

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО
МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПІДТРИМКИ
ЗДОРОВОГО СПОСОБУ ЖИТТЯ
SOFTWARE IMPLEMENTATION OF A
CROSS-PLATFORM MOBILE APPLICATION FOR
HEALTHY LIFESTYLE MANAGEMENT

Виконав(ла): студент(ка) 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЛЕЩИНСЬКА А.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

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

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЛЕЩИНСЬКОЇ Анни Русланівни

(ПРІЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Програмна реалізація кросплатформного застосунку для підтримки здорового способу життя. Software Implementation of a Cross-Platform Mobile Application for Healthy Lifestyle Management
 керівник проєкту (роботи) д-р філософ., доцент ЛЕОЩЕНКО Сергій Дмитрович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
 затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
- Строк подання студентом проєкту (роботи) 10 червня 2026 року
- Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

Підпис Анна ЛЕЩИНСЬКА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Підпис Сергій ЛЕОЩЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 88 с., 2 табл., 18 рис., 3 дод., 7 джерел.

ЗДОРОВИЙ СПОСІБ ЖИТТЯ, КБЖВ, REACT NATIVE, NODE.JS, FATSECRET API, ТРЕКІНГ ХАРЧУВАННЯ, ТРЕКІНГ АКТИВНОСТІ.

Об'єкт дослідження – процес програмної реалізації застосунку для моніторингу харчування та фізичної активності.

Предмет дослідження – програмні засоби розробки кросплатформних застосунків (React Native, Node.js, Express).

Мета роботи: підвищення наочності моніторингу харчування та фізичної активності, за рахунок розробки застосунку для автоматизації контролю.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, компонентний підхід, фреймворк React Native, платформа Node.js та фреймворк Express, СКБД PostgreSQL, Prisma ORM, FatSecret API, мова програмування JavaScript, ПК під управлінням ОС Windows 10, середовище розробки VS Code.

Результати. Розроблено мобільний застосунок, для розрахунку КБЖВ, введення щоденника харчування та активності, а також відстежування прогресу зміни ваги. Реалізовано клієнт-серверну архітектуру з надійним збереженням даних та актуальною базою продуктів харчування.

Висновки. Реалізовано кросплатформенний мобільний застосунок, який автоматизує процес контролю за станом здоров'я, спрощує підрахунок нутрієнтів та допомагає користувачам досягати поставлених цілей щодо корекції маси тіла.

Галузь використання – особисте використання для самоконтролю харчування, фізичної активності та показників здоров'я.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 88 pages, 2 tables, 18 figures, 3 appendices, 7 sources.

HEALTHY LIFESTYLE, CALORIES AND MACRONUTRIENTS, REACT NATIVE, NODE.JS, FATSECRET API, NUTRITION TRACKING, ACTIVITY TRACKING.

Object of research: the process of software implementation of an application for monitoring nutrition and physical activity.

Subject of research: software development tools for cross-platform applications (React Native, Node.js, Express).

The aim of the work: improving the visibility of nutrition and physical activity monitoring through the development of an application for control automation.

Materials, methods, and technical tools: object-oriented programming, component-based approach, React Native framework, Node.js platform and Express framework, PostgreSQL DBMS, Prisma ORM, FatSecret API, JavaScript programming language, PC running Windows 10, VS Code development environment.

Results. A mobile application was developed for calculating calorie and macronutrient intake, maintaining a nutrition and activity diary, and tracking weight change progress. A client-server architecture with reliable data storage and an up-to-date food database was implemented.

Conclusions. A cross-platform mobile application has been implemented that automates the process of health monitoring, simplifies nutrient counting, and helps users achieve their goals regarding body weight correction.

Field of application – personal use for self-monitoring of nutrition, physical activity, and health indicators.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Аналіз складових здорового способу життя	10
1.2 Аналіз існуючих програмних рішень	11
1.3 Постановка задачі	13
1.4 Межі системи та розробки	14
1.5 Функціональні вимоги	15
1.6 Висновки з розділу 1	16
2 Матеріали і методи	17
2.1 Вибір технологій розробки клієнтської частини	17
2.2 Вибір технологій розробки серверної частини	19
2.3 Проєктування бази даних та вибір СКБД	19
2.4 Вибір API	22
2.5 Висновки з розділу 2	23
3 Опис програми	24
3.1 Архітектура застосунку	24
3.2 Визначення сценаріїв взаємодії користувача із застосунком	25
3.3 Математична модель розрахунку КБЖВ	28
3.4 Інтеграція із зовнішнім сервісом FatSecret Platform API	30
3.5 Реалізація локалізації	31
3.6 Візуалізація історії змін ваги	34
3.7 Висновок з розділу 3	34
4 Експлуатація, тестування та експериментальне дослідження програми	35
4.1 Роль застосунку у підтримці здорового способу життя	35
4.2 Користувацький інтерфейс(UI)	36
4.3 Умови функціонування застосунку	37

4.4 Дослідження роботи застосунку.....	38
4.5 Висновок з розділу 4	40
Висновки.....	42
Перелік джерел посилання	43
Додаток А Технічне завдання.....	44
Додаток Б Текст програми.....	50
Додаток В Слайди презентації	80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID – Atomicity, Consistency, Isolation, Durability;

API – Application Programming Interface;

BMR – Basal Metabolic Rate;

JWT – JSON Web Token;

ORM – Object-Relational Mapping;

TDEE – Total Daily Energy Expenditure;

UI – User Interface;

UX – User Experience;

КБЖВ – калорії, білки, жири, вуглеводи;

ПК – персональний комп'ютер;

СКБД – система керування базами даних.

ВСТУП

У сучасному світі здоровий спосіб життя стає усе більш актуальним. Для підтримки належного фізичного стану важливими є не лише правильне харчування і фізична активність, а й системний контроль таких показників, як баланс нутрієнтів, рівень активності та динаміки змін стану організму.

Проте в умовах швидкого темпу життя самостійне відстеження цих параметрів стає складним завданням, що займає багато часу та часто призводить до зниження мотивації користувачів. Такі методи обліку як паперові щоденники або нотатки на телефоні не є зручними та не забезпечують необхідної точності розрахунків. Тому виникає потреба у розробці програмного рішення, що дозволяють зручно аналізувати дані про харчування та фізичну активність.

Розробка мобільного застосунку «Soma» забезпечує спрощення процесу моніторингу харчових звичок та фізичної активності, забезпечуючи користувача інструментами для точного розрахунку індивідуальних норм КБЖВ та моніторингу прогресу в режимі реального часу.

Метою дипломної кваліфікаційної роботи є розробка застосунку для автоматизації контролю харчування та фізичної активності.

Унікальність мобільного застосунку «Soma» у створенні системи для щоденного моніторингу. На відміну від традиційних підходів, що потребують використання декількох різних сервісів, «Soma» забезпечує синхронізацію даних у реальному часі. Система автоматично коригує енергетичний баланс користувача у єдиному інтерфейсі. Застосунок сфокусований виключно на інструментах контролю (вага, їжа, активність), уникаючи перевантаження інтерфейсу. Це дозволяє зосередитися на головному – швидкому внесенні та аналізу даних та відстеженні прогресу. Користувач отримує однаковий рівень зручності та функціональності незалежно від моделі смартфона (Android чи iOS).

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Обґрунтування розробки застосунку «Soma» потребує дослідження специфіки контролю харчування та фізичної активності. У цьому розділі буде розглянуто основні складові здорового способу життя, зокрема харчування, фізичну активність та контроль фізичних показників організму, що є ключовими елементами для формування ефективної системи моніторингу. Також буде проведено аналіз існуючих програмних рішень, що дозволить сформулювати вимоги до розроблюваного застосунку. На основі проведеного дослідження буде сформульовано задачі роботи, визначено межі системи.

1.1 Аналіз складових здорового способу життя

Можна виділити кілька ключових складових, які безпосередньо впливають на фізичний стан людини, а саме: харчування, фізична активність, загальний стан здоров'я.

Харчування визначає енергетичний баланс організму. Важливою є не лише калорійність раціону, а й співвідношення основних нутрієнтів: білків, жирів, вуглеводів. Контроль цих параметрів дозволяє досягати таких цілей, як підтримка ваги, схуднення, набір ваги.

Фізична активність є ще однією важливою складовою. Вона включає різні види навантажень, такі як кардіо, силові тренування та повсякденну активність. Регулярний облік фізичної активності дає змогу оцінювати витрати енергії та ефективність тренувань.

Загальний стан здоров'я характеризується сукупністю фізичних показників, серед яких маса тіла та загальне самопочуття. Відстеження їх змін дає змогу оцінювати вплив харчування та фізичної активності на загальний фізичний стан людини.

Таким чином, харчування, фізична активність та стан здоров'я потребують комплексного підходу до їх обліку та аналізу.

1.2 Аналіз існуючих програмних рішень

На сьогодні існує велика кількість програмних рішень, що дозволяють здійснювати облік харчування та фізичної активності.

MyFitnessPal – один з найпотужніших застосунків, який дозволяє вести облік калорійності раціону, має велику базу продуктів. Недоліком є велика кількість функцій доступ за підпискою, інтерфейс перевантажений рекламою, а локалізація бази продуктів на український ринок часто є некоректною. Інтерфейс застосунку продемонстровано на рисунку 1.1.

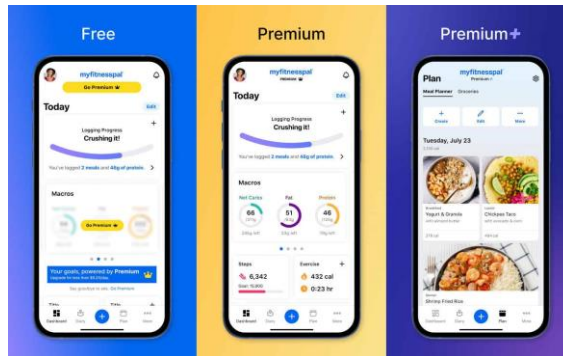


Рисунок 1.1 – Робота застосунку «MyFitnessPal»[1]

FatSecret – популярний лічильник калорій, який має безкоштовний доступ до основних інструментів. Недоліком є відсутній гнучкий планувальник тренувань. Інтерфейс застосунку продемонстровано на рисунку 1.2.

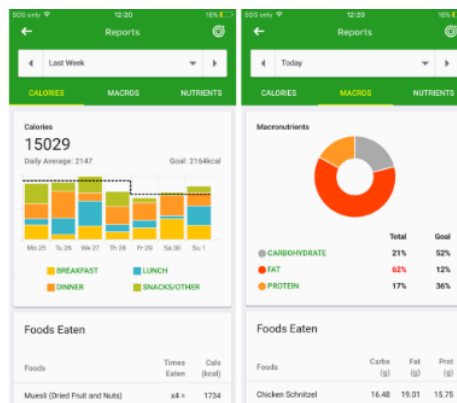


Рисунок 1.2 – Робота застосунку «FatSecret»[2]

Samsung Health – це застосунок від компанії Samsung, який працює як повноцінний хаб для управління здоров'ям. Переваги: автоматичне відстеження кроків, сну, серцевого ритму. Незважаючи на його багатогранність, Samsung Health має низку недоліків: перевантаженість інтерфейсу через спробу об'єднати занадто багато функцій, повний функціонал (наприклад, вимірювання складу тіла або ЕКГ) доступний лише власникам годинників Samsung Galaxy Watch. Інтерфейс застосунку продемонстровано на рисунку 1.3.

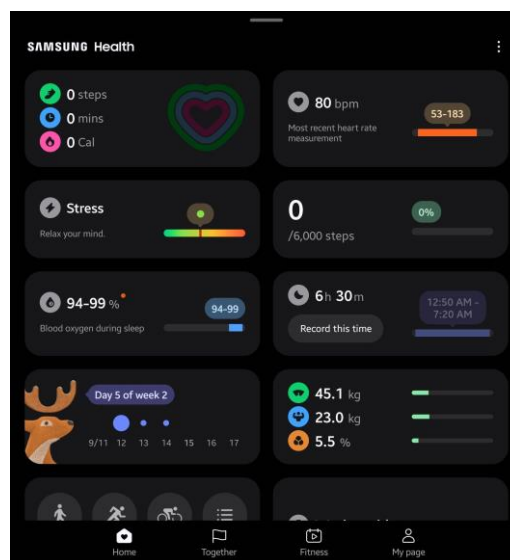


Рисунок 1.3 – Робота застосунку «Samsung Health»[3]

Порівняння аналогів за критеріями наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих програмних рішень

Критерій	MyFitnessPal	FatSecret	Samsung Health
Призначення	Облік харчування та фітнес-трекінг	Облік харчування	Облік харчування, фітнес-трекінг, відстеження показників здоров'я
База продуктів	Дуже велика	Велика	Середня

Продовження таблиці 1.1

Критерій	MyFitnessPal	FatSecret	Samsung Health
Доступ до статистики БЖВ	Лише за підпискою	Є	Є
Облік фізичної активності	Обмежений(повний доступ за підпискою)	Обмежений	Є(автоматичний)
Інтерфейс	Перевантажений	Простий	Перевантажений
Реклама в інтерфейсі	Багато	Мінімальна	Відсутня
Вартість	Платний функціонал	Безкоштовний	Безкоштовний
Залежність від пристрою	Немає	Немає	Є для повного функціоналу
Локалізація	Обмежена	Обмежена	Повна
Кросплатформність	Присутня	Присутня	Обмежена («обрізаний» функціонал для iOS)

1.3 Постановка задачі

Метою даного дипломної кваліфікаційної роботи є розробка мобільного застосунку для підтримки здорового способу життя, який забезпечує користувача інструментами для контролю харчування та фізичної активності.

Для досягнення поставленої мети реалізовано наступні задачі:

а) розробка системи автентифікації та заповнення профілю користувача:

1) реалізація автентифікації із використанням безпечних протоколів (JWT-токени);

2) реалізація анкетування користувача для збору антропометричних даних(вага, зріст, вік, стать) та визначення рівня фізичної активності й основної мети(схуднення, підтримка ваги, набір маси);

б) реалізація математичних обчислень:

1) алгоритм автоматичного розрахунку базового метаболізму (BMR) та загальних добових витрат енергії (TDEE) за формулою Міффіна-Сан Жеора;

в) створення модуля обліку харчування:

1) інтегрування зовнішнього API (FatSecret) для доступу до бази харчових продуктів;

2) розробка механізму локалізації результатів пошуку;

3) додавання власних продуктів та страв до локальної бази даних;

г) створення модуля обліку фізичних активностей.

1.4 Межі системи та розробки

Для розробки клієнтської частини було обрано фреймворк React Native. Він забезпечує взаємодію користувача з системою та візуалізацію даних.

Серверна логіка реалізується за допомогою Node.js та фреймворку Express. Це забезпечує асинхронну обробку запитів, що є критично важливим при взаємодії із зовнішніми сервісами (FatSecret API) та внутрішньою базою даних.

Для проектування реляційної структури обрано СКБД PostgreSQL. Також використано ORM-систему Prisma для забезпечення цілісності даних та швидкої маніпуляції складними об'єктами, такими як щоденні логи харчування/тренувань, антропометричні показники користувача.

1.5 Функціональні вимоги

На основі проведеного аналізу предметної області було визначено функціональні вимоги, які має забезпечувати застосунок:

- система повинна дозволяти користувачеві створити обліковий запис за допомогою електронної пошти та пароля;
- після створення облікового запису система має вимагати введення антропометричних даних (стать, вага, зріст, дата народження), мети (схуднення, підтримка ваги, набір ваги), рівня активності;
- можливість редагування персональних даних, зміни мови інтерфейсу(англійська/українська) та теми(світла/темна);
- система повинна автоматично розраховувати показники BMR (базовий метаболізм) та TDEE (добові витрати енергії) на основі введених даних користувача.
- визначення значень білків, жирів та вуглеводів (КБЖВ) відповідно до обраної мети користувача;
- перерахунок норм при зміні поточної ваги користувача;
- можливість пошуку харчових продуктів через зовнішнє API (FatSecret);
- система повинна автоматично перекладати назви продуктів, опис, одиниці вимірювання;
- можливість додавання обраних продуктів до щоденника із зазначенням ваги порції та вибору типу прийому;
- розрахунок та відображення кількості спожитих калорій та нутрієнтів у відносно добової норми;
- можливість додавання обраних активностей до щоденника із зазначенням їх тривалості;
- розрахунок та відображення спалених калорій під час виконання активностей в залежності від їх типу та тривалості;

- збереження та відображення даних(спожиті, спалені калорії, вага) для аналізу прогресу;
- інформування користувача про некоректно введені дані або відсутність результатів пошуку.

1.6 Висновки з розділу 1

У межах першого розділу було проведено аналіз предметної області та обґрунтовано доцільність розробки нового застосунку.

Виявлено, що для підтримки здоров'я необхідний повноцінний облік нутрієнтів (КБЖВ), антропологічних показників та фізичної активності. Аналіз аналогів показав, що є проблеми з недоступністю функціоналу, локалізацією для України, перевантаженість інтерфейсу та прив'язка до інших гаджетів. Визначено межі системи: для реалізації обрано стек React Native (Frontend), Node.js/Express (Backend) та PostgreSQL/Prisma (Database), що забезпечує кросплатформність та надійність системи. Визначено необхідність реалізації автоматичних розрахунків за формулами Міффіна-Сан Жеора, інтеграції з FatSecret API.

2 МАТЕРІАЛИ І МЕТОДИ

У цьому розділі буде наведено обґрунтування вибору технологічного стеку та опис методів, що були використані для розробки мобільного застосунку «Soma». Якість та продуктивність сучасного програмного продукту безпосередньо залежать від правильно обраних інструментів розробки, архітектурних патернів та методів організації даних.

Основним завданням етапу проектування є вибір засобів, які забезпечать кросплатформність застосунку, надійність збереження персональних даних користувачів та високу швидкість обробки запитів. Для розв'язання поставленої задачі було застосовано аналітичний метод порівняння сучасних фреймворків, а також методи об'єктно-орієнтованого проектування для створення структури бази даних та серверної логіки.

Нижче наведено детальний опис обраних компонентів системи, починаючи з платформи для розробки клієнтської частини, що є фундаментальним кроком у створенні мобільного інтерфейсу.

2.1 Вибір технологій розробки клієнтської частини

Вибір технології для розробки клієнтської частини застосунку є критично важливим етапом, оскільки він визначає якість користувацького досвіду. Для реалізації проєкту було розглянуто три основні підходи: нативна розробка (Swift для iOS, Kotlin для Android), кросплатформний фреймворк Flutter та бібліотека React Native.

Проведено порівняльний аналіз сучасних мобільних платформ. З аналізу випливає висновок, що найкращим вибором для розробки є React Native. Використання єдиної кодової бази для iOS та Android дозволяє зекономити час. Це є вирішальним фактором для розробки проєкту для дипломування, бо часовий ресурс обмежений. На відміну від Flutter, який «малює» власний інтерфейс, React Native звертається до нативних

компонентів операційної системи, що забезпечує відповідність стандартам. Оскільки серверна частина застосунку розробляється на Node.js, використання JavaScript на клієнті дозволяє використовувати спільні типи даних та бібліотеки. Результати аналізу наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння сучасних мобільних платформ

Характеристика	React Native	Swift/Kotlin	Flutter
Мова програмування	JavaScript / TypeScript	Swift (iOS), Kotlin (Android)	Dart
Швидкість розробки	Висока (завдяки повторному використанню коду)	Нижча (два окремі проєкти)	Висока
Складність входу	Низька–середня (для JS-розробників)	Висока (потрібно знати дві платформи)	Середня (Dart новий для більшості)
Підтримка та екосистема	Велика (JavaScript екосистема)	Стабільна, але розділена	Швидко зростає
UI/UX	Використовує нативні компоненти	Повністю нативний інтерфейс	Повністю кастомний UI
Кросплатформність	Кросплатформна (один код для iOS та Android)	Роздільна розробка для кожної платформи	Кросплатформна (єдиний код + власний UI рушій)

2.2 Вибір технологій розробки серверної частини

Для реалізації серверної частини застосунку було обрано Node.js та фреймворк Express. Що дозволило побудувати бекенд, який здатний ефективно обробляти велику кількість мережових запитів.

Ключовим фактором на користь Node.js стала його асинхронна модель введення-виведення. Яка дозволяє не зависати серверу в очікуванні відповіді від FatSecret API, а продовжувати свою роботу.

Використання фреймворку Express дозволило спростити маршрутизацію та обробку HTTP-запитів. Він є досить лаконічним, що підходить для архітектури, орієнтованої на швидку обробку даних.

Взаємодія між мобільним клієнтом та сервером побудована за архітектурним стилем REST API. Клієнтська частина відповідає за інтерфейс, а серверна – за бізнес-логіку та роботу з даними. Кожен запит містить усю необхідну інформацію для його обробки (включаючи JWT-токени для автентифікації), що спрощує масштабування системи та тестування окремих ендпоінтів. Обмін даними відбувається у форматі JSON, який є легким, зрозумілим та ідеально сумісним із JavaScript, що використовується як на фронтенді, так і на бекенді.

2.3 Проєктування бази даних та вибір СКБД

Функціонування застосунку потребує надійного збереження великої кількості інформації: профілів користувачів, налаштувань, записів прийому їжі, активностей, довідкової інформації(рецепти, відео тренувань) і тд. Для реалізації цієї задачі було обрано реляційну систему керування базами даних(СКБД) PostgreSQL. Такий вибір обґрунтований наступними перевагами:

- дані застосунку мають чіткі логічні зв'язки;

– підтримка атомарності узгодженості, ізолюваності гарантує точність даних. Це критично важливо при оновленні показників ваги або розрахунку добової норми калорій, де будь-яка помилка може призвести до некоректних результатів;

– PostgreSQL є потужним інструментом з відкритим кодом, який дозволяє легко розширювати структуру бази даних при додаванні нових функцій у майбутньому.

Для взаємодії серверної частини з базою даних використано Prisma ORM. У межах проєкту Prisma виконує роль проміжного шару, який забезпечує:

– уникнення помилок ще на етапі написання коду завдяки генерації клієнта на основі схеми даних;

– зручність маніпуляцій даними. Замість написання складних SQL-запитів вручну, Prisma надає зрозумілий API для роботи з об'єктами, що робить код бекенду чистішим та легшим для підтримки;

– автоматизацію міграцій, що дозволяє синхронізувати зміни в коді зі структурою БД без ризику втрати існуючої інформації.

Для візуалізації структури даних було розроблено ER-діаграму, яка відображає архітектуру бази даних застосунку. ER-діаграму продемонстровано на рисунках 2.1-2.3.

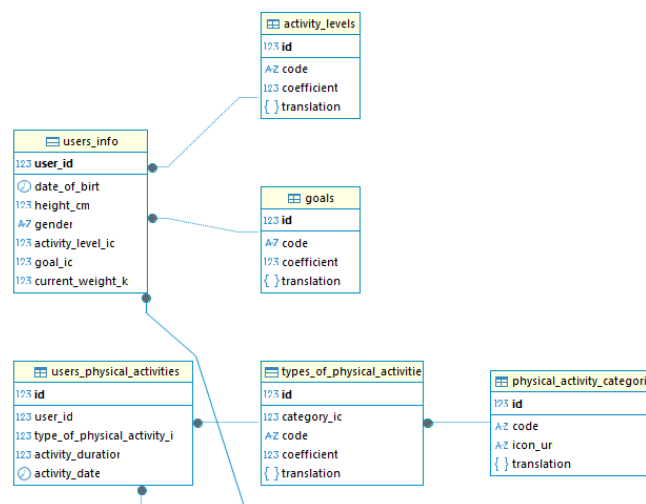


Рисунок 2.1 – Верхня частина ER-діаграми

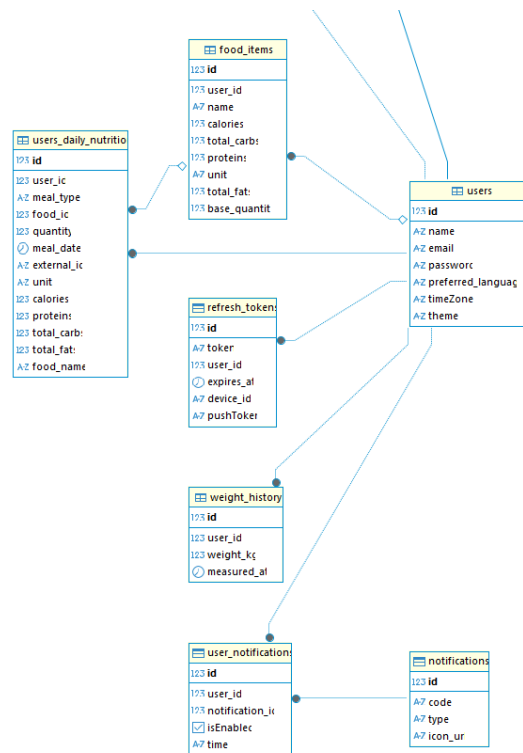


Рисунок 2.2 – Середня частина ER-діаграми

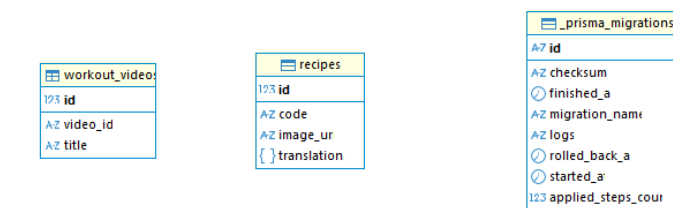


Рисунок 2.3 – Нижня частина ER-діаграми

Структура побудована за реляційною моделлю та включає наступні сутності:

- **users**: таблиця облікових записів. Зберігає ім'я, email, хешований пароль, тему та налаштування локалізації (*preferred_language*, *timeZone*);

- **users_info**: детальний профіль користувача. Містить антропометричні дані (зріст, поточна вага, стать, дата народження) та зв'язки з коефіцієнтами активності й цілями;

- **activity_levels** та **goals**: довідникові таблиці. Вони містять коефіцієнти для формул розрахунку калорій та об'єкт *translation*, де представлені локалізовані дані;

- `users_daily_nutrition`: журнал харчування. Фіксує тип прийому їжі (`meal_type`), кількість (`quantity`), нутрієнти та посилання на зовнішній `id` продукту/страви (`external_id`);
- `food_items`: локальна база продуктів, які користувач додає самотійно;
- `users_physical_activities`: журнал активності. Зберігає тривалість тренування (`activity_duration`) та дату;
- `types_of_physical_activities` та `physical_activity_categories`: довідники активностей. Містять дані про типи активностей та їх категорії, коефіцієнти інтенсивності та іконки для візуалізації;
- `weight_history`: таблиця для відстеження прогресу. Фіксує зміни ваги користувача з прив'язкою до дати вимірювання;
- `refresh_tokens`: забезпечує безпеку сесій. Зберігає токени для оновлення доступу, прив'язку до пристрою (`device_id`) та push-токени для сповіщень;
- `user_notifications` та `notifications`: дозволяє налаштовувати типи сповіщень (наприклад, нагадування про прийом їжі) та зберігати їхній статус (увімкнено/вимкнено);
- `recipes`: довідникова таблиця для збереження рецептів, яка містить об'єкт `translation`, де представлені локалізовані дані;
- `workout_videos`: зберігає посилання на відео тренувань.

2.4 Вибір API

В сучасній мобільній розробці нерідко використовують інтеграцію з існуючими хмарними сервісами для розширення функціоналу. У проєкті «Soma» використано API для роботи з продуктами та стравами – «FatSecret API».

FatSecret API надає дані про харчову цінність продуктів. Сервіс надає доступ до однієї з найбільших у світі верифікованих баз продуктів харчування,

що включає як міжнародні бренди, так і локальні продукти, представлені на ринку України. API дозволяє отримувати вичерпну інформацію про кожен продукт: калорійність, вміст білків, жирів (включаючи насичені), вуглеводів, клітковини та мікроелементів. Дане API використовується провідними світовими Wellness-застосунками, що гарантує стабільність роботи та високу точність наданої інформації. Використання REST-запитів до сервісу дозволяє реалізувати миттєвий пошук продуктів за назвою або категорією безпосередньо в інтерфейсі «Soma».

2.5 Висновки з розділу 2

Шляхом аналізу порівняння встановлено, що фреймворк React Native є найкращим рішенням для даного проєкту. Він забезпечує високу швидкість розробки завдяки спільній кодовій базі для iOS та Android, гарантує нативну продуктивність інтерфейсу та дозволяє використовувати спільну екосистему JavaScript із серверною частиною.

Для створення бекенду обрано середовище Node.js та фреймворк Express. Застосування асинхронної моделі обробки запитів та архітектурного стилю REST API дозволяє системі ефективно взаємодіяти із зовнішніми сервісами та мобільним клієнтом без блокування потоків виконання.

Для збереження даних обрано СКБД PostgreSQL, яка відповідає стандартам ACID та забезпечує цілісність інформації. Використання Prisma ORM дозволило автоматизувати роботу з базою даних, підвищити безпеку запитів та спростити підтримку коду.

Вибір FatSecret API забезпечив застосунку доступ до масштабної бази продуктів та дозволив реалізувати функцію миттєвого пошуку з високою точністю даних.

Таким чином, сформований технологічний стек та розроблена архітектура створюють надійний фундамент для роботи застосунку.

3 ОПИС ПРОГРАМИ

Даний розділ присвячений практичній частині розробки застосунку «Soma». У ньому розглядаються архітектурні рішення, логіка взаємодії компонентів системи та детальний опис функціональних модулів, що були реалізовані.

Програмна реалізація базується на сучасних інструментах (React Native, Node.js), що дозволило створити кросплатформне рішення з високим рівнем продуктивності та безпеки даних.

3.1 Архітектура застосунку

Архітектура застосунку базується на клієнт-серверній структурі, що забезпечує масштабованість та зручність підтримки коду. Система розділена на два модулі: backend (серверна логіка) та mobile (клієнтська частина). Взаємодія між клієнтом та сервером здійснюється через REST API. Клієнт відправляє запити у форматі JSON, отримуючи структуровані відповіді для відображення актуальної інформації. Архітектура застосунку продемонстрована на рисунку 3.1.

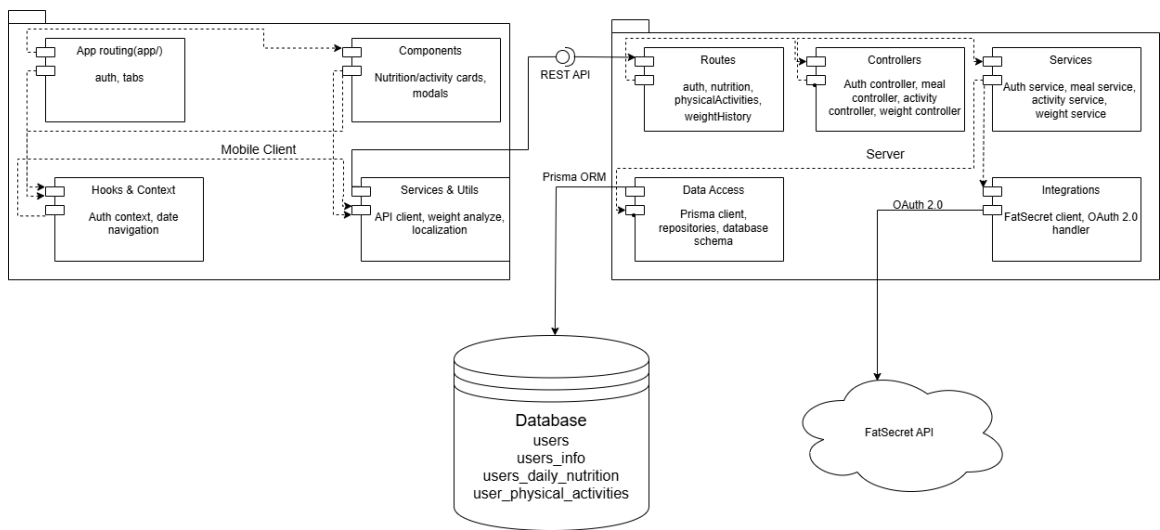


Рисунок 3.1 – Архітектура застосунку

Клієнтська частина розроблена на React Native з використанням файлової маршрутизації (Expo Router). Серверна частина розроблена на Node.js та Express, також використовується ORM Prisma для взаємодії з базою даних. Цей рівень відповідає за бізнес-логіку, безпеку та зв'язок із зовнішніми сервісами. Архітектура сервера побудована за принципом розподілу відповідальності.

3.2 Визначення сценаріїв взаємодії користувача із застосунком

Проектування інтерфейсу та внутрішньої логіки базувалося на аналізі поведінки цільового користувача. Для забезпечення зручності було виділено основні сценарії використання системи: реєстрація та анкетування, облік харчування та пошук продуктів, облік фізичних активностей, аналіз прогресу та редагування параметрів, взаємодія з рецептами та відеоматеріалом тренувань.

Реєстрація та анкетування важливий етап, від якого залежить точність усіх подальших розрахунків. Користувач після реєстрації проходить опитування, де вводить свої антропометричні дані (вага, зріст, вік, стать) та обирає свій рівень фізичної активності та ціль. Ці дані передаються на сервер для розрахунку норм КБЖВ. Діаграма цього сценарію представлена на рисунку 3.2.

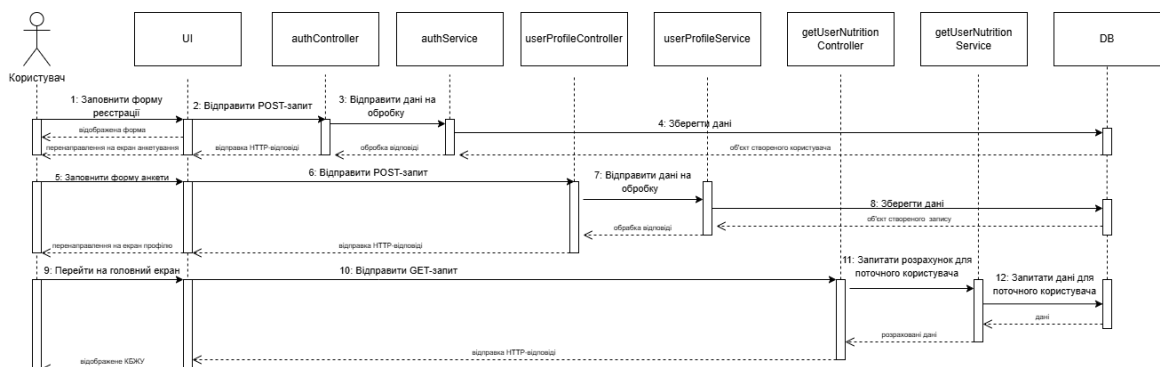


Рисунок 3.2 – Діаграма послідовності реєстрації та анкетування

Облік харчування та пошук продуктів – це найбільш складний сценарій, бо він передбачає взаємодію із зовнішнім API та можливу подальшу локалізацію. Діаграма цього сценарію представлена на рисунку 3.3.

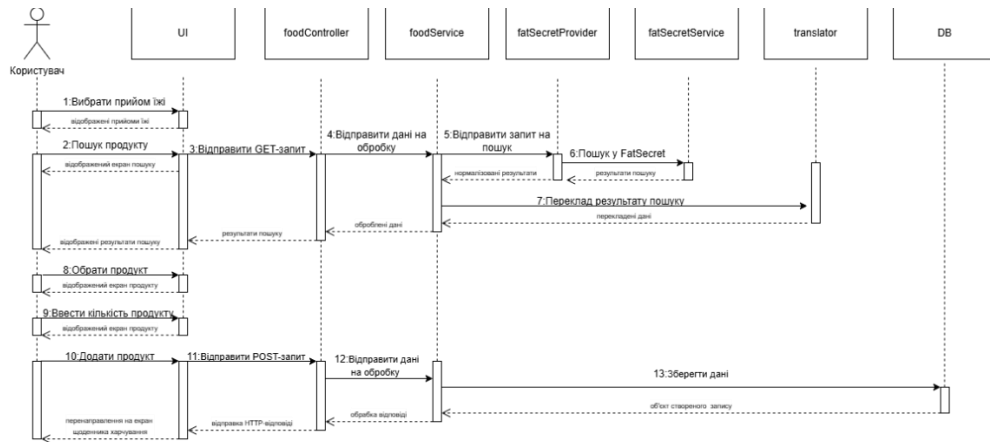


Рисунок 3.3 – Діаграма послідовності обліку харчування

Облік фізичних активностей передбачає вибір вправ із локальної бази даних та введення їх тривалості. Сценарій не потребує роботи з API, що спрощує його логіку. Діаграма цього сценарію представлена на рисунку 3.4.

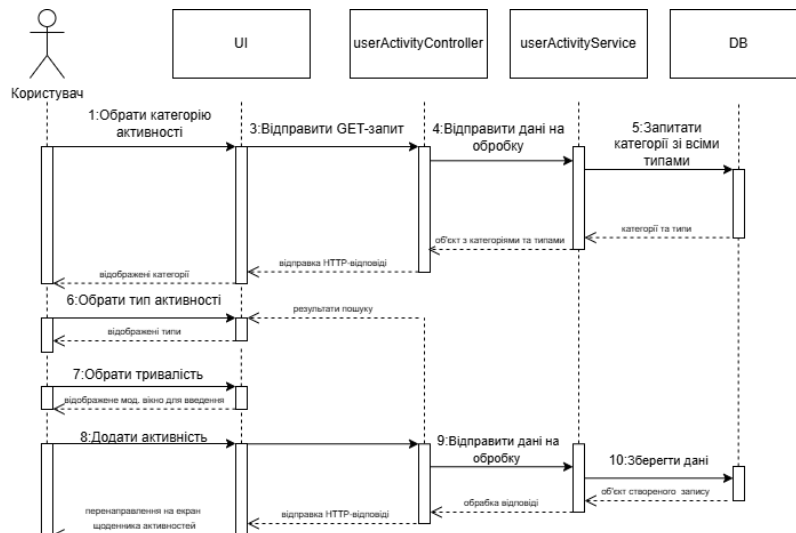


Рисунок 3.4 – Діаграма послідовності обліку фізичних активностей

Редагування параметрів забезпечує актуальність розрахунків. При зміні ваги, рівня активності чи мети сервер автоматично перераховує норми

КБЖВ. Нові дані про вагу фіксуються для побудови графіків прогресу. Діаграма цього сценарію представлена на рисунку 3.5.

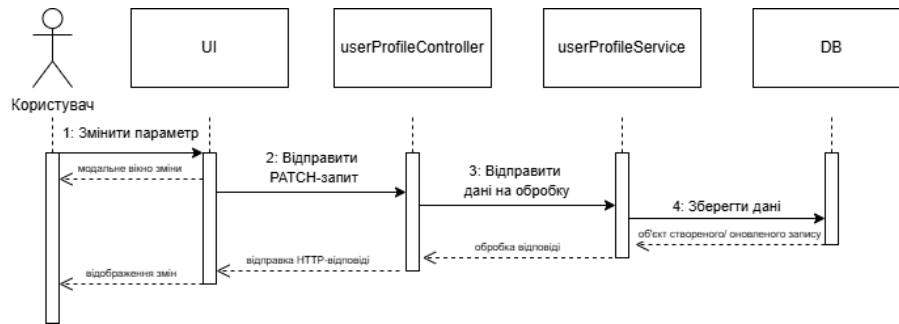


Рисунок 3.5 – Діаграма послідовності редагування параметрів

Взаємодія з рецептами та відеоматеріалом тренувань забезпечує мотиваційну складову застосунку «Soma». Користувач отримує доступ до бази рецептів з підтримкою локалізації та відео тренувань. Для оптимізації пам'яті в базі даних зберігаються лише посилання та метадані. Використання принципу «лінивого завантаження» (lazy loading) забезпечує високу швидкість інтерфейсу навіть при слабкому з'єднанні. Діаграма цього сценарію представлена на рисунку 3.6.

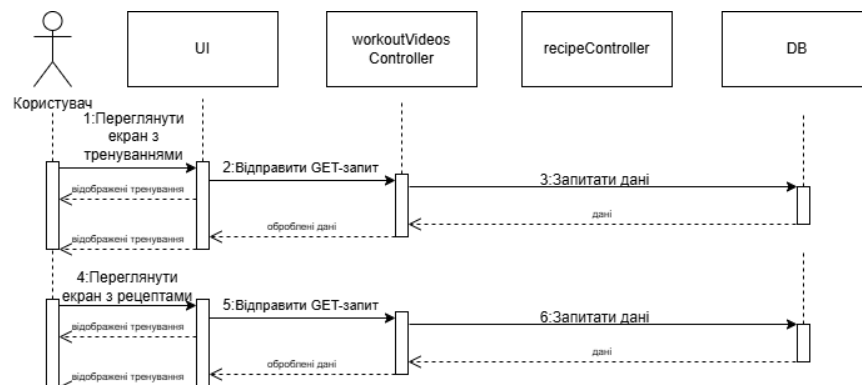


Рисунок 3.6 – Діаграма послідовності перегляду рецептів та тренувань

3.3 Математична модель розрахунку КБЖВ

Функціонування системи «Soma» базується на персоналізованому підході до кожного користувача, що реалізується через розрахунок КБЖВ. Основною метою є автоматизація розрахунку індивідуальних норм калорійності та балансу макронутрієнтів (білків, жирів, вуглеводів).

Базальний рівень метаболізму (Basic Metabolic Rate, BMR) – це кількість калорій, які організм спалює у стані спокою для підтримки важливих функцій, таких як дихання, кровообіг та відновлення клітин[4]. Для підрахунку BMR використовується формула Міффіна-Сан Жеора. Вона вважається найбільш точною та має відмінності для жінок (3.1) та чоловіків (3.2).

$$BMR = 10 \times \text{вага(кг)} + 6,25 \times \text{зріст(см)} - 5 \times \text{вік(роки)} - 161. \quad (3.1)$$

$$BMR = 10 \times \text{вага(кг)} + 6,25 \times \text{зріст(см)} - 5 \times \text{вік(роки)} + 5. \quad (3.2)$$

Кінцевий показник добових енерговитрат (Total Daily Energy Expenditure, TDEE) – це загальна кількість калорій, які організм спалює за день[4]. TDEE розраховується за формулою (3.3):

$$TDEE = BMR \times PAL, \quad (3.3)$$

де PAL – коефіцієнт фізичної активності, значення якого обирається з бази даних відповідно до способу життя користувача.

Після визначення добових енерговитрат організму (TDEE) наступним етапом є адаптація калорійності відповідно до цілі користувача. Зміна маси тіла досягається шляхом створення енергетичного дефіциту або профіциту відносно рівня підтримки (TDEE)[5], тобто споживання меншої або більшої кількості калорій, ніж витрачається протягом доби.

З метою автоматизації цього процесу в системі «Soma» використовується узагальнена формула розрахунку цільової калорійності (3.4):

$$\text{Цільова калорійність} = TDEE \times gf, \quad (3.4)$$

де gf – це коефіцієнт, що відображає ціль користувача.

Коефіцієнт в залежності від цілі користувача може бути таким:

- при підтримці ваги: $gf = 1$;
- при схудненні: $gf = 0,85$;
- при наборі ваги: $gf = 1,15$.

Такий підхід є математичною інтерпретацією класичної схеми дефіциту/профіциту калорій, згідно з якою потрібно зменшувати або збільшувати калорійність відносно TDEE на певну величину[5]. Використання коефіцієнта дозволяє перейти від фіксованих значень (у ккал) до відносних (у відсотках), що спрощує розрахунки.

Розрахунок норми білків, жирів та вуглеводів у системі базується на енергетичному підході, де цільова калорійність розподіляється між макронутрієнтами відповідно до коефіцієнтів цілі. Розрахунок здійснюється за формулою (3.5):

$$\text{Нутрієнт}_Г = \frac{\text{Цільова калорійність} \times \text{Частка макронутрієнта}}{\text{Калорійність 1 г нутрієнта}}. \quad (3.5)$$

Частка макронутрієнта визначає відсотковий розподіл загальної добової калорійності між білками, жирами та вуглеводами відповідно до обраної мети користувача (схуднення, набір маси або підтримання ваги).

Кожен макронутрієнт має фіксовану калорійність[6], яка використовується для переведення калорій у грами:

- білки – 4 ккал/г;
- вуглеводи – 4 ккал/г;
- жири – 9 ккал/г.

Ці значення є стандартними фізіологічними коефіцієнтами, що відображають кількість енергії, яка виділяється при метаболізмі одного грама відповідного макронутрієнта.

Для підрахунку кількості спалених калорій під час виконання фізичних вправ у системі «Soma» використовується метаболічний еквівалент (MET). Цей підхід дозволяє врахувати не лише тривалість тренування, а й інтенсивність конкретного виду активності та поточну масу тіла користувача. Розрахунок описується формулою (3.6):

$$E = MET \times m \times t, \quad (3.6)$$

де MET – коефіцієнт інтенсивності виду активності;

m – поточна маса тіла користувача, кг;

t – тривалість активності, годин.

3.4 Інтеграція із зовнішнім сервісом FatSecret Platform API

Для розширення функціональних можливостей застосунку та забезпечення доступу до глобальної бази нутрієнтів було реалізовано взаємодію із зовнішньою платформою FatSecret. FatSecret – найбільша у світі база даних про їжу та харчування, що дозволяє створювати інноваційні рішення в сфері дієт, харчування та здоров'я. API платформи fatsecret забезпечує аналітику щодо харчування для провідних світових продуктів у сфері здоров'я, фітнесу та добробуту.[7] Процес інтеграції побудований на використанні сучасних стандартів безпеки та асинхронних мережевих запитів.

Взаємодія із зовнішнім ресурсом базується на стандарті безпеки OAuth 2.0. Використання цього протоколу дозволяє застосунку отримати доступ до бази даних без необхідності передавати особисті паролі користувача, замість цього використовуються спеціальні ідентифікатори клієнта (Client ID та Client Secret).

В основі безпеки інтеграції лежить принцип розмежування доступу та захисту конфіденційних даних. Унікальні ключі доступу клієнта не впроваджуються безпосередньо у програмний код. Для зберігання цієї інформації використано змінні оточення на стороні сервера. Такий підхід запобігає витoku облікових даних у разі публікації вихідного коду (наприклад, у систему контролю версій GitHub) та забезпечує відповідність стандартам розробки безпечного програмного забезпечення.

Процес отримання дозволу на запит даних виконується через спеціальний авторизаційний вузол. Для передачі даних використовується HTTP-запит методу POST, що ініціюється серверною частиною застосунку. Дані передаються у форматі, сумісному з URL-кодуванням рядків (application/x-www-form-urlencoded), що є обов'язковою вимогою протоколу OAuth 2.0.

Реалізація запитів базується на асинхронних методах, що дозволяє серверу здійснювати інші операції, не очікуючи відповіді від API, тим самим оптимізуючи загальну продуктивність системи.

Останнім етапом встановлення зв'язку з API є отримання та використання цифрового токена доступу. Після успішної автентифікації зовнішній сервіс надає тимчасовий токен, який виступає у ролі «цифрового ключа». Кожен наступний запит користувача до бази даних продуктів (пошук продуктів) автоматично включає цей токен у заголовки передачі даних. Такий підхід забезпечує цілісність сесії та гарантує, що запити надходять саме від авторизованого джерела, захищаючи обмін інформацією між застосунком та зовнішнім сервісом.

3.5 Реалізація локалізації

У застосунку «Soma» впроваджено локалізацію, яка забезпечує підтримку декількох мов (українська та англійська) як для елементів

інтерфейсу, так і для динамічних даних, що надходять із зовнішнього ресурсу FatSecret.

Підтримка локалізації статичних елементів (заголовків, підписів кнопок, тощо) реалізована за допомогою стандартних інструментів React та застосуванням словників перекладів.

Словники організовані у вигляді JSON-об'єктів, де кожному унікальному ключу відповідає текстове значення конкретною мовою. Це дозволяє розділити програмну логіку та контент для локалізації.

Ключі в словниках мають ієрархічну структуру (наприклад, `auth.login`), що забезпечує унікальність ідентифікаторів та полегшує пошук потрібних фрагментів тексту в коді.

Бібліотека `i18next` забезпечує керування словниками перекладів. Надає можливість миттєвого оновлення всього інтерфейсу без перезавантаження застосунку.

При першому запуску систему аналізує налаштування пристрою користувача та автоматично встановлює відповідну мовну локаль. Якщо мова системи українська, відповідно у застосунку встановлюється українська мова, в інших випадках – англійська. Попри автоматичне визначення мови передбачена можливість мануального вибору мови через налаштування на екрані профілю, що має вищий пріоритет над системними параметрами, а обрана мова зберігається у локальному сховищі пристрою для подальших сесій.

У разі відсутності перекладу для конкретного ключа у словнику, система автоматично звертається до базового словника (за замовчуванням англійської мови). Це гарантує, що користувач у будь-якому випадку побачить назву елемента, а не порожнє поле або код помилки.

Для гнучкого керування багатомовним контентом у PostgreSQL використано тип `JSONB`. На відміну від стандартного `JSON`, він зберігає дані у бінарному форматі, що виключає потребу в повторному парсингу та значно прискорює обробку запитів.

Замість створення розгалуженої мережі зв'язаних таблиць для перекладів, сутності (рецепти, категорії тощо) містять єдине поле `translations`. Це дозволяє зберігати назви та описи різними мовами в одному записі, уникаючи складних операцій JOIN. Доступ до даних здійснюється за мовними ключами (ISO-кодами) `uk` та `en`. Для роботи з подібними даними на рівні програмного коду реалізовано логіку, яка автоматично вилучає потрібне мовне значення:

- система отримує повний об'єкт із бази;
- допоміжна функція `tData` звертається до поля `translations`;
- виконується пошук за ключем поточної мови (наприклад, `'uk'` або `'en'`);
- якщо конкретне значення відсутнє, повертається дефолтний текст, що запобігає появі помилок в інтерфейсі.

Обраний підхід роботи з даними забезпечує масштабованість та оптимізацію. Додавання нової мови в застосунок не потребує додавання нових таблиць для перекладу у базу даних, достатньо лише додати новий ключ у JSON-об'єкти. Робота з одним JSON-полем значно простіша, ніж керування розгалуженою мережею таблиць для перекладів.

Найбільш складним етапом є локалізація даних, що надходять від FatSecret API в режимі реального часу.

Алгоритм працює за наступною схемою:

- якщо користувач виконує пошук українською мовою, система перекладає запит на англійську для коректної взаємодії з базою даних FatSecret;
- отримані результати (назви та описи продуктів) групуються в один масив і надсилаються до сервісу перекладу одним запитом. Це мінімізує кількість звернень до мережі та прискорює роботу застосунку;
- для економії ресурсів реалізовано внутрішній кеш перекладів на основі структури Map. Якщо користувач повторно шукає той самий продукт, система миттєво повертає переклад з пам'яті, не витрачаючи час на повторний запит до сервісу перекладу.

3.6 Візуалізація історії змін ваги

Для наочності прогресу користувача у застосунку візуалізовано історію змін ваги у вигляді часової шкали. Шкала не тільки відображає значення ваги, а й різницю у вазі, яка підраховується динамічно. Різниця має забарвлення, яке залежить від обраної цілі користувача (набір ваги: збільшення ваги – зелене забарвлення, зменшення – червоне; схуднення: навпаки; підтримка ваги: будь-яка різниця у вазі – жовте).

3.7 Висновок з розділу 3

Визначено сценарії взаємодії користувача з застосунком. Вони охоплюють повний цикл: від реєстрації та анкетування до щоденного моніторингу харчування та фізичної активності.

Впроваджена математична модель, що базується на розрахунках за формулою Міффіна-Сан Жеора та метаболічних еквівалентах (MET), дозволяє системі «Soma» здійснювати персоналізовані обчислення з високою точністю, враховуючи індивідуальні параметри тіла, спосіб життя користувача та його мету. Інтеграція із зовнішнім сервісом FatSecret API реалізована з дотриманням стандартів безпеки OAuth 2.0.

Застосування бінарного формату JSONB у PostgreSQL дозволило створити універсальну структуру зберігання багатомовного контенту. Розроблена система локалізації разом з алгоритмами кешування та пакетного перекладу забезпечує безперебійну роботу застосунку для україномовних користувачів.

Результати розробки, описані в даному розділі, підтверджують ефективність обраного технологічного стеку для автоматизації процесів контролю харчування та фізичного стану користувачів.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Роль застосунку у підтримці здорового способу життя

Основним призначенням розробленого застосунку «Soma» є перехід від пасивного спостереження за показниками здоров'я у активне та персоналізована стратегія управління способом життя. У традиційному підході до самоконтролю користувачі часто зустрічаються із проблемою хаотичного аналізу: ваги, калорійності та енергетичної цінності продуктів, фізичної активності. В такому випадку ці всі дані виглядають як розрізнені цифри, що не складаються у цілісну картину. Більшість існуючих рішень або складні професійні інструменти або прості калькулятори, що не враховують індивідуальність користувача.

Застосунок «Soma» стає повноцінним цифровим помічником у смартфоні, що поєднує всі аспекти контролю ваги та активності в єдину систему. На відміну від статичних додатків, «Soma» ініціює повторний запуск алгоритму Міффіна-Сан Жеора при кожній зміні ваги користувача, його цілі чи рівня активності, адаптуючи норми КБЖВ в реальному часі. Завдяки інтеграції з FatSecret API користувач отримує доступ до світової бази продуктів, яка автоматично локалізується українською мовою через впроваджені алгоритми перекладу. Реалізований підрахунок спалених калорій, який базується на метаболічних еквівалентах дозволяє користувачеві точно оцінювати вплив фізичних активностей на свій добовий прогрес. Інтегровані рецепти корисних страв із зазначенням не тільки інгредієнтів та кроків приготування, а й кількості нутрієнтів та відеоматеріали тренувань створюють готову інфраструктуру для здорового способу життя, усуваючи необхідність пошуку інформації у сторонніх джерелах.

Крім функціонального аспекту, застосунок «Soma» несе важливу функцію формування у користувача звички свідомого споживання їжі та

розуміння користі фізичних навантажень. Користувач отримує ефективний інструмент для підтримки фізичного добробуту.

4.2 Користувацький інтерфейс(UI)

Користувацький інтерфейс застосунку «Soma» розроблений із акцентом на мінімалізм та високу швидкість взаємодії. Головною метою дизайну є забезпечення легкого доступу до функціоналу без надмірного когнітивного навантаження на користувача.

Дизайн системи побудований таким чином, щоб найважливіша інформація завжди залишалася в зоні першочергової уваги. Використання великих інтерактивних віджетів дозволяє зчитувати потрібну інформацію швидко. Дані розділені на логічні блоки, що забезпечує впорядкованість та консистентність інтерфейсу на всіх екранах застосунку.

Для забезпечення максимального комфорту при постійному використанні застосунку було розроблено навігацію таким чином: використано липкий футер (sticky footer) з іконками високої якості, що забезпечує миттєвий доступ до чотирьох основних розділів системи (Home, Activity, Meal, Profile) одним натисканням. Навігаційні ланцюжки дозволяють користувачеві легко орієнтуватись усередині розділів (наприклад, при виборі тренування в категорії) та швидко повертатися на попередній екран. Кожна дія користувача, будь то натискання кнопки чи перемикання таба, супроводжується візуальною зміною стану елемента. Це створює відчуття «живого» інтерфейсу та підтверджує успішність виконання дій.

У застосунку розроблено 2 теми: світла та темна. Світла тема використовує світлі відтінки фіолетового та білого для створення відчуття легкості та відкритості. Темна тема базується на глибоких темних відтінках фіолетового з контрастними світлими. Використання темного фону суттєво знижує навантаження на зір, особливо у вечірній час та забезпечує високу енергоефективність на пристроях з OLED/AMOLED екранами. Введення ваги

продуктів або тривалості вправ реалізовано через адаптивні модальні вікна та Time Picker, що мінімізує кількість помилок при введенні. У розділах рецептів та тренувань використовуються картки з «лінивим завантаженням» (lazy loading). Це дозволяє плавно відображати контент без затримок у роботі інтерфейсу. Використано графіки, які відображають прогрес зміни у вигляді лінійних діаграм на основі історії записів, що дозволяє користувачеві наочно бачити зміни.

Інтерфейс підтримує миттєве перемикання між світлою та темною темами без втрати контрастності елементів. Також інтерфейс має можливість повністю адаптуватись під обрану мову (українську або англійську), зберігаючи цілісність. Екран профілю містить гнучкий інструментарій для створення персоналізованих нагадувань про прийоми їжі та тренування.

4.3 Умови функціонування застосунку

Для стабільного функціонування застосунку «Soma» необхідно дотримуватися визначених апаратних та програмних вимог.

Апаратне забезпечення:

- тип пристрою: смартфон або планшет на базі iOS або Android;
- процесор: 4-ядерний процесор з тактовою частотою від 1.8 ГГц для швидкої обробки інтерфейсу та анімацій;
- оперативна пам'ять: не менше 2 ГБ RAM (рекомендовано 4 ГБ для плавної роботи модулів із відеоматеріалами тренувань);
- дисплей: сенсорний екран з роздільною здатністю від 720x1280 пікселів;
- мережа: наявність стабільного інтернет-з'єднання (Wi-Fi, 4G або 5G).

Програмне забезпечення:

- операційна система: Android 10.0 і вище або iOS 12.0 і вище;
- наявність актуальності сервісів Google Play (для Android) та Apple ID (для iOS).

4.4 Дослідження роботи застосунку

Для дослідження належної працездатності мобільного застосунку «Soma» було проведено тестування, що охоплює основні сценарії взаємодії користувача із застосунком. Кожен етап тестування зображено на скриншотах, отриманих під час роботи застосунку на ОС Android 16.

Першим етапом перевірки став процес створення облікового запису та анкетування. Тестування підтвердило, що після збереження параметрів, вони зберігаються в профілі користувача та на головному екрані коректно розраховуються норми КБЖВ. Результати зображено на рисунках 4.1-4.2.

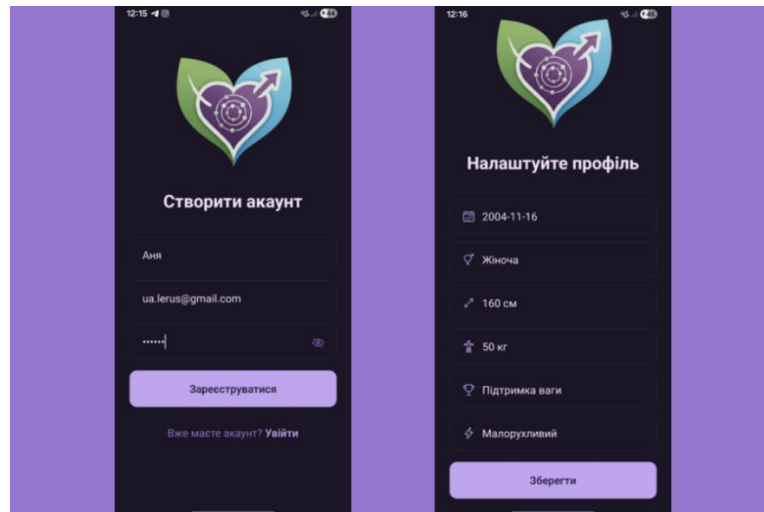


Рисунок 4.1 – Реєстрація та налаштування профілю

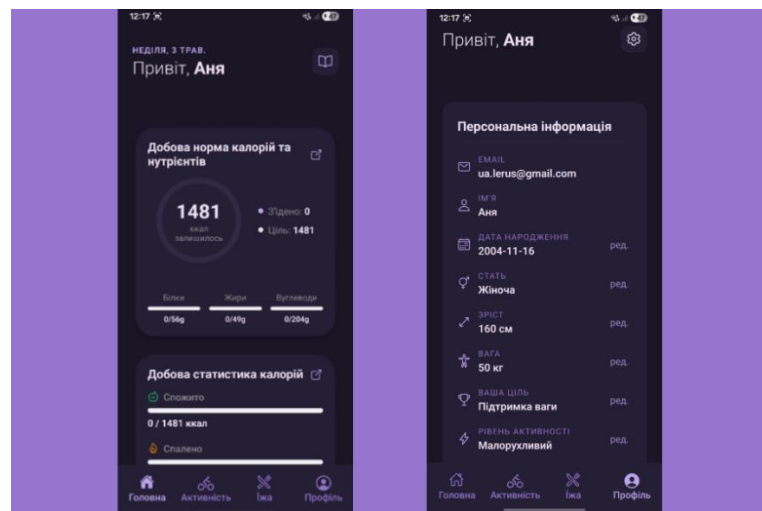


Рисунок 4.2 – Налаштований профіль користувача та розрахунок КБЖВ

Одним з ключових функцій застосунку є облік харчування. Він передбачає вибір прийому їжі, пошук продукту через зовнішнє API або додавання власного при відсутності результатів пошуку та додавання запису у щоденник. Тестування підтвердило коректність роботи функціоналу. Результати зображено на рисунках 4.3-4.4.

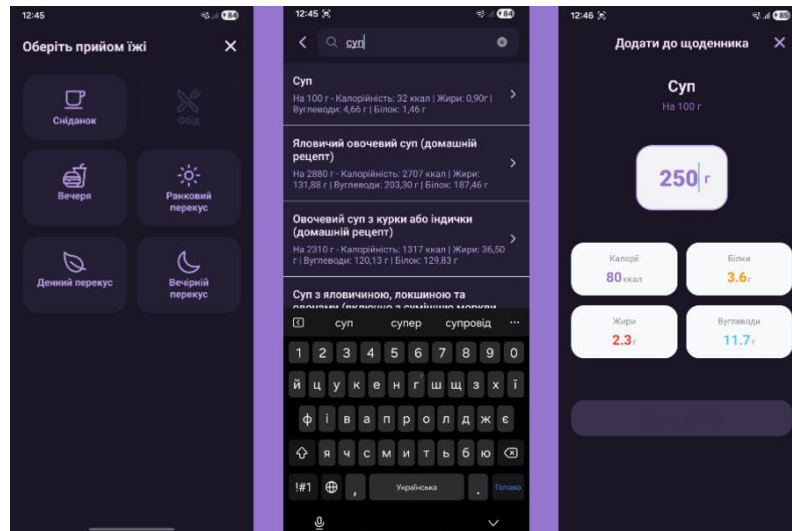


Рисунок 4.3 – Додавання запису про прийом їжі

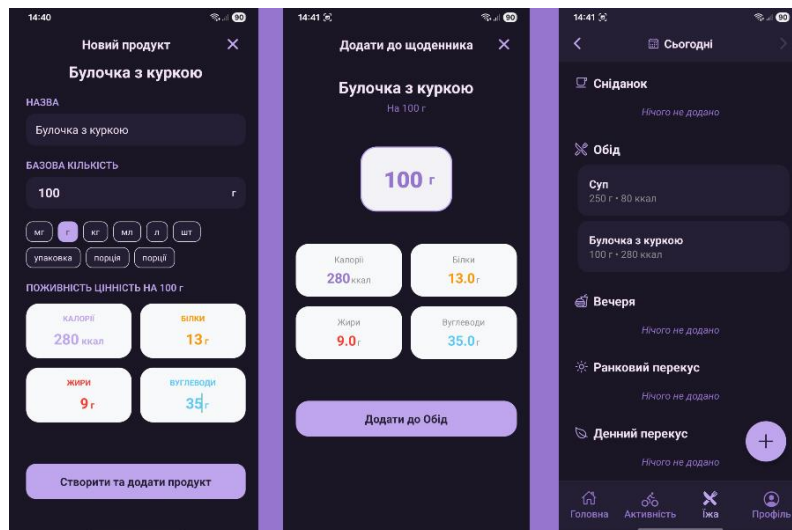


Рисунок 4.4 – Додавання свого продукту в запис прийому їжі

Облік фізичних активностей передбачає вибір категорії активності, її виду та тривалості. Добова статистика включає в себе добову норму калорій та нутрієнтів та статистику спожитих, спалених калорій. Тестування

підтвердило коректність додавання запису в щоденник активностей та коректність відображення статистики. Результати зображено на рисунках 4.5-4.6.

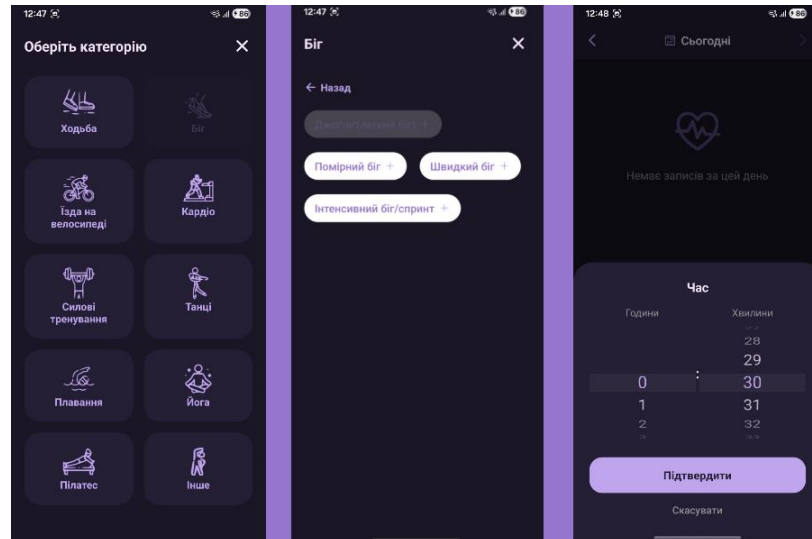


Рисунок 4.5 – Додавання запису про активність

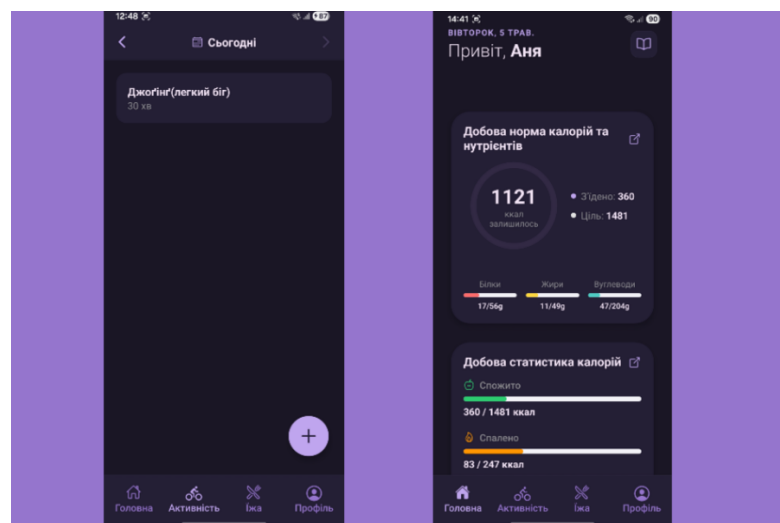


Рисунок 4.6 – Доданий запис та добова статистика

4.5 Висновок з розділу 4

Визначено, що застосунок «Soma» виконує роль персонального цифрового помічника, який забезпечує автоматизацію розрахунків КБЖВ в реальному часі на основі динамічних змін параметрів користувача.

Реалізований інтерфейс забезпечує високу швидкість взаємодії та мінімізують когнітивне навантаження. Наявність світлої та темної тем підвищує ергономічність продукту та енергоефективність на пристроях з OLED-екранами.

Визначено апаратні та програмні вимоги для стабільної роботи застосунку. Підтверджено коректне функціонування застосунку на сучасних пристроях.

Проведення тестування основних сценаріїв (реєстрація, облік харчування, фізичних активностей) підтвердило повну відповідність розробленого програмного забезпечення визначеним функціональним вимогам.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було реалізовано кросплатформний мобільний застосунок «Soma», призначений для підтримки здорового способу життя.

У ході роботи над проєктом було досягнуто наступних результатів та висновків:

- аналіз сучасних рішень на ринку показав, що більшість застосунків мають обмеження у локалізації для українського ринку та перевантажений інтерфейс або доступ до необхідного функціоналу лише за підписку, що обґрунтовує необхідність розробки власного продукту;

- для розробки було обрано фреймворк React Native для клієнтської частини та Node.js із Express для сервера. Такий технологічний стек забезпечив кросплатформність застосунку та високу швидкість обробки запитів;

- спроектовано реляційну базу даних на базі PostgreSQL. Що дозволило гнучко зберігати дані, а Prisma ORM спрощує взаємодію з даними під час розробки та підтримки продукту;

- у застосунок було інтегровано FatSecret API для отримання актуальних даних про продукти та їх нутрієнти. Реалізована математична модель забезпечила автоматичний та точний розрахунок добових норм КБЖВ індивідуально для кожного користувача;

- приділено особливу увагу до локалізації застосунку

- тестування основних сценаріїв взаємодії користувача з застосунком підтвердила повну працездатність системи та її відповідність усім функціональним вимогам.

Розроблений застосунок є готовим програмним продуктом, який допомагає користувачам ефективно керувати своїм раціоном і фізичною активністю, роблячи процес моніторингу здоров'я простим та автоматизованим.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Premium+ Is Here! Find Out Which MyFitnessPal Membership Is Right For You URL: <https://blog.myfitnesspal.com/myfitnesspal-membership-pricing-tiers/> (data of access: 01.05.2026).
2. New features for our free mobile apps - Reports and Goal Setting URL: <https://blog.fatsecret.com/post/144236150761/new-features-for-our-free-mobile-apps-reports> (data of access: 01.05.2026).
3. Here's the All New Samsung Health UI URL: <https://r2.community.samsung.com/t5/Tech-Talk/Here-s-the-All-New-Samsung-Health-UI/td-p/14537477> (data of access: 01.05.2026).
4. How Many Calories Do You Need To Eat Per Day To Gain Muscle? URL: <https://www.health.com/how-many-calories-a-day-to-gain-muscle-11801950> (data of access: 01.05.2026).
5. TDEE Explained: How to Calculate Your Total Daily Energy Expenditure (2026) URL: <https://hakarui.io/guides/tdee-guide> (data of access: 01.05.2026).
6. What Are Macronutrients? All You Need to Know URL: <https://www.healthline.com/nutrition/what-are-macronutrients> (data of access: 01.05.2026).
7. API платформи fatsecret для даних про їжу та харчування URL: <https://www.fatsecret.com/uk> (дата звернення: 01.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Об'єктом розробки є кросплатформний мобільний застосунок «Soma», призначений для підтримки здорового способу життя. Застосунок орієнтований на широке коло користувачів, які прагнуть перейти до свідомого управління власним способом життя шляхом відстеження своїх харчових звичок та фізичної активності без використання складного професійного обладнання чи платних сервісів.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація кросплатформного мобільного застосунку для підтримки здорового способу життя.», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

A.2 Призначення розробки

Застосунок призначений для щоденного контролю харчування та фізичної активності з метою підтримки здорового способу життя.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Даний підрозділ визначає повний перелік функціональних можливостей, які повинні бути реалізовані в мобільному застосунку «Soma» для забезпечення ефективного контролю показників користувача. Вимоги сформовані на основі аналізу потреб цільової аудиторії та математичних моделей енергетичного обміну, що лежать в основі системи.

Застосунок «Soma» повинен забезпечувати виконання таких функцій:

- створення унікального облікового запису шляхом введення імені користувача (логіна), електронної пошти та пароля. Вхід повинен здійснюватись за електронною поштою та паролем із подальшим доступом до повного функціоналу застосунку;
- збір та збереження даних користувача про стать, вік, поточну вагу, зріст, рівень щоденної фізичної активності та мету(вибір однієї з трьох опцій – схуднення, підтримка ваги або набір ваги);
- автоматизований розрахунок цільової добової норми КБЖВ на основі введених даних;
- додавання записів про прийоми їжі із використанням інтегрованої бази даних продуктів. У разі відсутності продукту в базі, система повинна дозволяти користувачеві самостійно вводити назву та склад продукту;
- додавання записів про фізичні активності шляхом вибору категорії та виду активності і вказання тривалості навантаження. Розрахунок витрачених калорій повинен проводитися з урахуванням коефіцієнтів інтенсивності вправ та ваги користувача;
- формування та відображення загальної статистики співвідношення спожитих та витрачених калорій відносно встановленої добової норми;
- формування та відображення історії змін ваги користувача з можливістю перегляду динаміки змін;
- надання доступу до переліку рецептів здорового харчування відео тренувань як довідкової інформації для користувача;
- реалізація механізму надсилання повідомлень про прийоми їжі або виконання тренувань;
- можливість для користувача самостійно вмикати/вимикати сповіщення та встановлювати конкретний час їх надсилання.

А.3.2 Вимоги до інтерфейсу програми

Ефективність використання мобільного застосунку безпосередньо залежить від якості проектування користувацького інтерфейсу (UI) та зручності взаємодії з ним (UX). Інтерфейс повинен бути розроблений таким чином, щоб ключова інформація зчитувалась швидко без надмірного когнітивного навантаження.

Розроблений інтерфейс має відповідати таким вимогам:

- колірна гама, шрифти та стилі кнопок мають бути однаковими на всіх екранах застосунку для забезпечення цілісного сприйняття інтерфейсу;
- будь-яка дія користувача (натискання кнопки, успішне збереження запису) має супроводжуватися візуальним підтвердженням або зміною стану елемента інтерфейсу;
- система повинна підтримувати перемикання між світлою та темною темами;
- доступ до основних функціональних модулів (Головна, Активність, Їжа, Профіль) має здійснюватися через фіксовану нижню панель навігації;
- добові показники повинні відображатися у вигляді інтерактивних віджетів на головному екрані;
- система має використовувати контекстні кольори для позначення статусу прогресу (наприклад, зелений для позитивної динаміки, червоний – відхилення від мети);
- вибір часу тренувань або дати народження має реалізовуватися через адаптивні нативні компоненти (time/date pickers) для мінімізації помилок при введенні;
- інтерфейс має повністю адаптуватися під обрану мову (українська чи англійська), включаючи підписи кнопок, системні повідомлення та одиниці вимірювання (кг, ккал).

А.3.3 Вимоги до надійності

Надійність функціонування мобільного застосунку «Soma» є критично важливим аспектом розробки, оскільки система працює з персональними даними користувачів. Забезпечення безперебійної роботи застосунку вимагає комплексного підходу до захисту даних, стійкості до мережових збоїв та безпеки автентифікації. Основні вимоги до надійності:

- система повинна забезпечувати коректну безперебійну обробку запитів, використовуючи асинхронну модель Node.js для запобігання блокуванню потоків при взаємодії з FatSecret API;

- програмне забезпечення має містити блоки перехоплення помилок (try-catch), що запобігають аварійному завершенню роботи застосунку при втраті інтернет-з'єднання або отриманні некоректних даних від зовнішніх сервісів;

- система повинна виконувати обов'язкову перевірку всіх даних, що вводяться користувачем для запобігання збереженню некоректних значень у базі даних;

- автентифікація користувачів має реалізовуватися через механізм JWT-токенів (Access та Refresh), що забезпечує безпеку сесій та захист від несанкціонованого доступу до персональних даних;

- паролі користувачів мають зберігатися обов'язково у хешованому вигляді, що запобігає їх викраденню у відкритому форматі;

- взаємодія із зовнішніми сервісами (FatSecret) повинна базуватися на протоколі OAuth 2.0, а конфіденційні ключі (Client Secret) мають обов'язково зберігатися у змінних оточення на сервері, без прямого доступу через клієнтську частину.

А.3.4 Вимоги до технічних засобів

Для забезпечення стабільної роботи застосунку необхідно дотримуватися встановлених мінімальних і рекомендованих апаратних вимог. Система базується на клієнт-серверній архітектурі, тому технічні засоби поділяються на пристрої кінцевих користувачів (смартфони, планшети) та пристрої для розгортання серверної логіки й бази даних.

Вимоги до пристроїв користувача:

- тип пристрою: смартфон або планшет на операційній системі Android (версії 10.0 і вище) або iOS (версії 12.0 і вище);
- процесор: 4-ядерний мобільний процесор із тактовою частотою від 1.8 ГГц для плавної обробки анімацій інтерфейсу та роботи з відеоматеріалами тренувань;
- оперативна пам'ять: мінімальний обсяг – 2 ГБ RAM, рекомендований обсяг для стабільної роботи при паралельному використанні інших застосунків – 4 ГБ;
- дисплей: сенсорний екран із роздільною здатністю від 720x1280 пікселів;
- інтернет-з'єднання: підтримка Wi-Fi, 4G або 5G.

Вимоги до пристроїв розгортання серверної логіки:

- процесор: віртуальний або фізичний сервер із 2-ядерним процесором;
- пам'ять: від 1 ГБ RAM для роботи середовища виконання та взаємодії з Prisma ORM;
- дисковий простір: не менше 5 ГБ вільного місця на SSD-накопичувачі для зберігання бази даних, логів сервера та кешу перекладів.

ДОДАТОК Б
Код програми

Б.1 Текст файла nutritionCalculator.util.js

```
import * as nutritionConstants from '../constants/nutrition.constants.js';
export const calculateBMR = ({ weight, height, age, gender }) => {
  let bmr
  const bmrConstants = nutritionConstants.BMR;
  if(gender === 'female') {
    bmr = bmrConstants.WEIGHT_COEFFICIENT * weight +
    bmrConstants.HEIGHT_COEFFICIENT * height -
    bmrConstants.AGE_COEFFICIENT * age +
    bmrConstants.GENDER_CONSTANT_FEMALE
  } else {
    bmr = bmrConstants.WEIGHT_COEFFICIENT * weight +
    bmrConstants.HEIGHT_COEFFICIENT * height -
    bmrConstants.AGE_COEFFICIENT * age +
    bmrConstants.GENDER_CONSTANT_MALE
  }
  return bmr
}
export const calculateTDEE = (bmr, activityLevelCoefficient) => {
  return bmr * activityLevelCoefficient
}
export const calculateCaloriesForUser = (tdee, goalCoefficient) => {
  const result = tdee * goalCoefficient
  return Math.ceil(result)
}

export const calculateMacros = (totalCalories, goal) => {
  const goalKey = goal.toUpperCase();
  const ratio = nutritionConstants.MACRO_RATIOS[goalKey];
```

```

    const caloriesPerGram = nutritionConstants.CALORIES_PER_GRAM;
    const proteinGrams = Math.round((totalCalories * ratio.PROTEIN) /
caloriesPerGram.PROTEIN);
    const fatGrams = Math.round((totalCalories * ratio.FAT) /
caloriesPerGram.FAT);
    const carbsGrams = Math.round((totalCalories * ratio.CARBS) /
caloriesPerGram.CARBS);
    return {
      protein: proteinGrams,
      fat: fatGrams,
      carbs: carbsGrams
    }
  }
}

```

Б.2 Текст файла **nutrition.service.js**

```

import * as userInfoRepo from '../repositories/userInfo.repository.js';
import * as nutritionCalculator from '../utils/nutritionCalculator.util.js';
import * as dateFormatter from '../utils/dateFormatter.util.js'
import AppError from '../utils/AppError.js';
export const getUserNutrition = async (userId) => {
  const userInfo = await userInfoRepo.getUserInfoByUserId(userId);
  if (!userInfo) throw new AppError('User profile not found. Please complete your
profile setup.', 404);
  const age = dateFormatter.calculateAge(userInfo.dateOfBirth)
  const bmr = nutritionCalculator.calculateBMR({
    weight: Number(userInfo.currentWeightKg),
    height: Number(userInfo.heightCm),
    age,
    gender: userInfo.gender
  })
}

```

```

    })
    const tdee = nutritionCalculator.calculateTDEE(bmr,
    Number(userInfo.activityLevel.coefficient));
    const caloriesForUser = nutritionCalculator.calculateCaloriesForUser(tdee,
    Number(userInfo.goal.coefficient));
    const macros = nutritionCalculator.calculateMacros(caloriesForUser,
    userInfo.goal.code);
    return { caloriesForUser, macros, details: { bmr, tdee } }
  }
}

```

Б.3 Текст файла **nutrition.controller.js**

```

import * as nutritionService from "../services/nutrition.service.js";
export const getUserNutrition = async (req, res, next) => {
  try {
    const userId = req.user.id;
    const result = await nutritionService.getUserNutrition(userId);
    res.json(result);
  } catch (e) {
    next(e)
  }
}

```

Б.4 Текст файла **userPhysicalActivities.service.js**

```

import * as userInfoRepo from "../repositories/userInfo.repository.js";
import * as userPhysicalActivitiesRepo from
"../repositories/userPhysicalActivities.repository.js"
import * as dateFormatter from '../utils/dateFormatter.util.js'
import AppError from "../utils/AppError.js";

```

```

export const addPhysicalActivity = async (userId, data) => {
  const {typeOfPhysicalActivityId, activityDuration, activityDate } = data
  const activityDateValue = prepareActivityDate(activityDate)
  const result = await userPhysicalActivitiesRepo.createDailyActivity({
    userId,
    typeOfPhysicalActivityId,
    activityDate: activityDateValue,
    activityDuration
  });
  return result;
}

export const updateActivityDuration = async (id, data ) => {
  return await userPhysicalActivitiesRepo.updateDailyActivityDuration(id, {
    activityDuration: data.activityDuration
  })
}

export const deleteActivity = async (id) => {
  return await userPhysicalActivitiesRepo.deleteDailyActivity(id)
}

const prepareActivityDate = (date) => {
  return date ? dateFormatter.toDate(date) : dateFormatter.getToday()
}

export const calculateDailyCaloriesBurned = async (userId, date) => {
  const user = await userInfoRepo.getUserInfoByUserId(userId);
  if (!user) throw new AppError("User info not found", 404);
  const userWeight = user.currentWeightKg;
  const activities = await
userPhysicalActivitiesRepo.getDailyActivitiesByDate(userId, date)
  const totalCalories = activities.reduce((total, activity) => {
    const coefficient = activity.typeOfPhysicalActivity.coefficient

```

```

    const duration = activity.activityDuration
    const hours = duration / 60
    return total + (coefficient * userWeight * hours)
  }, 0)
  return Math.round(totalCalories)
}

```

Б.5 Текст файла `userPhysicalActivities.controller.js`

```

import * as userPhysicalActivitiesService from
"../services/userPhysicalActivities.service.js";
import * as dateFormatter from '../utils/dateFormatter.util.js'
import * as physicalActivityRepo from
'../repositories/physicalActivities.repository.js'
import { getDailyActivitiesByDate } from
'../repositories/userPhysicalActivities.repository.js'
export const createDailyActivity = async (req, res, next) => {
  try {
    const userId = req.user.id
    const activity = await userPhysicalActivitiesService.addPhysicalActivity(userId,
req.body)
    res.status(201).json(activity)
  } catch (e) {
    next(e)
  }
}
export const updateDailyActivity = async (req, res, next) => {
  try {
    const id = Number(req.query.id)
    const data = req.body

```

```

    const updated = await userPhysicalActivitiesService.updateActivityDuration(id,
data)
    res.json(updated)
  } catch (e) {
    next(e)
  }
}

```

```

export const deleteDailyActivity = async (req, res, next) => {
  try {
    const id = Number(req.query.id)
    await userPhysicalActivitiesService.deleteActivity(id)
    res.status(204).send()
  } catch (e) {
    next(e)
  }
}

```

```

export const getDailyCaloriesBurned = async (req, res, next) => {
  try {
    const userId = req.user.id
    const date = req.query.date ? dateFormatter.toDate(req.query.date) :
dateFormatter.getToday();
    const totalCalories = await
userPhysicalActivitiesService.calculateDailyCaloriesBurned(userId, date)
    res.json({ date: dateFormatter.formatDate(date), totalCalories })
  } catch (e) {
    next(e)
  }
}

```

```

export const getAllCategoriesWithTypes = async (req, res, next) => {

```

```

    try {
      const categories = await physicalActivityRepo.findAllCategoriesWithTypes()
      res.status(200).json(categories)
    } catch (e) {
      next(e)
    }
  }
}

export const getDailyActivities = async (req, res, next) => {
  try {
    const userId = req.user.id
    const date = req.query.date ? dateFormatter.toDate(req.query.date) :
dateFormatter.getToday()

    const activities = await getDailyActivitiesByDate(userId, date)

    return res.status(200).json(activities)
  } catch (e) {
    next(e)
  }
}

```

Б.6 Текст файла index.jsx

```

import React, { useEffect, useState, useCallback } from 'react'
import { View, Text, ScrollView, RefreshControl, TouchableOpacity } from 'react-
native'
import { useAuth } from '../context/AuthContext.js'
import { useTheme } from '../constants/theme.js'
import { indexStyles } from '../assets/styles/index.styles.js'
import api from "../services/api"

```

```

import { Icons } from "@expo/vector-icons"
import { useRouter } from 'expo-router'
import { useTranslation } from 'react-i18next'
import i18n from '../i18n/index.js'
import { DailyNutritionCard } from '../components/DailyNutritionCard.jsx'
import { DailyCaloriesStatsCard } from
'../components/DailyCaloriesStatsCard.jsx'
import { WeightHistoryCard } from '../components/WeightHistoryCard.jsx'
const HomeScreen = () => {
  const router = useRouter()
  const { user } = useAuth()
  const { t } = useTranslation()
  const currentLocale = i18n.language === 'uk' ? 'uk-UA' : 'en-US';
  const COLORS = useTheme()
  const indexS = indexStyles(COLORS)
  const [refreshing, setRefreshing] = useState(false)
  const [nutritionData, setNutritionData] = useState({
    targets: { calories: 0, protein: 0, fat: 0, carbs: 0, bmr: 0, tdee: 0 },
    consumed: { calories: 0, protein: 0, fat: 0, carbs: 0 },
    stats: { burned: 0, net: 0 },
    weightHistory: [],
    goal: null,
    loading: true
  })
  const fetchHomeData = async () => {
    try {
      const [targetRes, consumedRes, statsRes, weightRes, userRes] = await
Promise.all([
        api.get('/nutrition/daily-calories'),
        api.get('/nutrition/daily-macros'),

```

```

    api.get('/nutrition/daily-stats'),
    api.get('/weight/stats'),
    api.get('/user/me')
  ])
  const consumedTotals = consumedRes.data.data.totals
  const dailyStats = statsRes.data
  const weightEntries = weightRes.data?.entries || []
  const goalCode = userRes.data.userInfo?.goal?.code
  setNutritionData({
    targets: {
      calories: targetRes.data.caloriesForUser || 0,
      protein: targetRes.data.macros?.protein || 0,
      fat: targetRes.data.macros?.fat || 0,
      carbs: targetRes.data.macros?.carbs || 0,
      bmr: targetRes.data.details?.bmr || 0,
      tdee: targetRes.data.details?.tdee || 0,
    },
    consumed: {
      calories: parseFloat(consumedTotals?.calories) || 0,
      protein: parseFloat(consumedTotals?.proteins) || 0,
      fat: parseFloat(consumedTotals?.totalFats) || 0,
      carbs: parseFloat(consumedTotals?.totalCarbs) || 0,
    },
    stats: {
      burned: parseFloat(dailyStats?.burned) || 0,
      net: parseFloat(dailyStats?.net) || 0,
    },
    weightHistory: weightEntries,
    goal: goalCode,
    loading: false
  })

```

```

    })
  } catch (e) {
    setNutritionData(prev => ({ ...prev, loading: false }))
  }
}

useEffect(() => {
  fetchHomeData()
}, [])

const onRefresh = useCallback(() => {
  setRefreshing(true)
  fetchHomeData().then(() => setRefreshing(false))
}, [])

const currentDate = new Date().toLocaleDateString(currentLocale, {
  weekday: 'long',
  day: 'numeric',
  month: 'short'
})

return (
  <ScrollView
    showsVerticalScrollIndicator={false}
    contentContainerStyle={{ paddingBottom: 50, backgroundColor:
COLORS.background }}
    refreshControl={<RefreshControl refreshing={refreshing}
onRefresh={onRefresh} />}
  >
    <View style={indexS.container}>
      <View style={indexS.header}>
        <View>
          <Text style={{ color: COLORS.primary, fontSize: 12, fontWeight: '700',
textTransform: 'uppercase', letterSpacing: 1 }}>

```

```

        {currentDate}
      </Text>
      <Text style={indexS.title}>
        <Text style={{ fontWeight: '300' }}>{t('home.hello')}</Text>
      {user?.name}
    </Text>
  </View>
  <TouchableOpacity
    style={indexS.headerButton}
    onPress={() => router.push('/healthy-vibe')}
  >
    <Icons name="book-outline" size={24} color={COLORS.primary} />
  </TouchableOpacity>
</View>
<DailyNutritionCard
  consumed={nutritionData.consumed}
  targets={nutritionData.targets}
  onPress={() => router.push('/nutrition-history')}
/>
<DailyCaloriesStatsCard
  consumed={nutritionData.consumed}
  targets={nutritionData.targets}
  stats={nutritionData.stats}
  onPress={() => router.push('/nutrition-history')}
/>
<WeightHistoryCard
  history={nutritionData.weightHistory}
  goalCode={nutritionData.goal}
  onPress={() => router.push('/weight-history')}
/>

```

```

    </View>
  </ScrollView>
)
}
export default HomeScreen

```

Б.7 Текст файла fatsecretAuth.api.js

```

import axios from 'axios';
const FATSECRET_OAUTH_URL = process.env.FATSECRET_OAUTH_URL;
export async function getAccessToken() {
  const response = await axios.post(
    FATSECRET_OAUTH_URL,
    new URLSearchParams({
      grant_type: 'client_credentials',
      scope: 'basic',
    }),
    {
      auth: {
        username: process.env.FATSECRET_CLIENT_ID,
        password: process.env.FATSECRET_CLIENT_SECRET,
      },
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
    }
  );
  return response.data.access_token;
}

```

Б.8 Текст файла `fatsecretApi.service.js`

```
import axios from 'axios';
import { getAccessToken } from '../integrations/fatsecretAuth.api.js';
const FATSECRET_API_URL = process.env.FATSECRET_API_URL;
export async function fatsecretRequest(method, params = {}) {
  const token = await getAccessToken();
  const response = await axios.post(
    FATSECRET_API_URL,
    new URLSearchParams({
      method,
      format: 'json',
      ...params,
    }),
    {
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/x-www-form-urlencoded',
      },
    }
  );
  return response.data;
}
```

Б.9 Текст файла `fatsecret.provider.js`

```
import { fatsecretRequest } from '../services/fatsecretApi.service.js'
export const searchExternalFoods = async (queryEn, limit) => {
  const data = await fatsecretRequest('foods.search', {
    search_expression: queryEn,
```

```

    max_results: limit
  })
  const foods = data.foods?.food || []
  const normalizedFoods = Array.isArray(foods) ? foods : [foods];
  return normalizedFoods.map(f => ({
    externalId: f.food_id,
    name: f.food_name,
    description: f.food_description
  })))
}

```

Б.10 Текст файлы translator.js

```

import { translate } from '@vitalets/google-translate-api'
const translationCache = new Map()
export const translateText = async (text, targetLang) => {
  if (!text || text.trim() === "") return text
  const cacheKey = `${text}_${targetLang}`
  if (translationCache.has(cacheKey)) return translationCache.get(cacheKey)
  try {
    const res = await translate(text, { to: targetLang })
    translationCache.set(cacheKey, res.text)
    return res.text
  } catch (e) {
    console.error(`[Translate Error] ${e.name}: returning original text.`)
    return text
  }
}
export const translateBatch = async (texts, targetLang) => {
  if (!texts || texts.length === 0) return []

```

```

const uniqueToTranslate = [...new Set(texts.filter(t => t && t.trim().length >
0))]

const missingInCache = uniqueToTranslate.filter(t =>
!translationCache.has(`${t}_${targetLang}`))

if (missingInCache.length > 0) {
  try {
    const separator = ' ||| '
    const combinedText = missingInCache.join(separator)
    const res = await translate(combinedText, { to: targetLang })
    const translatedArray = res.text.split(/\|\/).map(s => s.trim())
    missingInCache.forEach((original, index) => {
      const translatedValue = translatedArray[index] || original
      translationCache.set(`${original}_${targetLang}`, translatedValue)
    })
  } catch (e) {
    console.error(`[Batch Translate Error] ${e.name}: Fallback to originals.`)
  }
}

return texts.map(original => {
  if (!original || original.trim() === "") return original
  const cached = translationCache.get(`${original}_${targetLang}`)
  return cached || original
})
}

```

Б.11 Текст файлы userDailyNutrition.service.js

```

import * as foodItemRepo from '../repositories/foodItem.repository.js'
import * as userDailyNutritionRepo from
'../repositories/userDailyNutrition.repository.js'

```

```

import * as userPhysicalActivitiesService from
'../services/userPhysicalActivities.service.js'
import { parseNutrition } from '../utils/parseNutritionDescr.util.js'
import * as translator from '../localization/translator.js'
import { searchExternalFoods } from '../providers/fatsecret.provider.js'
import * as dateFormatter from '../utils/dateFormatter.util.js'
import AppError from '../utils/AppError.js'
export const foodSearch = async (query, targetLang, limit = 10, userId) => {
  try {
    const isEn = targetLang === 'en';
    let queryEn = query;
    if (!isEn) {
      try {
        queryEn = await translator.translateText(query, 'en');
      } catch (e) {
      }
    }
    const results = await Promise.allSettled([
      foodItemRepo.searchFoodsByName(query, limit, userId),
      searchExternalFoods(queryEn, limit)]);
    const localFoods = results[0].status === 'fulfilled' ? results[0].value : [];
    const externalFoods = results[1].status === 'fulfilled' ? results[1].value : [];
    const enrichedExternalFoods = (externalFoods || []).map(food => ({
      ...food,
      parsedNutrition: parseNutrition(food.description)
    }));
    if (isEn || enrichedExternalFoods.length === 0) {
      return { localFoods, externalFoods: enrichedExternalFoods };
    }
    try {

```

```

    const translatedExternal = await
translateExternalResults(enrichedExternalFoods, targetLang);
    return { localFoods, externalFoods: translatedExternal };
  } catch (e) {
    return { localFoods, externalFoods: enrichedExternalFoods };
  }
} catch (e) {
  return foods
}
};

const translateExternalResults = async (foods, targetLang) => {
  try {
    const stringsToTranslate = foods.flatMap(f => [f.name || "", f.description || ""])
    const translatedValues = await translator.translateBatch(stringsToTranslate,
targetLang)
    if (!Array.isArray(translatedValues)) {
      return foods
    }
    return foods.map((food, index) => ({
      ...food,
      name: translatedValues[index * 2] || food.name,
      description: translatedValues[index * 2 + 1] || food.description,
      parsedNutrition: food.parsedNutrition
    })))
  } catch (e) {
    return foods
  }
}

export const addMealToDailyNutrition = async ({
  userId,

```

```

    foodId,
    foodFromApi,
    quantity,
    mealType,
    mealDate,
  }) => {
    const mealDateValue = mealDate ? dateFormatter.toDate(mealDate) :
dateFormatter.getToday()
    let nutritionData = {}
    if (foodId) {
      const food = await foodItemRepo.getFoodItemById(foodId)
      if (!food) throw new AppError("Food item not found", 404)
      const multiplier = quantity / (food.baseQuantity || 100)
      nutritionData = {
        foodId: food.id,
        foodName: food.name,
        unit: food.unit,
        calories: parseFloat((food.calories * multiplier).toFixed(2)),
        totalFats: parseFloat((food.totalFats * multiplier).toFixed(2)),
        totalCarbs: parseFloat((food.totalCarbs * multiplier).toFixed(2)),
        proteins: parseFloat((food.proteins * multiplier).toFixed(2)),
      }
    } else if (foodFromApi) {
      const macros = foodFromApi.parsedNutrition
      if (!macros) throw new AppError("Nutrition data is missing", 400)
      const isWeightBased = macros.unit.toLowerCase() === 'g' ||
macros.unit.toLowerCase() === 'ml'
      const multiplier = isWeightBased
        ? quantity / macros.baseQuantity
        : quantity / (macros.baseQuantity || 1)
    }
  }

```

```

nutritionData = {
  foodId: null,
  externalId: foodFromApi.externalId,
  foodName: foodFromApi.name,
  unit: macros.unit,
  calories: parseFloat((macros.calories * multiplier).toFixed(2)),
  totalFats: parseFloat((macros.totalFats * multiplier).toFixed(2)),
  totalCarbs: parseFloat((macros.totalCarbs * multiplier).toFixed(2)),
  proteins: parseFloat((macros.proteins * multiplier).toFixed(2)),
}
} else {
  throw new AppError("Either foodId or foodFromApi must be provided", 400)
}
return userDailyNutritionRepo.createDailyNutrition({
  userId,
  quantity,
  mealType,
  mealDate: mealDateValue,
  ...nutritionData
}))}

export const calculateDailyNutrition = async (userId, mealDate) => {
  const dailyMeals = await
userDailyNutritionRepo.getDailyNutritionByDate(userId, mealDate)
  const totals = dailyMeals.reduce(
    (acc, meal) => {
      acc.calories += Number(meal.calories)
      acc.proteins += Number(meal.proteins)
      acc.totalFats += Number(meal.totalFats)
      acc.totalCarbs += Number(meal.totalCarbs)
    },
    {
      calories: 0,
      proteins: 0,
      totalFats: 0,
      totalCarbs: 0
    }
  )
  return acc
}

```

```

    },
    { calories: 0, proteins: 0, totalFats: 0, totalCarbs: 0 }
  )
  return {
    totals: {
      calories: Math.round(totals.calories),
      proteins: parseFloat(totals.proteins.toFixed(1)),
      totalFats: parseFloat(totals.totalFats.toFixed(1)),
      totalCarbs: parseFloat(totals.totalCarbs.toFixed(1)),
    },
    meals: dailyMeals.map(meal => ({
      id: meal.id,
      mealType: meal.mealType,
      name: meal.foodName,
      quantity: meal.quantity,
      unit: meal.unit,
      calories: meal.calories,
    })))
  }
}

export const addFoodItem = async (userId, data) => {
  return await foodItemRepo.createFoodItem({
    userId,
    name: data.name,
    baseQuantity: data.baseQuantity,
    unit: data.unit,
    calories: data.calories,
    proteins: data.proteins,
    totalFats: data.totalFats,
    totalCarbs: data.totalCarbs,
  })
}

```

```

    })
  }
export const dailyEnergyBalance = async (userId, date) => {
  const dailyNutrition = await calculateDailyNutrition(userId, date)
  const burned = await
userPhysicalActivitiesService.calculateDailyCaloriesBurned(userId, date)
  return {
    consumed: dailyNutrition.totals.calories,
    burned: burned,
    net: dailyNutrition.totals.calories - burned,
    date: dateFormatter.formatDate(date)
  }
}
export const updateMealQuantity = async (id, quantity) => {
  const currentEntry = await userDailyNutritionRepo.findMealById(id)
  if (!currentEntry) throw new AppError(`Entry with ID ${id} not found`, 404)
  const newQty = Number(quantity)
  const baseFood = currentEntry.food
  const source = baseFood || currentEntry
  const baseQty = Number(baseFood?.baseQuantity || currentEntry.quantity || 100)
  const ratio = newQty / baseQty
  const updatePayload = {
    quantity: newQty,
    calories: Math.round(Number(source.calories) * ratio),
    proteins: Number(source.proteins || 0) * ratio,
    totalFats: Number(source.totalFats || 0) * ratio,
    totalCarbs: Number(source.totalCarbs || 0) * ratio,
  }
  return await userDailyNutritionRepo.updateDailyMealData(id, updatePayload)
}

```

Б.12 Текст файла weightHistory.service.js

```
import prisma from "../prisma.js"
import * as weighHistoryRepo from "../repositories/weightHistory.repository.js"
import * as userInfoRepo from "../repositories/userInfo.repository.js"
import * as dateFormatter from "../utils/dateFormatter.util.js"

export const saveWeight = async ({userId, weightKg, measuredAt}) => {
  const measuredAtDate = measuredAt ? dateFormatter.toDate(measuredAt) :
dateFormatter.getToday()

  return await prisma.$transaction(async (tx) => {
    const enrty = await weighHistoryRepo.upsertWeightEntry({
      userId,
      weightKg,
      measuredAt: measuredAtDate
    }, tx)
    await userInfoRepo.updateCurrentWeightKg({
      userId,
      weightKg
    }, tx)
    return enrty
  })
}

export const monthlyStats = async ({ userId, year, month }) => {
  const { targetYear, targetMonth, startDate, endDate } =
dateFormatter.getMonthBounds(year, month)

  const entries = await weighHistoryRepo.findWeightEntriesByMonth({
    userId,
    startDate,
```

```

        endDate
    })
    return {
        year: targetYear,
        month: targetMonth,
        entries: entries.map(e => ({
            date: dateFormatter.formatDate(e.measuredAt),
            weightKg: Number(e.weightKg),
        })),
    }
}

```

Б.13 Текст файла `weightHistory.controller.js`

```

import * as weighHistoryService from "../services/weightHistory.service.js"
export const saveWeight = async (req, res, next) => {
    try {
        const userId = req.user.id
        const { weightKg, measuredAt } = req.body
        const weightEntry = await weighHistoryService.saveWeight({
            userId,
            weightKg,
            measuredAt
        })
        return res.status(200).json(weightEntry);
    } catch (e) {
        next(e)
    }
}

export const getMonthlyStats = async (req, res, next) => {

```

```

try {
  const userId = req.user.id
  const { year, month } = req.query
  const result = await weighHistoryService.monthlyStats({ userId, year, month
})
  res.status(200).json(result)
} catch (e) {
  next(e)
}
}

```

Б.14 Текст файла weight-history.jsx

```

import React, { useEffect, useState } from 'react'
import { View, Text, ScrollView, ActivityIndicator } from 'react-native'
import { SafeAreaView } from 'react-native-safe-area-context'
import { Ionicons } from "@expo/vector-icons"
import { useTranslation } from 'react-i18next'
import { useTheme } from '../constants/theme.js'
import api from '../services/api'
import i18n from '../i18n/index.js'
import { getDeltaColor, formatDelta } from '../utils/weightHelpers.js'
import { useMonthNavigation } from '../hooks/useMonthNavigation.js'
import MonthHeader from '../components/MonthHeader'
const WeightHistoryScreen = () => {
  const { t } = useTranslation()
  const { selectedDate, changeMonth, isCurrentMonth } = useMonthNavigation()
  const COLORS = useTheme()
  const [loading, setLoading] = useState(true)
  const [entries, setEntries] = useState([])

```

```

const [goal, setGoal] = useState(null)
const fetchData = async () => {
  try {
    setLoading(true)
    const year = selectedDate.getFullYear()
    const month = selectedDate.getMonth() + 1
    const [weightRes, userRes] = await Promise.all([
      api.get('/weight/stats', { params: { year, month } }),
      api.get('/user/me')
    ])
    setEntries(weightRes.data?.entries || [])
    setGoal(userRes.data.userInfo?.goal?.code)
  } catch (e) {
  } finally {
    setLoading(false)
  }
}

useEffect(() => { fetchData() }, [selectedDate])
const filteredEntries = entries.filter(entry => {
  const d = new Date(entry.date)
  return d.getMonth() === selectedDate.getMonth() &&
    d.getFullYear() === selectedDate.getFullYear()
}).reverse()
return (
  <SafeAreaView style={{ flex: 1, backgroundColor: COLORS.white }}
edges={['top']}>
    <View style={{ flex: 1, backgroundColor: COLORS.background }}>

      <MonthHeader
        selectedDate={selectedDate}

```

```

        changeMonth={changeMonth}
      />
      {loading ? <ActivityIndicator style={{marginTop: 20}}
color={COLORS.primary} /> : (
        <ScrollView contentContainerStyle={{padding: 20}}>
          <View style={{marginTop: 25, alignItems: 'center'}}>
            {filteredEntries.length === 0 ? (
              <View style={{marginTop: 40, alignItems: 'center', opacity:
0.5}}>
                <Icons name="bar-chart-outline" size={80}
color={COLORS.textLight} />
                <Text style={{marginTop: 15, fontSize: 16, color:
COLORS.text }}>{t('weightHistory.noData')}</Text>
              </View>
            ) : (
              filteredEntries.map((item, index) => {
                const prevItem = filteredEntries[index + 1]
                const rawDelta = prevItem ? (item.weightKg -
prevItem.weightKg).toFixed(1) : null
                const statusColor = getDeltaColor(rawDelta, goal)
                const displayDelta = formatDelta(rawDelta)

                const currentLocale = i18n.language === 'uk' ? 'uk-UA' :
'en-US';

                return (
                  <View key={item.date} style={{alignItems: 'center',
width: '100%'}}>
                    <View style={{flexDirection: 'row', alignItems:
'center', justifyContent: 'center', width: '100%'}}>

```

```

<View style={{ width: 80, alignItems: 'flex-end',
marginRight: 15 }}>

  <Text style={{ fontWeight: '600', color:
COLORS.text }}>

    {new
Date(item.date).toLocaleDateString(currentLocale, { day: 'numeric', weekday:
'short' })}

  </Text>
</View>
<View style={{
  width: 70, height: 70, borderRadius: 35,
  backgroundColor: index === 0 &&
isCurrentMonth ? COLORS.primary : '#FFF',
  borderWidth: 2, borderColor: COLORS.primary,
  justifyContent: 'center', alignItems: 'center',
  elevation: 3, shadowColor: '#000',
shadowOpacity: 0.1, shadowRadius: 4
  }}>
  <Text style={{
    color: index === 0 && isCurrentMonth ?
'#FFF' : COLORS.primary,
    fontWeight: 'bold', fontSize: 16
  }}>
    {item.weightKg}
  </Text>
  <Text style={{ fontSize: 10, color: index === 0
&& isCurrentMonth ? '#EEE' : COLORS.primary
  }}>{t('common.units.kg')}</Text>
</View>
<View style={{ width: 80, marginLeft: 15 }}>

```

```

        {displayDelta && (
            <View style={{ flexDirection: 'row',
alignItems: 'center' }}>

                <Icons
                    name={parseFloat(rawDelta) < 0 ? "arrow-
down" : "arrow-up"}

                    size={14}
                    color={statusColor}
                />

                <Text style={{ color: statusColor,
fontWeight: '700', marginLeft: 2 }}>

                    {displayDelta} {t('common.units.kg')}
                </Text>
            </View>
        )}
    </View>
</View>
    {index !== filteredEntries.length - 1 && (
        <View style={{
            width: 2, height: 30, backgroundColor:
COLORS.primary,
            opacity: 0.2, marginVertical: 4
        }} />
    )}
</View>
)
})
})
</View>
</ScrollView>

```

```
    })  
  </View>  
</SafeAreaView>  
)  
}  
export default WeightHistoryScreen
```

ДОДАТОК В
Слайди презентації

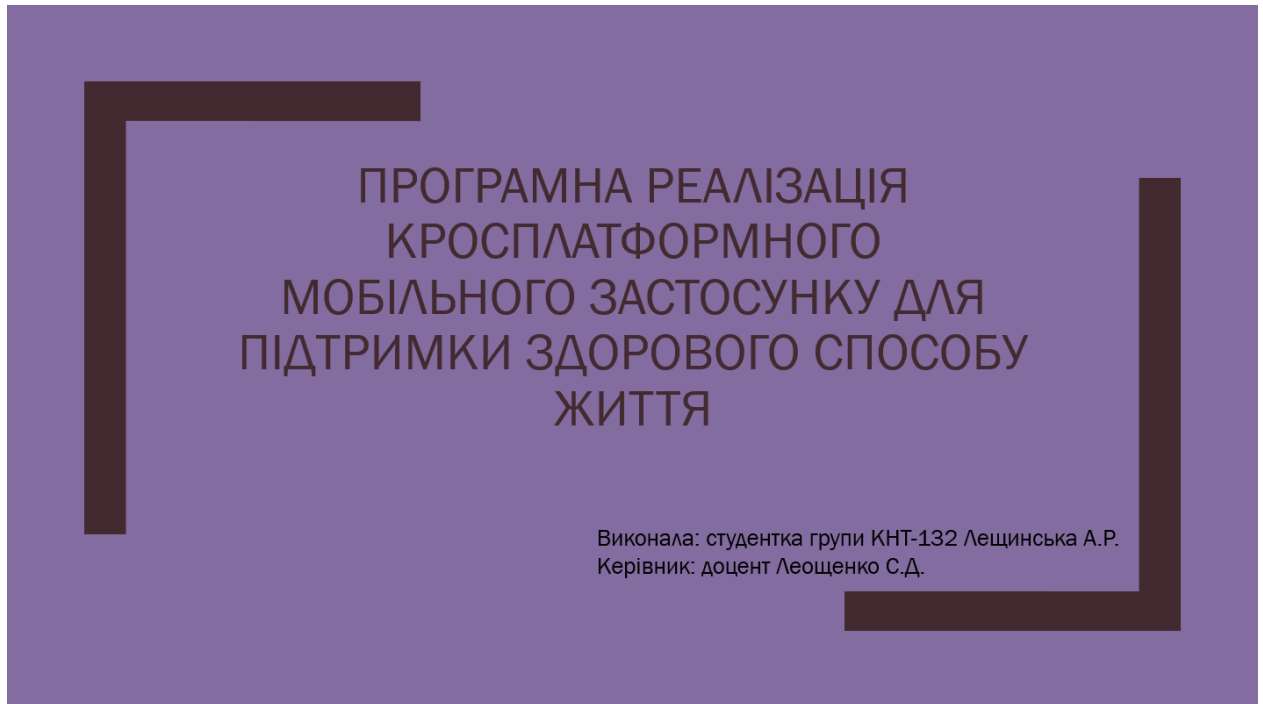


Рисунок В.1 – Слайд 1

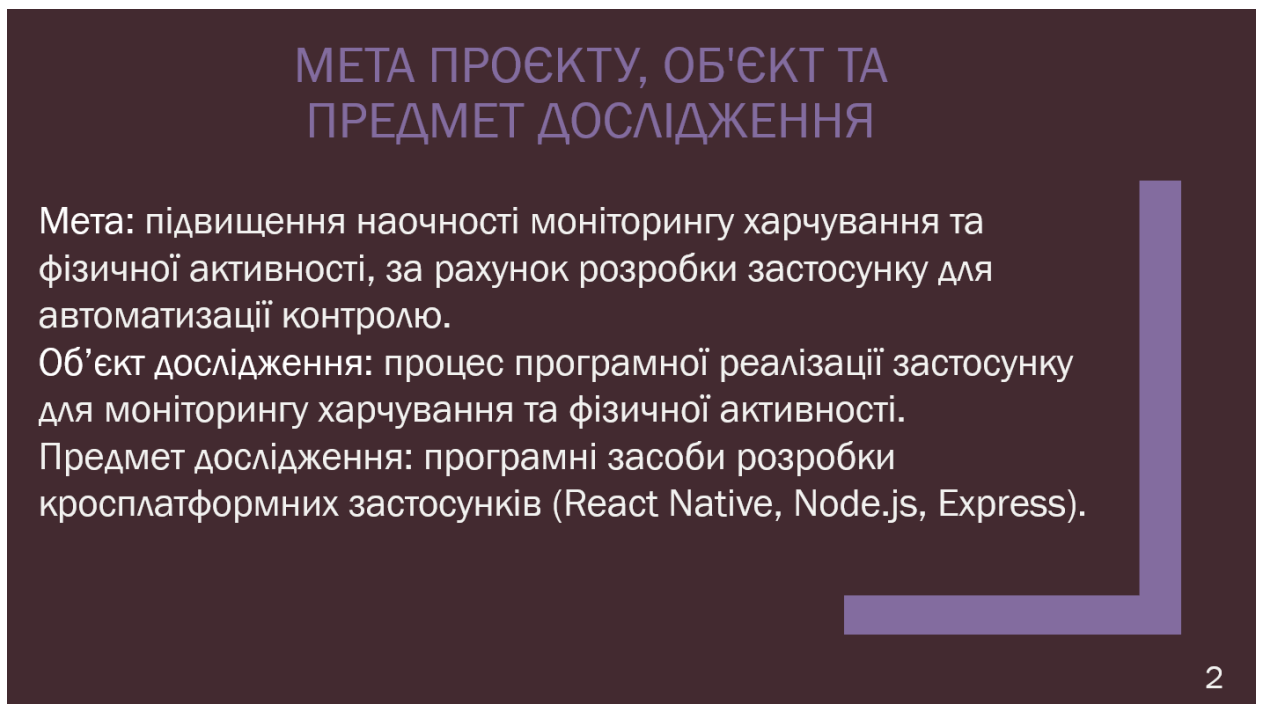


Рисунок В.2 – Слайд 2

АКТУАЛЬНІСТЬ ТЕМИ

Підтримка здорового способу життя вимагає балансу між харчуванням, фізичною активністю та змінами в організмі. Проте ручний облік цих параметрів є занадто трудомістким процесом, що призводить до швидкої втрати мотивації. Саме тому виникає необхідність у створенні єдиного програмного інструменту для автоматизації цього процесу.

3

Рисунок В.3 – Слайд 3

ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

Критерій	MyFitnessPal	FatSecret	Samsung Health
База продуктів	Дуже велика	Велика	Велика
Доступ до статистики БЖВ	Лише за підпискою	Є	Є
Облік фізичної активності	Обмежений(повний доступ лише за підпискою)	Обмежений	Є(автоматичний)
Вартість	Платний функціонал	Безкоштовний	Безкоштовний
Залежність від пристрою	Немає	Немає	Є(для повного функціоналу)
Локалізація	Обмежена	Обмежена	Повна
Кросплатформість	Присутня	Присутня	Відсутня

4

Рисунок В.4 – Слайд 4

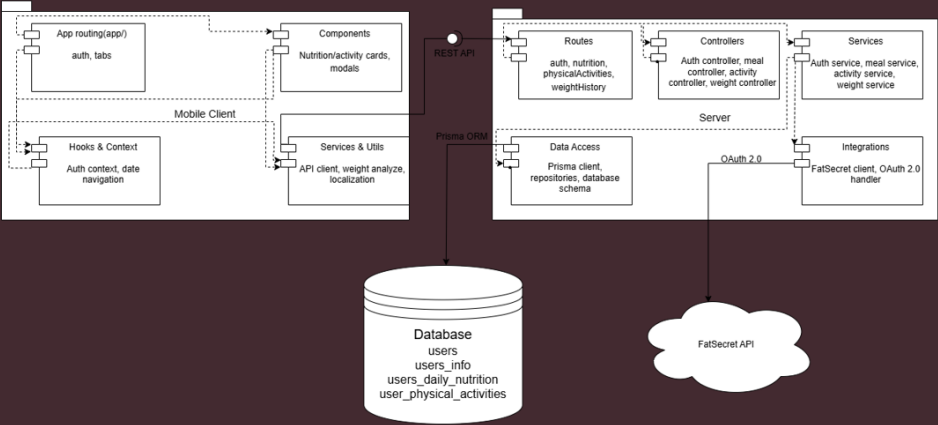
ТЕХНОЛОГІЧНИЙ СТЕК

Технологія	Призначення	Переваги
React Native	Розробка мобільного клієнта	Спільна кодова база для iOS та Android
Expo Router	Маршрутизація та навігація	Файлова маршрутизація, зручна ієрархічна навігація
Node.js + Express	Серверна частина	Ефективна обробка асинхронних запитів
PostgreSQL	Зберігання даних	Надійність, підтримка ACID
Prisma ORM	Робота з БД	Автоматизація міграцій, зручна взаємодія з БД

5

Рисунок В.5 – Слайд 5

АРХІТЕКТУРА ЗАСТОСУНКУ

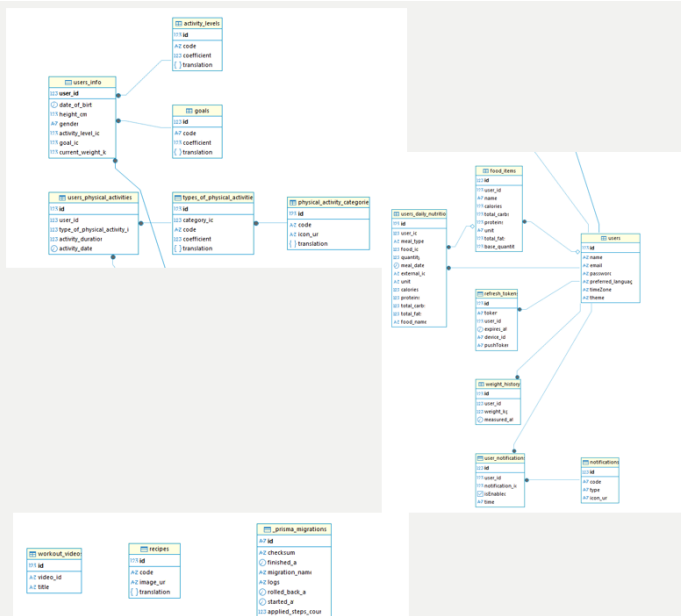


6

Рисунок В.6 – Слайд 6

АРХІТЕКТУРА БАЗИ ДАНИХ

Архітектура бази даних включає понад 10 взаємопов'язаних сутностей, основними з яких є users, userInfo, daily_nutrition та weight_history.

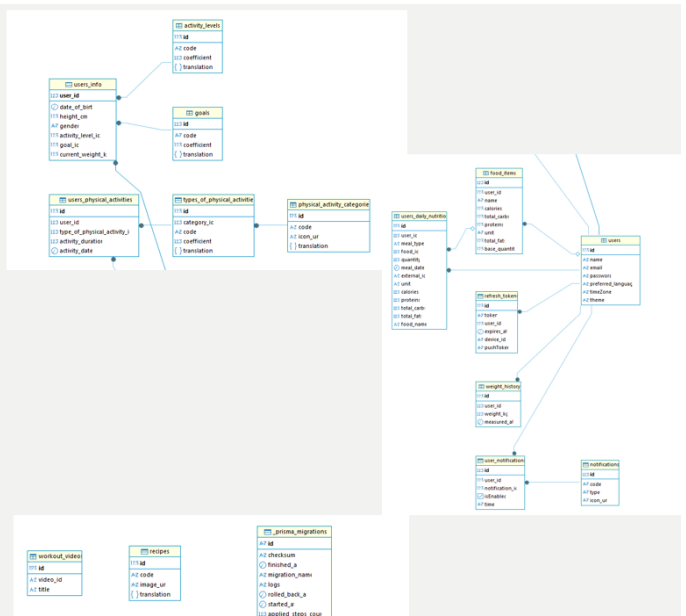


7

Рисунок В.7 – Слайд 7

АРХІТЕКТУРА БАЗИ ДАНИХ

Особливістю реалізації є використання типу даних JSONB для зберігання об'єктів перекладу прямо у таблицях, що дозволило оптимізувати структуру БД та забезпечити гнучку підтримку багатомовного контенту без складних операцій JOIN.



8

Рисунок В.8 – Слайд 8

МАТЕМАТИЧНА МОДЕЛЬ РОЗРАХУНКУ КБЖВ

Функціонування системи базується на формулі Міффіна-Сан Жеора, яка визнана найбільш точною для розрахунку базового метаболізму (BMR). Кінцевий показник добових енерговитрат (TDEE) коригується за допомогою коефіцієнтів фізичної активності (PAL), після чого алгоритм автоматично адаптує норми КБЖВ відповідно до стратегії користувача: схуднення, підтримка або набір ваги.

$$BMR = 10 \times \text{вага(кг)} + 6,25 \times \text{зріст(см)} - 5 \times \text{вік(роки)} - 161.$$

$$BMR = 10 \times \text{вага(кг)} + 6,25 \times \text{зріст(см)} - 5 \times \text{вік(роки)} + 5.$$

$$TDEE = BMR \times PAL,$$

9

Рисунок В.9 – Слайд 9

МАТЕМАТИЧНА МОДЕЛЬ РОЗРАХУНКУ ВИТРАТ ЕНЕРГІЇ ПРИ АКТИВНОСТІ

Для розрахунку спалених калорій під час виконання фізичних вправ використано метаболічний еквівалент (MET). Це дозволяє системі враховувати не лише тривалість тренування, а й специфічну інтенсивність виду діяльності у поєднанні з поточною масою тіла користувача за формулою.

$$E = MET \times m \times t,$$

10

Рисунок В.10 – Слайд 10

ІНТЕГРАЦІЯ ІЗ ЗОВНІШНІМ СЕРВІСОМ FATSECRET API

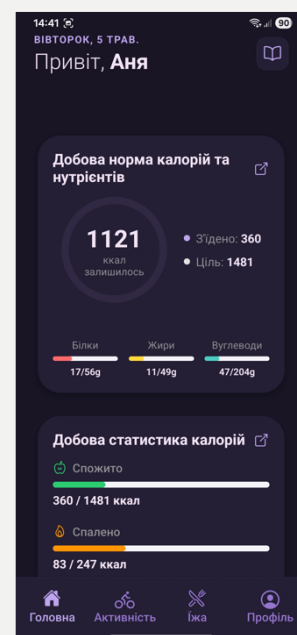
Для розширення функціоналу реалізовано інтеграцію з платформою FatSecret, що відкриває доступ до найбільшої у світі верифікованої бази нутрієнтів. Взаємодія побудована на стандарті безпеки OAuth 2.0, що забезпечує захищений доступ до хмарних даних про продукти без необхідності передачі особистих паролів користувача.

11

Рисунок В.11 – Слайд 11

ГОЛОВНИЙ ЕКРАН ЗАСТОСУНКУ

Головний екран застосунку виступає персоналізованим дашбордом, де візуалізовано добову норму калорій та стан енергетичного балансу. Використання кільцевих діаграм та лінійних індикаторів прогресу дозволяє оцінити залишок нутрієнтів та співвідношення спожитої й витраченої енергії протягом дня.



12

Рисунок В.12 – Слайд 12

ОБЛІК ХАРЧУВАННЯ

Забезпечено повний цикл моніторингу раціону. Пошук продуктів та їх додавання. Реалізована система адаптивних модальних вікон дозволяє точно вказувати вагу порції, після чого застосунок миттєво перераховує кількість білків, жирів та вуглеводів і вносить їх у щоденник.

13

Рисунок В.13 – Слайд 13

ОБЛІК ФІЗ. АКТИВНОСТІ

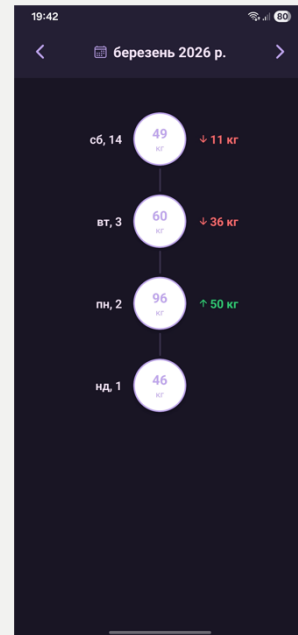
Процес обліку базується на інтуїтивно зрозумілій ієрархії категорій та встановленні тривалості. Застосунок миттєво проводить розрахунок спалених калорій.

14

Рисунок В.14 – Слайд 14

ВІЗУАЛІЗАЦІЯ ЗМІНИ ВАГИ

Застосунок не лише фіксує значення, а й динамічно вираховує та показує різницю з попереднім зважуванням. Різниця має забарвлення: зелене - позитивний прогрес до мети, червоний - відхилення, а жовтий використовується при підтримці ваги.



15

Рисунок В.15 – Слайд 15

ВИСНОВКИ

У результаті виконання дипломної роботи створено кросплатформний застосунок для підтримки здорового способу життя. Розроблений застосунок успішно поєднує функції щоденника харчування, моніторингу активності та аналізу прогресу, пропонуючи користувачам якісний сервіс.

16

Рисунок В.16 – Слайд 16