

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ
СИСТЕМИ ДЛЯ ФОРМУВАННЯ ХАРЧОВИХ ЗВИЧОК
RESEARCH AND DEVELOPMENT OF A RECOMMENDATION
SYSTEM TO FORM THE EATING HABITS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-222м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

СНУРНІЦІН Б.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ТАБУНЩИК Г.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СНУРНІЦИНА Богдана Романовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та розробка рекомендаційної системи
для формування харчових звичок.
Research and Development of a Recommendation System to Form the Eating Habits.

керівник проєкту (роботи) к.т.н., професор, ТАБУНЩИК Галина Володимирівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи яка проектується. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ТАБУНЩИК Г.В., професор		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст.викл.		

7. Дата видачі завдання « 04 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області та програмних засобів, що будуть використовуватись в роботі	2 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки	3 тиждень	Розділ 2
4	Розробка архітектури програми та логічної моделі рекомендаційної системи.	4-7 тижні	Розділ 3
5	Розробка графічного інтерфейсу програми.	7-8 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї	10 тиждень	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

_____ Богдан СНУРНЩИН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Галина ТАБУНЩИК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
171с., 56 рис., 3 табл., 3 дод., 16 джерел.

РЕКОМЕНДАЦІЙНА СИСТЕМА, СИСТЕМА ПОДІБНОСТІ,
ХАРЧУВАННЯ, ЗДОРОВ'Є, ЗАСТОСУНОК, МОБІЛЬНИЙ ЗАСТОСУНОК,
ANDROID, KOTLIN.

Об'єкт дослідження – програмні засоби для ведення раціону і формування харчових звичок.

Предмет дослідження – застосунок призначений для формування та вплив харчових звичок на здоров'я та якість життя з використанням рекомендаційної системи.

Мета роботи – розробка рекомендаційної системи та впровадження у застосунок для формування здорових харчових звичок з метою покращення якості життя та здоров'я користувачів, з легким для розуміння інтерфейсом.

Матеріали, методи та технічні засоби: функціональне та об'єктно-орієнтоване програмування, мобільний пристрій з операційною системою Android, мова програмування Kotlin.

Результати. Створено рекомендаційну систему для формування харчових звичок користувача, впроваджену в Android-застосунок.

Висновки. Розробка рекомендаційної системи для формування харчових звичок є актуальною та перспективною задачею. Отримані результати свідчать про успішність підходу та практичне значення системи для покращення якості харчування та здоров'я користувачів.

Галузь використання – використання системи користувачами задля розвитку та формування харчових звичок.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 171 pages, 56 figures, 3 tables, 3 appendixes, 16 sources.

RECOMMENDATION SYSTEM, SIMILARITY SYSTEM, NUTRITION, HEALTH, APP, MOBILE APP, ANDROID, KOTLIN.

The object of research is software tools for maintaining the diet and forming food habits.

The subject of the study is an application designed to form and influence food habits on health and quality of life using a recommendation system.

The aim of the work is to develop a recommendation system and implement it in the application for forming healthy eating habits in order to improve the quality of life and health of users, with an easy – to-understand interface. Materials, methods and technical means: functional and Object-Oriented Programming, mobile device with the Android operating system, Kotlin programming language.

Results. A recommendation system for forming the user's eating habits has been created, implemented in the Android app.

Conclusions. Developing a recommendation system for forming food habits is an urgent and promising task. The results obtained indicate the success of the approach and the practical importance of the system for improving the quality of nutrition and health of users.

The scope of use is the use of the system by corseters for the development and formation of food habits.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	7
Вступ	8
1 Аналіз предметної області	9
1.1 Знайомство з предметною областю	9
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання	9
1.3 Висновки.....	13
2 Вибір та обґрунтування структури системи яка проектується	15
2.1 Вибір інструментарію розробки програмного забезпечення.....	15
2.2 Структура файлів Android застосунку	20
3 Основні рішення щодо реалізації компонентів системи	22
3.1 Реалізація архітектури та патернів програмування	22
3.2 Огляд компонентів та розроблених класів	30
3.3 Огляд реалізації рекомендаційної системи та аналізу даних	44
3.4 Огляд візуального інтерфейсу застосунку	58
4 Експлуатація, тестування та експериментальне дослідження застосунку	71
4.1 Процес роботи застосунку	71
4.2 Тестування застосунку	72
4.3 Висновки за розділом 4.....	73
Висновки.....	74
Перелік джерел посилання	76
Додаток А Технічне завдання	78
Додаток Б Текст програми.....	82
Додаток В Слайди презентації	162

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- IDE – Integrated Development Environment;
ОС – операційна система;
ПЗ – програмне забезпечення;

ВСТУП

Сучасний ритм життя, постійний стрес та нестабільна екологічна ситуація суттєво впливають на харчові звички суспільства. Здорове харчування стає необхідністю для підтримання оптимального фізичного та психічного здоров'я. У зв'язку з цим, дослідження та розробка рекомендаційної системи для формування харчових звичок стають актуальними завданнями, спрямованими на поліпшення якості життя та профілактику хронічних захворювань.

Метою створення такої системи стало декілька факторів: по-перше, це надання користувачам ефективного інструмента, спрямованого на покращення якості їхнього харчування та сприяння формуванню здорових харчових звичок. По-друге, популяризація нутриціології та дієтології як інструмент для побудови якісного життя, якими все більше починає цікавитись суспільство. Третім фактором є сприяння зменшенню ризиків розвитку хронічних захворювань, пов'язаних із харчуванням, через формування здорових харчових звичок. Усі ці фактори об'єднують у собі одну головну ціль – створити загальну рекомендаційну систему для користувачів по всьому світу задля формуванню здорових харчових звичок для них, сприяючи збалансованому харчуванню та вибору продуктів, що сприяють здоров'ю.

YumHub стане у нагоді для кожного, хто прагне вести здоровий спосіб життя та отримувати індивідуалізовані поради з вибору продуктів та раціону на основі аналізу харчових вподобань та потреб користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Знайомство з предметною областю

Основні функції застосунку – надання користувачам персоналізованих та ефективних порад щодо харчування, а тобто: збір та аналіз даних про харчові вподобання користувачів на основі історії їхнього харчування, надання індивідуалізованих порад з вибору продуктів, раціону та збалансованого харчування, враховуючи особливості користувача, тощо. Зробивши дослідження на наявність аналогів, що вирішують такі ж проблеми, можна зазначити, що всі вони відрізняються як за інтерфейсом так і за функціоналом.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

1.2.1 Застосунок «MyFitnessPal»

MyFitnessPal – один з найпопулярніших мобільних додатків та веб-платформ для ведення щоденника харчування та моніторингу фізичної активності[1]. Застосунок став допоміжним інструментом для тих, хто прагне підтримувати здоровий спосіб життя, контролювати вагу та отримувати персоналізовані рекомендації з харчування (рис. 1.1). Користувачі зазначають, що застосунок може занадто сильно фокусуватися на кількості калорій, іноді недостатньо враховуючи інші важливі аспекти харчування, такі як якість продуктів чи збалансованість дієти. Також застосунок має чат для спілкування та поширенню між користувачами своїми досягненнями. Інтерфейс MyFitnessPal спроектований просто та логічно, що полегшує користувачам ведення щоденника та використання різних функцій, але в той же час для нових користувачів може здаватися, що інтерфейс містить багато інформації, що може

бути не важливо для кожного. Головним недоліком є те, що більшість функцій та сама рекомендаційна система не буде функціонувати без преміум-підписки.

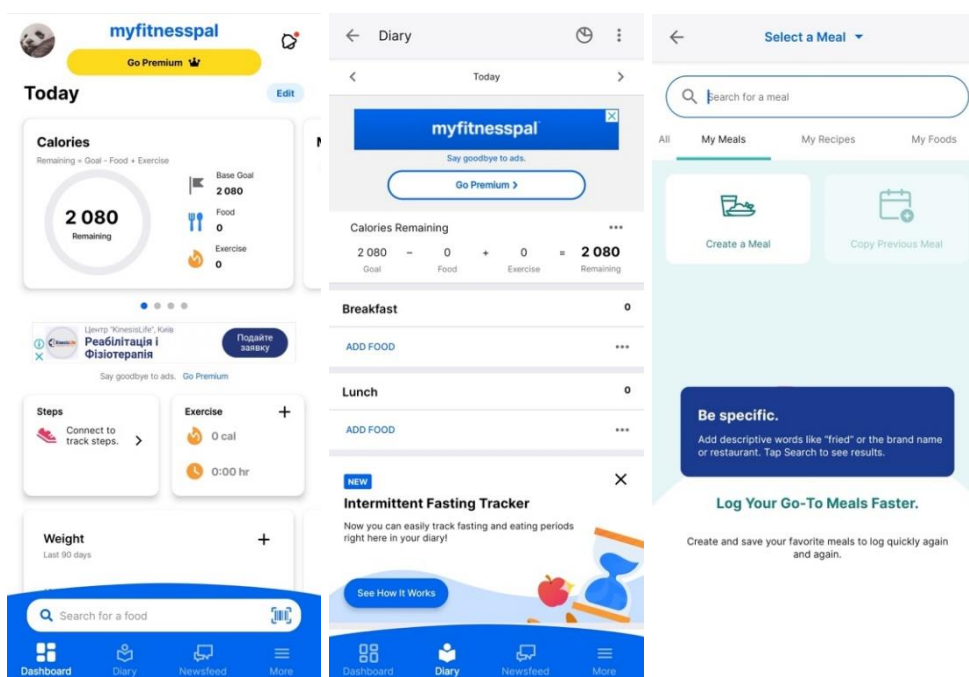


Рисунок 1.1 – Відображення застосунку «MyFitnessPal»

1.2.2 Застосунок «Yazio»

Yazio - це додаток для ведення щоденника харчування та підтримки здорового способу життя[2]. Додаток дозволяє користувачам вносити дані про свої щоденні харчові звички, створювати плани харчування, а також відстежувати свою фізичну активність (рис. 1.2). Деякі користувачі вважають, що вартість платної версії Yazio висока. Деякі розширені функції та можливості доступні лише у преміум-версії, що може бути не привабливим для користувачів, які шукають безкоштовні варіанти. Проаналізувавши, можна також зазначити, що реєстрація займає багато часу і має забагато зайвих питань.

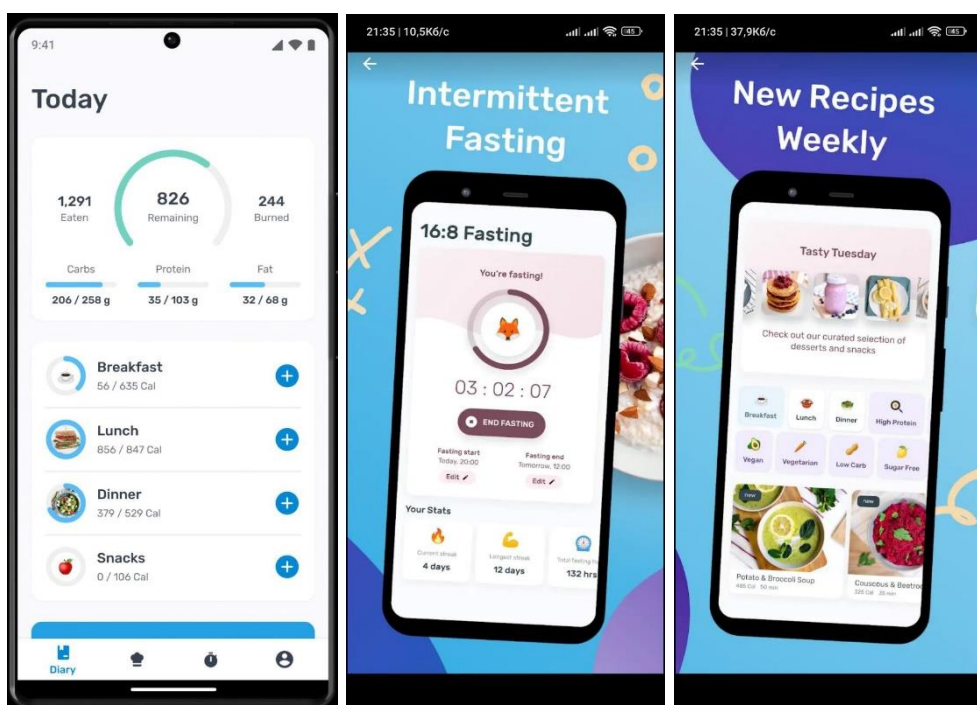


Рисунок 1.2 – Зображення інтерфейсу застосунку «Yazio»

1.2.3 Застосунок «Nutrilio»

Застосунок «Nutrilio» [3] додаток для ведення свого плану харчування зі налаштованою рекомендаційною системою, підрахунком калорій. Також він дозволяє створювати план харчування та вести щоденник вживаних страв та напоїв за день (рис. 1.3). З недоліків є трішки застарілий інтерфейс. Деякі користувачі зазначають, що у додатку недоступний автоматичний підрахунок калорій і відмічати час прийому їжі, що є значним недоліком аби налаштувати правильно харчові звички.

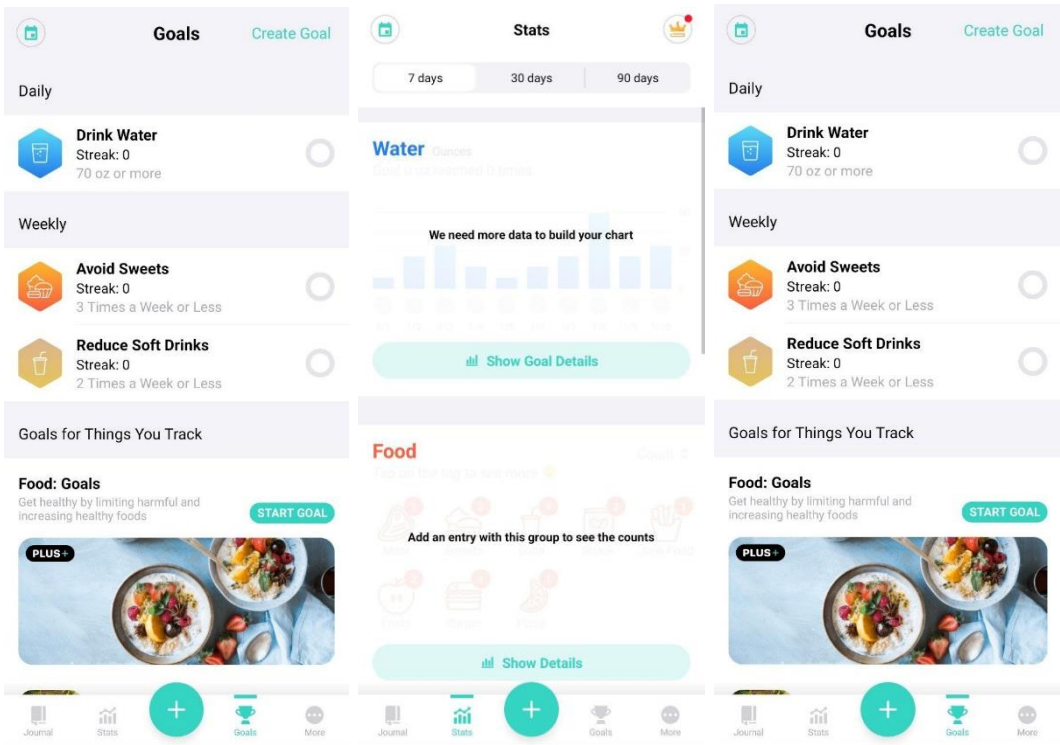


Рисунок 1.3 – Інтерфейс застосунку «Nutrilio»

1.2.4 Порівняльна таблиця програмних продуктів

Нижче в таблиці 1.1 відображено порівняльну характеристику аналогів та застосунком, що розроблюється. У якості оцінок виставляються мінуси та плюси. Чим більше плюсів отримано, тим краще себе показав застосунок під час порівняння за певним критерієм.

Таблиця 1.1 – Порівняльна характеристика застосунків-аналогів

Критерії оцінки	Назва застосунку			
	MyFitnessPal	Yazio	Nutrilio	YumHub
Інтуїтивна зрозумілість	++	++	+	++
Зручність	++	++	+	+++
Сучасність дизайну	+	++	±	++

Продовження таблиці 1.1

1	2	3	4	5
Споживання ОЗУ	Від 70МБ	Від 60 до 100МБ	Від 90МБ	Від 40 до 100МБ
Споживання електроживлення	≈100 mAh	≈180 mAh	≈220 mAh	≈190 mAh
Стабільність роботи	++	+	+	++

Результати статистики споживання ресурсів отримано за годину активного користуванням додатку в однакових умовах.

Можна зазначити, що MyFitnessPal показав себе як додаток із простим інтерфейсом, але в той же час вміщує в себе забагато інформації, що може бути складним для користувача.

Yazio добре оптимізований. Для повноцінного функціоналу додатку може знадобитися постійне інтернет-з'єднання, що може бути не зручно у певних ситуаціях. Значним недоліком є те, що за годину використання та при оформленні підписки додаток викидувало і повторне відкриття займало багато часу.

Останній додаток називає себе рекомендаційною системою, але таким не являється через деякі недопрацьовані функції, такі як автоматичний підрахунок калорій, встановлення часу вживаної їжі тощо. Nutrilio за годину використання не мав ніяких багів. Інтерфейс є трішки обмеженим і не сильно заохочує.

YumHub вирішує всі труднощі, які можна зустріти в аналогічних продуктах, і надає користувачам стабільний застосунок з пропрацьованою рекомендаційною системою.

1.2 Висновки

У першому розділі був проведений аналіз області розробки, включаючи докладний огляд трьох найпопулярніших додатків. Цей аналіз виявив як слабкі, так і сильні сторони цих додатків. На основі цього виявленої інформації

розроблено застосунок, який вирішує виявлені недоліки і використовує переваги з метою поліпшення користувацького досвіду та задоволення потреб у формуванні харчових звичок. Такий підхід робить розробку рекомендаційної системи обґрунтованою, оскільки вона включає в себе весь необхідний функціонал для зручного впровадження і контролю харчових звичок.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ ЯКА ПРОЕКТУЄТЬСЯ

У розділі розглянуті основні особливості мов програмування, вибір інструментарію та огляд структури за стосунку.

2.1 Вибір інструментарію розробки програмного забезпечення

2.1.1 Вибір платформи для якої розроблюється застосунок

Оскільки головна мета застосунку полягає в допомозі користувачам у формуванні здорових харчових звичок, вибір був зроблений на користь платформи мобільних пристроїв, що включає в себе не лише смартфони, але й планшети. Операційна система для розробки обрана Android, оскільки вона є лідером за популярністю, а отже і більше охоплення користувачів. За даними графіку [4], можна побачити, що частка ринку Android не змінюється в останні 3 роки та стабільно тримає 70% світового ринку смартфонів (рис. 2.1).

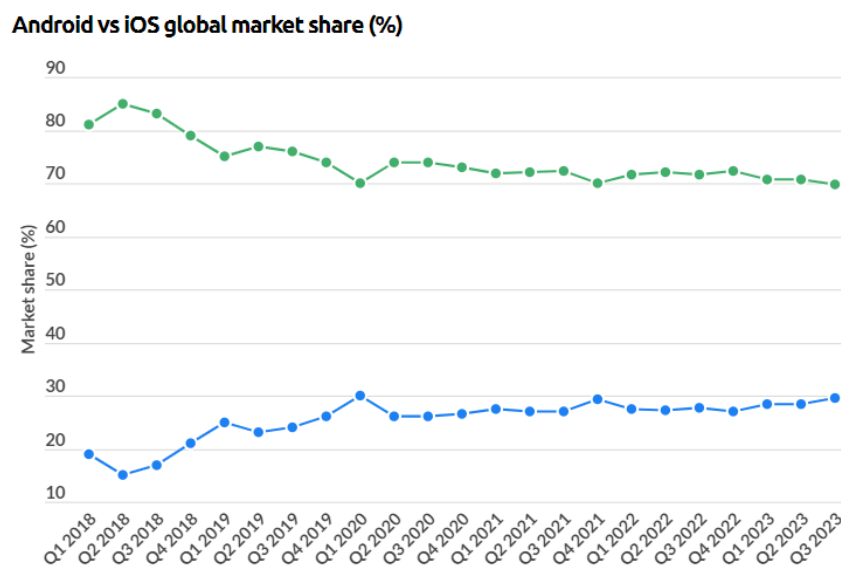


Рисунок 2.1 – Статистика глобального ринку Android та IOS [4]

Android - це мобільна ОС, розроблена компанією Google, яка є однією з найпопулярніших та широко використовуваних платформ для усіх мобільних пристроїв, як-от смартфони, планшети, годинники, компактні ПК, а також розумні телевізори. Завдяки відкритому середовищу для розробників, створення різноманітних додатків, систем та втілення найрізноманітніших ідей є реальністю і вагомою перевагою на відміну від Xcode – середовище розробки для IOS платформи, адже для цього потрібно купити принаймні один продукт компанії Apple.

Однією з основних переваг системи Android є її варіативність, яка виявляється у великій кількості пристроїв різних виробників, що пропонують різні функції та характеристики. Відкрита кодова база дозволяє модифікувати та пристосовувати операційну систему під свої потреби та бізнес ідеї.

Різнорманітність пристроїв від різних виробників та різні версії операційної системи можуть створювати проблеми для розробників, які стикаються з різними конфігураціями та специфікаціями пристроїв, адже це може призводити до проблем сумісності та ускладнювати розробку програмного забезпечення

Також, проблеми безпеки стають актуальними у зв'язку з великою кількістю додатків, доступних для завантаження, і різними політиками оновлення для різних пристроїв. Це може призвести до того, що деякі користувачі залишаються на застарілих версіях операційної системи з погіршеною безпекою.

Однак не зважаючи на ці виклики, Android продовжує залишатися однією з провідних операційних систем для мобільних пристроїв з широким спектром можливостей та активною розробкою.

2.1.2 Вибір середовища розробки

Команда Google постаралась та розробила офіційне програмне забезпечення для розробки застосунків та компонентів під операційну систему

Android – Android Studio, вона є необхідним інструментом для розробників, які вибрали платформу Android для своїх мобільних додатків. Це інтегроване середовище розробки, яке пропонує високий рівень функціональності та продуктивності для створення, тестування і вдосконалення додатків.

Android Studio має інтеграцію з системою збірки Gradle, що дозволяє ефективно керувати залежностями та здійснювати збірку проектів. Це полегшує розробникам управління складними структурами проектів та додає гнучкість в їхню роботу.

Емулятори, які надає Android Studio, спрощують розробку та тестування додатків на різних пристроях та версіях Android, враховуючи співвідношення, розміри, а також форм-фактор екранів. Це дозволяє розробникам ефективно виявляти і виправляти проблеми сумісності та забезпечує високу якість додатків.

Підтримка різних мов програмування, таких як Kotlin і Java, дає розробникам вибір у виборі технологій та забезпечує гнучкість в реалізації їхніх ідей.

Узагальнюючи, Android Studio - це не просто інструмент для написання коду, а повноцінна платформа розробки, яка допомагає творити інноваційні та високоякісні мобільні додатки для користувачів платформи Android.

Для розробки Android застосунку існує ще середовище IntelliJ IDEA від команди Jet Brains. IntelliJ IDEA виступає як потужний інструмент для розробників, наділений рядом інтелектуальних функцій, що полегшують та прискорюють процес програмування. Ця інтегрована середовище розробки пропонує вдосконалені можливості роботи з різними мовами програмування, дозволяє вести онлайн розробку у команді, а також наділений новітнім функціоналом із використанням ІІІ для доповнення коду. Це все дозволяє розробникам використовувати її у широкого спектру розробки продуктів.

Глибокий рівень інтеграції з системами збірки, такими як Maven та Gradle, робить IntelliJ IDEA ефективним інструментом для керування проектами будь-якої складності, а інтелектуальні підказки та автоматичне виправлення

коду допомагають уникнути типових помилок та забезпечують високу якість коду.

Графічний редактор інтерфейсів дозволяє розробникам зосередитися на створенні відмінних користувацьких інтерфейсів без зайвих труднощів. Підтримка Jetpack Compose Preview функціональності дає змогу у режимі реального часу, не встановлюючи застосунок на пристрій, перевіряти інтерактивні компоненти, такі як кнопки чи поля вводу. Також підтримується перегляд анімацій із середовища розробки, що полегшує та прискорює проектування продукту. Вбудована система відлагодження забезпечує швидке виявлення та усунення помилок, що дозволяє розробникам писати більш стабільний та ефективний код.

Узагальнюючи, IntelliJ IDEA - це не просто IDE, це розвинутий інструмент, який вирізняється своєю гнучкістю, інтегрованими функціями та здатністю пристосовуватися до потреб розробників на будь-якому етапі їхньої кар'єри.

2.1.3 Вибір мови програмування для розробки системи

Вибір мови програмування відіграє важливу роль у процесі створення продукту, хоч вони є лише інструментами для написання алгоритмів, важливо чітко аналізувати можливість підтримки кодової бази та наявність необхідних та актуальних бібліотек, які будуть використовуватися в проекті для вирішення тих чи інших задач. Розробки Android-додатків підтримується різними мовами програмування, таких як C#, C++ із використанням платформи розробки Unity, а також скриптові мови програмування Python та JavaScript. Проте, існують дві основні мови, які вважаються офіційними для розробки продуктів на платформі Android: Java та Kotlin.

Java вважається основою, які слід вивчити для ефективного програмування додатків, а також виступає офіційною мовою розробки для

платформи Android. Проте, з часом приходить нова та вдосконалена мова програмування - Kotlin [5].

Важливо відзначити, що Kotlin є альтернативою до мови Java і виступає обгорткою, оскільки код, написаний на Kotlin, спочатку компілюється у Java-код, а потім отримані вихідні дані передаються до компілятора Java [6] (рис. 2.3). Ця мова була створена компанією Jet Brains з метою лаконічного та типобезпечного програмування, перевищуючи можливості Java. Розробка Kotlin почалася в 2010 році, а перша публічна презентація відбулася у липні 2011 року.

Kotlin надає можливість розширювати клас або інтерфейс новою функціональністю без необхідності успадковувати клас або використовувати шаблони проектування, такі як Decorator. Це робиться за допомогою спеціальних декларацій, які називаються розширеннями[7]. Приклад функції-розширення:

```
interface PreferencesValue<T> {
    fun get(): T
    fun set(value: T)
    fun exists(): Boolean
}

fun <T> PreferencesValue<T>.update(transform: (T) -> T) =
    set(transform(get()))
```

Створення функції-розширення відбувається завдяки прописуванню імені класу-контексту (PreferencesValue) та імені функції, що розділена крапкою. У тілі функції-розширення будуть доступні усі публічні поля та методи класу-контексту. На прикладі розглянуто оновлення даних викликаючи блок, що трансформує актуальне значення отримане за допомогою методу get() та зберігає його викликаючи метод set().

Це значно спрощує розробку, адже допомагає виділяти код та виносити до іншого модулю, аби уникнути повторень.

2.2 Структура файлів Android застосунку

Кожний додаток для Android, розроблений у студії Android, відповідає чітко визначеній структурі розташування файлів, специфічній для операційної системи Android. Файли Gradle, які визначають процес збірки додатка в бінарний файл для подальшого встановлення в операційну систему Android, розташовані в корені проекту. При створенні нового проекту Android Studio автоматично створює базовий модуль з назвою "app" ("application"), включаючи компонент-активіті як точку входу в додаток. Різні типи модулів, доступні під час розробки Android-застосунку, визначено в рисунку 2.2.

Модуль діє як контейнер, що містить вихідний код програми, файли ресурсів та файли конфігурації для подальшої збірки. Кожен модуль може бути скомпільований незалежно від інших, що сприяє ефективності на великих проектах та полегшує роботу розробників, а також покращує читаємість коду.

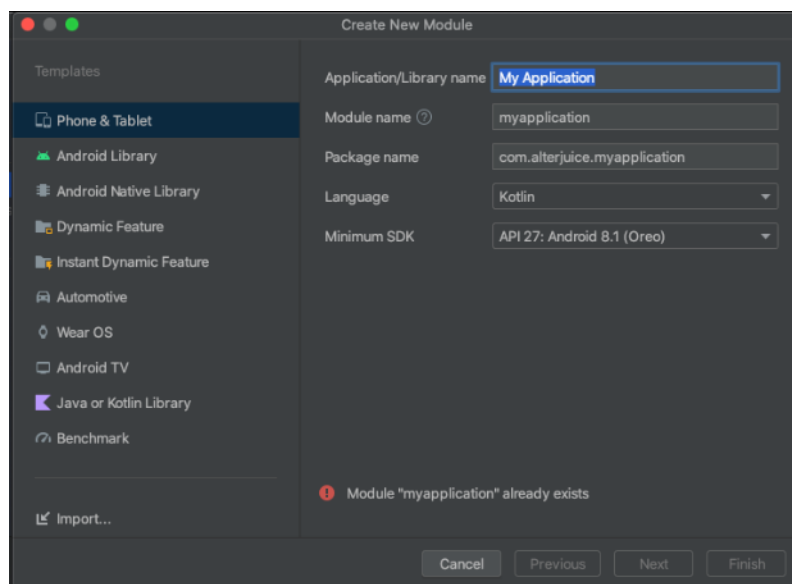


Рисунок 2.2 – Перелік шаблонів-модулів, що доступні для вибору

Для розробки додатку потребуються модулі «Phone & Tablet», «Android Library» та «Java or Kotlin Library». Інші модулі допомагають розробити

програму під інші системи, бо мають відповідні бібліотеки для роботи зі смарт-годинниками або телевізорами, та автомобільними пристроями [8].

Модулі «Phone & Tablet» та «Android Library» схожі між собою вмістом файлів маніфесту та ресурсів, але на відміну від «Phone & Tablet», що компілюється у файл APK, «Android Library» компілюється у файл Android Archive (AAR). Такий підхід дозволяє розроблювати функціонал та підключати цей модуль як залежність у інші проекти [9].

Мета модулю "Java or Kotlin Library" полягає в відокремленні бізнес-логіки, поділі на абстракції та реалізації, а також у забезпеченні доступу через інтерфейси, використовуючи відповідні модифікатори доступу, визначені у мовах програмування Java та Kotlin. Цей модуль призначений лише для зберігання коду у файлах Java або Kotlin.

Під час розробки проекту можна виявити такі директорії:

- Manifest – директорія, в якій знаходиться файл AndroidManifest.xml, який містить опис програми та перелік усіх компонентів Android, таких як Activity, Service, Receiver, ContentProvider;
- Java – директорія що містить кодову базу;
- Res – директорія, в якій зберігаються усі файли ресурсів, що можуть бути використані для роботи застосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

В даному розділі розглянуто архітектуру розробленої системи, патерни проектування, алгоритми, рішення стосовно частини інтерфейсу.

3.1 Реалізація архітектури та патернів програмування

3.1.1 Архітектура Stateful MVVM

На початковому етапі розробки програмного продукту вибір архітектури є фундаментально важливим етапом, що має величезний вплив на успішність проекту. Вірно обрана архітектура не лише полегшує процес розробки, але і сприяє підтримці, розширенню та масштабуванню системи у майбутньому.

Серед усіх популярних архітектур, таких як MVC, MVP, MVVM, була обрана MVVM (Model-View-ViewModel) як базова, із поміткою Stateful. MVVM допомагає чітко розділити бізнес-логіку системи, логіки роботи інтерфейсу а також обробку даних. Таким чином ViewModel нічого не знає про View з яким працює, а оновлення даних відбувається за допомогою реактивного підходу - використання поведінкового шаблону «Спостерігач».

Помітка Stateful для MVVM (Model-View-ViewModel) відзначається тим, що вона надає становість (statefulness) системі, що дозволяє зберігати та управляти станом даних. Це особливо важливо для додатків із складною логікою та великою кількістю інтерактивних елементів. Важливо зазначити, що такий підхід дозволяє легко вносити зміни в бізнес логіку, не впливаючи на інші вже налагоджені частини системи. Окрім цього, збереження стану додатку дає стійку та зручну взаємодію, адже стан можна зберігати і відновлювати при будь яких змінах системи.

3.1.2 Патерни проектування, що реалізовані у системі

«Спостерігач» – це поведінковий патерн проектування, який створює механізм підписки, та дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах [10]. Такий патерн був застосований для реагування на зміни у базі даних системи та своєчасно оновлювати інтерфейс.

Ідея «Singleton» полягає у гарантії створення та зберігання лише одного екземпляру класу у момент роботи програми, що може мати глобальну точку доступу до нього [11]. Це зроблено задля того щоб бути впевненим, що дані будуть актуальні в усіх частинах роботи додатку. Об'єкт-одинак імплементовано у мову Kotlin на рівні із іншими типами класів і його можна створити за допомогою ключового слова «object {ім'я}». Зазвичай такого підходу достатньо для зберігання даних під час роботи програми, але якщо є потреба зробити доступ до екземпляру потокобезпечним, то потрібно реалізувати це самостійно [12]. Реалізація сінглтону у проектованій системі досягається шляхом використання такого компоненту як Dependency Injection, що дає змогу налаштовувати правила створення об'єктів.

Dependency Injection це ще один шаблон проектування, що відокремлює створення об'єктів згідно із встановленими правилами і допомагає впроваджувати одні компоненти системи у інші. В результаті цього це робить розробку модулів більш легшими до тестуваннями. Як правило, для впровадження компонентів використовуються інтерфейси, а не реалізації.

Було також застосовано шаблон «Стан», що описує статус створення акаунту користувача (рис. 3.1) та шаблон «Адаптер», який дає змогу системі опрацьовувати одні типи даних та перетворювати на інші, згідно із встановленою логікою та потреб. У Андроїд поширеними адаптерами є ListAdapter та RecyclerView.Adapter.

```
sealed interface UserRegistrationStatus {
    object NotRegistered: UserRegistrationStatus
    data class InProgress(
        val userInfoSnapshot: UserInfo
    ): UserRegistrationStatus
    data class Registered(
        val userInfo: UserInfo
    ): UserRegistrationStatus
}
```

Рисунок 3.1 – Інтерфейс UserRegistrationStatus, що повторює патерн «Стан»

3.1.3 Вибір інструменту розпаралелювання задач

Основою будь-якої програми є потік Main, що також має називається UI-потік. Саме з нього створюються інші потоки та оновлюється інтерфейс користувача. Однак хорошою практикою є створення додаткових потоків задля виконання важких або довготривалих.

Важкими або довготривалими задачами є запити та робота з файловою системи (зчитування даних), зчитування та оновлення інформації у базі даних, посилення та підтримка з'єднання з віддаленим сервером. Для вирішення декількох задач одночасно якраз використовують розпаралелювання. Створення потоків відбувається за допомогою використання класу Thread із стандартної бібліотеки `java.util.concurrent` шляхом поміщення об'єкту типу `Runnable` у конструктор класу. Об'єкт типу `Runnable` реалізує єдиний метод інтерфейсу із назвою `run`, та має містити алгоритм-опис роботи, що має виконуватись на іншому потоці. Для запуску потоку використовується метод `start`, а для завершення – метод «`stop`» або «`interrupt`». Використання методу `stop` не гарантує вивільнення зайняті ресурси системи. Так як потоки не мають вбудованих методів інформування та передачі результатів обробки, прийнято отримувати оновлення реактивно, тобто використовувати функції зворотного виклику.

Ідея реактивного програмування полягає в реалізації патерну «Спостерігач» (див. розділ. 3.1.1), де слухачі даних підписуються на наявний об'єкт даних та реагують на його зміни. Приклад таких даних із можливістю прослуховування змін є `StateFlow/MutableStateFlow`. Додатково можна імпортувати бібліотеку з функціями розширення над класами `Kotlin Flow`. Вони дають змогу зв'язуватись із життєвим циклом компонентів в `Android` аби вчасно відписуватись від оновлень даних задля стабільності додатку та ефективної роботи системи. Підписуватись на дані можна за допомогою методу «`collect`». Метод очікує один аргументи – функцію зворотного виклику, що буде викликатись на зміни у даних.

Асинхронність та багатопотоковість – це різні речі. Виділення потоку є важкою операцією, тож не потрібно зловживати цим, а ініціалізацію треба проводити під час завантаження основних компонентів системи. При асинхронному програмуванні задачі будуть ділитись на одному потоці та поділяти процесорний час. Якщо такої можливості немає, то створюється додатковий потік.

`RxJava` та `Kotlin Coroutines` (далі – корутини) схожі між собою. Вони вміщують механізми асинхронності та патерн «Спостерігач». Також вони мають приблизно один функціонал, а саме: можливість створення даних, підписки на дані, наявність операторів трансформування даних(`map`, `flatMap`, `transform`, `filter`), та вибір потоку, на якому ці дані будуть створюватись, змінюватись та отримуватись підписниками. Це реалізується за допомогою методу `flowOn(..)` та аргументу одного із значень з класу `Dispatchers`. Визначення та перехід від одного потоку на інший виконується за допомогою вищевказаного класу `Dispatchers`, що має заздалегідь визначені поля що описують потоки: `Main`, `Default` та `IO`. Для коректної роботи з корутинами потрібен `CoroutineScope`. За замовчуванням існує `GlobalScope`, а у додаткових бібліотеках використовується `LifecycleScope`, що залежить від життєвого циклу форм активіті та фрагментів, та `ViewModelScope`, що використовується у `ViewModel` компонентах та відповідає за життєвий цикл вьюмоделі.

Так, як Kotlin Coroutines є «рідним» механізмом розпаралелювання задач у мові програмування Kotlin, у співпраці вони дають більшу читабельність та підтримку кодової бази, легшу побудову компонентів, легке відлагодження. Своєю популярністю та підтримкою зі сторони розробників Kotlin Coroutines ще більше витісняє RxJava, тож було вирішено обрати його як основний інструмент.

3.1.4 Типи користувацьких даних та спосіб їх збереження

Для збереження даних було обрано два типи сховища: Shared Preferences та база даних.

Shared Preferences це вбудований тип сховища, що дозволяє зберігати дані у вигляді словника, тобто у вигляді ключ-значення. Цей тип сховища використовується для збереження простих даних, таких як налаштування, стани системи тощо. У системі цей тип використовується для збереження даних про самого користувача: фізичні параметри, інформація про цілі користувача, рівень фізичної активності, а також прогрес «реєстрації» користувача, інформація про кроки створення профілю.

Другий тип сховища є базою даних, що реалізується та підтримується бібліотекою Room та базується на SQLite. Room дозволяє створювати стійкі до збоїв та легко супроводжувані бази даних за рахунок підтримки анотацій та допоміжних вбудованих утиліт. Перевагою збереження даних у базі даних є швидкий доступ до структурованих записів, що дає можливість фільтрувати, групувати та перетворювати дані не тримаючи в пам'яті системи величезні масиви даних.

3.1.5 Навігація між вікнами додатку

Для реалізації навігації було обрано бібліотеку NavigationUI Component [13]. Вона з легкістю розширює існуючий функціонал переходів між вікнами активіті та фрагментами, дозволяє створювати та налагоджувати анімацію переходів. Також є підтримка передачі даних між вікнами завдяки заздалегідь визначеному списку аргументів, що притаманний тому чи іншому вікну.

Було створено допоміжний клас DefinedRoutes для збереження усіх можливих кінцевих маршрутів та навігаційний граф для можливості відображення інтерфейсу систему згідно з цими самими маршрутами.

DefinedRoutes є сховищем для об'єктів класів NavCommand (рис. 3.2). NavCommand є інтерфейсом, що вміщує такі поля як маршрут, аргументи, та допоміжні параметри що інформують про видимість верхнього бару, нижнього бару-меню, а також заголовок, для розуміння користувача.

```
interface NavCommand {  
    val rootRoute: String  
    val arguments: List<NamedNavArgument>  
    val isBottomNavigationVisible: Boolean get() = true  
    val isTopBarVisible: Boolean get() = true  
    val topBarTitle: Str?  
}
```

```
object DefinedRoutes {
    val SplashScreen = NavCommand.new( rootRoute: "splashscreen",
        isBottomNavigationVisible = false,
        isTopBarVisible = false
    )

    val Onboarding = NavCommand.new( rootRoute: "onboarding",
        isBottomNavigationVisible = false,
        isTopBarVisible = false
    )

    val ChatAssistant = NavCommand.new( rootRoute: "chatAssistant")

    val Meals = NavCommand.new( rootRoute: "meals")

    object MealPage: NavCommand by NavCommand.new( rootRoute: "meals_page",
        isBottomNavigationVisible = false
    ) {
        const val argMealID = "mealID"

        override val arguments = listOf(
            navArgument(argMealID) { type = NavType.StringType }
        )

        operator fun invoke(mealID: String): NavRoute = getNavRoute(mealID)
    }
}
```

Рисунок 3.2 - Клас DefinedRoutes, що зберігає кінцеві шляхи навігаційного контролера

Створення навігаційного графу розроблюваної системи відбувається за допомогою YumHubNavHost, що використовує функції-розширення «composable» над «NavGraphBuilder» класом та приймає маршрут як основний параметр.

@Composable

```
internal fun YumHubNavHost(
    navHostController: NavHostController,
    modifier: Modifier,
    startDestination: String = DefinedRoutes.SplashScreen.getDestination()
) {
    NavHost(
        modifier = modifier,
        navController = navHostController,
```

```

        startDestination = startDestination
    ) {
        composable(DefinedRoutes.SplashScreen) {
            SplashScreen(modifier = Modifier.fillMaxSize())
        }
        composable(DefinedRoutes.Onboarding) {
            OnboardingScreen(modifier = Modifier.fillMaxSize())
        }
        composable(DefinedRoutes.Dashboard) {
            DashboardScreen(modifier = Modifier.fillMaxSize())
        }
        composable(DefinedRoutes.Meals) {
            MealsScreen(modifier = Modifier.fillMaxSize())
        }
    }
}

```

3.1.6 Висновки за розділом 3.1

У розділі 3.1 було розглянуто основні архітектурні рішення та підходи щодо реалізації внутрішньої роботи системи та збереження даних. Також було розглянуто спосіб навігації між вікнами із використанням сучасних бібліотек . Наведено приклади, рисунки та схеми.

3.2 Огляд компонентів та розроблених класів

3.2.1 Огляд створених файлів та структури системи що розроблюється

Розроблювана система була поділена на модулі задля створенню гнучких, легко супроводжуваних та розширюваних компонентів і є важливим принципом проектування програмного забезпечення, що дає значні переваги перед монолітним підходом (модулем-одинаком):

- легше супроводження та розвиток, адже зміни в одному модулі не повинні впливати на всю систему;
- дає можливість повторного використання коду, підключаючи один модуль до інших;
- можливість паралельної розробки системи різними командами розробників;
- масштабованість – новий функціонал може бути доданий шляхом розширення чи заміни конкретних модулів, не впливаючи на решту системи;
- можливість підтримки різних технологій або мов програмування у межах одного проекту;

Загалом було створено 17 модулів (рис. 3.3):

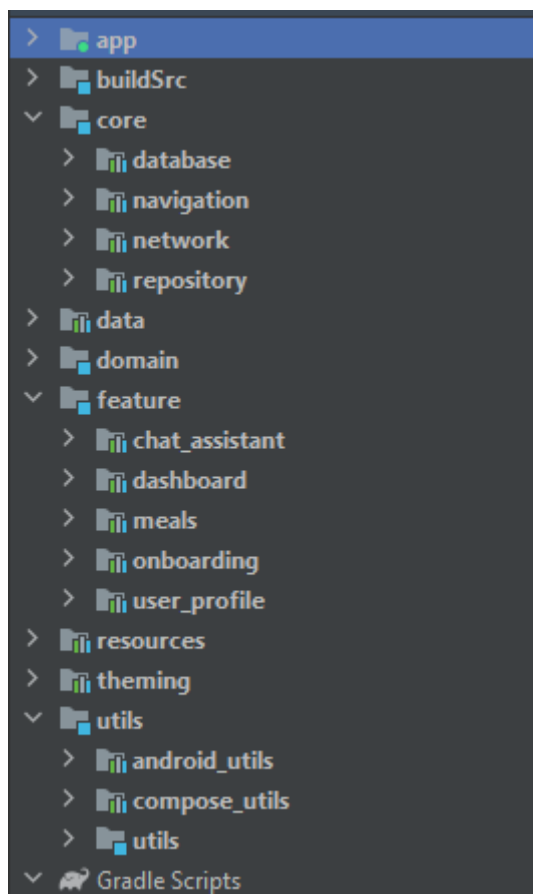


Рисунок 3.3 – Загальний вигляд модулів системи

Опис модулів:

- App – зберігає усю кодову базу, що має відношення до Android компонентів, містить компоненти, що є точками входу у систему;
- Domain – модуль, що описує усі базові типи даних з якими працює система, включаючи інтерфейси для роботи із цими даними;
- Theming, Resources – модулі, що зберігають дані щодо кастомізації інтерфейсу системи;
- Data – модуль, що зберігає загальну бізнес-логіку системи, а також містить реалізації базових інтерфейсів;
- Core:Database – модуль, що відповідає за опис структури бази даних;
- Core:Navigation – модуль, що відповідає за опис структури навігації та маршрутів;
- Core:Network – модуль, що відповідає за опис мережових запитів, структури даних;

- Core:Repository – модуль, що описує логіку збереження та отримання даних з будь яких ресурсів (віддалених, локальних);
- Feature:Chat_assistant – модуль, що зберігає UI компоненти та бізнес логіку перегляду чату із оператором;
- Feature:Dashboard – модуль, що зберігає UI компоненти та бізнес логіку основного екрану;
- Feature:Meals – модуль, що зберігає UI компоненти та бізнес логіку відображення даних;
- Feature:Onboarding – модуль, що зберігає UI та бізнес логіку створення профілю користувача;
- Feature:User_profile – модуль, що зберігає UI компоненти і функціонал функціоналу перегляду профіля користувача;
- Android_utils, Compose_utils, Utils – модулі, що зберігають допоміжні класи-утиліти.

3.2.2 Функціонал, що відповідає за відображення інтерфейсу

В традиційному підході до розробки Android-додатків, створення класів фрагментів та визначення layout файлів розмітки були невід’ємною частиною процесу створення інтерфейсу. Цей підхід, безумовно, забезпечує стабільність та визначеність, але призводить до значного обсягу коду та вимагає великої кількості файлів розмітки.

На відміну від цього, Jetpack Compose пропонує інноваційний шлях до побудови інтерфейсу, використовуючи декларативний синтаксис. Це дає можливість описувати інтерфейс динамічно, без створення окремих файлів розмітки чи класів фрагментів.

Іншою вагомою перевагою Jetpack Compose є його багатоплатформність, що дозволяє використовувати один і той самий код для розробки інтерфейсу на різних платформах, включаючи Андроїд, десктопні та веб-платформи.

Суттєвою перевагою Jetpack Compose є реактивність у рамках наявного на екрані компоненту. Це означає, що всі компоненти автоматично оновлюються при зміні даних. Це полегшує роботу зі станом системи та забезпечує ефективне перекомпонування інтерфейсу при необхідності.

3.2.3 Збереження та зчитування даних

Як вже розглядалось у розділі 3.1.4, для розроблюваної системи було реалізовано два типи сховища: `SharedPreferences` та база даних.

Кожна вибірка даних історії користувача була логічно розподілена на 5 окремих таблиць. Це дає змогу гнучкіше розподіляти дані та вести статистику.

Для покращення читаємості коду було розроблено додаткові функції, що створюють поліпшену версію роботи із полем у XML сховищі. Принцип роботи полягає в обертанні об'єкту типу `SharedPreferences` у об'єкт класу `PreferencesValue<T>` (рис. 3.4) надаючи можливість оперувати лише методами «`get()`», «`set(T)`» та «`exists()`». Обертання відбувається завдяки функціям, що відповідають шаблону назві «`Pref*Value`».

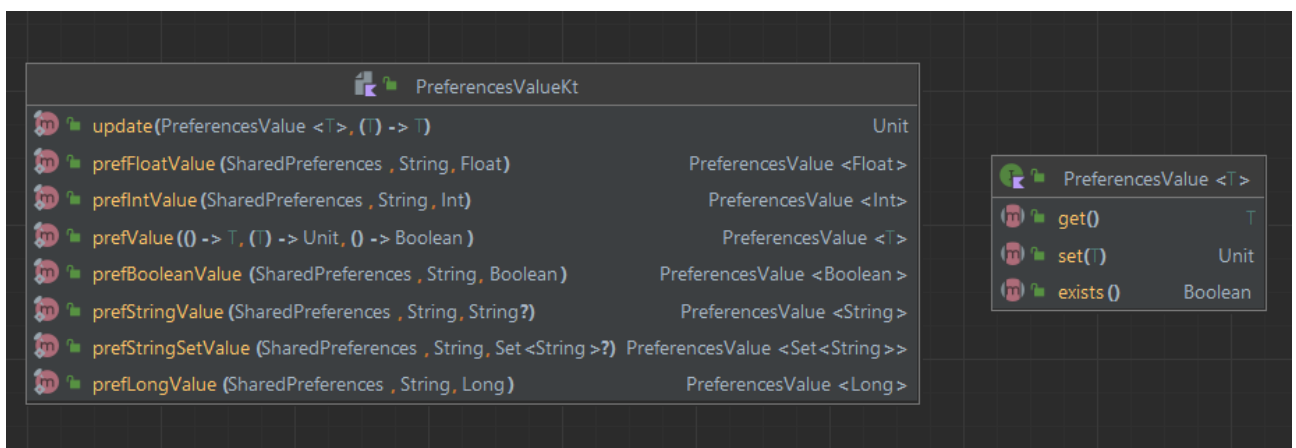


Рисунок 3.4 – Структура PreferencesValue та перелік функцій розширень

Структура інтерфейсу `PreferencesValue<T>`:

```
interface PreferencesValue<T> {
```

```

fun get(): T
fun set(value: T)
fun exists(): Boolean
}

```

Реалізація базового PrefValue метода:

```

fun <T> prefValue(
    getter: () -> T,
    setter: (value: T) -> Unit,
    exists: () -> Boolean
) = object : PreferencesValue<T> {
    override fun get() = getter()
    override fun exists() = exists()
    override fun set(value: T) = setter(value)
}

```

Реалізація методу PrefBooleanValue:

```

fun prefBooleanValue(
    preferences: SharedPreferences,
    key: String,
    defaultValue: Boolean
): PreferencesValue<Boolean> = prefValue(
    getter = { preferences.getBoolean(key, defaultValue) },
    setter = { preferences.edit { putBoolean(key, it) } },
    exists = { preferences.contains(key) }
)

```

Усі інші «Pref*Value» реалізовані за схожим принципом як і «PrefBooleanValue».

3.2.3.1 Базові класи для роботи зі сховищем

Базовий StorageCleaner інтерфейс має єдиний абстрактний метод clearData, що відповідає за очистку даних із сховища що його реалізує (рис.3.5).

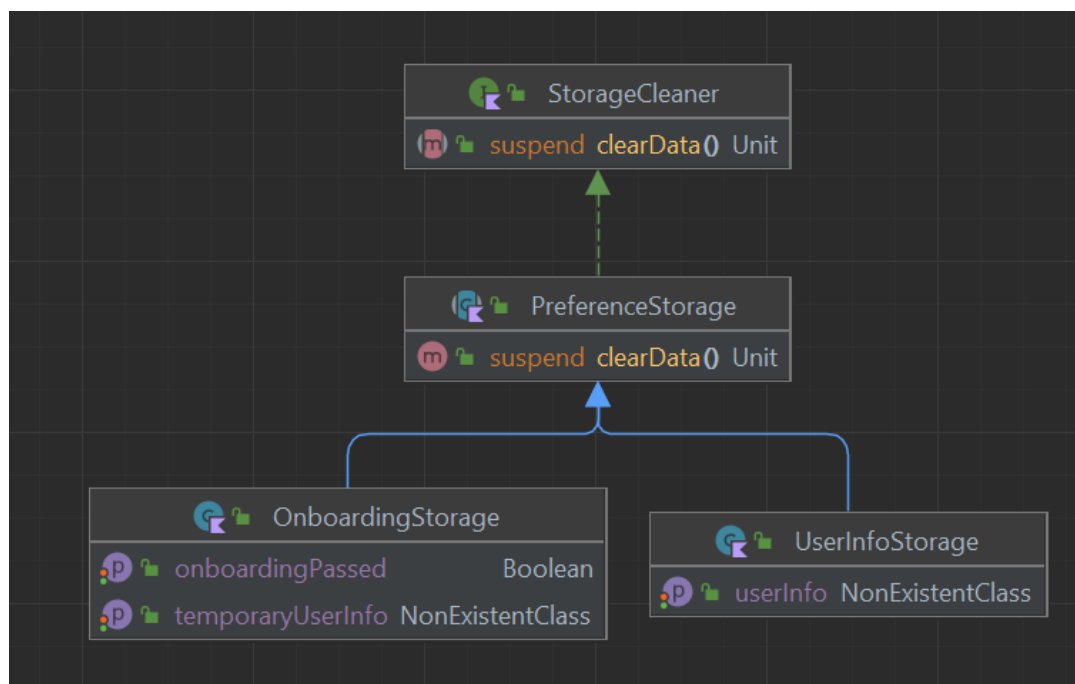


Рисунок 3.5 – Вигляд діаграми наслідувань класів для роботи із Shared Preferences

Абстрактний клас PreferenceStorage реалізує інтерфейс StorageCleaner та має наступну реалізацію:

```

abstract class PreferenceStorage: StorageCleaner {
    protected abstract val preferences: SharedPreferences

    @CallSuper
    override suspend fun clearData() {
        preferences.edit { clear() }
    }
}
  
```

3.2.3.2 Клас OnboardingStorage

Для збереження прогресу створення профілю користувача було розроблено сховище OnboardingStorage. Цей клас зберігає останні введені дані користувача: ім'я, вік, стать, вага, цілі щодо фізичної активності та поточна фізична активність. Це розраховано на випадки, коли користувач вимушений закінчити етап налагодження пізніше, продовжуючи із останнього моменту редагування.

3.2.3.3 Клас UserInfoStorage

Для роботи із даними профіля користувача було розроблено клас UserInfoStorage.

3.2.3.4 Структура таблиці MessageEntityDB

MessageEntityDB це структура, що описує таблицю для збереження історії спілкування між оператором та користувачем (рис. 3.6). Кожний запис зберігає текст повідомлення, час, автора повідомлення що є об'єктом перелічення MessageAuthor, та ідентифікатор повідомлення, що генерується автоматично при додаванні запису у базу даних. Таким чином зберігається історія спілкування користувача.

```

@Entity(tableName = "Messages")
data class MessageEntityDB(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "message_id") val messageID: Long,
    @ColumnInfo(name = "author") val author: MessageAuthor,
    @ColumnInfo(name = "time") val messageTimeStamp: String,
    @ColumnInfo(name = "text") val messageText: String,
)

```

Рисунок 3.6 – Структура MessageEntityDB

3.2.3.5 Структура таблиці MealsHistoryDB

MealsHistoryDB описує структуру таблиці, що зберігає об'єкт перелічення MealType, що є типом прийому їжі (сніданок, ланч, обід, вечеря, перекус), час прийому та об'єкт їжі класу YumHubMeal перетворений у JSON формат (рис. 3.7).

Головна мета – ведення історії споживаної їжі, задля ведення та перегляду статистики користувача.

```

@Entity(tableName = "MealsHistory")
data class MealsHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "date") val dateTimestampMs: Long,
    @ColumnInfo(name="meal_type") val mealType: MealType,
    @ColumnInfo(name="meal_info") val info: String, // JSON of YumHubMeal
)

```

Рисунок 3.7 – Структура MealsHistoryDB

3.2.3.6 Структура таблиці CategoryEatenHistoryDB

CategoryEatenHistoryDB – таблиця-лічильник, що допомагає відслідковувати кількість конкретно споживаної категорії їжі. Ці дані

використовуються для прогнозування та розрахунку рекомендацій користувача на основі його вподобань (рис. 3.8).

```
@Entity(tableName = "CategoryEatenHistory")
data class CategoryEatenHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "meal_category") val mealCategory: MealCategories,
    @ColumnInfo(name = "ate_count", defaultValue = "0") val categoryAteCount: Int,
)
```

Рисунок 3.8 – Структура CategoryEatenHistoryDB

3.2.3.7 Структура таблиці NutrientsHistoryDB

NutrientsHistoryDB – це спрощена таблиця MealsHistoryDB, що зберігає лише інформацію про харчову цінність продуктів що користувач спожив за певну дату. Це дозволяє відслідковувати прогрес нутрієнтів впродовж дня або відображати статистику у будь якому масштабі завдяки параметрам вибірки (рис. 3.9).

```
@Entity(tableName = "NutrientsHistory")
data class NutrientsHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "date") val dateTimeStampSec: Long,
    @ColumnInfo(name = "proteins", defaultValue = "0.0") val proteins: Double = 0.0,
    @ColumnInfo(name = "energy", defaultValue = "0.0") val energy: Double = 0.0,
    @ColumnInfo(name = "carbs", defaultValue = "0.0") val carbs: Double = 0.0,
    @ColumnInfo(name = "fat", defaultValue = "0.0") val fat: Double = 0.0,
    @ColumnInfo(name = "fiber", defaultValue = "0.0") val fiber: Double = 0.0,
    @ColumnInfo(name = "sugars", defaultValue = "0.0") val sugars: Double = 0.0,
    @ColumnInfo(name = "sugarsAdded", defaultValue = "0.0") val sugarsAdded: Double = 0.0,
    @ColumnInfo(name = "calcium", defaultValue = "0.0") val calcium: Double = 0.0,
)
```

Рисунок 3.9 – Структура NutrientsHistoryDB

3.2.3.8 Структура таблиці UserMeasurementsHistoryDB

Таблиця лічильник, що зберігає дані із зв'язкою із часом, але на відміну від CategoryEatenHistoryDB може зберігати будь які дані, якщо їх тип вказаний у

переліченні MeasurementType (рис. 3.10). За замовчуванням використовується два типи: Спожита вода (WaterBalance), та вага користувача. Завдяки зв'язці даних із часом можна будувати лінійні графіки та відслідковувати зміни користувача.

```
@Entity(tableName = "UserMeasurements")
data class UserMeasurementsHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "dayStartTimestampSec") val dayTimestampSec: Long,
    @ColumnInfo(name = "type") val measurementType: MeasurementTypes,
    @ColumnInfo(name = "value") val value: Float,
```

Рисунок 3.10 – Структура UserMeasurementsHistoryDB

3.2.4 Класи та функції утиліт

3.2.4.1 RestartableJob та RestartableJobArgs класи

RestartableJob та RestartableJobArgs це класи, що відповідають за запуск та зупинення об'єкту Job. Цей об'єкт створюється та повертається при створенні корутини – фоновій роботі. Для ініціалізації класи приймають об'єкт типу лямбда-функції що описує роботу яка буде виконуватись у фоні, а також, у випадку якщо це RestartableJobArgs приймає дженерік тип, що відповідає типу аргументу, який буде переданий при старті роботи.

RestartableJobArgs не є фінальним класом, це означає що від нього можна наслідуватись як від абстрактного класу та розширювати додатковим функціоналом. В даній ситуації RestartableJob наслідується від RestartableJobArgs, а у якості типу аргументу використовується встановлений тип «Nothing?», що по замовчанню є null-типом

Такі класи було створено задля вирішення проблем дублювання коду відміни створеної корутинами роботи та повторного їх запуску (рис. 3.11).

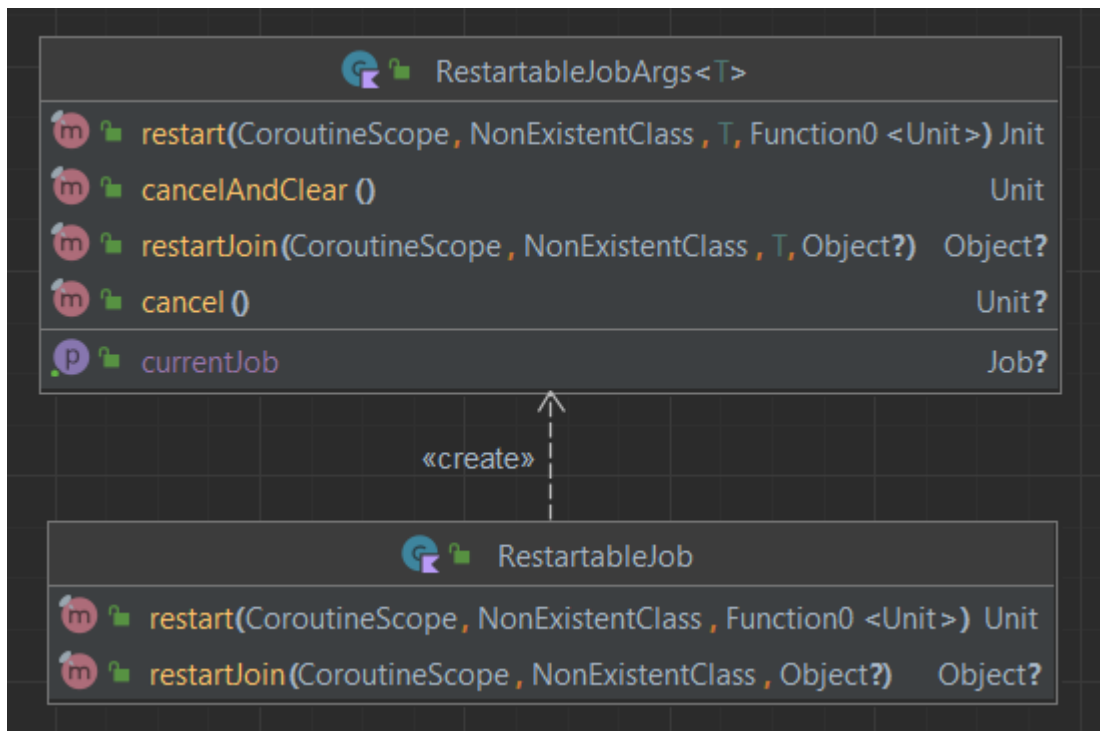


Рисунок 3.11 – Діаграма класу RestartableJob

Основні його методи це `cancel()` та `restart()`. Метод `Cancel` зупиняє роботу об'єкту `Job`, що виконувалась, а `restart()` повторно запускає роботу через виклик лямбда-функції, що передавалась при ініціалізації об'єкту.

Приклад використання такого класу-утиліти із `DashboardViewModel` класу:

...

```

private val dailyWaterBalanceCollectorJob = RestartableJob {
    userMeasurementsRepository.getMeasurementsForDateFlow(
        dayTimestampSec = DateTimeUtils.getNormalizedStartDateTodaySec(),
        types = listOf(MeasurementTypes.WATER_BALANCE)
    ).mapNotNull { it.firstOrNull()?.asWaterBalance() }.collect { measurement
  
```

->

```

        uiState.update {
            it.copy(dailyWaterBalance = measurement.balanceML)
        }
    }
}

```

Та приклад запуску такої роботи при ініціалізації об'єкту класу `DashboardViewModel`:

```

init {

    dailyWaterBalanceCollectorJob.restart(viewModelScope,Dispatchers.Default)

}

```

3.2.4.2 Клас `Str` як обгортка для базових строк

Для спрощеного використання та передавання строкових об'єктів було вирішено зробити клас `Str`. Це клас-обгортка має три підтипи, що допомагають створювати строки декількома способами, включаючи звичний для системи метод передачі строкових ресурсів.

```

sealed interface Str {
    class Res(@StringRes val id: Int, vararg val args: Any): Str
    class Text(val text: String): Str
    class Lambda(val block: () -> String): Str
    companion object {
        operator fun invoke(block: () -> String) = Lambda(block)
        operator fun invoke(text: String) = Text(text)
        operator fun invoke(@StringRes id: Int, vararg args: Any) = Res(id, *args)
    }
}

```

Клас `Str.Res` приймає у якості аргументів ідентифікатор строкового ресурсу системи, а також список аргументів, що будуть підставлені у передану строку.

Клас `Str.Text` приймає у якості аргументу звичайну строку типу `String`.

Клас `Str.Lambda` приймає у якості аргументу лямбда-функцію, що при виконанні має повернути строку.

Для перетворення об'єкту назад у звичний вигляд типу `String` було розроблено функцію-розширення «`Str.get(Context)`», що за допомогою виразу «`when`» перебирає тип об'єкту і обробляє його згідно із логікою:

```
fun Str.get(context: Context): String = when(this) {
    is Str.Res -> context.getString(this.id, *this.args)
    is Str.Text -> this.text
    is Str.Lambda -> this.block.invoke()
}
```

3.2.4.3 Функції-розширення

Функція розрахування найближчого наступного числа, що ділиться на інше число передане у якості аргументу.

```
/** Returns the next value dividable by divBy;
 *
 * (999L).getNextDividable(5); = 1000
 * (1000L).getNextDividable(5); = 1000
 * (1001L).getNextDividable(5); = 1005
 * (1004L).getNextDividable(5); = 1005
 * (-999L).getNextDividable(5); = -995
 * (-995L).getNextDividable(5); = -995
```

```

* (-1L).getNextDividable(5); = 0
* (1L).getNextDividable(5); = 5
* (0L).getNextDividable(5); = 0
* */

fun Long.getNextDividable(divBy: Long): Long {
    val tail = abs(this % divBy)
    return if (this < 0) {
        this + tail
    } else {
        this + (divBy - tail)
    }
}

```

Функція безперервного анімованого збільшення та зменшення елементу інтерфейсу для Composable об'єктів.

```

fun Modifier.scalingAnimationModifier(
    scaleFrom: Float,
    scaleTo: Float,
    scaleAnimationDurationMs: Int,
) = composed {
    val initial = remember { Animatable(scaleFrom) }
    LaunchedEffect(Unit) {
        initial.animateTo(scaleTo, infiniteRepeatable(
            animation = tween(
                durationMillis = scaleAnimationDurationMs,
                easing = LinearEasing,
            ), RepeatMode.Reverse),
        )
    }
}

```

```

    }
    val scale by initial.asState()
    Modifier.scale(scale)
}

```

Функція-розширення інтерфейсу `PreferencesValue<T>` для оновлення останнього збереженого значення.

```

fun <T> PreferencesValue<T>.update(transform: (T) -> T) =
    set(transform(get()))

```

3.3 Огляд реалізації рекомендаційної системи та аналізу даних

3.3.1 Дані та основні характеристики, що аналізуються

Дані для розробки рекомендаційної системи YumHub були взяті з відкритого джерела[14], яке надає інформацію про різноманітні продукти та їхні поживні характеристики.

3.3.2 Базові розрахування щоденних рекомендацій

Для розрахування базових щоденних рекомендацій щодо харчування та водного балансу користувача використовуються дані користувача про його фізичну активність, цілі, а також поточна вага, зріст, стать. Ці дані також дають змогу розраховувати рівень основного метаболізму.

3.3.2.1 Розрахунок рівня основного метаболізму

Для розрахунку базового основного метаболізму (Basal Metabolic Rate) було використано рівняння[15] Гарріса-Бенедикта, або, як його ще називають, принцип Гарріса-Бенедикта.

Формули для розрахунку наведені нижче (3.1, 3.2):

$$BMR_{(жін.)} = 447.593 + (9.247 * weight) + (3.098 * height) - (4.330 * age) \quad (3.1)$$

$$BMR_{(чол.)} = 88.362 + (13.397 * weight) + (4.799 * height) - (5.677 * age) \quad (3.2)$$

де *weight* – вага в кілограмах;

height – зріст у сантиметрах;

age – вік у роках.

3.3.2.2 Водний баланс

Для розрахунку рекомендації щодо споживчого балансу води була застосована спрощена формула апроксимації водного балансу тіла[16] та рекомендації з водопиття, засновану на основних фізіологічних параметрах та рівні фізичної активності. Вона не має конкретної назви, оскільки це не є стандартом або науковим методом.

3.3.3 Методи розрахунку рекомендацій. Класи PointwisePrognosis, AbsoluteDifferenceSimilarityAnlysis та CosineSimilarityAnalysis

У контексті рекомендаційної системи, було розроблено загальний метод контент-базового розрахунку та два методи аналізу схожості продуктів між

собою: метод косинусної подібності та метод абсолютної різниці з нормалізацією до діапазону.

3.3.3.1 Контент-базовий аналіз та розрахунок ваги. Клас PointwisePrognosis

Контент-базовий аналіз (Content-Based Analysis) - це метод рекомендаційних систем, де рекомендації надаються на основі характеристик (контенту) самого об'єкта і враховують індивідуальні вподобання користувача. У випадку рекомендацій харчування це включає аналіз харчового складу продуктів, попередніх вподобань користувача, категорій їжі тощо. Контент-базований аналіз може бути використаний, коли користувачам потрібні рекомендації на основі конкретних характеристик об'єктів.

Для такого аналізу було використано point-wise підхід оцінки кожного продукту, де результатом розрахунків є вага на основі харчового складу, фізичної активності та вподобань користувача, цілей та іншої статистики.

Алгоритм розрахунку ваги об'єкту (продукту) представлено у методі `calculateWeightsForProducts(...)` класу-об'єкту `PointWisePrognosis`:

```
fun calculateWeightsForProducts(
    products: List<YumHubMeal>,
    userInfo: UserInfo,
    eatenCategories: Map<MealCategories, Int>,
): List<Pair<YumHubMeal, Double>> {
    return products.map { product ->
        val favoriteWeightToAdd = product.categoriesTags.count {
            userInfo.favoriteCategories.contains(it)
        } * favoriteWeight
        val exactTypesWeights = product.categoriesTags.filter {
            it.isSpecifiedType() }.count {
```

```

        userInfo.favoriteCategories.contains(it)
    } * mainDishCategoryWeight
    val eatenWeightToAdd = product.categoriesTags.sumOf {
        calculateEatenWeight(eatenCategories, it)
    }
    val goalScore = calculateGoalScore(product, userInfo.fitnessGoal)
    val palScore = calculatePalScore(product, userInfo.pal)

    // Загальна вага продукту
    val totalScore = (goalScore * goalWeight) + (palScore * palWeight) +
        favoriteWeightToAdd + exactTypesWeights + eatenWeightToAdd

    product to totalScore
}
}

```

Особливістю цього алгоритму є врахування варіативності множників ваги в залежності від цілій користувача та його поточної фізичної активності. Такі розрахунки були логічно розподілені між функціями `calculateGoalScore(...)` та `calculatePalScore(...)`. Метод `calculateGoalScore(...)` розраховує вагу нутрієнтів та калорій, а потім помножує на додатковий коефіцієнт, в залежності від цілі користувача, наприклад:

```

private val proteinWeight = mapOf(
    FitnessGoal.LoseWeight to 1.0,
    FitnessGoal.BuildMuscle to 1.2,
    FitnessGoal.MaintainWeight to 1.0
)

```

Розрахунки ваг продуктів на основі цілей користувача у методі `calculateGoalScore(...)`:

```
private fun calculateGoalScore(product: YumHubMeal, userGoal:
FitnessGoal): Double {
    val fatGoalWeight = fatWeight[userGoal] ?: 1.0
    val caloriesGoalWeight = caloriesWeight[userGoal] ?: 1.0
    val proteinGoalWeight = proteinWeight[userGoal] ?: 1.0
    val carbsGoalWeight = carbsWeight[userGoal] ?: 1.0

    return (product.withOneServing(NutritionEnum.Fat) * fatGoalWeight) +
        (product.calories * caloriesGoalWeight) +
        (product.withOneServing(NutritionEnum.Protein) *
proteinGoalWeight) +
        (product.withOneServing(NutritionEnum.Carbs) * carbsGoalWeight)
}
```

Також в розрахунках використовується додатковий коефіцієнт в залежності від поточних фізичних навантажень, наприклад, додатковий коефіцієнт для протеїнів продукту:

```
private val proteinPalWeight = mapOf(
    UserPAL.Sedentary to 0.8,
    UserPAL.LightlyActive to 1.0,
    UserPAL.ModeratelyActive to 1.2,
    UserPAL.VeryActive to 1.5,
    UserPAL.ExtraActive to 1.5
)
```

Розрахунок ваг нутрієнтів на основі рівня фізичної активності користувача:

```
private fun calculatePalScore(product: YumHubMeal, userPal: UserPAL):
Double {
    val fatScore = product.nutrients.find { it.attr == NutritionEnum.Fat }?.value
    ?: 0.0

    val caloriesScore = product.calories

    val fatPalWeight = fatPalWeight[userPal] ?: 1.0
    val caloriesPalWeight = caloriesPalWeight[userPal] ?: 1.0
    val proteinPalWeight = proteinPalWeight[userPal] ?: 1.0
    val carbsPalWeight = carbsPalWeight[userPal] ?: 1.0

    return (fatScore * fatPalWeight) +
        (caloriesScore * caloriesPalWeight) +
        (product.protein.valueOrZero * proteinPalWeight) +
        (product.carbs.valueOrZero * carbsPalWeight)
}
```

Такий підхід дозволяє системі надавати перевагу більш релевантним продуктам для користувача.

3.3.3.2 Методи косинусної подібності та матриці схожості

Косинусна подібність - це метод, який використовується для визначення ступеня схожості між різними об'єктами в рекомендаційній системі. У нашому випадку, цей метод знаходить застосування при оцінці схожості між продуктами та врахуванні виборів користувача для надання персоналізованих рекомендацій.

Косинусна подібність ґрунтується на теорії векторів, де об'єкти (продукти) представлені векторами у просторі. Схожість між двома векторами обчислюється як косинус кута між ними. Чим більше кут між векторами, тим менша схожість, і навпаки.

У контексті рекомендаційної системи косинусна подібність використовується для оцінки схожості між продуктами на основі їхніх характеристик та властивостей. Наприклад, якщо користувач вибирає продукти, які мають схожі характеристики або життєві стилі, система використовує косинусну подібність для визначення ступеня схожості між цими продуктами.

Для розрахунку матриці схожості було розроблено клас `CosineSimilarityAnalysis`, що вміщує усі потрібні методи для створення матриці, розрахунку ваг на основі характеристик нутрієнтів та типі продукту.

Вхідною точкою розрахунку матриці схожості є метод `createSimilarityMatrixBetweenMeals(...)`:

```
fun createSimilarityMatrixBetweenMeals(products: List<YumHubMeal>):
    Array<DoubleArray> {
        val matrixSize = products.size

        val similarityMatrix = Array(matrixSize) { i ->
            DoubleArray(matrixSize) { j ->
                if (i == j) {
                    1.0 // Similarity between the same item is always 1
                } else {
                    calculateOverallSimilarity(
                        product1 = products[i],
                        product2 = products[j]
                    ) // Calculating similarity between two items
                }
            }
        }
    }
```

```

        }
    }
}

return similarityMatrix
}

```

Розрахунок схожості відбувається за допомогою методу `calculateOverallSimilarity`:

```

fun calculateOverallSimilarity(
    product1: YumHubMeal,
    product2: YumHubMeal
): Double {
    return calculateOverallSimilarity(
        product1 = product1,
        product2 = product2,
        nutrientWeight = nutrientWeight,
        categoryWeight = categoryWeight
    )
}

```

```

fun calculateOverallSimilarity(
    product1: YumHubMeal,
    product2: YumHubMeal,
    nutrientWeight: Double,
    categoryWeight: Double
): Double {
    val nutrientSimilarity = calculateCosineSimilarity(product1, product2)
    val categorySimilarity = calculateCategorySimilarity(product1, product2)

```

```

        return (nutrientWeight * nutrientSimilarity + categoryWeight *
categorySimilarity) / (nutrientWeight + categoryWeight)
    }

```

Наступним кроком є розрахунок косинуса подібності на основі нутрієнтів продукту методом `calculateCosineSimilarity` та розрахунок схожості за категоріями продуктів методом `calculateCategorySimilarity`. Нижче наведені реалізації цих методів та алгоритмів:

```

private fun calculateCosineSimilarity(
    product1: YumHubMeal,
    product2: YumHubMeal
): Double {
    val nutrients1 = product1.nutrients.filter { attr ->
        mainNutrientsToCompare.contains {
            it.attrID == attr.attr.attrID
        }
    }

    val nutrients2 = product2.nutrients.filter { attr ->
        mainNutrientsToCompare.contains {
            it.attrID == attr.attr.attrID
        }
    }

    val dotProduct = nutrients1.mapNotNull { nutrient1 ->
        nutrients2.firstOrNull { it.attr.attrID == nutrient1.attr.attrID }?.let {
            nutrient2 ->
                product1.withOneServing(nutrient1.attr) *
                product2.withOneServing(nutrient2.attr)
        }
    }

```

```
}.sum()
```

```
val magnitude1 = calculateMagnitude(product1, nutrients1)
```

```
val magnitude2 = calculateMagnitude(product2, nutrients2)
```

```
return if (magnitude1 > 0 && magnitude2 > 0) {
```

```
    dotProduct / (magnitude1 * magnitude2)
```

```
} else {
```

```
    0.0
```

```
}
```

```
}
```

```
private fun calculateCategorySimilarity(product1: YumHubMeal, product2:
YumHubMeal): Double {
```

```
    val categories1 = product1.categoriesTags.toSet()
```

```
    val categories2 = product2.categoriesTags.toSet()
```

```
    val intersectionSize = categories1.intersect(categories2).size.toDouble()
```

```
    val unionSize = categories1.union(categories2).size.toDouble()
```

```
    return if (unionSize > 0) {
```

```
        intersectionSize / unionSize
```

```
    } else {
```

```
        0.0
```

```
    }
```

```
}
```

```
// Функція розрахунку довжини векторів характеристик нутрієнтів
```

```
private fun calculateMagnitude(product: YumHubMeal, nutrients:
List<NutrientsItem>): Double {
```

```

return nutrients.map {
    product.withOneServing(it.attr)
}.toDoubleArray().let { vector ->
    sqrt(vector.fold(0.0) { acc, value -> acc + value * value })
}
}

```

Проте, на основі аналізів та порівняння розрахунків (рис. 3.12), виявилось, що такий підхід не дає чітких очікуваних результатів.

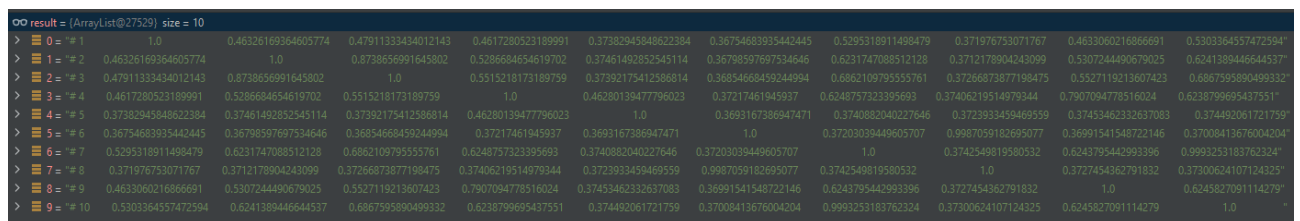


Рисунок 3.12 – Розрахунки методом косинусної подібності перших 10 продуктів

Для цього було порівняно два продукти, де схожість між ними, як було розраховано, була 0.99932531, тобто майже максимально схожі, проте це не так (табл. 3.1).

Таблиця 3.1 – Порівняння характеристик двох продуктів

№ Продукту	Протеїни	Жири	Вуглеводи	Цукри
Продукт №1	24	35	73	13
Продукт №2	12	21	30	6

Як зазначено в таблиці 3.1, різниця між продуктами майже вдвоє. Це ставить під сумнів використання системи косинусної подібності продуктів, тож було спроектовано та розроблено новий метод – метод розрахунку схожості на основі порівнянні абсолютної різниці характеристик продуктів з нормалізацією до діапазону. Такий підхід розглянуто у пункті 3.3.3.

3.3.3.3 Розрахунок абсолютної різниці та аналіз схожості

Абсолютна різниця – це простий метод вимірювання відстані, який показує різницю між двома значеннями.

Нормалізація до діапазону – це метод нормалізації даних, який перетворює усі дані значення на діапазон від 0 до 1, де 0 відповідає значенню максимальної різниці між об'єктами, а 1 – нульовою різницею.

У контексті рекомендаційної системи розрахунок абсолютної різниці використовується для вимірювання відстані між продуктами на основі розрахунку абсолютної різниці кожної з характеристик.

Робота такого підходу відбувається наступним чином: для кожної характеристики створюється своя матриця абсолютної різниці між всіма продуктами:

```
fun absoluteDifferenceMatrix(
    products: List<YumHubMeal>,
    nutrient: NutritionAttr
): Array<DoubleArray> {
    val matrixSize = products.size
    return Array(matrixSize) { i ->
        DoubleArray(matrixSize) { j ->
            if (i == j) {
                0.0
            } else {
                Math.abs(
                    products[i].withServings(1.0, nutrient) -
                    products[j].withServings(1.0, nutrient)
                )
            }
        }
    }
}
```

```
}
```

Далі матриця абсолютної різниці потребує трансформації у матрицю подібності, це досягається за рахунок її нормалізації, де кожне значення ділиться на число із максимальною різницею, таким чином максимальна різниця постає нульовим значенням подібності, а всі інші у межах від 0 до 1:

```
private fun toSimilarityMatrix(
    absoluteDifferenceMatrix: Array<DoubleArray>
): Array<DoubleArray> {
    val maxDifference = absoluteDifferenceMatrix.flatMap { it.toList()
}.maxOrNull()?: 1.0
    return Array(absoluteDifferenceMatrix.size) { i ->
        DoubleArray(absoluteDifferenceMatrix[i].size) { j ->
            1.0f - absoluteDifferenceMatrix[i][j].div(maxDifference)
        }
    }
}
```

Після цього настає завершальний етап - поєднання матриць схожості між собою. Це відбувається за допомогою методу `aggregateSimilarityMatrices` у класі `AbsoluteDifferenceSimilarityAnlysis`:

```
fun aggregateSimilarityMatrices(
    productsSize: Int,
    matrices: List<Array<DoubleArray>>
): Array<DoubleArray> {
    return Array(productsSize) { i ->
        DoubleArray(productsSize) { j ->
            matrices.map { matrix ->
                val value = matrix[i][j]
```

```

        if (value.isNaN()) 1.0 else value
    }.let { it.sum().div(it.size) }
}
}
}

```

Усі ці матриці сумуються між собою і кожне значення ділиться на кількість характеристик, що аналізувались.

На виході маємо такі дані порівняння (рис. 3.13):

0 = #1	1.0	0.5830258035474148	0.7301179002985952	0.6245909635296399	0.639002231093407	0.5311059444623868	0.5984590149806678	0.6286431574227703	0.7599981628072109	0.7340155909211362
1 = #2	0.5830258035474148	1.0	0.7076882650059254	0.5952757063077939	0.4557597314539141	0.3960115929710421	0.5743737565516812	0.44083562763685175	0.48462603353808825	0.6354672665851748
2 = #3	0.7301179002985952	0.7076882650059254	1.0	0.6108432552533568	0.42791642768829874	0.35520532624246378	0.6172884200556612	0.45274233920284724	0.49011606310580635	0.629426288400955
3 = #4	0.6245909635296399	0.5952757063077939	0.6108432552533568	1.0	0.6420731724329419	0.5341768858019217	0.9222585389033994	0.6724117731809098	0.633439474517116	0.7935830331455982
4 = #5	0.639002231093407	0.4557597314539141	0.42791642768829874	0.6420731724329419	1.0	0.7536036674894695	0.5643317113363413	0.8170564825419587	0.8604855497676777	0.7984901392873439
5 = #6	0.5311059444623868	0.3960115929710421	0.35520532624246378	0.5341768858019217	0.7536036674894695	1.0	0.5045835728534693	0.8220528993207682	0.7064106864889883	0.7202234822859532
6 = #7	0.5984590149806678	0.5743737565516812	0.6172884200556612	0.9222585389033994	0.5643317113363413	0.5045835728534693	1.0	0.6521207858138527	0.5556980134205155	0.7158415720489973
7 = #8	0.6286431574227703	0.44083562763685175	0.45274233920284724	0.6724117731809098	0.8170564825419587	0.8220528993207682	0.6521207858138527	1.0	0.7489332689833379	0.7743498425331584
8 = #9	0.7599981628072109	0.48462603353808825	0.49011606310580635	0.633439474517116	0.8604855497676777	0.7064106864889883	0.5556980134205155	0.7489332689833379	1.0	0.834586349144904
9 = #10	0.7340155909211362	0.6354672665851748	0.629426288400955	0.7935830331455982	0.7984901392873439	0.7202234822859532	0.7158415720489973	0.7743498425331584	0.834586349144904	1.0

Рисунок 3.13 – Результати розрахунків схожості методом абсолютної різниці перших 10 продуктів

Порівнюючи ті самі продукти, що розглядались у розділі 3.3.3.2, можна зазначити, що розрахунки наблизились до очікуваних результатів, а схожість між продуктами розглянутих у таблиці 3.1 мають значення 0.71584, на відміну від попереднього у вигляді 0.99932531.

3.3.4 Висновки та перспективи системи

У розділі 3.3 було розглянуто алгоритми та систему рекомендацій, кожну її ступінь обробки даних. Також було розглянуто результати розрахунків та робота над помилками у вигляді розробки нового методу аналізу – обробка вподобань за допомогою створення матриці абсолютної різниці із нормалізацією даних.

Наступним кроком для підвищення ефективності рекомендаційної системи є більш глибоке дослідження та аналіз даних користувача, надання більших пріоритетів тим нутрієнтам, що потрібні користувачу, а також налагодження рекомендаційної системи до годин доби та типів прийому їжі.

Ще одним важливим кроком є збільшення бази даних продуктів, збір статистики та відгуків користувачів для покращення алгоритмів системи. Підключення штучного інтелекту для обчислень великої кількості даних, тренування моделі та запровадження у застосунок.

3.4 Огляд візуального інтерфейсу застосунку

3.4.1 Основні рішення щодо дизайну за стосунку

Основним підходом до розробки інтерфейсу було використано «Material You». Це вважається найсучаснішим підходом в розробці Android застосунків.

Суть Material You 3 в тому, щоб використовувати стиль елементів подібний до карток. Додається тінь, що створює ілюзію плаваючих елементів на екрані. На поточний момент вже доступний новий підхід – «Material 3». Його ідея полягає в тому, щоб позбутися заздалегідь встановлених кольорів додатку, та використовувати системні кольори або такі, що залежать від улюбленої кольорової гами користувача.

Усі візуальні компоненти застосунку були власноруч намальовані через особливості Jetpack Compose, що надає унікальності продукту та її компонентів.

Ще однією деталлю є використання анімованого фону додатку. Це приємний напівпрозорий зелений колір із чотирма анімованими кругами, що збільшуються та зменшуються у розмірах із часом. Це майже непомітно, але дає відчуття приємної роботи із інтерфейсом.

3.4.2 Екран реєстрації користувача (OnboardingScreen)

Екран реєстрації є стовпчастим компонентом, що складається із текстом-привітанням та формою для заповнення даних користувача. Ці дані є основою для аналізу та розрахунку рекомендацій. Форма складається із 4 блоків.

Усі блоки містять заголовок та опис щодо цього блоку для розуміння користувачеві для чого ці дані використовуються.

Перший блок містить декілька полів вводу персональної інформації користувача, його ім'я, віку, статі, ваги у кілограмах та зросту у сантиметрах. Ці дані є обов'язкові до заповнення, та підсвічуються червоним у разі якщо користувач залишив його пустим (рис. 3.14).

The image shows a mobile app registration screen for 'YumHub'. At the top, a green rounded rectangle contains the text 'Welcome to YumHub'. Below this is a larger green rounded rectangle titled 'Step 1: Personal information' with a subtitle '*Your personal information is used to give You the best recommendations'. The form contains several input fields: 'Name' and 'Age' (the 'Age' field has a red border), 'Male' and 'Female' (radio button options), 'Weight (kg)', and 'Height (cm)'.

Рисунок 3.14 – Форма вводу персональної інформації

Другий блок (рис. 3.15) пропонує користувачу обрати його поточну фізичну активність шляхом вибору одну з 5 опцій: Активність відсутня, легка активність (заняття спортом 1-3 дні на тиждень), помірна активність (заняття

спортом 3-5 днів на тиждень), велика активність (6-7 днів на тиждень), та максимальна завантаженість (фіз. активність кожний день та робота пов'язана із фізичною активністю).

Рисунок 3.15 – Форма для вибору поточної фізичної активності

Третій блок пропонує обрати користувачу його ціль для фізичного здоров'я. Запропоновано 3 варіанти: Схуднення, підтримка ваги і набір мускульної ваги (рис. 3.16).

Рисунок 3.16 – Форма №3, вибір фізичних цілей

Четвертий блок (рис. 3.17) є останнім, але не менш важливим. Користувачу пропонується обрати його улюблені категорії їжі із запропонованих. Ці категорії розподілені на підкатегорії, та описують походження кухні (італійська, мексиканська, азійська тощо), спосіб

приготування (варена, жарена, готова до вживання, спечена тощо), на смак (солодка, гостра тощо) та тип продуктів (м'ясо, морепродукти, курятина, піца, салат, суп тощо).

Кожна підкатегорія розділена лінією, а елементи є частиною списку, що може бути прокручений по горизонталі і мають стан або обране, або ні.

Step 4: Favourite categories
Let us know your favourite food categories

Cuisine

Italian Mexican Asian Indian

Cook Type

Boiled Fried ReadyToEat Grilled

Flavor

Spicy Sweet Savory Sour

Food Types

Meat Seafood Chicken Beef

Save & continue

Рисунок 3.17 – Блок №4, вибір вподобань щодо продуктів

Після того, як користувач заповнить усі форми, буде активована кнопка «Save & Continue», що збереже користувацькі дані та перенесе користувача на головний екран застосунку.

3.4.3 Головний екран застосунку (DashboardScreen)

Головний екран застосунку відображає актуальні дані щодо спожитих калорій, випитої за день води а також головних нутрієнтів: вуглеводи, протеїни та жири (рис. 3.18).

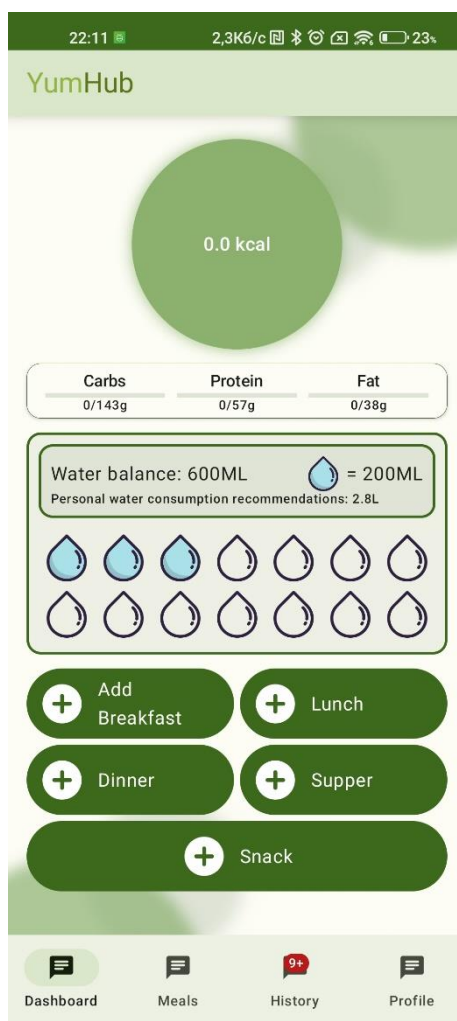


Рисунок 3.18 – Вигляд головного екрану

Компонент із відображенням водного балансу є інтерактивним і при натисканні на краплю води додає 200мл до добової статистики спожитої води (рис. 3.19). Його додаткова ідея це відображення особистих рекомендацій щодо споживання води на основі зазначених даних про фізичну активність та фізичні дані користувача при реєстрації.

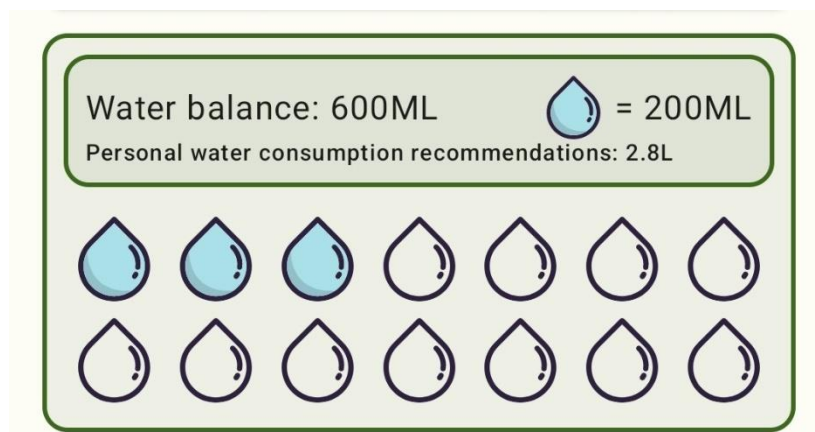


Рисунок 3.19 – Інтерактивна панель водного балансу

3.4.4 Екран вибору та додавання продуктів

Для додавання споживаних продуктів використовуються відповідні кнопки із текстом-приставкою «Add» на головному екрані (рис. 3.20).

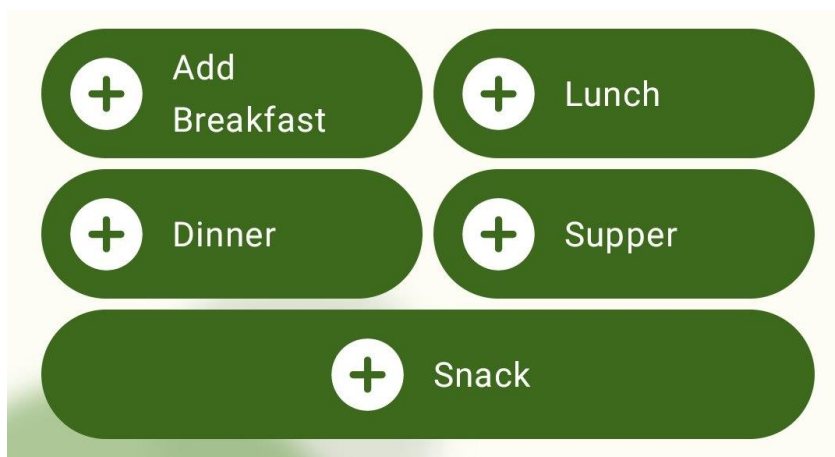


Рисунок 3.20 – Елементи для додавання споживаних страв

Обираючи тип прийому їжі, користувача перенаправляє до вікна вибору продукту.

При натисканні на продукт зі списку користувач має змогу ознайомитись із харчовою поживністю розрахованої на одну та максимальну кількість порцій (рис. 3.21).



Рисунок 3.21 – Перегляд харчової поживності продукту

При натисканні на кнопку із текстом «Add as ...» споживаний продукт додається до історії харчування користувача та відбувається оновлення статистики на головному екрані.

3.4.5 Екран із особистими рекомендаціями для користувача

При ініціалізації компоненту екрану (рис. 3.22) створюється об'єкт вьюмоделі `MealsViewModel` який у свою чергу при ініціалізації запускає процес оновлення рекомендацій на основі останніх даних про спожиту їжу та інформації з профілю. За розрахунок рекомендацій відповідає об'єкт типу `RestartableJob`, що описує виклик методу `calculateWeightsForProducts` класу `PointwisePrognosis` та оновлення даних реактивним шляхом:

```
private val recommendationsLoaderJob = RestartableJob {
```

```

PointWisePrognosis.calculateWeightsForProducts(
    products = dishes,
    userInfo = userInfoStorage.userInfo.get()!!,
    eatenCategories
categoriesEatenHistoryDao.getAllCategoriesAteCount().associate {
    it.mealCategory to it.categoryAteCount
},
).sortedByDescending { it.second }.let {
    recommendations.value = it.map { it.first }
}
}

```

Рекомендації відображаються реактивно, використовуючи патерн «Спостерігач», одразу після розрахунків ваг кожного продукту. Результати сортуються відповідно до значення розрахованої ваги PointWisePrognosis класом, що був розглянутий у розділі 3.3.3.1, та пропонуються користувачеві від найбільш релевантного до найменшого.



Рисунок 3.22 – Екран із відображенням рекомендацій та продуктів по категоріям

Також інтерфейс дає змогу користувачеві продивитись усі продукти відфільтровані по категоріях: найменший час готування, найменша кількість інгредієнтів, невелика собівартість, велика собівартість, мала кількість жирів, фастфуд та без фільтрів (рис. 3.23).

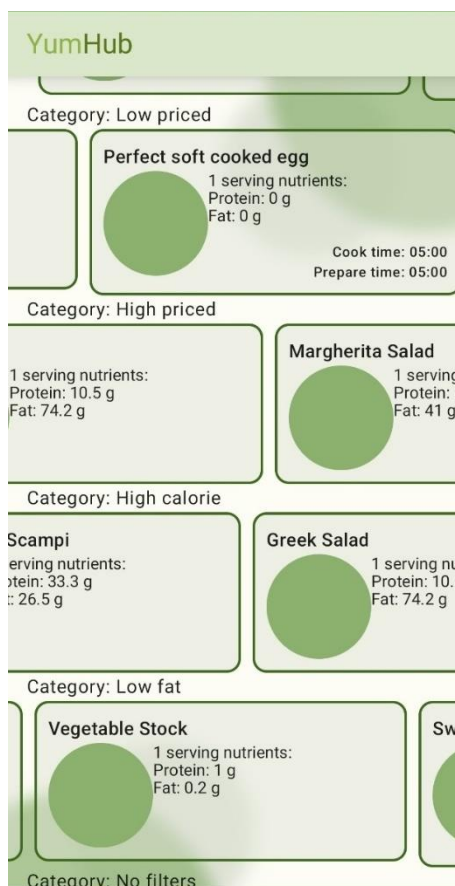


Рисунок 3.23 – Перегляд продуктів за категоріями

Кожний компонент продукту описує його назву, основну харчову поживність для однієї порції, а також час приготування та час очікування, якщо така інформація існує для певного продукту.

При натисканні на елемент продукту навігаційний компонент перенаправляє користувача на екран із детальною інформацією про страву, а також показує схожі продукти, використовуючи розроблену систему рекомендацій, що була розглянута у розділі 3.3.3.3.

3.4.6 Екран детальної інформації обраної страви та рекомендацій

Компонент MealInfoScreen відповідає за відображення детальної інформації про страву. Він вміщує декілька основних блоків даних: назву страви, харчову поживність на одну та максимальну кількість порцій, а також інформацію про приготування цієї страви, описуючи потрібні інгредієнти та рецепт (рис. 3.24). Це дає змогу користувачеві ознайомитись із різноманітними рецептами у режимі офлайн, а також продивитись схожі страви, завдяки рекомендаційній системі та її алгоритмів порівняння.

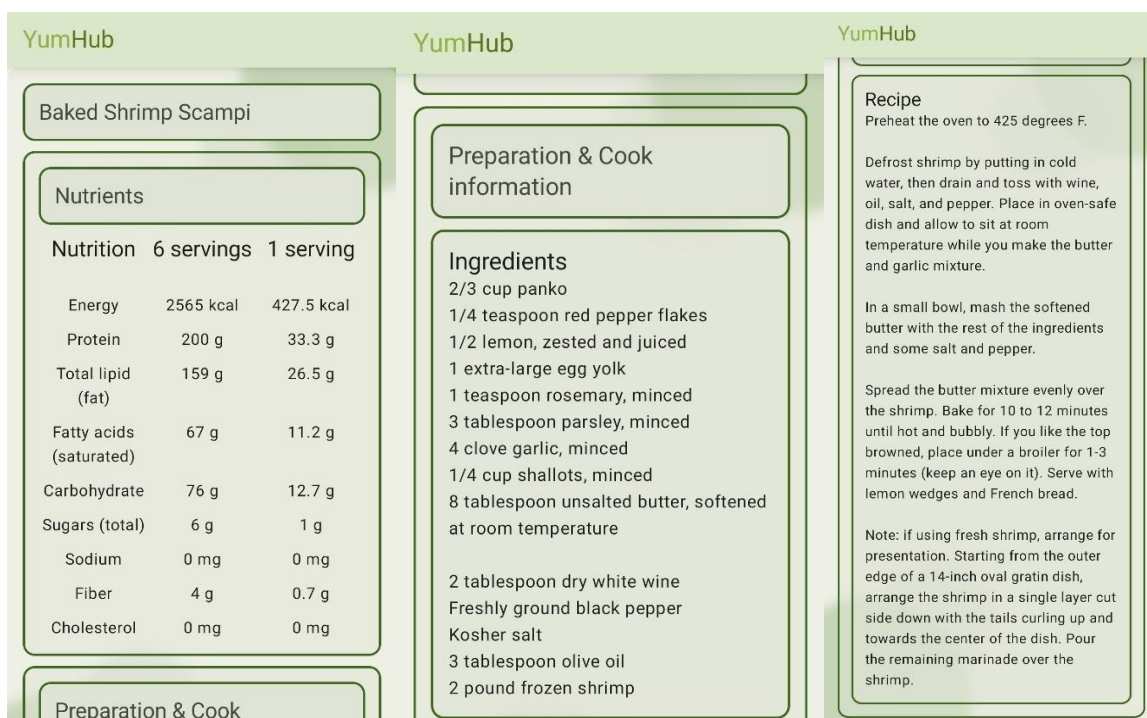


Рисунок 3.24 – Екран із детальною інформацією про обрану страву

В самому низу компоненту MealInfoScreen було поміщено блок із рекомендаціями, що відсортовані за критерієм максимальної подібності (рис. 3.25).

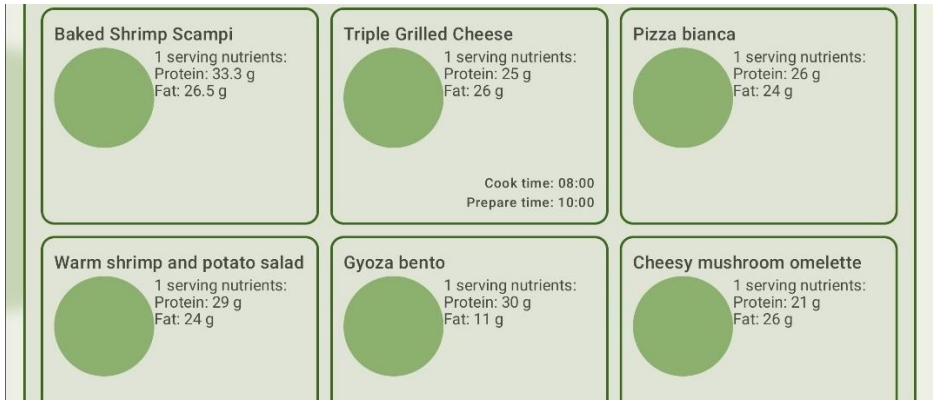


Рисунок 3.25 – Блок рекомендацій страв, що схожі на обрану страву

Для розрахунку схожості страв використовується алгоритми описані у класі AbsoluteDifferenceSimilarityAnlysis, що були розглянуті у розділі 3.3.3.3.

3.4.7 Екран історії споживаних продуктів та води

Інтерфейс застосунку дозволяє переглядати історію спожитих продуктів, їх нутрієнтів, а також спожитої води за конкретну дату. Для вибору дати було застосовано компонент «Календар» з бібліотеки Material3 за основу, та модифіковано для можливості вибору діапазону дат шляхом додавання компоненту «Чекбокс» із написом «Використовувати діапазон для вибору» (рис. 3.26).

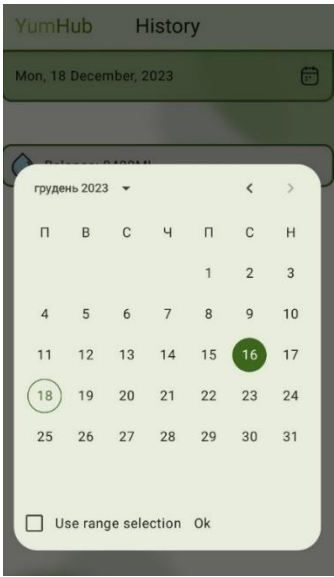


Рисунок 3.26 – Компонент «Календар» для перегляду історії за обрану дату

Після підтвердження вибору – інтерфейс створює моделі графіків та передає їх до відповідних компонентів для відображення цих даних. Користувач має змогу змінювати масштаб графіків, передивляється дані для кожного обраного дня. Доступні такі дані для перегляду: спожита вода, білки, жири та вуглеводи (рис. 3.27).

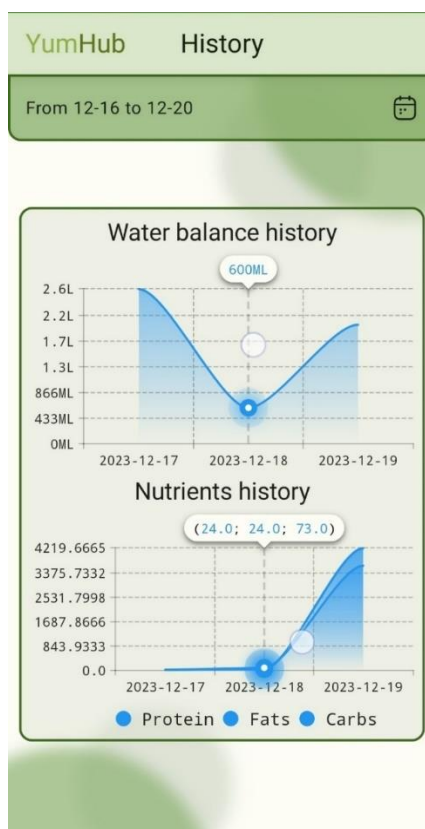


Рисунок 3.27 – Відображення історії харчування та споживаної води у діапазонному виборі дат

3.4.8 Огляд форми профіля (UserProfileScreen)

Компонент UserProfileScreen відповідає за перегляд даних, що користувач ввів під час реєстрації. Компонент має свій особистий інтерфейсний контроллер, що є водночас і вьюмоделлю - UserProfileViewModel, що відповідає за усю логіку відображення та оновлення даних користувача.

Також було додано елемент графіку, що відображає дату та вагу користувача для легкого відслідковування змін (рис. 3.28).

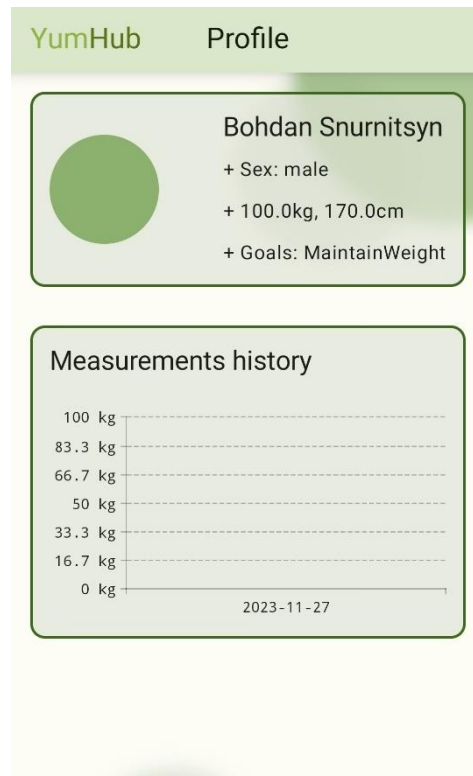


Рисунок 3.28 – Відображення екрану профілю

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУНКУ

У цьому розділі розглянуто процес роботи додатку, тестування на різних пристроях та версіях Android.

4.1 Процес роботи застосунку

Користувач запускає застосунок нажимаючи на іконку «YumHub».

Система запускає процес ініціалізації компонентів, що потрібні для роботи додатку, та повертає вікно MainActivity.

Користувачу відображається анімований екран завантаження.

Після закінчення анімації перед користувачем з'являється головний екран «Dashboard», для нових користувачів – вікно Реєстрації.

У разі реєстрації, користувачу потрібно вписати особисті фізичні дані у відповідні елементи вводу тексту: Ім'я, вік, стать, вага та зріст. У наступному кроці потрібно обрати поточний рівень фізичної активності серед запропонованих. У третьому етапі користувач має обрати цілі: схуднення, підтримка ваги, нарощування м'язів. У фінальному етапі – обрати улюблені категорії страв, спосіб приготування, та походження страв (кухня).

При переході на головний екран користувачу відображається Dashboard з його вживаними калоріями за день, кількістю випитої води, а також пункти для заповнення вживаної їжі на сніданок, обід, вечерю тощо.

У вікні «Meals» користувач має змогу переглядати рекомендації та перелік страв.

Для того, щоб переглянути історію вживаної їжі із можливістю вибору дня із календаря, користувач може перейти у вкладку «History». Цей календар користувач має змогу вести як сам, так і надати цю нагоду застосунку, який буде заповнювати цю інформацію відносно введеної інформації на Dashboard.

Остання вкладка під назвою «Profile» має стандартний набір параметрів, таких як перегляд персональної інформації, цілей та історії змін ваги.

4.2 Тестування за стосунку

Було проведено ряд тестувань серед декількох пристроїв із різними версіями Android. Результати занотовано у таблицю 4.1. Розглянуто такі властивості:

- плавність анімації та роботи застосунку;
- перевірка коректного завантаження застосунку;
- перевірка адаптивності інтерфейсу;
- перевірка повідомлень, їх відображений текст та відгук на натискання;
- правильність та вчасність обробки статусів з'єднання;
- коректність роботи сервісів, що встановлюють зв'язок із іншими пристроями;
- коректність збереження та зчитування даних.

Таблиця 4.1 – Результати проведених тестувань

Критерій тестування	Назва пристрою та версія Android		
	Xiaomi POCO X3 - 13	Honor 7A – 9.0	Xiaomi Mi10T – 12.0
Плавність анімації та роботи застосунку	Відмінно	Відмінно	Відмінно
Перевірка коректного завантаження	Коректно	Коректно	Відмінно
Правильність зміни і відображення даних	Відмінно	Коректно	Відмінно
Перевірка нагадувань	Коректно	Некоректно	Коректно
Коректність роботи календаря і перегляду історії	Відмінно	Непогано	Відмінно
Адаптивність інтерфейсу	Так	Місцями ні	Так

Тестування було пройдено успішно на всіх пристроях окрім Honor 7A. Через екран малого розміру та поганої продуктивності системи пристрою Honor

7А було виявлено проблему некоректної роботи календаря та перегляду історії, а також отримання нагадувань. Усі інші операції, що передбачені функціоналом застосунку, виконувались коректно та без проблем.

4.3 Висновки за розділом 4

У даному розділі було проведено аналіз процесу функціонування, описуючи ключові аспекти дії програмного забезпечення. Також було виконано тестування на різних версіях операційної системи для мобільних пристроїв та планшетів. В результаті виявлено недоліки у роботі продукту. Загалом можна визначити, що застосунок успішно виконує основні функції та відрізняється стабільністю у роботі.

ВИСНОВКИ

Під час розробки дипломного проєкту була створена рекомендаційна система, спрямована на формування здорових харчових звичок користувача з метою покращення якості життя та здоров'я.

Проведено огляд існуючих аналогів, порівняно їх і виявлено як недоліки, так і переваги. Для розробки застосунку було обрано мову програмування Kotlin та основне середовище розробки Android Studio, яке є офіційним продуктом для створення додатків під ОС Android. Допоміжне середовище IntelliJ IDEA використовувалося для створення діаграм-описів розроблених класів.

Під час розробки Android-додатку було використано архітектурний підхід Stateful MVVM. Також були розглянуті та впроваджені патерни проектування для покращення якості кодової бази та зручності читання коду. Для освоєння та використання додатку були обрані сучасні бібліотеки. Всю історію створення та код проєкту можна подивитись у репозиторію на GitHub: <https://github.com/AlterJuice/YumHub>

Було витрачено чимало часу на опанування документацій та порад щодо розробки продукту. Цей аналіз допоміг створити стабільну архітектуру, що у майбутньому допоможе легко модернізувати та доповнювати застосунок новим функціоналом.

Створений продукт відповідає основній меті розробки, та вирішує усі вищезазначені проблеми, а саме – ведення історії харчування, перегляд статистики і розрахунок особистих рекомендацій користувачеві.

Думки щодо вдосконалення застосунку у майбутньому:

- покращення точності розрахунків рекомендаційної системи;
- налагодження рекомендаційної системи до годин доби та типів прийому їжі;
- збільшення бази даних продуктів;

- збір статистики та відгуків користувачів для покращення алгоритмів системи;
- підключення штучного інтелекту для обчислень великої кількості даних, тренування моделі та запровадження у застосунок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. MyFitnessPal [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=uk&gl=US>
2. Yazio [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.yazio.android>
3. Nutrilio [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=net.nutrilio&hl=ru>
4. Android statistics 2022 [Electronic resource]. – Access mode: <https://www.businessofapps.com/data/android-statistics>
5. Kotlin programming language [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language))
6. Kotlin, компіляція в байткод і продуктивність(частина 1) [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/company/inforion/blog/330060/>
7. Kotlin Extensions [Electronic resource]. – Access mode: <https://kotlinlang.org/docs/extensions.html>
8. Документація Android Source Project [Electronic resource]. – Access mode: <https://developer.android.com/studio/projects/add-app-module#CreatingAModule>
9. Документація Android Source Project [Electronic resource]. – Access mode: <https://developer.android.com/studio/projects/android-library.html>
10. Патерн проектування «Спостерігач» [Електронний ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/observer>
11. Патерни проектування «Одинак» [Електронний ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/singleton>
12. «How to create a Singleton in Kotlin?» [Electronic resource]. – Access mode: <https://blog.mindorks.com/how-to-create-a-singleton-class-in-kotlin>
13. Документація Android Source Project - NavigationUI Component

[Electronic resource]. – Access mode:

<https://developer.android.com/jetpack/compose/navigation>

14. Recipe DB [Electronic resource]. – Access mode:

<https://github.com/tabatkins/recipe-db/tree/master>

15. Harris–Benedict equation [Electronic resource]. – Access mode:

https://en.wikipedia.org/wiki/Harris-Benedict_equation

16. Water Intake calculator [Electronic resource]. – Access mode:

<https://www.acko.com/calculators/water-intake-calculator/>

ДОДАТОК А
Технічне завдання

Вступ

Найменування продукту «YumHub». Застосунок призначений для формування та вплив харчових звичок на здоров'я та якість життя з використанням рекомендаційної системи.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та розробка рекомендаційної системи для формування харчових звичок», затверджене наказом Національного університету «Запорізька політехніка» № 437 від 14 листопада 2023 р.

A.2 Призначення розробки

Метою розробки програмного забезпечення є впровадження ефективної рекомендаційної системи для формування здорових харчових звичок з метою покращення якості життя та здоров'я користувачів.

A.3 Вимоги до програми чи програмного вибору

A.3.1 Вимоги до функціональних характеристик

- збереження та перегляд історії харчування;
- перегляд страв за категоріями для швидкого вибору;
- наявна рекомендаційна система, що аналізує вподобання користувача, його активність;
- можливість перегляду історії харчування у вигляді статистики споживаних нутрієнтів.

А.3.2 Вимоги до надійності

Вимоги до надійності роботи продукту:

- стабільна робота;
- обробка помилок для запобігання несанкціонованого закриття програми.

А.3.3 Умови експлуатації

Умови експлуатації співпадають з умовами експлуатації пристрою, на якому відбувається функціонування застосунку.

А.3.4 Вимоги до складу й параметрів технічних засобів

Технічні вимоги до обладнання:

- наявність екрану;
- не менше 1 ГБ ОЗУ;
- вільний простір не менш 50 МБ.

А.3.5 Вимоги до інформаційної й програмної сумісності

Програмні вимоги:

- сервіс віртуальної клавіатури;
- версія Android від 9.0.

A.3.6 Вимоги до маркування й упакування

Програма повинна мати маркування імені застосунку – «YumHub». У повному обсязі може бути записаний на компакт диск не менше 60МБ.

A.3.7 Вимоги до транспортування й збереження

Вимоги до транспортування і зберігання аналогічні вимогам, що пред'являються до лазерних дисків або карт пам'яті.

A.4 Вимоги до програмної документації

Програмна документація повинна включати: пояснювальну записку на розроблений продукт, вихідний код програми.

A.5 Техніко-економічні показники

Монетизація застосунку передбачає опублікування у Google Play маркет.

A.6 Порядок контролю й приймання

Передбачено захист дипломного проєкту у встановлені керівником проєкту строки.

ДОДАТОК Б
Текст програми

```

package com.alterjuice.yumhub

import android.app.Application
import org.koin.android.ext.koin.androidContext
import org.koin.android.ext.koin.androidLogger
import org.koin.core.context.startKoin
import org.koin.core.logger.Level

class YumHubApp: Application() {

    override fun onCreate() {
        super.onCreate()
        startKoin {
            androidLogger(if (BuildConfig.DEBUG) Level.ERROR else Level.NONE)
            androidContext(this@YumHubApp)
            modules(*AppDI.modules())
        }
    }
}

```

```

package com.alterjuice.yumhub

import com.alterjuice.android_utils.DIModulesHub
import com.alterjuice.dashboard.di.DashboardDI
import com.alterjuice.data.di.DataDI
import com.alterjuice.database.DatabaseDI
import com.alterjuice.meals.MealsDI
import com.alterjuice.navigation.di.NavigationDI
import com.alterjuice.network.di.NetworkDI
import com.alterjuice.onboarding.OnboardingDI
import com.alterjuice.repository.di.RepositoryDI
import com.alterjuice.user_profile.UserProfileDI

object AppDI: DIModulesHub {
    override fun modules() = arrayOf(
        *NetworkDI.modules(),
        *NavigationDI.modules(),
        *RepositoryDI.modules(),
        *DatabaseDI.modules(),
        *DashboardDI.modules(),
        *DataDI.modules(),
    )
}

```

```

        *MealsDI.modules(),
        *UserProfileDI.modules(),
        *OnboardingDI.modules(),
    )
}

```

```
@file:OptIn(ExperimentalMaterial3Api::class)
```

```
package com.alterjuice.yumhub
```

```

import android.graphics.Rect
import android.os.Build
import android.os.Bundle
import android.os.CancellationSignal
import android.view.ScrollCaptureCallback
import android.view.ScrollCaptureSession
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.Crossfade
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.imePadding
import androidx.compose.foundation.layout.navigationBarsPadding
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.wrapContentHeight
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SmallTopAppBar
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.CompositionLocalProvider
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.derivedStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember

```

```

import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalView
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.LayoutDirection
import androidx.compose.ui.unit.dp
import androidx.navigation.NavHostController
import androidx.navigation.compose.rememberNavController
import com.alterjuice.compose_utils.ui.components.ActionBar
import com.alterjuice.compose_utils.ui.components.BrandText
import com.alterjuice.compose_utils.ui.components.YumHubCircledBackground
import com.alterjuice.compose_utils.ui.extensions.get
import com.alterjuice.navigation.NavCommand
import com.alterjuice.navigation.NavCommand.Companion.getDestination
import com.alterjuice.navigation.NavigationManager
import com.alterjuice.navigation.routes DefinedRoutes
import com.alterjuice.theming.theme.LocalAppNavController
import com.alterjuice.theming.theme.YumHubTheme
import com.alterjuice.theming.theme.extension.primaryVariantSchema
import com.alterjuice.yumhub.navigation.YumHubNavigationBar
import com.alterjuice.yumhub.navigation.YumHubNavHost
import org.koin.core.component.KoinComponent
import org.koin.core.component.inject
import java.util.function.Consumer

class MainActivity : ComponentActivity(), KoinComponent {

    private val navigationManager by inject<NavigationManager>()
    private lateinit var navController: NavHostController

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            navController = rememberNavController()
            YumHubTheme {
                val isBottomBarVisible = rememberSaveable { mutableStateOf(true) }
                val isTopBarVisible = rememberSaveable { mutableStateOf(true) }
                var currentRoute by remember {
                    mutableStateOf<NavCommand?>(null)
                }
                LaunchedEffect(Unit) {

```

```

        navController.addOnDestinationChangedListener { _,
destination, _ ->
            val navCommand =
                DefinedRoutes.allRoutes.find
            { destination.route.equals(it.getDestination() ) }
            currentRoute = navCommand
            isBottomBarVisible.value =
navCommand?.isBottomNavigationVisible ?: true
            isTopBarVisible.value = navCommand?.isTopBarVisible ?:
true
        }
    }
    CompositionLocalProvider(
        LocalAppNavController provides navController
    ) {
        Scaffold(
            modifier = Modifier
                .navigationBarsPadding()
                .imePadding(),
            topBar = {
                if (!isTopBarVisible.value) return@Scaffold
                val title = remember(currentRoute) {
                    currentRoute?.topBarTitle
                }
                val isVisible = remember(currentRoute) {
                    currentRoute?.isTopBarVisible == true
                }
                if (isVisible) {
                    ActionBar(
                        modifier = Modifier
                            .fillMaxWidth()
                            .wrapContentHeight(),
                        centeredTitle = title?.get().orEmpty()
                    )
                }
            },
            bottomBar = {

                val isVisible = remember(currentRoute) {
                    currentRoute?.isBottomNavigationVisible ?: true
                }
                if (isVisible) {
                    YumHubNavigationBar(

```

```

        navigationManager = navigationManager,
        navController = navController,
        isVisibleState = isBottomBarVisible
    )
}
},
content = { paddings ->
    YumHubCircledBackground {
        YumHubNavHost(
            navHostController = navController,
            modifier = Modifier.padding(paddings),
            startDestination =
DefinedRoutes.SplashScreen.getDestination()
        )
    }
}
)
}
}
}
}
}

```

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    YumHubTheme {

    }
}
```

```
package com.alterjuice.yumhub
```

```
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.core.tween
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
```

```

import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.text.SpanStyle
import androidx.compose.ui.text.buildAnnotatedString
import androidx.compose.ui.text.font.FontStyle
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.withStyle
import androidx.compose.ui.tooling.preview.Preview
import com.alterjuice.compose_utils.ui.components.SurfaceWithConstraints
import com.alterjuice.compose_utils.ui.components.YumHubOutlinedCard
import com.alterjuice.navigation.NavCommand.Companion.getDestination
import com.alterjuice.navigation.routes DefinedRoutes
import com.alterjuice.repository.storage.OnboardingStorage
import com.alterjuice.resources.R
import com.alterjuice.theming.theme.LocalAppNavController
import kotlinx.coroutines.delay
import org.koin.compose.LocalKoinScope

@Composable
fun SplashScreen(
    modifier: Modifier
) {
    val koinScope = LocalKoinScope.current
    val navigation = LocalAppNavController.current
    val logoVisibility = remember { mutableStateOf(false) }

    LaunchedEffect(Unit) {
        delay(300)
        logoVisibility.value = true
        delay(1500)
        val onboardingStorage = koinScope.get<OnboardingStorage>()
        if (!onboardingStorage.isOnboardingPassed.get()) {
            navigation.navigate(DefinedRoutes.Onboarding.getDestination()) {
                popUpTo(0)
            }
        } else {
            navigation.navigate(DefinedRoutes.Dashboard.getDestination()) {

```

```

        popUpTo(0)
    }
}
YumHubOutlinedCard(
    modifier = modifier,
    backgroundAlpha = 0.5f,
    borderStroke = null,
    surfaceShape = RectangleShape
) {
    val isVisible by logoVisibility
    AnimatedVisibility(
        modifier = Modifier.align(Alignment.Center),
        visible = isVisible,
        enter = fadeIn(tween(1000)),
        exit = fadeOut(tween(1000)),
    ) {
        Text(
            text = buildAnnotatedString {
                withStyle(style = SpanStyle(color = Color(0xFF91B545))) {
                    append("Yum")
                }
                withStyle(style = SpanStyle(color = Color(0xFF597719))) {
                    append("Hub")
                }
            },
            modifier = Modifier,
            color = Color.Unspecified,
            fontStyle = FontStyle(R.font.kyiv_type_sans_regular),
            style = MaterialTheme.typography.displayMedium,
            textAlign = TextAlign.Center
        )
    }
}

@Composable
@Preview
private fun SplashScreenPreview() {
    SplashScreen(
        modifier = Modifier
    )
}

```

```

package com.alterjuice.database.category_history

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.alterjuice.domain.model.common.MealCategories

@Dao
interface CategoryEatenHistoryDao {

    @Query("SELECT * FROM CategoryEatenHistory")
    suspend fun getAllCategoriesAteCount(): List<CategoryEatenHistoryDB>

    // Will be only one or zero items in list
    @Query("SELECT ate_count FROM CategoryEatenHistory WHERE meal_category = :categoryName LIMIT 1")
    suspend fun getCategoryAteCount(categoryName: MealCategories): List<Int>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertCategoryAteCount(categoryEatenHistoryDB: CategoryEatenHistoryDB)

    suspend fun increaseCategoryAteCount(mealCategory: MealCategories) {
        val old = getCategoryAteCount(mealCategory).firstOrNull()?: 0
        val new = CategoryEatenHistoryDB(
            mealCategory = mealCategory,
            categoryAteCount = (old + 1).coerceAtLeast(0)
        )
        insertCategoryAteCount(new)
    }

    suspend fun decreaseCategoryAteCount(mealCategory: MealCategories) {
        val old = getCategoryAteCount(mealCategory).firstOrNull()?: 0
        val new = CategoryEatenHistoryDB(
            mealCategory = mealCategory,
            categoryAteCount = (old - 1).coerceAtLeast(0)
        )
        insertCategoryAteCount(new)
    }
}

```

```
}
```

```
package com.alterjuice.database.category_history
```

```
import androidx.room.ColumnInfo
```

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
import com.alterjuice.domain.model.common.MealCategories
```

```
@Entity(tableName = "CategoryEatenHistory")
```

```
data class CategoryEatenHistoryDB(
```

```
    @PrimaryKey(autoGenerate = false)
```

```
    @ColumnInfo("meal_category") val mealCategory: MealCategories,
```

```
    @ColumnInfo("ate_count", defaultValue = "0") val categoryAteCount: Int,
```

```
)
```

```
package com.alterjuice.database.meals_history
```

```
import androidx.room.Dao
```

```
import androidx.room.Insert
```

```
import androidx.room.OnConflictStrategy
```

```
import androidx.room.Query
```

```
import com.alterjuice.utils.DateTimeUtils
```

```
import kotlinx.coroutines.flow.Flow
```

```
@Dao
```

```
interface MealsHistoryDao {
```

```
    @Query("SELECT * FROM MealsHistory")
```

```
    suspend fun getAllMeals(): List<MealsHistoryDB>
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE, entity =
MealsHistoryDB::class)
```

```
    suspend fun addMealToHistory(mealDB: MealsHistoryDB)
```

```
    @Query("SELECT * FROM MealsHistory WHERE date BETWEEN :leftBoundTime
AND :rightBoundTime")
```

```
    suspend fun getMealsHistoryBetween(
```

```
        leftBoundTime: Long,
```

```
        rightBoundTime: Long
```

```
): List<MealsHistoryDB>
```

```

    @Query("SELECT * FROM MealsHistory WHERE date BETWEEN :leftBoundTime
AND :rightBoundTime")
    fun getMealsHistoryFlowBetween(
        leftBoundTime: Long,
        rightBoundTime: Long
    ): Flow<List<MealsHistoryDB>>

    @Query("DELETE FROM MealsHistory WHERE date = :dateTimestampMs")
    suspend fun deleteMealFromHistoryByID(dateTimestampMs: Long)

    suspend fun getMealsHistoryForDate(timeMs: Long): List<MealsHistoryDB> {
        val startDayAtMs =
DateTimeUtils.getNormalizedStartDateFromMsToMs(timeMs)
        val endDayAtMs = startDayAtMs + (24 * 60 * 59 * 1000)
        return getMealsHistoryBetween(
            leftBoundTime = startDayAtMs,
            rightBoundTime = endDayAtMs
        )
    }

    fun getMealsHistoryForDateFlow(timeMs: Long): Flow<List<MealsHistoryDB>> {
        val startDayAtMs =
DateTimeUtils.getNormalizedStartDateFromMsToMs(timeMs)
        val endDayAtMs = startDayAtMs + (24 * 60 * 59 * 1000)
        return getMealsHistoryFlowBetween(
            leftBoundTime = startDayAtMs,
            rightBoundTime = endDayAtMs
        )
    }

    suspend fun getMealsHistoryForToday(): List<MealsHistoryDB> {
        val startDayAt = DateTimeUtils.getNormalizedStartDateTodayMs()
        return getMealsHistoryForDate(
            timeMs = startDayAt
        )
    }

    fun getMealsHistoryForTodayFlow(): Flow<List<MealsHistoryDB>> {
        val startDayAt = DateTimeUtils.getNormalizedStartDateTodayMs()
        return getMealsHistoryForDateFlow(startDayAt)
    }
}

```

```

package com.alterjuice.database.meals_history

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.alterjuice.domain.model.common.MealType

@Entity(tableName = "MealsHistory")
data class MealsHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "date") val dateTimeStampMs: Long,
    @ColumnInfo(name="meal_type") val mealType: MealType,
    @ColumnInfo(name="meal_info") val info: String, // JSON of YumHubMeal
)

package com.alterjuice.database.messages

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import kotlinx.coroutines.flow.Flow

@Dao
interface MessagesDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun saveMessage(message: MessageEntityDB)

    @Query("SELECT * FROM Messages LIMIT 50")
    suspend fun getMessages(): List<MessageEntityDB>

    @Query("SELECT * FROM Messages LIMIT 50")
    fun getMessagesFlow(): Flow<List<MessageEntityDB>>

    @Query("SELECT * FROM Messages LIMIT :count OFFSET :offset")
    suspend fun getMessages(offset: Int, count: Int): List<MessageEntityDB>
}

package com.alterjuice.database.messages

```

```

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.alterjuice.domain.model.messages.MessageAuthor

@Entity(tableName = "Messages")
data class MessageEntityDB(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "message_id") val messageID: Long,
    @ColumnInfo(name = "author") val author: MessageAuthor,
    @ColumnInfo(name = "time") val messageTimeStamp: String,
    @ColumnInfo(name = "text") val messageText: String,
)

package com.alterjuice.database.nutrients_history

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutritionEnum
import kotlinx.coroutines.flow.Flow

@Dao
interface NutrientsHistoryDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE, entity =
NutrientsHistoryDB::class)
    suspend fun insertNutrientHistory(nutrientsHistoryDB: NutrientsHistoryDB)

    // Will be only one or zero items in list
    @Query("SELECT * FROM NutrientsHistory WHERE date = :dayTimeStampSec LIMIT
1")
    suspend fun getNutrientsHistoryForDate(dayTimeStampSec: Long):
List<NutrientsHistoryDB>

    // Will be only one or zero items in list
    @Query("SELECT * FROM NutrientsHistory WHERE date = :dayTimeStampSec LIMIT
1")

```

```

    fun getNutrientsHistoryFlowForDate(dayTimestampSec: Long):
Flow<List<NutrientsHistoryDB>>

    @Query("SELECT * FROM NutrientsHistory WHERE date
BETWEEN :fromDayTimestampSec AND :toDayTimestampSec")
    suspend fun getNutrientsHistoryForDates(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long
    ): List<NutrientsHistoryDB>

    @Query("SELECT * FROM NutrientsHistory WHERE date
BETWEEN :fromDayTimestampSec AND :toDayTimestampSec")
    fun getNutrientsHistoryFlowForDates(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long
    ): Flow<List<NutrientsHistoryDB>>

    suspend fun updateNutrientsHistoryByAddingMealForDate(meal: YumHubMeal,
dayTimestampSec: Long) {
        val old = getNutrientsHistoryForDate(dayTimestampSec).firstOrNull()
        val updated = if (old == null) {
            // we should just create empty object to add it to history. This
won't be even
            // happened, but the logic requires
            NutrientsHistoryDB(dayTimestampSec)
        } else {
            old.copy(
                proteins = old.proteins + meal.withServings(1.0,
NutritionEnum.Protein),
                energy = old.energy + meal.withServings(1.0,
NutritionEnum.EnergyCalories),
                carbs = old.carbs + meal.withServings(1.0, NutritionEnum.Carbs),
                fat = old.fat + meal.withServings(1.0, NutritionEnum.Fat),
                fiber = old.fiber + meal.withServings(1.0, NutritionEnum.Fiber),
                sugars = old.sugars + meal.withServings(1.0,
NutritionEnum.Sugars),
                sugarsAdded = old.sugarsAdded + meal.withServings(1.0,
NutritionEnum.SugarsAdded)
            )
        }
        insertNutrientHistory(updated)
    }

```

```

suspend fun updateNutrientsHistoryByRemovingMealForDate(
    meal: YumHubMeal,
    dayTimestampSec: Long
) {
    val old = getNutrientsHistoryForDate(dayTimestampSec).firstOrNull()
    val updated = if (old == null) {
        // we should just create empty object to add it to history. This
won't be even
        // happened, but the logic requires
        NutrientsHistoryDB(dayTimestampSec)
    } else {
        old.copy(
            proteins = old.proteins - meal.withServings(1.0,
NutritionEnum.Protein),
            energy = old.energy - meal.withServings(1.0,
NutritionEnum.EnergyCalories),
            carbs = old.carbs - meal.withServings(1.0, NutritionEnum.Carbs),
            fat = old.fat - meal.withServings(1.0, NutritionEnum.Fat),
            fiber = old.fiber - meal.withServings(1.0, NutritionEnum.Fiber),
            sugars = old.sugars - meal.withServings(1.0,
NutritionEnum.Sugars),
            sugarsAdded = old.sugarsAdded - meal.withServings(1.0,
NutritionEnum.SugarsAdded)
        )
    }
    insertNutrientHistory(updated)
}

```

```

package com.alterjuice.database.nutrients_history

```

```

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

```

```

@Entity(tableName = "NutrientsHistory")

```

```

data class NutrientsHistoryDB(

```

```

    @PrimaryKey(autoGenerate = false)

```

```

    @ColumnInfo(name = "date") val dateTimestampSec: Long,

```

```

    @ColumnInfo(name = "proteins", defaultValue = "0.0") val proteins: Double =
0.0,

```

```

    @ColumnInfo(name = "energy", defaultValue = "0.0") val energy: Double = 0.0,

```

```

        @ColumnInfo(name = "carbs", defaultValue = "0.0") val carbs: Double = 0.0,
        @ColumnInfo(name = "fat", defaultValue = "0.0") val fat: Double = 0.0,
        @ColumnInfo(name = "fiber", defaultValue = "0.0") val fiber: Double = 0.0,
        @ColumnInfo(name = "sugars", defaultValue = "0.0") val sugars: Double = 0.0,
        @ColumnInfo(name = "sugarsAdded", defaultValue = "0.0") val sugarsAdded:
Double = 0.0,
        @ColumnInfo(name = "calcium", defaultValue = "0.0") val calcium: Double =
0.0,
    )

```

```
package com.alterjuice.database.user_measurements
```

```

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.alterjuice.domain.model.MeasurementTypes
import com.alterjuice.utils.DateTimeUtils
import kotlinx.coroutines.flow.Flow

```

```
@Dao
```

```
interface InternalUserMeasurementsHistoryDao {
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
```

```
    suspend fun internalUpdateOrAddMeasurementsForDate(entity:

```

```
UserMeasurementsHistoryDB)
```

```

    @Query("SELECT * FROM UserMeasurements WHERE dayStartTimestampSec
= :dayTimestampSec AND type IN (:types)")

```

```
    suspend fun internalGetMeasurementsForDate(
```

```
        dayTimestampSec: Long,
```

```
        types: List<MeasurementTypes>
```

```
): List<UserMeasurementsHistoryDB>
```

```

    @Query("SELECT * FROM UserMeasurements WHERE type in (:types) AND
dayStartTimestampSec BETWEEN :fromDayTimestampSec AND :toDayTimestampSec")

```

```
    suspend fun internalGetMeasurementsForDates(
```

```
        fromDayTimestampSec: Long,
```

```
        toDayTimestampSec: Long,
```

```

        types: List<MeasurementTypes>
    ): List<UserMeasurementsHistoryDB>

    @Query("SELECT * FROM UserMeasurements WHERE type IN (:types)")
    suspend fun getAllMeasurementsByType(types: List<MeasurementTypes>):
List<UserMeasurementsHistoryDB>

    @Query("SELECT * FROM UserMeasurements WHERE dayStartTimestampSec
= :dayTimestampSec AND type IN (:types)")
    fun internalGetMeasurementsForDateFlow(
        dayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurementsHistoryDB>>

    @Query("SELECT * FROM UserMeasurements WHERE type in (:types) AND
dayStartTimestampSec BETWEEN :fromDayTimestampSec AND :toDayTimestampSec")
    fun internalGetMeasurementsForDatesFlow(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurementsHistoryDB>>

    @Query("SELECT * FROM UserMeasurements WHERE type IN (:types)")
    fun getAllMeasurementsByTypeFlow(types: List<MeasurementTypes>):
Flow<List<UserMeasurementsHistoryDB>>

}

@Dao
interface UserMeasurementsHistoryDao : InternalUserMeasurementsHistoryDao {

    suspend fun getMeasurementsForDate(
        dayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): List<UserMeasurementsHistoryDB> {
        return internalGetMeasurementsForDate(
            dayTimestampSec =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec),
            types = types
        )
    }
}

```

```

    }

    suspend fun getMeasurementsForDates(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): List<UserMeasurementsHistoryDB> {
        val normalizedFrom =
DateTimeUtils.getNormalizedStartDateFromSec(fromDayTimestampSec)
        val normalizedTo =
DateTimeUtils.getNormalizedStartDateFromSec(toDayTimestampSec)
        return internalGetMeasurementsForDates(
            fromDayTimestampSec = normalizedFrom,
            toDayTimestampSec = normalizedTo,
            types = types
        )
    }

    fun getMeasurementsForDateFlow(
        dayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurementsHistoryDB>> {
        return internalGetMeasurementsForDateFlow(
            dayTimestampSec =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec),
            types = types
        )
    }

    fun getMeasurementsForDatesFlow(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurementsHistoryDB>> {
        val normalizedFrom =
DateTimeUtils.getNormalizedStartDateFromSec(fromDayTimestampSec)
        val normalizedTo =
DateTimeUtils.getNormalizedStartDateFromSec(toDayTimestampSec)
        return internalGetMeasurementsForDatesFlow(
            fromDayTimestampSec = normalizedFrom,

```

```

        toDayTimestampSec = normalizedTo,
        types = types
    )
}
}

```

```
package com.alterjuice.database.user_measurements
```

```

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.alterjuice.domain.model.MeasurementTypes

```

```

@Entity(tableName = "UserMeasurements")
data class UserMeasurementsHistoryDB(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name = "dayStartTimestampSec") val dayTimestampSec: Long,
    @ColumnInfo(name = "type") val measurementType: MeasurementTypes,
    @ColumnInfo(name = "value") val value: Float,
)

```

```
package com.alterjuice.database
```

```

import com.alterjuice.android_utils.DIModulesHub
import com.alterjuice.database.category_history.CategoryEatenHistoryDao
import com.alterjuice.database.meals_history.MealsHistoryDao
import com.alterjuice.database.messages.MessagesDao
import com.alterjuice.database.nutrients_history.NutrientsHistoryDao
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDao
import org.koin.android.ext.koin.androidApplication
import org.koin.dsl.module

```

```

object DatabaseDI : DIModulesHub {
    override fun modules() = arrayOf(
        databaseModule()
    )

    private fun databaseModule() = module {
        single<YumHubDatabase> {
            createYumHubDatabase(androidApplication())
        }
        single<MessagesDao> {
            get<YumHubDatabase>().messagesEntities()

```

```

    }
    single<MealsHistoryDao> {
        get<YumHubDatabase>().mealsInfoEntities()
    }
    single<NutrientsHistoryDao> {
        get<YumHubDatabase>().nutrientsHistoryEntities()
    }
    single<CategoryEatenHistoryDao> {
        get<YumHubDatabase>().categoryEatenHistoryEntities()
    }
    single<UserMeasurementsHistoryDao> {
        get<YumHubDatabase>().measurementsHistoryDao()
    }
}
}

package com.alterjuice.database

import android.app.Application
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import com.alterjuice.database.category_history.CategoryEatenHistoryDB
import com.alterjuice.database.category_history.CategoryEatenHistoryDao
import com.alterjuice.database.meals_history.MealsHistoryDB
import com.alterjuice.database.meals_history.MealsHistoryDao
import com.alterjuice.database.messages.MessageEntityDB
import com.alterjuice.database.messages.MessagesDao
import com.alterjuice.database.nutrients_history.NutrientsHistoryDB
import com.alterjuice.database.nutrients_history.NutrientsHistoryDao
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDB
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDao

@Database(
    entities = [
        MessageEntityDB::class,
        MealsHistoryDB::class,
        CategoryEatenHistoryDB::class,
        NutrientsHistoryDB::class,
        UserMeasurementsHistoryDB::class
    ],
    version = 1,

```

```

        exportSchema = true,
        autoMigrations = arrayOf()
    )
}

abstract class YumHubDatabase : RoomDatabase() {
    abstract fun messagesEntities(): MessagesDao
    abstract fun mealsInfoEntities(): MealsHistoryDao
    abstract fun categoryEatenHistoryEntities(): CategoryEatenHistoryDao
    abstract fun measurementsHistoryDao(): UserMeasurementsHistoryDao
    abstract fun nutrientsHistoryEntities(): NutrientsHistoryDao
}

fun createYumHubDatabase(application: Application) = Room.databaseBuilder(
    context = application,
    klass = YumHubDatabase::class.java,
    name = "YumHubDatabase"
).build()

package com.alterjuice.navigation.routes

import androidx.navigation.NavType
import androidx.navigation.navArgument
import com.alterjuice.android_utils.Str
import com.alterjuice.domain.model.common.MealType
import com.alterjuice.navigation.NavCommand
import com.alterjuice.navigation.NavCommand.Companion.getNavRoute
import com.alterjuice.navigation.NavRoute

object DefinedRoutes {
    val SplashScreen = NavCommand.new("splashscreen",
        isBottomNavigationVisible = false,
        isTopBarVisible = false
    )

    val Onboarding = NavCommand.new("onboarding",
        isBottomNavigationVisible = false,
        isTopBarVisible = false
    )

    val ChatAssistant = NavCommand.new("chatAssistant")

    val Meals = NavCommand.new("meals")

```

```

object MealPage: NavCommand by NavCommand.new("meals_page",
    isBottomNavigationVisible = false,
    isTopBarVisible = false
) {
    const val argMealID = "mealID"

    override val arguments = listOf(
        navArgument(argMealID) { type = NavType.StringType }
    )
    operator fun invoke(mealID: String): NavRoute = getNavRoute(mealID)
}

val Authentication = NavCommand.new("authentication")

object AddMeal: NavCommand by NavCommand.new(
    rootRoute = "addMeal",
    isBottomNavigationVisible = false
) {
    const val argMealType = "mealType"
    override val arguments = listOf(
        navArgument(argMealType) { type = NavType.StringType }
    )
    operator fun invoke(mealType: MealType): NavRoute =
getNavRoute(mealType.name)
}

val Dashboard = NavCommand.new("dashboard")
val History = NavCommand.new(
    rootRoute = "history",
    topBarTitle = Str("History")
)
val Profile = NavCommand.new(
    rootRoute = "profile",
    topBarTitle = Str("Profile")
)

val allRoutes by lazy {
    mutableListOf<NavCommand>(
        SplashScreen, Authentication, Onboarding, ChatAssistant,
        Meals, MealPage, AddMeal, Dashboard, History, Profile)
}

```

```

}

package com.alterjuice.repository.mappers

import com.alterjuice.database.meals_history.MealsHistoryDB
import com.alterjuice.database.nutrients_history.NutrientsHistoryDB
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDB
import com.alterjuice.domain.model.MealsHistory
import com.alterjuice.domain.model.MeasurementTypes
import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutrientsHistory
import com.alterjuice.domain.model.user.UserMeasurements
import com.alterjuice.domain.model.water.WaterBalance
import com.google.gson.Gson

fun UserMeasurementsHistoryDB.toUserHeight() = UserMeasurements.Height(
    dayTimestampSec = dayTimestampSec,
    height = value
)

fun UserMeasurementsHistoryDB.toWaterBalance() = WaterBalance(
    dayTimestampSec = dayTimestampSec,
    balanceML = value.toInt()
)

fun UserMeasurementsHistoryDB.toWaterMeasurement() =
UserMeasurements.WaterBalance(
    dayTimestampSec = dayTimestampSec,
    balanceML = value.toInt()
)

fun UserMeasurements.toDB() = UserMeasurementsHistoryDB(
    dayTimestampSec = dayTimestampSec,
    measurementType = type,
    value = getValue()
)

fun UserMeasurementsHistoryDB.toUserWeight() = UserMeasurements.Weight(
    dayTimestampSec = dayTimestampSec,
    weight = value
)

```

```
fun UserMeasurementsHistoryDB.toUserMeasurement() = when (this.measurementType)
{
    MeasurementTypes.WATER_BALANCE -> toWaterMeasurement()
    MeasurementTypes.HEIGHT -> toUserHeight()
    MeasurementTypes.WEIGHT -> toUserWeight()
}
```

```
fun MealsHistoryDB.toDomain(
    gson: Gson
) = MealsHistory(
    dateTimeStampMs = dateTimeStampMs,
    mealType = mealType,
    yumHubMeal = gson.fromJson(info, YumHubMeal::class.java)
)
```

```
fun NutrientsHistoryDB.toDomain() = NutrientsHistory(
    dateTimeStampSec = dateTimeStampSec,
    proteins = proteins,
    energy = energy,
    carbs = carbs,
    fat = fat,
    fiber = fiber,
    sugars = sugars,
    sugarsAdded = sugarsAdded,
    calcium = calcium,
)
```

```
fun MealsHistory.toDB(gson: Gson) = MealsHistoryDB(
    dateTimeStampMs = dateTimeStampMs,
    mealType = mealType,
    info = gson.toJson(yumHubMeal)
)
```

```
package com.alterjuice.repository.repository
```

```
import com.alterjuice.database.meals_history.MealsHistoryDB
import com.alterjuice.database.meals_history.MealsHistoryDao
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDB
import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDao
import com.alterjuice.domain.model.MealsHistory
import com.alterjuice.domain.model.MeasurementTypes
```

```

import com.alterjuice.domain.model.common.MealType
import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.water.WaterBalance
import com.alterjuice.domain.repository.MealsHistoryRepository
import com.alterjuice.domain.repository.NutrientsHistoryRepository
import com.alterjuice.repository.mappers.toDomain
import com.alterjuice.repository.mappers.toWaterBalance
import com.alterjuice.utils.DateTimeUtils
import com.google.gson.Gson
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

class MealsHistoryRepositoryImpl(
    private val mealsHistoryDao: MealsHistoryDao,
    private val nutrientsHistoryRepository: NutrientsHistoryRepository,
    private val measurementsHistoryDao: UserMeasurementsHistoryDao,
    private val gson: Gson,
): MealsHistoryRepository {
    override suspend fun getAllMeals(): List<MealsHistory> {
        return mealsHistoryDao.getAllMeals().map {
            it.toDomain(gson)
        }
    }

    override suspend fun addMealInfo(meal: YumHubMeal, mealType: MealType) {
        val startOfDayDateSec = DateTimeUtils.getNormalizedStartDateTodaySec()
        val dateTimeStampMs = DateTimeUtils.getCurrentTimeMillis()
        nutrientsHistoryRepository.updateNutrientsHistoryByAddingMealForDate(
            meal, startOfDayDateSec
        )
        mealsHistoryDao.addMealToHistory(MealsHistoryDB(
            dateTimeStampMs = dateTimeStampMs,
            mealType = mealType,
            info = gson.toJson(meal)
        ))
        println(1)
    }

    override suspend fun removeMealInfo(meal: MealsHistory) {
        val startOfDaySec = DateTimeUtils.getNormalizedStartDateFromMsToSec(
            meal.dateTimeStampMs
        )
        nutrientsHistoryRepository.updateNutrientsHistoryByRemovingMealForDate(

```

```

        meal.yumHubMeal, startOfDaySec
    )
    mealsHistoryDao.deleteMealFromHistoryByID(meal.dateTimestampMs)
}

override suspend fun getMealsHistoryToday(): List<MealsHistory> {
    return mealsHistoryDao.getMealsHistoryForToday().map
{ it.toDomain(gson) }
}

override fun getMealsHistoryTodayFlow(): Flow<List<MealsHistory>> {
    return mealsHistoryDao.getMealsHistoryForTodayFlow().map { it.map
{ it.toDomain(gson) } }
}

override suspend fun getMealHistoryForDate(timeMs: Long): List<MealsHistory>
{
    return mealsHistoryDao.getMealsHistoryForDate(timeMs = timeMs).map {
        it.toDomain(gson)
    }
}

override fun getMealHistoryForDateFlow(timeMs: Long):
Flow<List<MealsHistory>> {
    return mealsHistoryDao.getMealsHistoryForDateFlow(timeMs = timeMs).map {
        it.map { it.toDomain(gson) }
    }
}
}

package com.alterjuice.repository.repository

import com.alterjuice.domain.model.messages.Message
import com.alterjuice.domain.repository.MessagesRepositoryExtended
import com.alterjuice.repository.datasources.messages.MessagesLocalDataSource
import com.alterjuice.repository.datasources.messages.MessagesRemoteDataSource
import com.alterjuice.utils.extensions.onCompletion
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableStateFlow

internal class MessagesRepositoryImpl(
    private val local: MessagesLocalDataSource,

```

```

        private val remote: MessagesRemoteDataSource,
    ): MessagesRepositoryExtended {
        override val waitingForResponse = MutableStateFlow<Boolean>(false)

        override suspend fun getMessages() = runCatching<List<Message>> {
            local.getMessages()
        }

        override suspend fun sendMessage(message: Message) = runCatching<Message> {
            saveMessage(message)
            val deferredMessage = remote.sendMessage(message)
            waitingForResponse.value = true
            deferredMessage.await().onCompletion {
                waitingForResponse.value = false
            }.onSuccess { saveMessage(it) }.getOrNull()
        }

        override fun getMessagesFlow(): Flow<List<Message>> {
            return local.getMessagesFlow()
        }

        override suspend fun saveMessage(message: Message) {
            local.saveMessage(message)
        }
    }
}

```

```
package com.alterjuice.repository.repository
```

```

import com.alterjuice.database.category_history.CategoryEatenHistoryDao
import com.alterjuice.database.nutrients_history.NutrientsHistoryDao
import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutrientsHistory
import com.alterjuice.domain.repository.NutrientsHistoryRepository
import com.alterjuice.repository.mappers.toDomain
import com.alterjuice.utils.DateTimeUtils
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.flow.mapNotNull

```

```

class NutrientsHistoryRepositoryImpl(
    private val nutrientsHistoryDao: NutrientsHistoryDao,
    private val categoryEatenHistoryDao: CategoryEatenHistoryDao,

```

```

): NutrientsHistoryRepository {
    override fun getNutrientsHistoryFlowForDate(dayTimestampSec: Long):
Flow<NutrientsHistory> {
        val normalizedDate =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec)
        return
nutrientsHistoryDao.getNutrientsHistoryFlowForDate(normalizedDate).mapNotNull {
            it.firstOrNull()?.toDomain()
        }
    }

    override suspend fun getNutrientsHistoryForDate(dayTimestampSec: Long):
NutrientsHistory? {
        val normalizedDate =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec)
        return
nutrientsHistoryDao.getNutrientsHistoryForDate(normalizedDate).firstOrNull()?.to
Domain()
    }

    override suspend fun getNutrientsHistoryForDates(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long
    ): List<NutrientsHistory> {
        return nutrientsHistoryDao.getNutrientsHistoryForDates(
            fromDayTimestampSec = fromDayTimestampSec,
            toDayTimestampSec = toDayTimestampSec,
        ).map {
            it.toDomain()
        }
    }

    override fun getNutrientsHistoryFlowForDates(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long
    ): Flow<List<NutrientsHistory>> {
        return nutrientsHistoryDao.getNutrientsHistoryFlowForDates(
            fromDayTimestampSec = fromDayTimestampSec,
            toDayTimestampSec = toDayTimestampSec,
        ).map { it.map { it.toDomain() } }
    }

    override suspend fun updateNutrientsHistoryByAddingMealForDate(

```

```

        meal: YumHubMeal,
        dayTimestampSec: Long
    ) {
        val normalizedDate =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec)
        meal.categoriesTags.forEach {
            categoryEatenHistoryDao.increaseCategoryAteCount(it)
        }
        nutrientsHistoryDao.updateNutrientsHistoryByAddingMealForDate(meal,
normalizedDate)
    }

    override suspend fun updateNutrientsHistoryByRemovingMealForDate(
        meal: YumHubMeal,
        dayTimestampSec: Long
    ) {
        val normalizedDate =
DateTimeUtils.getNormalizedStartDateFromSec(dayTimestampSec)
        meal.categoriesTags.forEach {
            categoryEatenHistoryDao.decreaseCategoryAteCount(it)
        }
        nutrientsHistoryDao.updateNutrientsHistoryByRemovingMealForDate(meal,
normalizedDate)
    }
}

package com.alterjuice.repository.repository

import com.alterjuice.database.user_measurements.UserMeasurementsHistoryDao
import com.alterjuice.domain.model.MeasurementTypes
import com.alterjuice.domain.model.user.UserMeasurements
import com.alterjuice.domain.repository.UserMeasurementsRepository
import com.alterjuice.repository.mappers.toDB
import com.alterjuice.repository.mappers.toUserMeasurement
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

class UserMeasurementsRepositoryImpl(
    private val userMeasurementsHistoryDao: UserMeasurementsHistoryDao
) : UserMeasurementsRepository {

    /*

```

```

*
userMeasurementsHistoryDao.internalUpdateOrAddMeasurementsForDate (UserMeasuremen
tsHistoryDB(
    dayTimestampSec = DateTimeUtils.getNormalizedStartDateTodaySec() -
10*24*60*60,
    measurementType = MeasurementTypes.WEIGHT,
    value = 90f
))
* */

override suspend fun addMeasurement(measurement: UserMeasurements) {
    userMeasurementsHistoryDao.internalUpdateOrAddMeasurementsForDate (
        entity = measurement.toDB()
    )
}

override suspend fun getMeasurementsForDate(
    dayTimestampSec: Long,
    types: List<MeasurementTypes>
): List<UserMeasurements> {
    return userMeasurementsHistoryDao.getMeasurementsForDate(
        dayTimestampSec = dayTimestampSec,
        types = types
    ).map { it.toUserMeasurement() }
}

override suspend fun getMeasurementsForDates(
    fromDayTimestampSec: Long,
    toDayTimestampSec: Long,
    types: List<MeasurementTypes>
): List<UserMeasurements> {
    return userMeasurementsHistoryDao.getMeasurementsForDates(
        fromDayTimestampSec = fromDayTimestampSec,
        toDayTimestampSec = toDayTimestampSec,
        types = types
    ).map { it.toUserMeasurement() }
}

override suspend fun getAllMeasurementsByType(types:
List<MeasurementTypes>): List<UserMeasurements> {
    return userMeasurementsHistoryDao.getAllMeasurementsByType(types).map {
        it.toUserMeasurement()
    }
}

```

```

        }
    }

    override fun getMeasurementsForDateFlow(
        dayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurements>> {
        return userMeasurementsHistoryDao.getMeasurementsForDateFlow(
            dayTimestampSec = dayTimestampSec,
            types = types
        ).map { it.map { it.toUserMeasurement() } }
    }

    override fun getMeasurementsForDatesFlow(
        fromDayTimestampSec: Long,
        toDayTimestampSec: Long,
        types: List<MeasurementTypes>
    ): Flow<List<UserMeasurements>> {
        return userMeasurementsHistoryDao.getMeasurementsForDatesFlow(
            fromDayTimestampSec = fromDayTimestampSec,
            toDayTimestampSec = toDayTimestampSec,
            types = types
        ).map { it.map { it.toUserMeasurement() } }
    }

    override fun getAllMeasurementsByTypeFlow(types: List<MeasurementTypes>):
Flow<List<UserMeasurements>> {
        return
userMeasurementsHistoryDao.getAllMeasurementsByTypeFlow(types).map {
            it.map { it.toUserMeasurement() }
        }
    }
}

package com.alterjuice.repository.storage

import android.content.SharedPreferences
import androidx.annotation.CallSuper
import androidx.core.content.edit

abstract class PreferenceStorage: StorageCleaner {
    protected abstract val preferences: SharedPreferences

```

```

    @CallSuper
    override suspend fun clearData() {
        preferences.edit { clear() }
    }
}

package com.alterjuice.repository.storage

fun interface StorageCleaner {
    suspend fun clearData()
}

package com.alterjuice.data.analyzers

import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutritionAttr
import com.alterjuice.domain.model.nutrition.NutritionEnum
import kotlin.math.abs

object AbsoluteDifferenceSimilarityAnalysis {

    val mainNutrientsToCompare = listOf(
        NutritionEnum.Fiber,
        NutritionEnum.Fat,
        NutritionEnum.Protein,
        NutritionEnum.Sugars,
        NutritionEnum.EnergyCalories
    )

    // Функція що створює та заповнює матрицю схожесті
    // Entry point
    fun createSimilarityMatrixBetweenMeals(
        products: List<YumHubMeal>,
        nutrientsToCompare: List<NutritionEnum> = mainNutrientsToCompare
    ): Array<DoubleArray> {
        // Розрахунок матриці схожості для кожного нутрієнта
        val matrices = nutrientsToCompare.map { nutrient ->
            absoluteDifferenceMatrix(products, nutrient)
                .let(::toSimilarityMatrix)
        }
        // val categorySimilarityMatrix =
        //     calculateSimilarityMatrixByCategories(

```

```

//          products = products
//      )

// Об'єднання матриць схожості
val aggregated = aggregateSimilarityMatrices(
    productsSize = products.size,
    matrices = matrices //+ listOf(categorySimilarityMatrix)
)
return aggregated
}

// SubEntry point
fun absoluteDifferenceMatrix(
    products: List<YumHubMeal>,
    nutrient: NutritionAttr
): Array<DoubleArray> {
    val matrixSize = products.size
    return Array(matrixSize) { i ->
        DoubleArray(matrixSize) { j ->
            if (i == j) {
                0.0
            } else {
                absoluteDifference(
                    products[i].withServings(1.0, nutrient),
                    products[j].withServings(1.0, nutrient)
                )
            }
        }
    }
}

// post entry point
fun aggregateSimilarityMatrices(
    productsSize: Int,
    matrices: List<Array<DoubleArray>>
): Array<DoubleArray> {
    return Array(productsSize) { i ->
        DoubleArray(productsSize) { j ->
            matrices.map { matrix ->
                val value = matrix[i][j]
                if (value.isNaN()) 1.0 else value
            }.let { it.sum().div(it.size) }
        }
    }
}

```

```

    }
}

// SubEntry point
fun calculateSimilarityMatrixByCategories(
    products: List<YumHubMeal>
): Array<DoubleArray> {
    return Array(products.size) { i ->
        DoubleArray(products.size) { j ->
            if (i == j) {
                1.0
            } else {
                calculateCategorySimilarity(
                    product1 = products[i],
                    product2 = products[j]
                )
            }
        }
    }
}

private fun toSimilarityMatrix(
    absoluteDifferenceMatrix: Array<DoubleArray>
): Array<DoubleArray> {
    val maxDifference = absoluteDifferenceMatrix.flatMap
{ it.toList() }.maxOrNull()?: 1.0
    return Array(absoluteDifferenceMatrix.size) { i ->
        DoubleArray(absoluteDifferenceMatrix[i].size) { j ->
            1.0f - absoluteDifferenceMatrix[i][j].div(maxDifference)
        }
    }
}

private fun calculateCategorySimilarity(product1: YumHubMeal, product2:
YumHubMeal): Double {
    val categories1 = product1.categoriesTags.toSet()
    val categories2 = product2.categoriesTags.toSet()

    val intersectionSize =
categories1.intersect(categories2).size.toDouble()
    val unionSize = categories1.union(categories2).size.toDouble()

```

```

        return if (unionSize > 0) {
            intersectionSize / unionSize
        } else {
            0.0
        }
    }
}

private fun absoluteDifference(a: Double, b: Double): Double {
    return abs(a - b)
}
}

package com.alterjuice.data.analyzers

import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutrientsItem
import com.alterjuice.domain.model.nutrition.NutritionEnum
import com.alterjuice.utils.extensions.contains
import kotlin.math.sqrt

class CosineSimilarityAnalysis {
    private val nutrientWeight = 0.3
    private val categoryWeight = 0.5

    val mainNutrientsToCompare = listOf(
        NutritionEnum.Protein,
        NutritionEnum.Carbs,
        NutritionEnum.Fat,
        NutritionEnum.FatSat,
        NutritionEnum.Sugars,
        NutritionEnum.EnergyCalories
    )

    // Entry point for one meal
    fun calculateCosineSimilarityForMeal(
        productToAnalyze: YumHubMeal,
        products: List<YumHubMeal>,
    ): List<Pair<YumHubMeal, Double>> {
        return products.map { product ->
            product to calculateOverallSimilarity(
                product1 = productToAnalyze,
                product2 = product,
                nutrientWeight = nutrientWeight,

```

```

        categoryWeight = categoryWeight
    )
}

// Entry point for all meals
fun createSimilarityMatrixBetweenMeals(products: List<YumHubMeal>):
Array<DoubleArray> {
    val matrixSize = products.size

    val similarityMatrix = Array(matrixSize) { i ->
        DoubleArray(matrixSize) { j ->
            if (i == j) {
                1.0 // Similarity between the same item is always 1
            } else {
                calculateOverallSimilarity(
                    product1 = products[i],
                    product2 = products[j]
                ) // Calculating similarity between two items
            }
        }
    }

    return similarityMatrix
}

// Entry point for between two meals
fun calculateOverallSimilarity(
    product1: YumHubMeal,
    product2: YumHubMeal
): Double {
    return calculateOverallSimilarity(
        product1 = product1,
        product2 = product2,
        nutrientWeight = nutrientWeight,
        categoryWeight = categoryWeight
    )
}

// Entry point for between two meals
fun calculateOverallSimilarity(
    product1: YumHubMeal,

```

```

        product2: YumHubMeal,
        nutrientWeight: Double,
        categoryWeight: Double
    ): Double {
        val nutrientSimilarity = calculateCosineSimilarity(product1, product2)
        val categorySimilarity = calculateCategorySimilarity(product1, product2)

        return (nutrientWeight * nutrientSimilarity + categoryWeight *
categorySimilarity) / (nutrientWeight + categoryWeight)
    }

private fun calculateCosineSimilarity(
    product1: YumHubMeal,
    product2: YumHubMeal
): Double {
    val nutrients1 = product1.nutrients.filter { attr ->
        mainNutrientsToCompare.contains {
            it.attrID == attr.attr.attrID
        }
    }

    val nutrients2 = product2.nutrients.filter { attr ->
        mainNutrientsToCompare.contains {
            it.attrID == attr.attr.attrID
        }
    }

    val dotProduct = nutrients1.mapNotNull { nutrient1 ->
        nutrients2.firstOrNull { it.attr.attrID ==
nutrient1.attr.attrID }?.let { nutrient2 ->
            product1.withOneServing(nutrient1.attr) *
product2.withOneServing(nutrient2.attr)
        }
    }.sum()

    val magnitude1 = calculateMagnitude(product1, nutrients1)
    val magnitude2 = calculateMagnitude(product2, nutrients2)

    return if (magnitude1 > 0 && magnitude2 > 0) {
        dotProduct / (magnitude1 * magnitude2)
    } else {
        0.0
    }
}

```

```

    }
}

private fun calculateCategorySimilarity(product1: YumHubMeal, product2:
YumHubMeal): Double {
    val categories1 = product1.categoriesTags.toSet()
    val categories2 = product2.categoriesTags.toSet()

    val intersectionSize =
categories1.intersect(categories2).size.toDouble()
    val unionSize = categories1.union(categories2).size.toDouble()

    return if (unionSize > 0) {
        intersectionSize / unionSize
    } else {
        0.0
    }
}

private fun calculateMagnitude(product: YumHubMeal, nutrients:
List<NutrientsItem>): Double {
    return nutrients.map
{ product.withOneServing(it.attr) }.toDoubleArray().let { vector ->
        sqrt(vector.fold(0.0) { acc, value -> acc + value * value })
    }
}
}

package com.alterjuice.data.analyzers

import com.alterjuice.domain.model.common.MealCategories
import com.alterjuice.domain.model.common.YumHubMeal
import com.alterjuice.domain.model.nutrition.NutritionEnum
import com.alterjuice.domain.model.nutrition.valueOrZero
import com.alterjuice.domain.model.user.FitnessGoal
import com.alterjuice.domain.model.user.UserInfo
import com.alterjuice.domain.model.user.UserPAL

object PointWisePrognosis {

    private val fatWeight = mapOf(
        FitnessGoal.LoseWeight to 0.7,
        FitnessGoal.BuildMuscle to 1.2,

```

```

        FitnessGoal.MaintainWeight to 1.0
    )

    private val caloriesWeight = mapOf(
        FitnessGoal.LoseWeight to 0.9,
        FitnessGoal.BuildMuscle to 1.1,
        FitnessGoal.MaintainWeight to 1.0
    )

    // Додатковий коефіцієнт нутрієнтів (протеїн) в залежності від цілей
    користувача
    private val proteinWeight = mapOf(
        FitnessGoal.LoseWeight to 1.0,
        FitnessGoal.BuildMuscle to 1.2,
        FitnessGoal.MaintainWeight to 1.0
    )

    private val carbsWeight = mapOf(
        FitnessGoal.LoseWeight to 0.8,
        FitnessGoal.BuildMuscle to 1.0,
        FitnessGoal.MaintainWeight to 1.0
    )

    private val proteinPalWeight = mapOf(
        UserPAL.Sedentary to 0.8,
        UserPAL.LightlyActive to 1.0,
        UserPAL.ModeratelyActive to 1.2,
        UserPAL.VeryActive to 1.5,
        UserPAL.ExtraActive to 1.5
    )

    private val carbsPalWeight = mapOf(
        UserPAL.Sedentary to 0.9,
        UserPAL.LightlyActive to 1.1,
        UserPAL.ModeratelyActive to 1.3,
        UserPAL.VeryActive to 1.5,
        UserPAL.ExtraActive to 1.5
    )

    private val fatPalWeight = mapOf(
        UserPAL.Sedentary to 0.8,
        UserPAL.LightlyActive to 1.0,
        UserPAL.ModeratelyActive to 1.2,
        UserPAL.VeryActive to 1.5,

```

```

        UserPAL.ExtraActive to 1.5
    )

private val caloriesPalWeight = mapOf(
    UserPAL.Sedentary to 0.9,
    UserPAL.LightlyActive to 1.1,
    UserPAL.ModeratelyActive to 1.3,
    UserPAL.VeryActive to 1.5,
    UserPAL.ExtraActive to 1.5
)

private val selectionWeight = 1.0
private val goalWeight = 0.5
private val palWeight = 0.5

fun calculateWeightsForProductsNormalized(
    products: List<YumHubMeal>,
    userInfo: UserInfo,
    eatenCategories: Map<MealCategories, Int>,
): Map<YumHubMeal, Double/*from 0 to 1*/> {
    val allWeights = calculateWeightsForProducts(
        products = products,
        userInfo = userInfo,
        eatenCategories = eatenCategories
    )
    val maxNonNormalizedWeight = allWeights.maxBy { it.second }.second
    return allWeights.map {
        it.copy(second = it.second.div(maxNonNormalizedWeight))
    }.toMap()
}

private val favoriteWeight = 5.0
private val mainDishCategoryWeight = 3.5
private val eatenWeightCoefficient = 0.2

fun calculateWeightsForProducts(
    products: List<YumHubMeal>,
    userInfo: UserInfo,
    eatenCategories: Map<MealCategories, Int>,
): List<Pair<YumHubMeal, Double>> {

```

```

return products.map { product ->
    val favoriteWeightToAdd = product.categoriesTags.count {
        userInfo.favoriteCategories.contains(it)
    } * favoriteWeight
    val exactTypesWeights = product.categoriesTags.filter
{ it.isSpecifiedType() }.count {
        userInfo.favoriteCategories.contains(it)
    } * mainDishCategoryWeight
    val eatenWeightToAdd = product.categoriesTags.sumOf {
        calculateEatenWeight(eatenCategories, it)
    }
    val goalScore = calculateGoalScore(product, userInfo.fitnessGoal)
    val palScore = calculatePalScore(product, userInfo.pal)

    // Загальна вага продукту
    val totalScore = (goalScore * goalWeight) + (palScore * palWeight) +
        favoriteWeightToAdd + exactTypesWeights + eatenWeightToAdd

    product to totalScore
}

// Calculate eaten weight with using a coefficient
private fun calculateEatenWeight(
    eatenCategories: Map<MealCategories, Int>,
    category: MealCategories
): Double {
    val timesEaten = eatenCategories.getOrElse(category) { 0 }
    return timesEaten * eatenWeightCoefficient
}

private fun calculateGoalScore(product: YumHubMeal, userGoal: FitnessGoal):
Double {
    val fatGoalWeight = fatWeight[userGoal] ?: 1.0
    val caloriesGoalWeight = caloriesWeight[userGoal] ?: 1.0
    val proteinGoalWeight = proteinWeight[userGoal] ?: 1.0
    val carbsGoalWeight = carbsWeight[userGoal] ?: 1.0

    return (product.withOneServing(NutritionEnum.Fat) * fatGoalWeight) +
        (product.calories * caloriesGoalWeight) +
        (product.withOneServing(NutritionEnum.Protein) *
proteinGoalWeight) +
        (product.withOneServing(NutritionEnum.Carbs) * carbsGoalWeight)

```

```

    }

    private fun calculatePalScore(product: YumHubMeal, userPal: UserPAL): Double
    {
        val fatScore = product.nutrients.find { it.attr ==
NutritionEnum.Fat }?.value ?: 0.0
        val caloriesScore = product.calories

        val fatPalWeight = fatPalWeight[userPal] ?: 1.0
        val caloriesPalWeight = caloriesPalWeight[userPal] ?: 1.0
        val proteinPalWeight = proteinPalWeight[userPal] ?: 1.0
        val carbsPalWeight = carbsPalWeight[userPal] ?: 1.0

        return (fatScore * fatPalWeight) +
            (caloriesScore * caloriesPalWeight) +
            (product.protein.valueOrZero * proteinPalWeight) +
            (product.carbs.valueOrZero * carbsPalWeight)
    }
}

object WaterIntakeAnalysis {

    // Рекомендована кількість води (у мілілітрах) за день
    fun calculateTotalDailyWaterIntakeML(
        user: UserInfo, roundedBy: Long = 200L
    ): Long {
        val rawWaterIntake = user.weight * 0.03
        return rawWaterIntake.toLong().getNextDividable(roundedBy)
    }
}

object YumHubGson {

    fun generateGson(): Gson {
        return GsonBuilder()
            .registerTypeAdapterFactory(getYumHubMealAadapter())
            .registerTypeAdapter(NutritionAttr::class.java,
NutritionAttrDeserializer())
            .create()
    }
}

```

```

        fun getYumHubMealAadapter() = RuntimeTypeAdapter.of(YumHubMeal::class.java,
"type")

        .registerSubtype(YumHubMeal.NutritionIXMeal::class.java, "nutritionix")
        .registerSubtype(YumHubMeal.TheMealDBItem::class.java, "local_database")

    }

internal class NutritionAttrDeserializer : JsonSerializer<NutritionAttr> {
    override fun deserialize(
        json: JsonElement?,
        typeOfT: Type?,
        context: JsonDeserializationContext
    ): NutritionAttr? {
        println("${json?.toString()}")
        if (json == null)
            return null
        if (json.isJsonObject)
            return context.deserialize(json, NutritionAttrData::class.java)
        else
            return NutritionEnum.valueOf(json.asString)

    }
}

@file:OptIn(ExperimentalMaterial3Api::class, ExperimentalLayoutApi::class)

package com.alterjuice.onboarding.ui.screens

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.horizontalScroll
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.ExperimentalLayoutApi
import androidx.compose.foundation.layout.FlowRow
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.heightIn

```

```

import androidx.compose.foundation.layout.wrapContentWidth
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.ElevatedFilterChip
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FilledTonalButton
import androidx.compose.material3.FilterChipDefaults
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateListOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.text.SpanStyle
import androidx.compose.ui.text.buildAnnotatedString
import androidx.compose.ui.text.font.FontStyle
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.withStyle
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.alterjuice.compose_utils.ui.components.YumHubOutlinedCard
import
com.alterjuice.compose_utils.ui.components.YumHubOutlinedCardColumnWithCardTitle
import com.alterjuice.compose_utils.ui.extensions.rememberLambda
import com.alterjuice.domain.model.common.MealCategories
import com.alterjuice.domain.model.user.FitnessGoal
import com.alterjuice.domain.model.user.UserPAL
import com.alterjuice.domain.model.user.UserSex
import com.alterjuice.navigation.NavCommand.Companion.getDestination
import com.alterjuice.navigation.routes DefinedRoutes

```

```

import com.alterjuice.onboarding.viewmodels.OnboardingScreenController
import com.alterjuice.onboarding.viewmodels.OnboardingViewModel
import com.alterjuice.resources.R
import com.alterjuice.theming.theme.LocalAppNavController
import org.koin.androidx.compose.koinViewModel

@Composable
fun OnboardingScreen(
    modifier: Modifier,
    controller: OnboardingScreenController =
koinViewModel<OnboardingViewModel>()
) = YumHubOutlinedCard(
    modifier = modifier,
    backgroundAlpha = 0.5f,
    borderStroke = null,
    surfaceShape = RectangleShape
) {
    val navController = LocalAppNavController.current
    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState()),
        verticalArrangement = Arrangement.spacedBy(35.dp)
    ) {

        val uiState by controller.uiState.collectAsState()

        var userName by rememberSaveable {
            mutableStateOf(uiState.userInfo?.username.orEmpty())
        }
        var userSex by rememberSaveable {
            mutableStateOf(uiState.userInfo?.sex)
        }
        var userWeight by rememberSaveable {
            mutableStateOf(uiState.userInfo?.weight?.toString().orEmpty())
        }
        var userHeight by rememberSaveable {
            mutableStateOf(uiState.userInfo?.height?.toString().orEmpty())
        }
        var userAge by rememberSaveable {
            mutableStateOf(uiState.userInfo?.age?.toString().orEmpty())
        }
    }
}

```

```

var userPAL by rememberSaveable {
    mutableStateOf(uiState.userInfo?.pal ?: UserPAL.Sedentary)
}
var userFitnessGoal by rememberSaveable {
    mutableStateOf(uiState.userInfo?.fitnessGoal ?:
FitnessGoal.MaintainWeight)
}
val favouriteCategories = remember {
    mutableStateListOf<MealCategories>().also {
        it.addAll(uiState.userInfo?.favoriteCategories ?: emptyList())
    }
}

var userNameError by rememberSaveable { mutableStateOf(false) }
var userAgeError by rememberSaveable {
    mutableStateOf(
        userAge.let { (it.toIntOrNull() ?: 0) !in 1..120 }
    )
}
var userWeightError by rememberSaveable { mutableStateOf(false) }
var userHeightError by rememberSaveable { mutableStateOf(false) }

val onMealCategorySelect: (MealCategories) -> Unit = rememberLambda
{ category ->
    if (category in favouriteCategories) {
        favouriteCategories.remove(category)
    } else {
        favouriteCategories.add(category)
    }
    controller.updateUserFavouriteCategories(favouriteCategories)
}

val onSexChipChange: (UserSex) -> Unit = rememberLambda {
    val final = if (userSex == it) UserSex.UNSPECIFIED else it
    userSex = final
    controller.updateUserSex(final)
}
val onFitnessGoalChange: (FitnessGoal) -> Unit = rememberLambda {
    userFitnessGoal = it
    controller.updateUserFitnessGoal(it)
}

val onUserPALChange: (UserPAL) -> Unit = rememberLambda {

```

```

        userPAL = it
        controller.updateUserPAL(it)
    }
    val yumHubChipModifier = remember {
        Modifier
            .wrapContentWidth()
            .heightIn(60.dp)
    }

    YumHubOutlinedCard(
        modifier = Modifier,
        backgroundAlpha = 0.5f,
        contentPaddingValues = PaddingValues(20.dp)
    ) {
        Text(
            text = remember {
                buildAnnotatedString {
                    append("Welcome to ")
                    withStyle(style = SpanStyle(color = Color(0xFF91B545)))
                }

                append("Yum")
            }
            withStyle(style = SpanStyle(color = Color(0xFF597719)))
        ) {
            append("Hub")
        }
    },
    modifier = Modifier
        .fillMaxWidth()
        .align(Alignment.Center),
    color = Color.Unspecified,
    fontStyle = FontStyle(R.font.kyiv_type_sans_regular),
    style = MaterialTheme.typography.titleLarge,
    textAlign = TextAlign.Center
)
}

YumHubOutlinedCardColumnWithCardTitle(
    modifier = Modifier,
    title = remember { "Step 1: Personal information" },
    subTitle = remember { "*Your personal information is used to give
You the best recommendations" },
    backgroundAlpha = 0.5f,

```

```

        verticalArrangement = Arrangement.spacedBy(10.dp)
    ) {
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(10.dp)
        ) {

            YumHubOutlinedTextField(
                modifier = Modifier.weight(0.75f),
                value = userName,
                onChange = { input ->
                    userName = input
                    runCatching {
                        check(input.isNotEmpty())
                        input
                    }.onFailure {
                        userNameError = true
                    }.onSuccess { name ->
                        userNameError = false
                        controller.updateUserName(name)
                    }
                },
                label = { Text("Name") },
                isError = userNameError,
                keyboardOptions = KeyboardOptions(
                    autoCorrect = true,
                    keyboardType = KeyboardType.Text,
                    imeAction = ImeAction.Next,
                )
            )

            YumHubOutlinedTextField(
                modifier = Modifier.weight(0.25f),
                value = userAge,
                onChange = {
                    userAge = it
                    runCatching {
                        val age = if (it.isEmpty()) 0 else it.toInt()
                        check(age in 1..120)
                        age
                    }.onFailure {
                        userAgeError = true
                    }.onSuccess { age ->

```

```

        userAgeError = false
        controller.updateUserAge(age)
    }
},
label = { Text("Age") },
isError = userAgeError,
keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Decimal,
    imeAction = ImeAction.Next,
)
)
}

Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.spacedBy(10.dp)
) {

    YumHubChip(
        selected = userSex == UserSex.MALE,
        modifier = Modifier
            .weight(1f)
            .aspectRatio(3f),
        onClick = rememberLambda { onSexChipChange(UserSex.MALE) },
        label = { Text(text = "Male") },
    )

    YumHubChip(
        selected = userSex == UserSex.FEMALE,
        modifier = Modifier
            .weight(1f)
            .aspectRatio(3f),
        onClick = rememberLambda
{ onSexChipChange(UserSex.FEMALE) },
        label = { Text(text = "Female") }
    )
}

Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.spacedBy(10.dp)
) {
    YumHubOutlinedTextField(
        modifier = Modifier.weight(1f),

```

```

        value = userWeight,
        onChange = {
            userWeight = it
            runCatching {
                if (it.isEmpty()) 0f else it.toFloat()
            }.onFailure {
                userWeightError = true
            }.onSuccess { weight ->
                userWeightError = false
                controller.updateUserWeight(weight)
            }
        },
        label = { Text(remember { "Weight (kg)" }) },
        isError = userWeightError,
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Decimal,
            imeAction = ImeAction.Next,
        )
    )
    YumHubOutlinedTextField(
        modifier = Modifier.weight(1f),
        value = userHeight,
        onChange = {
            userHeight = it
            runCatching {
                if (it.isEmpty()) 0f else it.toFloat()
            }.onFailure {
                userHeightError = true
            }.onSuccess { height ->
                userHeightError = false
                controller.updateUserHeight(height)
            }
        },
        label = { Text(remember { "Height (cm)" }) },
        isError = userHeightError,
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Decimal,
            imeAction = ImeAction.Next,
        )
    )
}
}

```

```

YumHubOutlinedCardColumnWithCardTitle(
    modifier = Modifier,
    title = remember { "Step 2: Physical activity" },
    subTitle = remember { "*Your physical information helps us to give
you best recommendations" },
    backgroundAlpha = 0.5f,
    verticalArrangement = Arrangement.spacedBy(10.dp)
) {
    FlowRow(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.spacedBy(7.dp),
        verticalArrangement = Arrangement.spacedBy(7.dp)
    ) {
        UserPAL.values().forEach { pal ->
            YumHubChip(
                modifier = yumHubChipModifier,
                selected = userPAL == pal,
                onClick = rememberLambda(pal) { onUserPALChange(pal) },
                label = {
                    Column {
                        Text(text = pal.description)
                        if (pal.subDescription.isNotEmpty()) {
                            Text(
                                text = "*${pal.subDescription}",
                                style =
MaterialTheme.typography.labelMedium
                            )
                        }
                    }
                }
            )
        }
    }
}

YumHubOutlinedCardColumnWithCardTitle(
    modifier = Modifier,
    title = remember { "Step 3: Fitness goals" },
    subTitle = remember { "Just remind yourself about your goals :)" },
    backgroundAlpha = 0.5f,
    verticalArrangement = Arrangement.spacedBy(10.dp),
) {
    FlowRow(
        modifier = Modifier

```

```

        .fillMaxWidth()
        .horizontalScroll(rememberScrollState()),
        horizontalArrangement = Arrangement.spacedBy(7.dp),
        verticalArrangement = Arrangement.spacedBy(7.dp)
    ) {
        FitnessGoal.values().forEach { goal ->
            YumHubChip(
                selected = userFitnessGoal == goal,
                modifier = yumHubChipModifier,
                onClick = rememberLambda(goal)
            { onFitnessGoalChange(goal) },
                label = { Text(text = goal.description) }
            )
        }
    }
}

YumHubOutlinedCardColumnWithCardTitle(
    modifier = Modifier,
    title = remember { "Step 4: Favourite categories" },
    subTitle = remember { "Let us know your favourite food
categories" },
    backgroundAlpha = 0.5f,
    verticalArrangement = Arrangement.spacedBy(10.dp),
) {
    Text(
        text = "Cuisine",
        style = MaterialTheme.typography.titleLarge
    )
    FlowRow(
        modifier = Modifier
            .fillMaxWidth()
            .horizontalScroll(rememberScrollState()),
        horizontalArrangement = Arrangement.spacedBy(7.dp),
        verticalArrangement = Arrangement.spacedBy(7.dp)
    ) {
        MealCategories.cuisine?.forEach { cuisine ->
            YumHubChip(
                selected = cuisine in favouriteCategories,
                modifier = yumHubChipModifier,
                onClick = rememberLambda
            { onMealCategorySelect(cuisine) },

```

```

        label = { Text(text =
cuisine.name.removePrefix(MealCategories.Cuisine.name)) }
    )
}

HorizontalDivider(
    modifier = Modifier
        .fillMaxWidth(0.8f)
        .align(Alignment.CenterHorizontally),
    thickness = 2.dp,
    color = MaterialTheme.colorScheme.primary
)

Text(
    text = "Cook Type",
    style = MaterialTheme.typography.titleLarge
)

FlowRow(
    modifier = Modifier
        .fillMaxWidth()
        .horizontalScroll(rememberScrollState()),
    horizontalArrangement = Arrangement.spacedBy(7.dp),
    verticalArrangement = Arrangement.spacedBy(7.dp)
) {
    MealCategories.cookTypes?.forEach { cookType ->
        YumHubChip(
            selected = cookType in favouriteCategories,
            modifier = yumHubChipModifier,
            onClick = rememberLambda
{ onMealCategorySelect(cookType) },
            label = { Text(text =
cookType.name.removePrefix(MealCategories.CookType.name)) }
        )
    }
}

HorizontalDivider(
    modifier = Modifier
        .fillMaxWidth(0.8f)
        .align(Alignment.CenterHorizontally),
    thickness = 2.dp,

```

```

        color = MaterialTheme.colorScheme.primary
    )

    Text(
        text = "Flavor",
        style = MaterialTheme.typography.titleLarge
    )

    FlowRow(
        modifier = Modifier
            .fillMaxWidth()
            .horizontalScroll(rememberScrollState()),
        horizontalArrangement = Arrangement.spacedBy(7.dp),
        verticalArrangement = Arrangement.spacedBy(7.dp)
    ) {
        MealCategories.flavors?.forEach { flavor ->
            YumHubChip(
                selected = flavor in favouriteCategories,
                modifier = yumHubChipModifier,
                onClick = rememberLambda
            { onMealCategorySelect(flavor) },
                label = { Text(text =
flavor.name.removePrefix(MealCategories.Flavors.name)) }
            )
        }
    }

    HorizontalDivider(
        modifier = Modifier
            .fillMaxWidth(0.8f)
            .align(Alignment.CenterHorizontally),
        thickness = 2.dp,
        color = MaterialTheme.colorScheme.primary
    )

    Text(
        text = "Food Types",
        style = MaterialTheme.typography.titleLarge
    )

    FlowRow(
        modifier = Modifier
            .fillMaxWidth()

```

```

        .horizontalScroll(rememberScrollState()),
        horizontalArrangement = Arrangement.spacedBy(7.dp),
        verticalArrangement = Arrangement.spacedBy(7.dp)
    ) {
        MealCategories.types?.forEach { type ->
            YumHubChip(
                selected = type in favouriteCategories,
                modifier = yumHubChipModifier,
                onClick = rememberLambda { onMealCategorySelect(type) },
                label = { Text(text =
type.name.removePrefix(MealCategories.Type.name)) }
            )
        }
    }
}

val hasErrors = remember(userName, userAge, userWeight, userHeight) {
    if (userHeight.isEmpty()) {
        userHeightError = true
    }
    if (userAge.isEmpty() || (userAge.toIntOrNull() ?: 0) !in 1..120) {
        userAgeError = true
    }
    if (userWeight.isEmpty()) {
        userWeightError = true
    }
    if (userName.isEmpty()) {
        userNameError = true
    }
    userNameError || userAgeError || userWeightError || userHeightError
}

val buttonColor = if (hasErrors)
    MaterialTheme.colorScheme.error
else MaterialTheme.colorScheme.primary

FilledTonalButton(
    modifier = Modifier
        .fillMaxWidth(0.7f)
        .height(55.dp)
        .align(Alignment.CenterHorizontally),
    enabled = !hasErrors,
    onClick = {
        controller.finishOnboarding {

```

```

navController.navigate(DefinedRoutes.Dashboard.getDestination()) {
    popUpTo(0)
}
},
border = BorderStroke(1.dp, buttonColor)
) {
    Text(
        text = remember { "Save & continue" },
        style = MaterialTheme.typography.titleMedium
    )
}
}
}

@Composable
fun YumHubChip(
    modifier: Modifier,
    selected: Boolean,
    onClick: () -> Unit,
    label: @Composable () -> Unit,
) {
    ElevatedFilterChip(
        selected = selected,
        modifier = modifier,
        onClick = onClick,
        label = label,
        border = FilterChipDefaults.filterChipBorder(
            borderColor = MaterialTheme.colorScheme.secondary,
            selectedBorderColor = MaterialTheme.colorScheme.primary,
            borderWidth = 1.dp,
            selectedBorderWidth = 2.dp
        )
    )
}

@Composable
fun YumHubOutlinedTextField(
    modifier: Modifier,
    value: String,

```

```

        onChange: (String) -> Unit,
        label: @Composable () -> Unit = {},
        singleLine: Boolean = true,
        isError: Boolean = false,
        keyboardOptions: KeyboardOptions = KeyboardOptions.Default
    ) {
        OutlinedTextField(
            modifier = modifier,
            value = value,
            onChange = onChange,
            enabled = true,
            readOnly = false,
            label = label,
            singleLine = singleLine,
            isError = isError,
            maxLines = 1,
            keyboardOptions = keyboardOptions
        )
    }
}

```

```
@Composable
```

```
@Preview
```

```
private fun OnboardingScreenPreview() {
```

```
    OnboardingScreen(
```

```
        modifier = Modifier
```

```
    )
```

```
}
```

```
package com.alterjuice.compose_utils.ui.components
```

```
import androidx.compose.foundation.background
```

```
import androidx.compose.foundation.border
```

```
import androidx.compose.foundation.layout.Arrangement
```

```
import androidx.compose.foundation.layout.Column
```

```
import androidx.compose.foundation.layout.Row
```

```
import androidx.compose.foundation.layout.fillMaxWidth
```

```
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.layout.size
```

```
import androidx.compose.material3.Icon
```

```
import androidx.compose.material3.MaterialTheme
```

```
import androidx.compose.material3.Text
```

```
import androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.remember
```

```
import androidx.compose.ui.Alignment
```

```

import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.alterjuice.resources.R
import com.alterjuice.utils.WaterMLFormatter

val DefaultDropValue = 200

@Composable
fun WaterBalanceWithLeadingIconComponent(
    waterConsumed: Int?
) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .background(
                color = MaterialTheme.colorScheme.surfaceVariant,
                shape = MaterialTheme.shapes.medium
            )
            .border(
                width = 2.dp,
                color = MaterialTheme.colorScheme.primary,
                shape = MaterialTheme.shapes.medium
            )
            .padding(10.dp),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(10.dp, Alignment.Start)
    ) {
        Icon(
            modifier = Modifier.size(30.dp),
            tint = Color.Unspecified,
            painter = painterResource(id = R.drawable.water_drop_filled),
            contentDescription = null,
        )

        Text(
            text = remember(waterConsumed) {
                val balanceTransformed = waterConsumed?.toString() ?: "-"
                "Balance: ${balanceTransformed}ML"
            },

```

```

        style = MaterialTheme.typography.bodyLarge
    )
}

@Composable
fun WaterBalanceComponent(
    waterConsumed: Int?,
    waterRecommended: Int? = null,
    showWaterDropContent: Boolean = true,
) {
    Column(
        modifier = Modifier.fillMaxWidth()
            .background(
                color = MaterialTheme.colorScheme.surfaceVariant,
                shape = MaterialTheme.shapes.medium
            )
            .border(
                width = 2.dp,
                color = MaterialTheme.colorScheme.primary,
                shape = MaterialTheme.shapes.medium
            )
            .padding(10.dp)
    ) {
        Row(
            modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = remember(waterConsumed) {
                    val balanceTransformed = waterConsumed?.let {
                        WaterMLFormatter.mlToLiters(it)
                    } ?: "-"
                    "Water balance: $balanceTransformed"
                },
                style = MaterialTheme.typography.bodyLarge
            )
            if (showWaterDropContent) {
                Row(
                    modifier = Modifier,
                    verticalAlignment = Alignment.CenterVertically,
                ) {

```

```

        Icon(
            modifier = Modifier.size(30.dp),
            tint = Color.Unspecified,
            painter = painterResource(id =
R.drawable.water_drop_filled),
            contentDescription = null,
        )
        Text(
            text = remember { " = ${DefaultDropValue}ML" },
            style = MaterialTheme.typography.bodyLarge
        )
    }
}

waterRecommended?.let { recommended ->
    Text(
        text = remember { "Personal water consumption recommendations:
${WaterMLFormatter.mlToLiters(recommended)}" },
        style = MaterialTheme.typography.labelSmall
    )
}
}

@Preview
@Composable
fun WaterBalanceWithLeadingIconComponentEmptyPreview() {
    WaterBalanceWithLeadingIconComponent(null)
}

@Preview
@Composable
fun WaterBalanceWithLeadingIconComponentPreview() {
    WaterBalanceWithLeadingIconComponent(1400)
}

@Preview
@Composable
fun WaterBalanceComponentWithIconPreview() {
    WaterBalanceComponent(1400)
}

```

```

@Preview
@Composable
fun WaterBalanceComponentWithoutIconPreview() {
    WaterBalanceComponent(1400, showWaterDropContent = false)
}

package com.alterjuice.onboarding.viewmodels

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.alterjuice.android_utils.update
import com.alterjuice.domain.model.common.MealCategories
import com.alterjuice.domain.model.user.FitnessGoal
import com.alterjuice.domain.model.user.UserInfoData
import com.alterjuice.domain.model.user.UserInfoDataNullable
import com.alterjuice.domain.model.user.UserInfoNullable
import com.alterjuice.domain.model.user.UserMeasurements
import com.alterjuice.domain.model.user.UserPAL
import com.alterjuice.domain.model.user.UserSex
import com.alterjuice.domain.repository.UserMeasurementsRepository
import com.alterjuice.repository.storage.OnboardingStorage
import com.alterjuice.repository.storage.UserInfoStorage
import com.alterjuice.utils.DateTimeUtils
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

data class OnboardingScreenUIState(
    val userInfo: UserInfoNullable? = null
)

interface OnboardingScreenController {
    val uiState: StateFlow<OnboardingScreenUIState>

    fun updateUserName(username: String)
    fun updateUserSex(userSex: UserSex)
    fun updateUserWeight(userWeight: Float)
    fun updateUserHeight(userHeight: Float)
    fun updateUserAge(userAge: Int)
    fun updateUserPAL(userPAL: UserPAL)
    fun updateUserFitnessGoal(fitnessGoal: FitnessGoal)

```

```

fun updateUserFavouriteCategories(categories: List<MealCategories>)

fun finishOnboarding(
    onSuccess: () -> Unit
)
}

class OnboardingViewModel(
    private val onboardingStorage: OnboardingStorage,
    private val userInfoStorage: UserInfoStorage,
    private val userMeasurementsRepository: UserMeasurementsRepository,
): ViewModel(), OnboardingScreenController {
    override val uiState = MutableStateFlow(OnboardingScreenUIState(
        userInfo = onboardingStorage temporaryUserInfo.get()
    ))

    override fun updateUserName(username: String) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(username = username)
        }
    }

    override fun updateUserSex(userSex: UserSex) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(sex = userSex)
        }
    }

    override fun updateUserWeight(userWeight: Float) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(weight = userWeight)
        }
    }

    override fun updateUserHeight(userHeight: Float) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(height = userHeight)
        }
    }

    override fun updateUserAge(userAge: Int) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(age = userAge)
        }
    }
}

```

```

        }
    }

    override fun updateUserPAL(userPAL: UserPAL) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(pal = userPAL)
        }
    }

    override fun updateUserFitnessGoal(fitnessGoal: FitnessGoal) {
        onboardingStorage temporaryUserInfo.update {
            it.copy(fitnessGoal = fitnessGoal)
        }
    }

    override fun updateUserFavouriteCategories(categories: List<MealCategories>)
{
        onboardingStorage temporaryUserInfo.update {
            it.copy(favoriteCategories = categories)
        }
    }

    override fun finishOnboarding(
        onSuccess: () -> Unit
    ) {
        viewModelScope.launch(Dispatchers.IO) {
            runCatching {
                val temporary = onboardingStorage temporaryUserInfo.get()

                userInfoStorage.userInfo.set(
                    UserInfoData(
                        username = temporary.username!!,
                        weight = temporary.weight!!,
                        height = temporary.height!!,
                        age = temporary.age!!,
                        pal = temporary.pal!!,
                        sex = temporary.sex!!,
                        fitnessGoal = temporary.fitnessGoal!!,
                        favoriteCategories = temporary.favoriteCategories!!
                    )
                )
                onboardingStorage.isOnboardingPassed.set(true)
            }
        }
    }
}

```

```
onboardingStorage temporaryUserInfo.set (UserInfoDataNullable())
userMeasurementsRepository.addMeasurement(
    measurement = UserMeasurements.Weight(
        dayTimestampSec =
DateTimeUtils.getNormalizedStartDateTodaySec(),
        weight = temporary.weight!!
    )
)
}.onSuccess {
    withContext(Dispatchers.Main) {
        onSuccess()
    }
}
}
}
```

```
package com.alterjuice.onboarding
```

```
import com.alterjuice.android_utils.DIModulesHub
import com.alterjuice.domain.model.user.UserMeasurements
import com.alterjuice.domain.repository.UserMeasurementsRepository
import com.alterjuice.onboarding.viewmodels.OnboardingViewModel
import com.alterjuice.repository.storage.OnboardingStorage
import com.alterjuice.repository.storage.UserInfoStorage
import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.dsl.module
```

```
object OnboardingDI : DIModulesHub {
    override fun modules() = arrayOf(
        viewModelModule()
    )

    private fun viewModelModule() = module {
        viewModel<OnboardingViewModel> {
            OnboardingViewModel(
                onboardingStorage = get<OnboardingStorage>(),
                userInfoStorage = get<UserInfoStorage>(),
                userMeasurementsRepository = get<UserMeasurementsRepository>()
            )
        }
    }
}
```

```

package com.alterjuice.android_utils

import org.koin.core.module.Module

interface DIModulesHub {
    fun modules(): Array<Module>
}

package com.alterjuice.utils

import com.alterjuice.utils.extensions.divMod
import com.alterjuice.utils.extensions.roundTo

object WaterMLFormatter {
    fun mlToLiters(waterMls: Long): String {
        val (liters, mls) = waterMls.divMod(1000)
        if (liters != 0L && mls != 0L) {
            return "${waterMls.div(1000.0).roundTo(1)}L"
        }
        val result = buildList<String> {
            if (liters != 0L) add("${liters}L")
            if (liters == 0L) add("${mls}ML")
        }.joinToString(" ").ifEmpty { "-" }

        return result
    }
    fun mlToLiters(waterMls: Int): String {
        return mlToLiters(waterMls.toLong())
    }
}

package com.alterjuice.compose_utils.ui.components

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.BoxScope
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.ColumnScope
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height

```

```

import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Shape
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp

@Composable
fun YumHubOutlinedCard(
    modifier: Modifier,
    backgroundAlpha: Float = 1f,
    color: Color = MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
backgroundAlpha),
    contentPaddingValues: PaddingValues = PaddingValues(16.dp),
    borderStroke: BorderStroke? = BorderStroke(2.dp,
MaterialTheme.colorScheme.primary),
    surfaceShape: Shape = MaterialTheme.shapes.medium,
    content: @Composable BoxScope.() -> Unit,
) {
    Surface(
        modifier = modifier, //e.padding(paddingValues),
        color = color,
        border = borderStroke,
        shape = surfaceShape,
        content = {
            Box(
                modifier = Modifier.padding(contentPaddingValues),
                content = content
            )
        }
    )
}

@Composable
fun YumHubOutlinedCardColumn(
    modifier: Modifier,

```

```

        backgroundAlpha: Float = 1f,
        color: Color = MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
backgroundAlpha),
        contentPaddingValues: PaddingValues = PaddingValues(16.dp),
        borderStroke: BorderStroke? = BorderStroke(2.dp,
MaterialTheme.colorScheme.primary),
        surfaceShape: Shape = MaterialTheme.shapes.medium,
        verticalArrangement: Arrangement.Vertical = Arrangement.Top,
        horizontalAlignment: Alignment.Horizontal = Alignment.Start,
        content: @Composable ColumnScope.() -> Unit,
    ) {
        YumHubOutlinedCard(
            modifier = modifier,
            backgroundAlpha = backgroundAlpha,
            color = color,
            contentPaddingValues = contentPaddingValues,
            borderStroke = borderStroke,
            surfaceShape = surfaceShape,
        ) {
            Column(
                verticalArrangement = verticalArrangement,
                horizontalAlignment = horizontalAlignment
            ) {
                content()
            }
        }
    }
}

```

@Composable

```

fun YumHubOutlinedCardColumnWithCardTitle(
    modifier: Modifier,
    title: String,
    subTitle: String? = null,
    backgroundAlpha: Float = 1f,
    color: Color = MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
backgroundAlpha),
    contentPaddingValues: PaddingValues = PaddingValues(16.dp),
    borderStroke: BorderStroke? = BorderStroke(2.dp,
MaterialTheme.colorScheme.primary),
    surfaceShape: Shape = MaterialTheme.shapes.medium,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable ColumnScope.() -> Unit,

```

```

) {
    YumHubOutlinedCardColumn(
        modifier = modifier,
        backgroundAlpha = backgroundAlpha,
        verticalArrangement = verticalArrangement,
        horizontalAlignment = horizontalAlignment,
        surfaceShape = surfaceShape,
        borderStroke = borderStroke,
        contentPaddingValues = contentPaddingValues,
        color = color,
    ) {
        YumHubOutlinedCardColumn(
            modifier = Modifier.fillMaxWidth(),
            verticalArrangement = Arrangement.spacedBy(10.dp)
        ) {
            Text(
                text = remember(title) { title },
                style = MaterialTheme.typography.titleLarge
            )
            subTitle?.let {
                Text(
                    text = remember(subTitle) { subTitle },
                    style = MaterialTheme.typography.titleSmall
                )
            }
        }
        content()
    }
}

@Composable
@Preview
private fun YumHubOutlinedCardPreview() {
    YumHubOutlinedCard(
        modifier = Modifier
            .fillMaxWidth()
            .height(200.dp)
    ) {}
}

package com.alterjuice.domain.model.common

import com.alterjuice.domain.model.nutrition.NutrientsItem

```

```

import com.alterjuice.domain.model.nutrition.NutritionAttr
import com.alterjuice.domain.model.nutrition.NutritionEnum
import com.alterjuice.domain.model.nutrition.by
import com.alterjuice.domain.model.nutrition.toNutrients
import com.alterjuice.domain.model.nutrition.valueOrZero
import com.alterjuice.utils.extensions.gracefulRound
import com.alterjuice.utils.extensions.toHex
import java.security.MessageDigest

data class CookInfo(
    val instructions: String? = null,
    val prepareTime: Long? = null,
    val cookTime: Long? = null,
    val waitTime: Long? = null,
    val ingredients: List<String> = emptyList(),
)

data class Serving(
    val servingQty: Double?,
    val servingUnit: String?,
    val servingWeightGrams: Double?,
)

data class Branded(
    val brandName: String? = null,
    val nixBrandName: String? = null
)

sealed class YumHubMeal(val type: String): MainNutrients<NutrientsItem?> {
    abstract val id: String
    abstract val foodName: String
    abstract val nutrients: List<NutrientsItem>
    abstract val calories: Double
    abstract val cookInfo: CookInfo?
    abstract val servings: List<Serving>
    abstract val servingsCount: Double
    abstract val branded: Branded?
    abstract val categoriesTags: List<MealCategories>

    @delegate:Transient

```

```

    val hash by lazy {
        MessageDigest
            .getInstance("MD5")
            .digest((id + foodName).toString().toByteArray()).toHex()
    }

    @delegate:Transient override val fiber by lazy
    { nutrientBy(NutritionEnum.Fiber) }

    @delegate:Transient override val carbs by lazy
    { nutrientBy(NutritionEnum.Carbs) }

    @delegate:Transient override val protein by lazy
    { nutrientBy(NutritionEnum.Protein) }

    @delegate:Transient override val fat by lazy
    { nutrientBy(NutritionEnum.Fat) }

    @delegate:Transient override val fatSat by lazy
    { nutrientBy(NutritionEnum.FatSat) }

    @delegate:Transient override val sugars by lazy
    { nutrientBy(NutritionEnum.Sugars) }

    @delegate:Transient override val sugarsAdded by lazy
    { nutrientBy(NutritionEnum.SugarsAdded) }

    @delegate:Transient override val sodiumNa by lazy
    { nutrientBy(NutritionEnum.SodiumNa) }

    @delegate:Transient override val cholesterol by lazy
    { nutrientBy(NutritionEnum.Cholesterol) }

    @delegate:Transient override val energyCalories by lazy
    { nutrientBy(NutritionEnum.EnergyCalories) }

    @delegate:Transient override val iron by lazy
    { nutrientBy(NutritionEnum.Iron) }

    @delegate:Transient override val potassium by lazy
    { nutrientBy(NutritionEnum.Potassium) }

    @delegate:Transient override val calcium by lazy
    { nutrientBy(NutritionEnum.Calcium) }

    @delegate:Transient override val vitaminD by lazy
    { nutrientBy(NutritionEnum.VitaminD) }

    @delegate:Transient override val caffeine by lazy
    { nutrientBy(NutritionEnum.Caffeine) }

    override val mainNutrients: List<NutrientsItem?> get() = super.mainNutrients

    fun withOneServing(nutrition: NutritionAttr): Double {
        return withServings(1.0, nutrition)
    }

```

```

fun withServings(servingsCount: Double, nutrition: NutritionAttr): Double {
    val nutrient = nutrientBy(nutrition)
    val nutrientValue = nutrient.valueOrNull
    val oneServingNutrients = nutrientValue / this.servingsCount
    return servingsCount * oneServingNutrients
}

fun withServingsWithUnits(
    servingsCount: Double,
    nutrition: NutritionAttr,
): String {
    return withServingsTransform(servingsCount, nutrition) { attr, value ->
        "${value.gracefulRound()} ${attr.unit}"
    }
}

fun withServingsTransform(
    servingsCount: Double,
    nutrition: NutritionAttr,
    transform: (NutritionAttr, value: Double) -> String
): String {
    return transform(nutrition, withServings(servingsCount, nutrition))
}

data class NutritionIXMeal(
    override val id: String,
    override val foodName: String,
    override val nutrients: List<NutrientsItem> = emptyList(),
    override val calories: Double,
    override val cookInfo: CookInfo?,
    override val servings: List<Serving> = emptyList(),
    override val servingsCount: Double,
    override val branded: Branded?,
    override val categoriesTags: List<MealCategories> = emptyList(),
): YumHubMeal("nutritionix") // https://www.nutritionix.com/food/bacon

data class TheMealDBItem(
    override val id: String,
    override val foodName: String,
    override val nutrients: List<NutrientsItem> = emptyList(),
    override val calories: Double,
    override val cookInfo: CookInfo,

```

```

        override val servings: List<Serving> = emptyList(),
        override val servingsCount: Double,
        override val branded: Branded?,
        override val categoriesTags: List<MealCategories> = emptyList(),
    ): YumHubMeal("local_database")

fun getDescription() = buildString {
    appendLine(foodName).appendLine().appendLine()
    appendLine(cookInfo?.ingredients?.joinToString("\n"))
    appendLine().appendLine().appendLine(cookInfo?.instructions)
}

fun nutrientBy(nutrient: NutritionAttr): NutrientsItem? {
    return this.nutrients.by(nutrient)
}

companion object {

    fun empty(name: String) = YumHubMeal.TheMealDBItem(
        id = "Empty${System.currentTimeMillis()}",
        foodName = name,
        nutrients = baseNutrientsFrom(),
        calories = 0.0,
        cookInfo = CookInfo(
            instructions = null,
            prepareTime = null,
            cookTime = null,
            waitTime = null,
            ingredients = emptyList()
        ),
        servings = emptyList(),
        branded = null,
        categoriesTags = emptyList(),
        servingsCount = 1.0,
    )

    fun baseNutrientsFrom(
        fiber: Number? = null,
        carbs: Number? = null,
        protein: Number? = null,
        fat: Number? = null,
        satfat: Number? = null,
    )

```

```

        sugar: Number? = null,
        sodium: Number? = null,
        cholesterol: Number? = null,
        calories: Number? = null,
    ): List<NutrientsItem> = listOfNotNull(
        NutritionEnum.Fiber to fiber,
        NutritionEnum.Carbs to carbs,
        NutritionEnum.Protein to protein,
        NutritionEnum.Fat to fat,
        NutritionEnum.FatSat to satfat,
        NutritionEnum.Sugars to sugar,
        NutritionEnum.SodiumNa to sodium,
        NutritionEnum.Cholesterol to cholesterol,
        NutritionEnum.EnergyCalories to calories
    ).toNutrients()
}
}

```

```
package com.alterjuice.domain.model.nutrition
```

```

interface NutritionAttr {
    val attrID: Int
    val usdaTag: String
    val nutritionName: String
    val unit: String
    val importanceKey: Int get() = 999
}

```

```

data class NutritionAttrData(
    override val attrID: Int,
    override val usdaTag: String,
    override val nutritionName: String,
    override val unit: String,
    override val importanceKey: Int
): NutritionAttr

```

```
package com.alterjuice.domain.model.nutrition
```

```

enum class NutritionEnum(
    override val attrID: Int,
    override val usdaTag: String,
    override val nutritionName: String,
    override val unit: String,

```

```

        override val importanceKey: Int = 999,
    ): NutritionAttr {
        EnergyCalories(208, "ENERC_KCAL", "Energy", "kcal", 1),
        EnergyKj(268, "ENERC_KJ", "Energy", "kJ", 1),
        Protein(203, "PROCNT", "Protein", "g", 2),
        Fat(204, "FAT", "Total lipid (fat)", "g", 3),
        FatSat(606, "FASAT", "Fatty acids (saturated)", "g", 4),
        Carbs( 205, "CHOCDF", "Carbohydrate", "g", 5),
        Calcium(301, "CA", "Calcium, Ca", "mg", 6),
        Sugars(269, "SUGAR", "Sugars (total)", "g", 7),
        SugarsAdded(539, "SUGAR_ADD", "Sugars (added)", "g", 8),
        SugarAlc(299, "SUGAR_ALC", "Sugar Alcohol", "g", 9),
        SodiumNa(307, "NA", "Sodium", "mg", 10),
        Iron(303, "FE", "Iron (Fe)", "mg", 11),
        Fiber(291, "FIBTG", "Fiber", "g", 12),
        Caffeine(262, "CAFFN", "Caffeine", "mg", 13),
        Alcohol(221, "ALC", "Alcohol, ethyl", "g", 15),
        Cholesterol(601, "CHOLE", "Cholesterol", "mg"),
        Item605(605, "FATRN", "Fatty acids", "g"),
        Potassium(306, "K", "Potassium", "mg"),
        VitaminD(324, "VITD", "Vitamin D", "IU", 20),
        VitaminD2(325, "ERGCAL", "Vitamin D2", "Âµg", 21),
        VitaminD3(326, "CHOCAL", "Vitamin D3 (cholecalciferol)", "Âµg", 22),
        Item261(261, "SORB", "Sorbitol", "g"),
        Item207(207, "ASH", "Ash", "g"),
        Item514(514, "ASP_G", "Aspartic acid", "g"),
        Item322(322, "CARTA", "Carotene, (alpha)", "Âµg"),
        Item321(321, "CARTB", "Carotene (beta)", "Âµg"),
        Item421(421, "CHOLN", "Choline", "mg"),
        Glycerin(1002, "N/A", "Glycerin", "g");

        companion object {
            val map by lazy { values().associateBy { it.attrID } }
        }
    }
}

package com.alterjuice.domain.model.user

enum class FitnessGoal(
    val description: String,
) {
    LoseWeight("Lose weight"),
    MaintainWeight("Maintain weight"),

```

```

BuildMuscle("Build muscle");

companion object {
    fun getByName(name: String?): FitnessGoal? {
        return values().find { it.name.equals(name, true) }
    }
}

interface UserInfoNullable {
    val favoriteCategories: List<MealCategories>?
    val username: String?
    val fitnessGoal: FitnessGoal? // Assume FitnessGoal is an enum indicating
user's goal
    val weight: Float?
    val height: Float?
    val age: Int?
    val sex: UserSex?
    val pal: UserPAL?
}

interface UserInfo: UserInfoNullable {
    override val favoriteCategories: List<MealCategories>
    override val username: String
    override val fitnessGoal: FitnessGoal // Assume FitnessGoal is an enum
indicating user's goal
    override val weight: Float
    override val height: Float
    override val age: Int
    override val sex: UserSex
    override val pal: UserPAL
}

data class UserInfoDataNullable(
    override val username: String? = null,
    override val weight: Float? = null,
    override val height: Float? = null,
    override val age: Int? = null,
    override val sex: UserSex? = null,
    override val pal: UserPAL? = null,
    override val fitnessGoal: FitnessGoal? = null,
    override val favoriteCategories: List<MealCategories>? = null,

```

```
) : UserInfoNullable
```

```
data class UserInfoData(
    override val username: String,
    override val weight: Float,
    override val height: Float,
    override val age: Int,
    override val sex: UserSex = UserSex.UNSPECIFIED,
    override val pal: UserPAL = UserPAL.Sedentary,
    override val fitnessGoal: FitnessGoal = FitnessGoal.MaintainWeight,
    override val favoriteCategories: List<MealCategories> = emptyList(),
) : UserInfo
```

```
package com.alterjuice.domain.model.user
```

```
import com.alterjuice.domain.model.MeasurementTypes
```

```
sealed interface UserMeasurements {
    val type: MeasurementTypes
    val dayTimestampSec: Long
    fun getValue(): Float

    data class Height(
        override val dayTimestampSec: Long,
        val height: Float
    ) : UserMeasurements {
        override val type = MeasurementTypes.HEIGHT
        override fun getValue(): Float = height
    }

    data class Weight(
        override val dayTimestampSec: Long,
        val weight: Float
    ) : UserMeasurements {
        override val type = MeasurementTypes.WEIGHT
        override fun getValue(): Float = weight
    }

    data class WaterBalance(
        override val dayTimestampSec: Long,
        val balanceML: Int
    ) : UserMeasurements {
        override val type = MeasurementTypes.WATER_BALANCE
    }
}
```

```

        override fun getValue(): Float = balanceML.toFloat()
    }

    fun asHeight() = this as? Height
    fun asWeight() = this as? Weight
    fun asWaterBalance() = this as? WaterBalance
}

package com.alterjuice.domain.model.common

import com.alterjuice.domain.model.nutrition.NutritionEnum

interface MainNutrients<T> {
    val fiber: T
    val carbs: T
    val protein: T
    val fat: T
    val fatSat: T
    val sugars: T
    val sugarsAdded: T
    val sodiumNa: T
    val cholesterol: T
    val energyCalories: T
    val iron: T
    val potassium: T
    val calcium: T
    val vitaminD: T
    val caffeine: T

    val mainNutrients get() = listOf(
        energyCalories, fiber, carbs, protein,
        fat, fatSat, sugars, sugarsAdded, sodiumNa, iron, cholesterol,
        potassium, calcium, vitaminD, caffeine
    )
}

fun <T> MainNutrients<T>.by(nutrient: NutritionEnum): T? {
    return when (nutrient) {
        NutritionEnum.EnergyCalories -> energyCalories
        NutritionEnum.Protein -> protein
        NutritionEnum.Fat -> fat
        NutritionEnum.FatSat -> fatSat
    }
}

```

```

        NutritionEnum.Carbs -> carbs
        NutritionEnum.Calcium -> calcium
        NutritionEnum.Sugars -> sugars
        NutritionEnum.SugarsAdded -> sugarsAdded
        NutritionEnum.SodiumNa -> sodiumNa
        NutritionEnum.Iron -> iron
        NutritionEnum.Fiber -> fiber
        NutritionEnum.Caffeine -> caffeine
        NutritionEnum.Cholesterol -> cholesterol
        NutritionEnum.Potassium -> potassium
        NutritionEnum.VitaminD -> vitaminD
        else -> null
    }
}

package com.alterjuice.domain.model.common

enum class MealCategories {

    Cuisine,
    CuisineItalian, CuisineMexican, CuisineAsian, CuisineIndian, CuisineGreek,
    CuisineThai,

    DietaryPreferences,
    DietaryPreferencesVegetarian, DietaryPreferencesVegan,
    DietaryPreferencesGlutenFree,
    DietaryPreferencesLowCarb, DietaryPreferencesKeto, DietaryPreferencesPaleo,

    Diets,
    DietsDiabeticFriendly, DietsLowCalorie, DietsLowFat,

    CookType,
    CookTypeBoiled, CookTypeFried, CookTypeReadyToEat, CookTypeGrilled,
    CookTypeBaked,

    Flavors,
    FlavorsSpicy, FlavorsSweet, FlavorsSavory, FlavorsSour,

    Occasions,
    OccasionsParty, OccasionsHoliday, OccasionsPicnic, OccasionsBarbecue,

    Type,

```

```

    TypeMeat, TypeSeafood, TypeChicken, TypeBeef, TypePizza, TypePasta,
    TypeSalad, TypeSoup,
    TypeSandwich, TypeFastFood, TypeFruits, TypeSweets, TypeDessert, TypeSnack,
    TypeChocolate, TypePie, TypePanini, TypeSauce, TypeVegetables, TypeMuffins,
    TypeDrink,

    Serving,
    ServingMain, ServingDessert, ServingSecond, ServingSides, ServingComponents,
    ServingDressing,

    MealType,
    MealTypeBreakfast, MealTypeDinner, MealTypeSnack, MealTypeLunch,
    MealTypeSupper,
    MealTypeDessert, MealTypeAppetizer,

    Category,
    CategoryBeef, CategoryBreakfast, CategoryChicken, CategoryDessert,
    CategoryGoat,
    CategoryLamb, CategoryMiscellaneous, CategoryPast, CategoryPork,
    CategorySeafood,
    CategorySide, CategoryStarter, CategoryVegan, CategoryVegetarian,
    CategoryDressing,

;
fun isSpecifiedCuisine() = this.name.startsWith(Cuisine.name)
fun isDietaryPreferences() = this.name.startsWith(DietaryPreferences.name)
fun isSpecifiedDiets() = this.name.startsWith(Diets.name)
fun isSpecifiedCookTypes() = this.name.startsWith(CookType.name)
fun isSpecifiedFlavors() = this.name.startsWith(Flavors.name)
fun isSpecifiedOccasions() = this.name.startsWith(Occasions.name)
fun isSpecifiedType() = this.name.startsWith(Type.name)
fun isSpecifiedServing() = this.name.startsWith(Serving.name)
fun isSpecifiedMealTypes() = this.name.startsWith(MealType.name)
fun isSpecifiedCategories() = this.name.startsWith(Category.name)

companion object {
    private val titles by lazy {
        listOf(
            Cuisine, DietaryPreferences, Diets,
            CookType, Flavors, Occasions,
            Type, Serving, MealType, Category,
        )
    }
}

```

```

    }

    val mealCategoriesMap by lazy {
        titles.associateWith { key -> values().filter
{ it.name.startsWith(key.name) && it.name.length > key.name.length } }
    }

    val cuisine get() = mealCategoriesMap[Cuisine]
    val dietaryPreferences get() = mealCategoriesMap[DietaryPreferences]
    val diets get() = mealCategoriesMap[Diets]
    val cookTypes get() = mealCategoriesMap[CookType]
    val flavors get() = mealCategoriesMap[Flavors]
    val occasions get() = mealCategoriesMap[Occasions]
    val types get() = mealCategoriesMap[Type]
    val serving get() = mealCategoriesMap[Serving]
    val mealTypes get() = mealCategoriesMap[MealType]
    val categories get() = mealCategoriesMap[Category]

    fun findTagsByContent(tag: String) = values().filter {
        it.name.contains(tag, ignoreCase = true)
    }
}

}

```

ДОДАТОК В
Слайди презентації

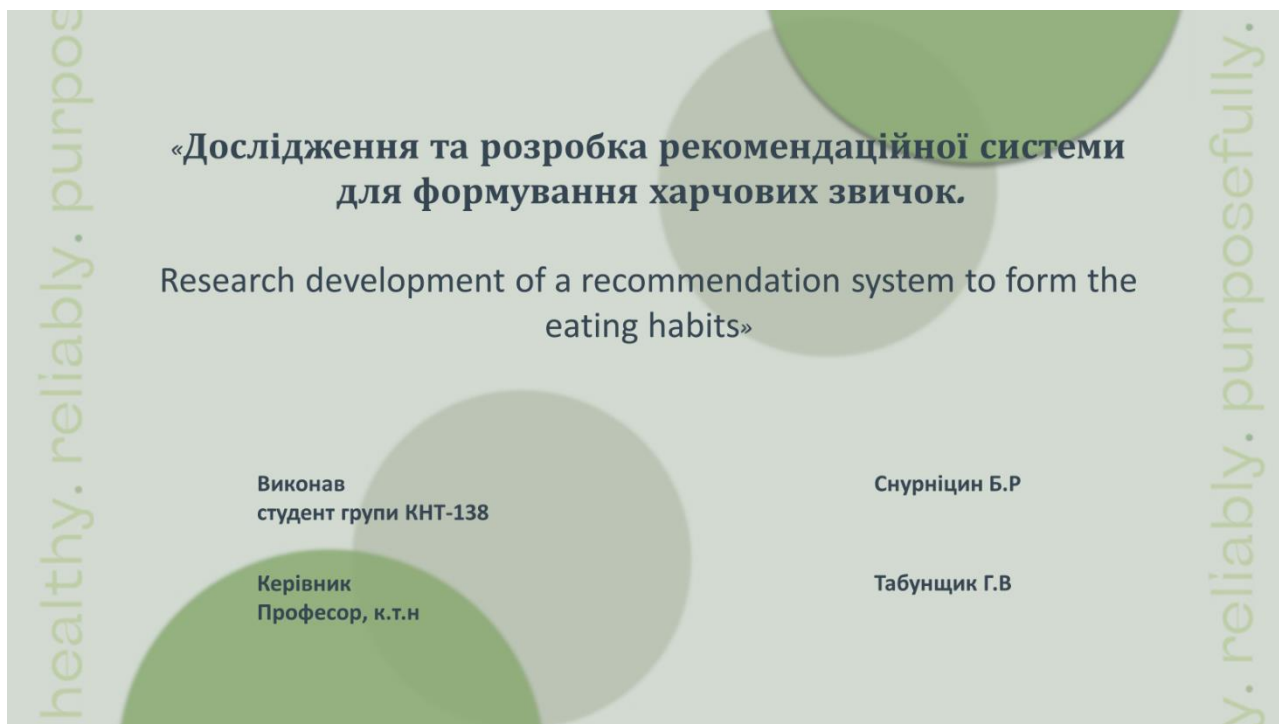


Рисунок В.1 – Слайд 1

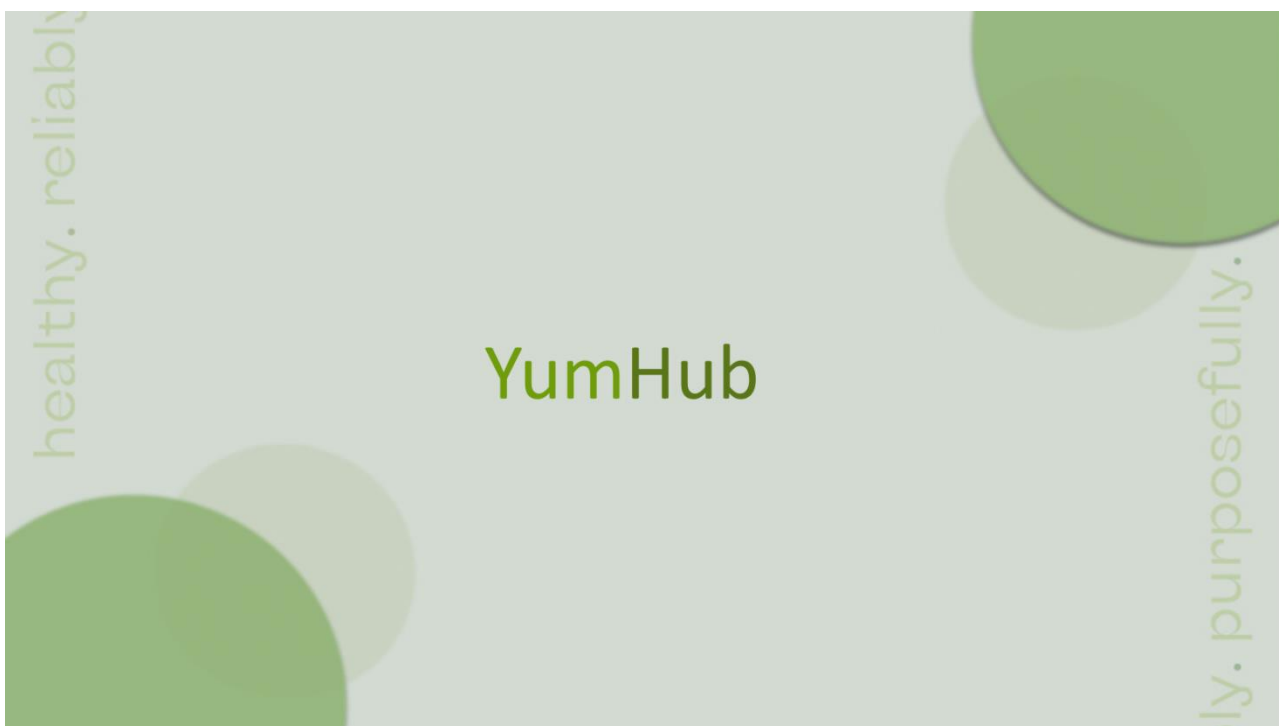
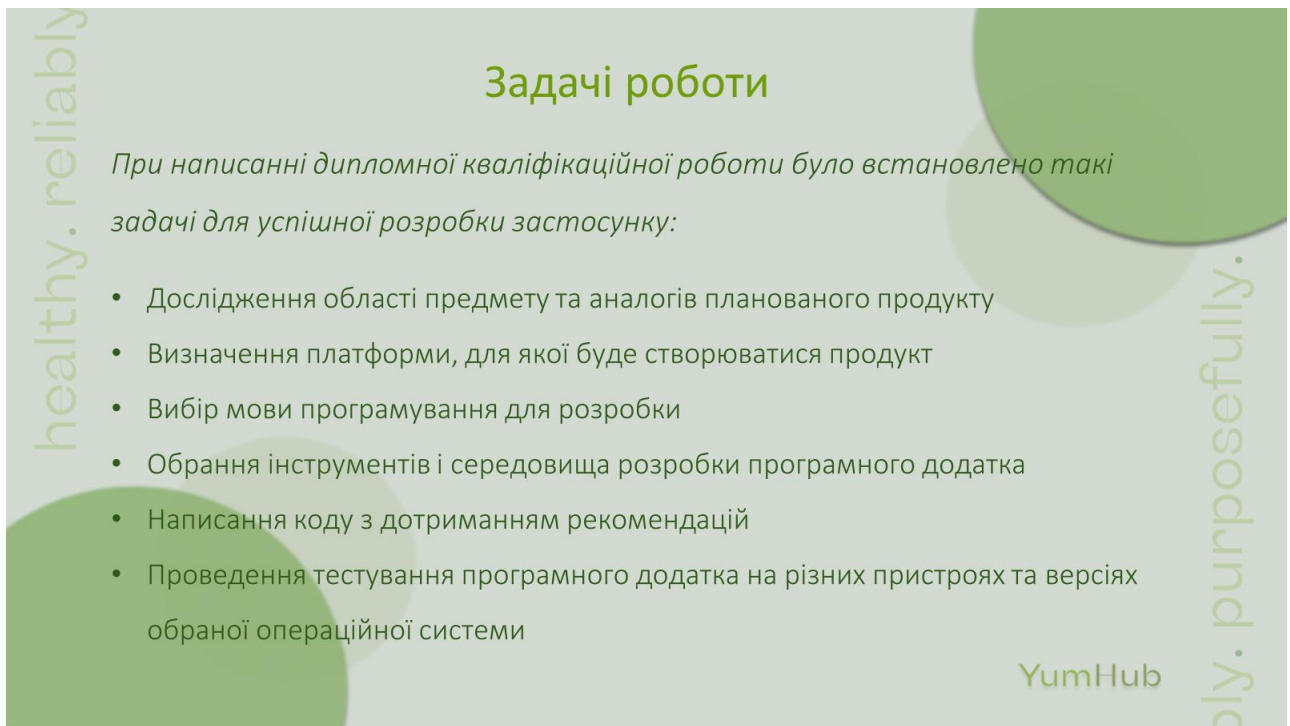


Рисунок В.2 – Слайд 2

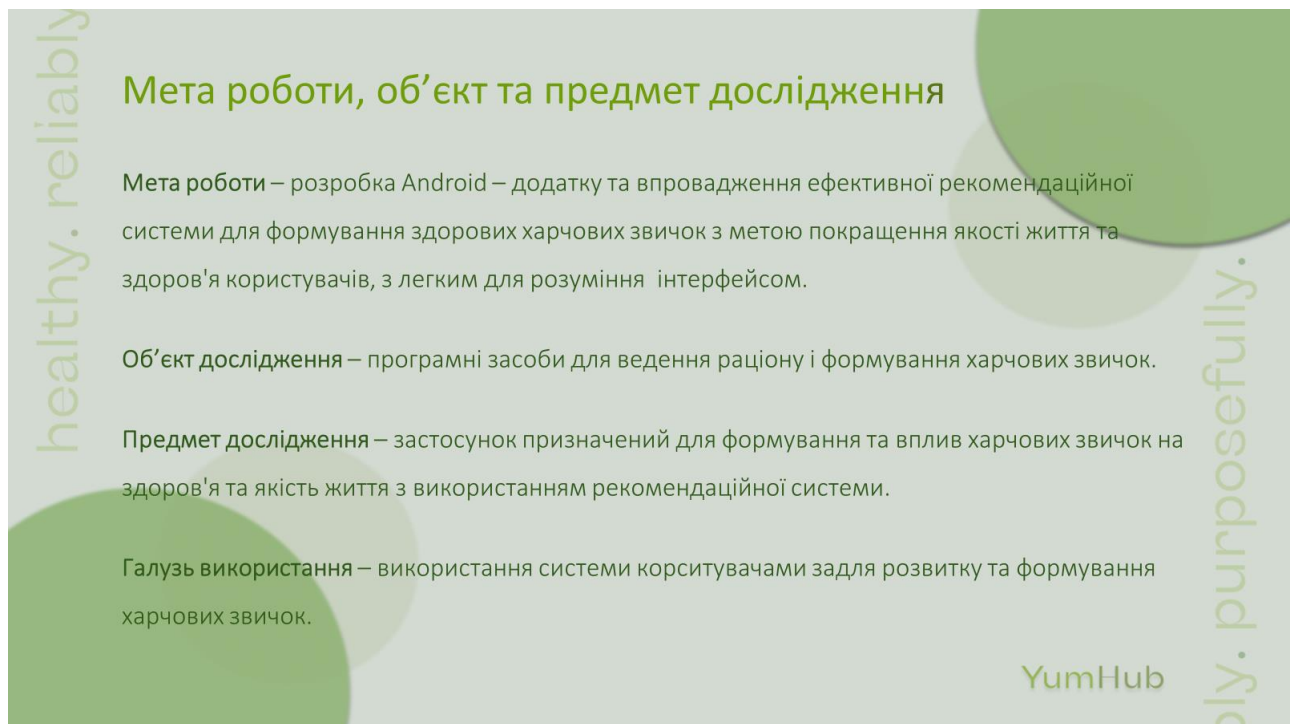


Задачі роботи

При написанні дипломної кваліфікаційної роботи було встановлено такі задачі для успішної розробки застосунку:

- Дослідження області предмету та аналогів планованого продукту
- Визначення платформи, для якої буде створюватися продукт
- Вибір мови програмування для розробки
- Обрання інструментів і середовища розробки програмного додатка
- Написання коду з дотриманням рекомендацій
- Проведення тестування програмного додатка на різних пристроях та версіях обраної операційної системи

Рисунок В.3 – Слайд 3



Мета роботи, об'єкт та предмет дослідження

Мета роботи – розробка Android – додатку та впровадження ефективної рекомендаційної системи для формування здорових харчових звичок з метою покращення якості життя та здоров'я користувачів, з легким для розуміння інтерфейсом.

Об'єкт дослідження – програмні засоби для ведення раціону і формування харчових звичок.

Предмет дослідження – застосунок призначений для формування та вплив харчових звичок на здоров'я та якість життя з використанням рекомендаційної системи.

Галузь використання – використання системи користувачами задля розвитку та формування харчових звичок.

Рисунок В.4 – Слайд 4

Порівняльна таблиця

Критерії оцінки	Назва застосунку			
	My FitnessPal	Yazio	Nutrilio	YumHub
Інтуїтивна зрозумілість	++	++	+	++
Зручність	++	++	+	++
Сучасність дизайну	+	++	±	++
Споживання ОЗУ	Від 70 МБ	Від 60 до 100МБ	Від 90 МБ	Від 40 до 100 МБ
Споживання електроживлення	≈100 mAh	≈180 mAh	≈220 mAh	≈180 mAh
Стабільність роботи	++	+	+	++

YumHub

Рисунок В.5 – Слайд 5

Основні функції

надання користувачам персоналізованих та ефективних порад щодо харчування, а тобто:

- збір та аналіз даних про харчові вподобання користувачів на основі історії їхнього харчування,
- надання індивідуалізованих порад з вибору продуктів, раціону та збалансованого харчування, враховуючи особливості користувача, тощо.

YumHub

Рисунок В.6 – Слайд 6

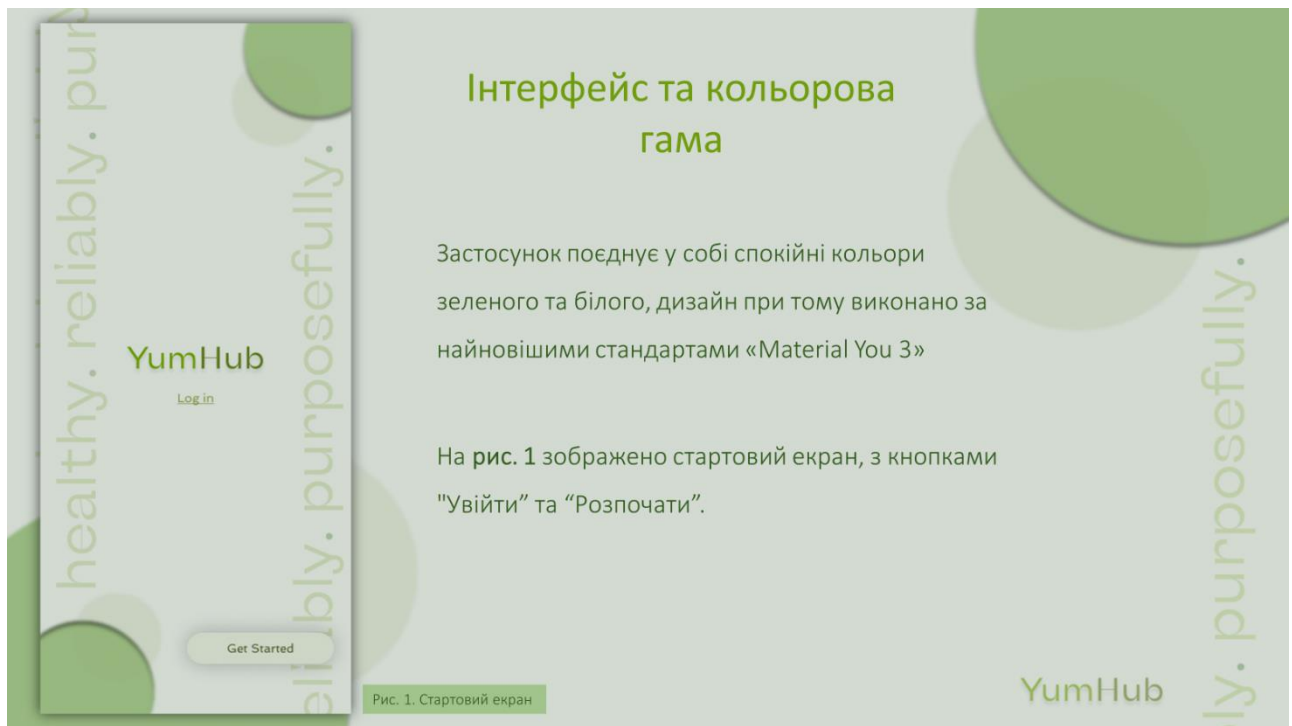


Рисунок В.7 – Слайд 7

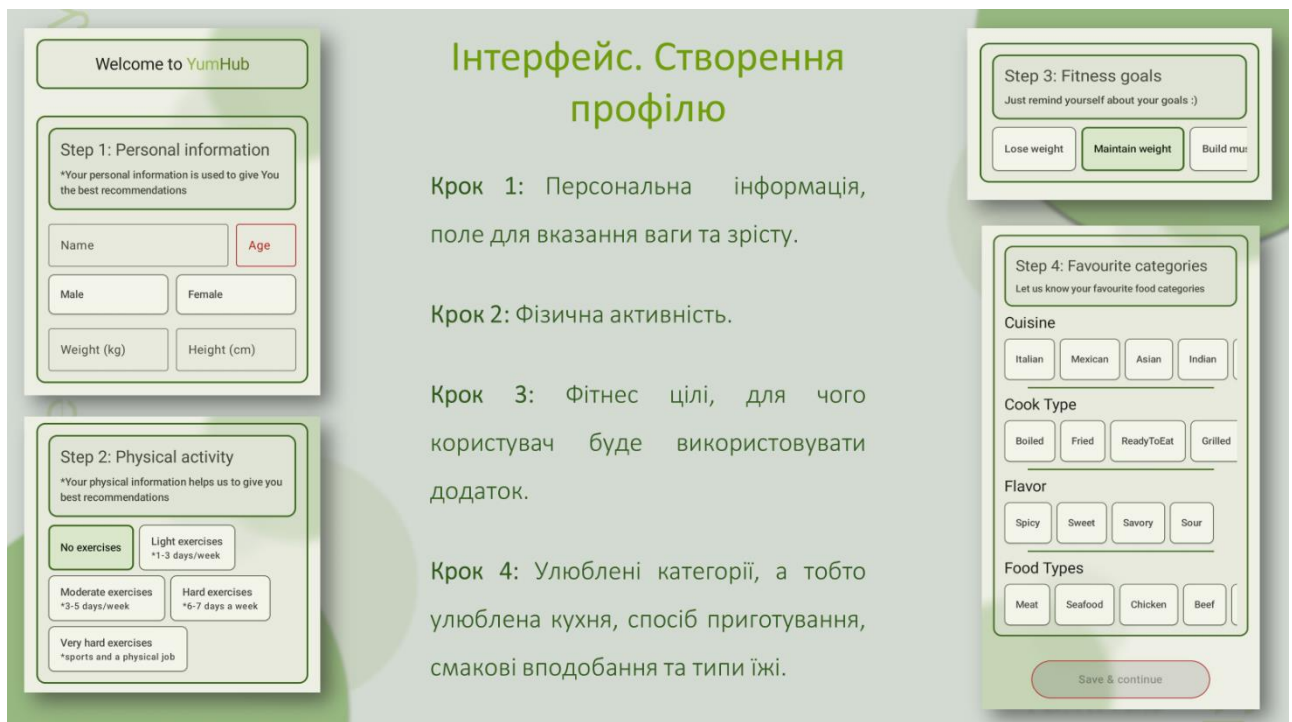


Рисунок В.8 – Слайд 8



Рисунок В.9 – Слайд 9

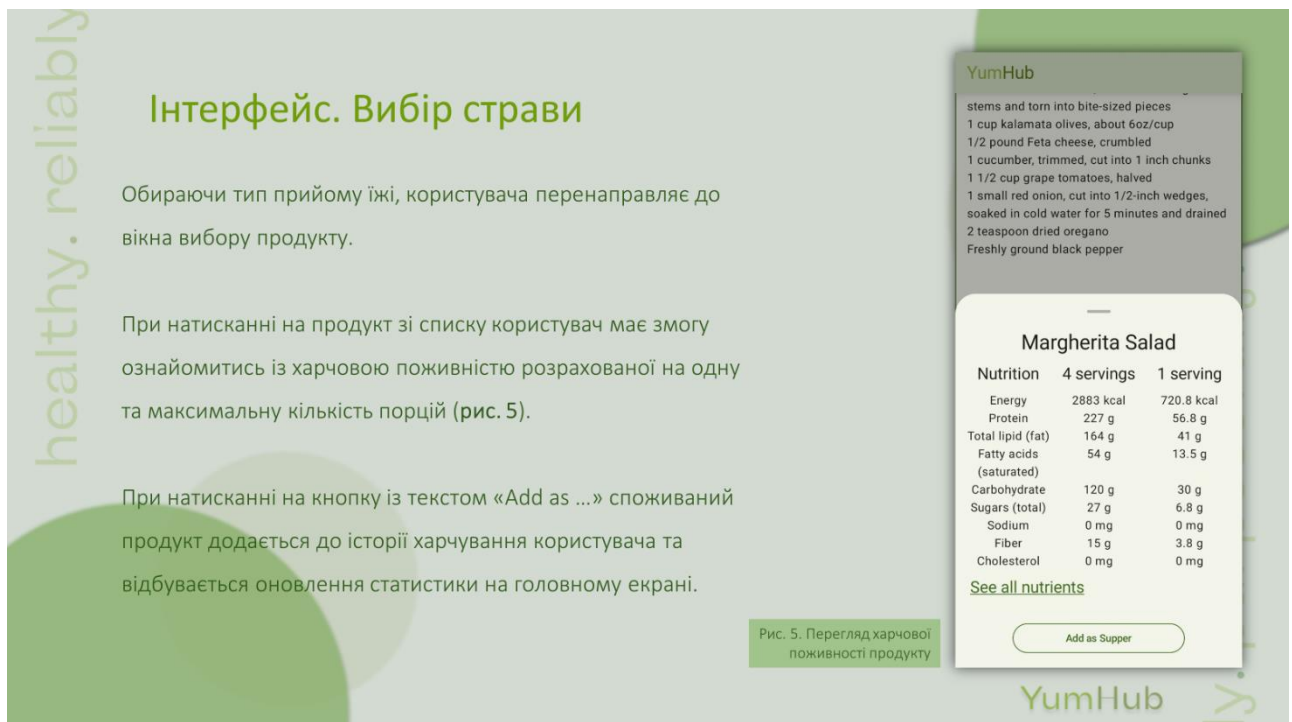


Рисунок В.10 – Слайд 10

Екран із відображенням рекомендацій та продуктів по категоріям

Також інтерфейс дає змогу користувачеві продивитись усі продукти відфільтровані по категоріях: найменший час готування, найменша кількість інгредієнтів, невелика собівартість, велика собівартість, мала кількість жирів, фастфуд та без фільтрів (рис. 6, 7).

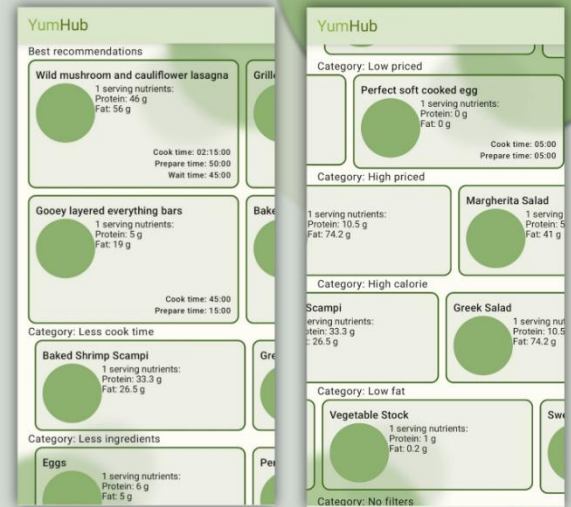


Рис. 6, 7. Екран із відображенням рекомендацій та продуктів по категоріям.

Рисунок В.11 – Слайд 11

Екран історії споживаних продуктів та води



Рис. 8. Компонент «Календар» для перегляду історії за обрану дату.

Інтерфейс застосунку дозволяє переглядати історію спожитих продуктів, їх нутрієнтів, а також спожитої води за конкретну дату. Для вибору дати було застосовано компонент «Календар» з бібліотеки Material3 за основу, та модифіковано для можливості вибору діапазону дат шляхом додавання компоненту «Чекбокс» із написом «Використовувати діапазон для вибору» (рис. 8).

Після підтвердження вибору – інтерфейс створює моделі графіків та передає їх до відповідних компонентів для відображення цих даних. Користувач має змогу змінювати масштаб графіків, передивляться дані для кожного обраного дня. Доступні такі дані для перегляду: спожита вода, білки, жири та вуглеводи (рис. 9).

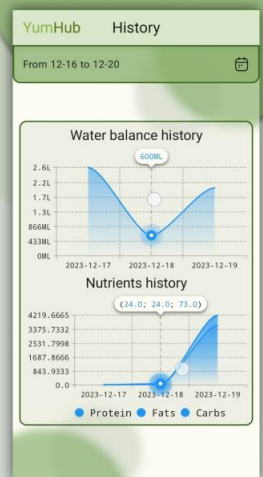


Рис. 9. Відображення історії харчування та споживаної води у діапазонному виборі дат.

Рисунок В.12 – Слайд 12

Огляд форми профіля (UserProfileScreen)

Рис. 10. Перегляд профілю

Компонент UserProfileScreen відповідає за перегляд даних, що користувач ввів під час реєстрації, він має свій особистий інтерфейсний контроллер, що є водночас і вьюмоделлю - UserProfileViewModel, що відповідає за усю логіку відображення та оновлення даних користувача.

Також було додано елемент графіку, що відображає дату та вагу користувача для легкого відслідковування змін (рис. 10).

Рисунок В.13 – Слайд 13

Екран детальної інформації обраної страви і рекомендації

Рис. 11. Екран з детальною інформацією страви

Компонент MealInfoScreen відповідає за відображення детальної інформації про страву та має декілька основних блоків даних: назву страви, харчову поживність та інформацію про приготування страви (інгредієнти та рецепт). Це дає змогу користувачеві ознайомитись із різноманітними рецептами у режимі офлайн, а також продивитись схожі страви, завдяки рекомендаційній системі та її алгоритмів порівняння (рис. 11).

Рисунок В.14 – Слайд 14



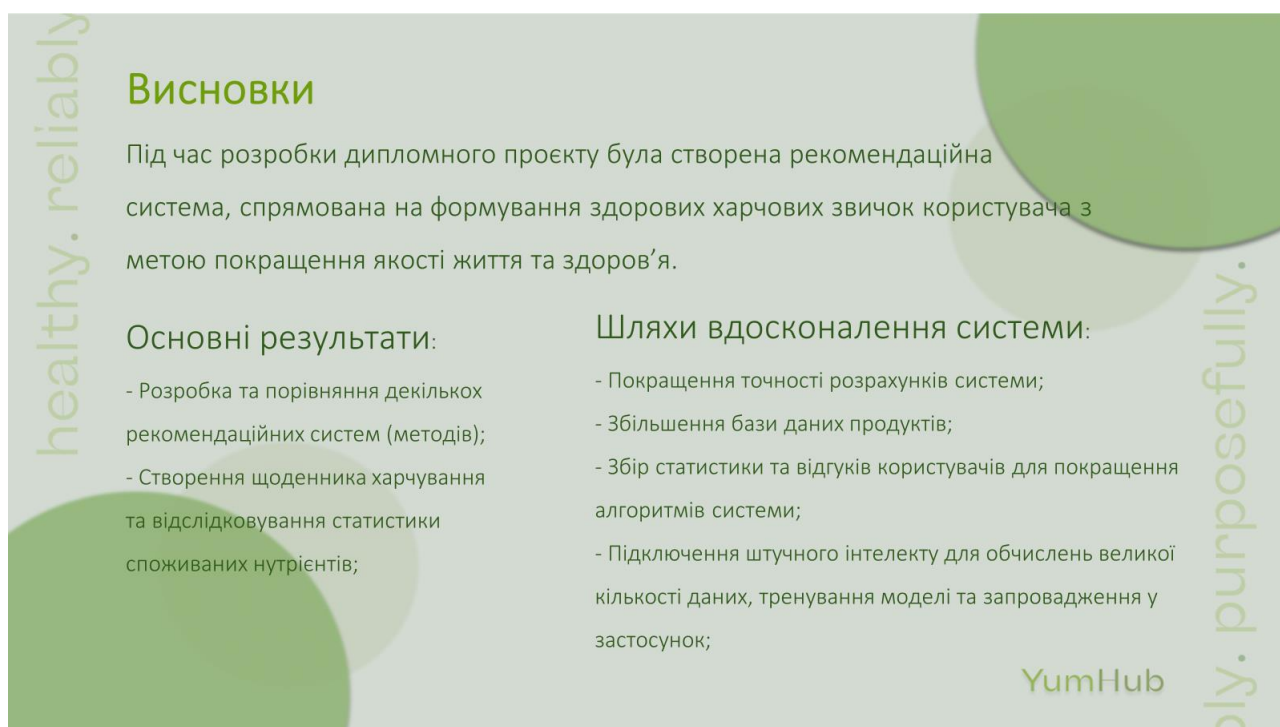
Рисунок В.15 – Слайд 15

Тестування застосунку на різних пристроях

Критерії тестування	Назва пристрою та версія Android		
	Xiaomi Poco X3 - 13	Honor 7A – 8.1	Xiaomi Mi10T – 12.0
Плавність анімації та роботи застосунку	Відмінно	Відмінно	Відмінно
Перевірка коректного завантаження	Коректно	Коректно	Відмінно
Правильність зміни і відображення даних	Відмінно	Коректно	Відмінно
Перевірка нагадувань	Коректно	Некоректно	Коректно
Коректність роботи календаря і перегляду історії	Відмінно	Непогано	Коректно
Адаптивність інтерфейсу	Так	Місцями - ні	Так

YumHub

Рисунок В.16 – Слайд 16



Висновки

Під час розробки дипломного проєкту була створена рекомендаційна система, спрямована на формування здорових харчових звичок користувача з метою покращення якості життя та здоров'я.

Основні результати:

- Розробка та порівняння декількох рекомендаційних систем (методів);
- Створення щоденника харчування та відслідковування статистики споживаних нутрієнтів;

Шляхи вдосконалення системи:

- Покращення точності розрахунків системи;
- Збільшення бази даних продуктів;
- Збір статистики та відгуків користувачів для покращення алгоритмів системи;
- Підключення штучного інтелекту для обчислень великої кількості даних, тренування моделі та запровадження у застосунок;

YumHub

Рисунок В.17 – Слайд 17



Кінець презентації проєкту рекомендаційної системи «YumHub»

Рисунок В.18 – Слайд 18