МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Мобільний додаток обліку спожитих калорій»

Здобувача групи ІТ-03 Чернова Олександра Олександровича
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Олександр ЧЕРНОВ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент кафедри ІТ, к.т.н., доцент Володимир НАГОРНИЙ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав.кафедри
____ Світлана ВАЩЕНКО

«__» _____ 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Чернову Олександру Олександровичу

1 Тема роботи *Мобільний додаток для обліку спожитих калорій*

керівник роботи *Нагорний Володимир В'ячеславович, к.т.н., доцент*,

затверджені наказом по університету від «07 »травня 2024 р.

2 Строк подання студентом роботи «26» травня 2024 р.

3 Вхідні дані до роботи технічне завдання на розробку мобільного додатку обліку спожитих калорій

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, моделювання та проектування мобільного додатку обліку спожитих калорій, розробка мобільного додатку обліку спожитих калорій

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) аналіз проблеми споживання калорій, визначення актуальності розробки мобільного додатку обліку спожитих калорій, порівняння продуктів-аналогів, функціональні вимоги до мобільного додатку, контекстна діаграма IDEF0, діаграма декомпозиції першого рівня, діаграма варіантів використання, логічна модель даних, архітектура мобільного додатку, сторінка щоденника, сторінка додавання страви, сторінка прогресу, редагування антропометричних показників тіла, висновки, презентація роботи мобільного додатку

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 04.04.2024

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної області	08.04.2024 – 12.04.2024	Виконано
2	Оформлення планування робіт	16.04.2024 - 18.04.2024	Виконано
3	Оформлення технічного завдання	18.04.2024 - 23.04.2024	Виконано
4	Аналіз та вибір засобів реалізації	23.04.2024 - 26.04.2024	Виконано
5	Проектування мобільного додатку	26.04.2024 - 04.05.2024	Виконано
6	Розробка мобільного додатку	04.05.2024 - 14.05.2024	Виконано
7	Тестування мобільного додатку	14.05.2024 - 16.05.2024	Виконано
8	Оформлення пояснювальної записки	16.05.2024 - 20.05.2024	Виконано

Студент _____ Олександр ЧЕРНОВ
(підпис)

Керівник роботи _____ к.т.н., доц. Володимир НАГОРНИЙ
(підпис)

АНОТАЦІЯ

Тема кваліфікаційної роботи бакалавра «Мобільний додаток обліку спожитих калорій».

Пояснювальна записка складається зі вступу, трьох розділів, висновку, списку використаних джерел із 25 найменувань, трьох додатків. Загальний обсяг робіт – 100 сторінок, у тому числі 26 сторінок основного тексту, 2 сторінки списку використаних джерел та 66 сторінок додатків.

Актуальність роботи полягає у необхідності спрощення можливості відстеження обліку калорій для людей, які прагнуть слідкувати за власним раціоном.

Метою проекту є створення мобільного додатку, який надаватиме користувачам можливість ведення обліку спожитих калорій. Результатом став мобільний додаток, який відповідає функціональним та системним вимогам.

У першому розділі проведено аналіз предметної області, де визначено актуальність розробки мобільного додатку. Розглянуто аналоги мобільних додатків обліку спожитих калорій. На основі розробки першого розділу було визначено мету та постановка завдання.

У другому розділі було проведено структурно-функціональне моделювання. В результаті роботи було створено контекстну діаграму в нотації IDEF0 та її декомпозиція, діаграму варіантів використання для показу процесів та взаємозв’язків між ними.

У третьому розділі було описано процес розробки мобільного додатку, показано функціонал та основні можливості розробки.

Ключові слова: мобільний додаток, облік калорій, Kotlin, Jetpack Compose, Firebase

ЗМІСТ

ВСТУП

В умовах, коли теми здоров’я та особистої фізичної форми є болючими, питання відстеження власного раціону харчування стає все більш широко популярною [1]. Одним з методів відстеження раціону харчування є облік калорій, спожитих протягом дня.

В таких умовах виникає проблема пошуку інструментів, за допомогою яких можна зручно налагодити процес ведення щоденного раціону. Спортсмени шукають способи відстеження власного харчування для досягнення спортивних результатів [2], послідовники здорового способу життя хочуть мати можливість слідкувати за своєю вагою. Таким чином, виникає потреба у зручному та якісному інструменті, проте на українському полі розробки мобільних додатків майже відсутні вітчизняні аналоги. Українські користувачі використовують різні інструменти для різних цілей, що може бути незручним у практиці.

Гідним рішенням може виступити мобільний додаток для обліку спожитих калорій. Метою проекту є створення мобільного додатку, який надаватиме користувачам можливість ведення обліку спожитих калорій.

В ході виконання кваліфікаційної роботи бакалавра необхідно виконати наступні задачі:

- провести огляд останніх досліджень і публікацій;
- проаналізувати програмні продукти-аналоги;
- сформулювати функціональні та нефункціональні вимоги;
- вибрати методи дослідження та інструменти розробки;
- виконати моделювання та проектування мобільного додатку;
- розробити мобільний додаток;
- виконати тестування мобільного додатку;

Практична цінність проекту полягає у покращенні відстеження обліку калорій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

За даними Всесвітньої організації охорони здоров'я [3], близько половини дорослого населення світу має зайву вагу, а третина з них страждає на ожиріння. Ця тенденція стає все більш загостреною в країнах з високим рівнем розвитку та в країнах, що розвиваються, де зростає споживання висококалорійних продуктів, а також зменшується фізична активність через сидячий спосіб життя та технологічний прогрес. Таким чином, щоденно зростає ризик виникнення у людей серцево-судинних захворювань, діабету, раку та інших [4].

Мобільні додатки можуть успішно допомагати користувачам контролювати власний стан здоров’я і є доступною альтернативою персональним програмам зі схуднення [6]. Такі додатки можуть бути гідною можливістю схуднення для людей, які мають обмежені фінансові можливості або хочуть зекономити кошти.

У 1919 році американський вчений Френсіс Бенедикт і його співавтор Джеймс Харріс опублікували наукову працю [5] про базальний метаболізм людини (кількість енергії, яка потрібна організму людини для нормального функціонування). У цій праці ними було наведено формулу розрахунку кількості калорій, яка враховувала вагу, вік та стать людини. Після них вчений Марк Міффлін вдосконалив її, та на основі їх роботи зміг вивести наступну формулу (1.1):

$$BMR = 10 * weight(kg) + 6.25 * height(cm) - 161 \qquad (1.1)$$

де BMR – показник базального метаболізму, weight – вага людини у кілограмах, та height – висота людини у сантиметрах. На основі рівняння рахується кількість необхідних калорій протягом дня.

Застосунки для схуднення ефективні для зниження ваги протягом 3-12 місяців [6]. Таким чином, подібні мобільні додатки можуть допомогти користувачам знизити вагу протягом короткого проміжку часу.

Також можемо виділити наступні переваги у використанні мобільних додатків[7] :

- зручність та доступність відстеження прийому їжі;
- індивідуальне планування харчування;
- графічне відображення прогресу;
- можливість аналізу та порівняння показників спожитих калорій з власними цілями.

1.2 Аналіз програмних продуктів – аналогів

Мобільний додаток [8] – це програмне забезпечення, призначене для використання на мобільних пристроях, таких як смартфони та планшети. Цей термін виник в результаті розвитку технологій та та поширення мобільних пристроїв серед користувачів. Мобільні додатки відіграють ключову роль у сучасному цифровому житті, дозволяючи отримувати доступ до різноманітних сервісів на інформації.

Існує кілька типів мобільних додатків [9], включаючи:

− Нативні додатки (розроблені для конкретної платформи, наприклад iOS, Android, тощо)

− Вебдодатки (запускаються у веббраузері та взаємодіють з вебсервером.)

− Гібридні додатки (комбінують елементи нативних та веб-додатків)

Перед тим як приступати до розробки додатку, потрібно проаналізувати аналоги. Виділимо кілька схожих додатків. Кожен з них має свої переваги та недоліки.

1.2.1. «Lose it»

Розглянемо існуючі безкоштовні мобільні додатки. Першим аналогом є «Lose it» [10]. Додаток містить статистику спожитих калорій та детальне пояснення до кожного параметру (рис.1.1).

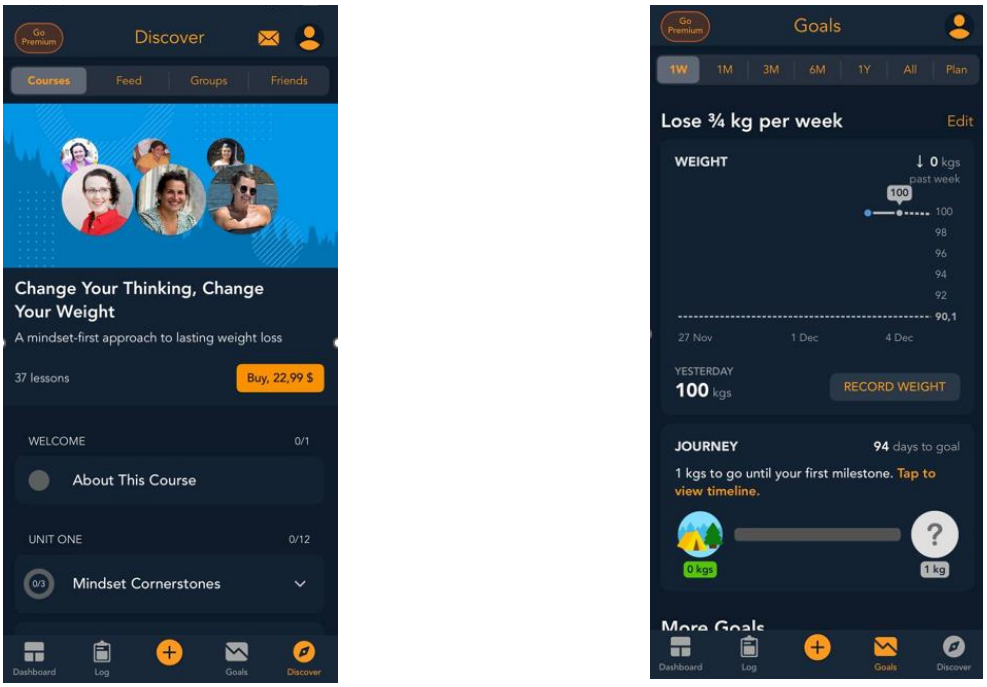

Рисунок 1.1 – Екрани додатку «Lose it»

До функціональних можливостей входять можливість додавання страв до раціону, пошуку продуктів, збереження історії користувача, підрахунок калорій та макроелементів, спостереження за вагою та фізичною активністю.

Основним недоліком можна виділити недостатню адаптацію під індивідуальні потреби (загальне опитування, яке не є оптимальними для всіх користувачів.)

1.2.2. «LifeSum»

Наступний аналог – «LifeSum» [11]. Додаток, функціонал якого включає прорахунок ліміту калорій на день, внесення страв у щоденний раціон, рецепти до страв. Головне меню додатка зображено на рисунку 1.2.

Lifesum дає змогу відслідкувати макроелементи (білки, жири, вуглеводи), які містяться в раціоні, що у свою чергу допомагає забезпечити правильне харчування. Додаток надає можливість відстежувати фізичну активність. Користувач може додавати вправи, тренування, прогулянки та інші види активності, якими займаєтеся.

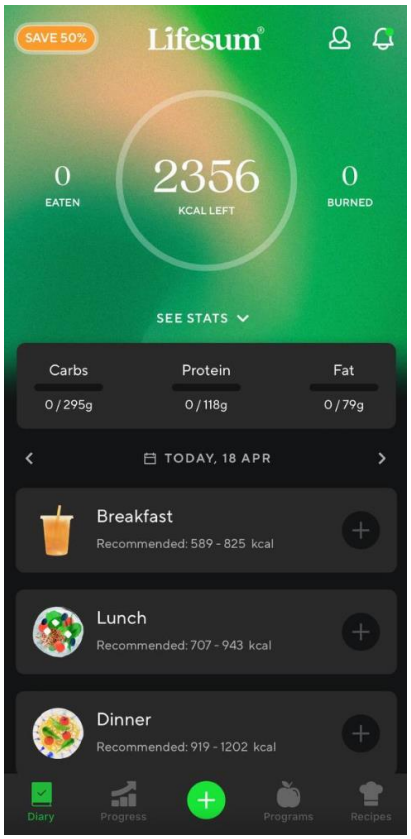

Рисунок 1.2 – Додаток «Lifesum»

У ході аналізу був виявлений недолік – недостатній обсяг дієт та режимів схуднення. Також іноді можуть зустрічатися неповні або неточні дані про калорійні значення продуктів.

1.2.3. «YAZIO»

Додаток «YAZIO» [12] − додаток, який спеціалізується на веденні персональних стратегіях схуднення. При запуску додатку пропонується пройти детальний тест, який дає змогу підібрати найбільш точний ліміт споживання калорій.

Зображення головного екрану мобільного додатку зображено на рисунку 1.3

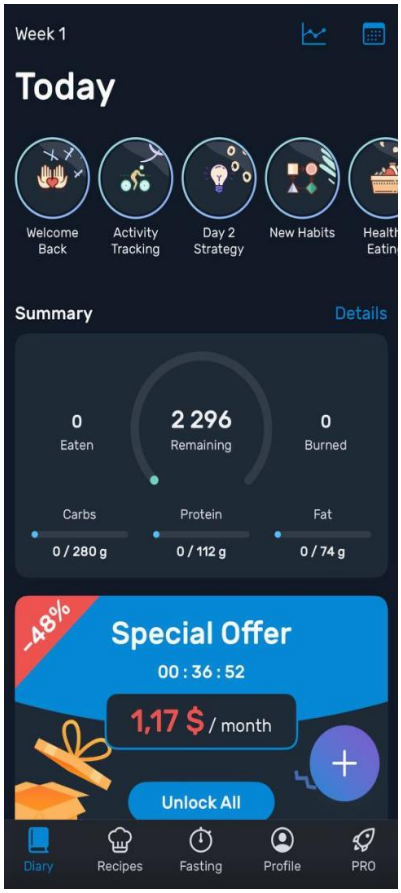

Рисунок 1.3 – Додаток «YAZIO»

YAZIO дозволяє додавати страви, має трекер фізичної активності, а також дозволяє встановити цілі та для отримання персоналізованих рекомендацій. Додаток також має щоденник харчування, який допомагає користувачам зберігати збалансоване харчування та досягати цілей.

Як недоліки можна виділити лиш те, що більш детальний опис по споживанню страв по білкам, жирам, та вуглеводам недоступний для користувачів та ефективність калькулятору підрахунку калорій сильно залежить від точності введеної користувачем інформації.

Складемо порівняльну таблицю розглянутих додатків. Результати аналізу представлено у таблиці 1.1.

Таблиця 1.1 - Порівняльна характеристика аналогів

Назва критерію	Lose it	Lifesum	YAZIO	Власна розробка
Підрахунок калорій	+	+	+	+
Достатня база даних страв	+	-	+	+
Опитування про фізичний стан користувача	-	+	+	+
Детальний опис страв по білкам, жирам та вуглеводам	+	+	-	+
Зручна навігація по мобільному додатку	+	+	+	+

На основі порівняльної характеристики аналогів було визначено наступні основні функціональні вимоги:

- підрахунок калорій;
- відстеження прогресу у схудненні;
- внесення продуктів харчування у раціон;
- редагування власних вагових показників;

Опис функціональних вимог зазначено в технічному завданні в додатку А.

1.3 Мета та задачі проекту

Метою проекту є розробка мобільного додатка для обліку споживання калорій, який сприятиме користувачам у веденні більш свідомого та збалансованого харчування. У ході роботи необхідно:

− провести аналіз предметної області для визначення актуальності розробки.

− проаналізувати наявні мобільні додатки для виділення особливостей існуючих аналогів. Результатом стануть функціональні та системні вимоги, на основі яких створюватиметься структура мобільного додатку;

− виконати проектування структури мобільного додатку. По закінченню моделювання буде виконано структурно-функціональне моделювання, результатом якого стане створена діаграма IDEF0 та діаграма декомпозиції функціональної моделі. У ході проектування буде створена діаграма варіантів використання, яка показуватиме взаємозв'язки між системою та акторами у ній. Результатом діяльності стане виконане проектування логічної моделі бази даних. Усі етапи проектування будуть необхідними для створення мобільного додатку з конкретними вимогами, що забезпечить точність у процесі розробки функціональних вимог мобільного додатку;

− виконати реалізацію мобільного додатку. Результатом розробки стане робочий мобільний додаток з відповідним функціоналом;

− виконати тестування додатку. Результатом роботи стане тест-репорт з наявності чи відсутності помилок;

На основі проведеної роботи виконано планування, яке знаходиться в додатку Б.

1.4 Методи дослідження та інструменти реалізації

Інструменти реалізації включають у себе мови програмування, фреймворки, системи керування базами даних тощо необхідних для реалізації системи. У ході аналізу було вибрано інструменти:

− мову програмування Kotlin [13], яка є найбільш популярною для розробки програмних продуктів для платформи Android. Kotlin пропонує зручний

синтаксис та має велику кількість бібліотек та інструментів, які сприяють розробці якісного програмного продукту.

− Jetpack Compose [14], набір інструментів від компанії Google для розробки інтерфейсів користувача для створення застосунків на платформі Android що базується на мові програмування Kotlin. У порівнянні з традиційним методом XML, Jetpack Compose пропонує більш простий та зрозумілий підхід до розробки інтерфейсу.

− Android Studio [15], інтегроване середовище розробки на платформі Android, яке включає можливість як створення програмного коду, так і можливість емуляції мобільних додатків на віртуальних мобільних пристроях.

− платформа Firebase від компанії Google [16], яка надає можливість створення системи керування базою даних для мобільних додатків.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

2.1 Структурно-функціональне моделювання

В ході розробки програмного продукту виділяють структурно-функціональне моделювання, яке допомагає отримати краще розуміння про систему, що у свою чергу дозволить оптимізувати роботу програмного додатку та зекономити ресурси на розробку.

Для представлення структурно-функціонального моделювання зазвичай використовують діаграму у нотації IDEF0 [17]. Діаграма складається з блоків, які відображають процеси та функції, які плануються для створення системи та стрілок, які показують вхідні та вихідні дані системи.

На рисунку 2.1 зображена контекстна діаграма IDEF0, центральним процесом якого є функціонування мобільного додатку обліку спожитий калорій:

Рисунок 2.1 – Контекстна діаграма IDEF0

З рисунку 2.2 можемо виділити наступні елементи системи:

- Вхідними даними для системи є: логін та пароль, запит на внесення страв у раціон, запит на додавання страв у базу даних, зміна антропометричних показників, запит на перегляд прогресу у схудненні.
- Для управління: інструкції по використанню мобільного додатка
- Механізми включають: мобільний додаток, база даних, користувач
- Вихід представлені як: авторизований користувач, нові/оновлені дані у раціоні, внесені страви у власну базу даних, редаговані антропометричні показники користувача, таблиця прогресу користувача.

Для розбиття функцій системи на більш дрібні й прості елементи використаємо діаграму декомпозиції функціонального проектування. Для нашої системи діаграма декомпозиції системи зображена на рисунку 2.2:

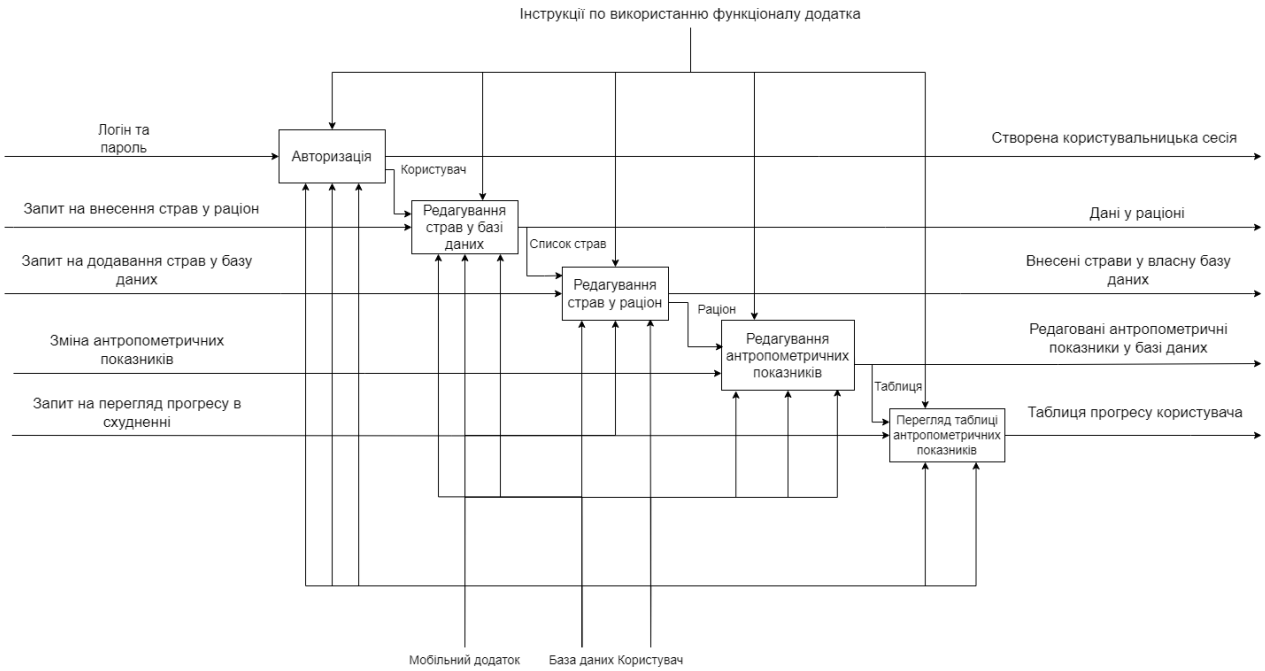

Рисунок 2.2 – Діаграма декомпозиції функціональної моделі першого рівня

2.2 Проектування мобільного додатку

Діаграма варіантів використання [18] (рис. 2.3) є одним з найважливіших елементів системи, так як дозволяє побачити взаємозв'язки між акторами та системою, що дозволяє більш точно виявити функціональні вимоги з боку користувачів.

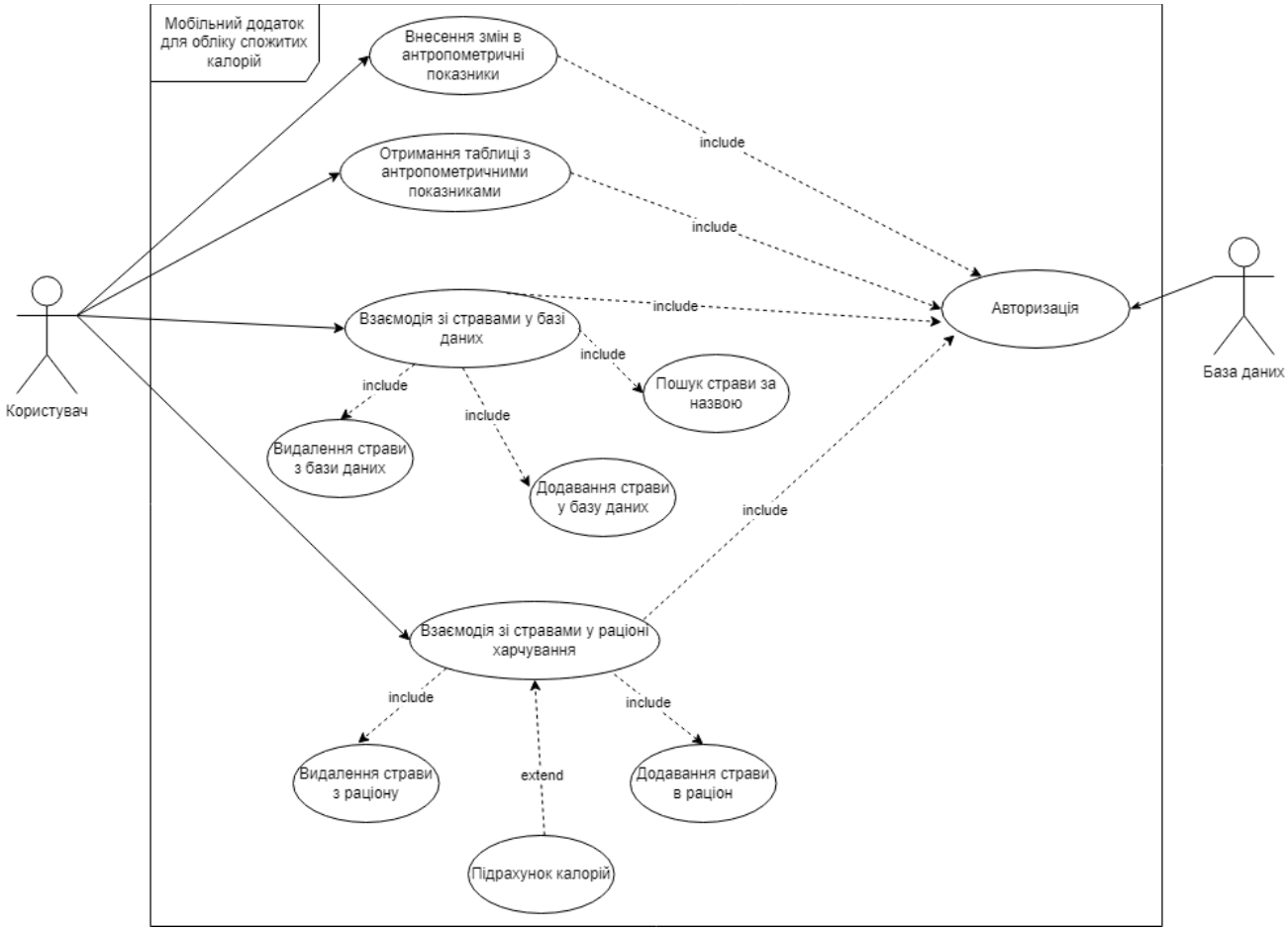

Рисунок 2.3 – Діаграма варіантів використання

З рисунку 2.3 можемо визначити основних акторів у системі – користувача та базу даних. Варіанти використання включають у себе авторизацію, реєстрацію, операціх зі стравами у базі даних (пошук,додавання та видалення), операція зі стравами у раціоні харчування (додавання та видалення, підрахунок калорій), внесення змін в антропометричні показники та можливість переглядання таблиці з антропометричними показниками

2.3 Проектування моделі бази даних

Логічна модель даних [19] необхідна для відображення структури мобільного додатку. У якості сутностей виступають:

– User: узагальнена сутність користувачів системи. Сутність включає себе вагу користувача, його зріст та стать.

– Dish: сутність яка відображає страви у системі. Поля сутності включає в себе назву страви, кількість калорій, білків, жирів та вуглеводів.

– ProgressTable: сутність, яка містить параметри користувача про його прогрес. Сутність відображає дату за певний день та ширину поясу користувача.

Сутність User пов’язана з сутністю ProgressTable зв’язком один-до-багатьох: кожен користувач може мати багато записів про свій прогрес. Між сутностями User та Dish існує зв'язок один-до-багатьох: кожен користувач може мати багато страв.

На рисунку 2.4 зображено логічна модель бази даних мобільного додатку:

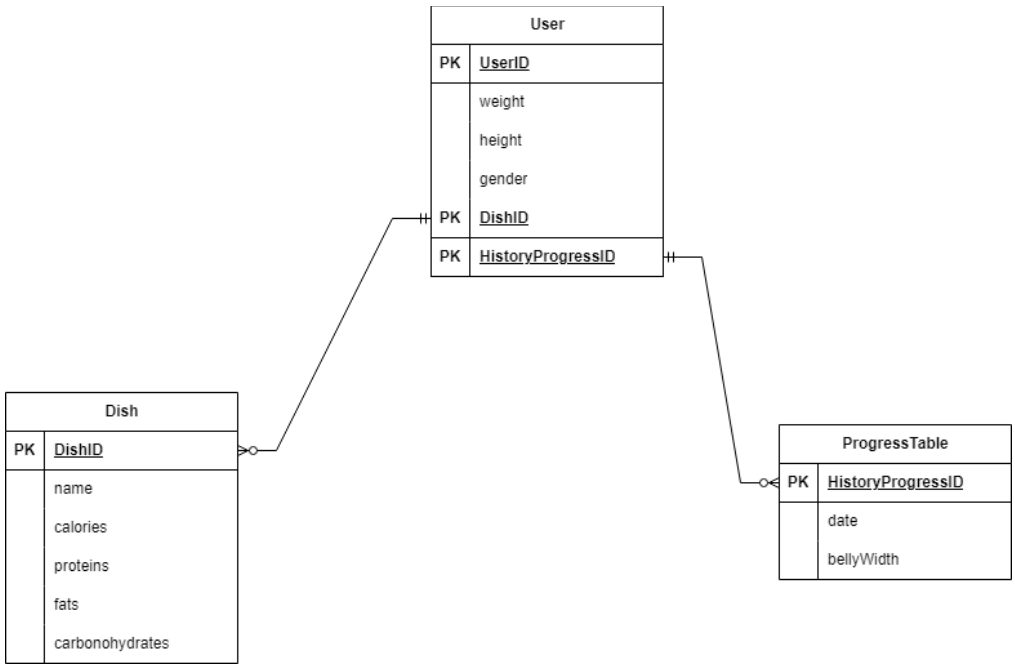

Рисунок 2.4 – Логічна модель даних

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Архітектура програмного продукту

Для представлення структури мобільного додатку використовується шаблон проектування MVVM (Model,View,Viewmodel) [20]. Даний шаблон дозволяє відділити рівень представлення від бізнес-логіки, що значно спрощує розробку мобільного додатку.

Модель (Model) представляє рівень даних застосунку. Модель відповідає за отримання даних, їх зберігання і виконання операцій над ними. У нашому випадку моделями виступають класи Dish та User.

Представлення (View) включає користувацький інтерфейс (UI) і керує тим, як дані відображаються користувачу. У нашому випадку це методи DailyScreen, DishAddingScreen та ProgressScreen.

Модель представлення (ViewModel) є посередником між Model і View. Вона містить логіку представлення, яка дозволяє View взаємодіяти з Model. У якості посередника виступає клас FirebaseViewModel разом з полями та методами які входять в нього.

На рисунку 3.1 відображено загальну архітектуру мобільного додатку:

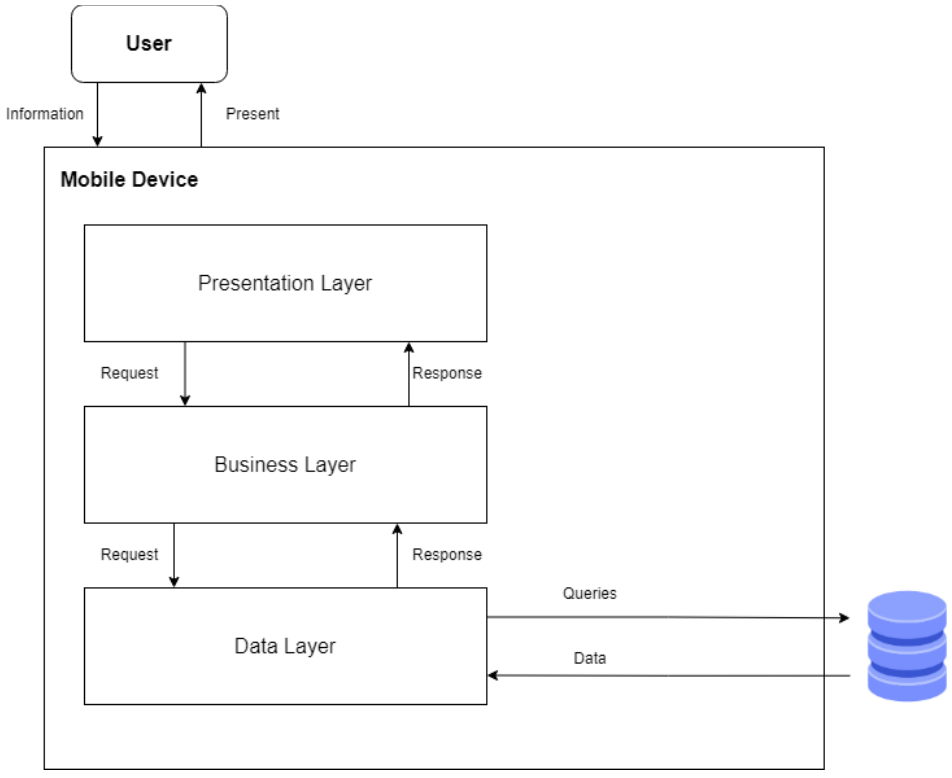

Рисунок 3.1 – Архітектура мобільного додатку

3.2 Програмна реалізація

Розробка мобільного додатку була проведена у середовищі Android з використанням мови програмування Kotlin та інструментів Jetpack Compose від компанії Google.

Програма складається з трьох сторінок (View), які поєднані між собою моделлю представлення (FirebaseViewModel) та двома моделями даних (класи Dish та User). Навігація між екранами, яке розташоване у нижній частині екрану здійснюється за допомогою контролерів.

У якості користувача було створено клас User:

```
data class User(
    val weight: Int = 0,
    val height: Int = 0,
    val gender: String = "",
    val date: String = "",
    val bellyWidth: Int = 0
)
```

Для страв було створено клас Dish, який містить наступні поля та методи класу:

```
data class Dish(
    val id: String = "",
    val name: String = "",
    val calories: Int = 0,
    val proteins: Int = 0,
    val fats: Int = 0,
    val carbohydrates: Int = 0
)
```

Для полегшення роботи з контролерами було створено клас Screen, який містить посилання на сторінки:

```
sealed class Screen(val route:String){
    object LoginScreen : Screen("loginscreen")
    object SignupScreen : Screen("signupscreen")
    object Homescreen : Screen("homescreen")
    object AddingDishcreen : Screen("addingdishscreen")
    object Progressscreen : Screen("progressscreen")
}
```

Для підрахунку базального метаболізму людини використовується метод calculateBasalMetabolicRate:

```
 fun calculateBasalMetabolicRate(weight:  Int, height:  Int): Int {
        val bmr = 10 * weight + 6.25 * height - 161
        return bmr.toInt()
    }
```

Метод calculateRequiredCalories повертає значення калорій для конкретного користувача:

```
fun calculateRequiredCalories(user: User): Int {
        val weight = user.weight
        val height = user.height
        val basalMetabolicRate = calculateBasalMetabolicRate(weight, height)
        return basalMetabolicRate
    }
```

Для розрахунку необхідних білків, жирів та вуглеводів використовується метод getRequiredMacronutrients. За це відповідає наступна частина коду:

```
user?.let {
                val requiredCalories = calculateRequiredCalories(user)
                val proteinPercentage = 0.15
                val fatPercentage = 0.30
                val carbohydratePercentage = 0.55
                val proteinGrams = (requiredCalories * proteinPercentage / 4).toInt()
                val fatGrams = (requiredCalories * fatPercentage / 9).toInt()
                val carbohydrateGrams = (requiredCalories * carbohydratePercentage /
4).toInt()
                callback(requiredCalories, proteinGrams, fatGrams, carbohydrateGrams}
```

Для відображення страв у раціоні використовується метод setSelectedDishesForCurrentUser. Метод видаляє старі страви цього типу, а потім додає нові страви зі списку. Якщо все проходить успішно, викликається функція onSuccess, а якщо ні - onFailure з інформацією про помилку.:

```
        fun setSelectedDishesForCurrentUser(
              mealType: String,
              selectedDishes: List<Dish>,
              onSuccess: () -> Unit,
              onFailure: (Exception) -> Unit
          ) {
              val userId = _currentUser?.uid ?: return
              val                                userRef                                =
fs.collection("users").document(userId).collection("selectedDishes")

              // Видаляємо існуючі страви поточного типу перед додаванням нових
              userRef.document(mealType).delete().addOnSuccessListener {
                  // Додаємо нові страви
                  userRef.document(mealType).set(mapOf("dishes" to selectedDishes.map {
dishToMap(it) }))
                      .addOnCompleteListener { task ->
                          if (task.isSuccessful) {
                              onSuccess()
                          } else {
                              onFailure(task.exception ?: Exception("Unknown error"))
                          }
                      }
              }.addOnFailureListener { exception ->
                  onFailure(exception)
              }
          }
```

Для видалення страви з раціону використовується метод deleteDishFromSelectedDishesList. Пошук страви відбувається за назвою у базі даних Firestore для авторизованого користувача. Якщо користувача знайдено, то результати запиту перетворюються на об'єкти класу Dish:

```
    val userId = _currentUser?.uid ?: return
        val userDishesRef = fs.collection("users").document(userId).collection("selectedDishes").document(mealType)

        fs.runTransaction { transaction ->
            val snapshot = transaction.get(userDishesRef)
            val selectedDishes = snapshot.get("dishes") as? List<Map<String, Any>> ?: emptyList()
            val updatedSelectedDishes = selectedDishes.filterNot { it["id"] == documentId }
            transaction.update(userDishesRef, "dishes", updatedSelectedDishes)
        }.addOnSuccessListener {
            onSuccess()
            startListeningForSelectedDishes(
                onSuccess = { breakfast, lunch, dinner ->
                },
                onFailure = { exception ->
                }
            )
        }.addOnFailureListener { exception -> onFailure(exception)}
```

Для пошуку страв у базі даних користувача використовується метод searchDishes. Запит відбувається за назвою, яку вказує користувач:

```
    fun searchDishes(query: String, callback: (List<Dish>) -> Unit) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid
        val fs = Firebase.firestore
        userId?.let { uid ->
            fs.collection("users").document(uid).collection("dishes")
                .whereEqualTo("name", query)
                .get()
                .addOnSuccessListener { querySnapshot ->
                    val dishes = mutableListOf<Dish>()
                    for (document in querySnapshot) {
                        val dish = document.toObject<Dish>()
                        dishes.add(dish)
                    }
                    callback(dishes)
                }
                .addOnFailureListener { exception ->
                }}}
```

Для видалення страви з бази даних за ім’ям використовується метод deleteDishByName. Метод отримує ідентифікатор поточного користувача, потім шукає страву за заданою назвою. Якщо страва знайдена, вона видаляється, викликаючи функцію onSuccess, в іншому випадку викликається функція onFailure:

```
    fun deleteDishByName(dishName: String, onSuccess: () -> Unit, onFailure: (Exception) -> Unit) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid
        val fs = Firebase.firestore

        userId?.let { uid ->
            // Пошук страв за ім'ям в колекції "dishes"
            fs.collection("users").document(uid).collection("dishes")
                .whereEqualTo("name", dishName)
                .get()
                .addOnSuccessListener { querySnapshot ->
                    // Перевірка на наявність знайденої страви
                    if (!querySnapshot.isEmpty) {
                        // Отримання страви перед його видаленням
                        val document = querySnapshot.documents[0]
                        document.reference.delete()
                            .addOnSuccessListener {
                                onSuccess()
                            }
                            .addOnFailureListener { exception ->
                                onFailure(exception)
                            }
                    } else {
                        onFailure(Exception("Error"))
                    }
                }
                .addOnFailureListener { exception ->
                    onFailure(exception)
                }
        }
    }
```

Для взаємодії між екраном користувача та базою даних було створено клас FirebaseViewModel, який дозволить використовувати відповідні поля та методи. Для взаємодії з Firebase створюється три поля:

```
private val auth: FirebaseAuth = Firebase.auth
private val fs = Firebase.firestore
private val _currentUser = FirebaseAuth.getInstance().currentUser
```

Поле auth дозволяє взаємодіяти з сервісом автентифікації сервісу Firebase, поле fs дозволяє повернути взаємодіяти з хмарною базою даних Firestore. Поле _currentUser дозволяє повернути поточного авторизованого користувача, у випадку якщо користувач не авторизований, повертає значення null.

Для реєстрації користувача було створено метод signInToAccount. Для обробки виключних ситуацій використовується блок try-catch, який перехоплює можливі помилки при реєстрації користувача та передає йому на екран:

```
try {
        auth.createUserWithEmailAndPassword(email, password)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            onSuccess()
                        } else {
                            onError("Помилка при реєстрації: ${task.exception?.message}")
                        }
                    }
                } catch (ex: Exception) {
                    onError("Помилка при реєстрації: ${ex.message}")
                }
```

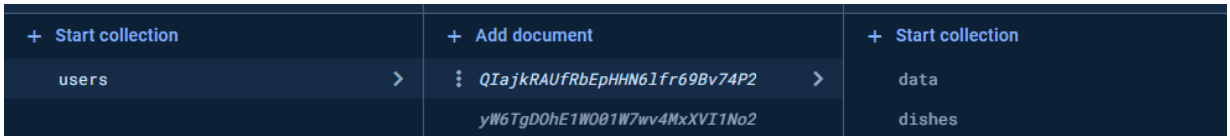

Рисунок 3.2 – Приклад успішного створення користувача

Аналогічним чином реалізовано методи для авторизації, операціями користувача з базою даних страв, стравами у раціоні та зміною антропометричних показників. Реалізовані методи представлені у додатку В.

3.3 Використання програмного додатку

Робота з додатком починається з реєстрації або авторизації в аккаунт користувача. У випадку, коли користувач не зареєстрований, йому необхідно це зробити. (рис. 3.3):

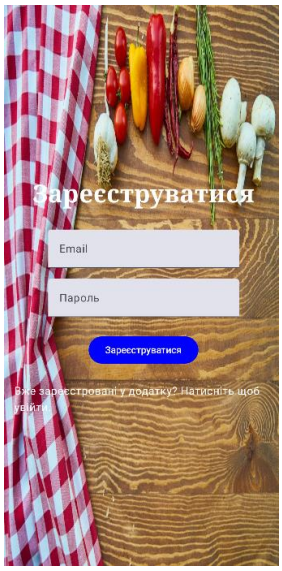

Рисунок 3.3 – Сторінка реєстрації

У іншому випадку, користувачу пропонується увійти в аккаунт з власним логіном та паролем (рис. 3.4):

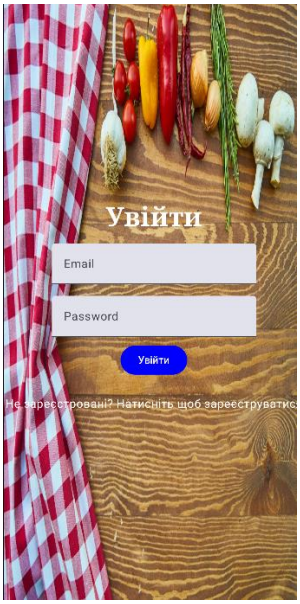

Рисунок 3.4 – Сторінка входу в аккаунт

У випадку, якщо користувач не зміг успішно авторизуватися або зареєструватися на екрані з’являється помилка (рис.3.5):

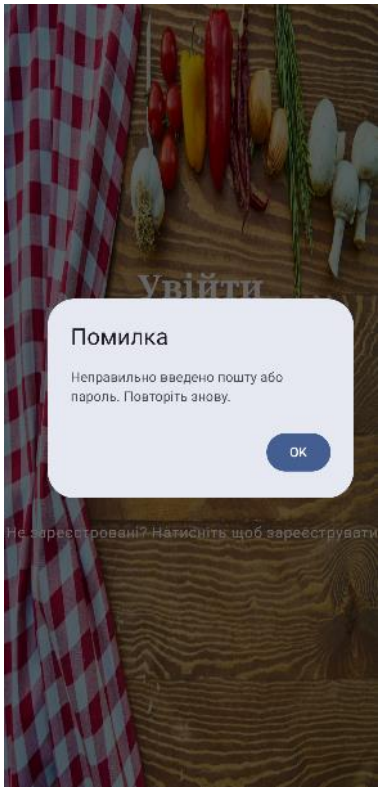

Рисунок 3.6 – Помилка при реєстрації та авторизації

Після входу в аккаунт, користувач потрапляє на головне меню додатку (рис.3.5):

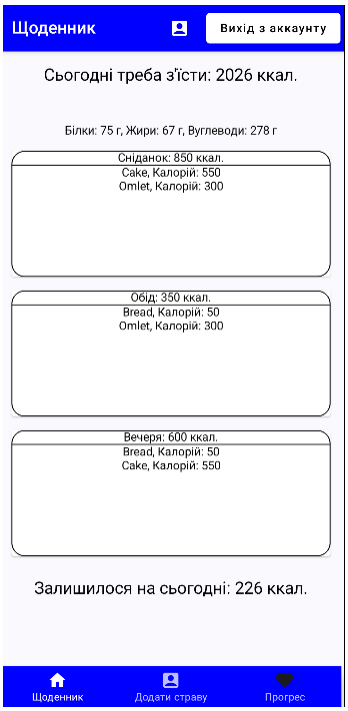

Рисунок 3.7 – Сторінка «Щоденник»

Для додавання страви у раціон харчування користувачу необхідно натиснути на прямокутник, де він має можливість додати страву, або видалити її з раціону (рис.3.6):

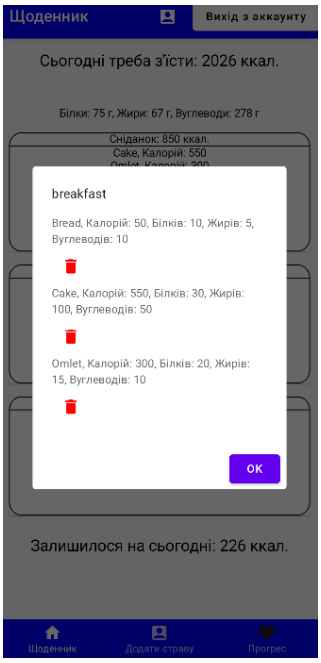

Рисунок 3.8 – Контекстне меню для перегляду страв

На сторінці «Додати страву» користувачу пропонується додати страву до власної бази даних, знайти її та видалити за необхідністю (рис. 3.7):

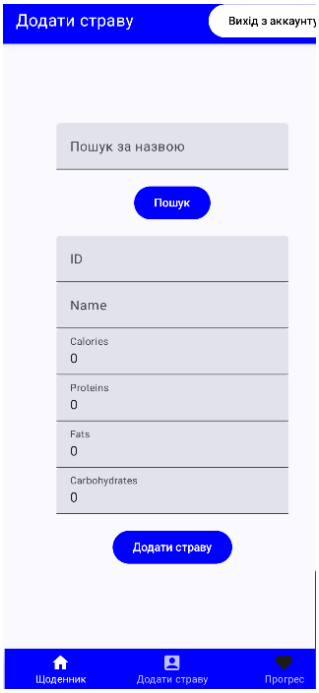

Рисунок 3.8 – Сторінка «Додати страву»

Для зміни ваги, зросту та ширини поясу користувачу необхідно натиснути іконку користувача, де він матиме змогу змінити показники власного тіла (рис.3.9):

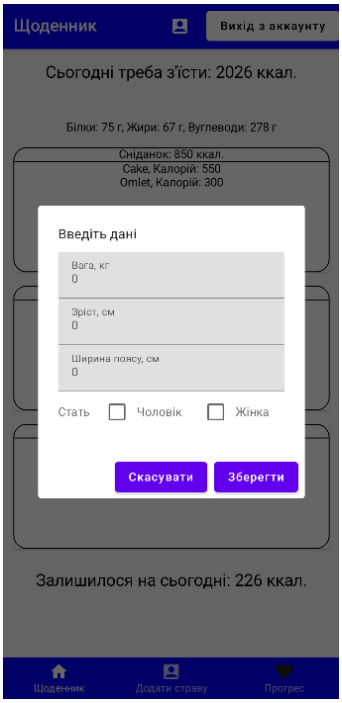

Рисунок 3.9 – Контекстне меню для редагування параметрів тіла

На сторінці «Прогрес» користувач бачить історію змін власної ваги та ширини поясу за певну дату (рис.3.10):

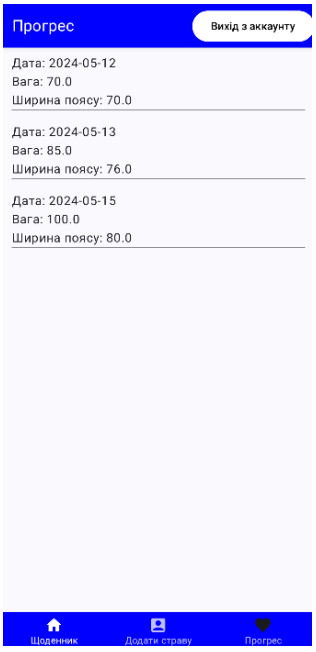

Рисунок 3.10 – Сторінка «Прогрес»

3.4 Тестування мобільного додатку

Для оцінки якості розробленого мобільного додатку необхідно виконати його тестування. Для цього використаємо метод білого ящика. Сам метод зосереджений на тестуванні внутрішнього устрою системи та буде найбільш доречним, оскільки відомо його внутрішню структуру. Для виконання тестування необхідно скласти тест-кейси та відобразити очікувані та фактичні результати.

Результати тестування відображені у таблиці 3.1:

Таблиця 3.1 – Тестування методом білого ящика

Назва	Очікуваний результат	Фактичний результат	0/1
Перевірка на правильність введення логіну та пароля при авторизації та реєстрації	Помилка, повідомлення про некоректність введення даних	Виведення помилки на екран користувача	1
Перевірка на неможливість реєстрації вже зареєстрованого користувача	Повідомлення, що користувач вже зареєстрований	Виведене повідомлення про зареєстрованого користувача	1
Перевірка на додавання страви у раціон	Додана страва в раціон	Страва додалася у раціон	1
Перевірка на видалення страви з раціону	Видалена страва з раціону	Видалена страва з раціону	1
Перевірка на коректність введення антропометричних показників користувача	Виведення помилки на екран користувача або неможливість введення букв	Можливе введення лише цифрових символів, неможливе введення букв та інших символів.	1

Продовження таблиці 3.1

Назва	Очікуваний результат	Фактичний результат	0/1
Перевірка на правильний підрахунок кількості калорій за весь день	Сумма калорій повинна співпадати	Загальний результат за співпадає сумі калорій	1
Перевірка пошуку страви у базі даних користувача	Знайдена страва у базі даних за назвою	Знайдена страва у базі даних за назвою	1
Перевірка можливості видалення страви з бази даних	Видалена страва з бази даних при натисненні на іконку видалення	Видалена страва з бази даних	1

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було створено мобільний додаток обліку спожитих калорій. У результаті роботи було зроблено аналіз предметної області, огляд останніх досліджень. Як результат, аналіз підтвердив актуальність створення мобільного додатку для ведення обліку калорій. Також було зроблено огляд аналогів, що дало змогу оцінити їх переваги та недоліки для врахування їх при створенні мобільного додатку. Результатом аналізу стало визначення функціональних та системних вимог до додатку. Також було спроектоване технічне завдання, на основі якого було створено мобільний додаток.

На основі технічного завдання було створено план виконання робіт, який включає в себе деталізацію мети методом SMART, планування структури робіт WBS та структури організації OBS, побудована діаграма Ганта та проведений аналіз ризиків.

В процесі проектування було створено контекстну діаграму в нотації IDEF0, яка дала можливість відобразити процеси у системі та її логіку роботи. Більш детальний опис процесів було зображено під час декомпозиції процесів системи, що було відображено на діаграмі. Для відображення сценаріїв взаємодії між користувачем та мобільним додатком було створено діаграму варіантів використання. Реалізація мобільного додатку була виконана засобами мови програмування Kotlin, засобами розробки інтерфейсу від компанії Google Jetpack Compose та сервісом для розробки системи керування базою даних Firebase. Результатом тестування став тест-репорт функціональних вимог додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WORLDWIDE SURVEY OF FITNESS TRENDS FOR 2020 : ACSM's Health & Fitness Journal. LWW. URL: https://journals.lww.com/acsm-healthfitness/fulltext/2019/11000/WORLDWIDE_SURVEY_OF_FITNESS_TRENDS_FOR_200.6.aspx (дата звернення: 29.04.2024)
2. Mobile applications for the sport and exercise nutritionist: a narrative review - BMC Sports Science, Medicine and Rehabilitation. SpringerLink. URL: https://link.springer.com/article/10.1186/s13102-022-00419-z (дата звернення: 24.04.2024).
3. Obesity and overweight. World Health Organization (WHO). URL: https://www.who.int/news-room/fact-sheets/detail/obesity-and overweight#:~:text=Worldwide%20adult%20obesity%20has%20more,16%%20were%20living%20with%20obesity. (дата звернення: 15.04.2024).
4. Frontiers | Body-Weight Fluctuation Was Associated With Increased Risk for Cardiovascular Disease, All-Cause and Cardiovascular Mortality: A Systematic Review and Meta-Analysis. Frontiers. URL: https://www.frontiersin.org/journals/endocrinology/articles/10.3389/fendo.2019.00728/full (дата звернення: 26.04.2024).
5. Mark M. A new predictive equation for resting energy expenditure in healthy individuals. https://www.sciencedirect.com/science/article/abs/pii/S0002916523166986?via%3Dihub. URL: https://www.sciencedirect.com/science/article/abs/pii/S0002916523166986?via=ihub (дата звернення: 16.05.2024).
6. The Efficacy of Mobile Applications for Weight Loss - Current Cardiovascular Risk Reports. SpringerLink. URL: https://link.springer.com/article/10.1007/s12170-023-00717-2 (дата звернення: 15.04.2024).

7. Sustainability of Weight Loss Through Smartphone Apps: Systematic Review and Meta-analysis on Anthropometric, Metabolic, and Dietary Outcomes. Journal of Medical Internet Research. URL: https://www.jmir.org/2022/9/e40141/ (дата звернення: 18.04.2024).
8. The Benefits of Using Mobile Apps for Tracking Nutrition. NutritionInformatics.info. URL: https://nutritioninformatics.info/the-benefits-of-using-mobile-apps-for-tracking-nutrition/ (дата звернення: 18.04.2024).
9. КУРИЛЕНКО А.О. Мобільний додаток для оптімізації роботи ритейлерів в сфері агробізнесу. URL: https://ela.kpi.ua/server/api/core/bitstreams/b35585ea-b8a2-4f69-9ba0-2c1b13492c08/content (дата звернення: 30.04.2024).
10. Цірюк С.О. Дослідження програмно-апаратних засобів оптимізації взаємодії з користувачами мобільних додатків. URL: https://openarchive.nure.ua/entities/publication/1f0bdfb0-a151-4d1b-9403-75513dacee11 (дата звернення: 30.04.2024).
11. Lose It! - Calorie counting made easy. Lose It! - Weight Loss That Fits. URL: https://www.loseit.com/ (дата звернення: 24.04.2024).
12. Lifesum - Healthy eating. Simplified. Lifesum - Healthy eating. Simplified. URL: https://lifesum.com/ (дата звернення: 22.04.2024).
13. Healthy Weight Loss & Eating: Lose Weight Fast with YAZIO. YAZIO. URL: https://www.yazio.com/en (дата звернення: 23.04.2024).
14. Kotlin Programming Language. Kotlin. URL: https://kotlinlang.org/ (дата звернення: 25.04.2024).
15. Download Android Studio & App Tools - Android Developers. Android Developers. URL: https://developer.android.com/studio (дата звернення: 29.04.2024).
16. Jetpack Compose UI App Development Toolkit - Android Developers. Android Developers. URL: https://developer.android.com/develop/ui/compose (дата звернення: 29.04.2024).
17. Firebase | Google’s Mobile and Web App Development Platform. Firebase. URL: https://firebase.google.com/ (дата звернення: 25.04.2024).

18. Методологія IDEF0 // Stud. URL: https://stud.com.ua/87184/ekonomika/metodologiya_idef0 (дата звернення: 15.05.2024)
19. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти // DOU. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 26.05.2024)
20. Maxym Z. Моделювання даних (Data Modelling). Maxym Zosym. URL: https://www.maxzosim.com/data-modelling/ (дата звернення: 16.05.2024).
21. Mounil-Shah. Mobile App Architecture: Types, Best Practices & More. Radixweb. URL: https://radixweb.com/blog/guide-to-mobile-app-architecture (дата звернення: 16.05.2024).
22. Work Breakdown Structure. workbreakdownstructure.com. URL: https://www.workbreakdownstructure.com/ (дата звернення: 20.04.2024).
23. Organization Breakdown Structure (OBS) - PSA - EN. PSA - EN. URL: https://uplandsoftware.com/psa/resources/glossary/organization-breakdown-structure-obs/#:~:text=Organization%20Breakdown%20Structure%20or%20OBS,profit%20reporting,%20and%20work%20management. (дата звернення: 22.04.2024).
24. Gantt.com. Gantt.com. URL: https://www.gantt.com/ (дата звернення: 25.04.2024).
25. ОГЛЯД ПРОЦЕСІВ УПРАВЛІННЯ РИЗИКАМИ В ІТ-ПРОЄКТАХ У КОНТЕКСТІ СТАНДАРТІВ ПРОЄКТНОГО МЕНЕДЖМЕНТУ | Управління розвитком складних систем. Управління розвитком складних систем. URL: http://mdcs.knuba.edu.ua/article/view/219812 (дата звернення: 25.04.2024).

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку мобільного додатку

« Мобільний додаток обліку спожитих калорій»

ПОГОДЖЕНО:

к.т.н., доцент

_____ Нагорний В.В

Студент групи ІТ-03

_____ Чернов О.О

Суми 2024

1 ПРИЗНАЧЕННЯ Й МЕТА СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

1.1 Призначення мобільного додатку

Призначений для обліку спожитих калорій.

1.2 Мета створення мобільного додатку

Полегшення процесу схуднення, а також надаватиме користувачам зручний та простий інструмент для ведення обліку калорій.

1.3 Цільова аудиторія

Спортсмени, які бажають досягти необхідних показників харчування при тренуваннях. Послідовники здорового способу життя, які хочуть слідкувати за власним раціоном харчування.

2 ВИМОГИ ДО МОБІЛЬНОГО ДОДАТКУ

2.1 Вимоги до мобільного додатку в цілому

2.1.1 Вимоги до структури й функціонування

Мобільний додаток повинен бути реалізований фреймворками, мовою програмування Kotlin, тощо необхідними для платформи Android. Мобільний продукт буде представлений у якості мобільного додатку з естетичним графічним інтерфейсом, наповненим необхідним функціоналом для схуднення.

2.1.2 Вимоги до персоналу

Персонал закладу не має необхідності практикувати особливі навички для роботи з мобільним додатком. Єдиною достатньою вимогою є навички користування мобільним телефоном.

2.1.3 Вимоги до збереження інформації

Уся інформацію повинна зберігатися у СКБД Firestore.

2.1.4 Вимоги до розмежування доступу

Система матиме обмеження доступу до функціоналу додатку для користувачів.

Можливості користувачів включатимуть можливість створення власного аккаунту та перегляду власного раціону харчування.

2.2 Структура мобільного додатку

2.2.1 Загальна інформація про структуру мобільного додатку

Мобільний додаток складається з 4 вкладок:

- Сторінка авторизації у власний аккаунт.
- Сторінка щоденника. Користувач матиме змогу обирати продукти харчування, кількість калорій у них та проглядати ліміт по калоріям.
- Сторінка прогресу. Користувач зможе бачити прогрес за відповідний період.
- Сторінка з додаванням страв. Можливості передбачатимуть додавання, пошук та видалення страв.

2.2.2 Навігаційне меню

Для навігації планується створення меню на нижній панелі екрану для швидкого переміщення користувачем по всім вкладкам мобільного застосунку.

2.2.3 Управління контентом

Керування інформацією всередині мобільного додатку здійснюватиметься в вкладці «Додати страву». Вся інформація міститиметься в системі управління базою даних.

2.2.4 Дизайн та структура додатку

Дизайн мобільного застосунку повинен бути реалізований у біло-синіх кольорах. Стиль графічних елементів інтерфейсу повинен бути ергономічним та зручним для користування.

Розташування елементів інтерфейсу на вході в застосунок зображено на рисунку А.1:

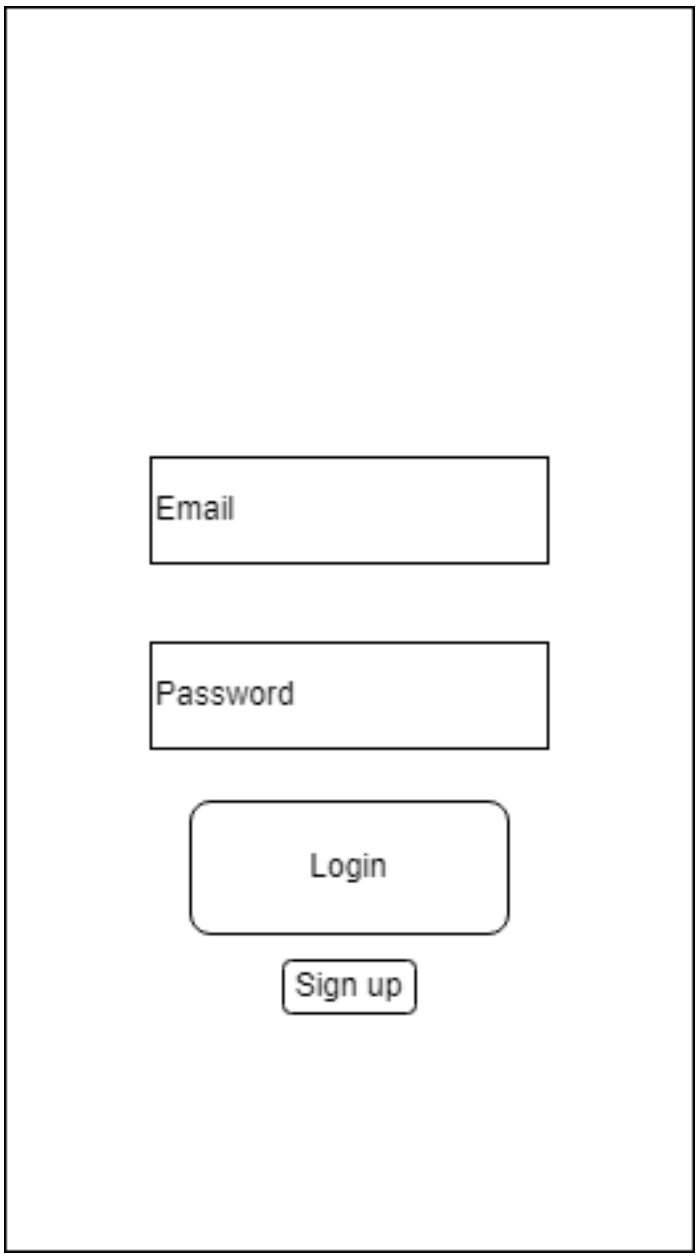

Рисунок А.1 – Схема сторінки входу в аккаунт

Розташування елементів інтерфейсу при реєстрації в застосунок зображено на рисунку А.2:

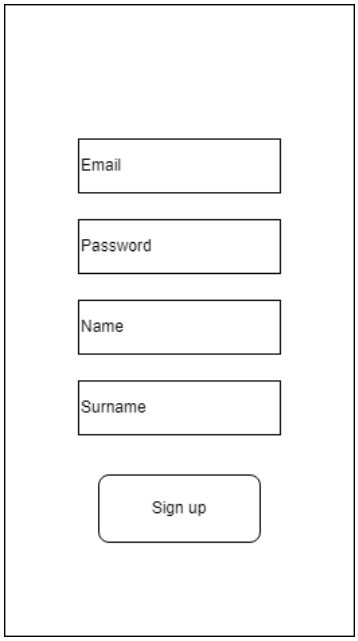

Рисунок А.2 – Схема сторінки реєстрації

Розташування елементів інтерфейсу у меню додатка зображено на рисунку А.4:

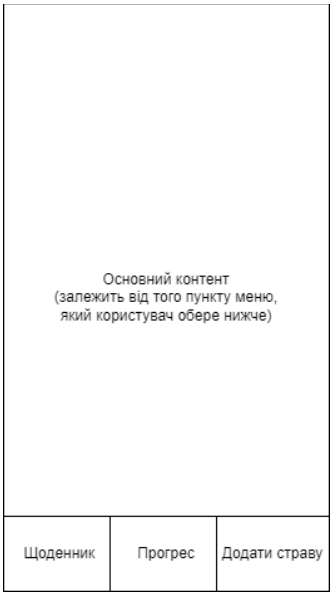

Рисунок А.3 – Схема головного меню

Розташування елементів інтерфейсу сторінки «Додати страву» зображено на рисунку А.5:

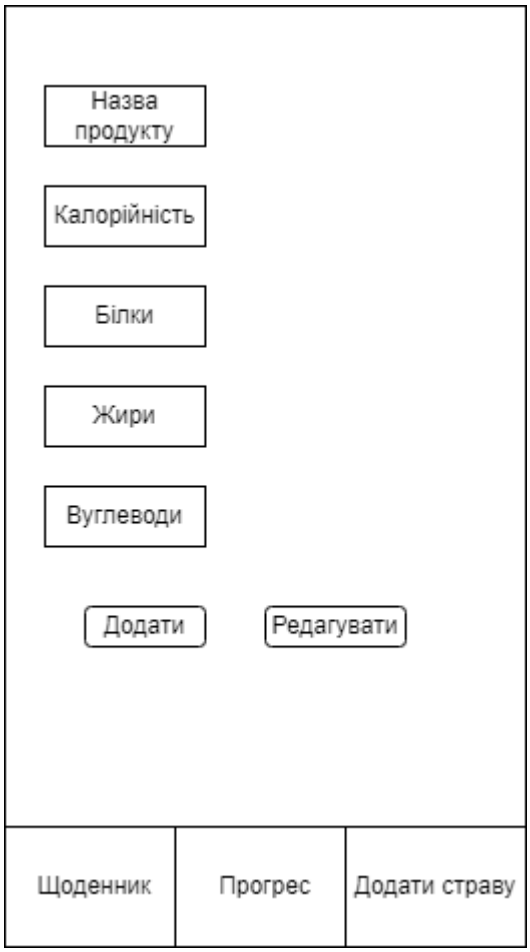

Рисунок А.4 – Сторінка ««Додати страву»

2.2.5 Карта мобільного додатку

Карта мобільного додатку зображена на рисунку А.6:

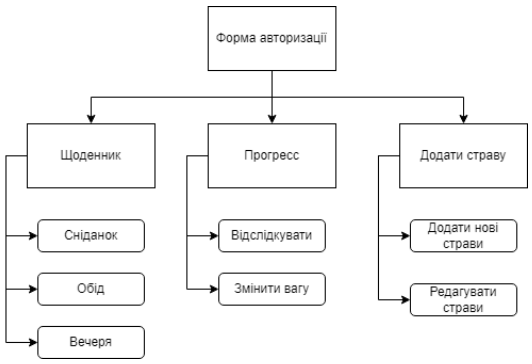

Рисунок А.5 – Карта мобільного додатку

2.3 Вимоги до видів забезпечення

2.3.1 Вимоги до лінгвістичного забезпечення

Мобільний додаток повинен бути реалізований українською мовою.

2.3.2 Вимоги до програмного забезпечення

Мобільний додаток повинен підтримувати від 12 версії Android та вище. Також повинен підтримувати застарілі версії Android нижче 12 версії.

2.4 Вимоги до функціонування системи

2.4.1 Потреби користувача

Потреби користувача представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN-01	Можливість підрахунку калорій	Користувач
UN-02	Можливість відстеження прогресу у схудненні	
UN-03	Можливість внесення нових продуктів у раціон	Користувач
UN-04	Можливість редагування власних вагових показників	Користувач
UN-05	Можливість авторизації у власний аккаунт	Користувач

2.4.2 Функціональні вимоги

На основі аналізу потреб мобільних користувачів мобільного додатку було виділено такі вимоги як:

- наявність власного аккаунту та можливість авторизації;
- можливість підрахунку калорій за день;
- можливість відстеження прогресу у схудненні;
- можливість внесення продуктів у раціон;
- можливість редагування власних вагових показників;
- редагування даних про страви

2.4.3 Системні вимоги

На основі функціональних вимог та потреб користувача було визначено системні вимоги. Системні вимоги наведені в таблиці А.2:

Таблиця А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR-01	Підрахунок калорій	M	Надає можливість підраховувати калорії на один день
SR-02	Внесення продуктів у раціон	M	Надає можливість вносити страви у раціон
SR-03	Відстеження прогресу у схудненні	M	Надає можливість відстежувати прогрес у вазі
SR-04	Редагування вагових показників	S	Надає можливість редагувати вагу
SR-05	База даних користувачів	M	База даних, яка містить дані про аккаунт користувача та його параметри тіла

3 СКЛАД І ЗМІСТ РОБІТ ЗІ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

Загальний опис етапів створення мобільного застосунку відтворено у таблиці А.3:

Таблиця А.3 – Загальний опис етапів створення мобільного застосунку

№	**Склад і зміст робіт**	**Строк розробки, дні**
1	Постановка завдання на створення мобільного застосунку	3
2	Створення технічного завдання	5
3	Створення прототипу	10
4	Процес розробки	30
5	Тестування	5
6	Написання документації до проекту	Протягом всього проекту
	Загальна тривалість робіт	53

ДОДАТОК Б

Деталізація мети проекту методом SMART. Результатом виконання проекту буде мобільний додаток для обліку спожитих калорій.

Результати деталізації проекту методом SMART відображено у таблиці Б.1.

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific	Спрощення обліку спожитих калорій користувача
Measurable	Дозволить збільшити користувачів які ведуть облік калорій в Україні на 1%
Achievable	Мета досяжна, оскільки розробник має достатній досвід розробки та закуплений необхідний комплект програмного забезпечення
Relevant	Для сприяння збільшення людей, які стежать за своїм раціоном харчування
Time-framed	Термін – до 30.06.2024

Планування змісту робіт [21]. Для планування ієрархічної структури виконання робіт використовують WBS (Work Breakdown Structure) діаграми, де графічно відображено усі задачі проекту, які необхідно виконати для досягнення мети проекту. Як головна задача зверху зображується сам програмний продукт, а задачі розбиваються до другого та третього рівнів. Виконання усіх задач повинно привести до реалізації поставлених цілей. WBS-діаграма зображена на рисунку Б.1.

Планування структури виконавців [22]. Для розподілу виконання робіт та графічного відображення поставлених задач на проекті використовують діаграми OBS (Organization Breakdown Structure). Таким чином, використання OBS діаграм дозволяє якісно визначити ролі кожного учасника проекту. OBS діаграма зображена на рисунку Б.2.

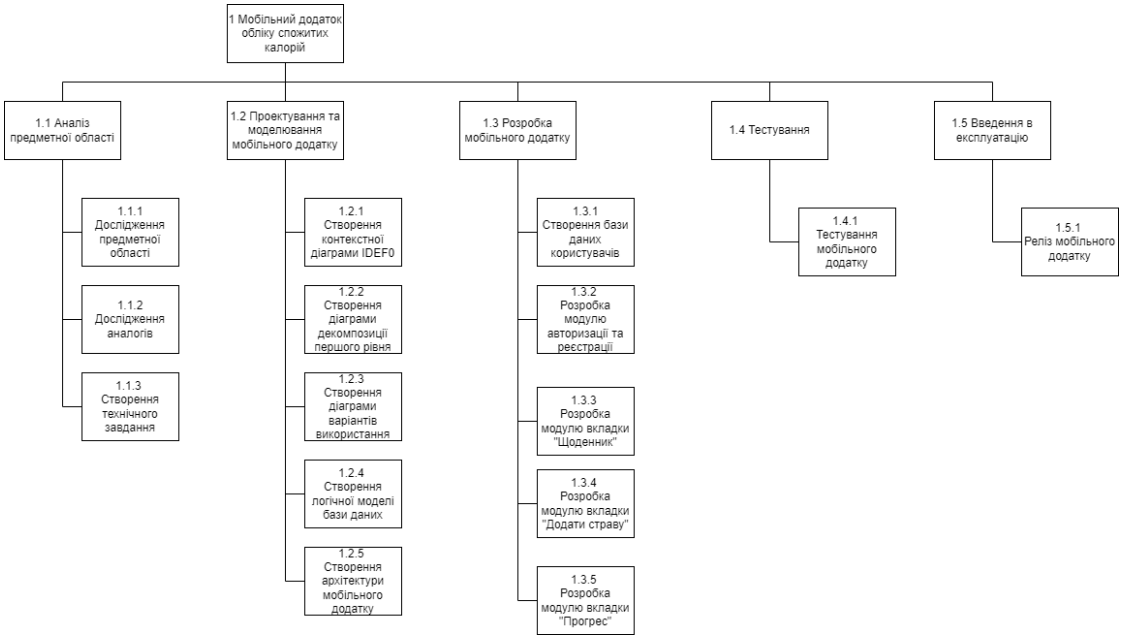

Рисунок Б.1 – Work Breakdown Structure

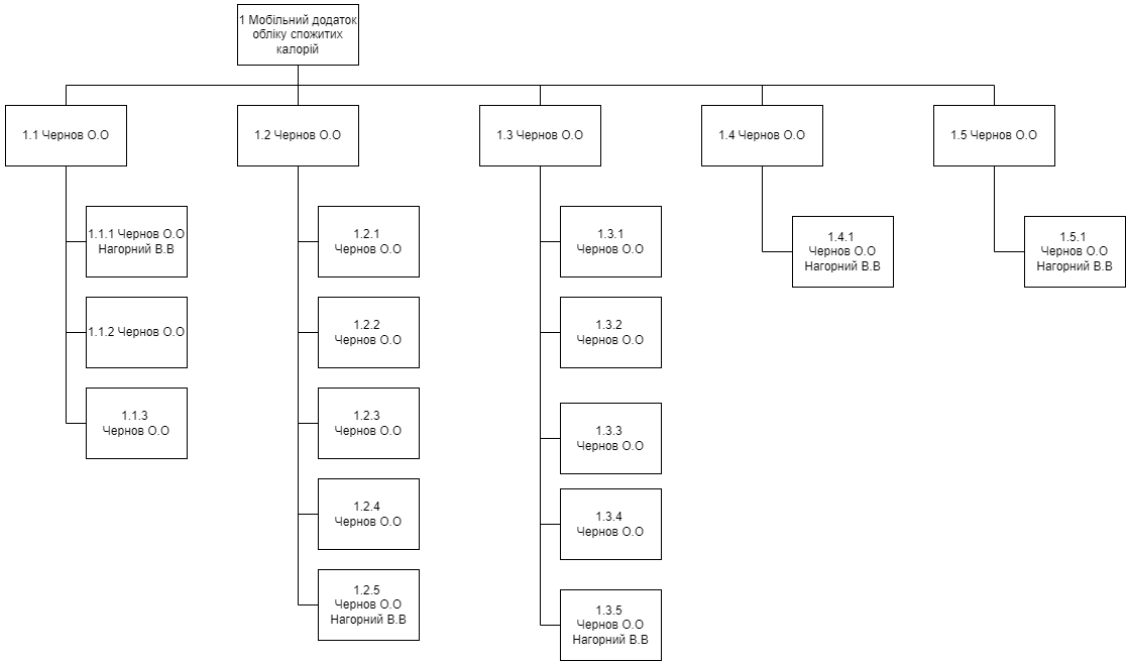

Рисунок Б.2 – Organization Breakdown Structure

Список виконавців проекту представлено у таблиці Б.2.

Таблиця Б.2 – Список виконавців проекту представлено

Роль	**Ім'я**	**Задача, яка стоїть**
Розробник	Чернов О.О	Виконує розробку додатку
Проектувальник	Чернов О.О	Розробляє структуру мобільного додатку
Тестувальник	Чернов О.О	Виконує тестування мобільного додатку
Менеджер проекту	Чернов О.О	Виконує аналіз предметної області, створює технічне завдання та графік виконання робіт
Керівник проекту	Нагорний В.В	Контролює хід виконання проекту

Діаграма Ганта [23] . Діаграма Ганта створюється для побудови календарного плану виконання робіт проекту. Діаграма дозволяє відслідкувати взаємозалежність між завданням проекту та часом на його виконання. Усі завдання представлені у вигляді смуг, які з'єднані між собою зв'язком. Довжина кожної окремої смуги відповідає за тривалість виконання кожної окремої роботи. Діаграма Ганта зображена на рисунку Б.3

Аналіз ризиків [24] . Кожен успішний проект потребує ідентифікації ризиків та їх облік. Ідентифікація ризиків дозволяє оцінити їх вплив на виконання проекту та застосувати стратегії реагування на ризики. Процес передбачає

виявлення, оцінку та управління ризиками. Основною метою цього процесу є зниження негативних наслідків ризиків на проект.

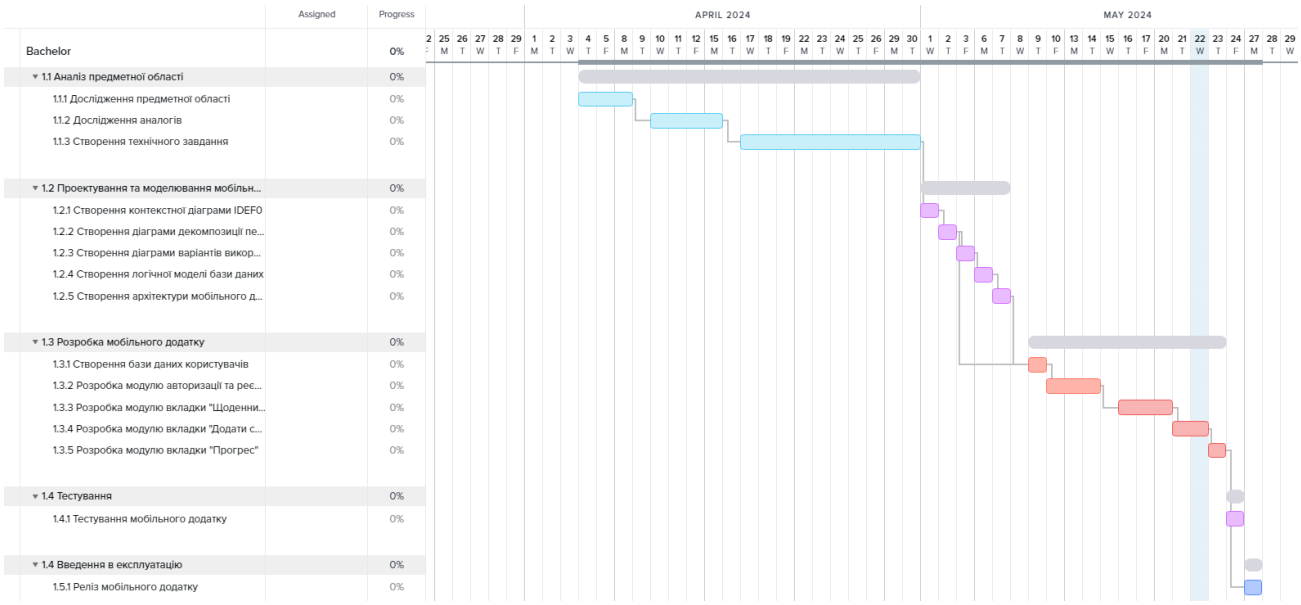

Рисунок Б.3 – Діаграма Ганта

У таблиці Б.3 представлено шкалу класифікації ризиків за впливом на проект та ймовірність виникнення.

Таблиця Б.3 – Шкала оцінювання ризиків за ймовірністю виникнення та ймовірність виникнення.

Оцінка	**Ймовірність виникнення**	**Вплив ризику**	**Тип ризику**
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

У таблиці Б.4 наведено ймовірність виникнення ризиків та їх вплив. Зеленим кольором відображено прийнятні ризики, жовтим – виправдані, а червоним – недопустимі.

Таблиця Б.4 – Матриця ймовірності та впливу згідно проекту.

Ймовірність ризику (Й)	**Вплив загрози (ризику)**				
	Дуже малий	**Малий**	**Середній**	**Великий**	**Дуже великий**
	0,05	**0,1**	**0,2**	**0,4**	**0,8**
0,9	0,045	0,09	0,18	0,36	0,72
0,7	0,035	0,07	0,14	0,28	0,56 R1
0,5	0,025	0,05	0,1	0,2 R2	0,4 R7
0,3	0,015	0,03	0,06 R8,R9	0,12 R5,R6	0,24 R4
0,1	0,005	0,01 R10	0,02	0,04 R3	0,08

Таблиця Б.5 – Ризики проекту та стратегії реагування

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив	Ранг)	Тип стратегії	План А (заходи запобігання виникненню ризика)	План Б (заходи усунення наслідків ризику)
1	Виправданий	Витік даних користувачів	0.2	0.7	0.14	Зменшення	Проведення тестувань з надійності системи	Усунення наслідків витоку
2	Виправданий	Вимкнення світла	0.14	1	0.14	Ухилення	Закупівля генератора	Пошук альтернативного живлення

Продовження таблиці Б.5

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив	Ранг)	Тип стратегії	План А (заходи запобігання виникненню ризика)	План Б (заходи усунення наслідків ризику)
3	Виправданий	Затримки реалізації проекту	0.2	0.5	0.1	Ухилення	Проведення ретельного планування проекту	Відтермінування проекту
4	Виправданий	Помилки при проектуванні бази даних	0.3	0.8	0.14	Зменшення	Проведення перевірки проектування БД	Виправлення в ході роботи

Продовження таблиці Б.5

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив	Ранг)	Тип стратегії	План А (заходи запобігання виникненню ризика)	План Б (заходи усунення наслідків ризику)
5	Виправданий	Поява серйозних помилок	0.2	0.5	0.1	Ухилення	Звернення до керівника роботи	Виправлення в ході роботи
6	Прийнятний	Поява помилок при тестуванні додатку	0.1	0.5	0.05	Зменшення	Розробка тест-кейсів в ході роботи над модулями	Виправлення помилок на етапі виявлення

Продовження таблиці Б.5

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив	Ранг)	Тип стратегії	План А (заходи запобігання виникненню ризика)	План Б (заходи усунення наслідків ризику)
7	Виправданий	Затримки в релізі додатку	0.3	0.4	0.12	Зменшення	–	Виправлення в ході роботи
8	Виправданий	Неактуальність додатку на момент випуску	0.3	0.4	0.12	Ухилення	Аналіз ринку аналогічних додатків	–
9	Виправданий	Проблеми з документуванням проекту	0.2	0.7	0.14	Зменшення	Узгодження форми документування з керівником проекту	Виправлення в ході роботи

ДОДАТОК В

Лістинг програмного коду основних модулів мобільного додатку

User.kt:

```
package com.example.caloriesapp

data class User(
    val weight: Int = 0,
    val height: Int = 0,
    val gender: String = "",
    val date: String = "",
    val bellyWidth: Int = 0
)
```

Dish.kt:

```
package com.example.caloriesapp

data class Dish(
    val id: String = "",
    val name: String = "",
    val calories: Int = 0,
    val proteins: Int = 0,
    val fats: Int = 0,
    val carbohydrates: Int = 0
)

fun Dish.displayDetailsforCard(): String {
    return "$name, Калорій: $calories"
}

fun Dish.displayDetails(): String {
```

```
    return "$name, Калорій: $calories, Білків: $proteins, Жирів: $fats, Вуглеводів:
$carbohydrates"
}
```

NavigationGraph.kt:

```
package com.example.caloriesapp

import androidx.compose.runtime.Composable
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable

@Composable
fun NavigationGraph(
    navController: NavHostController
) {
    NavHost(
        navController = navController,
        startDestination = Screen.LoginScreen.route
    ) {
        composable(Screen.SignupScreen.route) {
            SignupScreen(navController = navController)
        }
        composable(Screen.LoginScreen.route) {
            LoginScreen(navController = navController)
        }

        composable(Screen.Homescreen.route) {
            HomeScreen(navController = navController)
        }

        composable(Screen.AddingDishcreen.route) {
            DishAddingScreen(navController = navController)
        }

        composable(Screen.Progressscreen.route) {
            ProgressScreen(navController = navController)
        }
```

```
    }
}
```

Screen.kt:

```
package com.example.caloriesapp

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Home
import androidx.compose.ui.graphics.vector.ImageVector

sealed class Screen(val route:String){
    object LoginScreen : Screen("loginscreen")
    object SignupScreen : Screen("signupscreen")
    object Homescreen : Screen("homescreen")
    object AddingDishcreen : Screen("addingdishscreen")
    object Progressscreen : Screen("progressscreen")

}
```

MainActivity.kt:

```
package com.example.caloriesapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.navigation.compose.rememberNavController
import com.example.caloriesapp.ui.theme.CaloriesAppTheme

class MainActivity : ComponentActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            CaloriesAppTheme {
                val navController = rememberNavController()
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    NavigationGraph(navController = navController)
                }
            }
        }
    }
}
```

LoginScreen.kt:

```
package com.example.caloriesapp

import androidx.compose.foundation.Image
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
```

```
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
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.layout.ContentScale
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.google.firebase.auth.FirebaseAuth

@Composable
fun LoginScreen(
    navController: NavController,
    viewModel: FirebaseViewModel = androidx.lifecycle.viewmodel.compose.viewModel(),
) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    val focusManager = LocalFocusManager.current
    var errorDialogMessage by remember { mutableStateOf<String?>(null) }
    Box(
        modifier = Modifier.fillMaxSize()
    ) {

        Image(
            painter = painterResource(id = R.drawable.picture),
```

```
    contentDescription = null,
    modifier = Modifier.fillMaxSize(),
    contentScale = ContentScale.FillBounds
)

LazyColumn(
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center,
    modifier = Modifier
        .fillMaxSize()
        .clickable() { focusManager.clearFocus() }
) {
    item {
        Text(
            text = "Увійти",
            fontSize = 36.sp,
            color = Color.White,
            fontFamily = FontFamily.Serif,
            fontWeight = FontWeight.Bold,
            modifier = Modifier
                .padding(15.dp),
            onTextLayout = {}
        )
        TextField(
            value = email,
            onValueChange = { email = it },
            placeholder = { Text(text = "Email", onTextLayout = {}) })
        Spacer(modifier = Modifier.height(16.dp))
        TextField(
            value = password,
            onValueChange = { password = it },
            placeholder = { Text(text = "Password", onTextLayout = {}) })

        if (errorDialogMessage != null) {
            AlertDialog(
                onDismissRequest = { errorDialogMessage = null },
                title = { Text("Помилка") },
                text = { Text(errorDialogMessage!!) },
```

```
                        confirmButton = {
                            Button(onClick = { errorDialogMessage = null }) {
                                Text("OK")
                            }
                        }
                    )
                }
                Button(
                    onClick = {
                        viewModel.logInToAccount(email, password,
                            onError = { errorMessage ->
                                errorDialogMessage = errorMessage
                            },
                            onLoggedIn = {
                                navController.navigate(Screen.Homescreen.route)
                            }
                        )
                    },
                    modifier = Modifier.padding(8.dp),
                    colors = ButtonDefaults.buttonColors(Color.Blue),
                ) {
                    Text(text = "Увійти", onTextLayout = {})
                }
                Spacer(modifier = Modifier.height(16.dp))
                Row() {
                    Text("Не зареєстровані? Натисніть щоб зареєструватися",
                        modifier = Modifier.clickable {
navController.navigate(Screen.SignupScreen.route) },
                        color = Color.White
                    )
                }
            }
        }
    }
}
```

SignupScreen.kt:

```
package com.example.caloriesapp
```

```
import androidx.compose.foundation.Image
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
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.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import com.google.firebase.FirebaseException
import com.google.firebase.auth.FirebaseAuthException

@Composable
fun SignupScreen(
```

```
    navController: NavController,
    viewModel: FirebaseViewModel = androidx.lifecycle.viewmodel.compose.viewModel()
) {

    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var errorText by remember { mutableStateOf("") }
    val focusManager = LocalFocusManager.current


    Box(
        modifier = Modifier.fillMaxSize()
    ) {

        Image(
            painter = painterResource(id = R.drawable.picture),
            contentDescription = null,
            modifier = Modifier.fillMaxSize(),
            contentScale = ContentScale.FillBounds
        )
        LazyColumn(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp)
                .clickable { focusManager.clearFocus() },
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            item {
                Text(
                    text = "Зареєструватися",
                    fontSize = 36.sp,
                    fontFamily = FontFamily.Serif,
                    color = Color.White,
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier.padding(15.dp)
                )
                Spacer(modifier = Modifier.height(16.dp))
```

```
            TextField(
                value = email,
                onValueChange = { email = it },
                placeholder = { Text(text = "Email") })
            Spacer(modifier = Modifier.height(16.dp))
            TextField(
                value = password,
                onValueChange = { password = it },
                placeholder = { Text(text = "Пароль") })
            Spacer(modifier = Modifier.height(16.dp))

            if (errorText.isNotEmpty()) {
                AlertDialog(
                    onDismissRequest = { errorText = "" },
                    title = { Text("Помилка") },
                    text = { Text(errorText) },
                    confirmButton = {
                        Button(onClick = { errorText = "" }) {
                            Text("OK")
                        }
                    }
                )
            }

            Button(
                onClick = {
                    if (password.length < 6) {
                        errorText =
                            "Пароль повинен бути не менше 6 символів. Будь ласка,
спробуйте ще раз."
                    } else {
                        viewModel.signInToAccount(email, password,
                            onSuccess = {
                                navController.navigate(Screen.Homescreen.route)
                            },
                            onError = { errorMessage ->
                                errorText = errorMessage
                            }
```

```
                        )
                    }
                },
                modifier = Modifier.padding(8.dp),
                colors = ButtonDefaults.buttonColors(Color.Blue),
            ) {
                Text(text = "Зареєструватися")
            }

            Spacer(modifier = Modifier.height(16.dp))
            Row() {
                Text(
                    "Вже зареєстровані у додатку? Натисніть щоб увійти.",
                    modifier = Modifier.clickable { navController.navigate(Screen.LoginScreen.route) },
                    color = Color.White
                )
            }
        }
    }
}
```

DailyScreen.kt:

```
package com.example.caloriesapp

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AccountBox
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Favorite
```

```
import androidx.compose.material.icons.filled.Home
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

@Composable
fun HomeScreen(
    navController: NavController,
    viewModel: FirebaseViewModel = androidx.lifecycle.viewmodel.compose.viewModel()
) {
    var showDialogForBreakfast by remember { mutableStateOf(false) }
    var showDialogForLunch by remember { mutableStateOf(false) }
    var showDialogForDinner by remember { mutableStateOf(false) }
    var selectedBreakfastDishes by remember { mutableStateOf(mutableListOf<Dish>()) }
    var selectedLunchDishes by remember { mutableStateOf(mutableListOf<Dish>()) }
    var selectedDinnerDishes by remember { mutableStateOf(mutableListOf<Dish>()) }
    var showDialog by remember { mutableStateOf(false) }
    var snackbarVisible by remember { mutableStateOf(false) }
    var requiredCalories by remember { mutableStateOf(0) }
    var sumOfBreakfastCalories by remember { mutableStateOf(0) }
    var sumOfLunchCalories by remember { mutableStateOf(0) }
    var sumOfDinnerCalories by remember { mutableStateOf(0) }
    var proteinGrams by remember { mutableStateOf(0) }
    var fatGrams by remember { mutableStateOf(0) }
    var carbohydrateGrams by remember { mutableStateOf(0) }
    var currentDate = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).format(Date())
    val fs = Firebase.firestore
```

```
    val currentUser = FirebaseAuth.getInstance().currentUser

    Column(
        modifier = Modifier.fillMaxSize()
    ) {
        TopAppBar(
            title = {
                Text(
                    text = "Щоденник",
                    fontSize = 20.sp,
                    color = Color.White
                )
            },
            actions = {
                UserInputDialog(
                    showDialog = showDialog,
                    currentDate = currentDate,
                    onDismissRequest = { showDialog = false },
                    onSave = { userWeight, userHeight, userGender, currentDate,
bellyWidth ->
                        currentUser?.let { user ->
                            val userId = user.uid
                            val user = User(userWeight, userHeight, userGender,
currentDate,bellyWidth)
                            fs.collection("users")
                                .document(userId)
                                .collection("data")
                                .document(currentDate)
                                .set(user)
                                .addOnSuccessListener {
                                    snackbarVisible = true
                                }
                                .addOnFailureListener { e ->
                                }
                        }
                    }
                )
                UserInputDialogButton(onClick = { showDialog = true })
```

```
            Button(
                onClick = {
                    viewModel.signOut {
                        navController.navigate(Screen.LoginScreen.route)
                    }
                },
                colors = ButtonDefaults.buttonColors(Color.White)
            ) {
                Text(text = "Вихід з аккаунту", color = Color.Black)
            }
        },
        backgroundColor = Color.Blue,
        modifier = Modifier.fillMaxWidth()
    )

    LaunchedEffect(Unit) {
        viewModel.getRequiredMacronutrients(currentDate) { calories, protein, fat,
carbohydrate ->
            requiredCalories = calories
            proteinGrams = protein
            fatGrams = fat
            carbohydrateGrams = carbohydrate
        }
    }



    viewModel.startListeningForSelectedDishes(
        onSuccess = { breakfastDishes, lunchDishes, dinnerDishes ->
            selectedBreakfastDishes = breakfastDishes.toMutableList()
            selectedLunchDishes = lunchDishes.toMutableList()
            selectedDinnerDishes = dinnerDishes.toMutableList()
        },
        onFailure = { exception ->

        }
    )
```

```
        viewModel.sumForSelectedDishes(
            onSuccess = { breakfastCalories, lunchCalories, dinnerCalories ->
                sumOfBreakfastCalories = breakfastCalories
                sumOfLunchCalories = lunchCalories
                sumOfDinnerCalories = dinnerCalories
            },
            onFailure = { exception ->
            }
        )

        Column(
            modifier = Modifier
                .weight(1f)
                .padding(16.dp)
                .fillMaxSize(),
            verticalArrangement = Arrangement.Top,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = "Сьогодні треба з'їсти: $requiredCalories ккал.",
                fontSize = 20.sp,
                modifier = Modifier.padding(bottom = 46.dp,),
                onTextLayout = {}
            )

            Text(
                text = "Білки: $proteinGrams г, Жири: $fatGrams г, Вуглеводи:
$carbohydrateGrams г",
                modifier = Modifier.padding(bottom = 16.dp),
                onTextLayout = {}
            )

            Column(
                modifier = Modifier.fillMaxWidth()
            ) {
                AddRectangle(
                    title = "Сніданок: $sumOfBreakfastCalories ккал.",
                    selectedDishes = selectedBreakfastDishes,
```

```
                onClick = { showDialogForBreakfast = true }
            )
            Spacer(modifier = Modifier.height(16.dp))
            AddRectangle(
                title = "Обід: $sumOfLunchCalories ккал.",
                selectedDishes = selectedLunchDishes,
                onClick = { showDialogForLunch = true }
            )
            Spacer(modifier = Modifier.height(16.dp))
            AddRectangle(
                title = "Вечеря: $sumOfDinnerCalories ккал.",
                selectedDishes = selectedDinnerDishes,
                onClick = { showDialogForDinner = true }
            )
        }

        Spacer(modifier = Modifier.height(26.dp))
        val caloriesSumPerDay = sumOfBreakfastCalories + sumOfLunchCalories +
sumOfDinnerCalories
        val caloriesLimitPerDay = requiredCalories - caloriesSumPerDay
        val remainingCalories = if (caloriesLimitPerDay < 0) {
            "Ви перевищили кількість калорій на день на ${-caloriesLimitPerDay}
ккал."
        } else {
            "Залишилося на сьогодні: $caloriesLimitPerDay ккал."
        }

        Text(
            text = remainingCalories,
            fontSize = 20.sp,
            modifier = Modifier.padding(bottom = 22.dp),
            textAlign = TextAlign.Left,
            onTextLayout = {}
        )
        if (showDialogForBreakfast || showDialogForLunch || showDialogForDinner) {
            ShowDialogForDishSelection(
                showDialogForBreakfast = showDialogForBreakfast,
                showDialogForLunch = showDialogForLunch,
```

```
                showDialogForDinner = showDialogForDinner,
                selectedBreakfastDishes = selectedBreakfastDishes,
                selectedLunchDishes = selectedLunchDishes,
                selectedDinnerDishes = selectedDinnerDishes,
                onDismissRequest = {
                    showDialogForBreakfast = false
                    showDialogForLunch = false
                    showDialogForDinner = false
                },
                onSelectedDishesChanged    =    {    breakfastDishes,    lunchDishes,
dinnerDishes ->
                    viewModel.setSelectedBreakfastDishesForCurrentUser(
                        selectedBreakfastDishes = breakfastDishes,
                        onSuccess = {

                        },
                        onFailure = { e ->

                        }
                    )
                    viewModel.setSelectedLunchDishesForCurrentUser(
                        selectedLunchDishes = lunchDishes,
                        onSuccess = {

                        },
                        onFailure = { e ->

                        }
                    )
                    viewModel.setSelectedDinnerDishesForCurrentUser(
                        selectedDinnerDishes = dinnerDishes,
                        onSuccess = {

                        },
                        onFailure = { e ->

                        }
                    )
```

```
            }
        )
    }
}

BottomNavigation(
    modifier = Modifier
        .fillMaxWidth()
        .height(56.dp)
        .padding(3.dp),
    backgroundColor = Color.Blue,
    contentColor = Color.White
) {
    BottomNavigationItem(
        selected = true,
        onClick = { navController.navigate(Screen.Homescreen.route) },
        icon = {
            Icon(
                imageVector = Icons.Default.Home,
                contentDescription = ""
            )
        },
        label = { Text(text = "Щоденник", onTextLayout = {}) }
    )
    BottomNavigationItem(
        selected = false,
        onClick = { navController.navigate(Screen.AddingDishcreen.route) },
        icon = {
            Icon(
                imageVector = Icons.Default.AccountBox,
                contentDescription = ""
            )
        },
        label = { Text(text = "Додати страву", onTextLayout = {}) }
    )

    BottomNavigationItem(
```

```
                selected = false,
                onClick = { navController.navigate(Screen.Progressscreen.route) },
                icon = {
                    androidx.compose.material3.Icon(
                        imageVector = Icons.Default.Favorite,
                        contentDescription = ""
                    )
                },
                label = { Text(text = "Прогрес", onTextLayout = {}) }
            )

        }
    }
}


@Composable
fun AddRectangle(title: String, selectedDishes: List<Dish>, onClick: () -> Unit) {

    Card(
        modifier = Modifier
            .fillMaxWidth()
            .height(150.dp)
            .background(Color.Transparent, shape = RoundedCornerShape(8.dp))
            .border(1.dp, Color.Black, shape = RoundedCornerShape(16.dp))
            .clickable { onClick.invoke() },
    )
    {
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
            Text(text = title, color = Color.Black, onTextLayout = {})
            Divider(color = Color.Black, thickness = 1.dp)
            if (selectedDishes.isNotEmpty()) {
                selectedDishes.forEach { dish ->
                    Text("${dish.displayDetailsforCard()}", onTextLayout = {})
                }
            } else {
                Text("Не выбрано", onTextLayout = {})
            }
```

```
        }
    }
}

@Composable
fun UserInputDialog(
    showDialog: Boolean,
    currentDate: String,
    onDismissRequest: () -> Unit,
    onSave: (weight: Int, height: Int, gender: String, currentDate: String,bellyWidth:
Int) -> Unit
) {
    var weight by remember { mutableStateOf(0) }
    var height by remember { mutableStateOf(0) }
    var gender by remember { mutableStateOf("") }
    var bellyWidth by remember { mutableStateOf(0) }



    if (showDialog) {
        var isMale by remember { mutableStateOf(false) }
        var isFemale by remember { mutableStateOf(false) }

        AlertDialog(
            onDismissRequest = onDismissRequest,
            title = { Text(text = "Введіть дані") },
            text = {
                Column {
                    TextField(
                        value = weight.toString(),
                        onValueChange = { weight = it.toIntOrNull() ?: 0 },
                        label = { Text(text = "Вага, кг") }
                    )
                    TextField(
                        value = height.toString(),
                        onValueChange = { height = it.toIntOrNull() ?: 0 },
                        label = { Text(text = "Зріст, см") }
                    )
```

```
            TextField(
                value = bellyWidth.toString(),
                onValueChange = { bellyWidth = it.toIntOrNull() ?: 0 },
                label = { Text(text = "Ширина поясу, см") }
            )



            Row(verticalAlignment = Alignment.CenterVertically) {
                Text(text = "Стать")
                Spacer(modifier = Modifier.width(8.dp))
                Checkbox(
                    checked = isMale,
                    onCheckedChange = { isChecked ->
                        if (isChecked) {
                            isMale = true
                            isFemale = false
                            gender = "Male"
                        }
                    },
                    colors = CheckboxDefaults.colors(Color.Blue)
                )
                Text(text = "Чоловік")
                Spacer(modifier = Modifier.width(16.dp))
                Checkbox(
                    checked = isFemale,
                    onCheckedChange = { isChecked ->
                        if (isChecked) {
                            isFemale = true
                            isMale = false
                            gender = "Female"
                        }
                    },
                    colors = CheckboxDefaults.colors(Color.Blue)
                )
                Text(text = "Жінка")
            }
        }
    },
```

```
            confirmButton = {
                Button(
                    onClick = {
                        onDismissRequest()
                        onSave(weight, height, gender, currentDate, bellyWidth)
                    }
                ) {
                    Text(text = "Зберегти")
                }
            },
            dismissButton = {
                Button(
                    onClick = onDismissRequest
                ) {
                    Text(text = "Скасувати")
                }
            }
        )
    }
}

@Composable
fun UserInputDialogButton(onClick: () -> Unit) {
    IconButton(
        onClick = onClick,
        modifier = Modifier.padding(end = 8.dp)
    ) {
        Icon(
            imageVector = Icons.Default.AccountBox,
            contentDescription = "User Input Dialog",
            tint = Color.White
        )
    }
}



@Composable
fun ShowDialogForDishSelection(
```

```
    showDialogForBreakfast: Boolean,
    showDialogForLunch: Boolean,
    showDialogForDinner: Boolean,
    selectedBreakfastDishes: MutableList<Dish>,
    selectedLunchDishes: MutableList<Dish>,
    selectedDinnerDishes: MutableList<Dish>,
    onDismissRequest: () -> Unit,
    onSelectedDishesChanged: (List<Dish>, List<Dish>, List<Dish>) -> Unit,
) {
    if (showDialogForBreakfast || showDialogForLunch || showDialogForDinner) {
        ShowDialogForMealSelection(
            mealType = when {
                showDialogForBreakfast -> "breakfast"
                showDialogForLunch -> "lunch"
                else -> "dinner"
            },
            selectedDishes = when {
                showDialogForBreakfast -> selectedBreakfastDishes
                showDialogForLunch -> selectedLunchDishes
                else -> selectedDinnerDishes
            },
            onDismissRequest = onDismissRequest,
            onSelectedDishesChanged = { selectedDishes ->
                when {
                    showDialogForBreakfast  ->  onSelectedDishesChanged(selectedDishes,
selectedLunchDishes, selectedDinnerDishes)
                    showDialogForLunch                                          ->
onSelectedDishesChanged(selectedBreakfastDishes, selectedDishes, selectedDinnerDishes)
                    else      ->      onSelectedDishesChanged(selectedBreakfastDishes,
selectedLunchDishes, selectedDishes)
                }
            }
        )
    }
}


@Composable
```

```
fun ShowDialogForMealSelection(
    mealType: String,
    selectedDishes: List<Dish>,
    onDismissRequest: () -> Unit,
    onSelectedDishesChanged: (List<Dish>) -> Unit,
) {
    val fs = Firebase.firestore
    var list by remember { mutableStateOf(emptyList<Dish>()) }

    LaunchedEffect(true) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid
        userId?.let { uid ->
            val                                userDishesRef                                =
fs.collection("users").document(uid).collection("dishes")

            userDishesRef.get().addOnCompleteListener { dishes ->
                if (dishes.isSuccessful) {
                    list = dishes.result.toObjects(Dish::class.java)
                }
            }
        }
    }


    val selectedList = selectedDishes.toMutableList()

    AlertDialog(
        onDismissRequest = onDismissRequest,
        title = { Text("$mealType", onTextLayout = {}) },
        text = {
            Column {
                list.forEach { dish ->
                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        modifier = Modifier.clickable {
                            selectedList.add(dish)
                        }
```

```
                    ) {
                        Text(
                            text = dish.displayDetails()
                        )
                    }
                    IconButton(
                        onClick = {
                            val documentId = dish.id

FirebaseViewModel().deleteDishFromSelectedDishesList(documentId, mealType, {
                                selectedList.remove(dish)
                            }, { exception ->
                            })
                        }
                    ) {
                        Icon(
                            imageVector = Icons.Default.Delete,
                            contentDescription = "Delete",
                            tint = Color.Red
                        )
                    }
                }
            }
        },
        confirmButton = {
            Button(
                onClick = {
                    onSelectedDishesChanged(selectedList.toList())
                    onDismissRequest()
                }
            ) {
                Text("OK")
            }
        }
    )
}
```

DishAddingScreen.kt

```
package com.example.caloriesapp


import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.material.BottomNavigation
import androidx.compose.material.BottomNavigationItem
import androidx.compose.material.IconButton
import androidx.compose.material.TopAppBar
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AccountBox
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Home
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.Snackbar
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
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.platform.LocalFocusManager
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase

@Composable
fun DishAddingScreen(
    navController: NavController,
    viewModel: FirebaseViewModel = androidx.lifecycle.viewmodel.compose.viewModel()
){
    var id by remember { mutableStateOf("") }
    var name by remember { mutableStateOf("") }
    var calories by remember { mutableStateOf(0) }
    var proteins by remember { mutableStateOf(0) }
    var fats by remember { mutableStateOf(0) }
    var carbohydrates by remember { mutableStateOf(0) }

    val focusManager = LocalFocusManager.current
    val fs = Firebase.firestore

    var snackbarVisible by remember { mutableStateOf(false) }
    val snackbarMessage = "Операцію виконано успішно"

    var searchQuery by remember { mutableStateOf("") }
    var searchResult by remember { mutableStateOf<List<Dish>>(emptyList()) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .clickable() { focusManager.clearFocus() },
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        TopAppBar(
            title = {
                Text(
```

```
            text = "Додати страву",
            fontSize = 20.sp,
            color = Color.White
        )
    },
    actions = {

        Button(
            onClick = {
                viewModel.signOut {
                    navController.navigate(Screen.LoginScreen.route)
                }
            },
            colors = ButtonDefaults.buttonColors(Color.White)
        ) {
            Text(text = "Вихід з аккаунту", color = Color.Black)
        }
    },
    backgroundColor = Color.Blue,
    modifier = Modifier.fillMaxWidth()
)

Column(
    modifier = Modifier
        .weight(1f)
        .padding(16.dp)
        .fillMaxSize(),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
) {
    TextField(
        value = searchQuery,
        onValueChange = { searchQuery = it },
        label = { Text("Пошук за назвою") }
    )


    Button(
```

```
    onClick = {
        viewModel.searchDishes(searchQuery) { dishes ->
            searchResult = dishes
        }
    },
    colors = ButtonDefaults.buttonColors(Color.Blue),
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Пошук")
}

searchResult.forEach { dish ->
    Row(
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(text = dish.name)
        Spacer(modifier = Modifier.width(8.dp))
        Text(text = "${dish.calories} ккал")
        IconButton(
            onClick = {
                viewModel.deleteDishByName(dish.name,
                    onSuccess = {
                        snackbarVisible = true
                    },
                    onFailure = { exception ->
                    }
                )
            }
        ) {
            Icon(Icons.Default.Delete, contentDescription = "Delete")
        }

    }
}
Spacer(modifier = Modifier.height(16.dp))
TextField(
    value = id,
    onValueChange = { id = it },
```

```
    label = { Text("ID") }
)
TextField(
    value = name,
    onValueChange = { name = it },
    label = { Text("Name") }
)
TextField(
    value = calories.toString(),
    onValueChange = { calories = it.toInt() },
    label = { Text("Calories") }
)
TextField(
    value = proteins.toString(),
    onValueChange = { proteins = it.toInt()},
    label = { Text("Proteins") }
)
TextField(
    value = fats.toString(),
    onValueChange = { fats = it.toInt()},
    label = { Text("Fats") }
)
TextField(
    value = carbohydrates.toString(),
    onValueChange = { carbohydrates = it.toInt() },
    label = { Text("Carbohydrates") }
)

Button(
    onClick = {
        val dish = Dish(
            id = id,
            name = name,
            calories = calories,
            proteins = proteins,
            fats = fats,
            carbohydrates = carbohydrates
        )
```

```
            val currentUser = FirebaseAuth.getInstance().currentUser
            val userId = currentUser?.uid

            userId?.let { uid ->
                fs.collection("users").document(uid).collection("dishes")
                    .add(dish)
                    .addOnSuccessListener {
                        snackbarVisible = true
                    }
                    .addOnFailureListener { e ->
                    }
            }
        },
        colors = ButtonDefaults.buttonColors(Color.Blue),
        modifier = Modifier.padding(top = 16.dp)
    ) {
        Text(text = "Додати страву")
    }


    if (snackbarVisible) {
        Snackbar(
            modifier = Modifier.padding(16.dp),
            action = {
                Button(
                    onClick = { snackbarVisible = false }
                ) {
                    Text("Закрити")
                }
            }
        ) {
            Text(snackbarMessage)
        }
    }

}
```

```
        BottomNavigation(
            modifier = Modifier
                .fillMaxWidth()
                .height(56.dp)
                .padding(3.dp),
            backgroundColor = Color.Blue,
            contentColor = Color.White
        ) {
            BottomNavigationItem(
                selected = true,
                onClick = { navController.navigate(Screen.Homescreen.route) },
                icon = {
                    androidx.compose.material.Icon(
                        imageVector = Icons.Default.Home,
                        contentDescription = ""
                    )
                },
                label = { androidx.compose.material.Text(text = "Щоденник", onTextLayout
= {}) }
            )
            BottomNavigationItem(
                selected = false,
                onClick = { navController.navigate(Screen.AddingDishcreen.route) },
                icon = {
                    androidx.compose.material.Icon(
                        imageVector = Icons.Default.AccountBox,
                        contentDescription = ""
                    )
                },
                label = {
                    androidx.compose.material.Text(
                        text = "Додати страву",
                        onTextLayout = {})
                }
            )

            BottomNavigationItem(
                selected = false,
```

```
                onClick = { navController.navigate(Screen.Progressscreen.route) },
                icon = {
                    androidx.compose.material3.Icon(
                        imageVector = Icons.Default.Favorite,
                        contentDescription = ""
                    )
                },
                label = { androidx.compose.material.Text(text = "Прогрес", onTextLayout
= {}) }
            )

        }
    }
}
```

ProgressScreen.kt:

```
package com.example.caloriesapp

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.BottomNavigation
import androidx.compose.material.BottomNavigationItem
import androidx.compose.material.Divider
import androidx.compose.material.TopAppBar
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AccountBox
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Home
import androidx.compose.material3.Button
```

```
import androidx.compose.material3.ButtonDefaults
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.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController

@Composable
fun ProgressScreen(
    navController: NavController,
    viewModel: FirebaseViewModel = androidx.lifecycle.viewmodel.compose.viewModel()
) {
    val focusManager = LocalFocusManager.current

    var userDataList by remember { mutableStateOf<List<Triple<String, Float,
Float>>>(emptyList()) }

    LaunchedEffect(key1 = true) {
        viewModel.fetchUserDataFromFirebase(
            onSuccess = { userDataList = it },
            onFailure = {  }
        )
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .clickable { focusManager.clearFocus() },
        verticalArrangement = Arrangement.Center,
```

```
    horizontalAlignment = Alignment.CenterHorizontally
) {
    TopAppBar(
        title = {
            Text(
                text = "Прогрес",
                fontSize = 20.sp,
                color = Color.White
            )
        },
        actions = {
            Button(
                onClick = {
                    viewModel.signOut {
                        navController.navigate(Screen.LoginScreen.route)
                    }
                },
                colors = ButtonDefaults.buttonColors(Color.White)
            ) {
                Text(text = "Вихід з аккаунту", color = Color.Black)
            }
        },
        backgroundColor = Color.Blue,
        modifier = Modifier.fillMaxWidth()
    )

    LazyColumn(
        modifier = Modifier.weight(1f),
        content = {
            items(userDataList) { (date, weight, bellyWidth) ->
                Column(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(horizontal = 16.dp, vertical = 8.dp)
                ) {
                    Text(text = "Дата: $date")
                    Text(text = "Вага: $weight")
                    Text(text = "Ширина поясу: $bellyWidth")
```

```
                    Divider(color = Color.Gray, thickness = 1.dp)
                }
            }
        }
    )

    BottomNavigation(
        modifier = Modifier
            .fillMaxWidth()
            .height(56.dp)
            .padding(3.dp),
        backgroundColor = Color.Blue,
        contentColor = Color.White
    ) {
        BottomNavigationItem(
            selected = true,
            onClick = { navController.navigate(Screen.Homescreen.route) },
            icon = {
                androidx.compose.material.Icon(
                    imageVector = Icons.Default.Home,
                    contentDescription = ""
                )
            },
            label = { androidx.compose.material.Text(text = "Щоденник", onTextLayout
= {}) }
        )
        BottomNavigationItem(
            selected = false,
            onClick = { navController.navigate(Screen.AddingDishcreen.route) },
            icon = {
                androidx.compose.material.Icon(
                    imageVector = Icons.Default.AccountBox,
                    contentDescription = ""
                )
            },
            label = {
                androidx.compose.material.Text(
                    text = "Додати страву",
```

```
                        onTextLayout = {})
                }
            )

            BottomNavigationItem(
                selected = false,
                onClick = { navController.navigate(Screen.Progressscreen.route) },
                icon = {
                    androidx.compose.material3.Icon(
                        imageVector = Icons.Default.Favorite,
                        contentDescription = ""
                    )
                },
                label = { androidx.compose.material.Text(text = "Прогрес", onTextLayout
= {}) }
            )

        }
    }
}
```

FirebaseViewModel.kt:

```
package com.example.caloriesapp

import androidx.lifecycle.ViewModel
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import androidx.lifecycle.viewModelScope
import co.yml.charts.common.model.Point
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.ListenerRegistration
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.firestore.toObject
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await
```

```
class FirebaseViewModel: ViewModel() {

    private val auth: FirebaseAuth = Firebase.auth
    private val fs = Firebase.firestore
    private val _currentUser = FirebaseAuth.getInstance().currentUser

    fun signOut(onSignedOut: () -> Unit) {
        auth.signOut()
        onSignedOut()
    }

    fun signInToAccount(
        email: String,
        password: String,
        onSuccess: () -> Unit,
        onError: (String) -> Unit
    ) =
        viewModelScope.launch {
            try {
                auth.createUserWithEmailAndPassword(email, password)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            onSuccess()
                        } else {
                            onError("Помилка при реєстрації:
${task.exception?.message}")
                        }
                    }
            } catch (ex: Exception) {
                onError("Помилка при реєстрації: ${ex.message}")
            }
        }

    fun logInToAccount(
        email: String,
        password: String,
        onError: (String) -> Unit,
```

```
        onLoggedIn: () -> Unit
    ) =
        viewModelScope.launch {
            try {
                auth.signInWithEmailAndPassword(email, password)
                    .addOnCompleteListener() { task ->
                        if (task.isSuccessful) {
                            onLoggedIn()
                        } else {
                            onError("Неправильно введено пошту або пароль. Повторіть
знову.")
                        }
                    }
            } catch (ex: Exception) {
                onError("Помилка: ${ex.message}")
            }
        }

    fun calculateRequiredCalories(user: User): Int {
        val weight = user.weight
        val height = user.height
        val basalMetabolicRate = calculateBasalMetabolicRate(weight, height)
        return basalMetabolicRate
    }

    private fun calculateBasalMetabolicRate(weight: Int, height: Int): Int {
        val bmr = 10 * weight + 6.25 * height - 161
        return bmr.toInt()
    }

    fun getRequiredMacronutrients(currentDate: String, callback: (Int, Int, Int, Int) -
> Unit) {
        _currentUser?.let { user ->
            val userId = user.uid
            fs.collection("users")
                .document(userId)
                .collection("data")
                .document(currentDate)
```

```
                .get()
                .addOnSuccessListener { document ->
                    val user = document.toObject(User::class.java)
                    user?.let {
                        val requiredCalories = calculateRequiredCalories(user)
                        val proteinPercentage = 0.15
                        val fatPercentage = 0.30
                        val carbohydratePercentage = 0.55

                        val proteinGrams = (requiredCalories * proteinPercentage / 4).toInt()
                        val fatGrams = (requiredCalories * fatPercentage / 9).toInt()
                        val carbohydrateGrams = (requiredCalories * carbohydratePercentage / 4).toInt()

                        callback(requiredCalories, proteinGrams, fatGrams, carbohydrateGrams)
                    }
                }
                .addOnFailureListener { e ->
                }
        }
    }


    fun setSelectedBreakfastDishesForCurrentUser(
        selectedBreakfastDishes: List<Dish>,
        onSuccess: () -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        setSelectedDishesForCurrentUser("breakfast", selectedBreakfastDishes, onSuccess, onFailure)
    }

    fun setSelectedLunchDishesForCurrentUser(
        selectedLunchDishes: List<Dish>,
        onSuccess: () -> Unit,
        onFailure: (Exception) -> Unit
```

```
    ) {
        setSelectedDishesForCurrentUser("lunch",     selectedLunchDishes,     onSuccess,
onFailure)
    }

    fun setSelectedDinnerDishesForCurrentUser(
        selectedDinnerDishes: List<Dish>,
        onSuccess: () -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        setSelectedDishesForCurrentUser("dinner",    selectedDinnerDishes,    onSuccess,
onFailure)
    }

    fun fetchUserDataFromFirebase(
        onSuccess: (List<Triple<String, Float, Float>>) -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid

        userId?.let { uid ->
            val db = FirebaseFirestore.getInstance()
            val userCollection = db.collection("users").document(uid).collection("data")

            userCollection.addSnapshotListener { snapshot, error ->
                if (error != null) {
                    onFailure(error)
                    return@addSnapshotListener
                }

                val userDataList = mutableListOf<Triple<String, Float, Float>>()
                snapshot?.documents?.forEach { document ->
                    val user = document.toObject<User>()
                    val date = user?.date ?: ""
                    val weight = user?.weight?.toFloat() ?: 0f
                    val bellyWidth = user?.bellyWidth?.toFloat() ?: 0f
                    userDataList.add(Triple(date, weight, bellyWidth))
```

```
                }
                onSuccess(userDataList)
            }
        }
    }

    fun sumForSelectedDishes(
        onSuccess: (Int, Int, Int) -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        _currentUser?.let { user ->
            val userId = user.uid
            val                                userDishesRef                                =
fs.collection("users").document(userId).collection("selectedDishes")

            userDishesRef.get().addOnSuccessListener { querySnapshot ->
                var breakfastCalories = 0
                var lunchCalories = 0
                var dinnerCalories = 0

                val  breakfastDishes  =  querySnapshot.documents.firstOrNull  {  it.id  ==
"breakfast" }
                val lunchDishes = querySnapshot.documents.firstOrNull { it.id == "lunch"
}
                val  dinnerDishes  =  querySnapshot.documents.firstOrNull  {  it.id  ==
"dinner" }

                breakfastDishes?.let { documentSnapshot ->
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    for (dishMap in dishes) {
                        val dish = mapToDish(dishMap)
                        breakfastCalories += dish.calories
                    }
                }

                lunchDishes?.let { documentSnapshot ->
```

```
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    for (dishMap in dishes) {
                        val dish = mapToDish(dishMap)
                        lunchCalories += dish.calories
                    }
                }

                dinnerDishes?.let { documentSnapshot ->
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    for (dishMap in dishes) {
                        val dish = mapToDish(dishMap)
                        dinnerCalories += dish.calories
                    }
                }

                onSuccess(breakfastCalories, lunchCalories, dinnerCalories)
            }.addOnFailureListener { exception ->
                onFailure(exception)
            }
        }
    }

    fun setSelectedDishesForCurrentUser(
        mealType: String,
        selectedDishes: List<Dish>,
        onSuccess: () -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        val userId = _currentUser?.uid ?: return
        val                                  userRef                                      =
fs.collection("users").document(userId).collection("selectedDishes")

        // Видаляємо існуючі страви поточного типу перед додаванням нових
        userRef.document(mealType).delete().addOnSuccessListener {
            // Додаємо нові страви
```

```
            userRef.document(mealType).set(mapOf("dishes"   to   selectedDishes.map   {
dishToMap(it) }))
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        onSuccess()
                    } else {
                        onFailure(task.exception ?: Exception("Unknown error"))
                    }
                }
        }.addOnFailureListener { exception ->
            onFailure(exception)
        }
    }


    // Метод для перетворення даних з HashMap в об'єкт Dish
    private fun mapToDish(map: HashMap<String, Any>): Dish {
        val name = map["name"] as? String ?: ""
        val calories = (map["calories"] as? Long)?.toInt() ?: 0
        val id = (map["id"] as? String ?: "")
        return Dish(name = name, calories = calories, id = id)
    }

    private fun dishToMap(dish: Dish): HashMap<String, Any?> {
        val map = hashMapOf<String, Any?>(
            "id" to dish.id,
            "name" to dish.name,
            "calories" to dish.calories
        )
        return map
    }

    fun startListeningForSelectedDishes(
        onSuccess: (List<Dish>, List<Dish>, List<Dish>) -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        _currentUser?.let { user ->
            val userId = user.uid
```

```
            val                                userDishesRef                                =
fs.collection("users").document(userId).collection("selectedDishes")

            userDishesRef.get().addOnSuccessListener { querySnapshot ->
                val  breakfastDishes  =  querySnapshot.documents.firstOrNull  {  it.id  ==
"breakfast" }
                val lunchDishes = querySnapshot.documents.firstOrNull { it.id == "lunch"
}
                val  dinnerDishes  =  querySnapshot.documents.firstOrNull  {  it.id  ==
"dinner" }

                val breakfastDishList = breakfastDishes?.let { documentSnapshot ->
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    dishes.map { mapToDish(it) }
                } ?: emptyList()

                val lunchDishList = lunchDishes?.let { documentSnapshot ->
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    dishes.map { mapToDish(it) }
                } ?: emptyList()

                val dinnerDishList = dinnerDishes?.let { documentSnapshot ->
                    val dishes = documentSnapshot.get("dishes") as? List<HashMap<String,
Any>> ?: emptyList()
                    dishes.map { mapToDish(it) }
                } ?: emptyList()

                onSuccess(breakfastDishList, lunchDishList, dinnerDishList)
            }.addOnFailureListener { exception ->
                onFailure(exception)
            }
        } ?: onFailure(Exception("Current user is null"))
    }

    fun deleteDishFromSelectedDishesList(
        documentId: String,
```

```
        mealType: String,
        onSuccess: () -> Unit,
        onFailure: (Exception) -> Unit
    ) {
        val userId = _currentUser?.uid ?: return
        val userDishesRef = fs.collection("users").document(userId).collection("selectedDishes").document(mealType)

        fs.runTransaction { transaction ->
            val snapshot = transaction.get(userDishesRef)
            val selectedDishes = snapshot.get("dishes") as? List<Map<String, Any>> ?: emptyList()
            val updatedSelectedDishes = selectedDishes.filterNot { it["id"] == documentId }
            transaction.update(userDishesRef, "dishes", updatedSelectedDishes)
        }.addOnSuccessListener {
            onSuccess()
        }.addOnFailureListener { exception ->
            onFailure(exception)
        }
    }

    fun searchDishes(query: String, callback: (List<Dish>) -> Unit) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid
        val fs = Firebase.firestore

        userId?.let { uid ->
            fs.collection("users").document(uid).collection("dishes")
                .whereEqualTo("name", query)
                .get()
                .addOnSuccessListener { querySnapshot ->
                    val dishes = mutableListOf<Dish>()
                    for (document in querySnapshot) {
                        val dish = document.toObject<Dish>()
                        dishes.add(dish)
                    }
                    callback(dishes)
```

```
                }
                .addOnFailureListener { exception ->
                }
        }
    }

    fun deleteDishByName(dishName: String, onSuccess: () -> Unit, onFailure: (Exception)
-> Unit) {
        val currentUser = FirebaseAuth.getInstance().currentUser
        val userId = currentUser?.uid
        val fs = Firebase.firestore

        userId?.let { uid ->
            // Пошук страв за ім'ям в колекції "dishes"
            fs.collection("users").document(uid).collection("dishes")
                .whereEqualTo("name", dishName)
                .get()
                .addOnSuccessListener { querySnapshot ->
                    // Перевірка на наявність знайденої страви
                    if (!querySnapshot.isEmpty) {
                        // Отримання страви перед його видаленням
                        val document = querySnapshot.documents[0]
                        document.reference.delete()
                            .addOnSuccessListener {
                                onSuccess()
                            }
                            .addOnFailureListener { exception ->
                                onFailure(exception)
                            }
                    } else {
                        onFailure(Exception("Error"))
                    }
                }
                .addOnFailureListener { exception ->
                    onFailure(exception)
                }
        }
    }
```