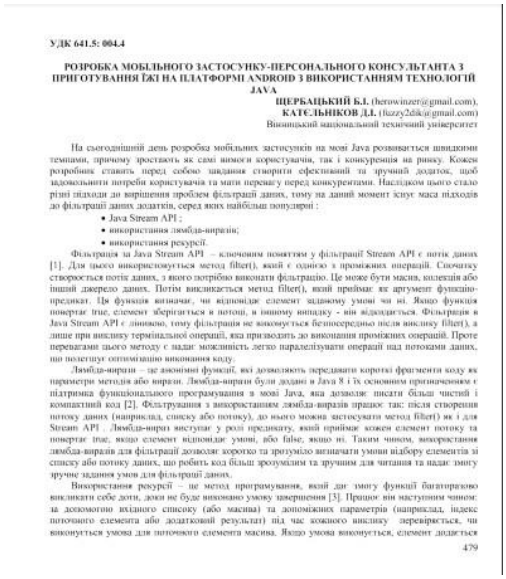


Блок-схема алгоритму фільтрації та сортування рецептів

Апробація та публікація результатів роботи

Матеріали бакалаврської дипломної роботи доповідалися на:

XXIV Всеукраїнська науково - технічна конференція молодих вчених, аспірантів та студентів «СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ» (ВНТУ, 2024).



УДК 641.5: 004.4

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ-ПЕРСОНАЛЬНОГО КОНСУЛЬТАНТА З ПРИГОТУВАННЯ ЇЖІ НА ПЛАТФОРМІ ANDROID З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ JAVA

ЩЕРБАЦЬКИЙ Б.І. (herowinzer@gmail.com),
КАТЄЛЬНІКОВ Д.І. (fuzzy2dik@gmail.com)
Вінницький національний технічний університет

На сьогоднішній день розробка мобільних застосунків на мові Java розвивається швидкими темпами, причому зростають як самі вимоги користувачів, так і конкуренція на ринку. Кожен розробник ставить перед собою завдання створити ефективний та зручний додаток, щоб задовольнити потреби користувача та мати перевагу перед конкурентами. Наслідком цього стало різні підходи до вирішення проблем фільтрації даних, тому на даний момент існує маса підходів до фільтрації даних додатків, серед яких найбільш популярні :

- Java Stream API ;
- використання лямбда-виразів;
- використання рекурсії.

Фільтрація за Java Stream API – ключовим поняттям у фільтрації Stream API є потік даних [1]. Для цього використовується метод filter(), який є однією з проміжних операцій. Спочатку створюється потік даних, з якого потрібно виконати фільтрацію. Це може бути масив, колекція або інший джерело даних. Потім викликається метод filter(), який приймає як аргумент функціональний предикат. Ця функція визначає, чи відповідає елемент заданому умові чи ні. Якщо функція повертає true, елемент зберігається в потоці, в іншому випадку - він відкидається. Фільтрація в Java Stream API є лінивою, тому фільтрація не виконується безпосередньо після виклику filter(), а лише при виклику термінальної операції, яка призводить до виконання проміжних операцій. Проте перевагами цього методу є надає можливість легко паралелізувати операції над потоками даних, що полегшує оптимізацію виконання коду.

Лямбда-вирази – це анонімні функції, які дозволяють передавати короткі фрагменти коду як параметри методів або вирази. Лямбда-вирази були додані в Java 8 і їх основним призначенням є підтримка функціонального програмування в мові Java, яка дозволяє писати більш чистий і компактний код [2]. Фільтрування з використанням лямбда-виразів працює так: після створення потоку даних (наприклад, списку або потоку), до нього можна застосувати метод filter() як і для Stream API . Лямбда-вирої виступає у ролі предикату, який приймає кожен елемент потоку та повертає true, якщо елемент відповідає умові, або false, якщо ні. Таким чином, використання лямбда-виразів для фільтрації дозволяє коротко та зрозуміло визначати умови відбору елементів зі списку або потоку даних, що робить код більш зрозумілим та зручним для читання та надає змогу зручно задання умов для фільтрації даних.

Використання рекурсії – це метод програмування, який дає змогу функції багаторазово викликати себе доти, доки не буде виконано умову завершення [3]. Працює він наступним чином: за допомогою вхідного списоку (або масива) та допоміжних параметрів (наприклад, індекс поточного елемента або додатковий результат) під час кожного виклику перевіряється, чи виконується умова для поточного елемента масива. Якщо умова виконується, елемент додається

479

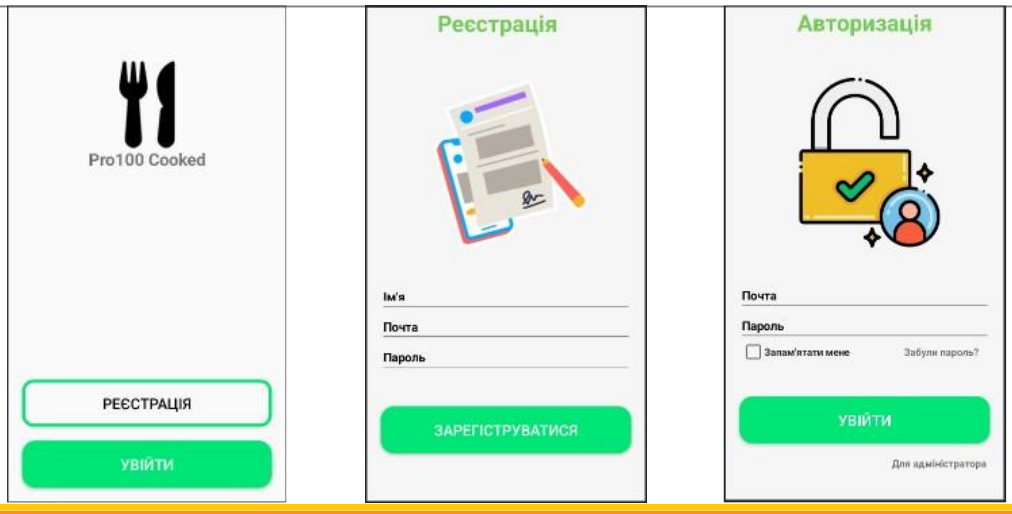
Використані технології

Android Studio

Java

Firebase

Реєстрація та вхід у додаток



Вибір фільтру та рецепту

Створення рецепту та зміна даних у профілі користувача

Висновок

У БАКАЛАВРСЬКІЙ ДИПЛОМНІЙ РОБОТІ БУЛО РОЗРОБЛЕНО МОБІЛЬНИЙ ДОДАТОК «PRO 100 COOKED ». ПІД ЧАС РОБОТИ БУЛО ОЗНАЙОМЛЕНА ЗІ СТАНОМ ПРОБЛЕМИ, ПОРІВНЯНО РОЗРОБЛЮВАНИЙ ПРОГРАМНИЙ ДОДАТОК З ІСНУЮЧИМИ АНАЛОГАМИ ТА ОПИСАНО АЛГОРИТМИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ПРОБЛЕМ ЗАСОБІВ МОБІЛЬНОГО ЗАСТОСУНКУ - ПЕРСОНАЛЬНОГО КОНСУЛЬТАНТА З ПРИГОТУВАННЯ ЇЖІ З ВИКОРИСТАННЯМ ФІЛЬТРАЦІЇ ДАНИХ .

БУЛО ОБРАНО ГРАФІЧНИЙ ІНТЕРФЕЙС . ОПИСАНІ ГРАФІЧНІ ІНТЕРФЕЙСИ ПРОГРАМНОГО ДОДАТКУ . РОЗРОБЛЕНО БЛОК - СХЕМИ АЛГОРИТМІВ ПРОГРАМНОГО ДОДАТКУ . РОЗРОБЛЕНО І ОПИСАНО РОБОТУ ШТУЧНОГО ІНТЕЛЕКТУ .

ПІД ЧАС АНАЛІЗУ ТЕХНОЛОГІЙ РОЗРОБКИ БУЛО ОБГРУНТОВАНО ВИБІР МОВИ ПРОГРАМУВАННЯ JAVA ТА БІБЛІОТЕКИ STREAM API .

БУЛО РОЗРОБЛЕНО СХЕМИ ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ МОБІЛЬНОГО ЗАСТОСУНКУ . А ТАКОЖ РОЗРОБЛЕНО МОДЕЛЬ СИСТЕМИ З ВИКОРИСТАННЯМ UML ДІАГРАМ .

ТЕСТУВАННЯ ПРОГРАМИ ДОВЕЛО ПОВНУ ПРАЦЕЗДАТНІСТЬ ДАНОГО ПРОГРАМНО ГО ПРОДУКТУ ТА ВІДПОВІДНІСТЬ ПОСТАВЛЕНОМУ ТЕХНІЧНОМУ ЗАВДАННЮТАКОЖ БУЛО РОЗРОБЛЕНО ІНСТРУКЦ ІЮ КОРИСТУВАЧА.

Дякую за увагу