

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ЗАСТОСУНКУ ДЛЯ ОПТИМІЗАЦІЇ ХАРЧУВАННЯ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
AN APPLICATION FOR NUTRITION OPTIMIZATION

Виконав(ла): студент(ка) 2 курсу, групи КНТ-142м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ШУТКО А.С.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
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
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ШУТКО Анни Сергіївни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація застосунку для оптимізації харчування. Research and Software Implementation of an Application for Nutrition Optimization.

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

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

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір програмного забезпечення та його обґрунтування. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Тестування і результати. 6. Додатки.

5. Перелік графічного матеріалу (з 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
1-6 Основна частина	КАПЛІЄНКО Т. І., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 08 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	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.	3-6 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	7 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	8-10 тиждень	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Анна ШУТКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КАПЛІЄНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
188 с., 6 табл., 108 рис., 2 дод., 89 джерел.

БАЗА ДАНИХ, ОПТИМІЗАЦІЯ ХАРЧУВАННЯ, ANGULAR, NEST.JS, ORM PRISMA, RXJS.

Об'єкт дослідження – програмні засоби для оптимізації харчування.

Предмет дослідження – застосунки для оптимізації харчування.

Мета роботи – створення застосунку для оптимізації харчування широкого кола користувачів.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване, структурне, програмування, текстовий редактор для розробки Visual Studio Code, мова програмування TypeScript (JavaScript), фреймворк для клієнтської частини Angular, бібліотеки ngx-translate, RxJS, Angular material, фреймворк для серверної частини Nest.JS, мова гіпертекстової розмітки HTML5, препроцесор мови каскадного опису стилів SCSS, база даних SQLite, об'єктно-реляційне відображення Prisma, персональний комп'ютер з процесором AMD Ryzen 5 під управлінням операційної системи Microsoft Windows 10.

Результати. Створено застосунок для оптимізації харчування, який дозволяє проводити: підрахунок калорій, ведення активностей, вимірювань параметрів (ваги, обсягу м'язової маси), надання статистики на основі введених даних.

Висновки. Розроблено застосунок для оптимізації харчування широкого кола людей, що вирішує поставлені задачі.

Галузь використання – лікарні, дієтологічні клініки, фітнес центри, спортивні центри та їх відвідувачі, спортсмени та люди, що бажають покращити свою фізичну форму.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 188 pages, 6 tables, 108 figures, 2 appendixes, 89 sources.

DATABASE, OPTIMIZATION OF NUTRITION, ANGULAR, NEST.JS, ORM PRISMA, RXJS

Object of study is a software tools for optimizing nutrition.

Subject of study are applications for optimizing nutrition.

The purpose of the work is to create an application for optimizing the nutrition of a wide range of users.

Materials, methods, and tools: object-oriented and structured programming, text editor for development Visual Studio Code, programming language TypeScript (JavaScript), framework for the client part Angular, libraries ngx-translate, RxJS, Angular material, framework for the server part Nest.JS, hypertext markup language HTML5, preprocessor of the language of cascade description of styles SCSS, SQLite database, object-relational mapping Prisma, personal computer with an AMD Ryzen 5 processor under the control of the Microsoft Windows 10 operating system.

Results. An application is created to optimize nutrition, which allows you to: count calories, conduct activities, measure parameters (weight, muscle mass), provide statistics based on the entered data.

Conclusions An application is developed to optimize the nutrition of a wide range of people, which solves the set tasks.

The field of use is hospitals, nutrition clinics, fitness centers, sports centers, and their visitors, athletes, and people who want to improve their physical condition.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Актуальність проблеми	11
1.2 Поняття підрахунку калорій та підходи до підрахунку	12
1.3 Важливість якості харчування для широкої громадськості та спортсменів	13
1.4 Підрахунок загальної витрати енергії.....	24
1.5 Огляд існуючих аналогів.....	31
1.6 Постановка задач роботи.....	39
2 Вибір програмного забезпечення та його обґрунтування.....	41
2.1 Основні патерни та підходи до розробки вебзастосунків.....	41
2.2 Визнані сучасні технології/протоколи для вебзастосунків	45
2.3 Огляд популярних фреймворків для написання клієнтської сторони вебзастосунку	46
2.4 Огляд популярних фреймворків для написання серверної частини вебзастосунків	52
2.5 Огляд популярних баз даних для серверної частини застосунку	59
2.6 Остаточний вибір стеку технологій та його обґрунтування.....	63
3 Основні рішення щодо реалізації компонентів системи.....	65
3.1 Функціональні можливості програми	65
3.2 Алгоритми розробленої програми.....	67
3.3 Навігація між сторінками застосунку	69
3.4 Архітектура залежностей програми.....	70
3.5 Файлова архітектура програми.....	75
3.6 Основні моделі бази даних.....	86
3.7 API контролери.....	88

3.8 Методи ідентифікації, аутентифікації, авторизації користувачів у програмі.....	94
4 Керівництво програміста.....	96
4.1 Призначення й умови застосування гри	96
4.2 Характеристики застосунку	97
4.3 Звертання до застосунку.....	100
4.4 Використання застосунку.....	102
5 Тестування і результати.....	115
5.1 Опис методики проведення тестування.....	115
5.2 Результати використання розробленого застосунка	115
5.3 Оптимізація розробленого застосунка	136
Висновки	145
Перелік джерел посилання	147
Додаток А Текст програми.....	155
Додаток Б Слайди презентації	176

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BCAA	– Branched-Chain Amino Acids – Гілковані амінокислоти;
BEE	– Basal Energy Expenditure – Базова енергетична витрата;
BMR	– Basal Metabolic Rate – Базовий метаболічний рівень;
CBPA	– Calories Burned from Physical Activity – Кількість калорій, які людина витрачає на рухову активність;
CSS	– Cascading Style Sheets – Каскадні таблиці стилів;
DTO	– Data Transfer Object – Об'єкт передачі даних;
DOM	– Document Object Model – Об'єктна модель документу;
GraphQL	– Graph Query Language – Мова запитів графічних структур даних;
HTML	– Hypertext Markup Language – Мова гіпертекстової розмітки;
HTTP	– Hypertext Transfer Protocol – Протокол передачі гіпертексту;
ISSN	– International Society of Sports Nutrition – Міжнародне товариство спортивного харчування;
MET	– Metabolic Equivalent of Task – Метаболічний еквівалент завдання;
MVC	– Model-View-Controller – Модель-Вид-Контролер;
MVVM	– Model-View-ViewModel – Модель-Вид-Модель представлення;
Nest.js	– Фреймворк для побудови ефективних, масштабованих серверних додатків на Node.js;
Node.js	– Платформа для виконання JavaScript поза браузером;
ORM	– Object-Relational Mapping – Об'єктно-реляційне відображення;
PAL	– Physical Activity Level – Рівень фізичної активності;
Pipes	– Елементи Angular для перетворення даних;

Prisma	– ORM для Node.js і TypeScript;
RDA	– Resource Description and Access – Опис ресурсів та доступ;
REST API	– Representational State Transfer Application Programming Interface – Інтерфейс програмування додатків з передачею стану представлення;
RMR	– Resting Metabolic Rate – Метаболічний рівень у стані спокою;
RxJS	– Reactive Extensions for JavaScript – Реактивні розширення для JavaScript;
SEO	– Search Engine Optimization – Оптимізація для пошукових систем;
SPA	– Single Page Application – Односторінкова програма;
SSR	– Server-Side Rendering – Рендеринг на стороні сервера;
SQLite	– Вбудована реляційна база даних;
TDEE	– Total Daily Energy Expenditure – Загальна щоденна енергетична витрата;
TWEE	– Total Weekly Energy Expenditure – Загальна щотижнева енергетична витрата;
TEF	– Thermic Effect of Food – Термічний ефект їжі;
UI	– User Interface – Користувацький інтерфейс;
ВООЗ	– World Health Organization (WHO) – Всесвітня Організація Охорони Здоров'я;
СКБД	– Система керування базами даних.

ВСТУП

Сучасний світ відзначається стрімким ростом інтересу до здорового способу життя та збалансованого харчування. В умовах загального розцвіту глобальної інформаційної епохи здорове харчування стає однією з ключових складових забезпечення активного та продуктивного життя. Водночас, зростає інтерес до використання інформаційних технологій для підвищення ефективності та якості харчування.

Харчування є важливою складовою здорового способу життя. Відомо, що раціональне харчування сприяє підтриманню нормальної ваги, запобіганню хронічних захворювань, покращенню психічного здоров'я та якості життя. Однак багато людей зіткнулися з проблемами при виборі оптимального раціону харчування, який би враховував їх індивідуальні потреби, уподобання, обмеження та цілі.

З огляду на це, актуальним є розробка застосунку для оптимізації харчування, який би допомагав користувачам планувати своє харчування на основі наукових рекомендацій, аналізу даних про фізичну активність. Такий застосунок мав би мати наступні функціональні можливості:

- аналіз даних про користувача;
- розрахунок індивідуальної норми калорійності та споживання макро- та мікронутрієнтів;
- представлення раціону харчування у вигляді списку продуктів та страв з вказанням калорійності та нутритивної цінності;
- моніторинг дотримання раціону харчування та надання статистики.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Харчування як один з ключових елементів життєдіяльності заслуговує на велику увагу. Сидячий спосіб життя, низький рівень фізичної активності та зміни в харчуванні, зокрема високе споживання рафінованих вуглеводів та оброблених продуктів, а також зміни у типі та обсязі споживання жирів призвели до збільшення в Україні поширеності надмірної ваги та ожиріння протягом останнього десятиліття. Згідно з інформацією Державної служби статистики на 2018 рік, 42.4% населення мало нормальну масу тіла, тоді як 39.9% мали надлишкову вагу, а ожиріння відзначалося у 16% чоловіків і жінок [1].

Для підтримки здоров'я людини важливим фактором харчування є не його кількість, а якість, пропорції та заходи. Надмірна вага збільшує ризик багатьох проблем зі здоров'ям, таких як діабет 2 типу, високий кров'яний тиск, хвороби серця, інсульт, проблеми із суглобами, захворювання печінки, каміння у жовчному міхурі, деякі види раку, проблеми зі сном та диханням тощо [2].

Основними причинами надмірної ваги є: генетика; звички сну; медичні умови чи ліки; малорухливий спосіб життя; незбалансованість харчування; надмірне споживання їжі; незнання основ правильного харчування; складність відстеження харчування.

Частину з цих проблем можна вирішити за допомогою застосунків, які виконують функцію контролю калорійності продуктів та складання статистики. Однак кожен з них має свої недоліки, більшості з яких буде позбавлено в майбутньому застосунку для оптимізації харчування.

1.2 Поняття підрахунку калорій та підходи до підрахунку

Підрахунок калорій – це процес документування споживаних калорій. Іншими словами, підрахунок калорій, які з'їдаються з різними прийомами їжі.

Є два підходи підрахунку калорій: низькотехнологічний та високотехнологічний.

Низькотехнологічний підхід – запис інформації про калорійність з етикеток товарів, що використовуються, та знаходження інформації про калорійність немаркованих товарів у спеціалізованих ресурсах [3]. Для цього можна використовувати щоденник харчування, в який запис робиться вручну. Це вимагає багато часу та зусиль, тому що людині доведеться все розрахувати самостійно. Але він також набагато більше думає про їжу, яку споживає.

Високотехнологічний підхід до підрахунку калорій – використання застосунків, які дозволяють швидко знайти продукти харчування з бази даних, щоб давати точні результати та додати їх до щоденника харчування, який автоматично враховує інформацію про калорійність, а іноді також про грами та відсотки вуглеводів, жирів та білків. Таким чином, потрібно просто ввести різні страви, продукти та дозволити застосунку підрахувати, скільки калорій користувач споживає щодня.

Підрахунок калорій працює тільки в тому випадку, якщо людина дійсно чесна із самою собою. Потрібно записувати все, що споживається. У тому числі напої, адже за винятком води та несолодкого чаю, кожен напій, яким би невеликим він не був, також містить калорії. Тому це єдиний спосіб зробити підрахунок калорій свідомим [4].

Дуже небагато людей, навіть серед тих, кому вдалося позбутися зайвої ваги та зберегти її, підраховують кожен калорію, яку вони споживають щодня. У цьому просто немає потреби. Більшість людей інтуїтивно підтримують звички в їжі. Як тільки у людини з'являються певні харчові звички, їй буде легко залишатися близьким до оптимальної ваги, просто дотримуючись цих звичок. Підрахунок калорій може допомогти їх визначити. Звички будуть

складатися з певних видів їжі, які людина споживає регулярно, рідко чи взагалі не споживає; певних розмірів порцій; графіка приймання їжі та перекусів. Такі звички дуже легко відтворювати день у день без підрахунку калорій.

Підрахунок калорій може допомогти визначити, як слід харчуватися, скільки і з якою частотою, щоб досягти своєї бажаної ваги. Але продовжуючи практикувати харчові звички, які прищепив підрахунок калорій, більше не треба буде рахувати калорії, і людина зможе підтримувати свою вагу інтуїтивно [3].

1.3 Важливість якості харчування для широкої громадськості та спортсменів

Харчування є одним з основних факторів, які впливають на здоров'я, самопочуття, продуктивність та довголіття людини. Якість харчування залежить від складу, кількості, розподілу їжі, а також від її взаємодії з іншими факторами, такими як фізична активність, стрес, генетика, захворювання тощо. Якість харчування має особливе значення для спортсменів, оскільки вони піддають свій організм підвищеним фізичним навантаженням, які потребують адекватного забезпечення енергією та поживними речовинами.

За даними Національного обстеження стану здоров'я та харчування (NHANES) за 2017-2018 роки 30.7% людей мають надмірну вагу, 42.4% людей мають ожиріння (включаючи тяжке ожиріння), 9.2% людей мають тяжке ожиріння і лише 17.7% людей мають нормальну вагу [2]. Як надмірна вага та ожиріння, так і недостатня вага можуть бути основними факторами ризику розвитку різних захворювань. Тому попередження та корекція порушень маси та складу тіла є важливим завданням для широкої громадськості.

Для спортсменів маса та склад тіла мають не лише медичне, а й спортивне значення. Вони впливають на функціональний стан організму, фізичну працездатність та естетичний вигляд спортсмена [5]. Залежно від виду спорту та специфіки змагальної діяльності існують різні оптимальні

показники маси та складу тіла. Наприклад, для легкоатлетів-бігунів на короткій дистанції необхідна низька маса тіла і м'язова маса з переважанням швидких м'язових волокон. Для важкоатлетів і борців необхідна висока м'язова маса з переважанням повільних м'язових волокон. Для гімнастів і фігуристів необхідна низька маса тіла і низький вміст жиру [5]. Тому спортсменам необхідно контролювати свою масу та склад тіла та коригувати їх за допомогою харчування та тренувань.

Харчування для спортсменів повинно враховувати їх індивідуальні особливості, цілі, вид спорту, рівень тренуваності, стадію підготовки, сезонність, кліматичні умови тощо. Загальними принципами харчування для спортсменів є:

- забезпечення достатнього енергетичного балансу для покриття витрат енергії під час тренувань та змагань, а також для підтримки росту та розвитку організму;
- забезпечення оптимального споживання макро- та мікронутрієнтів, води та інших поживних речовин, які необхідні для функціонування організму, синтезу м'язової тканини, відновлення після навантажень, захисту від окисного стресу тощо;
- забезпечення раціонального розподілу їжі протягом дня з урахуванням часу тренувань та змагань, а також режиму сну та відпочинку;
- забезпечення адекватної гідратації організму перед, під час і після навантажень для поповнення втрати рідини і електролітів через потовиділення, а також для регуляції температури тіла і кровообігу;
- забезпечення безпечності харчування шляхом уникнення або обмеження споживання продуктів і добавок, які можуть мати шкідливий вплив на здоров'я [6], [7].

1.3.1 Оптимальне співвідношення макронутрієнтів

Дієтичні рекомендації для американців на 2020-2025 роки [8] припускають, що оптимальні співвідношення макронутрієнтів для дорослих наступні: вуглеводи – 45-65% калорій, білок – 10-35% калорій, жири – 20-35% калорій.

Міжнародна асоціація спортивних наук (ISSA) наголошує, що люди можуть коригувати ці співвідношення залежно від мети та фізичної активності.

Наприклад, спортсмен, який займається тренуваннями на витривалість, збільшить кількість вуглеводів, що споживаються, а спортсмен, який займається силовими видами спорту, збільшить споживання білка.

За даними Міжнародного товариства спортивного харчування (ISSN) за 2018 рік [9], було виділено співвідношення макронутрієнтів для спортсменів [10].

1.3.2 Роль білків

Білок відіграє ключову роль у харчуванні для спортсменів, оскільки він забезпечує організм необхідною кількістю амінокислот, що сприяють процесам будівництва та відновлення м'язів та тканин.

Спортсмени, що займаються інтенсивними тренуваннями, можуть отримати користь від вживання у своєму раціоні більш ніж удвічі більшої добової кількості білка (RDA), що рекомендується.

Наприклад, зазвичай рекомендована кількість білка для дорослих жінок становить 46 г, а для дорослих чоловіків – 56 г. З цього можна зробити висновок, що для спортсменів корисною може бути збільшена кількість споживаного білка, наприклад, близько 92 г для жінок та 112 г для чоловіків.

ISSA вказує на те, що для багатьох спортсменів безпечно споживати 2 г білка на 1 кг маси тіла щоденно, що перевищує рекомендовану добову норму

в 0,8 г/кг. ISSN, також вказує на можливість оптимального вживання білка від 1,2 до 2,0 г/кг маси тіла на день.

Збільшена кількість споживаного білка може допомогти спортсменам уникнути катаболізму білка та сприяти швидшому відновленню. Як відзначає ISSN, це може в подальшому запобігти травмам та атрофії м'язів. При помірних обсягах інтенсивних тренувань спортсмену слід споживати 1,2-2 г білка на 1 кг маси тіла, що з спортсмена вагою 50-150 кг становить 60-300 г білка щодня. Для інтенсивних тренувань із великим об'ємом ISSN рекомендує вживати від 1,7 до 2,2 г білка на 1 кг маси тіла щоденно, що відповідає діапазону від 85 до 330 г білка для спортсмена з масою тіла від 50 до 150 кг [10]. Є нові дані, що свідчать про те, що більш високе споживання білка ($>3,0$ г/кг/день) може позитивно впливати на склад тіла у людей, які тренуються з обтяженнями (тобто сприяти втраті жирової маси) [11].

Здорові джерела білка включають: нежирне м'ясо, птицю, рибу і молочні продукти, морепродукти і яйця, квасолю і сочевицю, горіхи і насіння, сою, включаючи тофу і темп [10].

1.3.3 Роль жирів

Жир є ще одним важливим джерелом калорій. У невеликій кількості жир є ключовим джерелом енергії. Він виконує й інші функції, наприклад, підтримує здоров'я шкіри та волосся. Якщо замінити вуглеводи у своєму раціоні жирами, то це може уповільнити продуктивність, тому що більша частина енергії витрачається на перетравлення жирів для отримання енергії з них.

Жири повинні становити трохи більше 35% щоденних калорій. По можливості вибирають ненасичені жири, такі як оливкова олія та горіхи. Вони корисніші для здоров'я, ніж насичені та трансжири.

Занадто багато жирів або насичені жири та трансжири можуть спричинити проблеми зі здоров'ям. Вони можуть підвищити рівень поганого

холестерину (ЛПНЩ) та збільшити ризик серцево-судинних захворювань та діабету 2 типу [12].

Насичені жири необхідні для нашого організму, але тільки в малих дозах. Вони беруть участь в синтезі гормонів, побудові клітинних мембран і засвоєнню вітамінів [13]. Але їх надлишок може призвести до ожиріння, діабету, серцево-судинних хвороб і підвищення рівня холестерину [14]. Добова норма споживання насичених жирів – 15-20 грам на добу. Частка насичених жирів у добовій калорійності раціону повинна бути лише 5-7% [13].

Жири є необхідними в раціоні для підтримки різних процесів організму, таких як метаболізм гормонів та функція нейротрансмітерів. Додавання до раціону корисних жирів також сприяє насиченню і може бути концентрованим джерелом енергії, особливо для спортсменів з високими енергетичними потребами.

ISSN рекомендує, щоб спортсмени отримували помірну кількість жирів, що становить приблизно 30% щоденних калорій. Проте, для задоволення потреб у більш інтенсивних тренуваннях вони можуть безпечно збільшити вміст жирів до 50% щоденних калорій. Ті, хто має намір зменшити кількість жирів у своєму організмі, можуть обмежити споживання жирів до 20% калорій на день.

Деякі спортсмени можуть вибрати кетогенну дієту та споживати більшу кількість жирів. Підходить для людей із цукровим діабетом, оскільки дозволить зберегти низький рівень інсуліну. Однак огляд ISSN показує, що немає достатніх доказів, які б підтверджували ефективність дієти для здорових спортсменів.

Корисні джерела жирів включають оливкову олію, жирну рибу, авокадо, горіхи та насіння [10].

1.3.4 Роль вуглеводів

Вуглеводи – найбільше джерело калорій в організмі. Прості вуглеводи (фрукти, молоко та овочі) легше засвоюються організмом. Вони забезпечують швидкий приплив енергії. Складні вуглеводи перетравлюються організмом довше. Вони є найкращим тривалим джерелом енергії. Складні вуглеводи в цільнозернових продуктах є поживними. Прикладами складних вуглеводів можна назвати цільнозерновий хліб, картопля, коричневий рис, вівсянку та квасолю [12].

Велика увага приділяється вуглеводам у спортивному харчуванні через їх важливу роль у досягненні спортивних результатів. Для багатьох спортсменів вуглеводи є оптимальним джерелом палива, особливо під час високоінтенсивних та тривалих тренувань. Це пояснюється їхньою здатністю постачати достатню кількість глікогену та глюкози в крові для задоволення потреб у фізичних вправах.

Для підтримки запасів глікогену в печінці та м'язах спортсменам потрібна різна кількість вуглеводів залежно від обсягу тренувань. Для помірних обсягів інтенсивних тренувань, що визначаються як 2-3 години на день інтенсивних вправ, що виконуються 5-6 разів на тиждень, ISSN рекомендує споживати 5-8 грамів на кілограм (г/кг) маси тіла, або 250-1200 г вуглеводів на день для спортсменів вагою 50-150 кг. Для інтенсивних тренувань з великим об'ємом, що визначаються як інтенсивні тренування по 3-6 годин на день при 1-2 щоденних тренуваннях 5-6 днів на тиждень, ISSN рекомендує 8-10 г/кг маси тіла або 400-1500 г вуглеводів на день для спортсменів вагою 50-150 кг.

Наприклад, спортсмен вагою 150 кг, який виконує інтенсивні тренування з великим обсягом, споживатиме приблизно 1200-1500 г вуглеводів.

Корисні вуглеводи для дієти спортсмена можуть містити цільнозернові продукти, такі як коричневий рис, кіноа, овес та макарони, а також крохмалисті овочі, такі як картопля [10].

1.3.5 Роль мікронутрієнтів та харчових добавок

Паливо для функціонування організму береться з речовин, що споживаються людиною – макронутрієнтів, а засвоюється за допомогою мікронутрієнтів.

Мікронутрієнти – це вітаміни (С, В1, В2, Е та інші) та мінеральні речовини, такі як кальцій, фосфор, магній, калій та інші. Вітаміни, у свою чергу, діляться на водорозчинні та жиророзчинні [15].

У таблиці 1.1 наведено ролі деяких мікронутрієнтів у внутрішній роботі організму.

Таблиця 1.1 – Роль окремих мікронутрієнтів в організмі

Мікронутрієнт			Роль
1	2	3	4
Вітаміни	Водорозчинні	В1(тіамін)	Бере участь у метаболізмі білків та вуглеводів.
		В2 (рибофлавін)	Відіграє важливу роль в енергетичному метаболізмі, зорі, необхідному для утворення еритроцитів та антитіл.
		В3 (ніацин)	Бере участь у процесі одержання енергії з макронутрієнтів, утворення ферментів.
		В6 (піридоксин)	Необхідний для глюконеогенезу та гліколізу (процеси, що здійснюють повний цикл виробництва енергії з неуглеводних сполук), бере участь у жировому та білковому обміні, утворенні еритроцитів.
		В9 (фолієва кислота)	Необхідний для утворення червоних кров'яних клітин та реплікації ДНК.
		В12 (кобаламін)	Потрібний для утворення червоних кров'яних клітин.
		С	Антиоксидант (захищає клітинні мембрани від окислення), необхідний для нормального функціонування сполучної та кісткової тканин.

Продовження таблиці 1.1

1	2	3	4
Вітаміни	Жиророзчинні		Впливає на зір, бере участь у імунній функції, є структурним компонентом клітинних мембран.
		E	Антиоксидант та імуномодулятор.
		D	Регулює метаболізм та адсорбцію кальцію та фосфору, відіграє ключову роль у здоров'ї кісток, імунної функції.
Мінерали		Залізо	Відіграє критичну роль в енергетичних процесах, кровотворенні, реплікації ДНК, імунної функції.
		Кальцій	Важливий для зростання, підтримки щільності кісткової тканини, регуляції м'язових скорочень, провідності нервів та згортання крові.
		Магній	Виступає фізіологічним регулятором нервово-м'язової, серцево-судинної, імунної та гормональної функцій.
		Цинк	Бере участь в імунній та репродуктивній функціях, загоєнні ран, розвитку скелета, шлунково-кишкової функції.
		Хром	Регулятор рівня глюкози бере участь у метаболізмі жирів та білків.

Мікронутрієнти дуже важливі для організму, їхній недолік або надлишок завжди має негативні наслідки.

За недостатнього надходження мікронутрієнтів запаси організму поступово витрачаються. Наслідком такого способу життя стає дефіцит, який, залежно від рівня, може спричинити різні несприятливі наслідки та призупинити тренувальний процес [16].

Наприклад, при дефіциті заліза послаблюється м'язова функція, при дефіциті магнію збільшується кількість кисню, необхідного для субмаксимального навантаження, зменшується витривалість.

Дефіцит мінералів, як правило, позначається одночасно на різних функціях організму (імунної, гормональної, серцево-судинної, нервово-м'язової тощо). У зв'язку з цим споживання мінералів відіграє важливу роль у

харчуванні як спортсменів, так і людей з низьким або середнім рівнем фізичної активності.

У групі високого ризику дефіциту мінералів (особливо заліза) перебувають, насамперед, спортсмени жіночої статі. Найбільш уразливими з них є підлітки, що пов'язано з початком менструального циклу та стрибкоподібним зростанням, характерним для даного періоду [17].

Спортсмени-вегетаріанці жіночої статі також перебувають у групі високого ризику щодо дефіциту заліза у зв'язку з обмеженням списку залізовмісних продуктів, а також присутністю в раціоні субстанцій, що перешкоджають його адсорбції, таких як фітинова кислота, поліфеноли, кальцій, фосфатні солі [17].

Дефіцит мікронутрієнтів впливає також на імунну функцію. Імунна система функціонує постійно, проте підвищене навантаження на неї припадає на час присутності в організмі патогена. Для «боротьби» з патогеном необхідна додаткова кількість енергії, яку можна взяти із зовнішніх джерел чи внутрішніх резервів організму [18].

Крім енергії, організму потрібні нуклеотиди, амінокислоти, жири та інші речовини. У синтезі нуклеотидів беруть участь такі мікронутрієнти, як залізо, цинк, магній, фолієва кислота. Вітаміни А і D, своєю чергою, є регуляторами експресії генів і відіграють ключову роль розвитку імунних клітин.

За відсутності патогенів дефіцит мікронутрієнтів пригнічує імунну функцію та створює можливість для проникнення інфекцій.

Дефіцит вітаміну А погіршує бар'єрні функції, змінює імунні відповіді та збільшує сприйнятливність організму до низки інфекцій. Нестача вітаміну D підвищує ризик інфекцій дихальних шляхів. При нестачі вітаміну Е зменшується антиоксидантна дія на вільні радикали та окислені ліпіди, які пригнічують імунітет. Дефіцит цинку негативно впливає на синтез ДНК. Нестача заліза спричиняє атрофію тимусу (залізи, у якій відбувається дозрівання, поділ і «навчання» імунних клітин) [18].

Одним із рішень проблеми дефіциту є штучне додавання мікронутрієнтів до раціону у формі біологічно активних добавок.

Біологічно активні добавки (БАД) – композиції біологічно активних речовин, які призначені для безпосереднього вживання разом із їжею, або введення до складу харчових продуктів [19].

Фортифікація їжі – збагачення їжі поживними речовинами. Для досягнення мети (усунення дефіциту мікронутрієнтів) продукт, що фортифікується, повинен бути широко поширений і вживатися значною часткою населення. Одним з ефективних рішень є фортифікація борошна, що використовується у промисловому виробництві хлібобулочних та макаронних виробів [19].

Важливо знати, що використання добавок мікронутрієнтів в їжу може викликати і зворотний ефект (гіпервітаміноз), реакція у відповідь на який залежатиме від конкретної речовини.

Нерідко люди помилково вважають, що якщо невелика кількість вітаміну здатна творити чудеса, то більше викликає пропорційно більший позитивний ефект. При цьому дослідження не тільки не виявили такого зв'язку, але й показали зворотні результати.

Наприклад, надлишок вітаміну А викликає шкірні висипання, нудоту, блювання, розлад опорно-рухового апарату, печінки та інші небажані наслідки. Залізо є токсичним елементом: ознаками гемохроматозу (надлишку заліза) є сильна втома, біль у суглобах, збільшення розмірів печінки та артрит.

На думку експертів та експертних установ, всі необхідні для здоров'я та найбезпечніші дози вітамінів можна отримати з продуктів харчування [16].

Спортсмени повинні впевнитися, що вони отримують необхідні вітаміни та мінерали для підтримки загального стану здоров'я і досягнення спортивних результатів. Зазвичай це досягається через раціональне харчування, що включає в себе різноманітні та збалансовані продукти.

Деякі спортсмени можуть вирішити приймати вітамінні або мінеральні добавки, а також ергогенні засоби, такі як креатин.

Існує недостатньо доказів, які підтверджують ефективність або безпеку багатьох харчових добавок, таких як аденозин-5'-трифосфат, ВСАА, фосфатидна кислота, глютамін, аргінін. Проте деякі дослідження підтверджують безпеку та ефективність інших ергогенних засобів, таких як кофеїн та моногідрат креатину, для використання спортсменами [10].

1.3.6 Роль правильної гідратації

Вода є невід'ємною складовою життя і здоров'я людини. Вона входить до складу всіх клітин, тканин і органів організму, бере участь у всіх фізіологічних процесах, регулює температуру тіла, забезпечує транспорт харчових речовин, виводить зайві продукти обміну речовин [20].

Недостатнє споживання води може призвести до дегідратації, яка проявляється у втраті маси тіла, сухості слизових оболонок, спраги, зниженні артеріального тиску, слабкості, запамороченні, головному болю, порушенні роботи серця, нирок і печінки. Дегідратація також негативно впливає на психоемоційний стан, пам'ять, увагу, навчальну та робочу продуктивність [21].

Норма споживання води залежить від багатьох факторів, таких як вік, стать, фізична активність, кліматичні умови, стан здоров'я тощо. Загальна рекомендація ВООЗ для дорослих становить 2-2,5 л води на добу. Однак це середнє значення, яке може змінюватися в залежності від індивідуальних потреб [22].

Для людей похилого і старечого віку норма споживання води зменшується через зниження функції нирок, зменшення маси тіла і зниження обміну речовин. Однак це не означає, що вони можуть пити менше води, ніж їм потрібно. Недостатнє споживання води може призвести до згущення крові, утворення каменів у нирках, запорів, сухості шкіри і слизових оболонок, зниження імунітету. Норма споживання води для людей похилого віку становить приблизно 30 мл на кг маси тіла [23].

Для спортсменів і людей, які займаються фізичною активністю, норма споживання води збільшується через підвищений розхід енергії і втрату вологи через потовиділення. Вода необхідна для поповнення рідини в організмі, покращення кровообігу, транспортування кисню і поживних речовин до м'язів, попередження перегрівання та зневоднення. Норма споживання води для спортсменів і людей, які займаються фізичною активністю, складає приблизно 40 мл на кг маси тіла. Крім того, необхідно пити воду перед, під час і після тренувань або змагань [24].

1.4 Підрахунок загальної витрати енергії

Загальна витрата енергії – це кількість енергії, яку організм людини витрачає протягом доби на підтримку життєвих функцій, фізичну активність та перетравлення їжі. Цей показник залежить від багатьох факторів, таких як вік, стать, зріст, вага, рівень фізичної активності, харчування та здоров'я. Знання своєї загальної витрати енергії може допомогти людині контролювати свою вагу, фізичну форму, планувати своє харчування та покращувати своє самопочуття [25].

1.4.1 Підрахунок загальної щоденної витрати енергії

Загальна щоденна витрати енергії (TDEE) – це кількість калорій, які людина споживає і витрачає протягом одного дня. Цей показник складається з трьох компонентів: базального метаболізму (BMR), термогенезу харчових продуктів (TEF) та фактичної фізичної активності (PA).

BMR – це кількість калорій, які людина витрачає на підтримку життєвих функцій організму, таких як дихання, кровообіг, температуру та інше. BMR займає близько 60-70% загальної витрати енергії і залежить від ваги, зросту, статі та віку.

TEF – це кількість калорій, які людина витрачає на перетравлення, всмоктування та збереження харчових продуктів. TEF займає близько 10-15% загальної витрати енергії і залежить від складу та частоти харчування.

СВРА – це кількість калорій, які людина витрачає на рухову активність, таку як ходьба, біг, спорт, робота та інше. СВРА займає близько 20-30% загальної витрати енергії і залежить від типу, тривалості та інтенсивності фізичної активності [26].

Для підрахунку загальної щоденної витрати енергії потрібно помножити BMR на коефіцієнт фізичної активності (PAL), який залежить від того, наскільки часто і інтенсивно людина займається спортом або фізичною працею [27]. Знайти TDEE можна за формулою (1.1):

$$TDEE = BMR \cdot PAL. \quad (1.1)$$

Також можна враховувати TEF, що змінить формулу до такого виду (1.2):

$$TDEE = BMR \cdot PAL + TEF, \quad (1.2)$$

де TEF розраховується за формулою (1.3):

$$TEF = BMR \cdot 0.10. \quad (1.3)$$

Щоб визначити загальну щоденну потребу калорій, BMR необхідно помножити на відповідний коефіцієнт активності PAL, який можна обрати за таблицею 1.2.

Таблиця 1.2 – Коефіцієнти фізичної активності PAL

PAL	Опис
1,200	малорухливий спосіб життя (мало фізичних вправ або їх повна відсутність)
1,375	легка активність (легкі фізичні вправи/спорт 1–3 дні на тиждень, близько 590 калорій на день)
1,550	помірно активний (помірні фізичні навантаження/заняття спортом 3–5 днів на тиждень, близько 870 калорій на день)
1,725	дуже активний (важкі вправи/спорт 6–7 днів на тиждень, близько 1150 калорій на день)
1,900	надактивний (дуже важкі вправи/спорт та фізична робота, близько 1580 калорій/день)

Людина вагою 68 кг, що йде зі швидкістю 4 милі на годину, споживає близько 300 калорій на годину (5 ккал/хв). Чинник активності «легко активний» відповідає ходьбі 2 години на день, помірно активний відповідає ходьбі 3 години на день, дуже активний відповідає ходьбі 4 години на день, а надактивний відповідає ходьбі 5 годин на день (20 миль). До малорухливих належать ті, хто цілий день сидить за столом без будь-якої іншої діяльності, або ті, хто прикутий до інвалідного візка і не може займатися фізичними вправами [28].

Іншим способом підрахунку TDEE на день – буде знаходження BMR, після чого додавання знайдених витрат калорій кожної активності за відомим MET коефіцієнтом. Наприклад, якщо людина, яка важить 55 кг просиділа 8 годин за комп'ютером (MET = 1.5), провела 1 годину на прогулянці (MET = 2) і година силового інтенсивного тренування (MET = 6), то розрахунок спалених калорій за активність (CBPA) буде вестись за такою формулою (1.4):

$$CBPA = (MET \cdot 3.5 \cdot WB \div 200) \cdot DA, \quad (1.4)$$

де *MET* – метаболічний еквівалент активності;

W DA – тривалість активності в хвиликах.

B

Так потрібно знайти витрату калорій за кожною активністю та додати усе до BMR.

Тобто, повертаючись до прикладу розраховується кожна активність за формулою (1.4):

$$(1.5 \cdot 3.5 \cdot 55 \div 200) \cdot 480 = 693 \text{ калорій} - \text{робота за комп'ютером.}$$

$$(2 \cdot 3.5 \cdot 55 \div 200) \cdot 60 = 115.5 \text{ калорій} - \text{прогулянка.}$$

$$(6 \cdot 3.5 \cdot 55 \div 200) \cdot 60 = 346.5 \text{ калорій} - \text{силове інтенсивне тренування.}$$

Тоді TDEE буде розраховуватися за формулою (1.5):

$$TDEE = BMR + CBPA. \quad (1.5)$$

$TDEE = BMR + 693 + 115.5 + 346.5$, також сюди можна додати TEF та PAL для малорухливого способу життя, щоб не враховувати побутові дії [29], [30].

1.4.2 Порівняння формул для підрахунку BMR

BMR (англ. basal metabolic rate) – це кількість енергії, яку тратить організм в стані спокою для підтримки життєвих функцій. Знання свого BMR дозволяє розрахувати суточну норму калорій для досягнення цілей похудання або збереження здорового ваги. Існує багато формул для підрахунку BMR, але не всі з них однаково точні та доступні.

Для порівняння обрано формули Харріса-Бенедикта та Міффіна-Сан Жеора, які є двома найпопулярнішими методами для підрахунку BMR. Обидві формули мають свої переваги та недоліки. Перевагою є те, що вони не потребують знання безжирової маси тіла або індексу маси тіла, як деякі інші формули. Для їх застосування достатньо вказувати такі фактори, як стать, вік, ріст, вага та фізична активність. Недоліком є те, що вони не враховують

індивідуальні особливості організму, такі як спадковий фактор, гормональний статус, склад та об'єм м'язової маси. Тому вони можуть давати помилку до 20-30%. Для більш точного підрахунку BMR рекомендується звертатися до спеціаліста або використовувати спеціальне обладнання [31].

Рівняння Харріса-Бенедикта і Міффіна-Сан Жеора дають оцінку базальних енергетичних витрат (BEE), що також називаються швидкістю метаболізму у стані спокою (RMR) або базальною швидкістю метаболізму (BMR). Рівняння прогнозування енергії зазвичай використовують у лікарнях і дієтологічних клініках визначення потреби у калоріях різних пацієнтів. З цих двох найбільш часто використовуваних рівнянь прогнозування енергії рівняння Міффіна-Сан Жеора дають найбільш надійні результати [31].

Формула Харріса-Бенедикта була створена в 1919 після дослідження Джеймса Артура Харріса і Френсіса Гано Бенедикта. Формули були переглянуті в 1984 з використанням нових даних для підвищення точності. Формула Харріса-Бенедикта була основною формулою для розрахунку BMR до 1990 року, коли було опубліковано формулу Міффіна-Сан Жеора [32].

Формула Харріса-Бенедикта для чоловіків має такий вигляд (1.6):

$$BMR = 66.47 + (13.75 \cdot WB) + (5.003 \cdot H) - (6.755 \cdot A), \quad (1.6)$$

де WB – маса тіла в кг;

H – зріст у см;

A – вік у роках.

Формула Харріса-Бенедикта для жінок має такий вигляд (1.7)

$$BMR = 655.1 + (9.563 \cdot WB) + (1.85 \cdot H) - (4.676 \cdot A), \quad (1.7)$$

де WB – маса тіла в кг;

H – зріст у см;

A – вік у роках.

У 1990 році на науковій конференції обговорювалася нова формула прогнозування витрати енергії в стані спокою у здорових людей. Це була вдосконалена формула Харріса Бенедикта – формула Міффіна-Сан Жеора. Вона вважається більш прогнозованою для сучасного способу життя і зарекомендувала себе як стандарт для розрахунку BMR [33].

Формула Міффіна-Сан Жеора для чоловіків має такий вигляд (1.8):

$$BMR = (10 \cdot WB) + (6.25 \cdot H) - (5 \cdot A) + 5, \quad (1.8)$$

де WB – маса тіла в кг;

H – зріст у см;

A – вік у роках.

Формула Міффіна-Сан Жеора для жінок має такий вигляд (1.9):

$$BMR = (10 \cdot WB) + (6.25 \cdot H) - (5 \cdot A) - 161, \quad (1.9)$$

де WB – маса тіла в кг;

H – зріст у см;

A – вік у роках.

І формула Харріса-Бенедикта, і формула Міффіна-Сан Жеора сьогодні широко використовуються. Було проведено дослідження, яке показало, що різниця між рівняннями Харріса-Бенедикта та Міффіна-Сан Жеора становить близько 5%, при цьому більш високий рівень точності досягається за допомогою формули Міффіна-Сан Жеора [34].

1.4.3 Користь підрахунку калорій за тиждень

Рахування калорій протягом тижня допомагає планувати свій раціон та рівень фізичної активності відповідно до цілей, наприклад, схуднути, набрати м'язову масу або покращити здоров'я. Людина може стежити за своїм

калорійним балансом, тобто різницею між спожитими і витраченими калоріями, і регулювати його за необхідності. Якщо баланс калорій є позитивним, то організм накопичує надлишок калорій у вигляді жиру. Якщо баланс калорій негативний, організм виявляє дефіцит надходження калорій і використовує жирові запаси, щоб заповнити різницю [35].

Також рахування калорій протягом тижня дозволяє бачити загальну картину свого харчування і враховувати випадкові відхилення від норми, наприклад, святкові обіди або періоди голодування. Рахування калорій протягом тижня сприяє формуванню здорових харчових звичок і попереджає переїдання або недоїдання. Людина може бути більш уважною до свого організму і його потреб, а також до складу і якості продуктів, які вживаються [36].

Середня щоденною витратою енергії за тиждень – це середня кількість калорій, яку людина споживає і витрачає протягом одного тижня. Цей показник можна отримати шляхом додавання щоденних витрат енергії за сім днів за одним з методів, описаних у попередньому пункті і ділення їх на сім [36].

Загальна тижнева витрата енергії (TWEE) – це сумарна кількість енергії, яку людина споживає за один тиждень. Щоб знайти її, потрібно додавати щоденні витрати енергії за сім днів. І вже тоді виявляти в залежності від цілі на скільки людина пішла у профіцит або дефіцит калорій.

Наприклад, якщо жінка вагою 60 кг, зростом 165 см і віком 30 років має низьку рухову активність, то її загальна тижнева витрата енергії за формулою (1.9) буде:

$$BMR = (10 \cdot 60) + (6.25 \cdot 165) - (5 \cdot 30) - 161 = 1320 \text{ калорій.}$$

Для розрахунку TDEE використовується формула (1.2):

$$TDEE = 1320 \cdot 1.375 + 181.5 = 1996.5 \text{ калорій,}$$

де PA дорівнює 1815 калорій, TEF за формулою (1.3) дорівнює 181.5 калорій.

TWEE розраховується за формулою (1.10):

$$TWEE = TDEE \cdot 7. \quad (1.10)$$

Тобто TWEE за прикладом буде дорівнювати 12886.5 калорій.

Була розрахована загальна тижнева витрата енергії для підтримки власної маси тіла і вже від неї відштовхуватися для реалізації встановлених цілей.

Перевагою підрахунку загальної тижневої витрати енергії є те, що це дозволяє бачити загальну картину свого енергетичного балансу за довший період часу, а також компенсувати можливі коливання у споживанні і витраті калорій протягом різних днів тижня. Недолікою підрахунку загальної тижневої витрати енергії є те, що це не дає достатньої інформації про свій енергетичний баланс за кожен окремий день, а також не враховує можливих змін у факторах, які впливають на енергетичний баланс, таких як стрес, хвороба, гормональний стан тощо [36].

1.5 Огляд існуючих аналогів

Було підібрано 4 аналоги у "Google Play" (табл. 1.3) за низкою критеріїв: високий рейтинг та позитивні відгуки, цінова політика, зручність використання, кросплатформність, функціональність.

1.5.1 Застосунок «FatSecret»

FatSecret – це застосунок, за допомогою якого користувачі можуть відслідковувати їх тренування та дієту.

Застосунок надає:

– щоденник харчування, що допоможе визначити всі продукти, які

споживав користувач. Можна ввести в систему, усе що споживалося і в яких кількостях;

- інформація про харчову цінність, куди входить інформація про всі види брендів, продукти харчування та страви різних ресторанів. Є можливість перевірити загальний вміст макро- та мікронутрієнтів у всіх продуктах;

- щоденник тренувань, який полегшує відстеження загальної кількості калорій, що спалюються користувачем. Можна використовувати це, щоб визначити кількість калорій, які людина спалила протягом дня, в порівнянні з тим, що вона з'їла;

- графік ваги та журнал, який тримає в курсі коливань ваги та прогресу користувача. Це може допомогти відстежувати загальний прогрес та з'ясувати, на що потрібно звернути більше уваги;

- корисні рецепти, що містять конкретну інформацію про те, як готувати їжу. Сюди входить випічка, десерти, закуски, салати, обіди, напої, гарніри [37]. Інтерфейс застосунку зображено на рис. 1.1.



Рисунок 1.1 – Користувацький інтерфейс «FatSecret» [38]

1.5.2 Застосунок «Lifesum»

Lifesum – це застосунок для відстеження харчування, фізичних вправ та інших видів активності, щоб допомогти виробити здорові звички. Крім відстеження, Lifesum надає користувачам рейтинги продуктів харчування, плани харчування та рецепти.

Застосунок надає:

- щоденник харчування, що показує спрощений огляд щоденних цілей із зазначенням того, скільки калорій було з'їдено, спалено і скільки залишилося з'їсти;

- плани дієти, де у безкоштовній версії пропонується стандартна дієта Lifesum, яка включає в себе цілі по споживанню макронутрієнтів, засновані на загальних рекомендаціях. Це план, який сприяє вживанню різноманітних фруктів, овочів, цільнозернових продуктів і корисних жирів. А преміум-підписка пропонує широкий вибір планів дієти, які можна вибрати відповідно до своїх уподобань та потреб. Ці плани включають найпопулярніші дієти, а також плани періодичного голодування. Наприклад: середземноморська дієта, скандинавська дієта, чисте харчування, 6:1 (переривчасте голодування), 5:2 (переривчасте голодування), стандарт Lifesum, підвищений білок;

- плани харчування, де у преміум версії, також буде доступ до планів харчування, які мають списки продуктів. Більшість планів харчування підходять для досягнення будь-якої мети, але є деякі, які оптимізовані лише для схуднення та підтримки ваги. Деякі варіанти плану харчування: Втрата ваги за 3 тижні, Кето-підтримка, Білкова втрата ваги;

- відстеження тренувань, де запис тренувань можна виконати вручну або через сумісний застосунок, який використовується для запису тренувань, підключивши його до Lifesum. Відстеження тренувань може допомогти виробити здорові звички та спостерігати за своїм прогресом із часом. За бажанням щоденна мета за калоріями може враховувати калорії, які спалюються в результаті активності, але це преміум-функція [39]. Інтерфейс

застосунок зображено на рис. 1.2.



Рисунок 1.2 – Користувачський інтерфейс «Lifesum» [40]

1.5.3 Застосунок «MyNetDiary»

MyNetDiary – це зручний онлайн-щоденник харчування, який може допомогти відстежувати їжу та фізичні вправи при досягненні мети за складом тіла.

Застосунок надає:

- облік калорій, де надається сканер штрих-коду та величезна база даних про продукти харчування, яка містить понад 1 мільйон записів, що дозволяє легко відстежувати споживання їжі. Також застосунок дозволяє легко реєструвати продукти або страви, що часто вживаються, приготовані за власними рецептами. Крім того, є можливість імпортувати власні рецепти та розрахувати їхню харчову цінність за допомогою інструменту імпорту вебрецептів. Потім застосунок віднімає калорії, які споживалися із щоденного бюджету калорій, щоб показати скільки калорій залишилося на день;

- відстеження активності, що дозволяє реєструвати спалені калорії. Це

можна зробити вручну або синхронізувати дані з популярних фітнесзастосунків, таких як пристрої Garmin, Fitbit і Withings, щоб заощадити час;

– спільнота, де форуми модеруються зареєстрованими дієтологами та пропонують підтримку на кожному етапі шляху. Ці спільноти надають простір для спілкування з людьми, які поділяють схожі цілі, що дозволяє ділитися досвідом та звертатися за порадою до розуміючої спільноти;

– преміум-меню, бо більшості людей складно харчуватися правильно, коли вони намагаються схуднути чи набрати вагу. MyNetDiary позбавляє цієї проблеми, надаючи меню преміум-класу, створене зареєстрованими дієтологами. Наприклад, у застосунку можна знайти щоденне меню на 1500 калорій із докладними рекомендаціями щодо харчування. Це позбавить здогадів при виборі щоденних страв і допоможе харчуватися здоровою їжею [41]. Інтерфейс застосунку зображено на рис. 1.3.

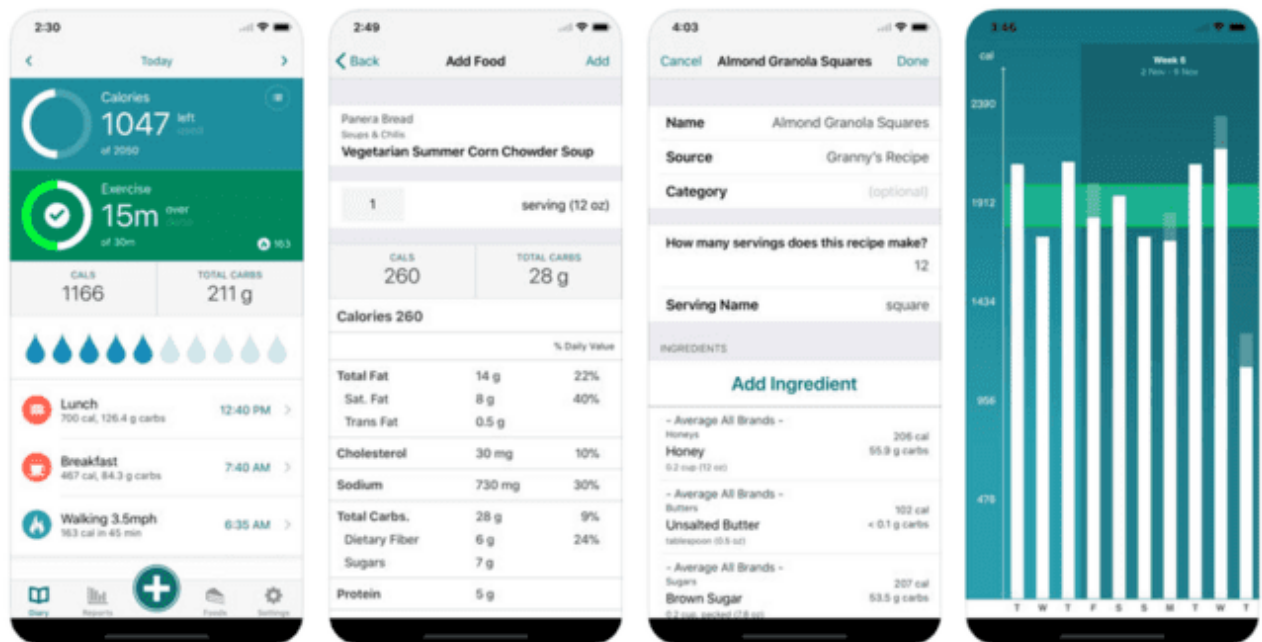


Рисунок 1.3 – Користувачський інтерфейс «MyNetDiary» [40]

1.5.4 Застосунок «Cronometer»

Cronometer – це застосунок для відстеження калорій, який допомагає реєструвати як спожиті, так і спалені калорії. Застосунок допомагає вимірювати щоденне споживання їжі та контролювати вміст поживних речовин у ній.

Застосунок надає:

- щоденник, в якому можна побачити огляд спожитих і спалених калорій, а також баланс «калорій, що залишилися». Крім того, там є огляд споживаних макронутрієнтів: білок, чисті вуглеводи та жири. І огляд цільових показників поживних речовин, таких як клітковина, залізо, вітамін С і т.п.;

- тенденції, де можна побачити графіки щоденного споживання калорій, тенденції ваги користувача та зведені дані про поживні речовини. Безкоштовна версія спрощена, але все одно можна побачити розбивку макросів та динаміку калорій;

- продукти харчування, де можна створювати нові продукти, яких ще немає в базі даних, або створювати рецепти, комбінуючи продукти в базі даних для створення страви, що допоможе заощадити час надалі, не записуючи кожен продукт окремо;

- налаштування, в якому точність цифр повністю визначається тим, наскільки точний профіль. Можна встановити цільові показники за макронутрієнтами та вагою. Можна отримати доступ до форуму спільноти, звернутися до служби підтримки та настроїти параметри щоденника [42]. Інтерфейс застосунку зображено на рис. 1.4.

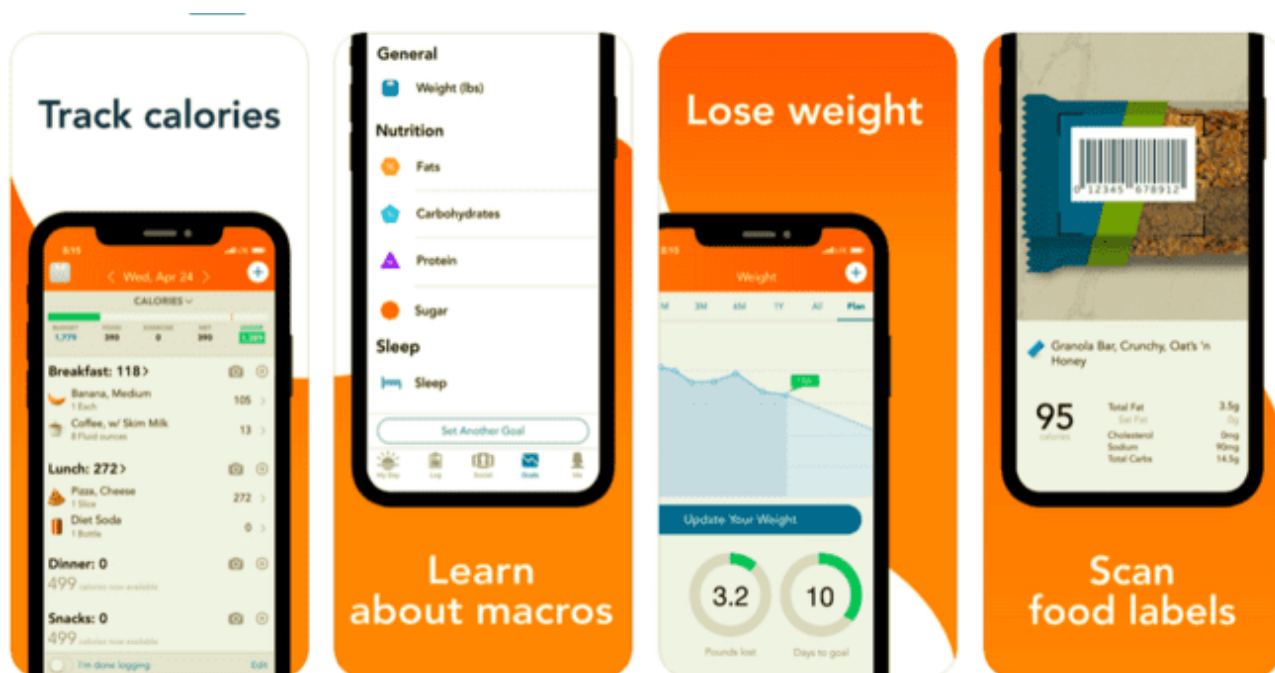


Рисунок 1.4 – Користувачський інтерфейс «Cronometer» [40]

1.5.5 Порівняння існуючих та розроблюваного програмних продуктів

Проаналізовані застосунки мають певні функції, що відповідають різноманітним потребам користувачів. Вони пройшли валідацію щодо відповідності медичним і бізнес-стандартам Healthline [43]. Усі застосунки обладнані інструментами для моніторингу кількості калорій і макронутрієнтів, основаних на введених або відсканованих продуктах і стравах. Крім того, вони дозволяють вести контроль за вагою та іншими показниками здоров'я, встановлювати індивідуальні цілі та плани для зниження або набору ваги. Ці програми також надають рецепти та поради з питань здорового харчування та способу життя. Крім того, вони можуть синхронізуватися з іншими застосунками або пристроями для вимірювання фізичної активності та стану здоров'я, такими як крокоміри, годинники або фітнес-браслети [43, 44].

В таблиці 1.3 проведено порівняння вже існуючих програмних продуктів та застосунку «NutriLight», що буде розроблено в ході виконання роботи.

Таблиця 1.3 – Порівняння програмних продуктів

Властивості	Назва застосунку				
	FatSecret	Lifesum	MyNetDiary	Cronometer	NutriLight
Зручність	4	5	3	4	5
Дизайн	3	4	3	4	5
Кросплатформність	Так	Так	Так	Так	Так
Функціональність	5	4	4	5	5
Наявність преміум версії	Так	Так	Так	Так	Ні
Доступність функцій в безкоштовній версії	3	2	3	3	5

Усі властивості, окрім кросплатформності та наявності преміум версії оцінюються за шкалою Лайкерта від 1 до 5, де 1 – найгірша оцінка, а 5 – найкраща.

Lifesum та NutriLight отримали найвищі оцінки за зручність, що свідчить про їхню інтуїтивність та легкість у використанні. Користувачі можуть легко орієнтуватися в програмі та використовувати всі її функції.

F

a

t

S

в

и

г

я

р

д

б

ж

ф

Є

о

п

п

За дизайн найвищу оцінку отримав NutriLight, бо має найестетичніший

Усі застосунки кросплатформні, бо вони можуть бути використані на різних операційних системах або пристроях.

FatSecret, Cronometer та NutriLight мають широкий спектр функцій, як базових, так і додаткових, тому оцінені на високий бал. Lifesum та MyNetDiary також високо оцінені. В них гарна функціональність, але відсутній деякий функціонал, який присутній в інших застосунках.

У всіх застосунках, окрім NutriLight є преміум версії, що вказує на

наявність додаткових функцій та інструментів за додаткову плату.

NutriLight має найвищу оцінку, бо усі функції доступні безкоштовно. FatSecret, MyNetDiary та Cronometer отримали середню оцінку, бо обмежена доступність функцій у безкоштовній версії, деякі з яких базові. Lifesum отримав найнижчу оцінку, бо застосунок дуже обмежений за функціоналом, мінімальний набір функцій доступний безкоштовно.

В застосунках більшість функцій платна або є багато реклами. В багатьох з них не зовсім інтуїтивно зрозумілий інтерфейс, немає україномовної локалізації. FatSecret недооцінює певні харчові цінності, може неточно визначати рівень натрію та білка в їжі [37]. Lifesum має обмежену базу даних продуктів, безкоштовна версія не синхронізується з усіма фітнес-застосунками та в наявності неузгоджені безкоштовні функції між версіями вебсайту та програми, а також дуже обмежений функціонал в безкоштовній версії[45]. У MyNetDiary складний інтерфейс та плани харчування зосереджено тільки на відстеженні їжі для схуднення [46]. В Cronometer обмежений вибір рецептів і не дуже зручне меню. Користувацькі рецепти потрібно спочатку ввести через вебсайт, перш ніж вони стануть доступними в застосунку [47], [48].

Підводячи підсумки, розробники застосунків акцентували увагу на різноманітності функціоналу та інтерфейсі користувача. Проте, багато з важливих функцій доступні тільки в розширених версіях (преміум), що може впливати на задоволення потреб більшої частини користувачів. Вирішення цих питань може включати розробку програми, яка збереже базові функції, а також забезпечить доступ до деяких розширених функцій без необхідності підписки на преміум версію.

1.6 Постановка задач роботи

Після проведеного аналізу було встановлено, що доцільно розпочати розробку інструменту для контролю харчування у вигляді вебсайту і реалізовувати як кросплатформенний застосунок під смартфони.

Мета роботи – створення застосунку для оптимізації харчування широкого кола користувачів.

Задачі роботи:

- виконати аналіз вимог до програмного забезпечення;
- спроектувати структуру застосунку та обрати інструментарій розробки;
- виконати програмну реалізацію;
- провести тестування застосунку;
- розробити програмну документацію.

2 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО ОБҐРУНТУВАННЯ

2.1 Основні патерни та підходи до розробки вебзастосунків

Побудова вебзастосунків вимагає ретельного вивчення різних підходів та патернів для забезпечення ефективності, розширюваності та справедливого управління проектами. У цьому розділі буде розглянуто різні підходи та патерни для побудови вебзастосунків.

2.1.1 MVC (Model-View-Controller)

MVC – це популярний паттерн для розділення логіки застосунку, що складається з таких ключових компонентів, як Модель (Model), Подання (View) і Контролер (Controller).

Модель відповідає за дані та бізнес-логіку. Вона представляє собою структуру даних та правила для їх обробки.

Подання відповідає за відображення інформації користувачу. Подання відображає дані з Моделі та може містити елементи інтерфейсу.

Контролер відповідає за обробку запитів користувача, керування потоком застосунку, приймає рішення на основі вхідних даних, взаємодіє з Моделлю та оновлює Подання.

MVC допомагає розділити обов'язки між цими компонентами, що полегшує розробку та підтримку.

Використання: MVC широко використовується в багатьох вебфреймворках, таких як Ruby on Rails, Django, та Spring [49].

2.1.2 MVVM (Model-View-ViewModel)

MVVM – це паттерн, схожий на MVC, але з введенням ViewModel, яка служить для розділення логіки відображення від бізнес-логіки.

ViewModel забезпечує двосторонню зв'язок між Моделлю та Поданням.

Модель так само, як у MVC відповідає за дані та бізнес-логіку.

Подання в MVVM також відповідає за відображення, але важливий аспект полягає в тому, що Подання взаємодіє з ViewModel, а не напряду з Моделлю.

ViewModel визначає, які дані мають відображатися в Поданні. ViewModel є проміжним рівнем, який відділяє бізнес-логіку від відображення. Він містить дані та методи, які дозволяють Поданню отримувати та відображати дані з Моделі. Він також реагує на дії користувача та оновлює Модель за потреби.

Використання: MVVM часто використовується в фреймворках, таких як Vue.js та Knockout.js [49].

2.1.3 SPA (Single Page Application)

SPA – це підхід, в якому весь контент завантажується одноразово при запуску застосунку, і подальша навігація відбувається без перезавантаження сторінки. Це дозволяє створювати динамічні та інтерактивні застосунки.

Основною ідеєю SPA є завантаження всього необхідного вмісту разом із застосунком при старті, після чого весь взаємодія з застосунком відбувається через асинхронні запити до сервера без необхідності перезавантаження сторінки.

SPA дозволяє створювати застосунки з високою інтерактивністю і реагуванням на дії користувача, оскільки весь процес навігації відбувається швидко та без відчутного перезавантаження.

Використання: SPA дуже поширений підхід для розробки вебзастосунків за допомогою бібліотек та фреймворків, таких як React, Angular, та Vue.js [50].

2.1.4 SSR (Server-Side Rendering)

SSR – це підхід до розробки вебзастосунків, при якому генерація HTML сторінки відбувається на сервері, а не в браузері. Це може підвищити швидкість завантаження сторінок та покращити SEO.

Замість того, щоб клієнтський браузер генерував HTML, весь HTML генерується на стороні сервера і надсилається користувачеві як готова сторінка.

Цей підхід дозволяє значно пришвидшити завантаження сторінок, оскільки браузер отримує готову до відображення сторінку. Крім того, сторінки, згенеровані на сервері, добре індексуються пошуковими системами, що покращує SEO [51].

Використання: SSR популярний для фреймворків, таких як Next.js для React, Nuxt.js для Vue.js та Angular Universal для Angular [52].

2.1.5 RESTful API

REST (Representational State Transfer) – це стиль архітектури для розробки вебсервісів та API, який базується на ресурсах та операціях над ними.

У REST, кожен об'єкт (ресурс) має свій унікальний ідентифікатор (URL), і взаємодія з цими ресурсами відбувається за допомогою стандартних HTTP методів, таких як GET (отримання даних), POST (створення нового ресурсу), PUT (оновлення існуючого ресурсу) та DELETE (видалення ресурсу).

Основна ідея REST – це представлення стану ресурсу в відповіді на запит. Кожен запит повинен містити необхідну інформацію про ресурс та операцію, яка виконується над ним.

Використання: RESTful API дуже поширений для створення серверних застосунків та взаємодії між клієнтами та серверами. Велика кількість

вебслужб, соціальних мереж і публічних API побудовані з використанням RESTful підходу [53].

2.1.6 GraphQL

GraphQL – це запитова мова та серверна технологія, яка дозволяє клієнтам визначати, які дані вони потребують. Замість того, щоб завантажувати всі дані разом, клієнти вказують, які саме поля їх цікавлять.

GraphQL надає гнучкий підхід до отримання даних. Клієнти можуть вказати, які саме поля та зв'язки їх цікавлять, і отримати лише цю інформацію.

У GraphQL існує одна єдина точка доступу (зазвичай /graphql), де клієнти надсилають всі свої запити. Сервер GraphQL розбирає ці запити та надає відповіді, відповідно до запитів.

GraphQL базується на схемі, яка описує всі доступні типи даних та операції. Це надає великий контроль над структурою даних та валідацію запитів.

Використання: GraphQL здобуває популярність як альтернатива REST для створення гнучких API. Великі компанії, такі як Facebook, GitHub і Shopify, використовують GraphQL для своїх API. Використання GraphQL особливо корисне там, де потрібна точна обробка запитів та динамічність взаємодії між клієнтами та серверами [54].

2.1.7 Мікросервісна архітектура

Мікросервіси – це архітектурний підхід, де великий застосунок розділяється на невеликі незалежні сервіси, кожен з яких виконує конкретну функцію. Це дозволяє спрощувати розгортання, масштабування та підтримку.

Використання: Мікросервісна архітектура часто використовується для складних застосунків, які складаються з багатьох компонентів. Це дозволяє

розробникам та операторам системи зосередитися на конкретних функціональних аспектах та полегшує розгортання та масштабування [55].

2.1.8 Шаблони проєктування (Design Patterns)

Шаблони проєктування – це загальні архітектурні рішення для типових проблем, які можуть виникати в розробці програмного забезпечення. Ці шаблони допомагають структурувати код та робити його більш зрозумілим, підтримуючи принципи відновлення інкапсуляції та повторного використання.

Декілька прикладів шаблонів проєктування включають:

- Singleton, що гарантує, що клас має лише один екземпляр і надає глобальну точку доступу до цього екземпляру;
- Observer, який встановлює залежність одного або більше об'єктів від стану іншого об'єкта, так щоб коли один об'єкт змінює свій стан, всі його залежні об'єкти будуть повідомлені та оновлені;
- Factory, що визначає інтерфейс для створення об'єктів, але залишає реалізацію створення об'єкта підкласам, рішення про те, який саме підклас створити, відкладається на підкласи [56].

2.2 Визнані сучасні технології/протоколи для вебзастосунків

В сучасному інтернеті, розробка вебзастосунків стає все більш складною і різноманітною завдяки росту технологій та протоколів. Розглянемо декілька ключових технологій та протоколів, які грають важливу роль у створенні сучасних вебзастосунків.

HTTP (Hypertext Transfer Protocol) – це протокол, який визначає спосіб, яким браузері та сервери обмінюються даними в інтернеті. Він є основою для взаємодії між клієнтом і сервером у світі вебзастосунків. HTTP працює на основі запит-відповідь, де клієнт (зазвичай браузер) відправляє запит на

сервер, і сервер надсилає відповідь. Протокол HTTP має різні методи, включаючи GET (отримати дані), POST (надіслати дані), PUT (оновити дані), DELETE (видалити дані) та інші. Ці методи дозволяють взаємодіяти з ресурсами на сервері [57].

CSS (Cascading Style Sheets) – є мовою стилізації, яка використовується для визначення вигляду та форматування вебсторінок. CSS відокремлює зовнішній вигляд вебсторінок від їх вмісту, що дозволяє розробникам керувати дизайном та відображенням контенту на сторінці. Основні концепції CSS включають в себе селектори, властивості та значення. Можна використовувати CSS для зміни кольорів, шрифтів, розмірів, відступів та багатьох інших аспектів дизайну сторінок [58].

HTML (Hypertext Markup Language) – є основною мовою розмітки для вебсторінок. Вона використовується для створення структури та організації вмісту на вебсторінках. HTML включає в себе різноманітні теги, які визначають заголовки, абзаци, зображення, посилання та інші елементи сторінки. В поєднанні з CSS, HTML дозволяє створювати красиво оформлені та добре структуровані вебсторінки [59].

Сучасні вебзастосунки використовують різні розширення і модифікації CSS та HTML для покращення функціональності та дизайну. Наприклад, CSS-препроцесори, такі як SASS і LESS, дозволяють розробникам використовувати змінні, вкладені стилі та інші покращення в CSS [60]. HTML5 вніс значні зміни у можливості HTML, додавши нові теги, аудіо та відео елементи, графіку без використання плагінів, та багато іншого.

2.3 Огляд популярних фреймворків для написання клієнтської сторони вебзастосунку

Фреймворк для фронтенду – це набір інструментів та бібліотек, які спрощують розробку вебінтерфейсів. Вибір фреймворку для фронтенду залежить від багатьох факторів, таких як:

- складність та обсяг проєкту;
- вимоги до продуктивності та швидкості завантаження;
- наявність готових компонентів та плагінів;
- крива навчання та документація;
- популярність та підтримка спільноти.

Для порівняти було обрано три найпопулярніші фреймворки для фронтенду: Angular, React та Vue. Їх буде порівняно за основними характеристиками, перевагами та недоліками. А також буде обґрунтовано вибір одного з них для реалізації застосунку [61].

2.3.1 Angular

Angular – це повноцінний фреймворк для фронтенду, який базується на TypeScript, розроблений Google. Angular використовує компонентний підхід до створення вебзастосунків, а також надає багато інтегрованих рішень, таких як маршрутизація, форми, анімація, HTTP-запити, тестування тощо.

Переваги Angular:

- дозволяє створювати масштабовані та структуровані вебзастосунки за допомогою TypeScript, який забезпечує статичну типізацію, класи, інтерфейси, декоратори тощо;
- має потужний механізм залежностей введення (dependency injection), який сприяє модульності, перевикористанню коду та зниженню залежності одного компонента по відношенню до іншого;
- має багатий набір готових компонентів та плагінів, як офіційних, так і сторонніх, які додають функціональність та полегшують розробку;
- має велику та активну спільноту розробників, яка підтримує та оновлює фреймворк, а також надає багато ресурсів для навчання та допомоги.

Недоліки Angular:

- має високу криву навчання, оскільки потребує знання TypeScript, а також особливостей фреймворку, таких як директиви, модулі, сервіси, пайпи

тощо;

- може мати проблеми з продуктивністю та швидкістю завантаження, особливо при роботі з великими та динамічними застосунками, через його механізм зв'язування даних (data binding) та відстеження змін (change detection);

- може бути несумісним з деякими старими браузерами та платформами, оскільки він використовує сучасні стандарти веброзробки [62].

2.3.2 Vue

Vue – це прогресивний фреймворк для фронтенду, який фокусується на шарі представлення (view layer). Vue використовує реактивний підхід до створення вебзастосунків, а також дозволяє поступово розширювати його функціональність за допомогою офіційних та сторонніх пакетів, таких як Vue Router або Vuex.

Переваги Vue:

- має низьку криву навчання, оскільки він базується на HTML, CSS та JavaScript, а також має простий та інтуїтивний синтаксис шаблонів;

- має високу продуктивність та швидкість завантаження, оскільки він має невеликий розмір (приблизно 20 КБ) та ефективний механізм зв'язування даних та відстеження змін;

- має гнучку та модульну архітектуру, яка дозволяє адаптувати фреймворк до різних потреб та сценаріїв. Vue може бути використаний як бібліотека для покращення існуючих сторінок або як повноцінний фреймворк для створення складних застосунків;

- має швидко зростаючу та ентузіастичну спільноту розробників, яка створює багато корисних ресурсів, компонентів та плагінів для фреймворку.

Недоліки Vue:

- має меншу популярність та підтримку, ніж Angular або React, оскільки він є новим і незалежним проєктом, який не належить жодному

великому компанії;

- може мати проблеми з інтеграцією з деякими сторонніми бібліотеками або існуючими проектами, оскільки вони можуть бути несумісними з його реактивною системою;

- може бути нестабільним або непередбачуваним при роботі з деякими функціями, такими як обчислювання властивостей (computed properties), спостереження (watchers) або перехоплення помилок (error handling) [63].

2.3.3 React

React – це бібліотека для фронтенду, яка базується на JavaScript, розроблена Facebook. React дозволяє створювати вебзастосунки за допомогою компонентів, які представляють дані, що змінюються в часі. React використовує віртуальний DOM, який оптимізує рендеринг сторінки та покращує продуктивність. React також підтримує JSX, який є синтаксичним розширенням JavaScript, що дозволяє писати HTML-подібний код у JavaScript.

Переваги React:

- має високу продуктивність та швидкість завантаження, оскільки він використовує віртуальний DOM, який зменшує кількість операцій з реальним DOM та оновлює лише ті частини сторінки, які змінилися;

- має простий та інтуїтивний синтаксис JSX, який дозволяє писати HTML-подібний код у JavaScript та використовувати всю потужність JavaScript для створення компонентів;

- має велику та активну спільноту розробників, яка підтримує та оновлює бібліотеку, а також надає багато ресурсів для навчання та допомоги;

- має гнучку та модульну архітектуру, яка дозволяє комбінувати його з іншими бібліотеками та фреймворками для створення складних функцій. Він також підтримує розробку крос-платформних застосунків за допомогою React Native.

Недоліки React:

- має середню криву навчання, оскільки він потребує знання JavaScript, а також особливостей бібліотеки, таких як компоненти, стани (states), властивості (props), хуки (hooks) тощо;
- не є повноцінним фреймворком для фронтенду, а лише бібліотекою для шару представлення (view layer). Тому, для створення повнофункціонального вебзастосунку потрібно вибрати інші бібліотеки та фреймворки для реалізації таких функцій, як маршрутизація, стан застосунку, аутентифікація тощо. Це може призвести до складності вибору, інтеграції та підтримки таких рішень;
- може мати проблеми з сумісністю з деякими старими браузерами та платформами, оскільки він використовує сучасні стандарти веброзробки та JSX, який потребує транспайлера (transpiler) для перетворення коду в звичайний JavaScript;
- може бути нестабільним або непередбачуваним при роботі з деякими функціями, такими як життєвий цикл компонентів (component lifecycle), хуки (hooks) або оптимізація продуктивності (performance optimization) [64].

2.3.4 Порівняння фреймворків та обґрунтування вибору

Нижче наведено порівняльну таблицю, що показує пряме порівняння між Angular, Vue і React (табл. 2.1).

Таблиця 2.1 – Порівняльна таблиця Angular, Vue і React

Характеристика	Angular	Vue	React
1	2	3	4
Тип	Повноцінний фреймворк	Прогресивний фреймворк	Бібліотека
Мова програмування	TypeScript	JavaScript	JavaScript

Продовження таблиці 2.1

1	2	3	4
Механізм зв'язування даних	Двосторонній (two-way)	Односторонній (one-way)	Односторонній (one-way)
Механізм відстеження змін	Зона (zone) та цикл (Lifecycle Hooks)	Гетери (getters) та сетери (setters)	Віртуальний DOM (virtual DOM)
Синтаксис шаблонів	HTML + директиви (directives)	HTML + директиви (directives) + інтерполяція (interpolation)	JSX
Компонентна архітектура	Міцна інтеграція зі структурою компонентів, сприяє модульності	Добра підтримка компонентної архітектури, але менш жорстка	Рекомендується використовувати зовнішні бібліотеки для компонентів
Структура компонентів	Класи + декоратори (decorators) + шаблони (templates) + стилі (styles)	Об'єкти + шаблони (templates) + скрипти (scripts) + стилі (styles)	Функції або класи + JSX
Готові рішення та плагіни	Багато офіційних та сторонніх	Менше офіційних, багато сторонніх	Мало офіційних, багато сторонніх
Крива навчання	Висока	Низька	Середня
Популярність та підтримка спільноти	Висока, Google та інші великі компанії	Середня, незалежний проєкт, багато малих компаній та проєктів	Висока, Facebook та інші великі компанії

З усього вищевказаного можна зробити висновок, що Angular є найбільш повноцінним та потужним фреймворком для фронтенду, який надає багато готових рішень та плагінів для розробки складних застосунків. Angular надає повний стек інструментів для розробки фронтенду, включаючи модулі, компоненти, маршрутизацію, HTTP запити та багато інших. Це дозволяє працювати з однією інтегрованою системою. Angular також має велику

популярність та підтримку спільноти, що забезпечує його стабільність та оновлення. Angular використовує TypeScript, який додає статичну типізацію та сучасні можливості мови програмування, що полегшують написання та покращують якість коду. Angular підтримує компонентну архітектуру, що сприяє модульності і підтримці великих застосунків. Також Angular задовольняє таким потребам:

- масштабованості та структурованості коду, оскільки Angular використовує TypeScript, який дозволяє створювати чистий та зрозумілий код з використанням класів, інтерфейсів, декораторів тощо;

- продуктивності та швидкості завантаження, оскільки Angular використовує механізм залежностей введення, який зменшує куплінг між компонентами та полегшує їх тестування та оптимізацію;

- функціональності та розширюваності, оскільки Angular надає багато готових компонентів та плагінів, які додають різноманітну функціональність до вебзастосунку, таку як маршрутизація, форми, анімація, HTTP-запити тощо. А також має велику кількість сторонніх бібліотек і інструментів, що полегшують розробку та розширюють можливості фреймворку.

Таким чином, Angular є найкращим варіантом для реалізації застосунку.

2.4 Огляд популярних фреймворків для написання серверної частини вебзастосунків

Вебфреймворк – це інструмент, який надає основні модулі для створення гнучкого вебсайту, забезпечуючи широкі можливості для подальшої модернізації та розширення функціональності через інтеграцію додаткових програм у майбутньому. Вебфреймворки можуть бути розділені на три категорії за типом архітектури:

- backend, де вебфреймворки використовуються виключно на серверній стороні. Вони спрямовані на виконання конкретних завдань,

пов'язаних із зберіганням і захистом даних, а також забезпеченням життєво важливих процесів;

- frontend, де вебфреймворки не взаємодіють із логікою програми, оскільки їх функціонал працює безпосередньо в браузері. Вони спрямовані на створення інтерфейсів користувача з різноманітною графікою та анімацією;

- повний стек, де вебфреймворки виступають у ролі універсальних інструментів, які виконують функції backend та будують клієнтську частину, як один застосунок.

Крім того, вебфреймворки можна розділити за мовою програмування, яку вони використовують. Далі буде розглянуто популярні фреймворки для п'яти мов програмування: Java, Node.js, C#, Python та PHP [65].

2.4.1 Java та її фреймворк

Java – високорівнева мова програмування, яка займає широку нішу в розробці масштабних backend проєктів. Java має багатий набір фреймворків, які спрощують розробку, тестування та розгортання вебзастосунків [66]. Один з найпопулярніших фреймворків для Java – це Spring Boot.

Spring Boot – це фреймворк, який дозволяє створювати самодостатні вебзастосунки на основі Spring Framework. Spring Boot автоматично налаштовує необхідну конфігурацію, забезпечує вбудований сервер, підтримує різні бази даних та інтеграцію з іншими технологіями. Spring Boot має такі переваги:

- швидкість розробки. Spring Boot дозволяє створювати вебзастосунки з мінімальним кодом та конфігурацією;

- гнучкість. Spring Boot підтримує різні стилі програмування, такі як імперативний, реактивний, функціональний та аспектно-орієнтований;

- масштабованість. Spring Boot дозволяє створювати високопродуктивні вебзастосунки, які можуть витримувати великі навантаження та легко масштабуватися [67].

2.4.2 Node.js та її фреймворки

Node.js – це не мова програмування, а середовище виконання для JavaScript, яке дозволяє запускати JavaScript код на стороні сервера. Node.js має багато фреймворків, які спрощують розробку вебзастосунків на JavaScript. Два з найпопулярніших фреймворків для Node.js - це Nest.js та Express.js [68].

Nest.js – це фреймворк, який дозволяє створювати ефективні, надійні та масштабовані серверні застосунки на Node.js. Nest.js використовує TypeScript - надбудову над JavaScript, яка додає строгу типізацію та інші можливості. Nest.js має такі переваги:

- архітектура – Nest.js базується на модульному патерні Dependency Injection, який сприяє структурованості, тестованості та замінності компонентів;

- інтеграція – Nest.js підтримує розширення за допомогою багатьох бібліотек та інструментів, таких як GraphQL, WebSocket, Swagger, TypeORM та інші;

- продуктивність – Nest.js дозволяє створювати швидкодіючі вебзастосунки, які можуть обробляти асинхронну логіку та події в режимі реального часу [69].

Express.js – це фреймворк, який дозволяє створювати мінімалістичні та гнучкі вебзастосунки на Node.js. Express.js не нав'язує жодної специфічної архітектури, ORM або шаблонного двигуна, а надає лише базовий функціонал для роботи з HTTP-запитами, маршрутизації, середніми програмами (middleware) та іншим. Express.js має такі переваги:

- простота – Express.js дозволяє створювати вебзастосунки з дуже мало коду та конфігурації;

- гнучкість – Express.js дозволяє вибирати будь-які бібліотеки та інструменти, які пасують до потреб проєкту;

– швидкість – Express.js використовує Node.js, який працює на основі подійної моделі та неблокуючого вводу-виводу, що забезпечує високу продуктивність та паралельну обробку запитів [70].

2.4.3 C# та її фреймворки

C# – це сучасна об'єктно-орієнтована мова програмування, яка розроблялася компанією Microsoft для платформи .NET. C# підтримує багато парадигм програмування, таких як імперативне, декларативне, функціональне, об'єктно-орієнтоване, узагальнене та компонентне. Два з найпопулярніших фреймворків для C# – це ASP.NET Core та SiteCore [71].

ASP.NET Core – фреймворк для створення сучасних вебзастосунків та служб на платформі .NET. ASP.NET Core підтримує розробку на C#, F# та Visual Basic. ASP.NET Core має такі переваги:

– кросплатформенність – ASP.NET Core може працювати на Windows, Linux та macOS;

– модульність в ASP.NET Core дозволяє використовувати лише ті компоненти, які є необхідними для конкретного проекту, а також інтегруватися з іншими бібліотеками та фреймворками;

– продуктивність – ASP.NET Core є одним з найшвидших фреймворків для веброзробки, який оптимізований для клаудних розгортань [72].

SiteCore – це фреймворк для створення вебсайтів та цифрових досвідів на платформі .NET. SiteCore є повнофункціональною системою управління контентом (CMS), яка надає можливості для маркетингу, аналітики, особистизації та автоматизації. SiteCore має такі переваги:

– надійність – SiteCore є стабільним та безпечним фреймворком, який використовується багатьма великими компаніями та організаціями;

– гнучкість – SiteCore дозволяє створювати вебсайти з різним дизайном, функціональністю та контентом, а також адаптувати їх до різних аудиторій та пристроїв;

– інтеграція – SiteCore може інтегруватися з багатьма іншими сервісами та платформами, такими як Azure, Salesforce, Dynamics 365, SharePoint та інші [73].

2.4.4 Python та її фреймворки

Python – це інтерпретована мова програмування високого рівня, яка відрізняється своєю простотою, гнучкістю та широким спектром застосувань. Python має багато фреймворків, які допомагають розробникам створювати вебзастосунки з різною складністю та функціональністю. Два з найпопулярніших фреймворків для Python – це Flask та Django [74].

Flask – це мікрофреймворк для Python, який надає основні можливості для створення вебзастосунків. Flask не має вбудованих компонентів для роботи з базами даних, ORM, аутентифікації, формами та іншим, але дозволяє використовувати будь-які сторонні бібліотеки за бажанням розробника. Flask має такі переваги:

- простота – Flask дозволяє створювати вебзастосунки з дуже мало коду та конфігурації;

- гнучкість – Flask дозволяє контролювати всі аспекти вебзастосунку та налаштовувати його за потребами проєкту [75].

Django – це повнофункціональний фреймворк для Python, який надає все необхідне для створення вебзастосунків будь-якої складності. Django має вбудовані компоненти для роботи з базами даних, ORM, аутентифікації, формами, адмін-панеллю, кешуванням та іншим. Django має такі переваги:

- надійність – Django є стабільним та безпечним фреймворком, який використовується багатьма великими сайтами та компаніями;

- масштабованість – Django дозволяє легко масштабувати вебзастосунки за допомогою горизонтального розподілу навантаження та балансування [76].

2.4.5 PHP та її фреймворки

PHP – це скриптова мова програмування, яка спеціалізується на розробці вебзастосунків. PHP є однією з найпопулярніших мов програмування для веброзробки, яка має багато фреймворків, які полегшують роботу розробників [77]. Два з найпопулярніших фреймворків для PHP – це Laravel та yii2.

Laravel – це фреймворк для PHP, який надає елегантний та виразний синтаксис для створення вебзастосунків. Laravel має вбудовані компоненти для роботи з базами даних, ORM, аутентифікації, кешуванням, шаблонами, валідацією та іншим. Laravel має такі переваги:

- швидкість розробки – Laravel дозволяє створювати вебзастосунки з мінімальним кодом та конфігурацією за допомогою артизанських команд, маршрутизатора, міграцій та інших інструментів;

- якість коду – Laravel дотримується принципів SOLID та MVC, які сприяють чистоті, зрозумілості та підтримці коду;

- безпека – Laravel захищає вебзастосунки від загальних загроз, таких як SQL-ін'єкції, XSS-атаки, CSRF-атаки та інші [78].

yii2 – це фреймворк для PHP, який надає потужний та гнучкий функціонал для створення вебзастосунків. yii2 має вбудовані компоненти для роботи з базами даних, ORM, аутентифікацією, кешуванням, RESTful API, шаблонами, валідацією та іншим. yii2 має такі переваги:

- масштабованість – yii2 дозволяє легко масштабувати вебзастосунки за допомогою горизонтального розподілу навантаження та балансування;

- розширення – yii2 має багатий набір розширень, які додають новий функціонал до фреймворку [79].

2.4.6 Порівняння фреймворків та обґрунтування вибору

Нижче наведено порівняльну таблицю, що показує пряме порівняння між такими фреймворками як Spring Boot, Nest.js, Express.js, ASP.NET Core, SiteCore, Flask, Django, Laravel, yii2 (табл. 2.2).

Таблиця 2.2 – Порівняльна таблиця фреймворків

Фреймворк	Популярність	Документація	Сумісність з Angular	Продуктивність	Масштабованість	Якість коду	Надійність
Spring-Boot	Висока	Висока	Низька	Висока	Висока	Висока	Висока
Nest.js	Середня	Висока	Висока	Середня	Висока	Висока	Висока
Express.js	Висока	Середня	Середня	Середня	Середня	Середня	Середня
ASP.NET Core	Висока	Висока	Висока	Висока	Висока	Висока	Висока
Site-Core	Низька	Середня	Низька	Середня	Середня	Середня	Середня
Flask	Висока	Висока	Низька	Низька	Середня	Середня	Середня
Django	Висока	Висока	Середня	Низька	Висока	Висока	Висока
Laravel	Висока	Висока	Низька	Низька	Висока	Висока	Висока
yii2	Середня	Середня	Низька	Низька	Середня	Середня	Середня

Для порівняння було враховано такі фактори: сумісність з клієнтським фреймворком Angular, продуктивність, масштабованість, якість коду та надійність.

З усіх розглянутих фреймворків найкращим вибором для Angular є Nest.js.

Nest.js використовує TypeScript, який є основною мовою програмування для Angular. Це спрощує розробку та підтримку коду, а також забезпечує строгу типізацію та помилковість.

Nest.js має схожу архітектуру з Angular, яка базується на модулях, сервісах, контролерах та інжекторах. Це дозволяє легко інтегрувати серверну та клієнтську частини вебзастосунку.

Nest.js підтримує розширення за допомогою багатьох бібліотек та інструментів, які можуть бути корисними для розробки вебзастосунку. Наприклад, Nest.js може використовувати GraphQL для створення гнучких та ефективних API, WebSocket для реалізації двосторонньої комунікації між сервером та клієнтом, Swagger для документування API та інші.

Nest.js дозволяє створювати ефективні, надійні та масштабовані серверні застосунки на Node.js. Nest.js використовує подійну модель та неблокуючий ввід-вивод Node.js, які забезпечують високу продуктивність та паралельну обробку запитів. Nest.js також дозволяє легко масштабувати вебзастосунок за допомогою кластеризації, мікросервісної архітектури та інших методик.

Nest.js має високий рівень якості коду та надійності, оскільки дотримується принципів SOLID та RESTful API, а також забезпечує строгу типізацію та помилковість.

2.5 Огляд популярних баз даних для серверної частини застосунку

База даних – це логічно організована сукупність інформації, або даних, яка зберігається на комп'ютері. База даних дозволяє зберігати, обробляти та отримувати дані за різними критеріями. Для роботи з базами даних використовуються спеціальні програми – системи керування базами даних (СКБД).

СКБД можна класифікувати за різними параметрами, такими як тип даних, модель даних, мова запитів, архітектура тощо. Буде розглянуто та

порівняно п'ять популярних СКБД для серверної частини застосунку: Mongo DB, SQLite, PostgreSQL, MySQL та Redis [80].

2.5.1 Mongo DB

Mongo DB – це одна з найпопулярніших СКБД типу NoSQL, яка працює з документами у форматі JSON. Mongo DB використовує модель даних без схеми, що означає, що структура документів може бути різною в межах однієї колекції. Mongo DB підтримує мову запитів MongoDB Query Language (MQL), яка дозволяє виконувати складні операції над даними. Mongo DB підтримує транзакції на рівні документа та колекції. Mongo DB має високу продуктивність завдяки індексації, шардингу (розподілу даних по різних серверах) та реплікації (створенню копій даних на резервних серверах). Mongo DB є відкритим програмним забезпеченням і ліцензується за умовами Server Side Public License (SSPL) [81, 82].

2.5.2 SQLite

SQLite – це легка СКБД типу SQL, яка зберігає дані у вигляді файлу на локальному диску. SQLite є реляційною базою даних, тому структура таблиць і стовпців повинна бути визначена заздалегідь. SQLite підтримує мову запитів SQL (Structured Query Language), яка є стандартом для роботи з реляційними базами даних. SQLite підтримує транзакції на рівні бази даних. SQLite має низьку продуктивність у порівнянні з іншими СКБД, але вигідно відрізняється своєю простотою, портативністю та незалежністю від сервера. SQLite є відкритим програмним забезпеченням і ліцензується за умовами Public Domain [81], [83].

2.5.3 PostgreSQL

PostgreSQL – це потужна СКБД типу SQL, яка працює з реляційними таблицями та рядками. PostgreSQL використовує модель даних з схемою, але також дозволяє зберігати дані у вигляді JSON, XML, геометричних об'єктів тощо. PostgreSQL підтримує мову запитів SQL, а також розширення PL/pgSQL, PL/Python, PL/Perl та інші. PostgreSQL підтримує транзакції на рівні бази даних, таблиць та рядків. PostgreSQL має високу продуктивність завдяки індексації, шардингу, реплікації та кластеризації. PostgreSQL є відкритим програмним забезпеченням і ліцензується за умовами PostgreSQL License [83].

2.5.4 MySQL

MySQL – це одна з найпопулярніших СКБД типу SQL, яка працює з реляційними таблицями та рядками. MySQL використовує модель даних з схемою, але також дозволяє зберігати дані у вигляді JSON. MySQL підтримує мову запитів SQL, а також розширення MySQL Stored Procedure Language (SPL). MySQL підтримує транзакції на рівні бази даних та таблиць. MySQL має високу продуктивність завдяки індексації, шардингу, реплікації та кешуванню. MySQL є відкритим програмним забезпеченням і ліцензується за умовами GNU General Public License (GPL) [81], [83].

2.5.5 Redis

Redis – це швидка key-value колекція даних типу NoSQL. Redis використовує модель даних без схеми, що означає, що структура даних може бути різною для кожного ключа. Redis підтримує мову запитів Redis Command Language (RCL), яка дозволяє виконувати прості операції над даними. Redis не підтримує транзакції, але гарантує атомарність окремих команд. Redis має

надвисоку продуктивність завдяки тому, що зберігає дані у оперативній пам'яті, а не на диску. Redis є відкритим програмним забезпеченням і ліцензується за умовами BSD License [84].

2.5.6 Порівняння баз даних та обґрунтування вибору

Нижче наведено порівняльну таблицю, що показує пряме порівняння між такими базами даних як Mongo DB, SQLite, PostgreSQL, MySQL та Redis (табл. 2.3).

Таблиця 2.3 – Порівняльна таблиця баз даних

Характеристика	Бази даних				
	Mongo DB	SQLite	PostgreSQL	MySQL	Redis
Тип даних	NoSQL	SQL	SQL	SQL	NoSQL
Модель даних	Документи JSON	Таблиці та рядки	Таблиці та рядки, JSON, XML, геометричні об'єкти тощо.	Таблиці та рядки, JSON	Ключ-значення
Мова запитів	MQL	SQL	SQL, PL/pgSQL, PL/Python, PL/Perl та інші.	SQL, SPL	RCL
Підтримка транзакцій	На рівні документа та колекції.	На рівні бази даних.	На рівні бази даних, таблиць та рядків.	На рівні бази даних та таблиць.	Немає.
Продуктивність	Висока	Середня	Висока	Висока	Надвисока
Масштабованість	Шардинг, реплікація.	Портативність, незалежність від сервера.	Шардинг, реплікація, кластеризація.	Шардинг, реплікація, кешування.	Реплікація, кластеризація.
Ліцензування	SSPL	Public Domain	PostgreSQL License	GPL	BSD License

З таблиці видно, що кожна СКБД має свої переваги та недоліки, і вибір залежить від конкретних потреб та цілей застосунку.

Логічним вибором було б зробити PostgreSQL, оскільки він має одну з найвищих продуктивностей, максимальний можливий функціонал, а також високу безпеку. Однак було обрано простішу версію бази даних – SQLite.

Основними причинами цього вибору є:

- SQLite є легкою та простою СКБД, яка не потребує окремого сервера або конфігурації;
- SQLite підтримує мову запитів SQL, яка є стандартом для роботи з реляційними базами даних;
- SQLite добре інтегрується з Nest.js та Angular за допомогою бібліотек TypeORM та SQLite3;
- SQLite має низькі вимоги до ресурсів та пам'яті, що позитивно впливає на швидкодію застосунку;
- SQLite є вільним програмним забезпеченням і ліцензується за умовами Public Domain, що не обмежує його використання.

Таким чином, можна зробити висновок, що SQLite є оптимальною СКБД для застосунку.

2.6 Остаточний вибір стеку технологій та його обґрунтування

Архітектура включає в себе організацію компонентів та інфраструктури для забезпечення ефективної роботи.

Фронтенд буде розроблено на фреймворку Angular, а його структура організована в модулі та компонентах. Кожен компонент відповідає за певну частину інтерфейсу та може включати в себе шаблон, логіку та стилізацію.

Angular сервіси будуть використовуватися для виконання HTTP-запитів до серверної частини, збереження глобальних даних та обробки логіки, яка не повинна бути прив'язана до конкретного компонента.

Застосунок буде використовувати маршрутизацію Angular для навігації між сторінками та компонентами.

Серверна частина буде побудована за допомогою фреймворку Nest.js і організована у модулях та контролерах. Контролери відповідають за обробку HTTP-запитів та взаємодію з базою даних, в той час як модулі групують схожі функціональність.

Серверна логіка, що не пов'язана з обробкою HTTP-запитів, буде реалізовуватися в сервісах та провайдерах, які можуть бути використані з різних контролерів та модулів.

Для взаємодії з базою даних буде використовуватися Prisma, яка надає ORM-інтерфейс для роботи з базою даних. Схема бази даних буде визначена за допомогою Prisma моделей, і завдяки цьому можливе зручне створення, оновлення та опитування даних.

SQLite обрана як легка реляційна база даних для зберігання даних застосунку, бо вона є відмінним варіантом для маленьких та середніх проєктів.

Фронтенд та серверна частина будуть взаємодіяти за допомогою HTTP-запитів, де серверна частина обробляє запити та відправляє відповіді, а фронтенд відображає інформацію користувачу.

Nest.js використовує Prisma для взаємодії з базою даних SQLite, що дозволяє зберігати та отримувати дані.

Для забезпечення безпеки застосунку можуть використовуватися різні методи, такі як аутентифікація та авторизація користувачів та застосування кращих практик забезпечення.

Застосунок може бути піддано моніторингу за допомогою спеціалізованих інструментів, що дозволяють відстежувати роботу застосунку та виявляти помилки та завантаження.

Ця архітектури демонструє взаємодію компонентів та технологій у стеці Angular, Nest.js, Prisma та SQLite для створення робочого застосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Функціональні можливості програми

На рисунку 3.1 наведено діаграму прецедентів, що визначає майбутній функціонал застосунка «NutriLight».

На рисунку 3.1 наведено діаграму прецедентів, що демонструє основні можливості застосунку.



Рисунок 3.1 – Use Case діаграма основних можливостей застосунку

Нижче наведено етапи основного потоку подій для користувача.

Основний виконавець (актор) – користувач.

Потоки подій.

Етапи основного потоку подій:

а) налаштування програми:

- 1) перемикання теми;
- 2) перемикання мови;

б) робота з акаунтом:

- 1) авторизація;

- 2) реєстрація;
- 3) зміна даних аккаунту;

в) робота з профілем:

- 1) налаштування профілю;
- 2) налаштування макронутрієнтів;

г) робота з підрахунком калорій:

- 1) відстеження гідратації людини;
- 2) перегляд балансу:
 - перегляд спалених та спожитих калорій за день/тиждень;
 - перегляд балансу калорій за день/тиждень;
- 3) робота з харчуванням:
 - робота з продуктами з бази даних:
 - додавання/видалення;
 - робота з кастомними продуктами:
 - додавання/видалення;
 - перегляд харчування за день/тиждень;
- 4) робота з активностями:
 - перегляд активностей за день/тиждень;
 - додавання/видалення;

д) робота з вимірюваннями:

- 1) перегляд графіку ваги;
- 2) менеджмент категоріями для вимірювань:
 - додавання/видалення;
- 3) відстеження вимірювань:
 - додавання/видалення;
- 4) відстеження маси тіла спортсмена:
 - додавання/видалення;

е) перегляд аналізу результатів:

- 1) розрахунок рекомпозиції тіла за заданими параметрами;

- 2) перегляд графіку гідратації;
- 3) перегляд графіку активностей;
- 4) перегляд графіку харчування;
- 5) перегляд графіку балансу калорій.

3.2 Алгоритми розробленої програми

Під час розробки застосунку було реалізовано наступні алгоритми:

- алгоритм реєстрації нового користувача;
- алгоритм авторизації користувача;
- алгоритм перемикання мовного інтерфейсу;
- алгоритм перемикання між темами оформлення;
- алгоритм збереження та оновлення основних особистих даних користувача;
- алгоритм введення та збереження значень макронутрієнтів;
- алгоритм автоматичного розрахунку кількості спожитих калорій на основі введених даних;
- алгоритм відстеження та обліку спожитих та спалених калорій за день/тиждень;
- алгоритм відстеження та обліку балансу калорій;
- алгоритм додавання та видалення кількості споживаної води;
- алгоритм додавання/видалення продуктів до/з щоденника харчування;
- алгоритм додавання/видалення продуктів до/з бази даних;
- алгоритм додавання/видалення активностей;
- алгоритм виведення інформації про харчування та активності за тиждень;
- алгоритм додавання/видалення ваги та вимірювань;
- алгоритм побудови графіків ваги, вимірювань, балансу калорій, харчування, активностей та гідратації за обраний період часу;

– алгоритм розрахунку та виведення рекомендацій щодо рекомпозиції тіла користувача.

На рисунках 3.2-3.3 наведено декілька алгоритмів, що відображено згідно стандарту ISO 5807-85 [89].

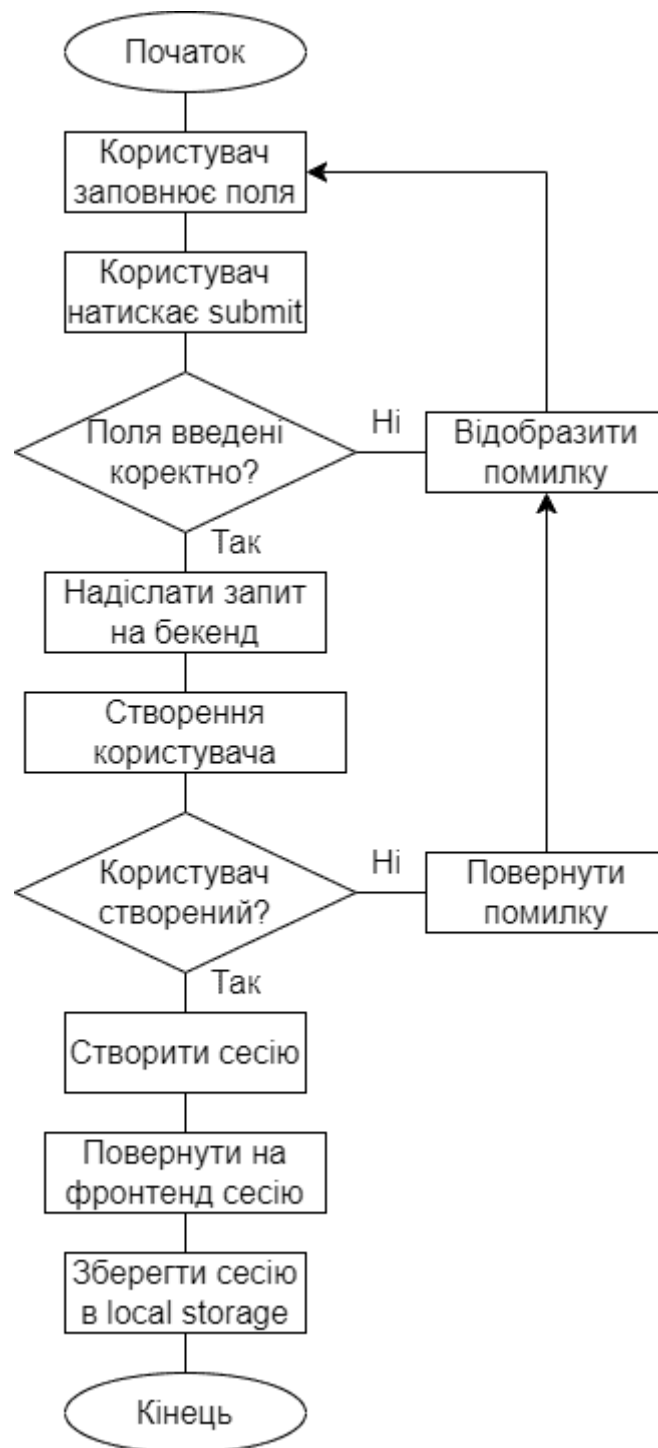


Рисунок 3.2 – Алгоритм реєстрації

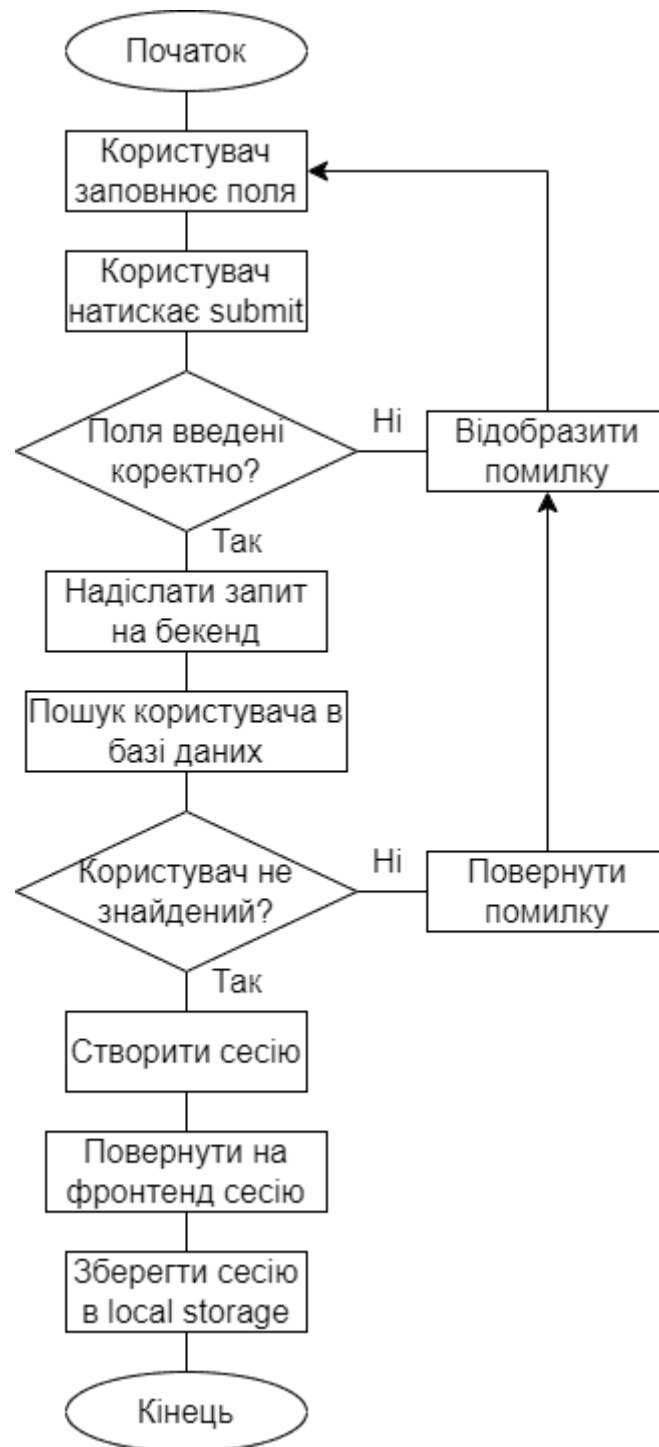


Рисунок 3.3 – Алгоритм авторизації

3.3 Навігація між сторінками застосунку

Контролери у Nest.js відповідають за обробку вхідних запитів і повернення відповідей клієнту.

У angular програмі використовується routing для навігації між сторінками, далі на рисунку 3.4 представлена UML діаграма діяльності, де відображена навігація з клієнтської сторони програми.

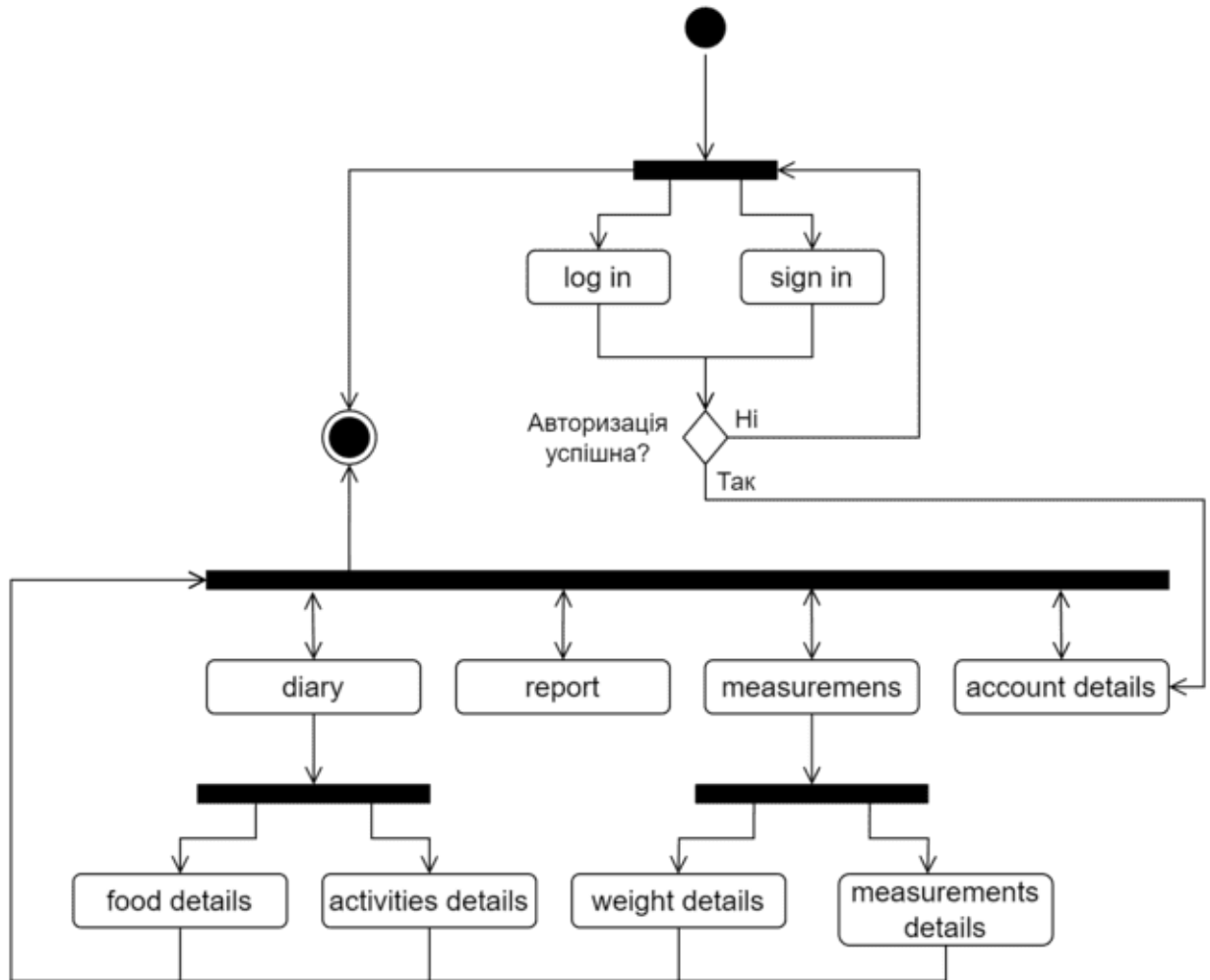


Рисунок 3.4 – Діаграма навігації з клієнтської сторони програми

3.4 Архітектура залежностей програми

Залежності – бібліотеки, які підключаються до Node.JS застосунку. Вони поділяються на `dependencies` і `devDependencies`. Обидва підключаються до програмного застосунку Node.JS. Однак різниця полягає в тому, що `dependencies` використовуються у вже скомпільованій програмі, тоді як `devDependencies` – лише під час розробки.

3.4.1 Архітектура залежностей для Backend

До основних залежностей (dependencies) належать:

- @nestjs/common – основний модуль NestJS, який надає основні функції для створення застосунків, включаючи декоратори та впровадження залежностей;
- @nestjs/core – ядро фреймворка NestJS, що забезпечує базові механізми управління залежностями та життєвим циклом застосунка;
- @nestjs/platform-express – модуль NestJS, який забезпечує інтеграцію з Express для створення вебзастосунків;
- @prisma/client – клієнт Prisma, що надає типобезпечний API для взаємодії з базою даних через Prisma ORM;
- reflect-metadata – пакет, який дозволяє працювати з метаданими в JavaScript / TypeScript, особливо корисний при використанні декораторів;
- rxjs – бібліотека для роботи з асинхронними послідовностями, що базується на концепції Observable;
- uuid – пакет для генерації унікальних ідентифікаторів, часто використовується для створення унікальних ключів.
- До залежності розробки (devDependencies) належать:
 - @nestjs/cli – командний рядок для управління проєктами NestJS;
 - @nestjs/schematics – інструменти генерації коду (schematics) для прискорення розробки в NestJS;
 - @nestjs/testing – пакет для тестування застосунків NestJS;
 - @types/express – набір типів TypeScript для фреймворка Express, що дозволяє використовувати Express з підтримкою статичної типізації;
 - @types/jest – набір типів TypeScript для фреймворка тестування Jest;
 - @types/node – набір типів TypeScript для Node.js, який забезпечує типобезпечну взаємодію з API Node.js;
 - @types/supertest – набір типів TypeScript для бібліотеки Supertest, що використовується для тестування HTTP-запитів;

- jest – фреймворк для тестування JavaScript / TypeScript, який надає набір функцій для написання та запуску тестів;
- prettier – інструмент для автоматичного форматування коду, сумісний з багатьма мовами програмування;
- prisma – ORM для взаємодії з базою даних, що надає зручні інструменти для роботи з даними;
- source-map-support – пакет, який покращує підтримку Source Maps в Node.js для кращого відстеження помилок та налагодження;
- supertest, ts-jest, ts-loader, ts-node, tsconfig-paths, typescript – інструменти для збірки, тестування та запуску коду TypeScript під час розробки. ts-jest, ts-loader, ts-node і typescript забезпечують підтримку TypeScript, а tsconfig-paths керує шляхами до файлів у проєкті. supertest використовується для тестування HTTP-запитів у застосунку.

3.4.2 Архітектура залежностей для Frontend

До основних залежностей (dependencies) належать:

- @angular/animations – модуль Angular, що надає можливості для додавання анімацій у застосунок Angular. Дозволяє створювати анімації при зміні стилів, додаванні та видаленні компонентів та інших елементів інтерфейсу;
- @angular/cdk (Component Dev Kit) – набір інструментів для створення користувацького інтерфейсу в Angular. Надає компоненти та утиліти, які можна використовувати для побудови складних та інтерактивних користувацьких інтерфейсів;
- @angular/common – модуль Angular, що включає загальні сервіси, директиви та фільтри, які використовуються при розробці застосунків Angular;
- @angular/compiler – модуль Angular, що містить інструменти для компіляції шаблонів та коду в застосунку Angular. Використовується для

перетворення шаблонів HTML та TypeScript коду в JavaScript код, який може виконувати браузер;

- @angular/core – основний модуль Angular, що містить базові функції та сервіси, необхідні для роботи всього застосунка. Тут визначені компоненти, директиви, сервіси та інші базові інструменти Angular;

- @angular/forms – модуль Angular, що надає інструменти для роботи з формами в застосунку. Цей модуль включає директиви, сервіси та інші інструменти для управління станом та валідацією форм;

- @angular/material – бібліотека Angular, що надає набір готових компонентів Material Design для побудови користувацького інтерфейсу в стилі Material Design;

- @angular/platform-browser – модуль Angular, що містить інструменти, необхідні для взаємодії застосунка Angular з браузером, такі як сервіси для завантаження застосунка в браузері та обробки різних подій;

- @angular/platform-browser-dynamic – модуль Angular, що містить інструменти для динамічного завантаження та компіляції Angular-застосунків у браузері;

- @angular/router – модуль Angular, що надає маршрутизацію в застосунку. Дозволяє визначати маршрути, виконувати навігацію по різних представленнях та керувати їх відображенням;

- @ngx-translate/core, @ngx-translate/http-loader – пакети для додавання підтримки мультимовності в застосунки Angular через переклади та завантажувачі HTTP;

- echarts – бібліотека для створення інтерактивних графіків та діаграм;

- moment – бібліотека для зручної роботи з датами та часом;

- ngx-echarts – обгортка для інтеграції ECharts в застосунки Angular;

- rxjs – бібліотека для роботи з асинхронними послідовностями в Angular;

- tslib – бібліотека з функціями, які допомагають TypeScript при оптимізації коду;

– zone.js – бібліотека для зон в JavaScript, що використовується в Angular для керування асинхронними операціями.

До залежності розробки (devDependencies) належать:

– @angular-devkit/build-angular – інструменти збірки для Angular, що надають можливості компіляції та збірки застосунка;

– @angular/cli – командний рядок для керування проектами Angular;

– @angular/compiler-cli – компілятор TypeScript для Angular;

– @types/jasmine – набір типів TypeScript для фреймворка тестування Jasmine;

– jasmine-core – основний модуль Jasmine для запуску тестів;

– karma – фреймворк для запуску тестів JavaScript у різних браузерах;

– karma-chrome-launcher – плагін для запуску тестів у браузері Google Chrome;

– karma-coverage – плагін для отримання звітів про покриття коду тестами;

– karma-jasmine – плагін для інтеграції Karma з Jasmine;

– karma-jasmine-html-reporter – плагін для створення красивих звітів про виконання тестів;

– prettier – інструмент для автоматичного форматування коду з урахуванням визначених правил;

– typescript – мова програмування TypeScript, що використовується для розробки Angular застосунків.

3.5 Файлова архітектура програми

3.5.1 Файлова архітектура Backend

3.5.1.1 Опис підходу до структуризації файлів

Розроблена файлова архітектура структурована, що сприяє читабельності та розширюваності програмного коду. Основні принципи підходу до структуризації включають:

- розділення на компоненти. Файли розділені на різні компоненти, такі як "Account", "Profile", "Activities", "Food", "Measurements" та "Shared". Це допомагає уявити різні області функціональності програми;
- модульність. Кожен компонент має свій власний модуль, який містить контролери, сервіси та інші компоненти, які необхідні для його роботи. Це сприяє гарній організації коду та уникненню змішування функціональності різних частин програми;
- підкаталоги для DTO (Data Transfer Objects). DTO є окремими об'єктами, які передають дані між сервісами і компонентами програми. Вони організовані в окремих підкаталогах для кожного компонента, що робить їх легко знаходжуваними;
- загальні служби та утиліти. У компоненті "Shared" розміщені загальні служби, утиліти та константи, які можуть використовуватися в різних частинах програми. Це дозволяє уникнути дублювання коду та забезпечує централізований доступ до загальних ресурсів;
- назви файлів і підкаталогів. Назви файлів і підкаталогів описують їхню функціональність. Це допомагає швидко зрозуміти, що саме міститься у відповідному файлі або підкаталозі.

3.5.1.2 Опис побудованих модулів

DataBase module, що зображено на рисунку 3.5. Цей модуль містить сервіси, що забезпечують взаємодію з базою даних, включаючи роботу з

різними типами даних, такими як профілі користувачів, активності, їжа, вага, об'єкти вимірювань тощо. Він також містить файли, пов'язані з підключенням до бази даних.

```

data-base
├── db-services
│   ├── account-db.service.ts
│   ├── activity-db.service.ts
│   ├── auto-complete-food-db.service.ts
│   ├── caloricity-tdee-db.service.ts
│   ├── category-db.service.ts
│   ├── food-db.service.ts
│   ├── measurement-db.service.ts
│   ├── measurement-object-db.service.ts
│   ├── micro-nutrient.service.ts
│   ├── preset-activity-db.service.ts
│   ├── profile-db.service.ts
│   ├── session-db.service.ts
│   ├── sub-category-db.service.ts
│   ├── water-db.service.ts
│   └── weight-db.service.ts
├── prisma
│   └── prisma.service.ts
└── data-base.module.ts
  
```

Рисунок 3.5 – DataBase module

Account module, що зображено на рисунку 3.6. Модуль відповідає за операції, пов'язані з обліковими записами користувачів. Включає контролер, сервіс та DTO (Data Transfer Objects) для реєстрації, входу, зміни та отримання даних про обліковий запис користувача.

```

account
├── dto
│   ├── account-dto.interface.ts
│   ├── change-account-dto.interface.ts
│   ├── log-in-account-dto.interface.ts
│   └── sign-up-account-dto.interface.ts
├── account.controller.ts
├── account.module.ts
└── account.service.ts
  
```

Рисунок 3.6 – Account module

Profile module, що зображено на рисунку 3.7. Відповідає за управління профілями користувачів, встановлення загальної щоденної витрати енергії та оновлення профілю. Включає контролер, сервіс та DTO для створення, оновлення та отримання даних профілю користувача.

```
profile
├── dto
│   ├── create-profile-dto.interface.ts
│   ├── profile-with-caloricity-dto.interface.ts
│   ├── update-caloricity-dto.interface.ts
│   └── update-profile-dto.interface.ts
├── profile.controller.ts
├── profile.module.ts
└── profile.service.ts
```

Рисунок 3.7 – Profile module

Activities module, що зображено на рисунку 3.8. Модуль для операцій, пов'язаних з активностями користувачів. Включає контролер, сервіс та DTO для створення, видалення та отримання даних про активності користувача.

```
activities
├── dto
│   ├── create-activity-dto.interface.ts
│   ├── create-preset-activity-dto.interface.ts
│   ├── delete-activity-dto.interface.ts
│   ├── delete-preset-activity-dto.interface.ts
│   └── get-activity-dto.interface.ts
├── activities.controller.ts
├── activities.module.ts
└── activities.service.ts
```

Рисунок 3.8 – Activities module

Food module, що зображено на рисунку 3.9. Відповідає за управління стравами користувачів, включаючи автозаповнення та основні дані щодо страв. Включає контролер, сервіс та DTO для створення, видалення та управління даними про продукти.

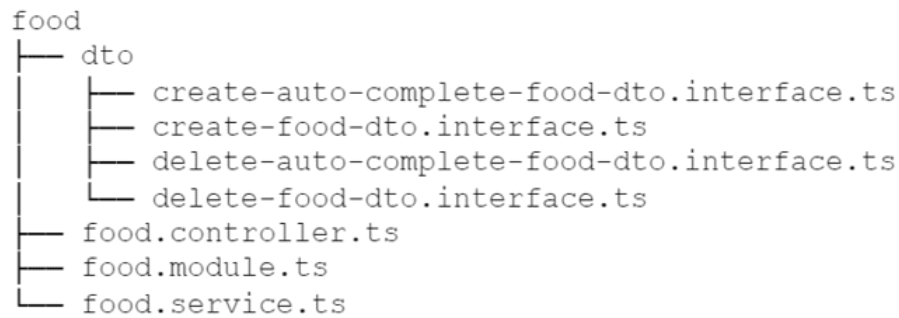


Рисунок 3.9 – Food module

Measurements module, що зображено на рисунку 3.10. Модуль для операцій, пов'язаних з вимірюваннями, включаючи вагу та об'єкти вимірювань. Включає контролер, сервіс та DTO для створення, видалення та управління даними про вимірювання.

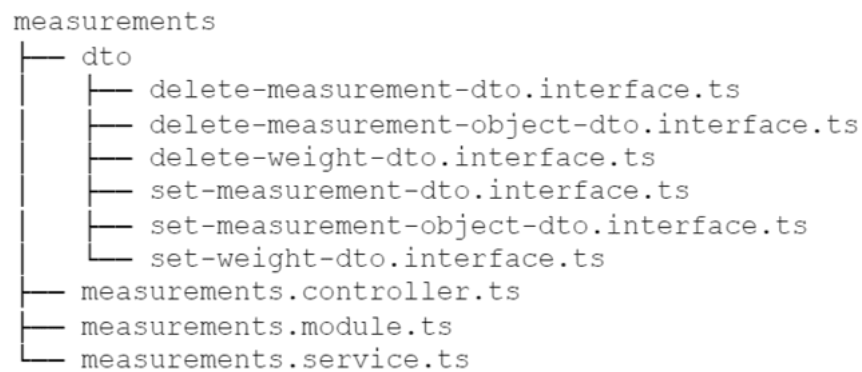


Рисунок 3.10 – Measurements module

Water module, що зображено на рисунку 3.11. Відповідає за операції з водопиттям користувачів. Містить контролер, сервіс та DTO для створення, видалення та отримання даних про водопиття.

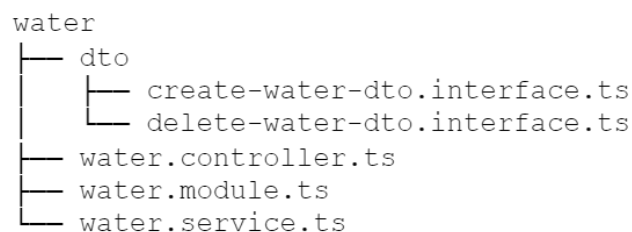


Рисунок 3.11 – Water module

Shared module, що зображено на рисунку 3.12. Модуль, що містить загальні константи, сервіси та утиліти, які можуть використовуватися в різних частинах застосунку, наприклад, константи помилок, сервіс аутентифікації тощо. Також включає модуль, який може бути спільно використаний у різних частинах застосунку.

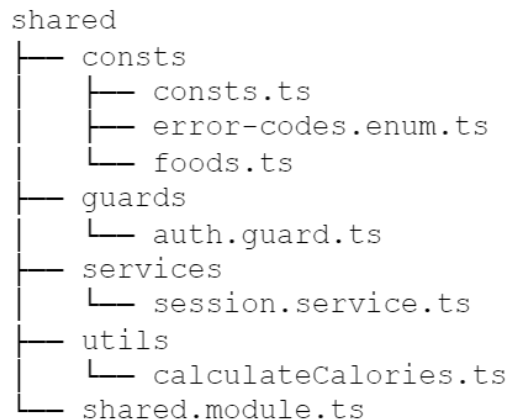


Рисунок 3.12 – Shared module

У цілому, ця файлова структура має гарну організацію, яка полегшує сприйняття програми та дозволяє легше управляти та розвивати її у майбутньому. Кожен з модулів відповідає за різні аспекти функціональності та операцій в системі.

3.5.2 Файлова архітектура Frontend

3.5.2.1 Опис підходу до структуризації файлів

Підхід до реалізації структури проєкту базується на модульності, розділенні функціональності та компонентів на логічні частини, що спрощує розробку, тестування та підтримку програмного продукту. Основні принципи цього підходу включають:

- розділення на модулі. Кожен функціональний блок програми (наприклад, облік користувачів, щоденник, вимірювання) має свій власний

модуль. Це сприяє підтримці незалежних частин системи, полегшує розширення та впровадження нових функцій;

- компонентно-орієнтований підхід. Кожен модуль складається з компонентів, які відповідають за різні частини функціоналу. Це дозволяє зберігати код локалізовано та організовано;

- спільний модуль. Існує загальний модуль (Shared module), де зберігаються ресурси, які використовуються у різних частинах програми, такі як загальні сервіси, компоненти, пайпи, константи та інші корисні елементи;

- структурованість коду. Файли розділяються на логічні підкаталоги та найменування, відображаючи їх призначення. Наприклад, у каталозі компонентів є підкаталоги для кожного окремого компонента, де зберігаються файли шаблону, стилів та коду компонента;

- ієрархія та логіка папок. Кожен модуль чи підмодуль має власну структуру каталогів, що відображає його функціональність. Наприклад, в модулі вимірювань (measurements) є папки для форм, деталей, основних компонентів тощо;

- легка розширюваність. Кожен модуль побудований таким чином, щоб його можна було легко розширювати або модифікувати без необхідності вносити зміни в інші частини системи;

- загальна структура та конвенції. Чітко визначені конвенції та стандарти назв файлів та структури каталогів для полегшення розуміння та співпраці між розробниками;

- реюзабельність коду. Створення компонентів та сервісів таким чином, щоб їх можна було легко використовувати в інших частинах програми чи навіть у різних проєктах.

Цей підхід допомагає зберігати код організованим, структурованим та легким для розуміння, що є важливими аспектами для розробки великих програмних продуктів.

3.5.2.2 Опис побудованих модулів

Account module, що зображено на рисунку 3.13. Модуль містить компоненти, які відповідають за управління обліковими записами користувачів і їх деталями, а також компоненти для авторизації та реєстрації в системі.

```

account
├── account.module.ts
├── auth.style.scss
├── details
│   ├── details.component.html
│   ├── details.component.scss
│   ├── details.component.ts
│   ├── profile-form
│   │   ├── profile-form.component.html
│   │   ├── profile-form.component.scss
│   │   └── profile-form.component.ts
│   ├── account-form
│   │   ├── account-form.component.html
│   │   ├── account-form.component.scss
│   │   └── account-form.component.ts
│   ├── macro-nutrients-form
│   │   ├── macro-nutrients-form.component.html
│   │   ├── macro-nutrients-form.component.scss
│   │   └── macro-nutrients-form.component.ts
│   └── sub-form.style.scss
├── login
│   ├── login.component.html
│   ├── login.component.scss
│   └── login.component.ts
└── signup
    ├── signup.component.html
    ├── signup.component.scss
    └── signup.component.ts
  
```

Рисунок 3.13 – Account module з клієнтської сторони програми

Diary module, що зображено на рисунку 3.14. Модуль відповідає за ведення щоденника, де користувач може вести записи про свої активності, спожиту їжу, спожиту воду та інші замітки. Включає компоненти для реєстрації активностей та харчування, налаштування цілей з калорій та поживних речовин, а також компоненти для відображення статистики

спожитих калорій, нутрієнтів та іншої інформації. Модуль об'єднує необхідні компоненти для ведення щоденного журналу та аналізу даних щодо харчування та активності.

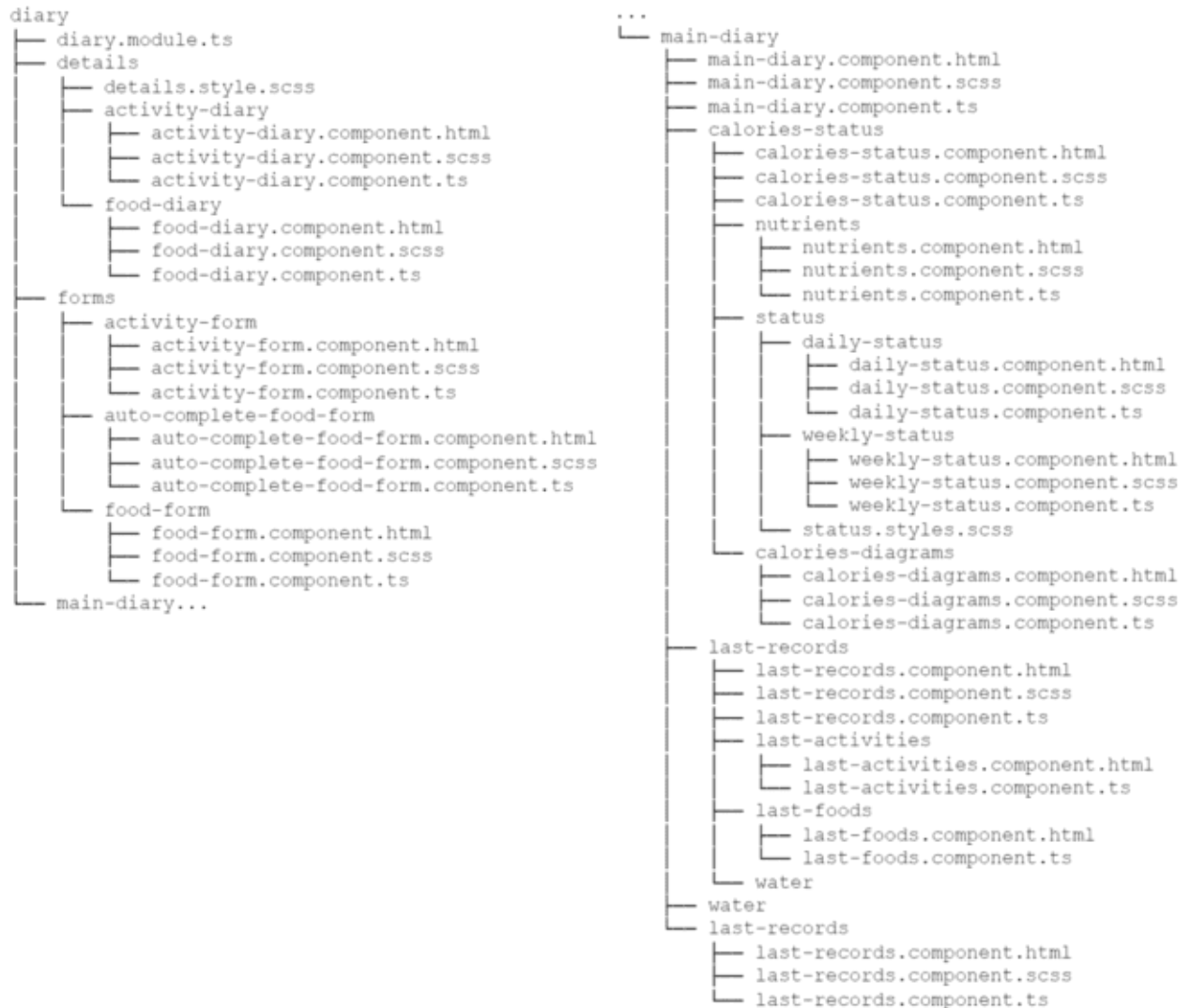


Рисунок 3.14 – Diary module

Measurements module, що зображено на рисунку 3.15. Модуль призначений для зберігання та обробки даних про вимірювання. Включає компоненти для ведення вимірювань, такі як вага та інші параметри, а також форми для введення цих даних. Окрім цього, модуль містить основний компонент, який узагальнює та відображає основні дані з вимірювань. Він забезпечує можливість редагування, видалення та додавання нових записів про вимірювання та їх обробку.

```

measurements
├── measurements.module.ts
├── details-measurements
│   ├── details-measurements.component.html
│   ├── details-measurements.component.scss
│   └── details-measurements.component.ts
├── forms
│   ├── measurement-form
│   │   ├── measurement-form.component.html
│   │   ├── measurement-form.component.scss
│   │   └── measurement-form.component.ts
│   ├── measurement-object-form
│   │   ├── measurement-object-form.component.html
│   │   ├── measurement-object-form.component.scss
│   │   └── measurement-object-form.component.ts
│   └── weight-form
│       ├── weight-form.component.html
│       ├── weight-form.component.scss
│       └── weight-form.component.ts
└── main-measurements
    ├── main-measurements.component.html
    ├── main-measurements.component.scss
    └── main-measurements.component.ts

```

Рисунок 3.15 – Measurements module з клієнтської сторони програми

Modals module, що зображено на рисунку 3.16. Модуль відповідає за управління модальними вікнами та відображенням спливаючих повідомлень для користувача. Він містить компоненти, які відповідають за окремі типи модальних вікон, такі як "push-notification".

```

modals
├── modals.component.html
├── modals.component.scss
├── modals.component.ts
├── modals.module.ts
└── push-notification
    ├── push-notification.component.html
    ├── push-notification.component.scss
    └── push-notification.component.ts

```

Рисунок 3.16 – Modals module

Navigation module, що зображено на рисунку 3.17. Модуль відповідає за організацію та управління навігаційними елементами, такими як header та footer. У цьому модулі містяться ці компоненти.

```

navigation
├── navigation.module.ts
├── footer
│   ├── footer.component.html
│   ├── footer.component.scss
│   └── footer.component.ts
└── header
    ├── header.component.html
    ├── header.component.scss
    └── header.component.ts

```

Рисунок 3.17 – Navigation module

Report module, що зображено на рисунку 3.18. Модуль відповідає за створення звітів та формування відповідної інформації для користувачів. У ньому містяться компоненти, що допомагають відображати різноманітну інформацію: "report.component", який відповідає за загальний звіт, та "recomposition-form.component", який дозволяє формувати результат щодо рекомпозиції.

```

report
├── report.module.ts
├── recomposition-form
│   ├── recomposition-form.component.html
│   ├── recomposition-form.component.scss
│   └── recomposition-form.component.ts
├── report.component.html
├── report.component.scss
└── report.component.ts

```

Рисунок 3.18 – Report module

Shared module. Модуль є центральним місцем для зберігання загальних ресурсів, які використовуються у різних частинах програми. Він містить компоненти, константи, сервіси, інтерсептори, перечислення, гварди, інтерфейси, пайпи та утилітарні функції.

В компонентах розміщені такі компоненти, як `chart.component`, `circle-diagram.component`, та `default-abstract.component`.

Константи включають перерахування різних типів, таких як кольори діаграм, мови, ключі локального сховища, типи сповіщень, теми, результати рекомпозиції та інші.

Гварди – `auth.guard` для захисту маршрутів та інтерсептори – `session-token.interceptor` для обробки токенів сесії при зверненні до серверу.

Інтерфейси включають визначені інтерфейси для об'єктів, які використовуються у взаємодії з API, такі як облікові записи, активності, харчування, виміри, профілі та інші.

Пайпи зберігають функції пайпів, такі як `calculate-max-nutrients.pipe`, `calculate-measure-by-percent.pipe`, та інші, які обробляють дані в шаблонах.

Сервіси – це сінглтон класи, які надають функціональність для роботи з даними, такі як `account.service`, `activities.service`, `foods.service`, `profile.service` та інші, які взаємодіють із сервером чи зберігають стан застосунку.

Утиліти включають утилітарні функції, такі як розрахунок метаболічної швидкості, витрати калорій за активністю, розрахунки для досягнення цілей щодо ваги та інші. На рисунку 3.19 зображено Shared module.

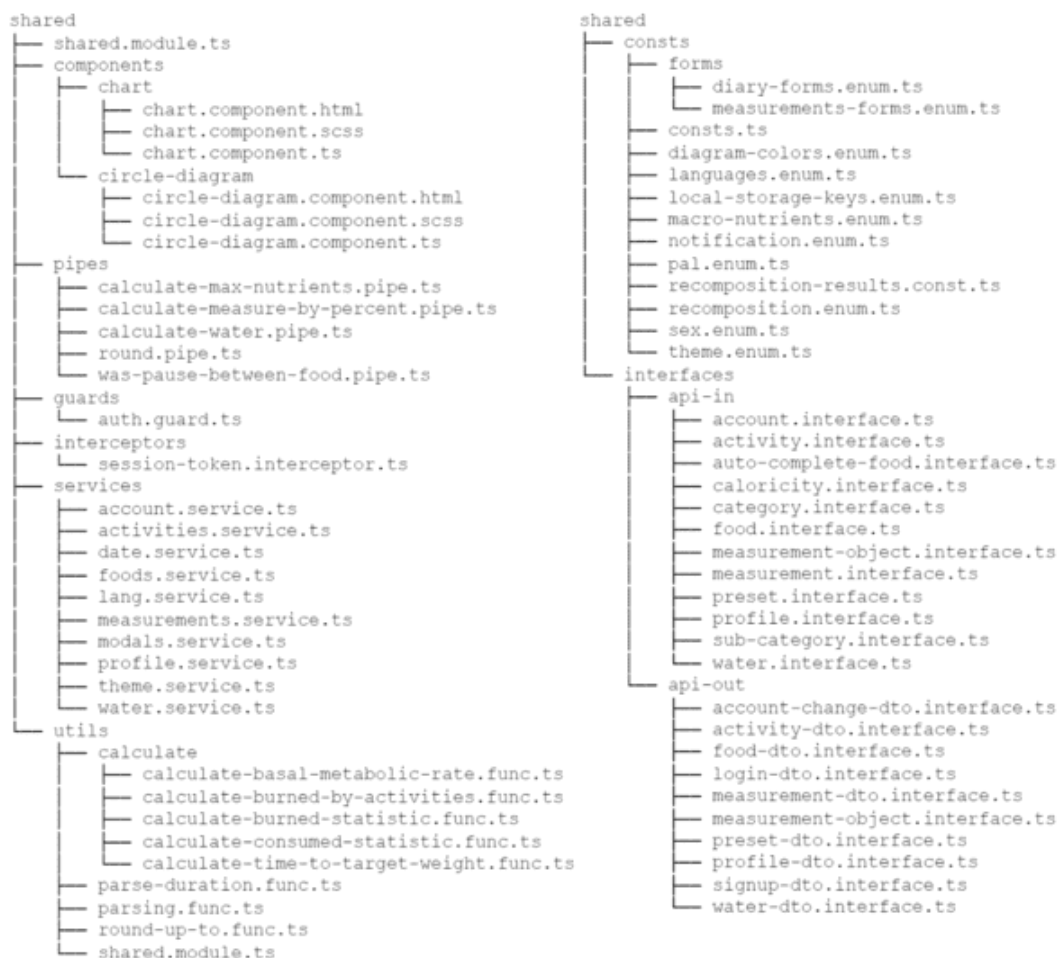


Рисунок 3.19 – Shared module з клієнтської сторони програми

3.6 Основні моделі бази даних

Для програми вибрано базу даних SQLite, проте вся комунікація з нею виконується за допомогою ORM Prisma. Для створення таблиць пишеться файл мовою Prisma, яка визначає структуру даних для зберігання інформації про облікові записи, сесії користувачів, профілі, харчування, активності, воду, вагу та інші параметри здоров'я. Основні моделі бази даних: Account, Session, Profile, Water, Weight, MeasurementObject, Measurement, CaloricityTDEE, Food, AutoCompleteFood, Activity, Category та SubCategory.

Account – модель, що містить інформацію про облікові записи користувачів. Містить такі поля, як ідентифікатор, електронна пошта, ім'я, URL зображення, пароль та зв'язки з сесіями та профілем користувача.

Session – модель, яка представляє сесії користувачів. Містить такі поля, як ідентифікатор сесії, дати створення та закінчення сесії та посилається на обліковий запис користувача.

Profile – модель, яка зберігає інформацію про профілі користувачів, таку як стать, фізичні характеристики, історія ваги, харчування, активності, вода тощо. Також містить зв'язок з обліковим записом користувача.

Water – модель, що відстежує споживання води користувачем. Містить інформацію про дату та об'єм вжитої води, пов'язаної з конкретним профілем.

Weight – модель, яка зберігає дані про вагу користувача в різні моменти часу.

MeasurementObject – модель, яка визначає об'єкти вимірювання, наприклад, частини тіла, та містить зв'язки з вимірюваннями.

Measurement – модель, яка визначає конкретне вимірювання.

CaloricityTDEE – модель, що зберігає дані про калорійність (білки, вуглеводи, жири) та зв'язана з профілем користувача.

Food – модель, яка містить інформацію про споживану їжу користувачем.

AutoCompleteFood – модель, яка надає попередньо заповнену інформацію про їжу для автодоповнення при введенні.

Activity – модель, що зберігає інформацію про фізичні активності користувача.

Category та SubCategory – моделі, що групують активності за категоріями та підкатегоріями.

Кожна модель містить поля, які визначають тип даних та зв'язки з іншими моделями у базі даних. Наприклад, ідентифікатори, рядки, числа, дати та зв'язки "багато до одного" або "один до одного" між об'єктами.

На рисунку 3.20 зображено загальну схему бази даних.

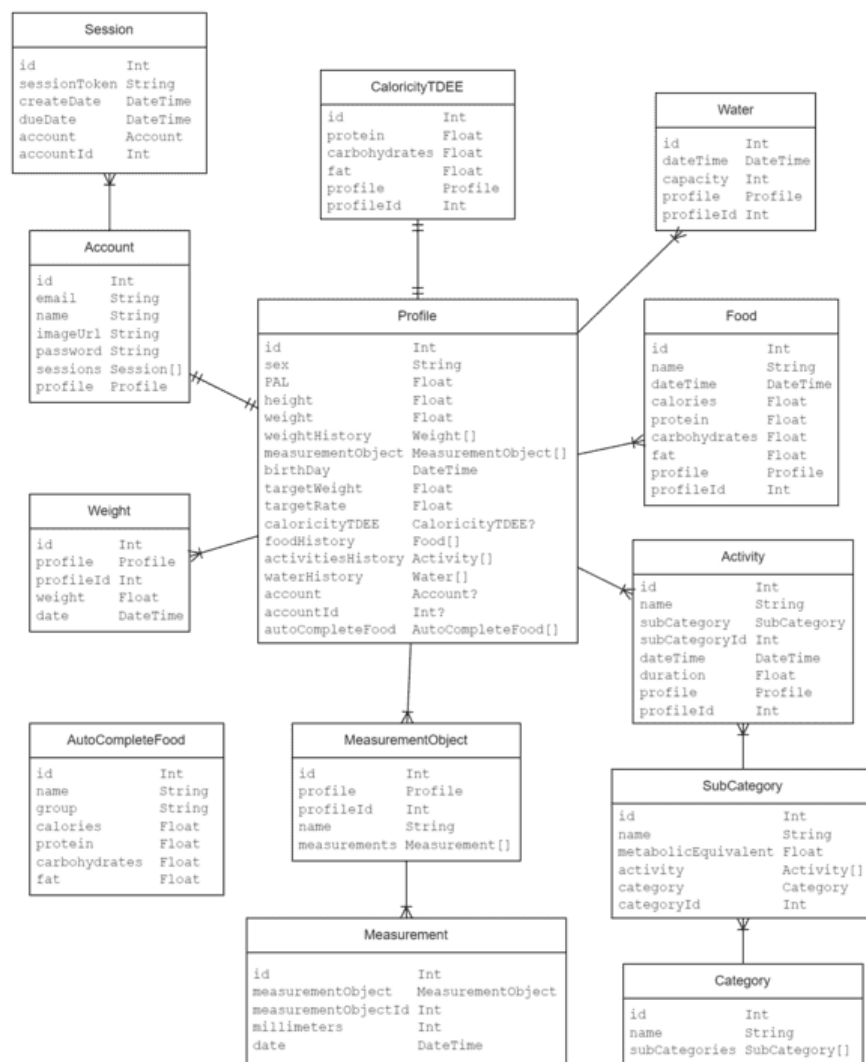


Рисунок 3.20 – Загальна схема бази даних

3.7 API контролери

Контролери у Nest.js відповідають за обробку вхідних запитів і повернення відповідей клієнту.

Метою контролера є отримання конкретних запитів для програми. Механізм маршрутизації контролює, який контролер отримує які запити. Часто кожен контролер має більше одного маршруту, і різні маршрути можуть виконувати різні дії [85].

3.7.1 Account

Account контролер використовується для обробки різних операцій з обліковими записами користувачів. Він має методи для реєстрації, авторизації, зміни облікового запису та виходу:

- `login(body, res)` – `[POST] /account/login` – метод для авторизації користувача за допомогою переданих даних. Повертає статус `HttpStatus.ACCEPTED` у випадку успішної авторизації та відповідь `JSON` з результатом, або `HttpStatus.NOT_FOUND` та код помилки `ErrorCodes.ACCOUNT_NOT_FOUND` у випадку, якщо обліковий запис не знайдено;

- `signup(body, res)` – `[POST] /account/signup` – метод для реєстрації нового облікового запису з використанням переданих даних. Повертає статус `HttpStatus.CREATED` та відповідь `JSON` із результатом успішної реєстрації або `HttpStatus.CONFLICT` та код помилки `ErrorCodes.ACCOUNT_ALREADY_EXIST`, якщо обліковий запис вже існує;

- `logout(sessionToken, res)` – `[POST] /account/logout` – метод для виходу користувача. Використовує `middleware AuthGuard` для перевірки аутентифікації користувача за токеном сесії. Повертає статус `HttpStatus.ACCEPTED` у разі успішного виходу або `HttpStatus.NOT_FOUND` та код помилки `ErrorCodes.SESSION_NOT_FOUND`, якщо сесія не знайдена;

– `change(body, sessionToken, res)` – [POST] `/account/change` – метод для зміни облікового запису користувача з використанням переданих даних та токена сесії. Використовує `middleware AuthGuard` для аутентифікації користувача. Повертає статус `HttpStatus.ACCEPTED` та відповідь JSON з оновленими даними облікового запису або `HttpStatus.CONFLICT` та відповідь JSON з помилкою у разі конфлікту змінюваних даних.

3.7.2 Activities

`Activities` контролер використовується для операцій над різними аспектами діяльності, такими як категорії, підкатегорії, набори, дії тощо. Він має різні методи HTTP для отримання, створення та видалення об'єктів в системі:

- `getCategories()` – [GET] `/activities/categories` – отримання категорій;
- `getSubCategories(categoryId)` – [GET] `/activities/subcategories/:categoryId` – отримання підкатегорій за певною категорією;
- `getPresets(sessionToken)` – [GET] `/activities/presets` – отримання наборів за токеном сесії;
- `createPreset(sessionToken, body)` – [POST] `/activities/presets` – створення нового набору за токеном сесії та даними, переданими в запиті;
- `deletePreset(sessionToken, body)` – [DELETE] `/activities/presets` – видалення набору за токеном сесії та даними, переданими в запиті.
- `getActivities(sessionToken, from, to)` – [GET] `/activities` – отримання діяльностей за певний період часу за токеном сесії та вказаними датами;
- `createActivity(sessionToken, body)` – [POST] `/activities` – створення нової діяльності за токеном сесії та даними, переданими в запиті;
- `deleteActivity(sessionToken, body)` – [DELETE] `/activities` – видалення діяльності за токеном сесії та даними, переданими в запиті.

Цей контролер використовує middleware AuthGuard для аутентифікації користувачів у багатьох методах та відповідає на запити HTTP з використанням статусів відповіді HTTP (наприклад, `HttpStatus.OK`, `HttpStatus.CREATED`, `HttpStatus.BAD_REQUEST`, тощо) та повертає відповідні результати у форматі JSON.

3.7.3 Food

Food контролер використовується для операцій, пов'язаних з обробкою інформації щодо харчування:

- `getAllFoods(sessionToken, from, to)` – [GET] /food – отримання списку продуктів за певний період часу за токеном сесії та вказаними датами. Повертає статус `HttpStatus.OK` з JSON відповіддю, що містить список продуктів, або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `createFood(sessionToken, body)` – [POST] /food – створення нового продукту з використанням переданих даних та токена сесії. Повертає статус `HttpStatus.CREATED` та JSON відповідь з результатом створення продукту або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `deleteFood(sessionToken, body)` – [DELETE] /food – видалення продукту за токеном сесії та даними, переданими в запиті. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом видалення продукту або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `searchAutoComplete(sessionToken, search)` – [GET] /food/search – пошук автозаповнення за певним пошуковим запитом для продуктів за токеном сесії та вказаним пошуковим запитом. Повертає статус `HttpStatus.OK` та JSON відповідь із результатами пошуку або `HttpStatus.NOT_FOUND` у випадку, якщо результати не знайдено;

- `getMyAutoComplete(sessionToken, skip, take)` – [GET] /food/my-autocomplete – отримання власного списку автозаповнення за токеном сесії, можливо з використанням параметрів `skip` та `take` для пагінації. Повертає

статус `HttpStatus.OK` та JSON відповідь з результатами або `HttpStatus.NOT_FOUND` у випадку помилки;

- `createMyAutoComplete(sessionToken, body)` – [POST] `/food/my-autocomplete` – створення власної інформації для автозаповнення продукту за токеном сесії та переданими даними. Повертає статус `HttpStatus.CREATED` та JSON відповідь з результатом створення або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `deleteMyAutoComplete(sessionToken, body)` – [DELETE] `/food/my-autocomplete` – видалення власної інформації для автозаповнення продукту за токеном сесії та переданими даними. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом видалення або `HttpStatus.BAD_REQUEST` у випадку помилки.

3.7.4 Measurements

Measurements контролер використовується для обробки різноманітних операцій з вимірюваннями та об'єктами вимірювань:

- `getAllWeight(sessionToken, skip, take)` – [GET] `/measurements/weight` – отримання списку вимірювань ваги за певний період часу за токеном сесії та параметрами для пагінації. Повертає статус `HttpStatus.OK` з JSON відповіддю, що містить дані про вагу або `HttpStatus.NOT_FOUND` у випадку помилки;

- `setWeight(sessionToken, body)` – [POST] `/measurements/weight` – збереження нових даних про вагу з використанням переданих даних та токена сесії. Повертає статус `HttpStatus.CREATED` та JSON відповідь з результатом збереження або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `deleteWeight(sessionToken, body)` – [DELETE] `/measurements/weight` – видалення даних про вагу за токеном сесії та даними, переданими в запиті. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом видалення або `HttpStatus.BAD_REQUEST` у випадку помилки;

- `getAllMeasurementObjects(sessionToken)` – [GET]
`/measurements/objects` – отримання списку об'єктів вимірювань за токеном сесії. Повертає статус `HttpStatus.OK` та JSON відповідь із списком об'єктів або `HttpStatus.BAD_REQUEST` у випадку помилки;
- `createMeasurementObject(sessionToken, body)` – [POST]
`/measurements/objects` – створення нового об'єкту вимірювань за токеном сесії та переданими даними. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом створення або `HttpStatus.BAD_REQUEST` у випадку помилки;
- `deleteMeasurementObject(sessionToken, body)` – [DELETE]
`/measurements/objects` – видалення об'єкту вимірювань за токеном сесії та переданими даними. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом видалення або `HttpStatus.BAD_REQUEST` у випадку помилки;
- `getAllMeasurements(sessionToken, measurementObjectId, skip, take)` – [GET]
`/measurements` – отримання списку вимірювань за певний об'єкт вимірювань та період часу за токеном сесії. Повертає статус `HttpStatus.OK` та JSON відповідь з результатами або `HttpStatus.NOT_FOUND` у випадку помилки;
- `setMeasurement(sessionToken, body)` – [POST] `/measurements` – збереження нових даних про вимірювання з використанням переданих даних та токена сесії. Повертає статус `HttpStatus.CREATED` та JSON відповідь з результатом збереження або `HttpStatus.BAD_REQUEST` у випадку помилки;
- `deleteMeasurement(sessionToken, body)` – [DELETE] `/measurements` – видалення даних про вимірювання за токеном сесії та даними, переданими в запиті. Повертає статус `HttpStatus.OK` та JSON відповідь з результатом видалення або `HttpStatus.BAD_REQUEST` у випадку помилки.

3.7.5 Profile

Profile контролер використовується для обробки профілів користувачів та їх калорійного рахунку:

- createProfile(sessionToken, body) – [POST] /profile – створення профілю користувача за допомогою наданої інформації та загальної щоденної витрати енергії. Повертає статус HttpStatus.CREATED та JSON відповідь з результатом створення або HttpStatus.BAD_REQUEST у випадку помилки;
- updateProfile(sessionToken, body) – [PUT] /profile – оновлення існуючого профілю користувача з новою інформацією, якщо передані дані для оновлення. Повертає оновлений профіль та його загальну щоденну витрату енергії або HttpStatus.BAD_REQUEST у випадку помилки;
- setCaloricity(sessionToken, body) – [POST] /profile/caloricity – встановлення загальної щоденної витрати енергії для конкретного профілю користувача. Повертає статус HttpStatus.OK з JSON відповіддю або HttpStatus.BAD_REQUEST у випадку помилки;
- getProfile(sessionToken) – [GET] /profile – отримання даних профілю користувача та його загальної щоденної витрати енергії за допомогою токenu сесії. Повертає статус HttpStatus.OK та JSON відповідь з результатом або HttpStatus.NOT_FOUND у випадку помилки.

3.7.6 Water

Water контролер використовується для обробки операцій з водопиттям користувачів:

- getActivities(sessionToken, from, to) – [GET] /water/activities – отримання даних щодо водопиття користувача за певний період часу на основі токenu сесії та параметрів from та to. Повертає дані про гідратацію за вказаний період або HttpStatus.NOT_FOUND у разі помилки;
- createActivity(sessionToken, body) – [POST] /water/activities – створення запису про водопиття для користувача на основі переданих даних та токenu сесії. Повертає статус HttpStatus.CREATED та створений запис про водопиття або HttpStatus.BAD_REQUEST у разі помилки;

– `deleteActivity(sessionToken, body)` – [DELETE] `/water/activities` – видалення запису про водопиття користувача на основі переданих даних та токена сесії. Повертає статус `HttpStatus.OK` у разі успішного видалення запису або `HttpStatus.BAD_REQUEST` у разі помилки.

Ці маршрути використовують сервіс `WaterService` для взаємодії з даними про гідратацію в базі даних та захищені з допомогою `AuthGuard` для перевірки аутентифікації користувача за токеном сесії.

3.8 Методи ідентифікації, аутентифікації, авторизації користувачів у програмі

Архітектура аутентифікації та авторизації працює на взаємодії між `Angular` та `Nest.js`, забезпечуючи безпеку та керування доступом для користувачів застосунку. Вона включає в себе передачу даних через HTTP-запити, перевірку та зберігання сесійних токенів, а також обробку дій користувача на сервері.

Цей підхід забезпечує ефективний механізм авторизації та аутентифікації, але може бути розширений або налаштований для покращення стандартів безпеки.

3.8.1 Ідентифікація

Ідентифікація визначає, хто саме звертається до системи. У цьому коді процес ідентифікації відбувається через взаємодію з базою даних (`AccountDbService`) у `Nest.js`. Методи `login` та `signin` використовуються для ідентифікації користувачів, перевірки наявності облікового запису за електронною адресою та паролем, а потім створення сесії для входу в систему та повернення інформації про обліковий запис користувача з токеном сесії.

3.8.2 Аутентифікація

Аутентифікація визначає, чи є дані, представлені користувачем, дійсними для конкретного користувача. У цьому коді аутентифікація відбувається шляхом перевірки електронної адреси та пароля користувача з даними у базі даних (accountDbService). Якщо обліковий запис знайдено, аутентифікація вважається успішною, і генерується токен сесії, який надсилається клієнту.

3.8.3 Авторизація

Авторизація визначає, чи має користувач доступ до певних ресурсів або функціоналу після успішної аутентифікації. У цьому коді авторизація відбувається через перевірку токенів сесій. Клас SessionService визначає методи для перевірки валідності сесії (sessionValid), отримання облікового запису за токеном сесії (accountBySession), а також отримання профілю за токеном сесії (profileBySession).

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування гри

Робоче ім'я застосунку – «NutriLight». Застосунок розроблено задля оптимізації харчування широкого кола людей.

Область застосування – оптимізація харчування та покращення здоров'я за допомогою застосунків.

Цей програмний продукт призначений для людей, які активно цікавляться фітнесом та спортом, і він стане корисним для тих, хто має намір вдосконалити свою фізичну форму, будь то зниження маси тіла або нарощування м'язової маси. Застосунок працює в онлайн режимі і потребує наявності стабільного Інтернет з'єднання, а також має середні системні вимоги. Для ефективної роботи застосунку необхідно належно налаштувати бекенд та базу даних, які розташовані на сервері.

Для функціонування застосунку необхідно виконати мінімальні умови щодо використання пристрою з наступними апаратними характеристиками:

- платформа – Windows (7, 8, 8.1, 10), Linux (будь-який сучасний дистрибутив), Android, Mac OS (OS X Yosemite 10.10 або вище);
- розрядність – x86 (32-бітна), x64 (64-бітна);
- відеоадаптер – Nvidia, Intel, AMD, ATI;
- пам'ять – мінімум 20 гб;
- оперативна пам'ять – мінімум 2 гб;
- контролер – клавіатура, миша;
- інтернет – стабільне Інтернет-з'єднання;
- роздільна здатність екрана – мінімум 800x600.

Ці вимоги визначають мінімальну конфігурацію системи для комфортного використання програми та її функцій.

4.2 Характеристики застосунку

4.2.1 Характеристики Backend

Backend написано на фреймворку Nest.js для Node.JS. Мовою TypeScript.

Скомпільований код в js файли з мови TypeScript має розмір 4,84 МБ (5 079 606 байт), містить 208 файлів, і 20 папок.

Файлова структура скомпільованого коду виглядає зображена на рисунку 4.1.

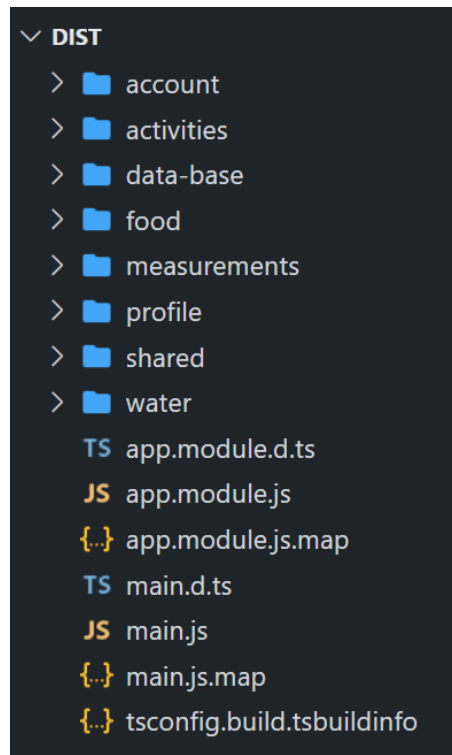


Рисунок 4.1 – Файлова структура Backend

4.2.2 Характеристики Frontend

Frontend написано на фреймворку Angular для Node.JS. Мовою TypeScript. На рисунку 4.2 можна побачити результат виведення консолі після компіляції.

✓ Browser application bundle generation complete.
 ✓ Copying assets complete.
 ✓ Index html generation complete.

Initial Chunk Files	Names	Raw Size	Estimated Transfer Size
main.63df1e16f14f514e.js	main	357.46 kB	93.17 kB
styles.241dc8081fea3d5a.css	styles	72.29 kB	7.60 kB

Lazy Chunk Files	Names	Raw Size	Estimated Transfer Size
707.c442e6b3cc107ed7.js	echarts	1002.24 kB	270.75 kB
543.f4b5b3feb79e4221.js	diary-diary-module	374.21 kB	60.54 kB
307.92f8317f65072bd9.js	account-account-module	74.46 kB	14.70 kB
763.cb451735d249e6a7.js	diary-diary-module	54.75 kB	8.09 kB
549.8114a9a529eb2037.js	measurements-measurements-module	25.73 kB	4.79 kB
608.d20a1170532d5558.js	report-report-module	12.86 kB	3.21 kB
common.3f8533dd91dcb949.js	common	9.46 kB	2.50 kB

Build at: 2023-11-25T17:10:53.109Z - Hash: 5322c09684c71e1b - Time: 30796ms

Рисунок 4.2 – Результат виведення консолі після компіляції

Скомпільований код у js файли з мови TypeScript має розмір 233 КБ (239 077 байт), містить 260 файлів і 60 папок. Файлова структура скомпільованого коду зображена на рисунку 4.3.

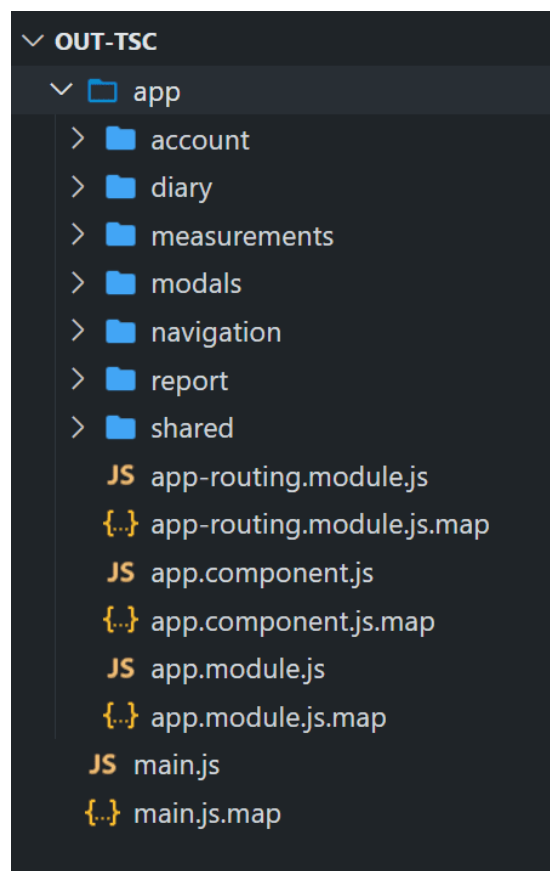


Рисунок 4.3 – Файлова структура Frontend

Крім того, Angular генерує ефективний код для завантаження користувачами.

Компіляція коду для фронтенд-застосунків має кілька важливих причин:

- використання високорівневих мов та функцій. Браузери розуміють лише JavaScript, HTML і CSS. Проте розробники можуть використовувати більш продуктивні мови програмування, такі як TypeScript, CoffeeScript або нові версії JavaScript (наприклад, ES6+), що мають більше можливостей. Компіляція допомагає перетворити ці мови в JavaScript, зрозумілий браузерам;

- підтримка різних браузерів та версій. Різні браузери мають різні можливості та підтримку нових функцій JavaScript. Під час компіляції можна включити транспіляцію коду у формат, підтримуваний більшістю браузерів, або автоматично додати поліфіли для тих, що не підтримують певні функції;

- оптимізація та зменшення розміру файлів. Компілятори можуть оптимізувати код, зменшуючи його розмір. Це важливо для підвищення швидкості завантаження вебсторінок через мережу, оскільки менший розмір файлів дозволяє їм швидше завантажуватися;

- робота з модулями та залежностями. Компіляція дозволяє використовувати різноманітні модулі та імпортувати їх у ваш проєкт, забезпечуючи кращу організацію коду та зменшуючи його складність.

Отже, компіляція коду для Angular дозволяє розробникам писати більш зрозумілий, оптимізований та функціональний код, який можна легко розгорнути на вебсторінках для покращення досвіду користувача.

Скомпільований і стислий код, готовий до завантаження користувачем має розмір 2,04 МБ (2 147 862 байт), містить 18 файлів, і 3 папки.

Файлова структура стислого коду зображено на рисунку 4.4.

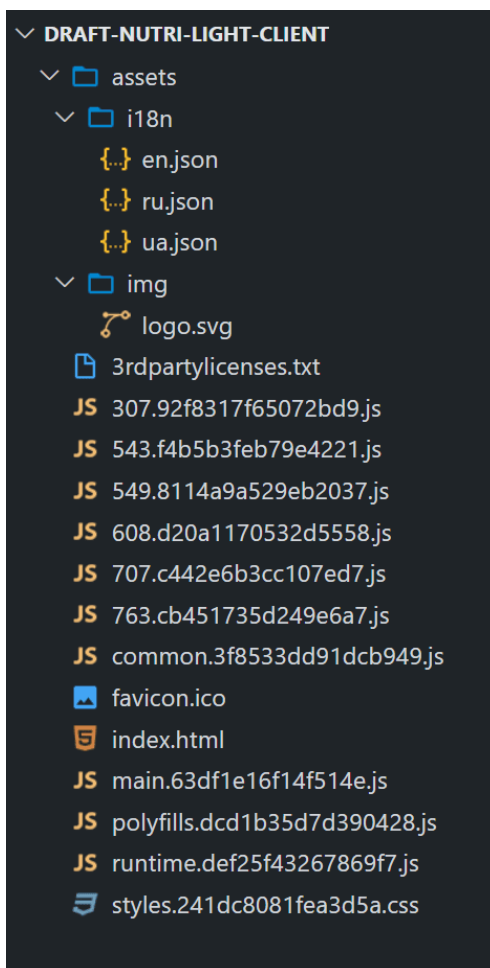


Рисунок 4.4 – Файлова структура Frontend із стислим кодом

4.3 Звертання до застосунку

Застосунок складається з трьох частин: база даних, Backend та Frontend.

Для розгортання працездатного застосунку необхідно правильно налаштувати всі три частини.

4.3.1 Встановлення Backend та бази даних

Для встановлення backend необхідно клонувати репозиторій із backend застосунком на місце деплою. Клонування виконується за допомогою команди: `git clone https://path/to/repository`

Далі потрібно встановити пакети залежностей node.js. Робиться це за допомогою наступної команди: `npm install`.

Далі потрібно встановити базу даних. Можна встановити існуючу базу даних, а можна створити нову.

У разі встановлення існуючої необхідно завантажити та перемістити файл бази даних до папки prism.а.

Далі необхідно виконати міграцію бази даних для ініціалізації за допомогою наступної команди: `prx prism migrate dev --name init` [86].

Коли пакети залежностей встановлені та база даних ініціалізована – можна запускати програму. Робиться це наступною командою: `npm start`.

Або, якщо на пристрій встановлено nest cli можна зробити наступною командою: `nest start`.

Якщо під час запуску в консоль буде виведено таке повідомлення, що зображено на рисунку 4.5, то на цьому етапі backend можна вважати запущеним.

```
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [NestFactory] Starting Nest application ...
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] AppModule dependencies initialized +13ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] DataBaseModule dependencies initialized +4ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] SharedModule dependencies initialized +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] ProfileModule dependencies initialized +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] MeasurementsModule dependencies initialized +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] ActivitiesModule dependencies initialized +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] FoodModule dependencies initialized +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] WaterModule dependencies initialized +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [InstanceLoader] AccountModule dependencies initialized +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] AccountController {/account}: +12ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/account/log-in, POST} route +2ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/account/sign-up, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/account/log-out, HEAD} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/account/change, PATCH} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] ProfileController {/profile}: +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/profile, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/profile, PATCH} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/profile, GET} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/profile/caloricity, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] MeasurementsController {/measurements}: +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/weights, GET} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/weight, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/weight, DELETE} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurement-objects, GET} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurement-object, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurement-object, DELETE} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurements, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurement, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/measurements/measurement, DELETE} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] ActivitiesController {/activities}: +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities/categories, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities/sub-categories, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities/presets, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities/preset, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities/preset, DELETE} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/activities, DELETE} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] FoodController {/food}: +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food, DELETE} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food/auto-complete/search, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food/auto-complete/my, GET} route +1ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food/auto-complete/my, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/food/auto-complete/my, DELETE} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RoutesResolver] WaterController {/water}: +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/water, GET} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/water, POST} route +0ms
(Nest) 940 ~ 23.11.2023, 22:33:11 LOG [RouterExplorer] Mapped {/water, DELETE} route +0ms
```

Рисунок 4.5 – Завантажений backend

4.3.2 Встановлення Frontend

Для встановлення frontend необхідно клонувати репозиторій з frontend застосунком на місце деплою. Клонування виконується за допомогою команди: `git clone https://path/to/repository`.

Далі потрібно встановити пакети залежностей `node.js`. Робиться це за допомогою наступної команди: `npm install`.

Далі необхідно запустити програму за допомогою наступної команди: `npm start`.

Або, якщо на пристрій встановлено `angular cli` можна зробити наступною командою: `ng serve`.

Якщо під час запуску в консоль буде виведено таке повідомлення, що зображено на рисунку 4.6, то на цьому етапі frontend можна вважати запущеним.

```
> ng serve
. Browser application bundle generation complete.

Initial Chunk Files                                         Names          Raw Size
vendor.js                                                  vendor         3.29 MB
polyfills.js                                              polyfills     333.26 kB
styles.css, styles.js                                   styles        318.07 kB
main.js                                                  main          119.57 kB
runtime.js                                              runtime        13.01 kB

Initial Total                                             4.05 MB

Lazy Chunk Files                                           Names          Raw Size
node_modules_echarts_index.js                           echarts        4.46 MB
src_app_account_account_module_ts.js                   account-account-module  479.66 kB
src_app_diary_diary_module_ts.js                       diary-diary-module    230.71 kB
src_app_measurements_measurements_module_ts.js         measurements-measurements-module  110.18 kB
src_app_report_report_module_ts.js                     report-report-module   46.23 kB
default-src_app_shared_services_activities_service_ts-src_app_shared_services_foods_service_t-6e0fde.js  diary-diary-module    20.62 kB
common.js                                               common           728 bytes

Build at: 2023-11-23T20:41:12.186Z - Hash: c33d085d27936f67b - Time: 7216ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
. Browser application bundle generation complete.

12 unchanged chunks

Build at: 2023-11-23T20:41:14.277Z - Hash: c33d085d27936f67b - Time: 1809ms

✓ Compiled successfully.
```

Рисунок 4.6 – Завантажений frontend

4.4 Використання застосунку

Якщо немає облікового запису, потрібно зареєструватися (рис. 4.7).

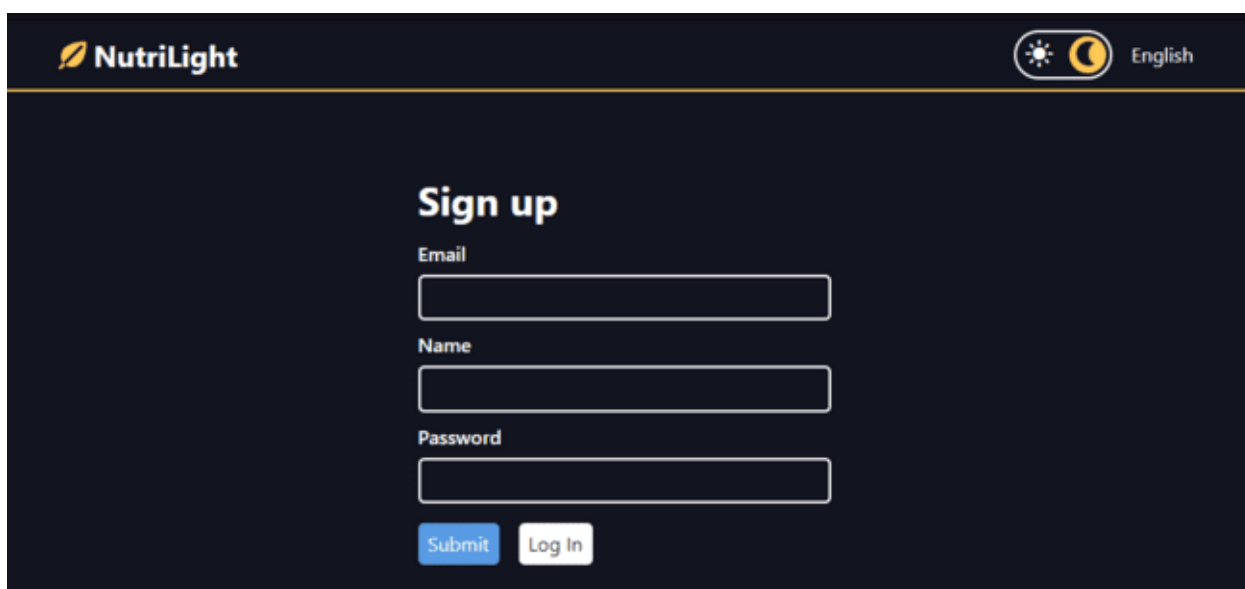


Рисунок 4.7 – Сторінка реєстрації

Якщо користувач має обліковий запис, він може увійти в свій аккаунт через сторінку авторизації (рис. 4.8).

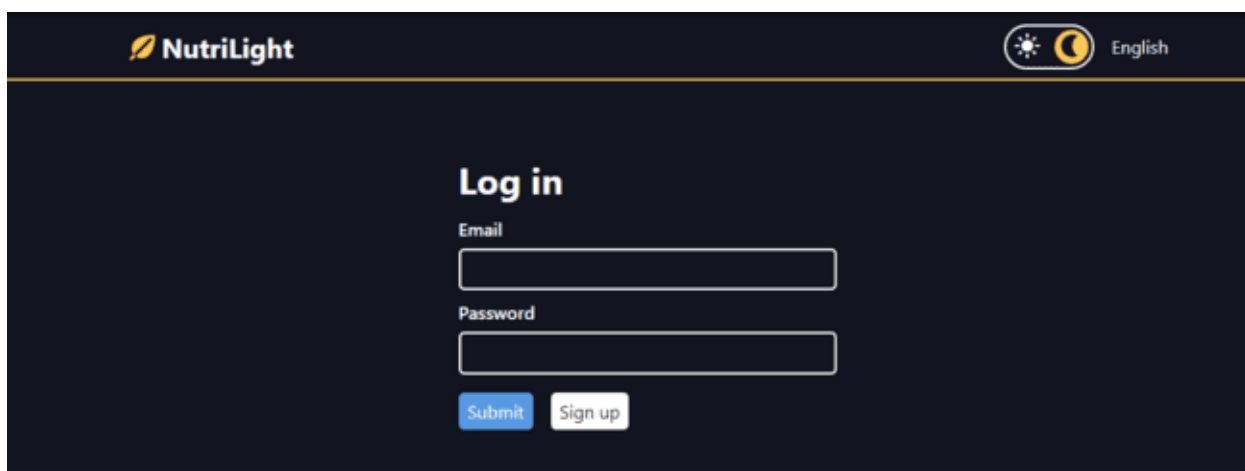


Рисунок 4.8 – Сторінка реєстрації

У вкладці Profile можна налаштувати: акаунт, профіль, розподіл макронутрієнтів (рис. 4.9). Розділ налаштування акаунту дозволяє користувачам керувати своєю обліковою інформацією, включаючи електронну адресу, пароль та ім'я користувача. Розділ профілю дозволяє користувачам вводити та редагувати свою особисту інформацію, включаючи ім'я, зріст, вагу, стать, рівень фізичної активності, дату народження, цільову

вагу та темп схуднення/набору. Розділ макронутрієнтів дозволяє користувачам переглядати та редагувати свій розподіл калорій між білками, вуглеводами та жирами.

NutriLight Diary Report Measurements Profile English Log Out

Account

Email:

Name:

Password:

Confirm password:

Apply changes

Profile

Height:

Weight:

Sex:

Physical activity level:

Birth Day:

Target Weight:

Target Rate:

Apply changes

Macronutrients

100

Protein 25% 142g 2.52g/kg 567.21cal

Carbohydrates 45% 255g 4.53g/kg 1021.2cal

Fats 30% 76g 1.35g/kg 681.54cal

Apply changes

Developed by: Annya Shutko

Рисунок 4.9 – Сторінка профілю

На рисунку 4.10 зображено розділ щоденнику, де діаграма калорій показує баланс калорій користувача за поточний день або тиждень. Користувач може використовувати цю діаграму, щоб відстежувати свій прогрес у досягненні своїх цілей щодо споживання калорій.

Діаграма макронутрієнтів показує розподіл калорій користувача між білками, вуглеводами та жирами за поточний день та тиждень. Користувач може використовувати цю діаграму, щоб переконатися, що він споживає достатньо білків, вуглеводів та жирів для своїх потреб.

Баланс гідратації відображає, скільки користувач випив води за поточний день. Користувач може використовувати цю інформацію, щоб переконатися, що він споживає достатньо рідини.

Розділ "Activities" показує список останніх активностей користувача. Користувач може використовувати цей розділ, щоб переглядати свою фізичну активність за поточний день.

Розділ "Foods" показує список останньої їжі користувача. Користувач може використовувати цей розділ, щоб переглядати раціон його споживання за поточний день.



Рисунок 4.10 – Сторінка щоденника

Нову їжу можна додати за допомогою меню створення їжі, приклад такої дії можна побачити на рисунках 4.11-4.12. Для додавання через цю форму їжі потрібно знайти їжу з бази даних натиснувши на кнопку «Перемкнути режим», написати назву продукту, обрати його зі списку, ввести кількість у грамах, ввести дату та час. Після цього при натисканні на "Надіслати" буде створюватись запис у базі даних. Також можна додати свою їжу. Ввести кількість білків, жирів та вуглеводів на 100 грам продукту та налаштувати яку кількість грам продукту потрібно врахувати в щоденнику та обрати дату та час запису.

Нова їжа

Пошук
Куряча грудка

Обрати
Куряча грудка Oscar Mayer (в медовій глазурі)

Дані про продукт на 100 грамів

Назва		
Куряча грудка Oscar Mayer (в медовій глазурі)		
Група	Калорії	
М'ясо	109	
Білки	Вуглеводи	Жири
19.85	4.3	1.5

Калорії: 164 Білки: 30 Вуглеводи: 6 Жири: 2

Грами: 150

Дата: 09.12.2023 Час: 15:10

Надіслати Перемкнути режим

Рисунок 4.11 – Додавання їжі з бази даних темна та світла тема

Нова їжа

Назва
Кешью горіхи

Дата: 09.12.2023 Час: 10:20

Макронутрієнти на 100 грам

Білки: 18.2

Вуглеводи: 26.9

Жири: 43.9

Сумарно

Грами: 40

Калорії: 230

Надіслати Перемкнути режим

Рисунок 4.12 – Додавання своєї їжі темна та світла тема

Переглянути їжу за обраний тиждень можливо при натисканні на перегляд детальної інформації. Вона буде виведена у вигляді таблиці (рис. 4.13).



Їжа на тиждень

Оберіть Тиждень: Неделя 49, 2023

Назва	Дата	Час	Калорії	Білки	Вуглеводи	Жири
Куряча грудка Опсат ...	09.12.2023	15:10	165	30	6	2
Сирники зі см.	09.12.2023	10:20	331	17	23	19
Каша з горіш	09.12.2023	10:20	230	7	11	18
Вівсянка з медом	09.12.2023	07:30	304	11	51	6
Яблуко червоне	09.12.2023	07:30	124	1	30	0
Чайник з вишнею	08.12.2023	21:15	462	14	34	30
Рибу з куркою	08.12.2023	19:40	126	11	8	6
Хліб цз.	08.12.2023	19:40	98	5	16	1
Сир з медом	08.12.2023	16:40	208	22	15	7
Фруктовий салат	08.12.2023	16:40	75	1	17	0
Вівсянка з медом	08.12.2023	14:05	316	14	51	6

Рисунок 4.13 – Сторінка з детальною інформацією по спожитій їжі

На рисунку 4.14 зображено приклад додавання активностей з бази даних. Для додавання активностей потрібно написати назву активності, обрати назву категорії та підкатегорії з бази даних, заповнити дату, час та тривалість активності.

Нова активність ×

Назва
Боулінг

Категорія
Спорт

Підкатегорія
боулінг

Дата
09.12.2023

Час
12:00

Тривалість
01:30

Надіслати

Рисунок 4.14 – Форма додавання активностей темна та світла тема

На рисунку 4.15 можна переглянути активності за обраний тиждень при натисканні на перегляд детальної інформації. Вона буде виведена у вигляді таблиці

Активності на тиждень

Оберіть Тиждень: Неделя 49, 2023

Назва	Дата	Час	Тривалість	Категорія	Калорії
Боулінг	09.12.2023	12:00	90	боулінг	270

Рисунок 4.15 – Сторінка з детальною інформацією активностей

На рисунку 4.16 зображено сторінку "Measurements", де можна побачити категорії ваги та кнопку для створення нової категорії вимірювання.

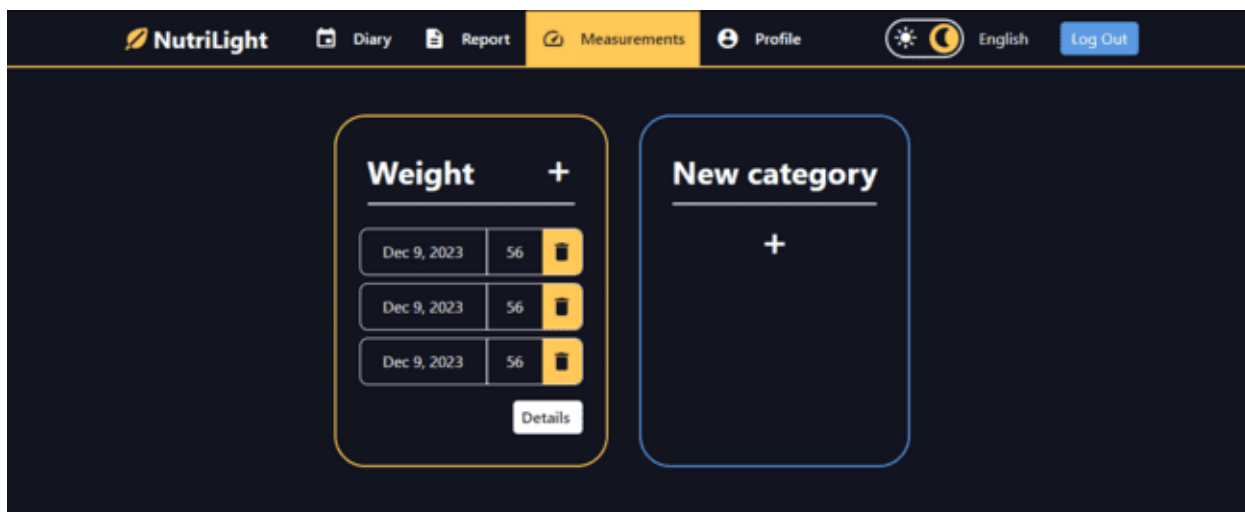


Рисунок 4.16 – Сторінка вимірювань

Для перегляду статистики для вимірювання або ваги користувача потрібно перейти до "Measurements" та натиснути на перегляд потрібної категорії. Сторінка з переглядом даних по категорії буде мати графік, назву, та останні записи (рис. 4.17). До того ж якщо обрано категорію ваги буде відображатися кількість днів до бажаної ваги. Як ця сторінка виглядає у мобільній адаптації можна побачити на рисунку 4.18.

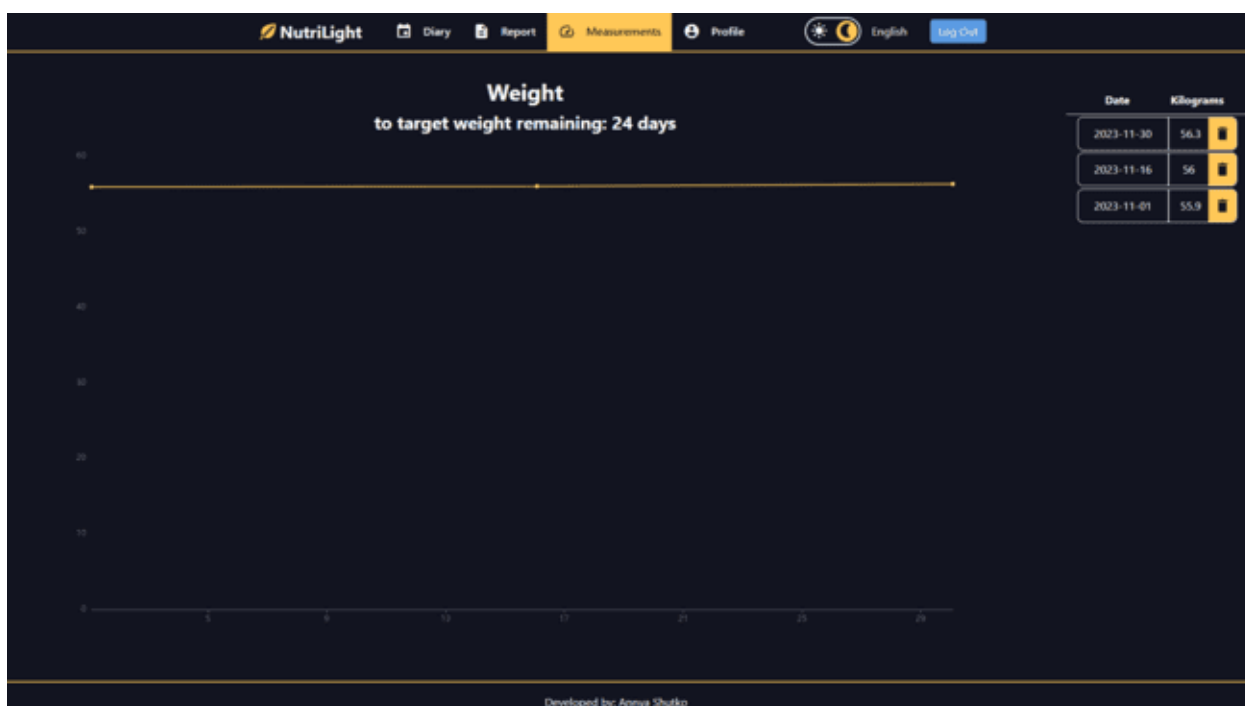


Рисунок 4.17 – Десктопна сторінка вимірювань ваги

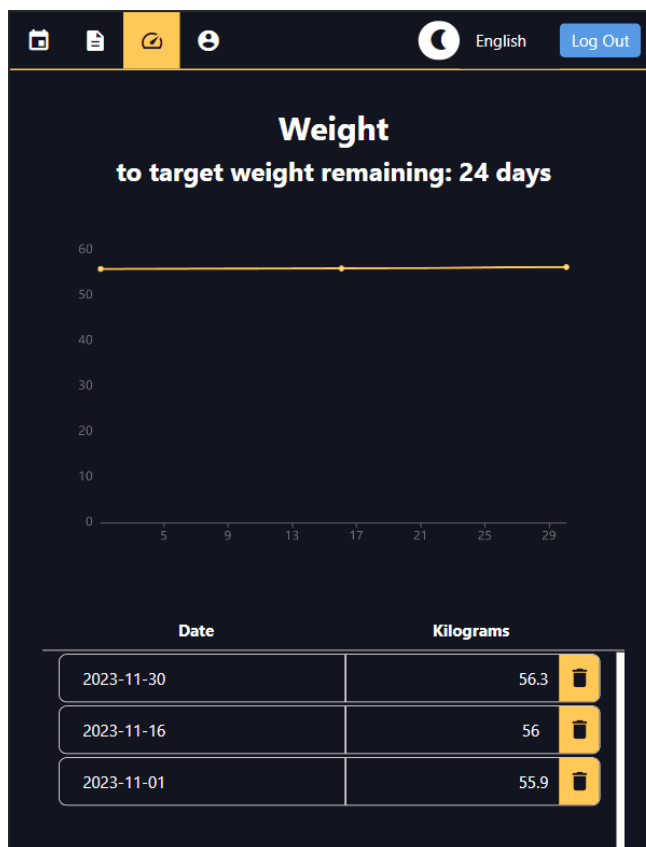


Рисунок 4.18 – Сторінка з інформацією по вазі користувача у мобільній адаптації

На сторінці "Report" (рисунок 4.19) можна побачити статистику за обраний період часу, у вигляді діаграм.

Діаграма балансу відображає спожиті, спалені калорії та плани користувача на прийом калорій. Якщо спожито менше, ніж спалено, вага зменшується; в іншому випадку – збільшується.

Графік макронутрієнтів показує щоденну кількість макронутрієнтів (білки, вуглеводи, жири), які споживає користувач у грамах. Графік спалених калорій показує декілька параметрів про спалення калорій, де:

- базовий обмін речовин (BMR) визначає, скільки енергії тіло витрачає у стані спокою на необхідні функції, як: дихання, кровообіг, і терморегуляцію. Метаболічне навантаження – пов'язане з рівнем фізичної активності, вказує на кількість енергії, яка споживається під час вправ. Різні типи фізичних занять вимірюються за допомогою метаболічних еквівалентів (METs);

- графік гідрації відображає інформацію про обсяг споживаної води.
- Також на сторінці "Report" можна розрахувати рекомпозицію.



Рисунок 4.19 – Сторінка зі статистикою користувача

Мобільну адаптацію для цієї сторінки можна побачити на рисунках 4.20-4.21.

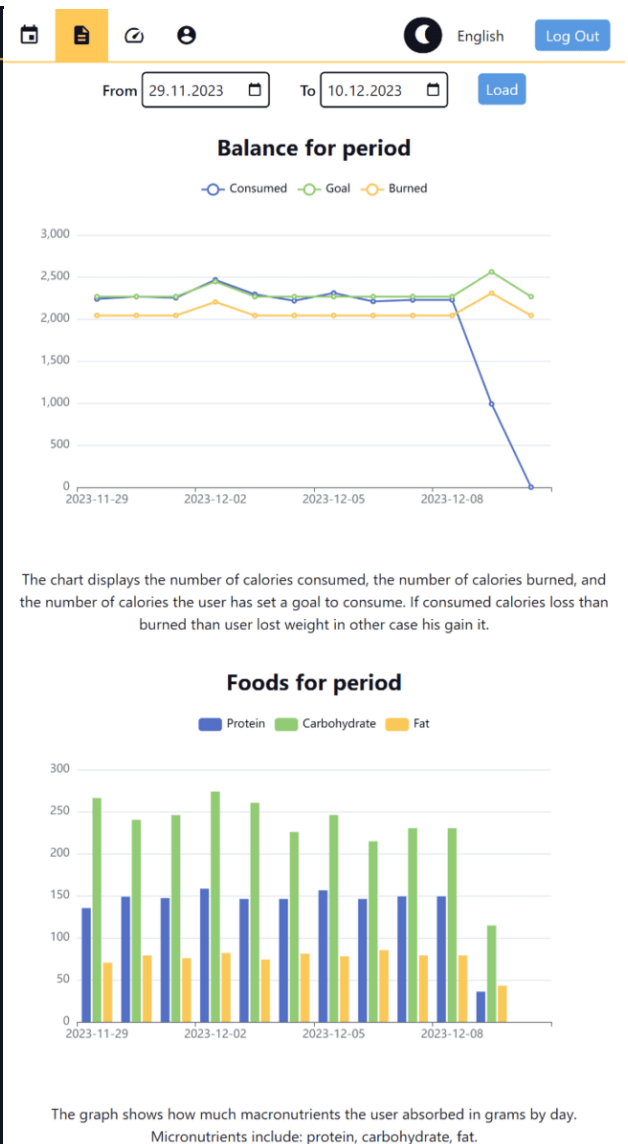
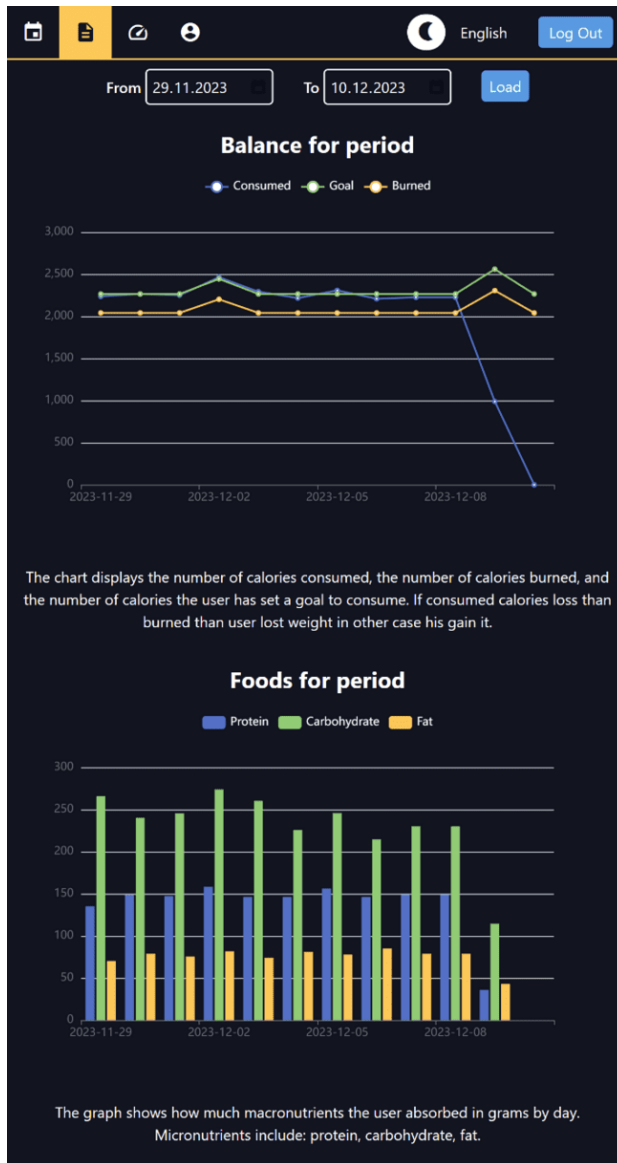
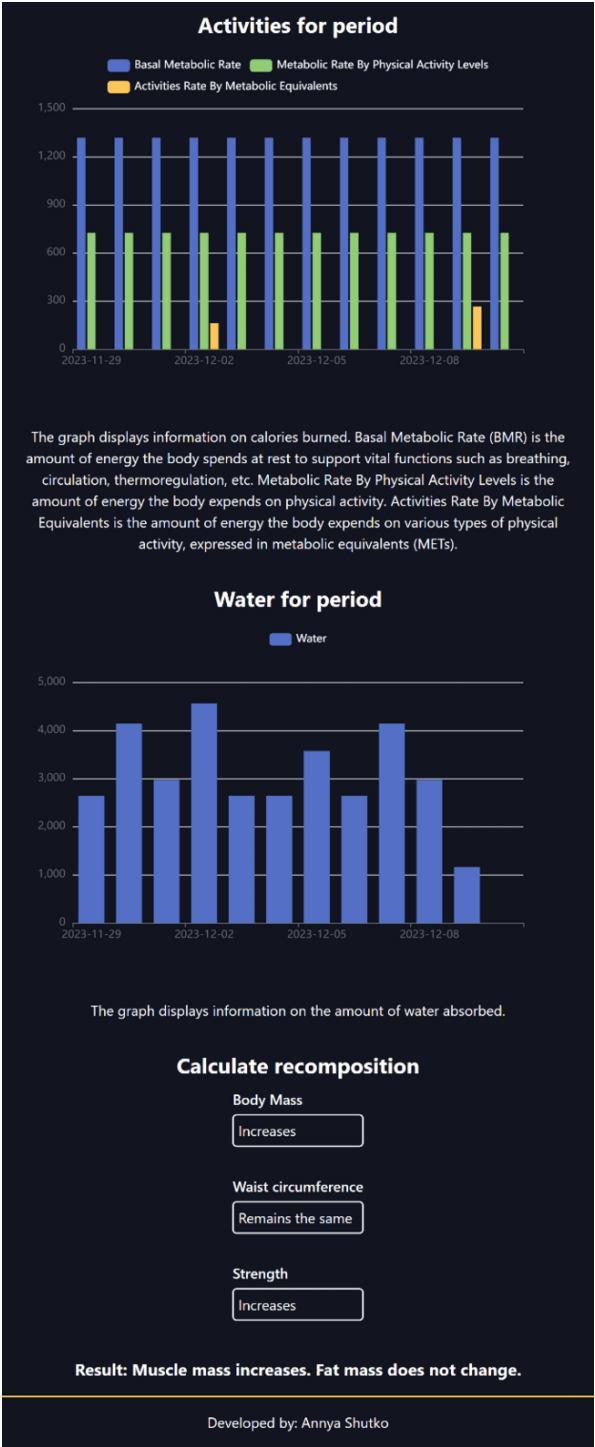


Рисунок 4.20 – Сторінка зі статистикою користувача у мобільній адаптації



5 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ

5.1 Опис методики проведення тестування

Для тестування програми виконуватиметься ручне тестування фронтенду, які включатимуть перевірку коректності відображення елементів інтерфейсу, функціональності кнопок та взаємодії користувача з застосунком.

5.2 Результати використання розробленого застосунка

Запускаємо backend застосунку (рис. 5.1).

```
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [NestFactory] Starting Nest application...
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] AppModule dependencies initialized +14ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] DataBaseModule dependencies initialized +23ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] SharedModule dependencies initialized +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] ProfileModule dependencies initialized +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] MeasurementsModule dependencies initialized +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] ActivitiesModule dependencies initialized +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] FoodModule dependencies initialized +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] WaterModule dependencies initialized +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [InstanceLoader] AccountModule dependencies initialized +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] AccountController {/account}: +17ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/account/log-in, POST} route +2ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/account/sign-up, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/account/log-out, HEAD} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/account/change, PATCH} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] ProfileController {/profile}: +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/profile, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/profile, PATCH} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/profile, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/profile/caloricity, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] MeasurementsController {/measurements}: +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/weights, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/weight, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/weight, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurement-objects, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurement-object, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurement-object, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurements, GET} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurement, POST} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/measurements/measurement, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] ActivitiesController {/activities}: +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities/categories, GET} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities/sub-categories, GET} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities/presets, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities/preset, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities/preset, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities, GET} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/activities, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] FoodController {/food}: +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food/auto-complete/search, GET} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food/auto-complete/my, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food/auto-complete/my, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/food/auto-complete/my, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RoutesResolver] WaterController {/water}: +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/water, GET} route +1ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/water, POST} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [RouterExplorer] Mapped {/water, DELETE} route +0ms
[Nest] 16172 - 25.11.2023, 20:37:20 LOG [NestApplication] Nest application successfully started +9ms
```

Рисунок 5.1 – Запуск backend

Запускаємо frontend застосунку (рис. 5.2).

```

✓ Browser application bundle generation complete.

Initial Chunk Files
vendor.js
polyfills.js
styles.css, styles.js
main.js
runtime.js

Names
vendor
polyfills
styles
main
runtime

Raw Size
3.29 MB
333.26 kB
318.07 kB
119.57 kB
13.01 kB

Initial Total
4.05 MB

Lazy Chunk Files
node_modules_echarts_index.js.js
src_app_account_account_module_ts.js
src_app_diary_diary_module_ts.js
src_app_measurements_measurements_module_ts.js
src_app_report_report_module_ts.js
default-src_app_shared_services_activities_service_ts-src_app_shared_services_foods_service_t-6e0fde.js
common.js

Names
echarts
account-account-module
diary-diary-module
measurements-measurements-module
report-report-module
diary-diary-module
common

Raw Size
4.46 MB
479.66 kB
235.83 kB
110.18 kB
46.23 kB
20.62 kB
728 bytes

Build at: 2023-11-25T18:52:38.247Z - Hash: a08c9426ef54ae7e - Time: 4257ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.

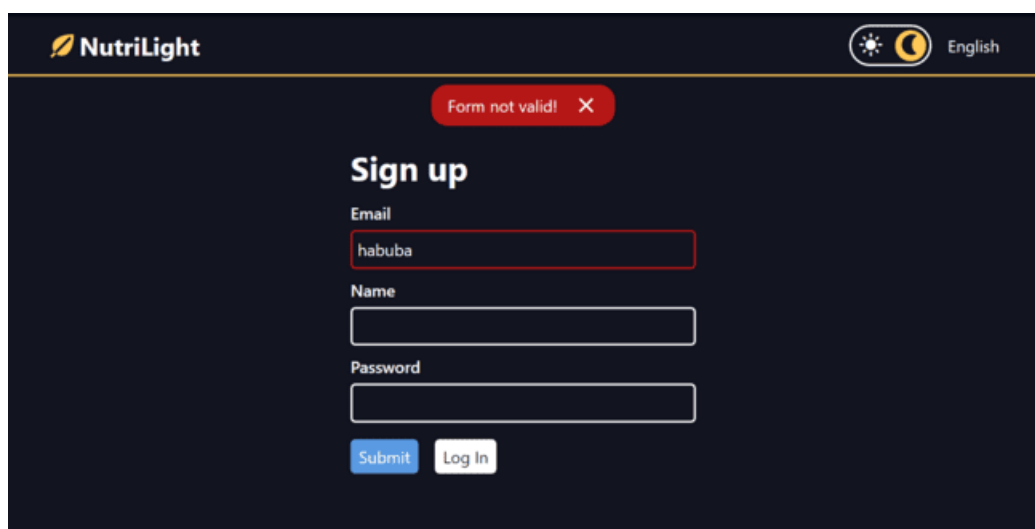
```

Рисунок 5.2 – Запуск Frontend

Відкриваємо <http://localhost:4200/>, потрапляємо на сторінку реєстрації (рис. 5.3).

Рисунок 5.3 – Сторінка реєстрації

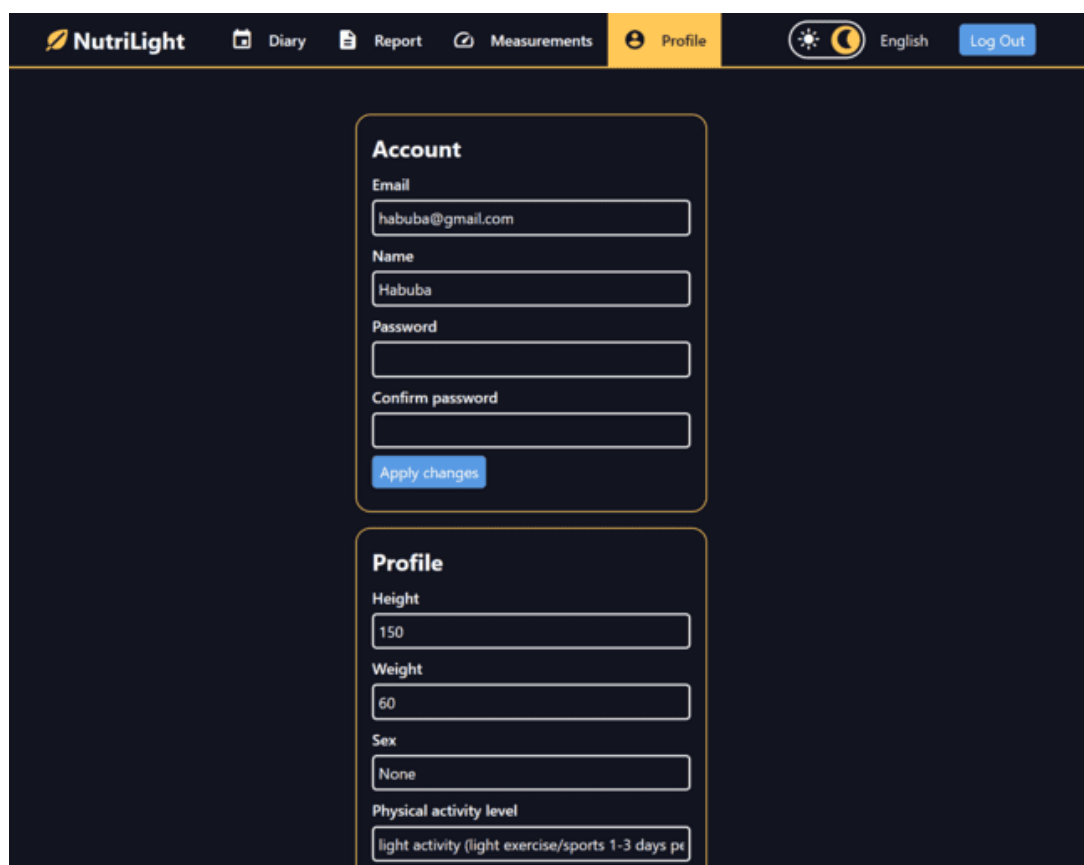
Вводимо некоректні дані в поле Email та натискаємо "Submit"(рис 5.4).



The image shows the 'Sign up' page of the NutriLight application. At the top, there is a dark blue header with the 'NutriLight' logo on the left and a toggle for 'English' on the right. Below the header, a red error message 'Form not valid!' with a close icon is displayed. The main content area is dark blue and contains the 'Sign up' title. Below the title are three input fields: 'Email' (containing 'habuba'), 'Name' (empty), and 'Password' (empty). At the bottom of the form are two buttons: 'Submit' (blue) and 'Log In' (white).

Рисунок 5.4 – Введені некоректні дані на сторінці реєстрації

Вводимо коректні дані та відправляємо форму. Відкриється сторінка профілю (рис. 5.5).



The image shows the 'Profile' page of the NutriLight application. The top header is dark blue with the 'NutriLight' logo and navigation links: 'Diary', 'Report', 'Measurements', and 'Profile' (highlighted in orange). On the right side of the header are a toggle for 'English' and a 'Log Out' button. The main content area is dark blue and contains two sections: 'Account' and 'Profile'. The 'Account' section has input fields for 'Email' (containing 'habuba@gmail.com'), 'Name' (containing 'Habuba'), 'Password' (empty), and 'Confirm password' (empty), followed by an 'Apply changes' button. The 'Profile' section has input fields for 'Height' (containing '150'), 'Weight' (containing '60'), 'Sex' (containing 'None'), and 'Physical activity level' (containing 'light activity (light exercise/sports 1-3 days pe)').

Рисунок 5.5 – Сторінка профілю

Вводимо дані для створення профілю: зріст, маса тіла, стать, коефіцієнт активності, дату народження, цільову вагу та швидкість її досягнення (рис. 5.6).

NutriLight Diary Report Measurements **Profile** English Log Out

Apply changes

Profile

Height
160

Weight
60

Sex
Female

Physical activity level
moderately active (moderate exercise/sports)

Birth Day
01.10.2000

Target Weight
65

Target Rate
0.11

Apply changes

Рисунок 5.6 – Налаштування профілю

На рисунку 5.7 підтверджуємо зміни та бачимо повідомлення про успішне оновлення даних, а нижче з'явився компонент з налаштуванням розподілу макронутрієнтів(за замовчуванням: білки – 30%, жири – 30%, вуглеводи – 40%).

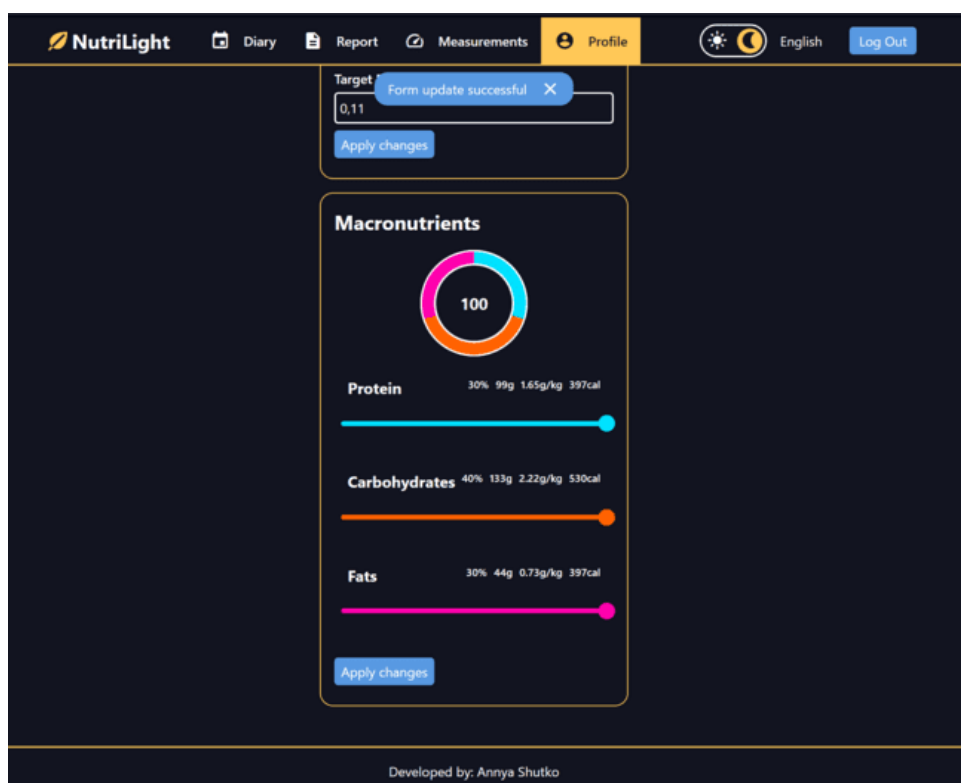


Рисунок 5.7 – Налаштування профілю

Змінюємо розподіл макронутрієнтів (рис. 5.8).

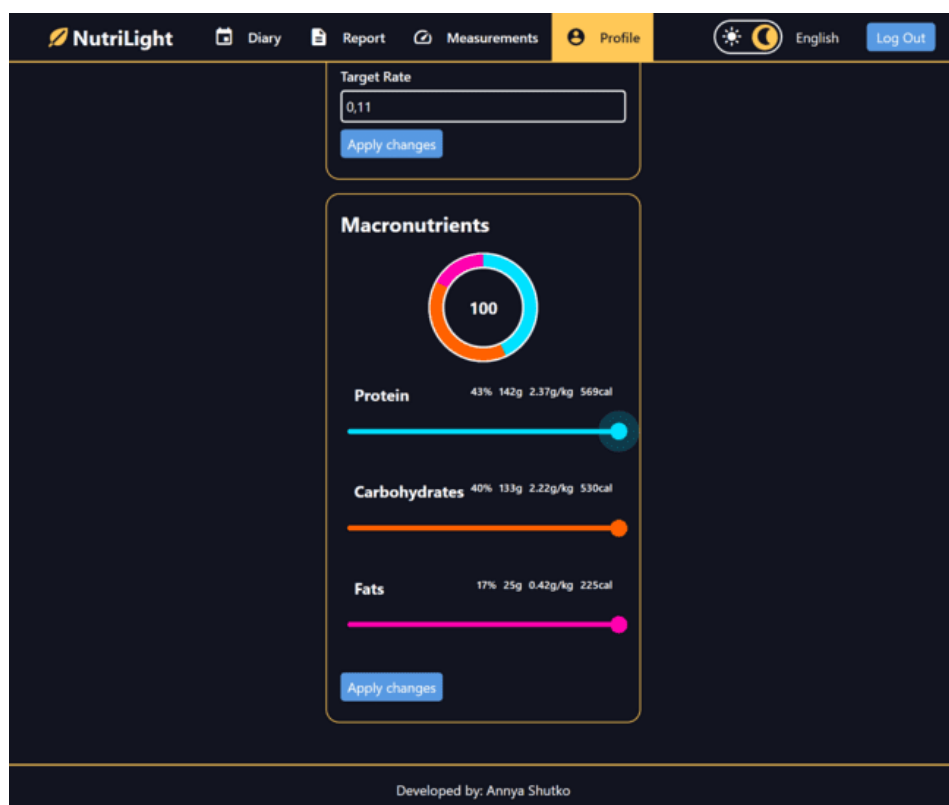


Рисунок 5.8 – Налаштування макронутрієнтів

Підтверджуємо налаштовані макронутрієнти, з'являється повідомлення про успішне збереження даних (рис. 5.9)

NutriLight Diary Report Measurements Profile English Log Out

Target: Form update successful X
0,11
Apply changes

Macronutrients

100

Protein 43% 142g 2.37g/kg 569cal
Apply changes

Carbohydrates 40% 133g 2.22g/kg 530cal
Apply changes

Fats 17% 25g 0.42g/kg 225cal
Apply changes

Developed by: Annya Shutko

Рисунок 5.9 – Збереження макронутрієнтів

Перезавантажуємо сторінку для перевірки завантаження даних (рис. 5.10).

The screenshot shows the 'Profile' page of the NutriLight application. At the top, there is a navigation bar with icons for 'Diary', 'Report', 'Measurements', and 'Profile' (which is highlighted). To the right of the navigation bar are icons for a sun/moon toggle, the text 'English', and a 'Log Out' button. The main content area is divided into two sections. The top section, titled 'Target Rate', contains a text input field with the value '0,11' and an 'Apply changes' button below it. The bottom section, titled 'Macronutrients', features a circular progress indicator showing '100'. Below this, there are three rows of macronutrient data, each with a corresponding horizontal slider: 'Protein' (43% 142g 2.37g/kg 569cal), 'Carbohydrates' (40% 133g 2.22g/kg 530cal), and 'Fats' (17% 25g 0.42g/kg 225cal). Each row has a colored slider bar (blue for Protein, orange for Carbohydrates, and pink for Fats). At the bottom of the 'Macronutrients' section is an 'Apply changes' button. The footer of the page states 'Developed by: Annya Shutko'.

Macronutrient	Percentage	Weight (g)	Weight (g/kg)	Calories (cal)
Protein	43%	142g	2.37g/kg	569cal
Carbohydrates	40%	133g	2.22g/kg	530cal
Fats	17%	25g	0.42g/kg	225cal

Рисунок 5.10 – Збережені налаштування макронутрієнтів

На рисунку 5.11 виконуємо вихід із системи (log out)

The screenshot shows the 'Log in' page of the NutriLight application. The top navigation bar includes the 'NutriLight' logo and a sun/moon toggle with the text 'English'. The main content area is titled 'Log in' and contains two text input fields: 'Email' and 'Password'. Below these fields are two buttons: 'Submit' (in blue) and 'Sign up' (in white). The background is a dark blue color.

Рисунок 5.11 – Сторінка авторизації

Вводимо коректні дані, підтверджуємо вхід та потрапляємо до сторінки профілю (рис. 5.12).

NutriLight Diary Report Measurements Profile English Log Out

Account

Email
habuba@gmail.com

Name
Habuba

Password

Confirm password

Apply changes

Profile

Height
160

Weight
60

Sex
Female

Physical activity level
light activity (light exercise/sports 1-3 days pt)

Рисунок 5.12 – Сторінка профілю

Перемикаємо тему на рисунку 5.13.

NutriLight Diary Report Measurements **Profile** English Log Out

Account

Email
habuba@gmail.com

Name
Habuba

Password

Confirm password

Apply changes

Profile

Height
160

Weight
60

Sex
Female

Physical activity level
light activity (light exercise/sports 1-3 days pt)

Birth Day

Рисунок 5.13 – Сторінка профілю зі світлою темою

Перемикаємо мову на українську (рис. 5.14).

NutriLight Щоденник Звіт Вимірювання **Профіль** Українська Вийти

Акаунт

Електронна пошта
habuba@gmail.com

Ім'я
Habuba

Пароль

Підтвердити пароль

Застосувати зміни

Профіль

Зріст
160

Вага
60

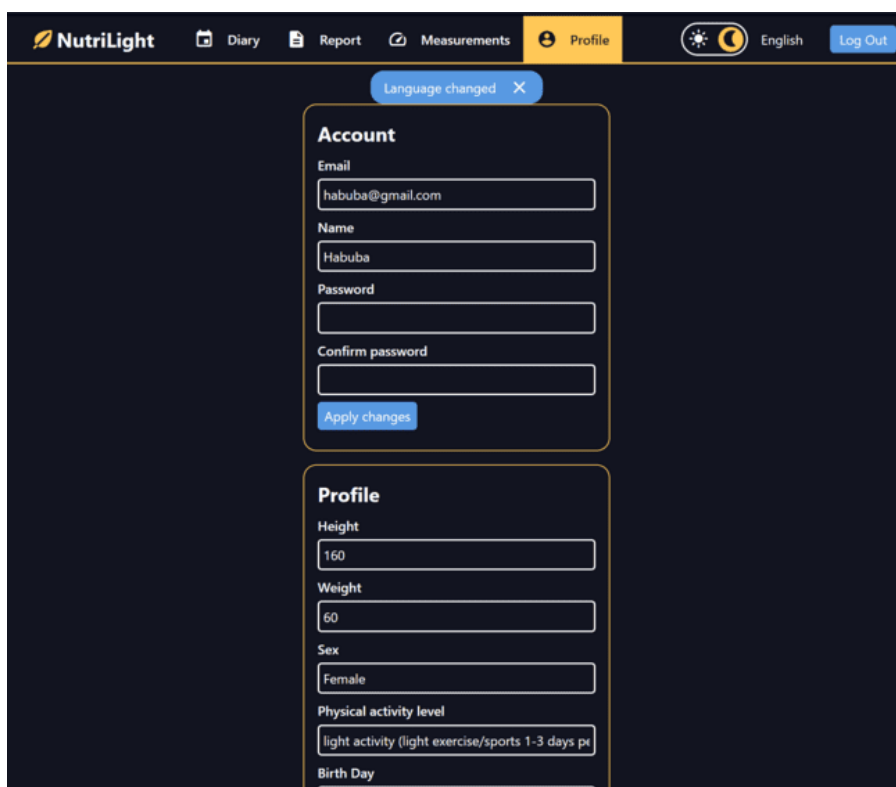
Стать
Жіноча

Рівень фізичної активності
Легка активність (легкі вправи/спорт 1-3 ра)

Дата народження

Рисунок 5.14 – Сторінка профілю зі світлою темою з українською локалізацією

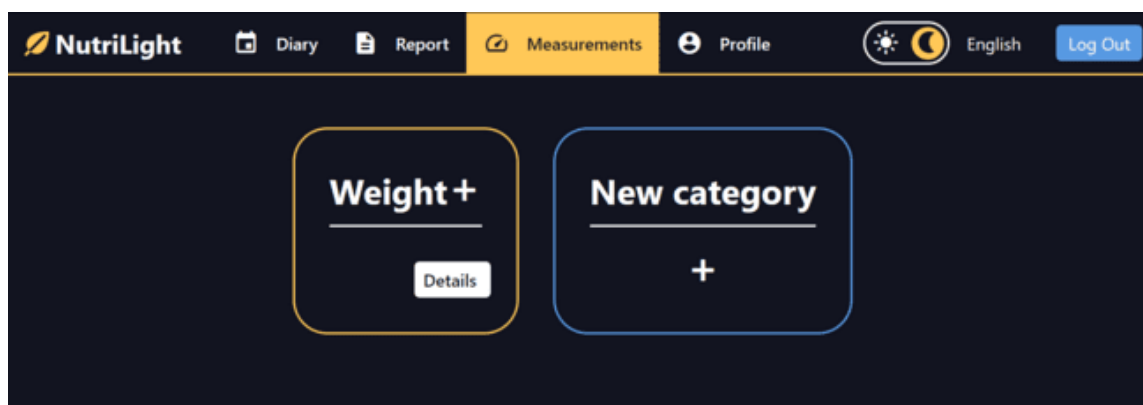
Перемикаємо назад тему та мову (рис. 5.15).



The screenshot shows the 'Profile' page of the NutriLight application. The top navigation bar includes the NutriLight logo and links for Diary, Report, Measurements, Profile, and a Log Out button. A 'Language changed' notification is visible. The main content area is divided into two sections: 'Account' and 'Profile'. The 'Account' section contains fields for Email (habuba@gmail.com), Name (Habuba), Password, and Confirm password, with an 'Apply changes' button. The 'Profile' section contains fields for Height (160), Weight (60), Sex (Female), Physical activity level (light activity (light exercise/sports 1-3 days per week)), and Birth Day.

Рисунок 5.15 – Сторінка профілю із темною темою з англійською локалізацією

Відкриваємо сторінку вимірювань (рис. 5.16).



The screenshot shows the 'Measurements' page of the NutriLight application. The top navigation bar includes the NutriLight logo and links for Diary, Report, Measurements, Profile, and a Log Out button. The main content area features two large buttons: 'Weight +' and 'New category'. The 'Weight +' button has a 'Details' button below it. The 'New category' button has a '+' button below it.

Рисунок 5.16 – Сторінка вимірювань

Відкриваємо форму для додавання ваги (5.17).

The screenshot shows the NutriLight app interface. The top navigation bar includes 'NutriLight', 'Diary', 'Report', 'Measurements' (active), 'Profile', a theme toggle, 'English', and 'Log Out'. The main content area has two buttons: 'Weight +' (highlighted with a yellow border) and 'New category'. A modal titled 'New weight' is open, showing a 'Kilograms' input field with the value '55' and a 'Date' input field with the value '22.11.2023'. A 'Send' button is at the bottom of the modal.

Рисунок 5.17 – Форма додавання ваги

Додаємо 5 різних записи до форми ваги (5.18).

The screenshot shows the NutriLight app interface. The top navigation bar is the same as in Figure 5.17. The main content area has two buttons: 'Weight +' (highlighted with a yellow border) and 'New category'. The 'Weight +' button now displays a list of three weight entries:

Date	Weight (kg)	Action
Nov 25, 2023	59	🗑️
Nov 24, 2023	58	🗑️
Nov 23, 2023	57	🗑️

A 'Details' button is located at the bottom of the list.

Рисунок 5.18 – Форма з доданою вагою

Відкриваємо детальну інформацію і бачимо графік за вагою і прогнозоване значення до досягнення цільової ваги (5.19).



Рисунок 5.19 – Форма детальної інформації з вимірюваннями ваги

Відкриваємо сторінку вимірювань та створюємо нову категорію (5.20).

Weight +

Nov 25, 2023	59	
Nov 24, 2023	58	
Nov 23, 2023	57	

New category +

New category ×

Name

Biceps

Send

Рисунок 5.20 – Форма створення нової категорії

Відкриваємо форму для додавання записів у створену категорію (5.21).

The screenshot shows the NutriLight app's 'Measurements' section. It features three main categories: 'Weight', 'New category', and 'Biceps'. The 'Weight' category displays a list of three measurements: Nov 25, 2023 (59), Nov 24, 2023 (58), and Nov 23, 2023 (57). The 'New category' section is currently active, showing a 'New measurement' form with a value of 30 in centimeters and a date of 22.11.2023. The 'Biceps' category shows a 'Delete All' button and a 'Details' button.

Рисунок 5.21 – Форма додавання записів у створену категорію

Додаємо 4 записи, бачимо, що форма виводить останні 3 (рис. 5.22).

The screenshot shows the NutriLight app's 'Measurements' section after adding four records. The 'Weight' category now displays three measurements: Nov 25, 2023 (59), Nov 24, 2023 (58), and Nov 23, 2023 (57). The 'New category' section is still active. The 'Biceps' category now displays three measurements: Nov 25, 2023 (33), Nov 24, 2023 (32), and Nov 23, 2023 (31). The 'Delete All' and 'Details' buttons are still present in the 'Biceps' category.

Рисунок 5.22 – Форма додавання записів у створену категорію

Відкриваємо детальну інформацію за новою категорією (рис. 5.23).



Рисунок 5.23 – Форма детальної інформації з вимірюваннями нової категорії

Відкриваємо сторінку щоденника харчування (5.24).



Рисунок 5.24 – Сторінка щоденника харчування

Додаємо нові різні записи води (5.25).



Рисунок 5.25 – Сторінка щоденника харчування з доданою водою

Видаляємо кілька записів води (5.26).



Рисунок 5.26 – Результат видалення води

Відкриваємо форму для створення користувацької їжі та заповнюємо поля (5.27).

Рисунок 5.27 – Форма додавання користувацького продукту

Додаємо запис продукту, який відображається в блоці переліку їжі (5.28).

Рисунок 5.28 – Доданий продукт в блоці переліку їжі

Знову переходимо до форми додавання продукту та натискаємо Switch mode, переходимо до пошуку продукту в базі даних, знаходимо продукт та додаємо 300 грам цього продукту (рис. 5.29).

New food

Search:

Select:

Data about product for 100g

Name		
Bologna Beef And Pork Low Fat		
Group	Calories	
Meats	230	
Protein	Carbohydrates	Fat
11.5	2.6	19.3

Calories: Protein: Carbohydrates: Fat:

Grams:

Date: Time:

Рисунок 5.29 – Доданий продукт в блоці переліку їжі

Додаємо новий запис продукту, який відображається в блоці переліку їжі (5.30).



Рисунок 5.30 – Доданий новий продукт в блоці переліку їжі

Відкриваємо форму для створення активності та заповнюємо (5.31).

The screenshot shows the 'New activity' form overlaid on the app's main interface. The form has a dark background with white text and input fields. It includes a close button (X) in the top right corner. The fields are as follows:

- Name:** Horse cart
- Category:** Sport
- Sub category:** horse cart, driving, standing or sitting
- Date:** 25.11.2023
- Time:** 21:27
- Duration:** 01:30

At the bottom of the form is a blue 'Submit' button. The background of the app shows the 'Consumed' and 'Burned' donut charts and the 'Foods' table.

Рисунок 5.31 – Форма додавання активності

Додаємо новий запис активності, який відображається в блоці переліку активностей (5.32).



Рисунок 5.32 – Додана активність в блоці пепеліку доданих активностей

Відкриваємо детальну інформацію харчування за останній тиждень (5.33).

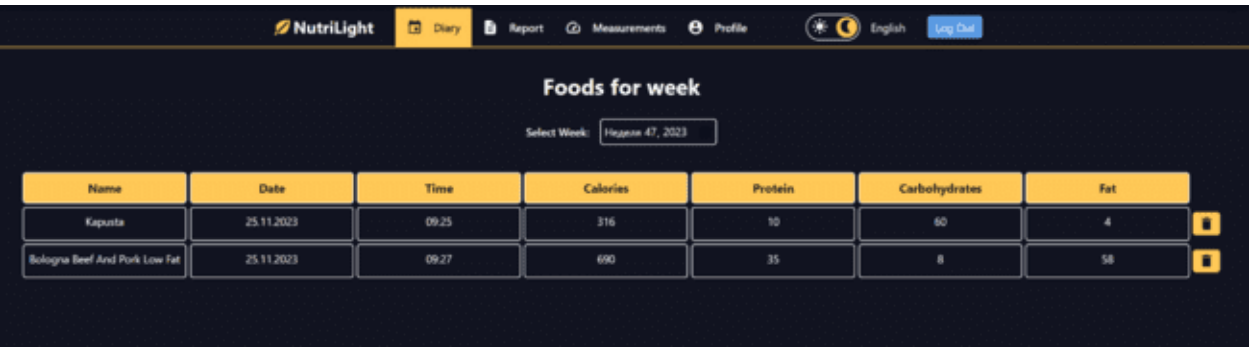


Рисунок 5.33 – Перелік харчування за тиждень

Відкриваємо детальну інформацію активностей за останній тиждень (5.34).

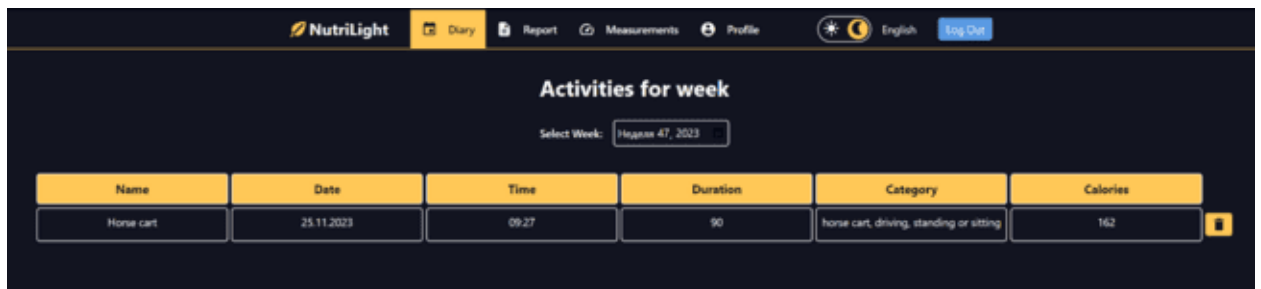


Рисунок 5.34 – Перелік активностей за тиждень

Відкриваємо сторінку звіту та перевіряємо графік балансу за певний період (рис. 5.35).



Рисунок 5.35 – Графік балансу

Перевіряємо графік харчування за період (рис. 5.36)



Рисунок 5.36 – Графік харчування

Перевіряємо графік активностей за період (рис. 5.37).

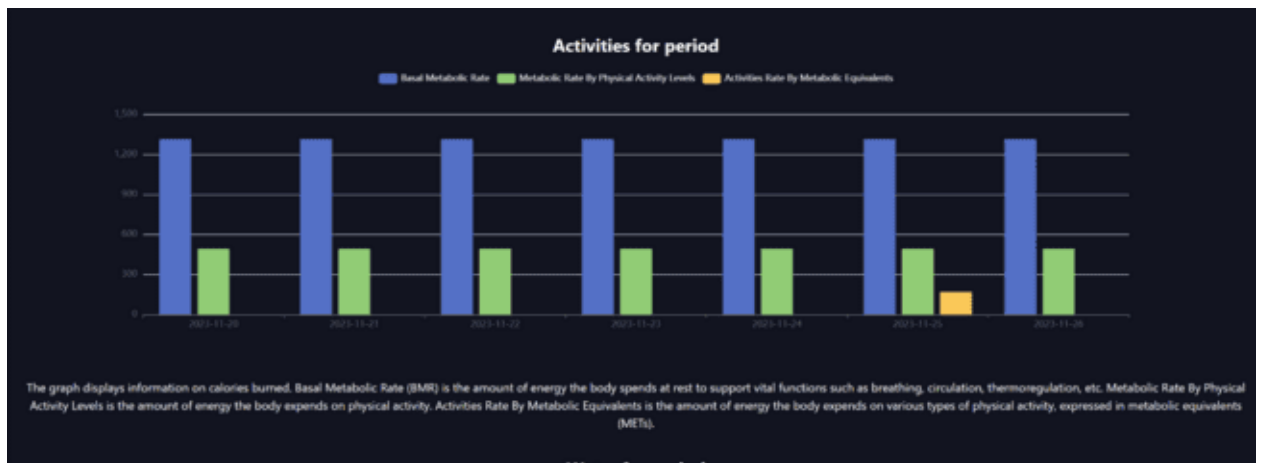


Рисунок 5.37 – Графік активностей за період

Перевіряємо графік споживання води за період (рис. 5.38).



Рисунок 5.38 – Графік споживання води за період

Вводимо дані для розрахунку рекомпозиції тіла та бачимо результат (5.39).

Calculate recomposition

Body Mass

Increases

Waist circumference

Remains the same

Strength

Increases

Result: Muscle mass increases. Fat mass does not change.

Рисунок 5.39 – Розрахунок рекомпозиції

Базуючись на виконаних кроках та результатах, можна зробити висновок, що тестування програми пройшло успішно. Усі вказані етапи ручного тестування фронтенду були виконані без помилок та з відображенням очікуваних результатів. Функціональність елементів інтерфейсу, кнопок, а також взаємодія користувача з застосунком були перевірені, і програма правильно реагувала на введені дії.

Під час тестування було виконано послідовність складних дій, що охоплювали різноманітні функції програми, такі як створення профілю, зміна налаштувань, введення даних та їх обробка, що підтверджує стабільну та правильну роботу програми у реальних умовах використання. Таким чином, можна зазначити, що тестування пройшло успішно, а програма виявилася працездатною та готовою до подальшого впровадження.

5.3 Оптимізація розробленого застосунка

Основний акцент оптимізації був зроблений на фронтенді, оскільки при повільній роботі бекенду завжди можна покращити компоненти сервера, у той час коли оптимізація з боку користувача є життєво важливою, особливо при розробці складних застосунків.

Для оптимізації було виконано багато ефективних рішень, пов'язаних з особливостями Angular.

5.3.1 Lazy Loading Modules

Angular надає можливість використовувати lazy-loading, що дозволяє завантажувати модулі тільки тоді, коли вони необхідні. Це означає, що модуль буде завантажуватися лише у випадку, якщо користувач переходить до маршруту, пов'язаного з цим модулем. Команда Angular вбудувала цю функцію безпосередньо в маршрутизатор, тому вся ця логіка відбувається автоматично під капотом Angular.

Однак для використання цієї функціональності потрібно розділити програму на окремі модулі. У наведеному прикладі представлений код реалізації маршрутів з використанням підходу lazy-loading.

```
const routes: Routes = [
  { path: '', redirectTo: '/diary', pathMatch:
'full' },
  {
    path: 'diary',
    loadChildren: () =>
      import('./diary/diary.module').then((m) =>
m.DiaryModule),
    canMatch: [AuthGuard],
  },
  {
    path: 'report',
    loadChildren: () =>
      import('./report/report.module').then((m) =>
m.ReportModule),
  },
  {
    path: 'measurements',
    loadChildren: () =>
      import('./measurements/measurements.module')
.then(
        (m) => m.MeasurementsModule,
      ),
    canMatch: [AuthGuard],
  },
  {
    path: 'account',
    loadChildren: () =>
      import('./account/account.module').then((m)
=> m.AccountModule),
  },
  { path: '**', redirectTo: '/account/details' },
];
```

Цей код представляє об'єкт маршрутів у Angular, де кожен шлях (path) пов'язаний з певним lazy-loaded модулем. loadChildren вказує на те, який модуль потрібно завантажувати при активації цього маршруту.

Це дозволяє покращити продуктивність застосунка, оскільки лише необхідні модулі завантажуються при роботі з конкретними функціями, зменшуючи загальний час завантаження і прискорюючи завантаження сторінок.

5.3.2 Change Detection OnPush

Angular використовує механізм перевірки змін (Change Detection), який за замовчуванням автоматично перевіряє всі компоненти для виявлення змін у даних та оновлення представлення. Однак, цей процес, якщо програма стає складнішою та має значну кількість компонентів, може призвести до сповільнення швидкості роботи програми через постійні перевірки та оновлення.

Стратегія Change Detection OnPush – це альтернативний підхід до перевірки змін, який спрацьовує ефективніше. У відміну від стратегії за замовчуванням, яка перевіряє компоненти при кожній зміні у програмі, стратегія OnPush реагує лише на певні події, такі як зміни в параметрах `@Input` або явне викликання механізму виявлення змін.

Для активації стратегії OnPush, необхідно використовувати декоратор `@Component` у компоненті та вказати параметр `changeDetection` зі значенням `ChangeDetectionStrategy.OnPush`, як показано у наступному прикладі.

```
@Component({
  selector: 'app-details',
  templateUrl: './details.component.html',
  styleUrls: ['./details.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
```

Це дозволяє Angular використовувати оптимізований підхід до виявлення змін: компонент буде оновлювати своє представлення тільки при зміні вхідних параметрів або в разі явного виклику виявлення змін у коді

компонента. Такий підхід покращує продуктивність застосунка, особливо при роботі з великою кількістю даних чи складних компонентів, допомагаючи уникнути зайвих перевірок та оновлень, які не є необхідними.

5.3.3 Async Pipe & Unsubscription

У контексті Angular та RxJS, підписка відноситься до ключового аспекту управління потоками даних. Observable є об'єктом, який генерує послідовність подій або значень, що змінюються з часом, а підписка є механізмом, який надає можливість реагувати на ці події чи значення, отримуючи їх з Observable. Підписка представляє собою одноразовий ресурс, який дозволяє слідкувати за потоком даних, але може бути скасована в будь-який момент для звільнення ресурсів [87].

Важливим аспектом підписки є її метод для скасування. Цей метод, зазвичай називається `unsubscribe()`, є інструментом для звільнення ресурсів, пов'язаних з підпискою. Виклик цього методу призводить до припинення слідкування за Observable та припинення отримання подій чи значень. Відміна підписки є важливою практикою уникнення утечок пам'яті та надмірного використання ресурсів, особливо при роботі з великою кількістю Observable або в складних програмах Angular, де керування пам'яттю має значення.

Стандартний метод підписки дуже простий – результат підписки записується у змінну з типом `Subscription`. І коли необхідно завершити цю підписку, наприклад, у lifecycle hook-у `ngOnDestroy`, викликайте метод `unsubscribe()`.

Проте існують багато більш зручних і практичних методів відписки, які, використовуються у написаному застосунку. Наприклад, це відписка за допомогою методу оператора `takeUntil`.

```
ngOnInit(): void {  
    this.selectedWeek.valueChanges  
        .pipe(takeUntil(this.destroy$))
```

```

        .subscribe(() => this.loadActivities());
    }

```

У цьому фрагменті коду підписка автоматично буде відписана після спрацювання наступної ітерації об'єкта Observable, який передається в оператор `takeUntil`. У цьому випадку `this.destroy$` спрацює при спрацюванні lifecycle hook-а `ngOnDestroy`. І це реалізовано за допомогою успадкування загального абстрактного класу `DefaultAbstractComponent`.

```

@Component({ template: '' })
export abstract class DefaultAbstractComponent
implements OnDestroy {
    protected readonly destroy$: Subject<void> = new
    Subject<void>();

    ngOnDestroy(): void {
        this.destroy$.next();
        this.destroy$.complete();
    }
}

```

У випадку, коли ми очікуємо отримати лише одне значення, можна використовувати `take(1)` або `first()`. У цьому випадку підписка автоматично скасовується після отримання першого значення. Різниця між ними полягає в тому, що якщо у потоці немає елементів, то `first()` видасть помилку, тоді як `take(1)` закриє потік без елементів [88].

```

this.measurementsService
    .getMeasurements(+this.id, 0, 200)
    .pipe(first())
    .subscribe((measurements: IMeasurement[]) => {
        this.measurements$.next(measurements);
    });
this.http
    .patch<IAccount>('/api/account/change', body)
    .pipe(
        take(1),
        catchError(() => {

```

```

        return EMPTY;
    }),
)
.subscribe((account: IAccount) => {
    this.setAccount(account);
    this.router.navigateByUrl(DEFAULT_PAGE);
    this.modalsService.addNotification({
        text:
NotificationText.FORM_UPDATE_SUCCESSFUL,
        type: NotificationType.INFO,
    });
});
});

```

Також існує надзвичайно зручний функціонал у вигляді ріпе під назвою `asynps`, який використовується наступним чином.

```

<div [class]='wrapper ' + (theme$ | asynps) + '-
theme' ">

```

Він автоматично підписується на переданий йому `Observable`, а також автоматично відписується від нього при знищенні компонента.

Отже, підписка виступає як засіб з'єднання з `Observable`, а її скасування може бути важливим кроком для ефективного управління ресурсами, що допомагає підтримувати застосунок ефективним та уникнути можливих проблем із пам'яттю. Існує багато зручних способів її виконання.

5.3.4 Pipes

`Pipes` може використовуватися для підвищення продуктивності застосунку та ангулярної оптимізації змінної області видимості (`Change Detection Optimization`).

У реалізації застосунків, де велика кількість даних обробляється або відображається в шаблонах `Angular`, використання `pipes` може сприяти підвищенню продуктивності. Наприклад, при відображенні великої таблиці або списку даних, де потрібно виконати фільтрацію, сортування або інші

операції над цими даними, pipes можуть бути ефективнішими, ніж виклик функцій безпосередньо у шаблоні.

При використанні pipes, результати обробки даних кешуються, тобто Angular зберігає результати операцій над даними та перевикористовує їх при наступних викликах. Це може зменшити кількість операцій, які потрібно виконати для кожного циклу виявлення змін та, таким чином, покращити продуктивність.

Однією з ключових особливостей Angular є механізм виявлення змін (Change Detection), який перевіряє зміни в даних і оновлює відповідні частини UI. У разі великої кількості даних або складної логіки, яка пов'язана з обробкою цих даних, може виникати проблема швидкодії через постійні перевірки змін.

Pipes можуть допомогти оптимізувати цей процес, оскільки Angular має певні вбудовані механізми для оптимізації роботи з пайпами. Angular може застосовувати стратегії кешування (memoization) та інші оптимізації для пайпів. Використання pipes дозволяє Angular краще контролювати та оптимізувати процес виявлення змін, зменшуючи навантаження на систему.

Приклад використання пайпів можна побачити тут.

```
<li class="item__property">{{ food.name }}</li>
<li class="item__property">{{ food.dateTime |
date: 'dd.MM.yyy' }}</li>
<li class="item__property">{{ food.dateTime |
date: 'hh:mm' }}</li>
<li class="item__property">{{ food.calories |
round }}</li>
<li class="item__property">{{ food.protein | round
}}</li>
<li class="item__property">{{ food.carbohydrates |
round }}</li>
<li class="item__property">{{ food.fat | round
}}</li>
```

5.3.5 Dependencies and Minification

Оптимізація Angular застосунків важлива для їх швидкої роботи та ефективного використання ресурсів. Інструменти, такі як webpack-bundle-analyzer, сприяють цьому процесу через аналіз та візуалізацію збірок застосунку. Це дозволяє краще розуміти структуру та розмір файлів, які включені до кінцевої збірки застосунку.

Використання webpack-bundle-analyzer спільно з аналізом статистики дозволяє ефективно виявляти та вирішувати проблеми з розміром та продуктивністю Angular застосунку. Це сприяє оптимальній роботі та використанню ресурсів.

На рисунку 5.40 зображено приклад аналізу за допомогою webpack-bundle-analyzer.

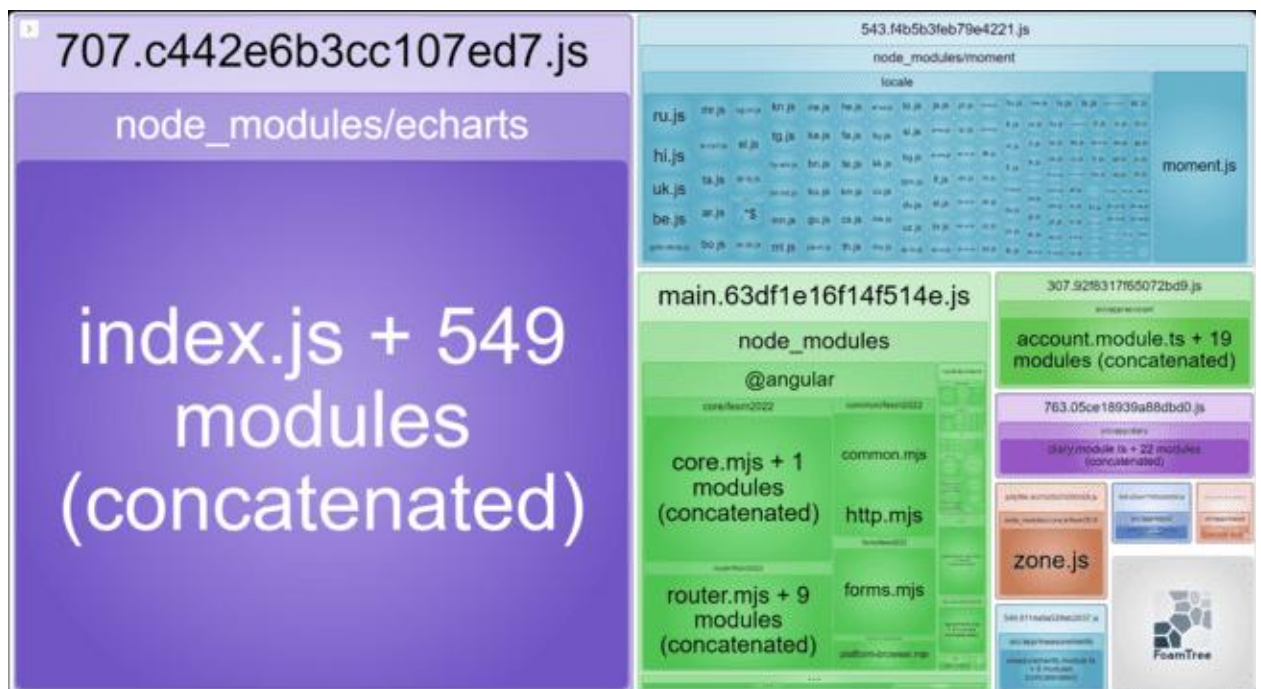


Рисунок 5.40 – Візуалізація збірок застосунку

Мініфікація коду також є потужним інструментом для оптимізації. Багато символів у JavaScript-кодi, такі як пробіли, символи нового рядка, коментарі та розділювачі блоків, використовуються лише для читабельності та

візуальних цілей. Вони не є обов'язковими для коректної роботи коду. Процес мініфікації видаляє ці символи, спрощує імена та ігнорує недосяжний код. Зменшуючи обсяг коду, можна прискорити завантаження сторінки та час виконання.

При компіляції Angular автоматично створює ефективний код. Ось приклад такого коду.

```
"use strict";
(self.webpackChunknutri_light_client=self.webpackChunknutri_light_client||[]).push([[179],{8451:(Ve,ne,E)=>{E.d(ne,{G:()=>j});var a=E(8645),N=E(8926);let j=()=>{var
...

```

Цей підхід спрощує код, поліпшує його продуктивність та допомагає забезпечити оптимальну роботу Angular застосунків.

ВИСНОВКИ

Дослідження та програмна реалізація застосунку для оптимізації харчування відкривають нові можливості для користувачів у сфері здорового способу життя. Розроблена програма створена з метою полегшення процесу ведення контролю за харчуванням, фізичною активністю та досягненням поставлених цілей у плані здоров'я.

Засоби розробки, які використовуються у даному проєкті, включають Angular для фронтенду, Nest.js для серверної частини, Prisma для взаємодії з базою даних SQLite, а також використання методів ідентифікації, аутентифікації, авторизації для забезпечення безпеки застосунку.

Angular було обрано за його модульність та компонентну структуру, а також він містить у собі за замовчуванням основну необхідну частину функціоналу для реалізації проєкту.

Nest.js, як і Angular використовує TypeScript, що робить код більш безпечним і зрозумілим, а також спрощує його тестування. Також Nest.js використовує архітектуру мікросервісів, яка також підтримується в Angular і є популярною тенденцією в розробці вебзастосунків. Ця архітектура дозволяє створювати програми, які легко масштабуються та підтримуються. Крім того, фреймворк використовує декоратори для розширення функціональності класів. Це робить код більш виразним та читабельним.

У якості ORM-інтерфейсу обрано Prisma, оскільки вона полегшує взаємодію з базою даних та спрощує операції з даними (такі як створення та управління) за допомогою моделей, які дозволяють визначити схему бази даних, та зручному інтерфейсу взаємодії через сервіси.

У якості БД обрано SQLite через свою легкість та компактність, що робить її ідеальним варіантом для маленьких та середніх проєктів. До того ж SQLite є реляційною базою даних, що дозволяє зберігати та отримувати дані в структурованому форматі.

Архітектура аутентифікації та авторизації працює на взаємодії між Angular та Nest.js, забезпечуючи безпеку та керування доступом для користувачів застосунку за допомогою sessionToken, що зберігаються в базі даних.

Однією з головних переваг цього проєкту для фітнесу є його комплексність. Застосунок не лише надає можливість вести облік спожитих та спалених калорій, а й дозволяє користувачам контролювати макронутрієнти, водний баланс, вагу, фізичні показники тіла та активності. Інтеграція цих функцій у єдиному інтерфейсі робить цей застосунок надзвичайно корисним для фітнесу та здорового способу життя.

Користь цього проєкту полягає в тому, що він допомагає користувачам усвідомити їхні щоденні харчові звички та фізичну активність, що є ключовими компонентами здорового способу життя. Можливість відстеження продуктів споживання, підрахунок калорій, контроль над спожитими поживними речовинами, а також аналіз графіків ваги, активності та харчування за певний період дозволяє користувачам більш об'єктивно підходити до досягнення своїх цілей у сфері фітнесу та здоров'я.

Такий застосунок може сприяти у створенні та підтримці здорових звичок, мотивувати до активного способу життя та допомагати досягати бажаних результатів у фітнесі. Він є важливим інструментом для тих, хто прагне не лише контролювати своє харчування та активність, а й досягти оптимального стану здоров'я й фізичної форми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Соціально-демографічні характеристики домогосподарств України (за даними вибіркового обстеження умов життя домогосподарств). Державна служба статистики України. [Електрон. ресурс]. – Режим доступу: https://ukrstat.gov.ua/operativ/operativ2020/gdvdg/sdhd/Arh_sdhd.htm
2. Overweight & Obesity Statistics [Electronic resource]. – Access mode: <https://www.niddk.nih.gov/health-information/health-statistics/overweight-obesity>
3. The Art of Counting Calories [Electronic resource]. – Access mode: <https://www.trainingpeaks.com/blog/the-art-of-counting-calories/>
4. Counting calories – useful or overrated? [Electronic resource]. – Access mode: <https://www.verival.at/english/blog/calorie-counting-meaningful-or-overrated/>
5. Харчування для спортсменів: їжа для спортсменів на кожен день foodex [Електрон. ресурс]. – Режим доступу: <https://foodexhub.com.ua/blog/yizha-dlya-sportsmeniv>
6. Їжа та спорт. Як скласти правильний режим харчування — поради лікаря-дієтолога [Електрон. ресурс]. – Режим доступу: <https://health.nv.ua/ukr/tvov-zdorove-fundament-tvoego-budushchego/pravilne-harchuvannya-dlya-sportsmena-yak-pravilno-sklasti-rezhim-harchuvannya-ekspert-50156864.html>
7. Основи правильного харчування для спортсменів [Електрон. ресурс]. – Режим доступу: <https://bcaa.ua/ua/podderzhka/osnovy-pravilnogo-pitaniya-dlya-sportsmenov>
8. Dietary Guidelines for Americans 2020–2025 [Electronic resource]. – Access mode: <https://www.dietaryguidelines.gov/>
9. ISSN exercise & sports nutrition review update: research & recommendations [Electronic resource]. – Access mode: <https://jissn.biomedcentral.com/articles/10.1186/s12970-018-0242-y>

10. Why is diet so important for athletes? [Electronic resource]. – Access mode: <https://www.medicalnewstoday.com/articles/nutrition-for-athletes#macronutrients>
11. International Society of Sports Nutrition Position Stand: protein and exercise [Electronic resource]. – Access mode: <https://jissn.biomedcentral.com/articles/10.1186/s12970-017-0177-8>
12. Nutrition for Athletes [Electronic resource]. – Access mode: <https://familydoctor.org/nutrition-for-athletes/>
13. Які продукти містять корисні та шкідливі жири [Електрон. ресурс]. – Режим доступу: https://health.24tv.ua/yaki_produkty_mistyat_korisni_ta_shkidlivi_zhiri_spisok_n894155
14. Вчимося розрізняти насичені та ненасичені жири [Електрон. ресурс]. – Режим доступу: https://korisno.24tv.ua/yak-rozriznyati-nasicheni-nenasicheni-zhiri-yaki-zhiri-shkidlivi_n2228158
15. Vitamins and minerals as ergogenic aids [Electronic resource]. – Access mode: <https://pubmed.ncbi.nlm.nih.gov/18607225/>
16. Food and Nutrition Board, Institute of Medicine. Dietary reference intakes for vitamin A, vitamin K, arsenic, boron, chromium, copper, iodine, iron, manganese, molybdenum, nickel, silicon, vanadium, and zinc [Electronic resource]. – Access mode: <https://www.ncbi.nlm.nih.gov/books/NBK222310/>
17. Micronutrient requirements of physically active women: what can we learn from iron? [Electronic resource]. – Access mode: <https://www.sciencedirect.com/science/article/pii/S0002916523282167>
18. Solutions for micronutrient deficiency [Electronic resource]. – Access mode: <https://blogs.scientificamerican.com/guest-blog/solutions-for-micronutrient-deficiency/>
19. Guidelines on Food Fortification with Micronutrients [Electronic resource]. – Access mode: https://cdn.who.int/media/docs/default-source/micronutrients/gff-contents-en.pdf?sfvrsn=f964afe1_2

20. Nutrition and athletic performance [Electronic resource]. – Access mode: <https://medlineplus.gov/ency/article/002458.htm>
21. Користь води для організму людини [Електрон. ресурс]. – Режим доступу: <https://mywatershop.com.ua/blog/korist-vodi-dlya-organizmu-lyudini/>
22. Гідратація: Вода як Ключовий Елемент Здорового Способу Життя [Електрон. ресурс]. – Режим доступу: <https://rd168.com.ua/gidratacija-voda-jak-kljuchovij-element-zdorovogo/>
23. Water – a vital nutrient [Electronic resource]. – Access mode: <https://www.betterhealth.vic.gov.au/health/healthyliving/water-a-vital-nutrient>
24. Скільки води пити в день? [Електрон. ресурс]. – Режим доступу: <https://a.com.ua/articles/voda-i-pitnijj-rezhim>
25. TDEE Calculator: Calculate Your Total Daily Energy Expenditure [Electronic resource]. – Access mode: <https://www.forbes.com/health/body/tdee-calculator/>
26. Exercise is great for health—but it isn't the key to weight loss. [Electronic resource]. – Access mode: <https://guidingstars.com/nutrition-science/exercise-is-great-for-health-but-it-isnt-the-key-to-weight-loss/>
27. Cunningham Equation [Electronic resource]. – Access mode: <https://www.apollodiet.com/blog/post/cunningham-equation>
28. Calorie Restriction Calculator [Electronic resource]. – Access mode: <https://www.scientificpsychic.com/health/cron1.html>
29. Energy expenditure during common sitting and standing tasks: examining the 1.5 MET definition of sedentary behaviour [Electronic resource]. – Access mode: <https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-015-1851-x>
30. What Exactly Are METs, and What Should You Know About Them? [Electronic resource]. – Access mode: <https://www.healthline.com/health/what-are-mets#definition>

31. Resting metabolic rate: how to calculate and improve yours [Electronic resource]. – Access mode: <https://blog.nasm.org/nutrition/resting-metabolic-rate-how-to-calculate-and-improve-yours>

32. Harris-Benedict Equation Calculator for Nutrition Professionals [Electronic resource]. – Access mode: <https://nutrium.com/blog/harris-benedict-equation-calculator-for-nutrition-professionals/>

33. Mifflin-St. Jeor for nutrition professionals [Electronic resource]. – Access mode: <https://nutrium.com/blog/mifflin-st-jeor-for-nutrition-professionals/>

34. Which formula are recommended by nutritionists and scientists to measure "BASAL METABOLIC RATE" ? [Electronic resource]. – Access mode: https://www.researchgate.net/post/which_formula_are_recommended_by_nutritionists_and_scientists_to_measure_BASAL_METABOLIC_RATE

35. The Weekly Caloric Method [Electronic resource]. – Access mode: <https://www.elitefts.com/education/training/the-weekly-caloric-method/>

36. Should You Track Your Calories Weekly Instead Of Daily? [Electronic resource]. – Access mode: <https://www.burnthefatinnercircle.com/public/should-you-track-calories-weekly-not-daily.cfm>

37. FatSecret Diet Review [Electronic resource]. – Access mode: https://dietsinreview.com/diet_column/12/fatsecret-diet-review/

38. What are the Best Weight Loss Apps for iPhone? [Electronic resource]. – Access mode: <https://www.matellio.com/blog/what-are-the-best-weight-loss-apps-for-iphone/>

39. Lifesum Review: A Complete Breakdown Including Pros, Cons, and More [Electronic resource]. – Access mode: <https://www.active.com/nutrition/articles/lifesum-review>

40. 8 Best calorie counting apps to track progress & stay on top of your healthy eating [Electronic resource]. – Access mode: <https://www.womenshealthmag.com/uk/gym-wear/tech/a705360/best-calorie-counting-app-apps/>

41. Could MyNetDiary Help Me Lose Weight? My Honest Review [Electronic resource]. – Access mode: <https://healthreporter.com/mynetdiary-review/>
42. Cronometer Review: Easy Calorie Counting on the Go! [Electronic resource]. – Access mode: <https://thefitcareerist.com/cronometer-review/>
43. Best Free Food Tracking Apps in 2023 [Electronic resource]. – Access mode: <https://blog.gitnux.com/best-free-food-tracking-apps/>
44. The 12 Best Apps to Help You Meet Your Weight Loss Goals in 2023 [Electronic resource]. – Access mode: <https://www.healthline.com/nutrition/best-weight-loss-apps>
45. Lifesum Review [Electronic resource]. – Access mode: <https://www.pcmag.com/reviews/lifesum>
46. MyNetDiary Review [Electronic resource]. – Access mode: <https://feastgood.com/mynetdiary-review/>
47. Cronometer Review [Electronic resource]. – Access mode: <https://www.fithealthymomma.com/cronometer-review/>
48. Cronometer vs MyFitnessPal: Expert Verdict [Electronic resource]. – Access mode: <https://www.healthyperformers.com/cronometer-vs-myfitnesspal/>
49. MVC vs MVVM – Difference Between Them [Electronic resource]. – Access mode: <https://www.guru99.com/mvc-vs-mvvm.html>
50. SPA (Single-page application) [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>
51. What is server-side rendering: definition, benefits and risks [Electronic resource]. – Access mode: <https://solutionshub.epam.com/blog/post/what-is-server-side-rendering>
52. Server-side rendering (SSR) with Angular Universal [Electronic resource]. – Access mode: <https://angular.io/guide/universal>
53. REST API Tutorial – REST Client, REST Service, and API Calls Explained With Code Examples [Electronic resource]. – Access mode:

<https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/>

54. GraphQL [Electronic resource]. – Access mode: <https://graphql.org>

55. Microservices architecture design [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/azure/architecture/microservices/>

56. Free book on design patterns and component patterns for building powerful web apps [Electronic resource]. – Access mode: <https://www.patterns.dev>

57. An overview of HTTP [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

58. CSS: Cascading Style Sheets [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/CSS>

59. HTML: HyperText Markup Language [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTML>

60. CSS preprocessor [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor

61. List of 10 Best Front end Frameworks to Use For Web Development [Electronic resource]. – Access mode: <https://www.monocubed.com/blog/best-front-end-frameworks/>

62. What is Angular?: Architecture, Features, and Advantages [Electronic resource]. – Access mode: <https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular>

63. What is Vue js? The Pros and Cons of Vue.JS [Electronic resource]. – Access mode: <https://www.tatvasoft.com/outsourcing/2021/10/what-is-vue-js-and-its-benefits.html>

64. What are the advantages of React.js ? [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/what-are-the-advantages-of-react-js/>

65. 30 Best Web Development Frameworks for 2023: A Comprehensive Guide [Electronic resource]. – Access mode: <https://dev.to/bravinsimiyu/30-best-web-development-frameworks-for-2023-a-comprehensive-guide-512i>

66. What is Java? [Electronic resource]. – Access mode:
<https://www.ibm.com/topics/java>

67. Spring Boot [Electronic resource]. – Access mode:
<https://spring.io/projects/spring-boot>

68. Node.js v21.1.0 documentation [Electronic resource]. – Access mode:
<https://nodejs.org/api/>

69. Advantages and disadvantages of NestJS [Electronic resource]. – Access mode:
<https://themobilereality.com/blog/advantages-and-disadvantages-of-nestjs>

70. An Intro to ExpressJS [Electronic resource]. – Access mode:
<https://flexiple.com/express-js/deep-dive>

71. C# documentation [Electronic resource]. – Access mode:
<https://learn.microsoft.com/en-us/dotnet/csharp/>

72. Overview of ASP.NET Core [Electronic resource]. – Access mode:
<https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0>

73. Sitecore Documentation [Electronic resource]. – Access mode:
<https://doc.sitecore.com>

74. Python 3.12.0 documentation [Electronic resource]. – Access mode:
<https://docs.python.org/3/>

75. Flask's documentation [Electronic resource]. – Access mode:
<https://flask.palletsprojects.com/en/3.0.x/>

76. Django Web Framework (Python) [Electronic resource]. – Access mode:
<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>

77. PHP Manual [Electronic resource]. – Access mode:
<https://www.php.net/manual/en/>

78. Benefits of Laravel Web App Development [Electronic resource]. – Access mode:
<https://www.advanceecom solutions.com/blog/benefits-of-laravel-web-app-development/>

79. Yii 2.0 Benefits and Features [Electronic resource]. – Access mode:
<https://www.optisolbusiness.com/insight/yii-2-0-benefits-and-features>

80. What Is a Database? [Electronic resource]. – Access mode:
<https://www.coursera.org/articles/what-is-database>

81. System Properties Comparison MongoDB vs. MySQL vs. SQLite [Electronic resource]. – Access mode: <https://db-engines.com/en/system/MongoDB%3BMySQL%3BSQLite>

82. MongoDB [Electronic resource]. – Access mode:
<https://www.techtarget.com/searchdatamanagement/definition/MongoDB>

83. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems [Electronic resource]. – Access mode:
<https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>

84. What Is Redis? [Electronic resource]. – Access mode:
<https://developer.redis.com/explore/what-is-redis/>

85. NestJS Docs Controllers [Electronic resource]. – Access mode:
<https://docs.nestjs.com/controllers>

86. Prisma Docs Set up Prisma [Electronic resource]. – Access mode:
<https://docs.nestjs.com/recipes/prisma#set-up-prisma>

87. RxJS Docs Subscription [Electronic resource]. – Access mode:
<https://rxjs.dev/guide/subscription>

88. take(1) vs first() [Electronic resource]. – Access mode:
[https://stackoverflow.com/questions/42345969/take1-vs-first\(\)](https://stackoverflow.com/questions/42345969/take1-vs-first())

89. ISO 5807-85: [Electronic resource]. – Access mode:
<https://standards.iteh.ai/catalog/standards/iso/e1a4e558-03f1-46cd-9ff7-a9abc479294a/iso-5807-1985>

ДОДАТОК А
Текст програми

A.1 src\shared\guards\auth.guard.ts

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private sessionService: SessionService) {}

  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    const request = context.switchToHttp().getRequest();

    const sessionToken = request.headers['session-token'];

    return this.sessionService.sessionValid(sessionToken);
  }
}
```

A.2 src\shared\services\session.service.ts

```
@Injectable()
export class SessionService {
  public constructor(
    private readonly sessionDbService: SessionDbService,
    private readonly accountDbService: AccountDbService,
    private readonly profileDbService: ProfileDbService,
  ) {}

  public async sessionValid(sessionToken?: string): Promise<boolean> {
    if (!sessionToken) return false;
    const session: Session | null = await this.sessionDbService.session({
      sessionToken,
    });
    return session ? session.dueDate.getTime() > Date.now() : false;
  }

  public async accountBySession(sessionToken: string): Promise<Account> {
    const session = await this.sessionDbService.session({
      sessionToken,
    });
    const account = await this.accountDbService.account({
      id: session.accountId,
    });
    return account;
  }

  public async profileBySession(sessionToken: string): Promise<Profile> {
    const account = await this.accountBySession(sessionToken);

    const profile = await this.profileDbService.profile({
      accountId: account.id,
    });

    return profile;
  }
}
```

A.3 src\profile\profile.service.ts

```
@Injectable()
export class ProfileService {
```

```

public constructor(
  private readonly profileDbService: ProfileDbService,
  private readonly sessionService: SessionService,
  private readonly weightDbService: WeightDbService,
  private readonly caloricityTdeeDbService: CaloricityTdeeDbService,
) {}

public async createProfile(
  sessionToken: string,
  body: ICreateProfileDto,
): Promise<IProfileWithCaloricityDto> {
  const account = await this.sessionService.accountBySession(sessionToken);

  const profile = await this.profileDbService.createProfile({
    ...body,
    account: {
      connect: { id: account.id },
    },
  });

  const
      caloricity
      =
      await
this.caloricityTdeeDbService.createCaloricityTDEE({
  protein: 30,
  carbohydrates: 40,
  fat: 30,
  profile: {
    connect: { id: profile.id },
  },
});

  return { ...profile, caloricity };
}

public async updateProfile(
  sessionToken: string,
  body: IUpdateProfileDto,
): Promise<IProfileWithCaloricityDto> {
  const profile = await this.profileDbService.profile({
    accountId:
      (await
this.sessionService.accountBySession(sessionToken)).id,
  });

  if (body.weight)
    await this.weightDbService.createWeight({
      weight: body.weight,
      date: new Date(),
      profile: {
        connect: {
          id: profile.id,
        },
      },
    });

  const newProfile = await this.profileDbService.updateProfile({
    where: { id: profile.id },
    data: {
      ...body,
    },
  });

  const caloricity = await this.caloricityTdeeDbService.caloricityTDEE({
    profileId: profile.id,
  });
}

```

```

    return { ...newProfile, caloricity };
  }

  public async setCaloricity(
    sessionToken: string,
    body: IUpdateCaloricityDto,
  ): Promise<CaloricityTDEE> {
    const profile = await this.sessionService.profileBySession(sessionToken);
    let caloricity: CaloricityTDEE;
    try {
      caloricity = await this.caloricityTdeeDbService.updateCaloricityTDEE({
        data: {
          protein: body.protein,
          carbohydrates: body.carbohydrates,
          fat: body.fat,
        },
        where: {
          profileId: profile.id,
        },
      });
    } catch {
      caloricity = await this.caloricityTdeeDbService.createCaloricityTDEE({
        protein: body.protein,
        carbohydrates: body.carbohydrates,
        fat: body.fat,
        profile: { connect: { id: profile.id } },
      });
    }
    return caloricity;
  }

  public async getProfile(
    sessionToken: string,
  ): Promise<IProfileWithCaloricityDto> {
    const profile = await this.sessionService.profileBySession(sessionToken);
    const caloricity = await this.caloricityTdeeDbService.caloricityTDEE({
      profileId: profile.id,
    });
    return { ...profile, caloricity };
  }
}

```

A.4 prisma\schema.prisma

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Account {
  id          Int          @id @default(autoincrement())
  email       String       @unique
  name        String       @unique
  imageUrl    String
  password    String
  sessions    Session[]
  profile     Profile?
}

```

```

model Session {
  id          Int          @id @default(autoincrement())
  sessionToken String      @unique
  createDate  DateTime     @default(now())
  dueDate     DateTime
  account     Account      @relation(fields: [accountId], references: [id])
  accountId   Int
}

model Profile {
  id          Int          @id @default(autoincrement())
  sex         String       @default("none")
  PAL         Float        @default(1.375)
  height      Float
  weight      Float
  weightHistory Weight[]
  measurementObject MeasurementObject[]
  birthDay    DateTime
  targetWeight Float
  targetRate  Float
  caloricityTDEE CaloricityTDEE?
  foodHistory Food[]
  activitiesPresets PresetActivity[]
  activitiesHistory Activity[]
  waterHistory Water[]
  account     Account?     @relation(fields: [accountId], references:
[id])
  accountId   Int?         @unique
  AutoCompleteFood AutoCompleteFood[]
}

model Water {
  id          Int          @id @default(autoincrement())
  dateTime    DateTime
  capacity    Int
  profile     Profile      @relation(fields: [profileId], references: [id])
  profileId   Int
}

model Weight {
  id          Int          @id @default(autoincrement())
  profile     Profile      @relation(fields: [profileId], references: [id])
  profileId   Int
  weight      Float
  date        DateTime
}

model MeasurementObject {
  id          Int          @id @default(autoincrement())
  profile     Profile      @relation(fields: [profileId], references: [id])
  profileId   Int
  name        String
  measurements Measurement[]
}

model Measurement {
  id          Int          @id @default(autoincrement())
  measurementObject MeasurementObject @relation(fields:
[measurementObjectId], references: [id], onDelete: Cascade)
  measurementObjectId Int
  millimeters Int
  date        DateTime
}

```

```

model CaloricityTDEE {
    id            Int            @id @default(autoincrement())
    protein       Float
    carbohydrates Float
    fat           Float
    profile       Profile @relation(fields: [profileId], references: [id])
    profileId     Int            @unique
}

model Food {
    id            Int            @id @default(autoincrement())
    name          String
    dateTime      DateTime
    calories      Float
    protein       Float
    carbohydrates Float
    fat           Float
    profile       Profile @relation(fields: [profileId], references: [id])
    profileId     Int
}

model AutoCompleteFood {
    id            Int            @id @default(autoincrement())
    name          String
    group         String
    calories      Float
    protein       Float
    carbohydrates Float
    fat           Float
    profile       Profile?      @relation(fields: [profileId], references:
[id])
    profileId     Int?
    microNutrients MicroNutrient[]
}

model MicroNutrient {
    id            Int            @id @default(autoincrement())
    name          String
    count         Float
    autoCompleteFood AutoCompleteFood @relation(fields: [AutoCompleteFoodId],
references: [id])
    AutoCompleteFoodId Int
}

model PresetActivity {
    id            Int            @id @default(autoincrement())
    name          String
    createdAt     DateTime
    weekDay       Int
    dateTime      Int
    profile       Profile @relation(fields: [profileId], references: [id])
    profileId     Int
    subCategory   SubCategory @relation(fields: [subCategoryId], references:
[id])
    subCategoryId Int
}

model Activity {
    id            Int            @id @default(autoincrement())
    name          String
    subCategory   SubCategory @relation(fields: [subCategoryId], references:
[id])
    subCategoryId Int
    dateTime      DateTime

```

```

    duration      Float
    profile        Profile      @relation(fields: [profileId], references: [id])
    profileId      Int
  }

model Category {
  id              Int              @id @default(autoincrement())
  name            String
  subCategories   SubCategory[]
}

model SubCategory {
  id              Int              @id @default(autoincrement())
  name            String
  metabolicEquivalent Float
  activity         Activity[]
  presetActivity   PresetActivity[]
  category         Category        @relation(fields: [categoryId], references:
[id])
  categoryId       Int
}

```

A.5 src\app\app-routing.module.ts

```

const routes: Routes = [
  { path: '', redirectTo: '/diary', pathMatch: 'full' },
  {
    path: 'diary',
    loadChildren: () =>
      import('./diary/diary.module').then((m) => m.DiaryModule),
    canMatch: [AuthGuard],
  },
  {
    path: 'report',
    loadChildren: () =>
      import('./report/report.module').then((m) => m.ReportModule),
  },
  {
    path: 'measurements',
    loadChildren: () =>
      import('./measurements/measurements.module').then(
        (m) => m.MeasurementsModule,
      ),
    canMatch: [AuthGuard],
  },
  {
    path: 'account',
    loadChildren: () =>
      import('./account/account.module').then((m) => m.AccountModule),
  },
  { path: '**', redirectTo: '/account/details' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}

```

A.6 src\app\shared\utils\calculate\calculate-time-to-target-weight.func.ts

```

export function calculateTimeToTargetWeight(
  currentWeight: number,
  targetWeight: number,
  burningKcalByDay: number,
): number {
  if (
    (targetWeight - currentWeight < 0 && burningKcalByDay >= 0) ||
    (targetWeight - currentWeight > 0 && burningKcalByDay <= 0)
  )
    return Infinity;

  return ((targetWeight - currentWeight) * KCAL_PER_KILO) / burningKcalByDay;
}

```

A.7 src\app\shared\services\lang.service.ts

```

@Injectable({
  providedIn: 'root',
})
export class LangService {
  public constructor(
    private translateService: TranslateService,
    private modalsService: ModalsService,
  ) {
    this.translateService.addLangs(Object.values(Languages));
    this.translateService.setDefaultLang(Languages.EN);

    const langFromLocal = localStorage.getItem(localStorageKeys.LANG);
    this.translateService.use(langFromLocal ?? Languages.EN);
  }

  public changeLang(lang: Languages): void {
    this.translateService.use(lang);
    localStorage.setItem(localStorageKeys.LANG, lang);
    this.modalsService.addNotification({
      text: NotificationText.LANG_CHANGED,
      type: NotificationType.INFO,
    });
  }

  public getCurrentLang(): Languages {
    return this.translateService.currentLang as Languages;
  }
}

```

A.8 src\app\shared\services\theme.service.ts

```

@Injectable({
  providedIn: 'root',
})
export class ThemeService {
  private theme$: BehaviorSubject<Themes> = new BehaviorSubject<Themes>(
    Themes.DARK,
  );

  public constructor() {
    const themeFromLocal = localStorage.getItem(localStorageKeys.THEME);
    this.theme$.next((themeFromLocal as Themes) ?? Themes.DARK);
  }

  public switchTheme(): void {

```

```

        this.theme$.next(
            this.theme$.value === Themes.DARK ? Themes.LIGHT : Themes.DARK,
        );
        localStorage.setItem(localStorageKeys.THEME, this.theme$.value);
    }

    public getTheme(): Observable<Themes> {
        return this.theme$.asObservable();
    }
}

```

A.9 src\app\shared\services\modals.service.ts

```

@Injectable({
    providedIn: 'root',
})
export class ModalsService {
    private readonly notifications$: BehaviorSubject<INotification[]> =
        new BehaviorSubject<INotification[]>([]);

    public constructor() {}

    public addNotification(notification: INotification): void {
        const notifications = this.notifications$.value;
        const newNotificationWithId: INotification = {
            ...notification,
            id: Date.now(),
        };
        notifications.push(newNotificationWithId);
        this.notifications$.next(notifications);
        setTimeout(() => {
            const notifications = this.notifications$.value;
            this.notifications$.next(
                notifications.filter((i) => i.id !== newNotificationWithId.id),
            );
            this.notifications$.value;
        }, NOTIFICATION_DURATION);
    }

    public closeNotification(id: number): void {
        const notifications = this.notifications$.value;
        this.notifications$.next(notifications.filter((i) => i.id !== id));
        this.notifications$.value;
    }

    public getNotification(): Observable<INotification[]> {
        return this.notifications$.asObservable();
    }
}

```

A.10 src\app\shared\services\foods.service.ts

```

@Injectable({
    providedIn: 'root',
})
export class FoodsService {
    private readonly foodsForCurrentWeek$: BehaviorSubject<IFood[]> =
        new BehaviorSubject<IFood[]>([]);

    public constructor(
        private readonly http: HttpClient,
    ) {}
}

```

```

    private readonly dateService: DateService,
  ) {
    this.getFoods(
      +this.dateService.weekStart,
      +this.dateService.weekEnd,
    ).subscribe((foods) => this.foodsForCurrentWeek$.next(foods));
  }

  public getFoodsForCurrentWeek(): Observable<IFood[]> {
    return this.foodsForCurrentWeek$.asObservable();
  }

  public getFoods(from: number, to: number): Observable<IFood[]> {
    return this.http
      .get<IFood[]>('/api/food', {
        params: { from, to },
      })
      .pipe(
        first(),
        map((foods) =>
          foods.sort(
            (a: IFood, b: IFood) =>
              +new Date(b.dateTime) - +new Date(a.dateTime),
          ),
        ),
      );
  }

  public getFoodsForCurrentDay(): Observable<IFood[]> {
    return this.getFoodsForCurrentWeek().pipe(
      map((foods: IFood[]) => {
        return foods.filter(
          (food) =>
            new Date(food.dateTime) > this.dateService.dayStart &&
            new Date(food.dateTime) < this.dateService.dayEnd,
        );
      }),
    );
  }

  public createFood(body: IFoodDto): void {
    this.http
      .post<IFood>('/api/food', body)
      .pipe(first())
      .subscribe((food) => {
        if (
          new Date(food.dateTime) > this.dateService.weekStart &&
          new Date(food.dateTime) < this.dateService.weekEnd
        )
          this.foodsForCurrentWeek$.next(
            this.foodsForCurrentWeek$.value.concat(food),
          );
      });
  }

  public deleteFood(id: number): void {
    this.http
      .delete<IFood>('/api/food', { body: { id } })
      .pipe(first())
      .subscribe((deletedFood) =>
        this.foodsForCurrentWeek$.next(
          this.foodsForCurrentWeek$.value.filter(
            (food) => food.id !== deletedFood.id,
          ),
        ),
      );
  }

```

```

    ),
  );
}

public searchAutoComplete(search: string): Observable<IAutoCompleteFood[]> {
  return this.http.get<IAutoCompleteFood[]>({
    url: '/api/food/auto-complete/search',
    params: { search },
  });
}
}

```

A.11 src\app\shared\pipes\calculate-measure-by-percent.pipe.ts

```

@Pipe({
  name: 'calculateMeasureByPercent',
})
export class CalculateMeasureByPercentPipe implements PipeTransform {
  transform(
    percent: number,
    profile: IProfile,
    type: MacroNutrients,
  ): IMeasurements {
    const basalMetabolicRate: number = calculateBasalMetabolicRate(
      profile.weight,
      profile.height,
      moment().diff(profile.birthDay, 'years'),
      profile.sex,
    );

    let calories = Math.round(
      ((basalMetabolicRate * +profile.PAL) / 100) * percent,
    );
    calories = calories + calories * profile.targetRate;
    const grams = Math.round(calories / (type === MacroNutrients.FAT ? 9 : 4));
    const gramsToKilograms = roundUpTo(grams / profile.weight);

    return { calories, grams, percent, gramsToKilograms };
  }
}

```

A.12 src\app\shared\interceptors\session-token.interceptor.ts

```

@Injectable()
export class SessionTokenInterceptor implements HttpInterceptor {
  public constructor(private readonly accountService: AccountService) {}

  public intercept(
    request: HttpRequest<unknown>,
    next: HttpHandler,
  ): Observable<HttpEvent<unknown>> {
    return this.accountService.getAccount().pipe(
      switchMap((account) => {
        request = request.clone({
          headers: request.headers.set(
            'session-token',
            account?.sessionToken ?? '',
          ),
        });
        return next.handle(request);
      }),
    );
  }
}

```

```

    );
  }
}

```

A.13 src\app\shared\components\chart\chart.component.ts

```

@Component({
  selector: 'app-chart',
  templateUrl: './chart.component.html',
  styleUrls: ['./chart.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class ChartComponent
  extends DefaultAbstractComponent
  implements OnChanges
{
  public options: BehaviorSubject<EChartsOption | null> =
    new BehaviorSubject<EChartsOption | null>(null);

  @Input() public dataForPeriod!: IDataForPeriod;
  @Input() public type: 'line' | 'bar' = 'bar';
  @Input() public bars: string[] = [];

  public readonly rewrite$: Subject<void> = new Subject<void>();

  public constructor(private readonly themeService: ThemeService) {
    super();
  }

  ngOnChanges(): void {
    this.rewrite$.next();
    const xAxisData: string[] = [];
    const dataArray: number[][] = [];

    for (const _ of this.dataForPeriod.values[0].value) {
      dataArray.push([]);
    }

    for (let data of this.dataForPeriod.values) {
      xAxisData.push(moment(data.day).format('YYYY-MM-DD'));
      for (let i = 0; i < data.value.length; i++) {
        dataArray[i].push(data.value[i]);
      }
    }

    this.themeService
      .getTheme()
      .pipe(takeUntil(merge(this.rewrite$, this.destroy$)))
      .subscribe((theme) =>
        this.options.next({
          legend: {
            textStyle: {
              color: theme === Themes.DARK ? 'white' : 'black',
            },
            data: this.bars,
            align: 'left',
          },
          tooltip: {},
          xAxis: {
            data: xAxisData,
            silent: false,
            splitLine: {
              show: false,
            },
          },
        }));
  }
}

```

```

    },
  },
  yAxis: {},
  series: dataArray.map((array, index) => {
    return {
      name: this.bars[index],
      type: this.type,
      data: array,
      animationDelay: (idx) => idx * 10,
    };
  }),
  animationEasing: 'elasticOut',
  animationDelayUpdate: (idx) => idx * 5,
}),
);
}
}

```

A.14 src\app\shared\components\circle-diagram\circle-diagram.component.ts

```

@Component({
  selector: 'app-circle-diagram',
  templateUrl: './circle-diagram.component.html',
  styleUrls: ['./circle-diagram.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class CircleDiagramComponent implements OnChanges {
  @Input() values: {
    value: number;
    color: string;
    volume?: number;
    shift?: number;
  }[] = [{ value: 0, color: '#f00' }];

  @Input() maxValue: number = 0;
  @Input() isRelative: boolean = false;
  @Input() coefficient: number = 0;

  public volume: number[] = [0];
  public sum: number = 0;
  public sumValue: number = 0;

  public ngOnChanges(): void {
    this.sum = 0;
    this.sumValue = this.values.reduce(
      (prev, current) => prev + current.value,
      0,
    );

    if (this.isRelative) this.maxValue = this.sumValue;

    let shift = 0;

    for (const value of this.values) {
      value.shift = shift;
      value.volume =
        (value.value /
          (this.sumValue > this.maxValue ? this.sumValue : this.maxValue)) *
        360;
      this.sum += value.value;
      shift += value.volume;
    }
  }
}

```

```

    }
  }
}

```

A.15 diagram.component.html

src\app\shared\components\circle-diagram\circle-

```

<div class="chart">
  <div
    class="chart__filling"
    *ngFor="let value of values; index as i"
    [style.--volume]="value.volume + 'deg'"
    [style.--shift]="value.shift + 'deg'"
    [style.--color]="value.color"
  ></div>
  <div class="chart__border chart__border--inside"></div>
  <div class="chart__border chart__border--outside"></div>
  <div class="chart__values">
    <h3 class="chart__value">{{ sum | round }}</h3>
    <h3 class="chart__max-value" *ngIf="!isRelative">{{ maxValue | round
  }}</h3>
  </div>
</div>

```

A.16 src\app\shared\components\default-abstract.component.ts

```

@Component({ template: '' })
export abstract class DefaultAbstractComponent implements OnDestroy {
  protected readonly destroy$: Subject<void> = new Subject<void>();

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }
}

```

A.17 src\app\report\report.component.html

```

<section class="section">
  <form class="range" [formGroup]="timeRangeForm" (ngSubmit)="loadAll()">
    <div class="field field--row">
      <label for="from" class="field__label">
        {{ 'report.form.from' | translate }}
      </label>
      <input
        type="date"
        name="from"
        id="from"
        class="field__input"
        [formControl]="timeRangeForm.controls.from"
      />
    </div>

    <div class="field field--row">
      <label for="to" class="field__label">
        {{ 'report.form.to' | translate }}
      </label>
      <input
        type="date"
        name="to"

```

```

        id="to"
        class="field__input"
        [formControl]="timeRangeForm.controls.to"
      />
    </div>

    <button class="button button--submit" type="submit">
      {{ 'report.form.load' | translate }}
    </button>
  </form>

  <section class="block">
    <h2 class="block__title">
      {{ 'report.header.balance-for-period' | translate }}
    </h2>

    <app-chart
      *ngIf="balanceForPeriod$ | async as balanceForPeriod"
      [dataForPeriod]="balanceForPeriod"
      [bars]="[
        'report.params.consumed' | translate,
        'report.params.goal' | translate,
        'report.params.burned' | translate
      ]"
      type="line"
    />

    <p class="block__description">
      {{ 'report.description.balance-for-period' | translate }}
    </p>
  </section>

  <section class="block">
    <h2 class="block__title">
      {{ 'report.header.foods-for-period' | translate }}
    </h2>

    <app-chart
      *ngIf="foodsForPeriod$ | async as foodsForPeriod"
      [dataForPeriod]="foodsForPeriod"
      [bars]="[
        'report.params.protein' | translate,
        'report.params.carbohydrate' | translate,
        'report.params.fat' | translate
      ]"
    />

    <p class="block__description">
      {{ 'report.description.foods-for-period' | translate }}
    </p>
  </section>

  <section class="block">
    <h2 class="block__title">
      {{ 'report.header.activities-for-period' | translate }}
    </h2>

    <app-chart
      *ngIf="activitiesForPeriod$ | async as activitiesForPeriod"
      [dataForPeriod]="activitiesForPeriod"
      [bars]="[
        'report.params.bmr' | translate,
        'report.params.activity-levels' | translate,
        'report.params.activities' | translate

```

```

    ]"
  />

  <p class="block__description">
    {{ 'report.description.activities-for-period' | translate }}
  </p>
</section>

<section class="block">
  <h2 class="block__title">
    {{ 'report.header.water-for-period' | translate }}
  </h2>

  <app-chart
    *ngIf="waterForPeriod$ | async as waterForPeriod"
    [dataForPeriod]="waterForPeriod"
    [bars]="['report.params.water' | translate]"
  />

  <p class="block__description">
    {{ 'report.description.water-for-period' | translate }}
  </p>
</section>

<section class="block">
  <h2 class="block__title">
    {{ 'report.header.calculate-recomposition' | translate }}
  </h2>

  <app-recomposition-form />
</section>
</section>

```

A.18 src\app\report\report.component.ts

```

@Component({
  selector: 'app-report',
  templateUrl: './report.component.html',
  styleUrls: ['./report.component.scss'],
})
export class ReportComponent extends DefaultAbstractComponent {
  public readonly timeRangeForm: TimeRangeFormGroup =
    this.fb.group<TimeRangeForm>({
      from: this.fb.control<string>(
        moment().startOf('isoWeek').format('YYYY-MM-DD'),
      ),
      to: this.fb.control<string>(
        moment().endOf('isoWeek').format('YYYY-MM-DD'),
      ),
    });

  public readonly balanceForPeriod$: BehaviorSubject<IDataForPeriod | null> =
    new BehaviorSubject<IDataForPeriod | null>(null);

  public readonly foodsForPeriod$: BehaviorSubject<IDataForPeriod | null> =
    new BehaviorSubject<IDataForPeriod | null>(null);

  public readonly activitiesForPeriod$: BehaviorSubject<IDataForPeriod | null>
=
    new BehaviorSubject<IDataForPeriod | null>(null);

  public readonly waterForPeriod$: BehaviorSubject<IDataForPeriod | null> =
    new BehaviorSubject<IDataForPeriod | null>(null);

```

```

public constructor(
    private readonly fb: NonNullableFormBuilder,
    private readonly foodsService: FoodsService,
    private readonly activitiesService: ActivitiesService,
    private readonly profileService: ProfileService,
    private readonly waterService: WaterService,
) {
    super();
    this.loadAll();
}

public loadAll() {
    const from = new Date(this.timeRangeForm.controls.from.value);
    const to = new Date(this.timeRangeForm.controls.to.value);

    this.loadBalanceForPeriod(from, to);
    this.loadFoodsForPeriod(from, to);
    this.loadActivitiesForPeriod(from, to);
    this.loadWaterForPeriod(from, to);
}

private loadBalanceForPeriod(from: Date, to: Date) {
    let profile: IProfile | null = null;
    let activities: IActivity[] | null = null;

    this.profileService
        .getProfile()
        .pipe(
            switchMap((value: IProfile | null) => {
                profile = value;
                return value
                    ? this.activitiesService.getActivities(+from, +to)
                    : EMPTY;
            }),
            switchMap((value: IActivity[] | null) => {
                activities = value;
                return value ? this.foodsService.getFoods(+from, +to) : EMPTY;
            }),
            first(),
        )
        .subscribe((foods: IFood[]) => {
            if (profile && activities) {
                const values: IDataByDay[] = [];

                for (let i = +from; i <= +to; i += ONE_DAY) {
                    const burnedStatistic: IBurnedStatistic =
calculateBurnedStatistic(
                        profile,
                        activities.filter((current) =>
                            moment(i).isSame(current.dateTime, 'day'),
                        ),
                        1,
                    );

                    values.push({
                        day: new Date(i),
                        value: foods
                            .reduce(
                                (prev, current) =>
                                    moment(i).isSame(current.dateTime, 'day')
                                        ? [prev[0] + current.calories]
                                        : prev,
                                [0],

```



```

.subscribe((activities: IActivity[]) => {
  if (profile) {
    const values: IDataByDay[] = [];

    for (let i = +from; i <= +to; i += ONE_DAY) {
      const burnedStatistic: IBurnedStatistic =
calculateBurnedStatistic(
        profile,
        activities.filter((current) =>
          moment(i).isSame(current.dateTime, 'day'),
        ),
        1,
      );

      values.push({
        day: new Date(i),
        value: [
          burnedStatistic.basalMetabolicRate,
          burnedStatistic.metabolicRateByPhysicalActivityLevels,
          burnedStatistic.activitiesRateByMetabolicEquivalents,
        ],
      });
    }

    this.activitiesForPeriod$.next({
      from,
      to,
      values,
    });
  }
});
}

private loadWaterForPeriod(from: Date, to: Date) {
  this.waterService
    .getWater(+from, +to)
    .pipe(first())
    .subscribe((water: IWater[]) => {
      const values: IDataByDay[] = [];

      for (let i = +from; i <= +to; i += ONE_DAY) {
        values.push({
          day: new Date(i),
          value: water.reduce(
            (prev, current) =>
              moment(i).isSame(current.dateTime, 'day')
                ? [prev[0] + current.capacity]
                : prev,
            [0],
          ),
        });
      }

      this.waterForPeriod$.next({
        from,
        to,
        values,
      });
    });
}
}

```

```

<header class="header">
  <div class="logo">
    
    <h1 class="logo__title">NutriLight</h1>
  </div>
  <nav class="navigation">
    <ul class="navigation__list" *ngIf="account | async">
      <a [routerLink]="['/', 'diary']" routerLinkActive="active-link">
        <li class="navigation__item">
          <span class="navigation__icon material-icons">event</span>
          <span class="navigation__title">
            {{ 'header.link.diary' | translate }}
          </span>
        </li>
      </a>
      <a [routerLink]="['/', 'report']" routerLinkActive="active-link">
        <li class="navigation__item">
          <span class="navigation__icon material-icons">description</span>

          <span class="navigation__title">
            {{ 'header.link.report' | translate }}
          </span>
        </li>
      </a>
      <a [routerLink]="['/', 'measurements']" routerLinkActive="active-link">
        <li class="navigation__item">
          <span class="navigation__icon material-icons">speed</span>
          <span class="navigation__title">
            {{ 'header.link.measurements' | translate }}
          </span>
        </li>
      </a>
      <a [routerLink]="['/', 'account']" routerLinkActive="active-link">
        <li class="navigation__item">
          <span class="navigation__icon material-icons">account_circle</span>
          <span class="navigation__title">
            {{ 'header.link.profile' | translate }}
          </span>
        </li>
      </a>
    </ul>
  </nav>
  <div class="actions">
    <div class="theme" (click)="switchTheme()">
      <div class="theme__mod">
        <span class="material-icons">light_mode</span>
      </div>
      <div class="theme__mod theme__mod--active">
        <span class="material-icons">nightlight</span>
      </div>
      <div
        [ngClass]="[
          'theme__selector',
          (theme$ | async) === Themes.DARK
            ? 'theme__selector--right'
            : 'theme__selector--left'
        ]"
      ></div>
    </div>

    <select class="language" name="language" [formControl]="langControl">
      <option

```

```

        class="language__option"
        *ngFor="let lang of Languages | keyvalue"
        [value]="lang.value"
    >
        {{ 'header.language.' + lang.value | translate }}
    </option>
</select>
<button
    class="button button--submit"
    (click)="logout()"
    *ngIf="account | async"
>
    {{ 'header.log-out' | translate }}
</button>
</div>
</header>

```

A.20 src\app\navigation\header\header.component.ts

```

@Component({
    selector: 'app-header',
    templateUrl: './header.component.html',
    styleUrls: ['./header.component.scss'],
    changeDetection: ChangeDetectionStrategy.OnPush,
})
export class HeaderComponent
    extends DefaultAbstractComponent
    implements OnInit
{
    public readonly theme$: Observable<Themes> = this.themeService.getTheme();
    public readonly langControl: FormControl<Languages> =
        this.fb.control<Languages>(this.langService.getCurrentLang());
    public readonly account: Observable<IAccount | null> =
        this.accountService.getAccount();

    public readonly Themes = Themes;
    public readonly Languages = Languages;

    public constructor(
        private readonly fb: NonNullableFormBuilder,
        private readonly langService: LangService,
        private readonly accountService: AccountService,
        private readonly themeService: ThemeService,
    ) {
        super();
    }

    public ngOnInit(): void {
        this.langControl.valueChanges
            .pipe(takeUntil(this.destroy$))
            .subscribe((lang) => this.langService.changeLang(lang));
    }

    public switchTheme(): void {
        this.themeService.switchTheme();
    }

    public logout() {
        this.accountService.logout();
    }
}

```

ДОДАТОК Б
Слайди презентації

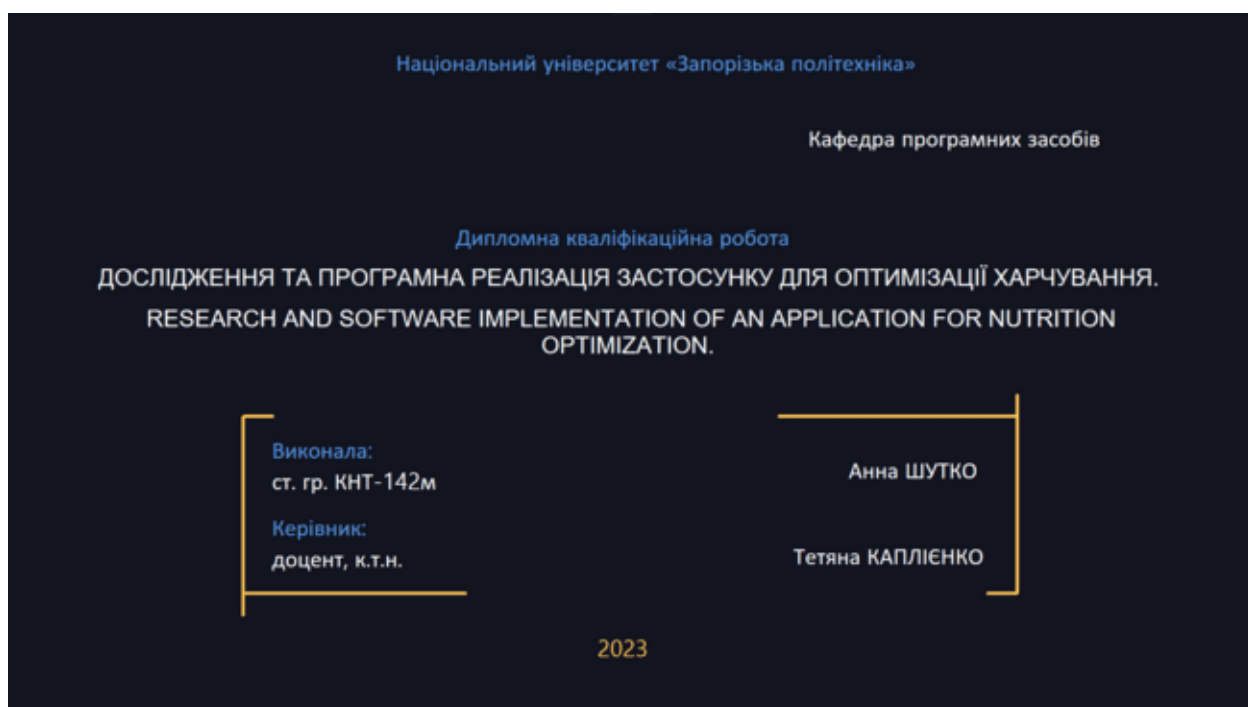


Рисунок Б.1 – Слайд №1

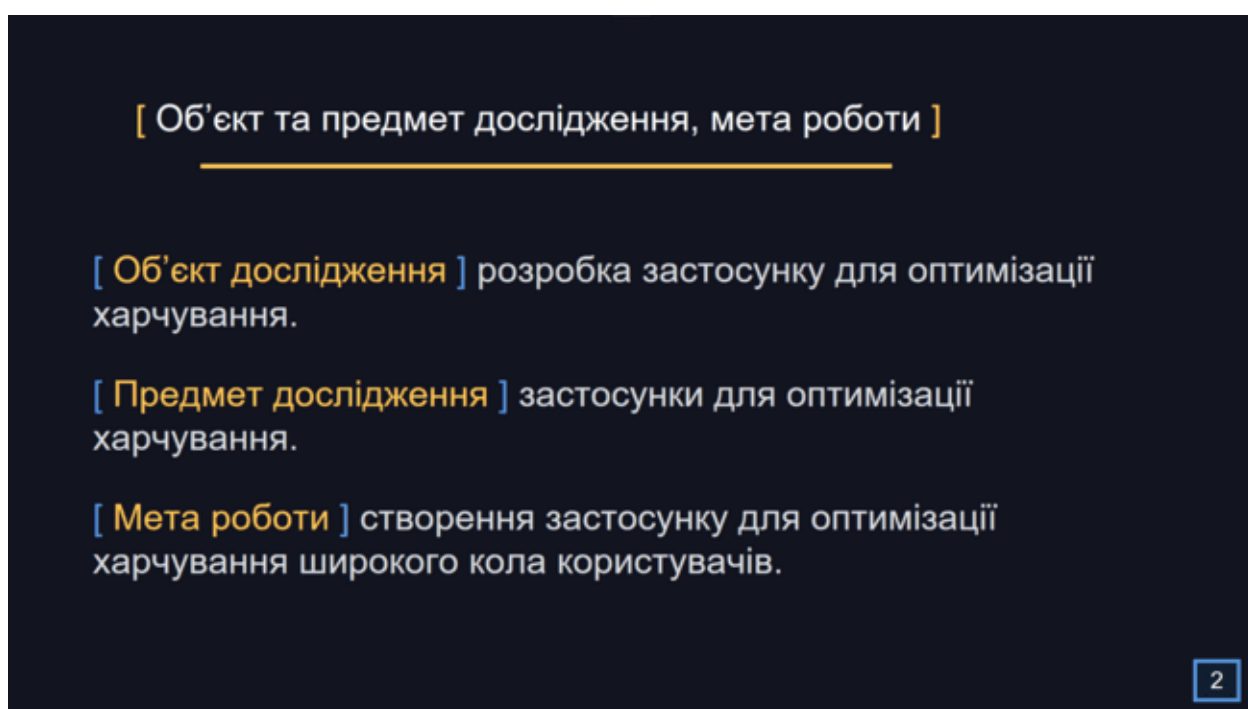


Рисунок Б.2 – Слайд №2

[Аналоги]

Властивості	Назва застосунку				
	FatSecret	Lifesum	MyNetDiary	Cronometer	NutriLight
Зручність	4	5	3	4	5
Дизайн	3	4	3	4	5
Кроссплатформність	Так	Так	Так	Так	Так
Функціональність	5	4	4	5	5
Наявність преміум версії	Так	Так	Так	Так	Ні
Доступність функцій в безкоштовній версії	3	2	3	3	5

3

Рисунок Б.3 – Слайд №3

[Функції програми]



4

Рисунок Б.4 – Слайд №4

[Порівняння фреймворків для фронтенду]

Характеристика	Angular	Vue	React
Тип	Повноцінний фреймворк	Прогресивний фреймворк	Бібліотека
Мова програмування	TypeScript	JavaScript	JavaScript
Механізм зв'язування даних	Двосторонній (two-way)	Односторонній (one-way)	Односторонній (one-way)
Механізм відстеження змін	Зона (zone) та цикл (Lifecycle Hooks)	Гетери (getters) та сетери (setters)	Віртуальний DOM (virtual DOM)
Синтаксис шаблонів	HTML + директиви (directives)	HTML + директиви (directives) + інтерполяція (interpolation)	JSX
Компонентна архітектура	Міцна інтеграція зі структурою компонентів, сприяє модульності	Добра підтримка компонентної архітектури, але менш жорстка	Рекомендується використовувати зовнішні бібліотеки для компонентів
Структура компонентів	Класи + декоратори (decorators) + шаблони (templates) + стилі (styles)	Об'єкти + шаблони (templates) + скрипти (scripts) + стилі (styles)	Функції або класи + JSX
Готові рішення та плагіни	Багато офіційних та сторонніх	Менше офіційних, багато сторонніх	Мало офіційних, багато сторонніх
Крива навчання	Висока	Низька	Середня
Популярність та підтримка спільноти	Висока, Google та інші великі компанії	Середня, незалежний проєкт, багато малих компаній та проєктів	Висока, Facebook та інші великі компанії

5

Рисунок Б.5 – Слайд №5

[Порівняння фреймворків для бекенду]

Фреймворк	Популярність	Документація	Сумісність з Angular	Продуктивність	Масштабованість	Якість коду	Надійність
Spring Boot	Висока	Висока	Низька	Висока	Висока	Висока	Висока
Nest.js	Середня	Висока	Висока	Середня	Висока	Висока	Висока
Express.js	Висока	Середня	Середня	Середня	Середня	Середня	Середня
ASP.NET Core	Висока	Висока	Висока	Висока	Висока	Висока	Висока
SiteCore	Низька	Середня	Низька	Середня	Середня	Середня	Середня
Flask	Висока	Висока	Низька	Низька	Середня	Середня	Середня
Django	Висока	Висока	Середня	Низька	Висока	Висока	Висока
Laravel	Висока	Висока	Низька	Низька	Висока	Висока	Висока
yii2	Середня	Середня	Низька	Низька	Середня	Середня	Середня

6

Рисунок Б.6 – Слайд №6

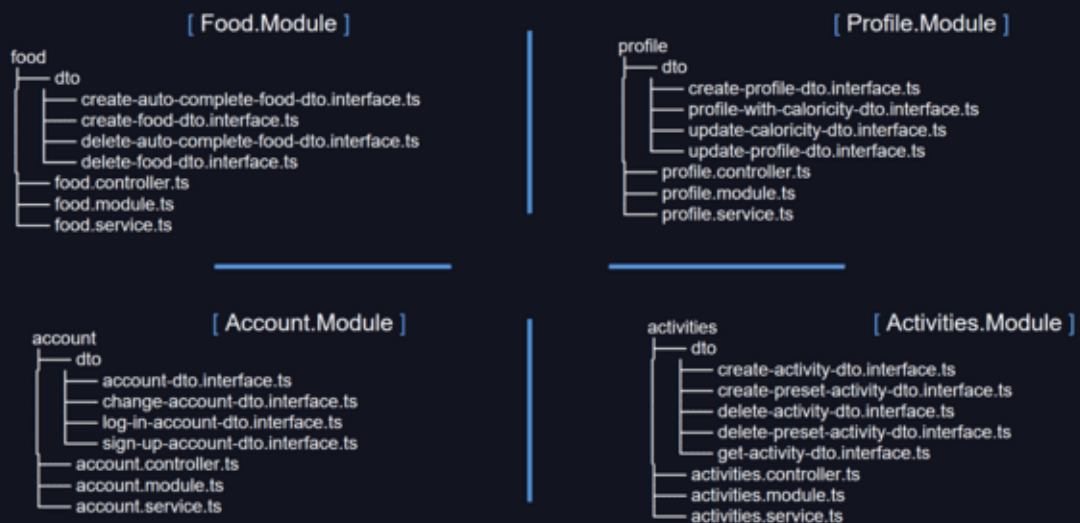
[Порівняння баз даних]

Характеристика	Бази даних				
	Mongo DB	SQLite	PostgreSQL	MySQL	Redis
Тип даних	NoSQL	SQL	SQL	SQL	NoSQL
Модель даних	Документи JSON	Таблиці та рядки	Таблиці та рядки, JSON, XML, геометричні об'єкти тощо.	Таблиці та рядки, JSON	Ключ-значення
Мова запитів	MQL	SQL	SQL, PL/pgSQL, PL/Python, PL/Perl та інші.	SQL, SPL	RCL
Підтримка транзакцій	На рівні документа та колекції.	На рівні бази даних.	На рівні бази даних, таблиць та рядків.	На рівні бази даних та таблиць.	Немає.
Продуктивність	Висока	Середня	Висока	Висока	Надвисока
Масштабованість	Шардинг, реплікація.	Портативність, незалежність від сервера.	Шардинг, реплікація, кластеризація.	Шардинг, реплікація, кешування.	Реплікація, кластеризація.
Ліцензування	SSPL	Public Domain	PostgreSQL License	GPL	BSD License

7

Рисунок Б.7 – Слайд №7

[Файлова архітектура програми | Nest]



8

Рисунок Б.8 – Слайд №8



Рисунок Б.9 – Слайд №9

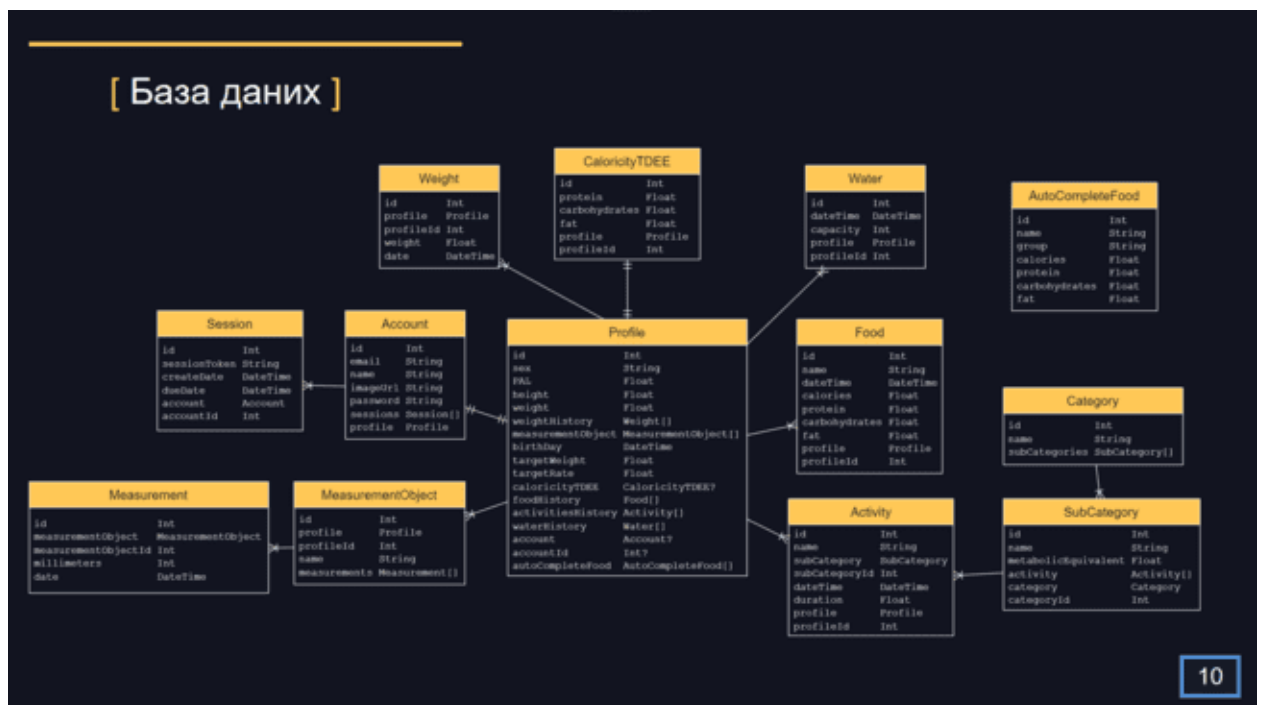


Рисунок Б.10 – Слайд №10

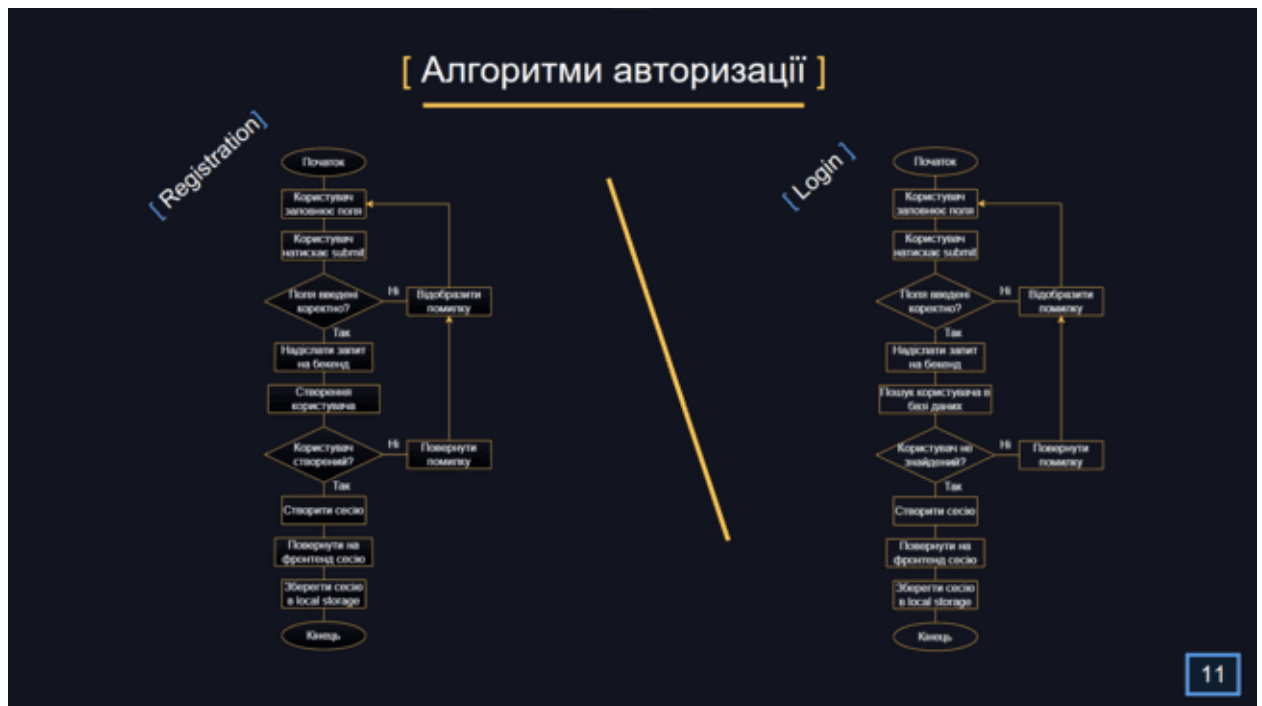


Рисунок Б.11 – Слайд №11



Рисунок Б.12 – Слайд №12

[Оптимізація | Unsubscription]

[take | first]

```

this.http
  .patch<IAccount>('/api/account/change', body)
  .pipe(
    take(1),
    catchError(() => {
      return EMPTY;
    })
  )
  .subscribe((account: IAccount) => {
    this.setAccount(account);
    this.router.navigateByUrl(DEFAULT_PAGE);
    this.modalService.addNotification({
      text: NotificationText.FORM_UPDATE_SUCCESSFUL,
      type: NotificationType.INFO,
    });
  });

this.measurementsService
  .getMeasurements(+this.id, 0, 200)
  .pipe(first())
  .subscribe((measurements: IMeasurement[]) => {
    this.measurements$.next(measurements);
  });

```

[takeUntil]

```

@Component({ template: '' })
export abstract class DefaultAbstractComponent implements OnDestroy {
  protected readonly destroy$: Subject<void> = new Subject<void>();

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }

  ngOnInit(): void {
    this.selectedWeek.valueChanges
      .pipe(takeUntil(this.destroy$))
      .subscribe(() => this.loadActivities());
  }
}

```

[Async Pipe]

```

<div [class]="wrapper" + (theme$ | async) + '-theme">

```

13

Рисунок Б.13 – Слайд №13

[Оптимізація | Dependencies]

707.c442e6b3cc107ed7.js
 node_modules/echarts

index.js + 549
 modules
 (concatenated)

[Analysis]

[Compressed code]

```

"use
strict";(self.webpackChunknutri
_light_client=self.webpackChunk
nutri_light_client||[]).push([
179],[0451:(Ve,ne,E)=>{E.d(ne,{
G:()=>{};var
a=E(8645),N=E(8926);let
j=()=>{var F;class
k{constructor(){this.destroy$=
a.x}ngOnDestroy(){this.destroy$
.next(),this.destroy$.complete(
)}}return(F=k).\u0275fac=functi
on(Z){return
new Z||F}},F.\u0275cmp=N.Xpm({t
ype:F,selectors:[["ng-
component"]],decls:0,vars:0,tem
plate:function(Z,S){},encapsula
tion:2)},k)}),7415:(Ve,ne,E)=
>{E.d(ne,{DV:()=>N,XZ:()=>a,_I:
()=>F,dV:()=>k,jS:()=>j,sZ:()=>
Q,zS:()=>Z});const
a="/account/details",N="/accou
nt/login",j=3e3,Fa3,k=3de5,Q=24*
k,Z=7716,6126:(Ve,ne,E)=>{E.d(
ne,{U:()=>a});var
a=function(N){return
N.THEME="theme",N.ACCOUNT="acco
unt",N.LANG="lang",N}({})},7
274:(Ve,ne,E)=>{E.d(ne,{U:()=>N
,k:()=>a});var
a=function(j){return
j.ERADA="error",j.WARNING="warn
ing",j.INFO="info",j}({}),N=
0

```

14

Рисунок Б.14 – Слайд №14



Рисунок Б.15 – Слайд №15

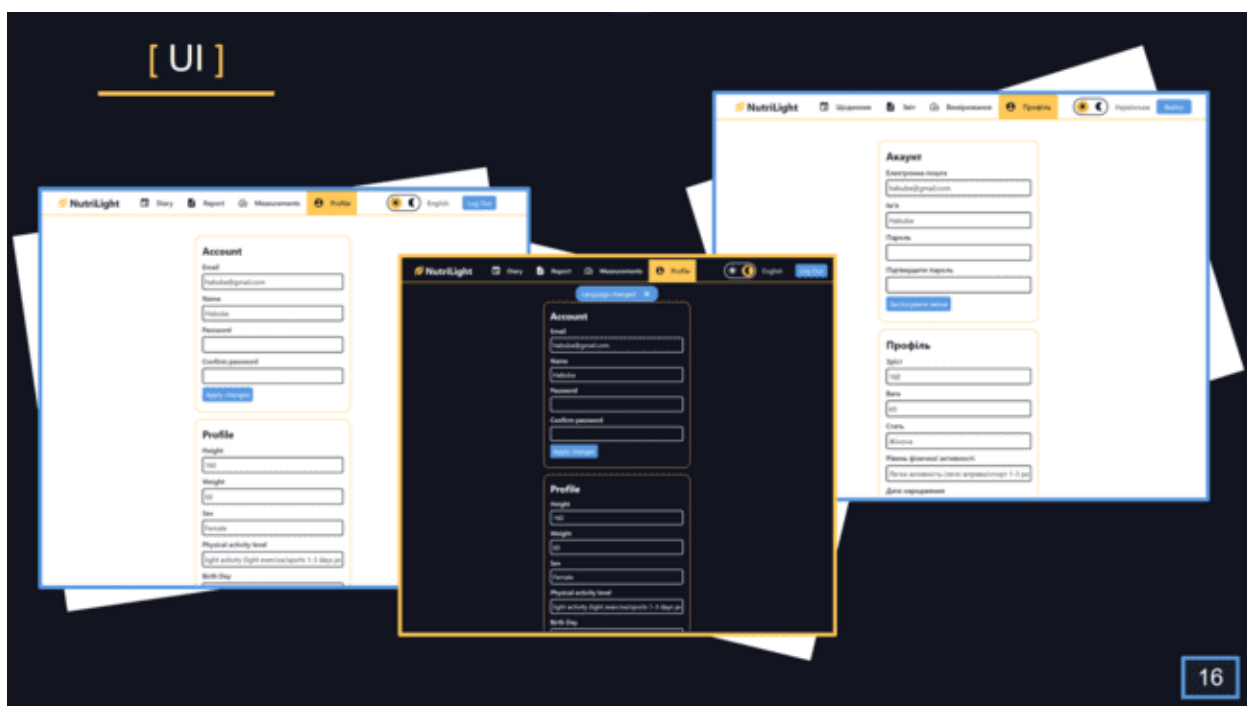


Рисунок Б.16 – Слайд №16



Рисунок Б.17 – Слайд №17



Рисунок Б.18 – Слайд №18

Рисунок Б.20 – Слайд №20

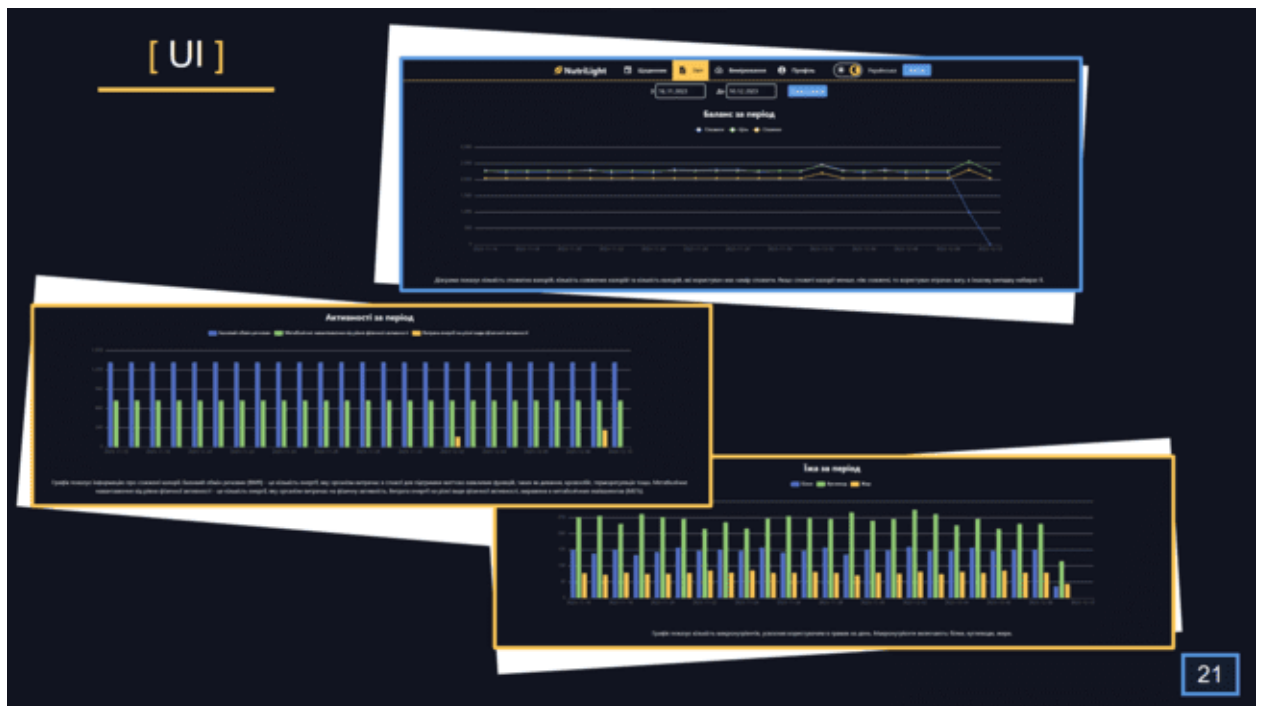


Рисунок Б.21 – Слайд №21

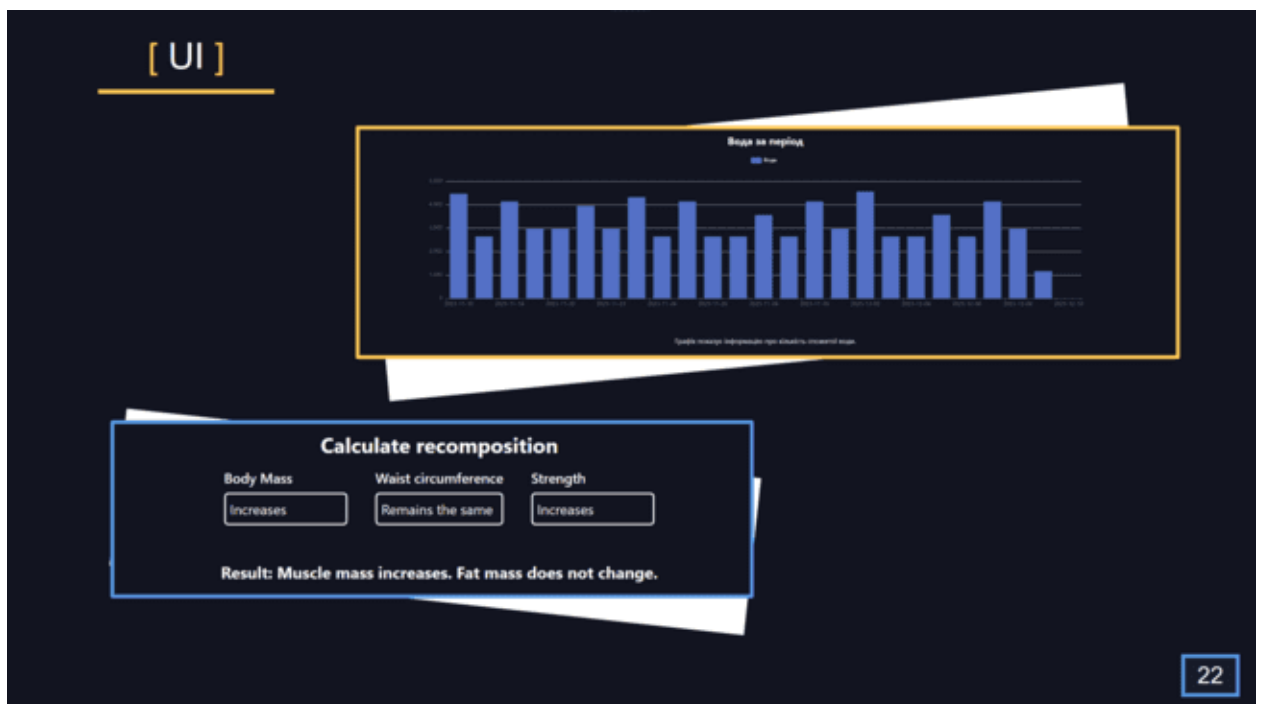


Рисунок Б.22 – Слайд №22

[Висновок]

Створено повнофункціональний застосунок для оптимізації харчування широкого кола користувачів.

Проведено аналіз аналогів програми та визначено функціонал розроблюваного застосунку.

Обрано наступний стек технологій: Angular для фронтенду, Nest.js для бекенду, Prisma ORM для роботи з базою даних та SQLite як базою даних.

Описано файлову архітектуру програми.

Описано алгоритми реєстрації та авторизації.

Проведено оптимізацію застосунку.

Також, за темою створено та опубліковано тези доповіді на науково-практичній конференції «Тиждень науки-2023» у квітні 2023 року.

Рисунок Б.23 – Слайд №23