

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка мобільного застосунку для
індивідуального підбору програм харчування з
урахуванням особистих потреб користувача»**

Виконав студент групи ІПЗ-20-1
ІВЧЕНКО Тетяна Володимирівна

Керівник роботи:
ТОЛСТОЙ Ігор Анатолійович

Рецензент:
МОРОЗОВ Андрій Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ІВЧЕНКО Тетяна Володимирівна**

Керівник роботи: **ТОЛСТОЙ Ігор Анатолійович**

Тема роботи: **«Розробка мобільного застосунку для індивідуального підбору програм харчування з урахуванням особистих потреб користувача»**,
затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: процес автоматизації розрахунку та підбору програм харчування.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Толстой Ігор Анатолійович	27.02.2024р.	20.03.2024р.
2	Толстой Ігор Анатолійович	29.03.2024р.	09.05.2024р.
3	Толстой Ігор Анатолійович	10.05.2024р.	10.06.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з розробки мобільного застосунку для створення індивідуальних програм харчування. Пояснювальна записка до випускної роботи містить 109 сторінок, 40 ілюстрацію та 41 таблицю.

Головною метою даної роботи є проектування та реалізація мобільного застосунку, який дозволяє формувати індивідуальні програми харчування з можливістю налаштовувати особисті потреби та цілі користувача.

В роботі детально розглянуто основні завдання та етапи проектування системи. Проведено аналітичний огляд існуючих рішень. Представлено основні сценарії роботи застосунку, його структуру та архітектуру. Детально описано використані алгоритми та інструменти розробки. Додатково наведено результати проведення тестування системи.

КЛЮЧОВІ СЛОВА: ЗАСТОСУНОК, АРХІТЕКТУРА, ХАРЧУВАННЯ, КАЛОРІЇ, МАКРОНУТРИЄНТИ.

					ІПЗ.КР.Б – 121 – 11 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка мобільного застосунку для індивідуального підбору програм харчування з урахуванням особистих потреб користувача	Літ.	Арк.	Аркушів
Розроб.		Івченко Т. В.						
Керівник		Толстой І. А.					3	109
Рецензент		Морозов А. В.				Житомирська політехніка, група ІПЗ-20-1		
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's qualification work consists of a mobile application for creating personalized nutrition programs. The explanatory note to the final thesis contains 109 pages, 40 illustrations, and 41 tables.

The main goal of this work is the design and implementation of a mobile application that allows users to create personalized nutrition programs with the ability to adjust their personal needs and goals.

The work provides a detailed examination of the main tasks and stages of system design. An analytical review of existing solutions has been conducted. The main use cases of the application, its structure, and architecture are presented. The algorithms and development tools used are described in detail. The results of system testing are presented.

KEYWORDS: APPLICATION, ARCHITECTURE, NUTRITION, CALORIES, MACRONUTRIENTS.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ТЕОРЕТИЧНИХ ТА ФУНКЦІОНАЛЬНИХ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	11
1.1. Опис проекту та постановка задачі	11
1.2. Аналітичний огляд існуючих аналогів	13
1.3. Аналіз вимог та обґрунтування вибору засобів середовища розробки та програмного забезпечення	19
Висновки до першого розділу.....	27
розділ 2. ПРОТОТИПУВАННЯ, ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ	28
2.1. Опис структури та варіантів використання системи.....	28
2.2. Проектування та реалізація роботи алгоритмів застосунку	35
2.3. Проектування бази даних.....	41
2.4. Реалізація серверної частини застосунку підбору індивідуальних програм харчування	50
2.5. Реалізація клієнтської частини застосунку підбору індивідуальних програм харчування	56
Висновки до другого розділу	60
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	62

3.1. Порядок встановлення та налаштування параметрів системи	62
3.2. Опис структури інтерфейсу та роботи додатку	63
3.3. Тестування програмного продукту	78
Висновки до третього розділу	81
ВИСНОВКИ.....	82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	84
ДОДАТКИ.....	89

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс;

BMR – рівень базального метаболізму;

LBM – м'язова маса тіла;

RBAC – контроль доступу на основі ролей;

БД – база даних;

ІМТ – індекс маси тіла;

КБЖВ – калорії, білки, жири, вуглеводи;

СУБД – система управління базами даних;

ПЗ – програмне забезпечення.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

За сучасних умов, де усе відбувається швидко та динамічно, людина повинна не лише бути продуктивною, а й постійно дбати про збереження свого фізичного та психічного благополуччя. Харчування є одним із ключових факторів здоров'я, і навіть найменші зміни у щоденному раціоні можуть помітно вплинути на самопочуття та стан організму в цілому.

Наукові дослідження Всесвітньої організації охорони здоров'я доводять, що 70% стану організму людини залежить від того що вона їсть. Це свідчить про те що нездорова їжа безпосередньо впливає на тривалість життя та сприяє розвитку різних неінфекційних захворювань. За оцінками експертів, у 2017 році, нехтування нормами здорового харчування стало причиною смерті кожної п'ятої людини, що становить близько 11 мільйонів смертей у світі. Неправильна культура харчування призводить не лише до розвитку діабету та порушення обміну речовин, але і є причиною низки інших захворювань, наприклад серцево-судинних, онкологічних тощо. У 2016 році Україна посіла перше місце у Європі за смертністю людей від хвороби серця, викликаних неправильним раціоном. Не менш актуальною проблемою сучасності є загроза ожиріння, адже крім шкоди для здоров'я, надмірна вага також може мати негативний ефект на соціальне життя хворого. [7, 22, 25, 31, 34]

Попри те що Інтернет надає легкий доступ до незмірної кількості інформації про харчові та енергетичні потреби людини, на жаль, він поки не здатен замінити професійного дієтолога чи фітнес-консультанта. У випадку самостійного формування програми споживання важливих макронутрієнтів та калорій – можна зіштовхнутися з різними труднощами. Для прикладу, в мережі завжди існує ризик натрапити на недостовірні дані. Відповідно до дослідження Trattner і Elsweller чимало онлайн-рецептів не мають жодного відношення до здорових дієт і, навпаки, можуть зашкодити людині. [26]

Враховуючи бурхливий розвиток технологій та зростаючий інтерес людей до власного благополуччя, стає очевидною актуальність впровадження інноваційних підходів та технологій для розрахунку щоденного раціону та

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ознайомлення користувачів із основними принципами та ідеями контролю ваги і правильного харчування.

Мета роботи полягає в тому, щоб розробити мобільний застосунок спрямований на індивідуальний підбір та розрахунок програм харчування з урахуванням особистих потреб, вподобань і цілей користувача.

Досягнення визначеної мети обумовлює вирішення наступних завдань:

- визначення основної проблематики та аналіз існуючих рішень;
- дослідження основних принципів побудови програм харчування;
- формування ключових можливостей застосунку, проектування архітектури системи;
- підбір та порівняння підходящих технологій;
- проектування структурних модулів та розрахункових алгоритмів;
- побудова та впровадження інформаційних структур;
- реалізація застосунку для індивідуального розрахунку програм харчування.

Об'єктом дослідження є процес автоматизації розрахунку та підбору програм харчування.

Предметом дослідження є ознайомлення з існуючими технологіями та засобами розробки мобільних застосунків для операційних систем iOS та Android та їх застосування для реалізації додатку індивідуального підбору харчування.

Практична значимість розробки полягає у забезпеченні користувачів персональними рекомендаціями, що враховують унікальні фізіологічні потреби, обмеження тощо. Оскільки даний інструмент дозволяє налаштовувати дієти для конкретних цілей, це має позитивний вплив на профілактику хронічних захворювань та формування здорових харчових звичок. Додатково застосунок економить час користувачів, надаючи готові рецепти і пропонуючи страви відповідно до встановленої програми харчування.

Методи дослідження. У роботі використовувалися емпіричні та теоретичні методи дослідження. Метод порівняльного аналізу застосовувався для оцінки існуючих аналогів та потенційних розробок. Експериментальний метод використовувався для тестування та визначення ефективності

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

програмного продукту. Опис використовувався для систематичного викладу та пояснення інформації щодо результатів роботи. Метод абстрагування використовувався для виділення основних аспектів проблеми.

Технічні підходи та джерела інформації. У роботі було використано функціональні та кросплатформенні методи розробки, методи системного аналізу, а також принципи створення нативного дизайну. Інформаційна складова дослідження базується на актуальних наукових публікаціях з дієтології, технічній документації і матеріалах зі створення мобільних застосунків.

Випускна робота складається з вступу, трьох розділів, висновків, списку використаних джерел із 46 найменувань (5 ст.), 5 додатків (20 ст.), 41 таблицю та 40 ілюстрацію по тексту, з них 5 оформлені на окремій сторінці. Повний обсяг роботи становить 109 сторінок, з них 71 сторінки основного тексту.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ТЕОРЕТИЧНИХ ТА ФУНКЦІОНАЛЬНИХ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Опис проекту та постановка задачі

Протягом останнього десятиліття цифровізація торкнулася різних сфер життя. Сучасні технології перепрофілювали та поліпшили спосіб, яким ми живемо і взаємодіємо один з одним сьогодні. Програмні продукти дозволяють планувати тренування, зустрічі, навчатися та працювати на ходу. Вони поступово стають незамінними помічниками, і дозволяють заощаджувати час.

Саме тому мобільні застосунки визначають новий стандарт зручності, де кожен може отримати доступ до необхідних функцій та даних за декількома торканнями екрану. Середній час використання мобільних застосунків у 2020 році сягав 3-5 годин, що дорівнює 88% часу проведеного людиною в Інтернеті, і в останні роки цей показник лише збільшився. Це пов'язано з легкою доступністю та мобільністю смартфонів і планшетів. Ще однією важливою перевагою є інтеграція з функціоналом пристроїв. Мобільні додатки можуть використовувати камери, геолокацію, датчики руху та інші можливості техніки для надання додаткових сервісів та покращення користувацького досвіду. [1]

Враховуючи широку розповсюдженість мобільних пристроїв та важливість вирішення питання здорового харчування, було прийнято рішення що у даній роботі буде розроблено спеціалізований застосунок, який дозволить користувачам завжди мати під рукою інструменти для підтримки здорового раціону, та перегляду сформованих індивідуальних програм харчування. Мобільний додаток є найбільш зручним рішенням для цього, оскільки процес приготування та споживання їжі часто відбувається в умовах, де доступ до комп'ютера обмежений або неможливий.

Для досягнення поставлених цілей необхідно виконати наступне:

- обрати і створити архітектуру, яка буде стійкою і надійною, а також забезпечить гнучкість системи для майбутнього розширення;

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

- забезпечити надійний зв'язок між серверною і клієнтською частинами програмного комплексу для ефективної та надійної передачі даних;
- розробити модуль адміністрування доступу користувачів до системи та даних;
- забезпечити можливість зберігання та редагування ключових даних про користувача, таких як зріст, вага, рівень активності, а також індивідуальні потреби, наприклад, улюблені продукти, алергії, розміри порцій тощо;
- забезпечити розмежування розрахунків відповідно до вимог клієнта;
- реалізувати функціонал для ведення журналу харчування, включаючи можливість вибору страв;
- забезпечити можливість щоденного оновлення ваги користувача та перегляду історії планів;
- представити елементи обчислення та ведення статистики для відслідковування прогресу досягнення поставленої мети користувача.

Успішна реалізація поставленого завдання передбачає виконання наступних етапів роботи:

1. Проведення аналізу аналогічних систем.
 - 1.1. Оцінка аналогічних систем які вже існують на ринку або в інших сферах.
 - 1.2. Аналіз існуючих функціональних можливостей, особливостей і обмежень.
 - 1.3. Визначення і обґрунтування інструментів та технологій для розробки продукту, встановлення вимог до апаратної бази.
2. Визначення архітектури для розробки системи та проектування бази даних.
 - 2.1. Вибір архітектури системи з урахуванням вимог проекту та можливостей розробки.
 - 2.2. Визначення необхідних інструментів розробки.
 - 2.3. Проектування структури БД.
 - 2.4. Планування, розробка та оптимізація серверної частини застосунку.
 - 2.5. Планування, розробка та оптимізація клієнтської частини застосунку.
3. Проведення комплексного тестування системи та її оптимізація.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

3.1. Розробка тестових сценаріїв для перевірки різних аспектів роботи системи.

3.2. Проведення тестування системи на різних етапах розробки для виявлення та виправлення помилок.

3.3. Оптимізація системи після тестування для забезпечення її стабільності та надійності.

Після завершення виконання усіх етапів визначеного плану буде представлено готовий мобільний застосунок, що дозволить підбирати індивідуальні програми харчування з урахуванням особистих потреб користувачів.

1.2. Аналітичний огляд існуючих аналогів

Ринок мобільних застосунків пропонує різноманітний вибір засобів для покращення харчування і ведення здорового способу життя. Кожен з них має власні корисні функції, починаючи з простого прорахунку калорій, закінчуючи послугами створення персоналізованих дієт, прогнозуванням ваги тощо. Під час дослідження існуючих аналогів, було виокремлено наступні програмні продукти: «PlateJoy», «MyFitnessPal», «Lifesum» та «Dine4Fit: таблиця калорійності».

«**PlateJoy**» – це застосунок, що підбирає індивідуальні плани харчування, пропонує рецепти та формує списки продуктів, що значно заощаджує час і допомагає харчуватися краще.

Збір даних у «PlateJoy», відбувається у вигляді опитування, що містить запитання про такі речі, як харчові звички, захворювання, розмір сім'ї, часові обмеження прийомів їжі тощо. Алгоритми системи враховують введену інформацію та надають персоналізовані рекомендації з щоденного раціону, аналізуючи вибір інгредієнтів, яких користувач хотів би уникати, страви, яким він віддає перевагу, і бажані розміри порцій. Застосунок має мінімалістичний і легкий для сприйняття інтерфейс, що забезпечує гарний користувацький досвід (Рис. 1.1).

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

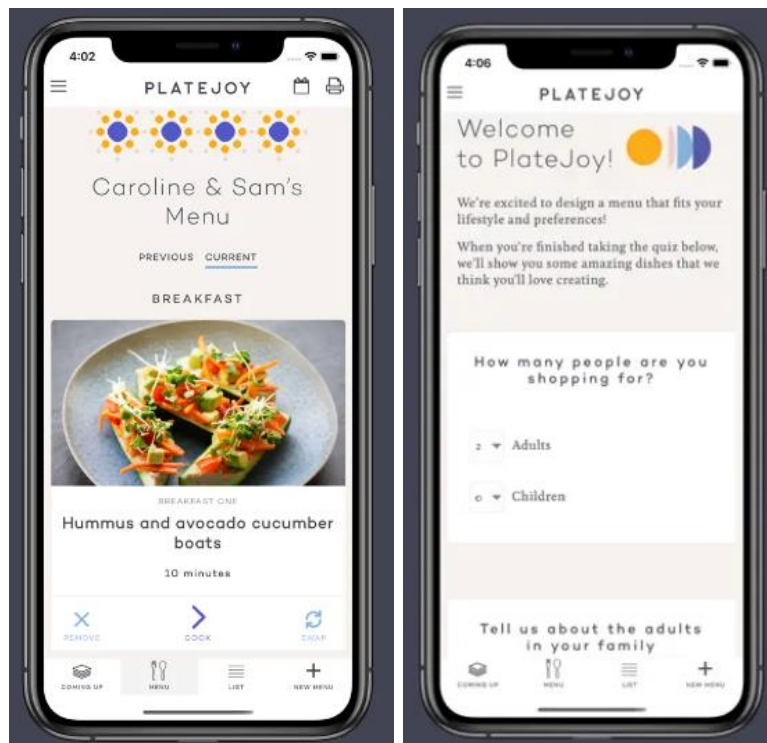


Рисунок 1.1. – Інтерфейс застосунку «PlateJoy»

Перевагою застосунку є те, що він автоматично пропонує список страв на певний період, і даний список можна з легкістю змінювати. Недоліком застосунку є відсутність української локалізації, висока вартість підписки, яка є обов'язковою для використання функціоналу у повній мірі, а також відсутність ведення обліку калорій.

«**MyFitnessPal**» – інноваційний додаток, розроблений для того, щоб досягати поставлених фітнес-цілей. Він є одним з найбільш популярних додатків для відстеження харчування та фізичних навантажень, і ним користуються понад 200 мільйонів користувачів по всьому світу.

Рекомендації з харчування у «MyFitnessPal» базуються на цілях користувача, таких як втрата ваги, набір м'язової маси тощо. Додаток використовує простий алгоритм, який обчислює потреби в калоріях з урахуванням рівня активності. Система також визначає КБЖУ та навіть враховує клітковину. Щодня користувачам потрібно реєструвати споживані страви та напої, враховуючи добову норму калорій, встановлену програмою. Перевагою є те, що для користувача створюється персональна схема досягнення мети, що формується з даних введених у щоденнику.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

Мінімалістичний інтерфейс дозволяє швидко відстежувати всі прийоми їжі та слідкувати за прогресом (Рис. 1.2).

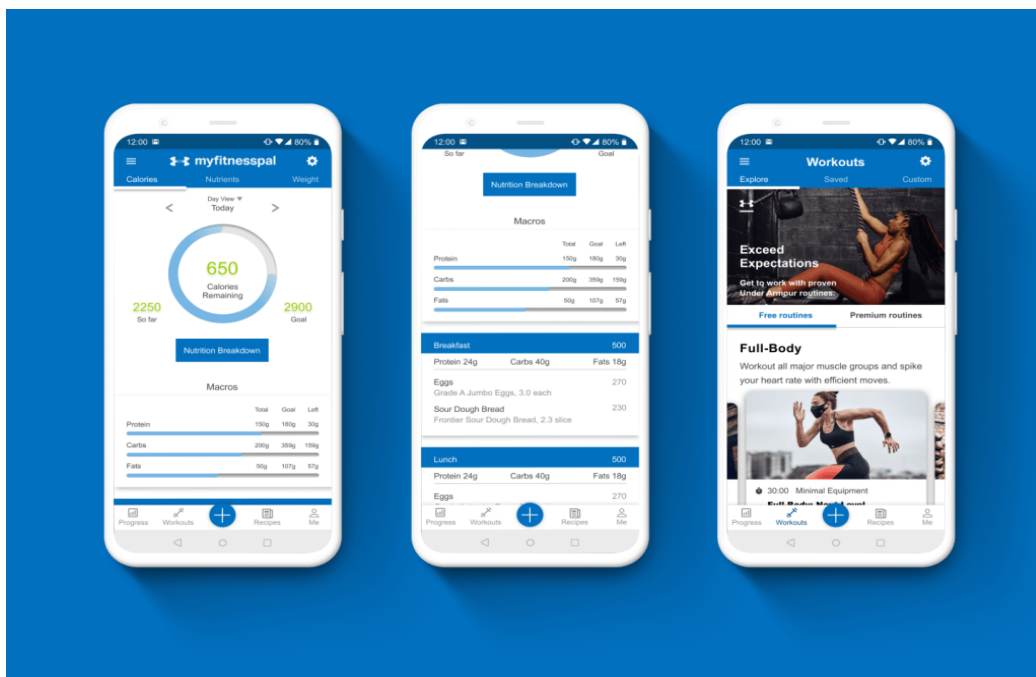


Рисунок 1.2. – Інтерфейс застосунку «MyFitnessPal»

Для наочного прикладу досягнення цілі можна робити фото. Дані про активність, можна отримувати з інших додатків для здоров'я: «Garmin», «Samsung Health», «MapMyRun» тощо. Додаток дозволяє ділитись своїм результатом з іншими користувачами за допомогою відповідних опцій.

Помітним недоліком застосунку є використання загальних формул для розрахунку кількості калорій та макроелементів, що не враховує індивідуальні потреби користувача. Також велика база даних та можливість додавання нових продуктів користувачами часто призводять до неточної інформації про склад продуктів, калорійність тощо.

«**Lifesum**» – проста, мінімалістична, натомість дуже зручна програма з повним відображенням калорійності і макронутрієнтів для кожного прийому їжі. Основна функція застосунку – це ведення журналу споживання їжі. З урахуванням фізичних показників, система сама визначає норму калорій, а також постійно рекомендує нові рецепти для різноманітності. Крім цього, програма аналізує харчові звички користувача та надає рекомендації щодо покращення раціону.

На рисунку 1.3 зображено інтерфейс застосунку «Lifesum». Дизайн є доволі простим і легким для сприйняття. Інтуїтивна структура спрощує навігацію та використання програми. Кольорова палітра і розташування елементів відповідають основним принципам UI та UX дизайну.

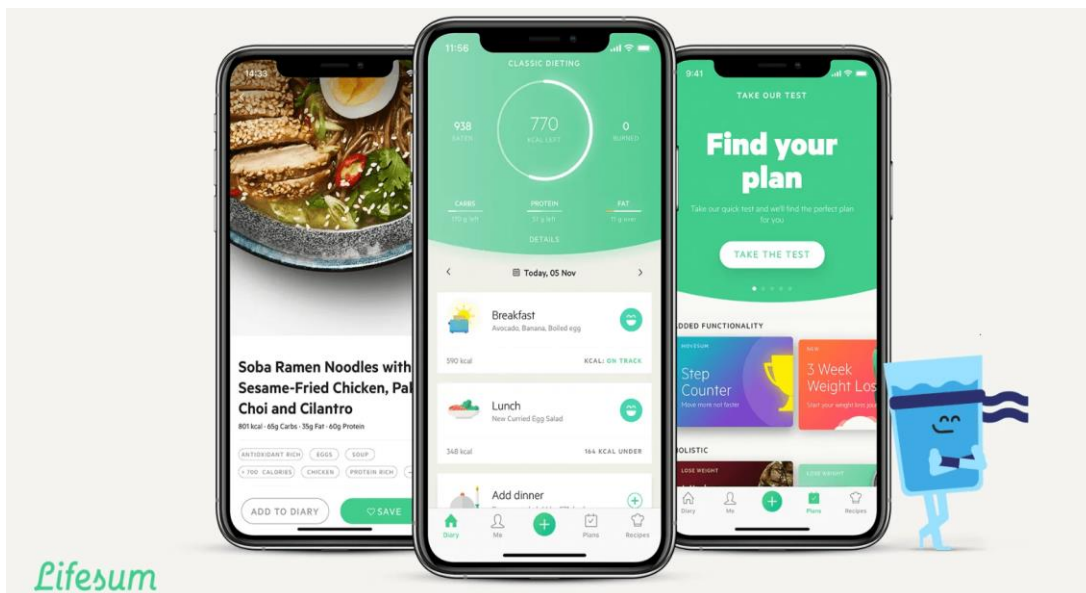


Рисунок 1.3. – Інтерфейс застосунку «Lifesum»

База даних продуктів – доволі різноманітна, проте завжди можна додати їжу самостійно. Користувачу доступна можливість самостійного вибору плану харчування, який буде побудовано на певний період. Це може бути кето-дієта, чисте харчування тощо, проте дана функція доступна лише у преміальній підписці. Застосунок можна синхронізувати з іншими додатками для здоров'я, як от: «Samsung Health», «Apple Health», «Google Fit», «Android Health», «Runkeeper», «Fitbit», «Jawbone», «Endomondo».

Перевагою «Lifesum» є широкий спектр функціоналу, а саме: ведення щоденника харчування, лічильник калорій, трекер вживання води, макронутрієнтів, схеми інтервального голодування тощо. Недоліком, який відзначає значна частка користувачів є нелогічний розрахунок кількості білків, жирів та вугливодів, а також необхідність постійного використання застосунку, для отримання кращих результатів.

«Dine4Fit: таблиця калорійності» – це калькулятор калорій та щоденник правильного харчування з рецептами на кожен день. Даний застосунок схожий на своїх попередників. Він пропонує максимально простий функціонал для

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

розрахунку спожитих калорій, планування раціону, досягнення оптимальної ваги, нарощування м'язів тощо. Проте, це один з небагатьох додатків, що представлений українською мовою. Інтерфейс застосунку «Dine4Fit: таблиця калорійності» представлено на рисунку 1.4.

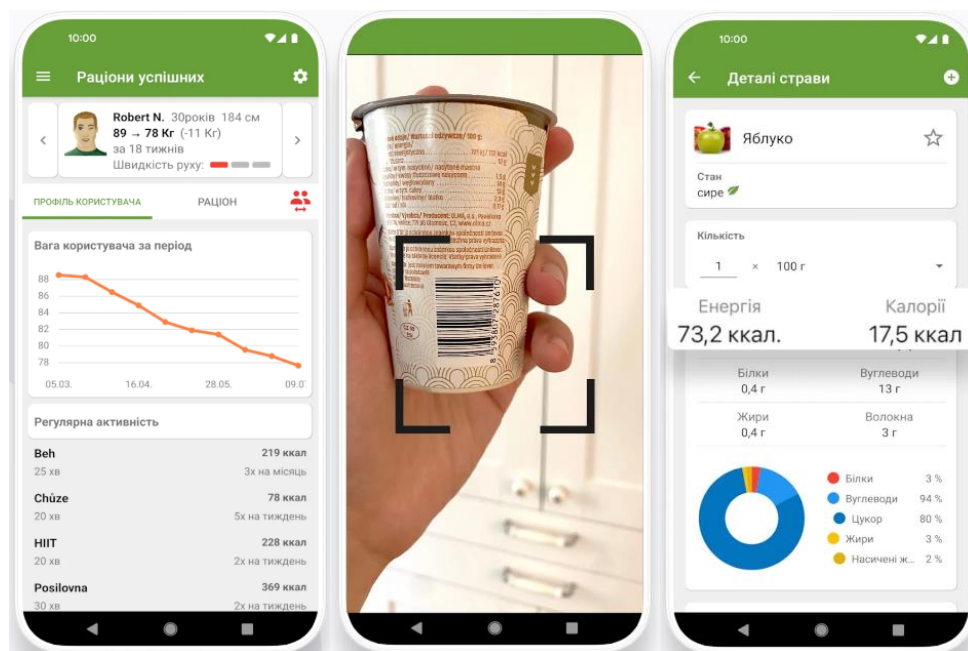


Рисунок 1.4. – Інтерфейс застосунку «Dine4Fit: таблиця калорійності»

Користувачу пропонується велика кількість рецептів, які допоможуть харчуватися не тільки якісно, а й з дотриманням всіх принципів здорового харчування. Калорійність продуктів можна дізнатися через пошук або сканувавши штрих-код. У такий же спосіб можна додавати продукти в щоденний раціон. Застосунок пропонує безкоштовну статистику щодо кількості спожитих калорій, води та темпів зміни ваги, а також щоденник активності. Ці всі корисні опції дозволяють відстежувати динаміку витрат зайвих кілограмів та систематизувати свою активність в поєднанні з прийомами їжі. На жаль, багато інших функцій доступні лише у преміальній версії.

До переваг застосунку можна віднести його доступність, оскільки Dine4Fit з легкістю можна запустити на ПК, телефоні, планшетах та розумних годинниках. Недоліком є те що для використання великої більшої частини функціоналу потрібно мати преміальну підписку. Отже, поряд із позитивними сторонами проаналізованих застосунків було виявлено ряд спільних недоліків, а саме:

- хоча кожен програмний продукт пропонує обширну базу даних продуктів та страв, час від часу можна зустріти неповні або недостовірні дані про калорійність, енергетичну цінність або поживні речовини;
- переважно застосунки зосереджені або на індивідуальних планах харчування, або на звичайному відстеженні калорій. Доволі складно знайти систему, яка виконує ці дві задачі одночасно;
- більшість застосунків не враховують заборону продуктів, алергії у людей тощо, що стає суттєвим викликом для тих, хто дотримується певної лікувальної дієти або має харчові обмеження.

З метою забезпечення наочності та кращого порівняння харчових застосунків було розроблено таблицю 1.1, яка враховує різні аспекти їхньої функціональності та можливостей. Таблиця допомагає зробити об'єктивний аналіз представлених аналогів і власної розробки.

Таблиця 1.1

Порівняльна таблиця аналогічних рішень

Властивості	PlateJoy	MyFitnessPal	Lifesum	Dine4Fit	Власна розробка
Зручність інтерфейсу	+	+	+	+	+
Доступність	+	+	—	—	+
Персоналізація	+	+	+	—	+
Вибір кількості прийомів їжі	+	-	-	-	+
Відслідковування прогресу	—	+	+	+	+
Українська мова	—	—	—	+	+
Статистика і аналітика	—	+	+	+	+
Алергії та обмеження	+	—	+	—	+
Історія програм/планів харчування	—	—	—	—	+
Щоденне оновлення даних	—	+	—	+	+

Безсумнівно, кожен застосунок має набагато більше функціоналу, який може стати перевагою в тій чи іншій ситуації. Попри те, що обмеженість ресурсів

може суттєво вплинути на якість та можливості розробки, для власного проекту було вдало виокремлено і поєднано основні функції кожного з аналогів, необхідні для задоволення ключових потреб користувачів та наближення до поставленої цілі.

1.3. Аналіз вимог та обґрунтування вибору засобів середовища розробки та програмного забезпечення

Розробка мобільних додатків – це комплекс дій, спрямованих на створення програмного забезпечення для невеликих портативних бездротових комп’ютерних пристроїв, наприклад смартфонів, планшетів тощо.

Першим кроком у розробці мобільного застосунку – є вибір платформи. Це може бути конкретна операційна система мобільного, веб-інтерфейс чи інші технічні рішення. При виборі необхідно врахувати такі фактори, як цільова аудиторія, особливості додатку, ресурси для розробки тощо.

Статистика показує, що Android є провідною мобільною операційною системою в усьому світі, друге місце займає iOS від Apple (Рис.1.5).

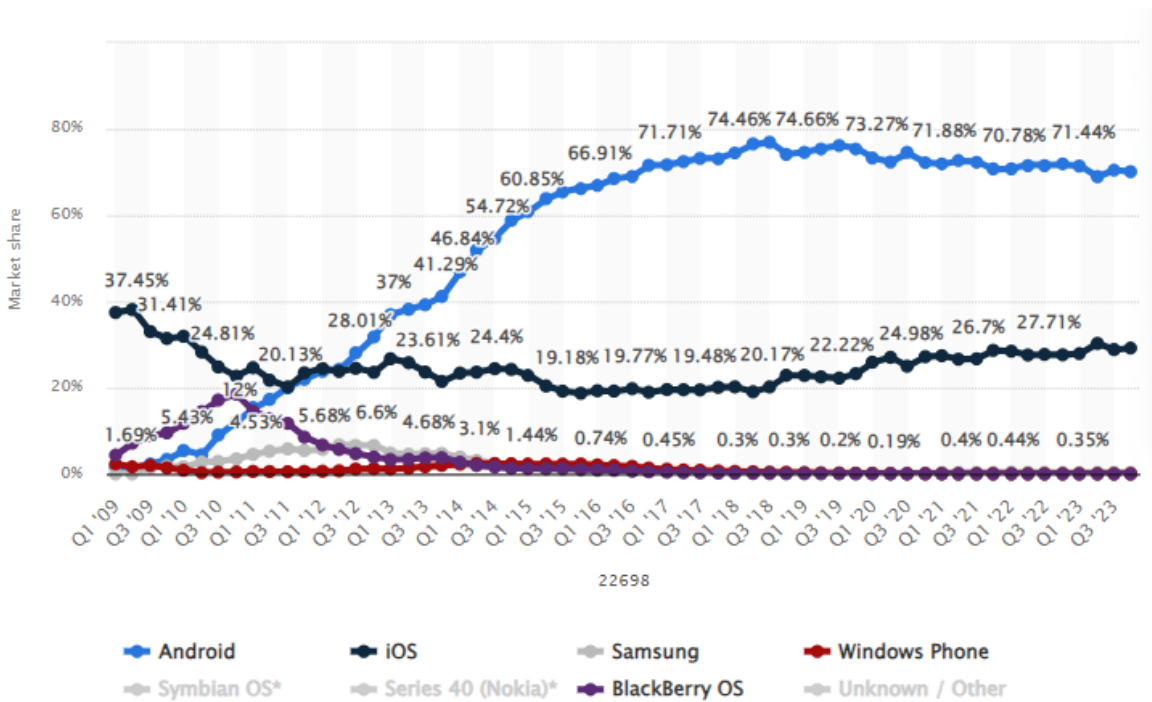


Рисунок 1.5. – Частка ринку мобільних операційних систем у всьому світі з 2009 по 2024 рік [37]

Оскільки стоїть завдання охопити велику аудиторію, є сенс розробляти застосунок, який буде мультиплатформеним, тобто безперебійно працюватиме на кількох операційних системах. Цього можна досягти за допомогою наступних шляхів:

1. Використовуючи інструменти та мови програмування, розроблені під конкретну платформу, цим самим розробляючи програмні продукти для кожної операційної системи окремо. Однак, такий підхід призводить до необхідності розробки і підтримки окремих версій програми для кожної платформи окремо, що може збільшувати трудомісткість та витрати на розробку. [3, 9, 15, 17]
2. Розробляючи веб-додаток, що поєднує у собі функції мобільної програми із рішеннями та інструментами, що використовуються у веб-розробці (PWAs). З одного боку це доволі зручно, адже такий застосунок не буде вимагати окремої збірки та може бути опублікованим онлайн. Такі програми можна легко відкрити через браузер на комп'ютері, смартфоні або іншому пристрої. Проте веб-додаток не може повністю використати потенціал пристрою, що може призвести до посереднього користувацького досвіду та обмеження взаємодії з користувачем. [3, 17]
3. Створюючи гібридний додаток, який поєднує нативні та веб-технології. Для цього потрібно додати код, написаний мовою веб-розробки, як-от HTML, CSS або JavaScript, у рідну програму. Це можна зробити за допомогою фреймворків, таких як Ionic Capacitor і Apache Cordova, використовуючи додаткові плагіни. Одним з основних недоліків даного методу є обмежені можливості доступу до нативного API та функціональності платформи. Крім того, гібридні додатки можуть стикатися з проблемами щодо швидкодії та оптимізації, особливо при обробці великої кількості даних. [3, 14, 17]
4. Використовуючи інструменти та мови програмування, розроблені для однакової роботи на різних мобільних платформах. Даний варіант має суттєві переваги, а саме: використання єдиної кодової бази, економія часу, ефективне управління ресурсами, швидкий вихід на ринок і налаштування. З недоліків

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

можна виокремити нестабільність, обмежену адаптивність і гнучкість у взаємодії з різними платформами. [3, 9, 14, 17]

На практиці ні одне рішення не є найкращим, кожне має свої переваги та недоліки. Для успішної розробки необхідно вивчати та враховувати особливості кожної платформи, а також ретельно вибирати інструменти для розробки.

Доволі важливо забезпечити мінімум проблем при адаптації проекту на кожную з платформ. Ця можливість реалізується за допомогою використання високорівневих мов програмування, інтегрованих середовищ розробки та платформ, що забезпечують універсальну компіляцію, компонування та виконання коду на різних операційних системах.

Сучасні високорівневі мови програмування, такі як C, C++ та Java, є кросплатформенними на рівні компіляції, тобто для цих мов є компілятори під різні платформи. Це дозволяє змінювати лише особливі системо-залежні частини, без переписування основного двигуна програми. Проте однієї лише мови програмування є недостатньо для такої розробки. Важливим елементом створення мультиплатформених застосунків є фреймворки. [18]

Фреймворк – це набір інструментів та програмних рішень, які спрощують створення і підтримку складних проєктів. Він, як правило, включає лише базові програмні модулі, інші компоненти реалізуються на їх основі. Це забезпечує високу швидкість розробки, гарантує велику продуктивність і надійність рішень. Фреймворків для мультиплатформених програм є чимало, серед них найпопулярніші це – React Native, Flutter, Xamarin, Ionic. [19]

Flutter – найновіший крок у розробці додатків, інноваційний фреймворк від компанії Google з відкритим вихідним кодом для створення мобільних і веб-програм з єдиною кодовою базою. Основою для розробки застосунків є об'єктно-орієнтована мова програмування Dart. Процес розробки з використанням цього інструменту є доволі зручним, оскільки він також підтримує гаряче перезавантаження, що значно прискорює роботу. Для створення унікальних користувацьких інтерфейсів у Flutter використовуються віджети, а великий набір

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

бібліотек та інструментів, роблять кастомізацію цих віджетів максимально гнучкою, чим забезпечують високу продуктивність продукту. [9, 9]

Xamarin – ще один інструмент розробки, що належить Microsoft. Даний фреймворк дозволяє розробникам кодувати нативні, кросплатформені додатки для iOS, Android та Windows Phone на мові C#. Це дозволяє написати єдину кодову базу, яку можна використовувати для створення додатків для обох платформ, що може заощадити багато часу і зусиль, оскільки не потрібно вивчати дві різні мови програмування або середовища розробки. Інтерфейс користувача може бути розроблений лише за допомогою коду на C# або XAML кодами (Extensible Application Markup Language). Xamarin є потужною платформою, яка надає абстракційний рівень для забезпечення ефективної взаємодії між загальним кодом та кодом, специфічним для кожної мобільної платформи. Вона функціонує в керованому середовищі, яке включає в себе важливі можливості, такі як автоматичне виділення пам'яті та сміттєвий збір. [9, 46]

Ionic – це фреймворк, що дозволяє створювати кросплатформені застосунки. Перевагою Ionic є можливість створювати гібридні мобільні програми, з яскравим дизайном. Компоненти побудовані за допомогою вебстандартів з використанням HTML, CSS та JavaScript. Кожен Ionic елемент адаптує свій зовнішній вигляд до платформи, на якій працює програма. Програма Ionic, завантажена з Apple App Store, отримує тему iOS, а з Play Store Android – отримує тему Material Design. Для даного фреймворку не потрібен емулятор та сторонні рішення, щоб перевірити якість коду та наявність помилок. Цикл тестування програмних продуктів досить короткий. Ionic поставляється разом з новим середовищем виконання гібридних додатків з відкритим вихідним кодом Capacitor, який дозволяє запускати сучасні веб-застосунки для iOS, Android (з використанням технології Progressive Web App), надаючи потужний та простий у використанні інтерфейс для доступу до нативних пакетів SDK та API на кожній платформі. [3, 9, 32]

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

React Native – фреймворк з відкритим вихідним кодом, призначений для розробки кросплатформних мобільних застосунків, використовуючи лише JavaScript. React Native надає можливість створювати інтерфейс користувача декларативно, перетворюючи стандартні елементи VDOM на нативні візуальні елементи для кожної конкретної мобільної платформи. Запуск програми відбувається за допомогою проміжних бібліотек, тому створені програми однаково добре працюють на базі iOS та Android. Внесені зміни в компонент викликають оновлення одразу для двох платформ. Це відбувається з використанням функції гарячого перезавантаження, яка дозволяє оновлювати працюючий додаток за лічені секунди, коли у вихідний код вносяться зміни. Загалом, використання React Native дозволяє зменшити витрати на розробку майже на половину, не погіршуючи при цьому якість та продуктивність продукту. [9, 24, 29]

Вибір відповідної технології є досить непростим завданням. Щоб показати переваги та недоліки кожної технології, було проведено порівняння (Табл. 1.2).

Таблиця 1.2

Порівняльна таблиця технологій

Критерії	React Native	Flutter	Xamarin	Ionic
Підтримка спільнотою	5	4	4	4
Продуктивність	4	4	4	4
Гаряче перезавантаження	5	5	1	1
Підтримувані платформи	5	5	4	4
Популярність	5	5	3	3
Відкритість ресурсу	5	5	4	4

Для мультиплатформеної розробки мобільних застосунків усі розглянуті фреймворки встигли зарекомендувати себе, такими, які прагнуть скоротити час та витрати на розробку. Кожен із фреймворків може бути корисним залежно від пріоритетів та потреб проекту. У підсумку, для написання мобільного застосунку був обраний фреймворк React Native. Цей фреймворк поширюється безкоштовно і має відкритий вихідний код. Крім того, він використовує JavaScript. Мова

програмування JavaScript відзначається своєю динамічністю, і відповідністю стандарту ECMAScript. Вона широко застосовується для написання сценаріїв на веб-сторінках, що дозволяє реалізовувати взаємодію з користувачем, керувати браузером, виконувати асинхронний обмін даними з сервером, а також змінювати структуру і вигляд веб-сторінок на стороні клієнта. JavaScript відносять до прототипних мов програмування (що є підмножиною об'єктно-орієнтованих), зі скриптовим підходом та динамічною типізацією. Крім прототипної, дана мова частково підтримує інші програмні парадигми, такі як імперативна та функціональна, а також має певні архітектурні особливості, включаючи динамічну та слабку типізацію, автоматичне управління пам'яттю, прототипне наслідування і трактування функцій як об'єктів першого класу.

У React Native є два способи налаштування проекту – це React Native CLI або Expo CLI. React Native CLI – є офіційним інтерфейсом командного рядка для React Native. Він дозволяє отримати повний контроль над конфігурацією проекту та процесу збірки, надаючи більш традиційний підхід до розробки мобільних додатків. [24]

Отже перевагами React Native CLI є:

- повний контроль над процесом розробки і конфігурацією проекту;
- вільне використання сторонніх бібліотек та додавання власних модулів;
- створений програмний продукт важить менше на відміну від розробки на Expo CLI.

До недоліків React Native CLI можна віднести:

- труднощі під час створення та розробки проекту;
- для запуску проектів потрібні Android Studio і Xcode;
- відсутність вбудованих бібліотек для інтерфейсу та функціоналу;

React Native Expo – це набір інструментів, бібліотек і служб, створених навколо React Native, що має на меті спрощення та оптимізацію складного процесу розробки за допомогою React Native CLI. Так як він надає вже налаштоване середовище розробки, розпочати роботу над проектом можна доволі швидко, не витрачаючи час на складні конфігурації. [24]

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Ехро CLI має ряд переваг:

- запуск додатку React Native на фізичному пристрої через Ехро Go без необхідності налаштування робочого середовища значно спрощує процес тестування на різних пристроях;
- використання Ехро дозволяє швидко створювати прототипи, впроваджувати новий функціонал і оперативно реагувати на зміни;
- має обширну документацію, що спрощує пошук відповідей на питання або вирішення проблем, що виникають під час розробки;
- пропонує вбудований сервіс для публікації застосунків.

Серед недоліків можна виділити те що:

- розроблений проект важить більше ніж та сама розробка на React Native CLI;
- необхідність встановлення додаткових залежностей;
- обмеженість у роботі зі сторонніми модулями.

Для оптимізації процесу розробки мобільного застосунку було вирішено використовувати Ехро CLI, оскільки він надає зручний набір інструментів для швидкого створення, оновлення та тестування застосунків на різних платформах і його недоліки не є суттєвими для даної розробки.

Щоб проект був успішним необхідно продумати всі деталі до початку його розробки. Важливим фактором, який визначає майбутній результат – є правильно підібрана архітектура програмного продукту. Обрана структура прямо впливає на продуктивність, масштабованість, безпеку та організацію взаємодії об'єктів системи.

Для даного застосунку було обрано клієнт-серверну архітектуру. Вже з назви випливає, що у ній зосереджено два важливих компоненти: це сервер та клієнт. Модель цієї системи працює таким чином: програмне забезпечення з боку клієнта відправляє запит на сервер, і після обробки сервером результат повертається клієнту (Рис. 1.6). Передача даних між клієнтами та сервером буде реалізована за допомогою API.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		25



Рисунок 1.6. – Робота клієнт-серверної архітектури

Сторону клієнта буде створено на основі фреймворка React Native з використанням Expo CLI. Для серверної частини передбачається використання Nest.js – інноваційного фреймворку Node.js для розробки потужних, стабільних і легко масштабованих серверних застосунків. Він забезпечує структурований підхід до розробки, поєднуючи концепції об'єктно-орієнтованого та функціонального програмування. Nest працює на базі Express, але також є гнучким для легкої інтеграції з іншими бібліотеками, що дозволяє використовувати широкий спектр сторонніх рішень. Крім того у ньому підтримуються прогресивний JavaScript з повною підтримкою TypeScript, та широкий набір інструментів, таких як Dependency Injection, Middleware та багато іншого. [28]

Не менш важливим при визначенні засобів реалізації є вибір СУБД, яка найбільшою мірою відповідає визначеним вимогам програмного продукту. За рейтингом найпопулярніших систем управління базами даних за 2023 рік та аналізом відповідності різних систем до вимог проекту, було прийнято рішення використовувати СУБД об'єктно-реляційного типу, а саме – PostgreSQL. Дана СУБД забезпечує високу надійність, гнучкість та розширюваність, що відповідає потребам проекту. PostgreSQL підтримує автоматично оновлювані представлення, що дозволяє переглядати дані в режимі реального часу, підтримує тригери, що забезпечують виконання дій при зміні даних у таблицях, а також використовує зовнішні ключі для створення зв'язку між таблицями.

Для комунікації з базою даних було обрано фреймворк TypeORM. Він забезпечує зручний взаємозв'язок між об'єктами застосунку та таблицями, що дозволяє ефективно працювати з реляційними базами даних. TypeORM дозволяє легко визначати моделі даних, спрощуючи розробку і підтримку застосунку.[19, 39, 44]

Висновки до першого розділу

Проведений аналіз теоретичних та функціональних вимог до програмного забезпечення дав можливість окреслити ключові ідеї та етапи реалізації дипломного проекту. Було визначено, що програмний продукт для підбору індивідуальних програм харчування буде реалізовано у вигляді мобільного застосунку, оскільки це найбільше відповідає сучасним тенденціям та забезпечить зручний доступ користувачів до сервісу.

Також було виконано порівняльну оцінку аналогічних розробок, зокрема: «PlateJoy», «MyFitnessPal», «Lifesum», «Dine4Fit». Проведений аналіз дозволив виділити найбільш важливі функції та особливості, які необхідно врахувати при розробці власного продукту, щоб забезпечити конкурентоспроможність на ринку та задовольнити усі вимоги потенційних користувачів. Ними стали:

- забезпечення користувачів історією програм/планів харчування;
- повна українська локалізація інтерфейсу додатку;
- вибір кількості прийомів їжі;

У підсумку було обґрунтовано вибір архітектури та засобів реалізації програмного продукту:

- обрано клієнт-серверну архітектуру;
- React Native Expo обрано для клієнтської частини;
- Nest.js та TypeORM для побудови серверної частини;
- PostgreSQL для бази даних.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2. ПРОТОТИПУВАННЯ, ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1. Опис структури та варіантів використання системи

Для узгодження основного функціонування програмного комплексу та ідентифікації і візуалізації різноманітних сценаріїв роботи користувачів з системою було створено діаграму варіантів використання (Рис. 2.1). Даний тип діаграм є відмінним інструментом для відображення можливостей взаємодії користувачів різних ролей із програмним продуктом, без прив'язки до порядку виконання дій або конкретного функціоналу. [6]

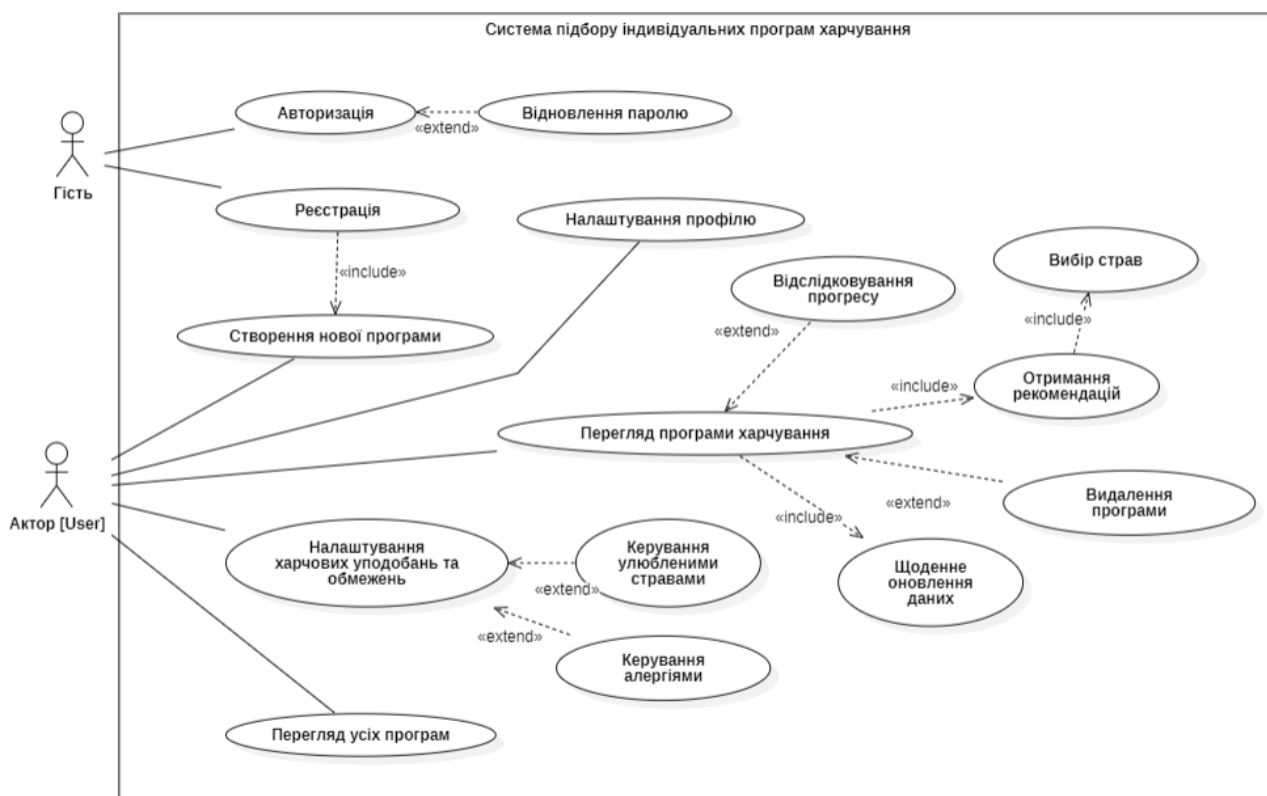


Рисунок 2.1. – Діаграма варіантів використання

Зображення варіантів використання застосунку передбачає, що із системою буде взаємодіяти лише два актори: гість, та актор, якому надано роль User. Для управління доступом до різних функцій системи застосовується модель контролю доступу на основі ролей (RBAC). Дана модель дозволяє чітко розмежувати доступ до функціональності системи, тим самим забезпечити

безпеку та зручність використання. Завдяки цьому підходу, можна легко керувати ролями доступу та змінювати їх залежно від потреб. [28]

У таблицях 2.1 – 2.13 подано опис сценаріїв використання системи.

Таблиця 2.1

Прецедент «Авторизація»

Назва	Авторизація
Актори	Гість
Опис сценарію	Гість вводить дані обліковго запису (ім'я та пароль) для доступу до системи. Система перевіряє введені дані і надає доступ до особистого кабінету у разі успішної авторизації.

Таблиця 2.2

Прецедент «Реєстрація»

Назва	Реєстрація
Актори	Гість
Опис сценарію	Гість заповнює форму реєстрації, вводячи необхідні дані (ім'я, прізвище, електронна пошта, пароль тощо). Після заповнення форми система створює нового користувача та надає йому роль User.

Таблиця 2.3

Прецедент «Відновлення паролю»

Назва	Відновлення паролю
Актори	Користувач (User)
Опис сценарію	Користувач натискає на посилання "Забули пароль?", вводить свою електронну пошту. На пошту відправляються інструкції для відновлення паролю. Користувач слідує інструкціям.

Таблиця 2.4

Прецедент «Створення нової програми»

Назва	Створення нової програми
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу створення нової програми. Вводить необхідні дані та виконує налаштування. Система зберігає нову програму та надає доступ до неї користувачу.

Таблиця 2.5

Прецедент «Видалення програми»

Назва	Видалення програми
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу профіль, потім програма харчування. Обирає видалення. Програма змінює статус і стає недоступною.

Таблиця 2.6

Прецедент «Налаштування профілю»

Назва	Налаштування профілю
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу налаштувань профілю. Він може змінювати особисті дані, такі як ім'я, прізвище, електронна пошта. Після внесення змін користувач зберігає нові налаштування, і система оновлює відповідну інформацію.

Таблиця 2.7

Прецедент «Перегляд усіх програм»

Назва	Перегляд усіх програм
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу перегляду всіх програм. Він може переглядати список усіх його програм, фільтрувати їх за різними критеріями, а також отримувати детальну інформацію про кожну програму.

Таблиця 2.8

Прецедент «Керування улюбленими стравами»

Назва	Керування улюбленими стравами
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу улюблених страв. Він може додавати нові страви до списку улюблених, видаляти існуючі. Система зберігає зміни, внесені користувачем.

Прецедент «Керування алергіями»

Назва	Керування алергіями
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу керування алергіями. Він може додавати інформацію про свої алергії, або видаляти існуючі записи. Система зберігає дані, внесені користувачем також система використовує цю інформацію для персоналізації рекомендацій страв.

Таблиця 2.10

Прецедент «Керування алергіями»

Назва	Керування алергіями
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу керування алергіями. Він може додавати інформацію про свої алергії, або видаляти існуючі записи. Система використовує цю інформацію для персоналізації рекомендацій страв.

Таблиця 2.11

Прецедент «Щоденне оновлення даних»

Назва	Щоденне оновлення даних
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та оновлює свої дані про вагу. Система зберігає ці дані та використовує їх для аналізу прогресу та перерахунку щоденної норми калорій та макронутрієнтів.

Таблиця 2.12

Прецедент «Відслідковування прогресу»

Назва	Відслідковування прогресу
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу профілю. Він може переглядати графіки про свій прогрес, аналізувати зміни у своїх фізичних показниках та результатах виконання програми.

Прецедент «Вибір страв»

Назва	Відслідковування прогресу
Актори	Користувач (User)
Опис сценарію	Користувач входить в систему та переходить до розділу вибору страв. Він може переглядати доступні страви, фільтрувати їх за категоріями, та вподобаннями, а також додавати вибрані страви до свого меню.

Однією з ключових вимог при проектуванні будь-якої системи є створення зрозумілої та легко розширюваної архітектури. Nest JS має перевагу у цьому, оскільки використовує модульну архітектуру, що робить його масштабування простішим, а розробку та підтримку – гнучкішою. Діаграму компонентів серверної частини застосунку представлено в додатках (Рис. А.1). Стислий опис модулів серверної частини наведено у таблицях 2.14 – 2.24.

Таблиця 2.14

Модуль «users»

Назва	Users
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи з користувачами).
	Module (підключення компонентів).
Призначення	Модуль керує всіма операціями, пов'язаними з користувачами.
Взаємодія з іншими модулями	healthinfo, meal-program, dish, ingredient.

Таблиця 2.15

Модуль «meal-program»

Назва	meal-program
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи з програмами харчування).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними з програмами харчування.
Взаємодія з іншими модулями	user, dish, daily-info, goal, activitylevel, macrosplit.

Таблиця 2.16

Модуль «dish»

Назва	Dish
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи зі стравами).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними зі стравами.
Взаємодія з іншими модулями	daily-info, dish-category, meal-program, ingredient, user.

Таблиця 2.17

Модуль «dish-category»

Назва	dish-category
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи із категоріями страв).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із категоріями страв.
Взаємодія з іншими модулями	Dish.

Таблиця 2.18

Модуль «ingredient»

Назва	ingredient
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи із інгредієнтами які використовують у стравах і можуть бути алергіями).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із інгредієнтами.
Взаємодія з іншими модулями	dish, ingredient-category.

Таблиця 2.19

Модуль «ingredient-category»

Назва	ingredient-category
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи із категоріями, які використовують для інгредієнтів).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із категоріями інгредієнтів.
Взаємодія з іншими модулями	ingredient-category.

Таблиця 2.20

Модуль «healthInfo»

Назва	healthInfo
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (бізнес-логіка для роботи із даними про здоров'я і фізичними параметрами користувача).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із даними про здоров'я.
Взаємодія з іншими модулями	user, mealprogram

Таблиця 2.21

Модуль «daily-info»

Назва	daily-info
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (керує щоденними даними користувачів (ваги, споживанням калорій)).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із щоденними даними.
Взаємодія з іншими модулями	mealprogram, dish, healthInfo

Таблиця 2.22

Модуль «activityLevel»

Назва	activityLevel
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (керує даними про рівень активності).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із рівнем активності.
Взаємодія з іншими модулями	mealprogram.

Таблиця 2.23

Модуль «goal»

Назва	goal
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (керує даними про ціль).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із цілями користувача у програмах харчування.
Взаємодія з іншими модулями	mealprogram.

Модуль «macrosplit»

Назва	macrosplit
Основні компоненти	Controller (обробка HTTP-запитів).
	Service (керує даними про розподіл БЖВ).
	Module (підключення компонентів).
Призначення	Керує операціями, пов'язаними із розподілом білків, жирів, вуглеводів.
Взаємодія з іншими модулями	mealprogram.

Щодо клієнтської частини, то у світі React архітектура означає розподіл обов'язків між спеціальними компонентами, які виконують лише одну функцію. Тому логічно, що архітектура клієнтської частини містить декілька рівнів. Основою усієї архітектури і вхідною точкою є компонент App, до якого під'єднані усі інші сервіси. Важливим елементом цього компоненту є глобальне середовище Store, яке зберігає дані, необхідні для роботи програми і створюється при кожному запуску застосунку. Наступний рівень містить частину бізнес-логіки, а саме компоненти: Router, Stack, Containers, CombineViews та Views:

- **router** відповідає за переміщення користувача між різними екранами програми;
- **stack** відповідає за вікна програми, які відкриваються за допомогою маршрутизатора і передають параметри до Containers та CombineViews;
- **container** отримує дані зі сховища, перетворює їх у більш придатну для перегляду форму, зберігає деякі проміжні стани, які призначені для певної частини програми;
- **view** є найменшими будівельними блоками програми, як кнопки, чекбокси, заголовки тощо.

Діаграму компонентів клієнтської частини застосунку представлено в додатках (Рис. А.2).

2.2. Проектування та реалізація роботи алгоритмів застосунку

Рациональне харчування – це грамотне планування прийомів їжі, яке базується на таких чинниках, як: стать, вік, тип діяльності, хвороби та інші фактори. Воно включає дотримання наступних принципів [7, 11, 13, 34]:

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

- харчування має розподілятися рівномірно протягом доби і бути сталим за годинами;
- прийоми їжі мають бути частими і подрібненими (для здорової людини найбільш корисні 3-5 прийомів їжі на день);
- харчові продукти мають бути збалансованими і правильно розподіленими на різні прийоми їжі протягом дня. Найбільш ефективний режим передбачає, що близько третини загальної кількості їжі припадає на сніданок, трохи більше третини – на обід і менше третини – на вечерю;
- інтервал між сніданком і обідом, а також між обідом і вечерею, повинен бути однаковим і становити 5-6 годин. У цей проміжок часу важливо включати перекуси.

Розрахунок денної калорійності є важливою складовою забезпечення достатньої енергії для організму. Цей процес дозволяє визначити кількість калорій та знайти баланс розподілу білків, жирів та вуглеводів в раціоні, необхідних для підтримки оптимального фізичного і психічного функціонування організму протягом дня.

На сьогоднішній день існує декілька методів розрахунку калорійності [20]:
Формула калорійності Харріса–Бенедикта.

На жаль, зараз ця формула надто завищує калорійні потреби людини, приблизно на 5-10%. Цей ефект пропорційно залежить від ваги: чим вона вища, тим вище цей показник. Розрахунки виконуються таким чином [2, 21]:

$$BMR_m = 66,4730 + (13,7516 \times a) + (5,0033 \times b) - (6,7550 \times c), \quad (2.1)$$

$$BMR_w = 655,0955 + (9,5634 \times a) + (1,8496 \times b) - (4,6756 \times c), \quad (2.2)$$

де a – маса тіла в кілограмах;

b – зріст в сантиметрах;

c – вік в роках.

Залежно від того, який у людини спосіб життя, вибирається відповідний коефіцієнт. У таблиці 2.25 подано значення коефіцієнту денної активності.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Значення коефіцієнтів денної активності

Малорухливий спосіб життя	$k = 1.2$
Легка активність, 1-3р на тиждень	$k = 1.375$
Помірна активність, 4-5 р на тиждень	$k = 1.55$
Інтенсивні тренування, 3-4р на тиждень	$k = 1.725$
Дуже інтенсивні тренування, тяжка фізична робота щодня	$k = 1.9$

Розрахунок добової норми калорій визначається за формулою:

$$E_{\text{день}} = BMR \times k, \quad (2.3)$$

де BMR – значення базального метаболізму;

k – коефіцієнт активності.

Формула калорійності Мафін–Джеора.

Сьогодні дана формула вважається однією з найбільш точних та ефективних для розрахунку калорій. Вона актуальна для осіб віком від 13 до 80 років. Розрахунки виконуються наступним чином:

$$BMR_{\text{ж}} = (10 \times a + 6.25 \times b - 5 \times c - 161) \times k_i, \quad (2.4)$$

$$BMR_{\text{ч}} = (10 \times a + 6.25 \times b - 5 \times c + 5) \times k_i, \quad (2.5)$$

де a – маса тіла в кілограмах;

b – зріст в сантиметрах;

c – вік в роках;

k_i – коефіцієнт денної активності.

Коефіцієнт денної активності визначається залежно від ступеня фізичних навантажень (див. табл. 2.25). Недоліком даної формули є відсутність врахування співвідношення м'язів і жиру в організмі. Оскільки рівень метаболізму залежить від м'язової маси, формула не враховує цей показник при визначенні потреби в енергії.

Формула калорійності Кетч–МакАрдила

Дана формула відмінна від інших тим, що враховує вміст жиру в організмі, однак вона не враховує інших параметрів (стать, вік тощо), бо передбачається,

що ці показники можуть бути враховані при визначенні вмісту жиру. Формула розрахунку базального метаболізму виглядає наступним чином [2, 21]:

$$BMR = 370 + 21,6 \times m, \quad (2.6)$$

де m – це маса тіла за вирахуванням жиру.

Маса тіла за вирахуванням жиру обчислюється за формулою:

$$LBM = \frac{a \cdot (100 - k)}{100}, \quad (2.7)$$

де a – маса тіла в кілограмах;

k – відсоток жиру

Підсумкове значення, отримане за цією формулою, варто множити на коефіцієнт активності (Табл. 2.25).

Розрахована кількість калорій на день дозволяє визначити показник білків, жирів та вуглеводів, використовуючи наступні формули [5, 13]:

$$\text{Білки (1 г на 1 кг ваги)} = \frac{E_{\text{день}} \times k}{4}, \quad (2.8)$$

$$\text{Жири (0,7 – 0,9 г на 1 кг ваги)} = \frac{E_{\text{день}} \times k}{9}, \quad (2.9)$$

$$\text{Вуглеводи (4 г на 1 кг ваги)} = \frac{E_{\text{день}} \times k}{4}, \quad (2.10)$$

де k – відсоток макронутрієнтів у раціоні;

$E_{\text{день}}$ – обчислена вище енергетична потреба на день.

В залежності від виду фітнес-дієти визначається відсоток поживних речовин у щоденному раціоні. Так білкова фітнес-дієта базується на збільшенні вмісту білків у раціоні, і раціон набуває наступного вигляду: 20% – вуглеводів, 30% – жирів, 50% – білків, у той час як збалансована дієта має розподіл: 40% – вуглеводів, 30% – жирів та 30% – білків. [2, 5]

Не менш важливим моментом для реалізації алгоритмів майбутнього застосування є розрахунок індексу маси тіла (ІМТ), цей показник є найпростішим та найпоширенішим методом оцінки ступеня ожиріння і вираховується за формулою:

$$IMT = \frac{m}{h^2}, \quad (2.11)$$

де m – маса тіла в кілограмах;

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

h – зріст в сантиметрах.

З огляду на дані, опрацьовані вище можна визначити основні етапи на яких базується розрахунок денної калорійності:

- розрахунок базального метаболізму;
- розрахунок енергетичної потреби на день;
- розрахунок необхідної калорійності на день, що визначається з урахуванням цілі та дієтичних обмежень користувача;
- розрахунок БЖВ.

Оскільки кожна з формул має різні підходи та методики для розрахунку базального метаболізму, обрати одну з них для реалізації алгоритмів додатку доволі складно. Щоб забезпечити простий та доступний процес розрахунку калорійності буде використано дві формули які будуть визначатися і використовуватися на базі введених користувачем параметрів. Якщо користувач вказуватиме відсоток жиру, то для обрахунків денної потреби у калоріях буде використано формулу Кетч-МакАрдила. В інакшому випадку система застосовуватиме формулу Мафін-Джеора. Діаграму послідовності обрахунків наведено в додатках (див. Рис. Б.1).

Обрахунки виконуються на частині клієнта. Функції обрахунків виокремлені у окремий файл під назвою calculateParams.js.

Нижче наведено код для обрахунку базального рівня метаболізму:

```
export const countBMR = (
  gender,
  numericWeight,
  numericAge,
  numericHeight,
  bodyFat
) => {
  let res = 0;
  if (!isNaN(bodyFat)) {
    // Katch-McArdle
    const lbm = (numericWeight * (100 - bodyFat)) / 100;
    res = 370 + 21.6 * lbm;
  } else {
    // Mifflin-St Jeor Equation
    if (gender === "female") {
      res = 10 * numericWeight + 6.25 * numericHeight - 5 * numericAge - 161;
    } else {
      res = 10 * numericWeight + 6.25 * numericHeight - 5 * numericAge + 5;
    }
  }
}
```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

```
    return res;
  };
```

На базі отриманих результатів виконуються розрахунки денної норми калорій. Лістинг коду наведено нижче:

```
export const countCaloriesWithGoal = (bmr, activity, goal) => {
  const bareCal = Math.round(bmr * activity.coefficient);
  if (goal?.value !== null && goal?.value !== undefined) {
    return bareCal + goal.value;
  } else if (goal?.coefficient !== null && goal?.coefficient !== undefined) {
    const caloriesReduction = bareCal * goal.coefficient;
    return bareCal - caloriesReduction;
  } else {
    return bareCal;
  }
};
```

Не менш важливим є розрахунок добової норми нутрієнтів, який бере до уваги значення денної норми калорій та відсоток БЖУ, який визначається на основі обраної дієти. Лістинг коду обрахунків:

```
export const countPCF = (
  cal,
  proteinPercentage,
  carbsPercentage,
  fatPercentage
) => {
  const proteinGrams = Math.round((cal * (proteinPercentage / 100)) / 4.0);
  const carbsGrams = Math.round((cal * (carbsPercentage / 100)) / 4.0);
  const fatGrams = Math.round((cal * (fatPercentage / 100)) / 9.0);
  return { proteinGrams, carbsGrams, fatGrams };
};
```

Повний лістинг коду з обрахунками наведено в Додатку Б.

Застосунок не побудований на самих лише обрахунках. У системі передбачено ряд інших функцій та алгоритмів які забезпечують його правильне функціонування. Важливим етапом роботи програми є перегляд самої програми харчування. Саме цей крок є основним і буде виконуватись ледь не при кожному запуску. На рисунку 2.2 зображено діаграму активності для одержання програми харчування. Вона описує логічну послідовність дій та рішень, необхідних для створення та управління програмою харчування як для клієнта, так і системи. [6]

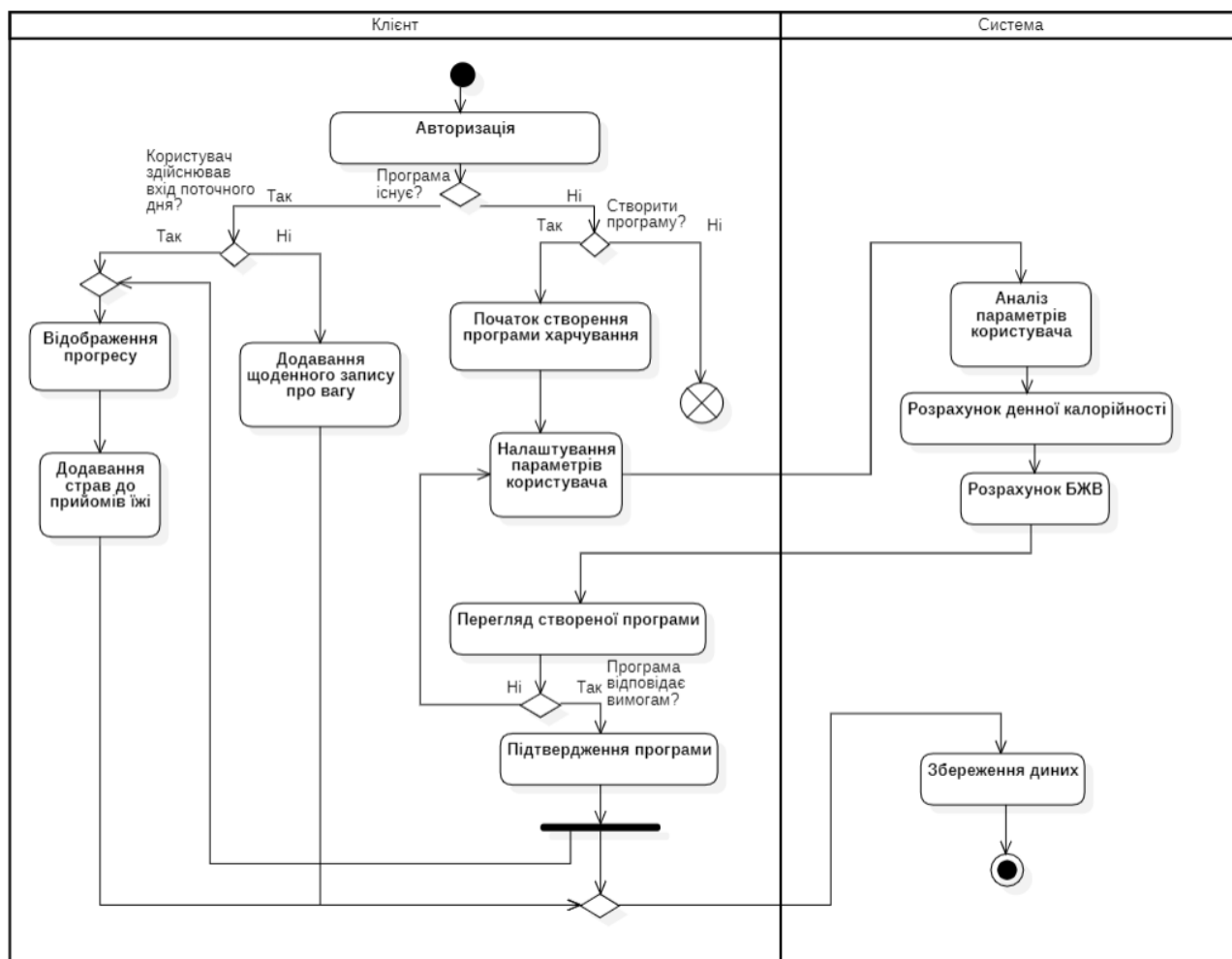


Рисунок 2.2. – Діаграма активності для одержання програми харчування

Завдяки даній діаграмі можна побачити, як одна діяльність призводить до іншої, що дозволяє зрозуміти приховану взаємодію користувача і системи, не заглиблюючись у подробиці технічної реалізації. Для початку активності користувачу потрібно увійти в систему. Якщо він має програму і це не перший його вхід – буде відображено прогрес користувача і він зможе додавати страви до прийомів їжі. Якщо такий вхід перший, то варто для початку вказати інформацію про вагу. Якщо програми не існує – користувач може створити її. Для цього необхідно зробити деякі налаштування, після яких системою будуть здійснені обрахунки КБЖВ. Програму можна зберегти або переналаштувати.

2.3. Проектування бази даних

База даних є фундаментом для зберігання інформації в організованому та доступному вигляді. Будь-яка програма взаємодіє з різноманітними видами

даних, серед яких можуть бути текстова інформація, зображення, відео, тощо. Завдяки БД можна досягти структурованого підходу до зберігання важливих елементів даних, що значно полегшує і прискорює роботу.

Створення таблиць і зв'язків між ними відбувалось з використанням однієї з функцій TypeORM, а саме автоматичного створення схеми таблиць БД на основі моделей. Це значно полегшило розробку та розгортання системи, оскільки усі зміни підтягуються автоматично завдяки синхронізації схеми даних з кодом. Головну роль у цьому відіграють спеціально визначені класи для кожної сутності, яким додано декоратор @Entity. [35, 44]

Класи сутностей мають наступний вигляд:

```
@Entity()
export class DailyInfo extends AbstractEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  @IsString()
  date: Date;

  @Column('float')
  @IsNumber()
  dailyCarbohydrates: number;

  @Column('float')
  @IsNumber()
  dailyProteins: number;

  @Column('float', { nullable: true })
  @IsNumber()
  dailyCalories: number | null;

  @Column('float')
  @IsNumber()
  dailyFat: number;

  @Column({ nullable: true })
  @IsNumber()
  mealsNumber?: number | null;

  @Column('float')
  @IsNumber()
  currentWeight: number;

  @ManyToOne(() => MealProgram, (mealProgram) => mealProgram.dailyInfo)
  mealProgram: MealProgram;

  @OneToMany(() => DailyDishes, (dailyDishes) => dailyDishes.dailyInfo)
  dailyDishes: DailyDishes[];
}
```

Даний клас визначає сутність DailyInfo і має усі необхідні поля та зв'язки для представлення щоденної інформації про харчування користувача в базі даних. За допомогою декораторів бібліотек class-validator та TypeORM у легкий спосіб було визначено типи полів, зв'язки та забезпечено перевірку даних. DailyInfo, як і більшість інших класів-сутностей у даному проєкті, успадковує властивості та методи від класу AbstractEntity. Це дозволяє повторно використовувати спільний функціонал, визначений у батьківському класі. [44]

База даних, спеціально спроектована для застосунку, містить 17 таблиць, що включають у себе важливі для функціонування програми дані. Діаграма БД створена за допомогою інструменту адміністрування і управління – pgAdmin, розміщена на рисунках В.1 та В.2. Спрощений варіант представлення таблиць і зв'язків наведено на рисунку 2.3.

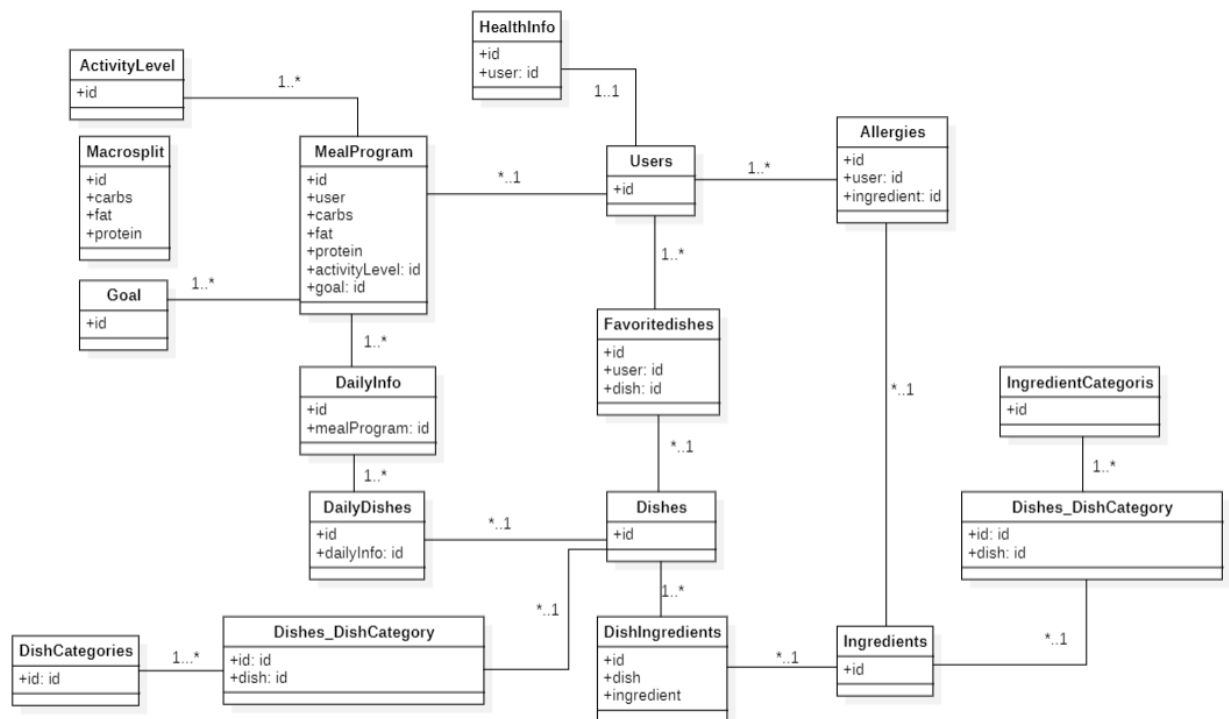


Рисунок 2.3. – Спрощена діаграма БД

Діаграма допомагає зрозуміти, як дані організовані в додатку і які зв'язки між таблицями. Оскільки це спрощений варіант, в таблицях наведено лише деякі поля таблиць, в більшості ті, які необхідні для реалізації зв'язків між таблицями. Також деякі назви таблиць можуть дещо відрізнятись від тих, що є в БД.

На схемі чітко проглядається використання різних типів зв'язків. За допомогою зв'язку «один-до-одного» (one-to-one), за якого одна з таблиць посилається на іншу, реалізовано зв'язок між сутностями «HealthInfo» та «Users». Таким чином «HealthInfo» має зовнішній ключ «User», що дає змогу визначити якому користувачу належить запис про здоров'я. Між таблицями також є ряд інших зв'язків, як от «один-до-багатьох» (one-to-many) та «багато-до-багатьох» (many-to-many). Для реалізації зв'язку «багато-до-багатьох» було створено додаткові таблиці які виконують розподільну функцію і розбивають даний тип зв'язку на зв'язки «один-до-багатьох». [12]

Короткий опис деяких елементів бази наведено у табл. 2.26 – 2.38.

Таблиця 2.26

Опис таблиці «Users»

Полу	Тип	Опис
id	varchar	Унікальний ідентифікатор користувача
firstName	varchar(50)	Ім'я користувача
lastName	varchar(50)	Прізвище користувача
email	varchar	Email користувача
profile	varchar	Профіль користувача
token	varchar	Токен користувача
roles	enum	Роль користувача. Поле NOT NULL, значення з переліку Role (по замовчуванню User)
healthInfo	HealthInfo	Інформація про здоров'я користувача. Зв'язок OneToOne відношення з таблицею HealthInfo, зв'язок через поле user
allergies	Ingredients[]	Алергії користувача. Зв'язок ManyToMany відношення з таблицею Ingredients, зв'язок через проміжну таблицю
favoritedishes	Dishes[]	Улюблені страви користувача. Зв'язок ManyToMany відношення з таблицею Dishes, зв'язок через проміжну таблицю
mealProgram	MealProgram[]	Програми харчування користувача. Зв'язок OneToMany відношення з таблицею MealProgram, зв'язок через поле user
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.27

Опис таблиці «IngredientCategory»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
name	varchar	Назва інгредієнта
ingredient	Ingredients[]	Зв'язок ManyToOne, відношення з таблицею Ingredients
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.28

Опис таблиці «HealthInfo»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
age	int	Вік
weight	float	Вага користувача
height	float	Зріст користувача
bodyfat	float	Відсоток жирової маси тіла
gender	enum	Стать користувача, значення з переліку Gender (по замовчуванню Gender.MALE)
user	Users	Посилання на користувача. Зв'язок OneToOne з таблицею Users, зв'язок через поле user, NOT NULL
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.29

Опис таблиці «Ingredients»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
title	varchar	Назва інгредієнта
description	varchar	Опис інгредієнта
ingredient_photo	varchar	Фото інгредієнта
carbohydrates	float	Кількість вуглеводів
protein	float	Кількість білків
fat	float	Кількість жирів
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Опис таблиці «MealProgram»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
requiredCalories	float	Необхідна кількість калорій
requiredFat	float	Необхідна кількість жирів
requiredCarbohydrates	float	Необхідна кількість вуглеводів
requiredProteins	float	Необхідна кількість білків
goal	Goal	Ціль програми харчування. Зв'язок ManyToOne з таблицею Goal
activityLevel	ActivityLevel	Рівень активності користувача. Зв'язок ManyToOne з таблицею ActivityLevel
macroSplitDietCarbs	float	Співвідношення макроелементів: вуглеводи
macroSplitDietFat	float	Співвідношення макроелементів: жири
macroSplitDietProtein	float	Співвідношення макроелементів: білки
startDate	Date	Дата початку програми
endDate	Date	Запланована дата закінчення програми
actualEndDate	Date	Фактична дата закінчення програми
startWeight	float	Початкова вага користувача
expectedWeight	float	Очікувана вага після завершення програми
user	Users	Посилання на користувача. Зв'язок ManyToOne, відношення з таблицею Users.
dailyInfo	DailyInfo[]	Щоденна інформація про програму. OneToMany відношення з таблицею DailyInfo
status	enum	Статус програми
created	default: current_timestamp	Дата та час створення
updated	default: timestamp, on update: timestamp	Дата та час оновлення

Таблиця 2.31

Опис таблиці «Dishes»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
title	varchar	Назва страви
description	varchar	Опис страви
dish_photo	varchar	Фото
dish_video	varchar	Відео
carbohydrates	float	Кількість вуглеводів
calories	float	Кількість калорій
protein	float	Кількість білків
fat	float	Кількість жирів
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.32

Опис таблиці «ActivityLevel»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
name	varchar	Назва
coefficient	float	Коефіцієнт рівня активності
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.33

Опис таблиці «DishCategory»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
name	varchar	Назва категорії страви
dish	Dishes[]	Зв'язок ManyToOne, відношення з таблицею Dishes
dish_photo	varchar	Фото
dish_video	varchar	Відео
carbohydrates	float	Кількість вуглеводів
calories	float	Кількість калорій
protein	float	Кількість білків
fat	float	Кількість жирів
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.34

Опис таблиці «DishIngredients»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
count	float	Кількість інгредієнта
unit_of_measurement	varchar	Одиниця виміру
description	varchar	Опис інгредієнта
dish	Dishes	Відношення до таблиці Dishes
ingredient	Ingredients	Відношення до таблиці Ingredients
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.35

Опис таблиці «Goal»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
name	varchar	Назва
coefficient	float	Коефіцієнт цілі (залишається порожнім, якщо value не null)
value	float	Значення цілі (залишається порожнім, якщо coefficient не null)
weightChangePerWeek	float	Зміна ваги на тиждень
bmiCategory	BMICategory[]	Масив категорій BMI
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.36

Опис таблиці «DailyInfo»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
date	Date	Дата
dailyCarbohydrates	float	Кількість вуглеводів
dailyProteins	float	Кількість білків
dailyCalories	float	Кількість калорій
dailyFat	float	Кількість жирів
mealsNumber	int	Кількість прийомів їжі за день
currentWeight	float	Поточна вага
mealProgram	MealProgram	Посилання на програму харчування, ManyToOne
dailyDishes	DailyDishes[]	Щоденні страви, OneToMany відношення з таблицею DailyDishes
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Таблиця 2.37

Опис таблиці «DailyDishes»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
dishId	Date	Дата
mealType	enum	Тип прийому їжі
dish	Dishes	Відношення до таблиці Dishes, ManyToOne
dailyInfo	DailyInfo	Відношення до таблиці DailyInfo, ManyToOne
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Опис таблиці «Macrosplit»

Полю	Тип	Опис
id	int	Унікальний ідентифікатор запису
name	varchar	Назва дієти
carbs	float	Кількість вуглеводів
fat	float	Кількість жирів
protein	float	Кількість білків
created	default: current_timestamp	Дата та час створення
updated	default: current_timestamp, on update: current_timestamp	Дата та час оновлення

Для автоматичного оновлення статусу програм харчування користувача було створено тригер та функцію в базі даних PostgreSQL. Лістинг коду функції та тригера наведено нижче [38]:

```
CREATE OR REPLACE FUNCTION update_meal_program_status() RETURNS TRIGGER AS $$
BEGIN
  --Перевіряємо чи існує запис зі статусом 'Триває'
  IF EXISTS (
    SELECT 1
    FROM meal_program
    WHERE "meal_program"."userId" = NEW."userId"
      AND status = 'Триває'
  ) THEN
    --Оновлюємо існуючий запис
    UPDATE meal_program
    SET status = 'Не завершено'
    WHERE "meal_program"."userId" = NEW."userId"
      AND status = 'Триває';
  END IF;

  -- Додаємо запис
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER update_meal_program_status_trigger
BEFORE INSERT ON meal_program
FOR EACH ROW
EXECUTE FUNCTION update_meal_program_status();
```

Функція `update_meal_program_status()` відповідає за перевірку активної програми харчування користувача зі статусом 'Триває'. Якщо при вставці даних така програма вже існує, то її статус буде змінено на 'Не завершено', а нова програма буде додана. Це дозволяє уникнути ситуацій, коли у користувача одночасно є декілька активних програм харчування.

Тригер `update_meal_program_status_trigger` виконує функцію `update_meal_program_status()` перед вставкою нового запису в таблицю `meal_program`. Після виконання цього тригера статуси програм харчування

користувача автоматично оновлюються. Це дозволяє забезпечити послідовність та коректність роботи системи.

Для забезпечення оновлення поточної ваги користувача при вставці нових записів було створено ще одну комбінацію тригера та функції.

Лістинг коду:

```
CREATE OR REPLACE FUNCTION update_health_info_weight() RETURNS TRIGGER AS $$
DECLARE
    v_userId VARCHAR;
BEGIN
    SELECT "userId"
    INTO v_userId
    FROM "meal_program"
    WHERE "meal_program"."id" = NEW."mealProgramId";
    -- Оновлюємо поле weight в таблиці healthInfo
    UPDATE "health_info"
    SET "weight" = NEW."currentWeight"
    WHERE "health_info"."userId" = v_userId;
    -- Повертаємо новий запис для вставки в daily_info
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER update_health_info_weight_trigger
AFTER INSERT ON daily_info
FOR EACH ROW
EXECUTE FUNCTION update_health_info_weight();
```

Функція `update_health_info_weight()` відповідає за оновлення ваги користувача при вставці даних в таблицю `daily_info`. Це дозволяє значно заощадити ресурси та забезпечити актуальність даних користувача.

2.4. Реалізація серверної частини застосунку підбору індивідуальних програм харчування

У серверній частині застосунку було реалізовано структуру, що включає сервіси (services), контролери (controllers) та модулі (modules), для забезпечення функціональності та організованості додатку.

Бізнес логіку застосунку було побудовано у сервісах. Саме в них зосереджено методи для обробки даних, взаємодії з базою даних та виконання інших завдань. Наприклад, `UserService` містить методи для створення, оновлення, пошуку та видалення користувачів, а також для управління їх алергіями та улюбленими стравами. Одним з таких є метод пошуку користувачів за електронною поштою. Лістинг коду методу:

```
async findOneUserByEmail(email: string) {
    return await this.userRepository.findOne({ where: { email: email } });}
```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Ще один метод, який є дещо складнішим, дозволяє додавати страви до улюблених. Спершу він перевіряє, чи існують користувач та обрана страва, а згодом додає новий елемент у масив улюблених страв. Лістинг методу приведено нижче:

```
async addFavDish(id, addfavdishDto: FavDishDto): Promise<Users> {
  const user = await this.userRepository.findOne({
    where: { id: id },
    relations: ['favoritedishes'],
  });

  if (!user) {
    throw new NotFoundException('Користувача не знайдено');
  }

  const dish = await this.dishRepository.findOne({
    where: { id: addfavdishDto.dishId },
  });
  if (!dish) {
    throw new NotFoundException('Страви не знайдено');
  }
  const dishExists = user.favoritedishes.some(
    (favDish) => favDish.id === addfavdishDto.dishId,
  );
  if (dishExists) {
    throw new HttpException(
      'Ця страва вже додана до улюблених користувача',
      HttpStatus.CONFLICT,
    );
  }
  user.favoritedishes = [...user.favoritedishes, dish];
  return this.userRepository.save(user);
}
```

Загальний лістинг UsersService можна побачити у додатку Г.

Роль контролерів полягає в обробці HTTP-запитів та взаємодії з сервісами. Вони отримують запити від клієнта, викликають відповідні методи сервісів і повертають результати клієнту. Для прикладу, контролер MealProgramController відповідає за обробку запитів, пов'язаних з програмами харчування користувачів. Він забезпечує створення, оновлення, отримання та видалення програм харчування для автентифікованих користувачів. Контролер використовує сервіси для виконання основної логіки, а також інтерсептори для серіалізації даних і охоронців для забезпечення доступу на основі ролей (дані поняття будуть описані згодом). [26]

Приклад контролера для отримання програми харчування поточного користувача за статусом:

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

```

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@Get('mealprogram/:status')
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
async getCurrentUserMealProgramsByStatus(
  @CurrentUser('sub') id: string,
  @Param('status') status: Status,
): Promise<any> {
  const res = await this.mealProgramService.findUserMealProgramByStatus(
    id,
    status,
  );
  if (res) {
    return {
      success: true,
      result: res,
    };
  } else {
    return {
      success: false,
      result: [],
    };
  }
}

```

Загальний лістинг MealProgramController можна побачити у додатку Г.

Модулі дозволяють групувати пов'язані компоненти системи. Вони відкривають можливість керування залежностями між різними частинами проекту, що робить його гнучкішим і масштабованішим. Основний модуль додатку, який відповідає за взаємодію всіх сутностей представлено у додатку Г. Одним із головних етапів розробки серверної частини стало підключення БД. Доступ до неї у даному проекті був забезпечений із застосуванням TypeORM. Підключення здійснюється за допомогою створеного класу ConfigDB та конфігураційних параметрів, які зберігаються у змінних середовища поза вихідним кодом програми. [44]

Лістинг коду підключення БД (загальний код класу наведено в додатку Г):

```

public getTypeOrmConfig(): TypeOrmModuleOptions {
  return {
    type: 'postgres',
    host: this.getValue('POSTGRES_HOST'),
    port: parseInt(this.getValue('POSTGRES_PORT')),
    username: this.getValue('POSTGRES_USER'),
    password: this.getValue('POSTGRES_PASSWORD'),
    database: this.getValue('POSTGRES_DATABASE'),
    entities: [__dirname + '/*/**/*.entity{.js, .ts}'],
    autoLoadEntities: true,
    synchronize: true,
  };
}

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

TypeORM підтримує патерн репозиторій, тому кожна сутність має власний репозиторій, який може бути отриманий з підключення до БД. Це абстрагує логіку доступу до даних від решти програми. Кожен репозиторій відповідає за виконання CRUD операцій і взаємодію з базою даних, забезпечуючи чистий і структурований підхід до управління даними, та приховуючи складнощі зберігання і пошуку даних. [35]

Для роботи з базою даних, яка відбувалась у сервіс-файлах було застосовано 2 різні сценарії відправки запитів. За першого підходу відбувається використання методів `findOne`, `find` і подібних. Приклад запиту для отримання усіх даних з таблички `dailyInfo` з вибором лише стовпців поточної ваги і дати за ідентифікатором програми харчування наведено нижче:

```
const dailyInfoList = await this.dailyInfoRepository.find({
  where: { mealProgram: { id: mealProgramId } },
  select: ['currentWeight', 'date'],
  relations: ['mealProgram'],
});
```

Цей підхід є простим та зрозумілим. Він дозволяє використовувати вбудовані методи репозиторію для отримання даних. Однак у складніших сценаріях він може виявитись обмеженим. Другий підхід базується на використанні `QueryBuilder`, що забезпечує більше можливостей і є зручнішим у випадку створення складних запитів. [39]

Приклад запиту з використанням другого підходу:

```
const res = await this.dailyInfoRepository
  .createQueryBuilder('dailyInfo')
  .leftJoinAndSelect('dailyInfo.mealProgram', 'mealProgram')
  .leftJoinAndSelect('dailyInfo.dailyDishes', 'dailyDishes')
  .leftJoinAndSelect('dailyDishes.dish', 'dish')
  .where('mealProgram.id = :idMealProgram', { idMealProgram })
  .andWhere('DATE(dailyInfo.date) = :date', { date: dateString })
  .getOne();
```

Даний запит дозволяє отримати один запис щоденних записів за датою та ідентифікатором програми харчування з пов'язаними даними.

У фреймворку NestJS декоратори застосовуються у багатьох аспектах. Їх застосовують для визначення маршрутів, валідації даних і т.д. Під час проектування застосунку було розроблено власний декоратор `CurrentUser`, який інтегрується з механізмом обробки HTTP-запитів NestJS. Він дозволяє зручно

отримувати інформацію про поточного користувача з об'єкта запиту. Реалізація декоратора приведена нижче:

```
export const CurrentUser = createParamDecorator(
  (_data, ctx: ExecutionContext): any => {
    const request = ctx.switchToHttp().getRequest();
    if (_data) {
      return request.user[_data];
    }
    return request;
  },
);
```

Даний декоратор дозволяє легко отримувати дані про користувача в контролерах. Це зменшує кількість коду та знижує ризик виникнення помилок, пов'язаних з неправильним доступом до даних або маніпуляцією з ними. У майбутніх версіях застосунку можна додати додаткові перевірки прав доступу або інші валідаційні механізми до даного декоратора.

Забезпечення автентифікації було застосовано систему контролю доступу, що складається з охоронців (guard) JwtAuthGuard і RoleGuard і відповідних декораторів:

```
@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
```

JwtAuthGuard забезпечує аутентифікацію користувача, перевіряючи наявність і дійсність JWT-токена в запиті. JSON Web Token – це один з найпопулярніших способів аутентифікації та авторизації користувачів. Він дозволяє передавати зашифровану інформацію між клієнтом і сервером у вигляді компактного і безпечного токена. У даному проекті, якщо токен відсутній або недійсний, доступ до заданого маршруту буде заборонено. [20, 28]

RoleGuard забезпечує авторизацію, перевіряючи, чи має автентифікований користувач необхідну роль. Якщо роль користувача не відповідає необхідній – доступ до маршруту також буде заборонено. [28]

Для забезпечення узгодженості структури відповідей сервера було створено інтерсептор SerializeInterceptor, який виконує роль перетворення та серіалізації даних. [41]

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг інтерсептора:

```
export class SerializeInterceptor<T> implements NestInterceptor {
  constructor(private dto: ClassConstructor<T>) {}
  intercept(context: ExecutionContext, next: CallHandler): Observable<T> {
    return next.handle().pipe(
      map((data: any) => {
        if (data && data.result) {
          data.result = plainToClass(this.dto, data.result, {
            excludeExtraneousValues: true,
          });
        }
        return data;
      }),
    );
  }
}
```

З його допомогою вдалося реалізувати автоматичне перетворення об'єктів JavaScript у класи DTO (Data Transfer Object), які є структурованими контейнерами для передачі даних. Класи DTO також були використані для створення і оновлення даних. [41, 45]

Приклад класу для сутності HealthInfo:

```
export class HealthInfoCreateDto {
  @IsNotEmpty({ message: 'Age must be not empty' })
  @IsNumber({}, { message: 'Age must be a number' })
  @Min(18, { message: 'Age must be at least 18' })
  @Max(99, { message: 'Age must not exceed 99' })
  age: number;

  @IsNotEmpty({ message: 'Weight must not be empty' })
  @IsNumber({ maxDecimalPlaces: 3 }, { message: 'Weight must be a number with
maximum 3 decimal places' })
  @Min(4.999, { message: 'Weight must be at least 4.999' })
  @Max(999.999, { message: 'Weight must not exceed 999.999' })
  weight: number;

  @IsNotEmpty({ message: 'Height must not be empty' })
  @IsNumber({ maxDecimalPlaces: 3 }, { message: 'Height must be a number with
maximum 3 decimal places' })
  @Min(10.0, { message: 'Height must be at least 10.0' })
  @Max(300.0, { message: 'Height must not exceed 300.0' })
  height: number;

  @IsNotEmpty({ message: 'Gender must not be empty' })
  @IsEnum(Gender, { message: 'Invalid gender' })
  gender: Gender;

  @IsNotEmpty()
  user: Users;

  @IsOptional()
  @Min(0, { message: 'Height must be at least 10.0' })
  @Max(100, { message: 'Height must not exceed 300.0' })
  bodyfat: number | null;
}
```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

2.5. Реалізація клієнтської частини застосунку підбору індивідуальних програм харчування

Для того щоб розділити інтерфейс користувача на незалежні та повторно використовувані частини, у розробці було використано компоненти. Завдяки ним можна легко створювати модульну структуру інтерфейсу, у якій усі елементи будуть відповідати за власну функціональність, але при цьому будуть цілком придатними для взаємодії один з одним. Такий підхід дозволяє використовувати один і той самий тип компонентів на будь-якому рівні деталізації.

У застосунку було створено ряд компонентів, готових для багаторазового використання. Одним з таких є `FoodListItem`, що дозволяє виводити інформацію про харчові продукти. Через те що даних про продукти може бути дуже багато створення окремих блоків для кожного продукту стало б нелогічним. Тому було прийнято рішення розробити один єдиний компонент, який забезпечить виведення всіх даних належним чином. Це дозволило оптимізувати код та полегшити його підтримку:

```
<FlatList
  data={data}
  renderItem={({ item }) => (
    <FoodListItem
      MenuOptions={menuOptions}
      item={item}
      info={info}
      ell={ell}
      key={item.id}
      keyExtractor={(item) => item.id}
      onRemoveItem={handleRemoveItem}
      addToDaily={addToDaily}
    />
  )}
  contentContainerStyle={{ gap: 5 }}
/>
```

За допомогою елемента `FlatList` було легко відображено список елементів продуктів, де кожен рендериться за допомогою компонента `FoodListItem`.

Ще одним гарним прикладом повторного використання компонентів є модальне вікно (`Tooltip`) створене для виведення різних повідомлень користувачу. Приклад реалізації `AlertComponent`:

```

const AlertComponent = ({
  info,
  title = null,
  modalVisible,
  setModalVisible,
  showContinue = false,
  onContinue = () => {},
}) => {
  return (
    <Modal
      animationType="fade"
      transparent={true}
      visible={modalVisible}
      onRequestClose={() => setModalVisible(!modalVisible)}>
      <View style={styles.modalBackground}>
        <View style={styles.modalContent}>
          <View style={styles.header}>
            {title && <Text style={styles.title}>{title}</Text>}
            <TouchableOpacity onPress={() => setModalVisible(!modalVisible)}>
              <AntDesign
                name="close"
                size={fontSize.lg}
                color="black"
                style={styles.icon}
              />
            </TouchableOpacity>
          </View>
          <ScrollView contentContainerStyle={styles.infoContainer}>
            <Text style={styles.infoText}>{info}</Text>
          </ScrollView>
          {showContinue && (
            <View style={styles.buttonContainer}>
              <TouchableOpacity style={styles.button} onPress={onContinue}>
                <Text style={styles.buttonText}>Вийти</Text>
              </TouchableOpacity>
            </View>
          )}
        </View>
      </View>
    </Modal>
  );
};
export default AlertComponent;

```

Це дозволило замінити повторне створення подібних компонентів на використання наступного фрагменту коду:

```

<AlertComponent
  modalVisible={modalVisible}
  setModalVisible={setModalVisible}
  info={bodyFatInfo}
  title={"Відсоток жиру"}
/>

```

Крім компонентів у проекті також було використано інструмент відомий як хуки. Вони створені для керування станом компонентів та їх життєвим циклом. У проекті було використано не лише вбудовані хуки, а і такі як `useState`, `useEffect`, `useCallback`, `useContext` тощо, але і створено свої. [7]

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Таким є хук `useAxios`, створений для спрощення взаємодії з HTTP-запитами за допомогою бібліотеки `Axios`. Даний хук дозволяє у легкий спосіб здійснювати запити до сервера, зберігати дані, відстежувати стан завантаження та помилок. Код розміщено в Додатку Г.

Щоб забезпечити можливість збереження даних користувача для повторного входу було використано пакет `@react-native-async-storage/async-storage` і реалізовано ряд допоміжних функцій. [23]

Для збереження даних у локальному сховищі було реалізовано функцію `storeData`:

```
export const storeData = async (key, value) => {
  try {
    const jsonValue = JSON.stringify(value);
    await AsyncStorage.setItem(key, jsonValue);
  } catch (e) {
    console.error(e);
  }
};
```

Вона отримує ключ і значення, перетворює значення у JSON-рядок і зберігає їх у локальному сховищі. Для отримання даних з локального сховища було забезпечено функцію `getData`, яка отримує ключ, зчитує відповідне значення з локального сховища, перетворює і повертає його.

```
export const getData = async (key) => {
  try {
    const jsonValue = await AsyncStorage.getItem(key);
    return jsonValue != null ? JSON.parse(jsonValue) : null;
  } catch (e) {
    console.error(e);
  }
};
```

І для видалення значення з локального сховища було створено функцію `removeValue`:

```
export const removeValue = async (key) => {
  try {
    await AsyncStorage.removeItem(key);
  } catch (e) {
    console.error(e);
  }
};
```

Важливим елементом майже кожної системи є аутентифікація користувачів та контроль їх доступу до функцій програми. Даний процес у застосунку було забезпечено за допомогою популярної платформи з хмарними

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

сервісами Firebase. Використання цього сервісу дозволило інтегрувати метод автентифікації за допомогою електронної пошти та пароллю. [30]

Приклад використання Firebase Authentication для авторизації користувача:

```
const onSignInPressed = async (data) => {
  try {
    const auth = getAuth();
    const res = await signInWithEmailAndPassword(
      auth,
      data?.email,
      data?.password
    );
    setUserID(res?.user?.uid);

    if (res?.user?.stsTokenManager?.accessToken) {
      dispatch(setToken(res?.user?.stsTokenManager?.accessToken));
    }
  } catch (error) {
    setError(error);
  }
};
```

Для переходу між різними екранами додатку було використано бібліотеку React Navigation. Даний інструмент спростив проектування навігаційної логіки застосунку та дозволив реалізувати просту передачу даних між сторінками. Для забезпечення кращого користувацького досвіду було використано стекову та табову навігацію для різних сценаріїв роботи. [43]

Приклад реалізованого стеку навігації для роботи з профілем користувача:

```
const ProfileNavigatorStack = () => {
  const navigation = useNavigation();
  useEffect(
    React.useCallback(() => {
      return () =>
        navigation.reset({
          index: 0,
          routes: [{ name: "ProfileScreen" }],
        });
    }, [])
  );
  return (
    <MealProgramsProvider>
      <Stack.Navigator>
        <Stack.Screen
          name="ProfileScreen"
          component={ProfileScreen}
          options={{ headerShown: false }}
        />
        <Stack.Screen
          name="EditProfileScreen"
          component={EditProfileScreen}
          options={{ headerShown: false }}
        />
        <Stack.Screen
          name="ViewAllProgramsScreen"

```

```

        component={ViewAllProgramsScreen}
        options={{ headerShown: false }}
      />
      <Stack.Screen
        name="ViewPlanInfoScreen"
        component={ViewPlanInfoScreen}
        options={{ headerShown: false }}
      />
      <Stack.Screen
        name="ViewPlanInfo"
        component={ViewPlanInfo}
        options={{ headerShown: false }}
      />
    </Stack.Navigator>
  </MealProgramsProvider>
);
};

```

`export default ProfileNavigatorStack;`

У цьому фрагменті визначено різні екрани, такі як "ProfileScreen", "EditProfileScreen" тощо, які відображаються при навігації користувача. Кожен екран має свої власні налаштування, наприклад, можливість приховати заголовок. Для скидання сторінок до початкової було використано функцію `useFocusEffect` яка є частиною бібліотеки `React Navigation` і спрацьовує тільки тоді коли екран у фокусі або поза ним. В даному випадку, коли екран втрачає фокусування викликається `navigation.reset`, що дозволяє забезпечити повернення користувача з будь якого екрану стеку на початок. [43]

Висновки до другого розділу

У другому розділі дипломного проекту було виконано прототипування, проектування та реалізацію мобільного застосунку. Проведено детальний аналіз варіантів роботи із системи. За допомогою діаграми варіантів використання візуалізовано основні сценарії взаємодії користувачів із застосунком. Це допомогло зрозуміти всі можливі способи застосування системи, виявити потенційні проблеми та врахувати їх при розробці функціоналу.

Для визначення структури системи, основних модулів та їх взаємодії, було розроблено діаграми компонентів серверної та клієнтської частин.

Окрім цього було розглянуто можливі алгоритми роботи системи. Досліджено ряд формул для обрахунку КБЖУ. Вирішено використати дві з них:

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

формули Кетча-МакАрдила та Маффіна-Джеора. Послідовність обрахунків та інші алгоритми було окреслено за допомогою діаграм.

Проектування бази даних завершилося створенням 17 таблиць з різними типами зв'язків, а також двох тригерів і функцій для автоматичного оновлення даних після виконання запитів.

Заключними етапами другого розділу стали проектування серверної та клієнтської частин. Було реалізовано ряд важливих функцій, зв'язків та компонентів для забезпечення коректної роботи системи. Деякі з них було представлено та пояснено.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Порядок встановлення та налаштування параметрів системи

Налаштування параметрів є важливим етапом забезпечення ефективної та довготривалої роботи будь-якої інформаційної системи. Щоб досягти успішного розгортання мобільного застосунку та серверної частини, перш за все, необхідно визначити вимоги до апаратно-програмного забезпечення. Це може бути опис необхідної платформи, інтерфейсів комунікації, операційних вимог тощо. Важливим інструментом для візуалізації архітектури системи та зв'язків між її компонентами є діаграма розгортання. Вона допомагає визначитись із необхідними ресурсами та конфігураціями для успішного функціонування програмного забезпечення. [3]

На рисунку 3.1 зображено діаграму розгортання системи, що розроблялась:

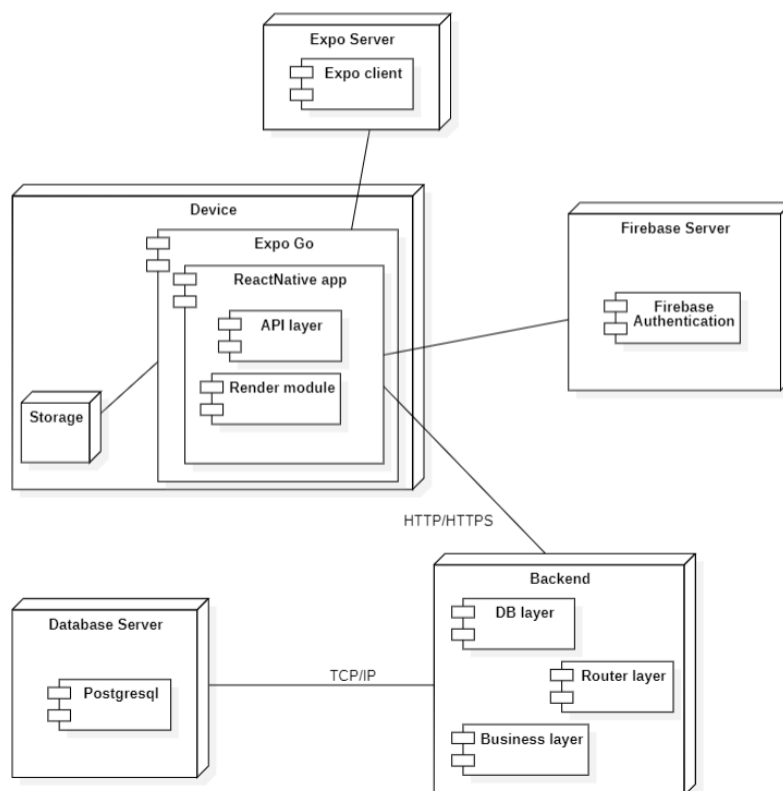


Рисунок 3.1. – Діаграма розгортання

Expo Server та Expo client – використовуються для розробки та розгортання мобільних додатків на Expo Go. На пристрої запускається сам Expo Go, і він

відтворює потрібний застосунок. React Native додаток взаємодіє з Firebase Server для забезпечення автентифікації користувачів. Також додаток пов'язаний із Backend сервером, який забезпечує роботу з базою даних. Для комунікації між мобільним додатком та Backend сервером, а саме виконання запитів (GET, POST, PUT, DELETE) та передачі даних між клієнтом і сервером використовується протокол HTTP/HTTPS.

3.2. Опис структури інтерфейсу та роботи додатку

Інтерфейс користувача є одним із основних елементів будь-якого мобільного застосунку, адже завдяки йому відбувається уся взаємодія із системою. У даній роботі було акцентовано увагу на принципах швидкого відгуку та інтуїтивності застосунку, тобто всі команди та запити обробляються з мінімальним часом очікування, наскільки це можливо.

Початок роботи із застосунком починається з відкриття програми. Якщо у локальному сховищі немає збережених даних користувача, першим, що з'явиться на екрані – буде вікно входу. Якщо раніше було створено профіль – користувач може вказати особисті дані і перейти до основного функціоналу програми (Рис. 3.2). В іншому випадку – він може перейти до створення профілю. (Рис. 3.3).

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Вхід

Email

Пароль

☐ Запам'ятати мене

Увійти

[Забули пароль?](#)
 Вперше тут? [Зареєструватися](#)

Рисунок 3.2. – Вікно входу у систему

+
Реєстрація

Email

 Email є обов'язковим

Ім'я

 Ім'я є обов'язковим

Прізвище

 Прізвище є обов'язковим

Пароль

 Пароль є обов'язковим

Продовжити

Вже маєте акаунт? [Увійти](#)

Рисунок 3.3. – Вікно реєстрації нового користувача

Мобільний додаток має вбудовану систему помилок, яка інформує користувача про будь-які проблеми, неправильні дані тощо. На рисунку 3.4 представлено неправильно заповнені поля форми.

Рисунок 3.4. – Поля форми з помилкою

Повідомлення може змінюватися в залежності від типу поля та його змісту. Однак, не завжди є можливість відобразити помилку у такий спосіб. Для забезпечення гнучкості виведення різних сповіщень та помилок в додатку було реалізовано компонент спливаючого повідомлення. Він дозволяє динамічно змінювати повідомлення в залежності від ситуації та контексту (Рис. 3.5).

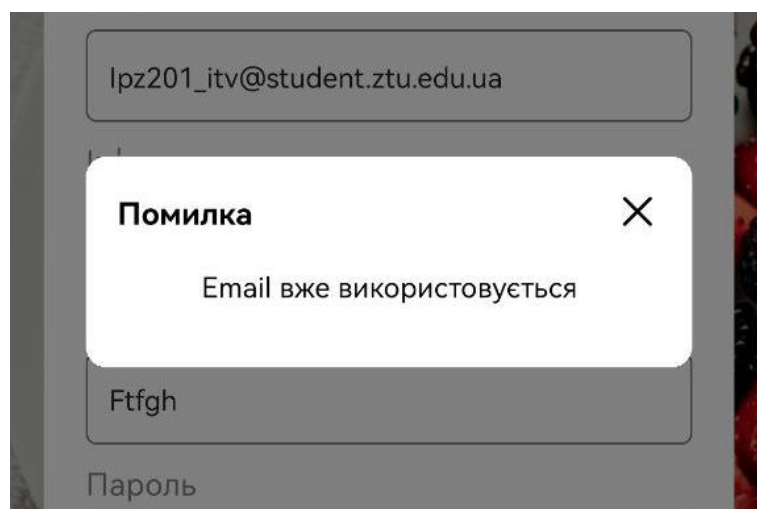


Рисунок 3.5 – Tooltip (модальне вікно)

Для забезпечення зручного користувацького досвіду було спроектовано покрокове налаштування програми харчування. Це дозволило розділити необхідні форми для збору даних на окремі розділи і забезпечити поступовий та логічний процес введення інформації. Такий підхід спростив взаємодію користувача із системою, зробивши її більш зрозумілою та менш обтяжливою. Кожен крок розміщений на окремому екрані. Деякі кроки представлені на рисунках 3.6–3.8.

Створення програми
Фізичні показники

Вік
22

Вага, кг
48

Зріст, см
168

Відсоток жиру (не обов'язково)
Відсоток жиру (не обов'язково) ⓘ

Стать
Жінка ▼

Продовжити

Рисунок 3.6 – Вікно налаштування фізичних показників

← Фізичні показники

Створення програми

ІМТ: 17.01
Рекомендована вага: 52.21 – 70.28 кг

Збільшення ваги, 1/2 кг на тиждень

Швидке збільшення ваги, 1 кг на тиждень

Цільова вага, кг

50

Вага не відповідає рекомендації

Продовжити

Рисунок 3.7 – Вікно вибору цілі користувача

← Фізична активність

Створення програми

Дієта

Збалансована

Білки % Жири % Вуглеводи %

30 30 40

Продовжити

Рисунок 3.8 – Вікно вибору дієти

За потреби користувач може повернутися до попередніх кроків і змінити параметри, для цього додано панель навігації у верхній частині екрану (Рис. 3.9).

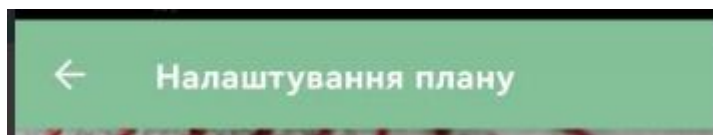


Рисунок 3.9. – Кнопка навігації між вікнами

Розрахована програма відображається на окремому екрані. На ній представлено розраховані параметри КБЖВ відповідно до налаштувань користувача (Рис. 3.10).



Рисунок 3.10. – Вікно підтвердження програми харчування

Після створення програми харчування автоматично створюється щоденний план і користувача перенаправляє на головну сторінку. Дана сторінка встановлена за замовчуванням. На ній представлено графік денного прогресу споживання КБЖВ, деяка особиста інформації та засоби для перемикання між прийомами їжі.

Наповнення сторінки може дещо відрізнятись. Це залежить від того, чи має користувач активну програму харчування, та чи створений план на день. Приклади можливих варіантів головної сторінки наведено на рисунках 3.11–3.13



Рисунок 3.11 – Головний екран, коли у користувача немає активної програми

Вітаємо, Тетяна



Вкажіть параметри щоб оновити програму харчування

Вага, кг

53

Оновити

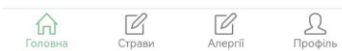


Рисунок 3.12 – Головний екран, коли у користувача є активна програма і це перший вхід у систему поточного дня

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

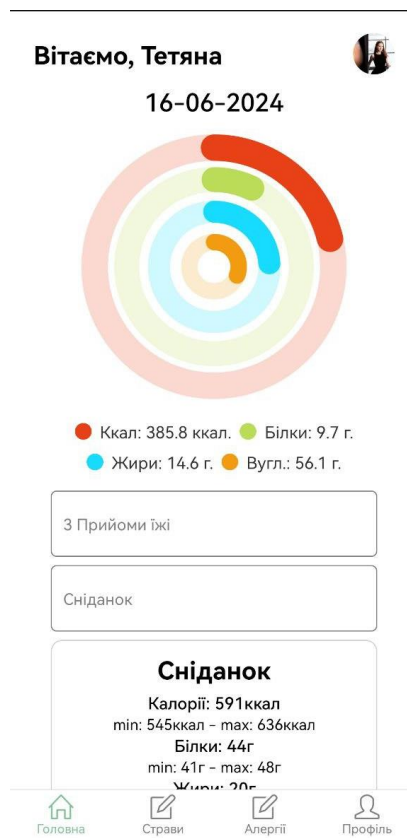


Рисунок 3.13 – Головний екран, коли у користувача є активна програма і користувач вказав оновлену вагу

За допомогою кнопок нижньої навігації авторизовані користувачі можуть переміщатися між екранами (Рис. 3.14).



Рисунок 3.14. – Нижня панель навігації

На екрані «Страви» розміщено усі рецепти, які наявні у базі даних і які не місять алергенів серед інгредієнтів (Рис. 3.15). За допомогою пошуку та фільтрації користувач може віднайти бажані страви, та при бажанні додати їх до улюблених (Рис. 3.16). Усі улюблені страви користувача винесені на окремий екран, який розміщений в одному стеці з екраном «Страви».



Рисунок 3.15. – Екран «Страви»

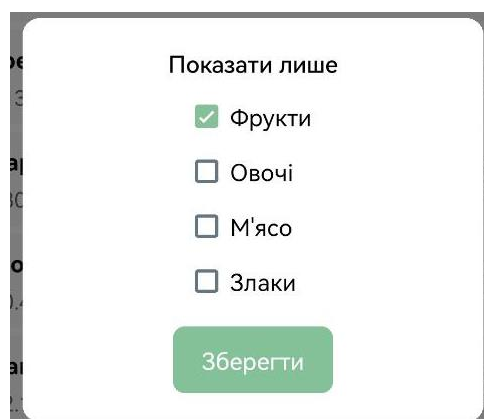


Рисунок 3.16. – Модальне вікно фільтрації

Для встановлення алергій на певні продукти користувачу необхідно перейти на сторінку «Алергії», де розміщені усі інгредієнти наявні у базі даних. Можливість вказувати алергії дозволяє убезпечити користувача від страв, які містять небажаний інгредієнт.

Додавання страв до щоденного плану відбувається на окремому екрані, який має незначні відмінності від екрану страви (Рис. 3.17).

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

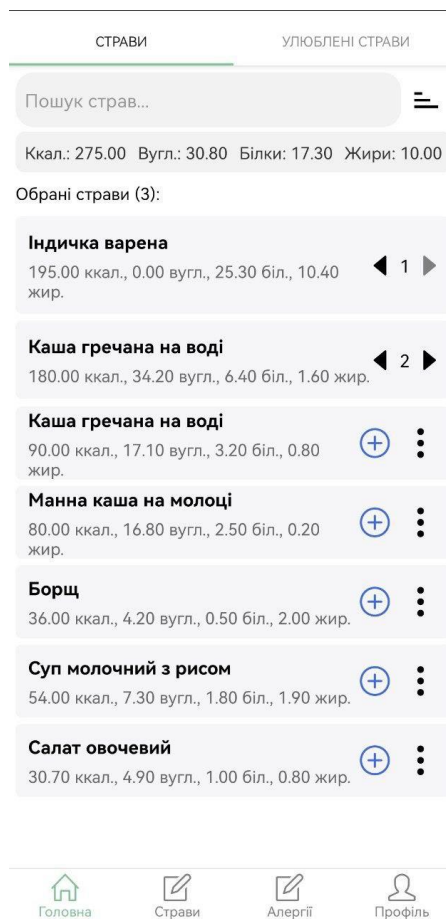


Рисунок 3.17. – Вікно додавання страв до щоденного плану

Коли користувач додає страву до певного прийому їжі, кількість КБЖВ зменшується і відбувається фільтрація доступних страв, які не перевищують визначену норму. У будь-який момент часу користувач може змінити свій вибір.

За потреби користувач може переглянути рецепт та детальну інформацію про страву, натиснувши на блок з потрібною назвою. На сторінці що відкриється, можна буде побачити фото, текстовий опис та відео приготування бажаного продукту (Рис. 3.18).



Гречана каша

КБЖВ на 100 грам:



Опис

Порцій: 1

Гречана каша може бути самостійною стравою (часто вживається з молоком, вершковим маслом, цукром), також її використовують як гарнір до риби та м'яса.



Рисунок 3.18. – Екран з описом страви

На вкладці «Профіль» розміщені основні елементи для керування особистими даними та програмами харчування, а також інтерактивний графік зміни ваги для поточної програми харчування (Рис. 3.19). Даний графік став ефективним інструментом для візуалізації зміни ваги користувача протягом усього часу проходження програми. Якщо користувач не має програми харчування зі статусом «Триває», то він і вкладка «Програма харчування» будуть відсутні на екрані. Усе це дозволяє зручно управляти особистою інформацією та слідкувати за прогресом у досягненні своїх цілей. Також на цьому екрані розміщена кнопка завершення сеансу.

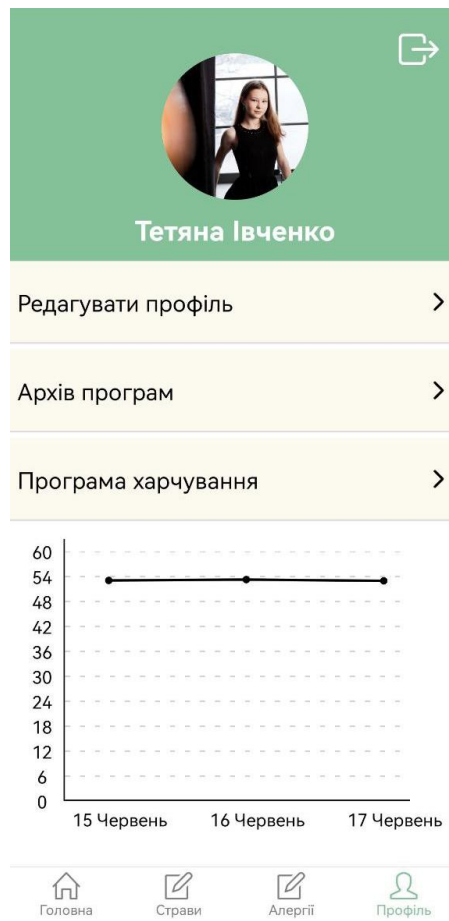


Рисунок 3.19. – Вкладка «Профіль»

Розділ «Програма харчування» містить інформацію про тривалість, вираховану калорійність тощо (Рис. 3.20). Саме тут розміщена кнопка «Видалити» яка дозволяє завершити актуальну програму. Однак це завершення не є правдивим, оскільки для програми лише оновлюється статус.

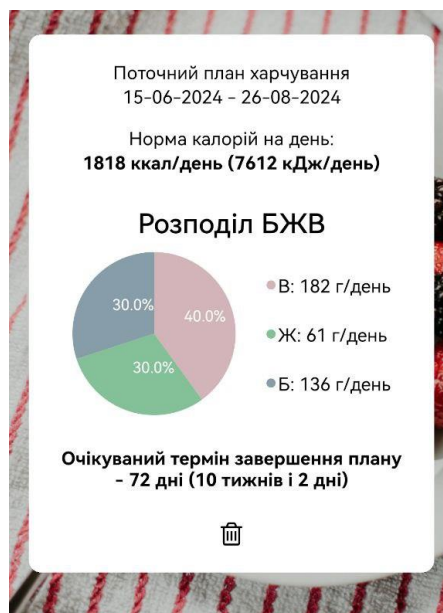


Рисунок 3.20. – Фрагмент розділу «Програма харчування»

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

Архів програм харчування дозволяє переглядати усі програми користувача (Рис. 3.21). Для пошуку потрібної програми можна застосувати фільтрацію та сортування за різними параметрами.

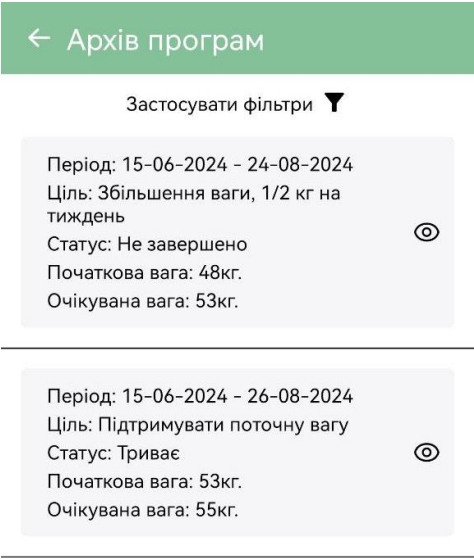


Рисунок 3.21. – Архів програм харчування

Вікно редагування профілю призначене для зміни деякої особистої інформації користувача (Рис. 3.22). З метою забезпечення точності та релевантності даних, фізичні показники оновлюються лише при створенні нової програми харчування, окрім ваги, яка зміна якої виконується щоразу при створенні плану на день. Таке рішення дозволило уникнути ситуацій, коли дані користувача застаріють або будуть змінені на недостовірні.

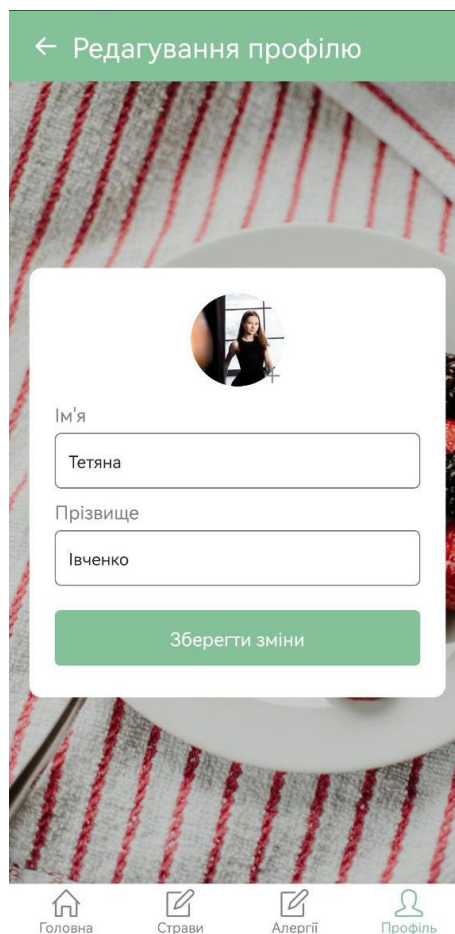


Рисунок 3.22. – Вікно редагування профілю

Мобільний застосунок, розроблений в рамках цього проекту, також демонструє гарний рівень продуктивності на різних розмірах екрану (Рис. 3.23) та операційній системі iOS (Рис. 3.24). Елементи інтерфейсу на різних операційних системах майже не відрізняються, це забезпечує єдиний користувацький досвід незалежно від пристрою.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

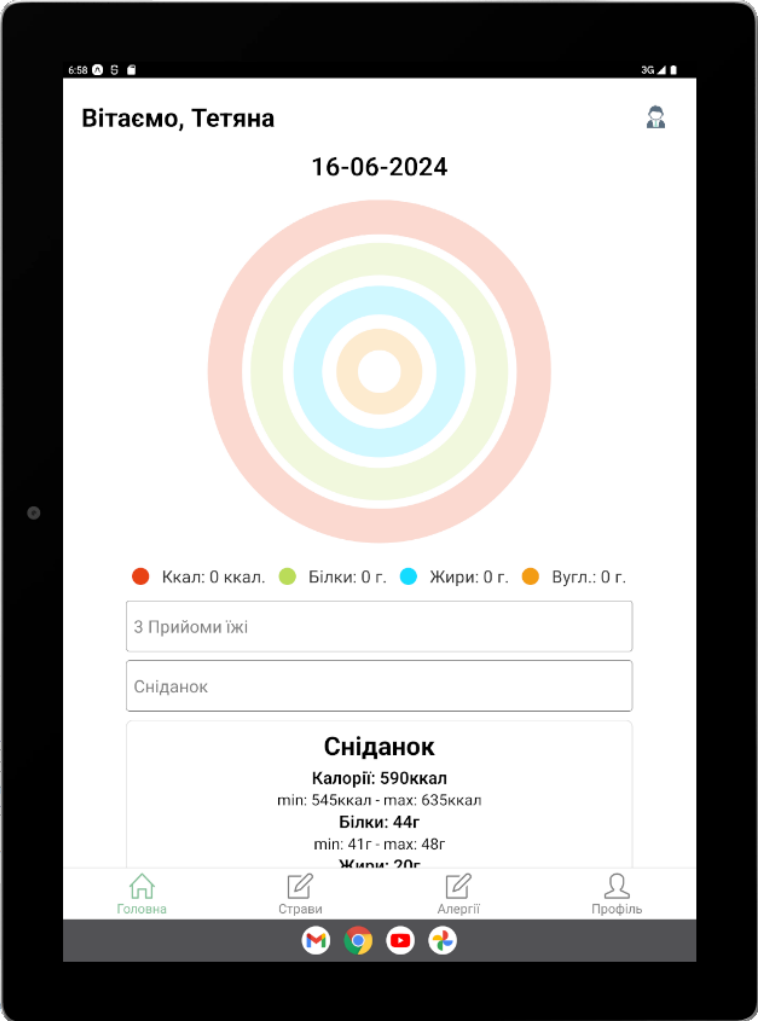


Рисунок 3.23. – Вікно «Головна» на великому екрані

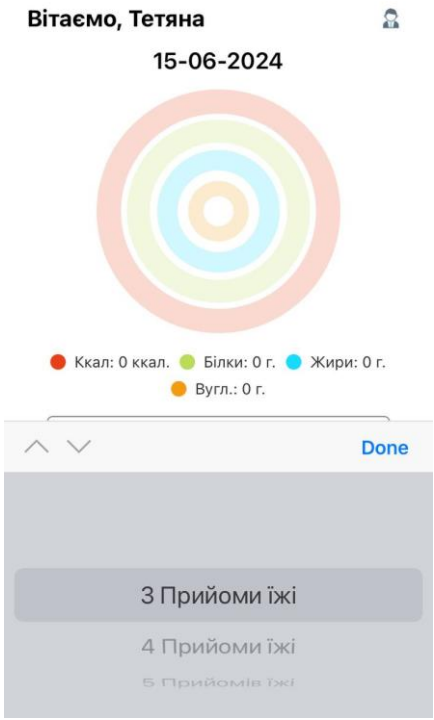


Рисунок 3.24. – Вікно застосунку запущеного на операційній системі iOS

3.3. Тестування програмного продукту

Перед тим як програмний продукт стане доступним для користувачів, його необхідно протестувати, щоб переконатися, що він працює належним чином і відповідає всім вимогам. Цей процес відомий як тестування, і його метою є виявлення дефектів, помилок та проблем на різних етапах розробки.

Для тестування проекту було використано різні підходи та інструменти. Головним засобом тестування став застосунок Ехро Го, який дозволив проводити швидко та зручне тестування мобільного застосунку на власному пристрої. Оскільки сервер розробки Ехро був запущений протягом усього часу роботи і система підтримує гаряче перезавантаження, кожна зміна миттєво надходила у пристрій. Даний вид тестування називають ручним, або мануальним.

З ціллю полегшення процесу проведення тестів було розроблено низку чеклістів, що сприяли ефективному плануванню, перевірці та виконанню тестування, а також полегшили відстеження прогресу.

Таблиця 3.1

Чекліст тестування застосунку

№	Тестовий сценарій	Результат	Примітки
1.	Реєстрація нового користувача	Пройдено	Успішна операція
2.	Збереження даних при переході між екранами	Пройдено	Працює коректно
3.	Вхід у систему	Пройдено	Успішна операція
4.	Відображення кнопки створення нового плану	Пройдено	Якщо користувач не має активного плану
5.	Відображення вікна введення щоденної інформації про вагу	Пройдено	Дані у поле мають підтягуватись автоматично, повинні запускатись обрахунки, оновлення даних в healthInfo
6.	Перегляд щоденного раціону	Пройдено	Повинен заповнюватись графік, відображаються лише доступні дані

Продовження таблиці 3.1

7.	Перевірка коректності відображення даних у профілі користувача	Пройдено	Дані відображені коректно
8.	Перевірка функціоналу відновлення паролю	Пройдено	Успішна операція
9.	Додавання і видалення страв в улюблені	Пройдено	Успішна операція
10.	Додавання і видалення алергій	Пройдено	Успішна операція

Оскільки в основі програмного продукту лежить клієнт-серверна архітектура, очевидно, що не можна обійтись без тестування API. Для даного тестування було використано спеціальне програмне забезпечення, Postman, яке дозволило реалізувати легкий процес відправки запитів та отримання відповідей (Рис. 3.24). [40]

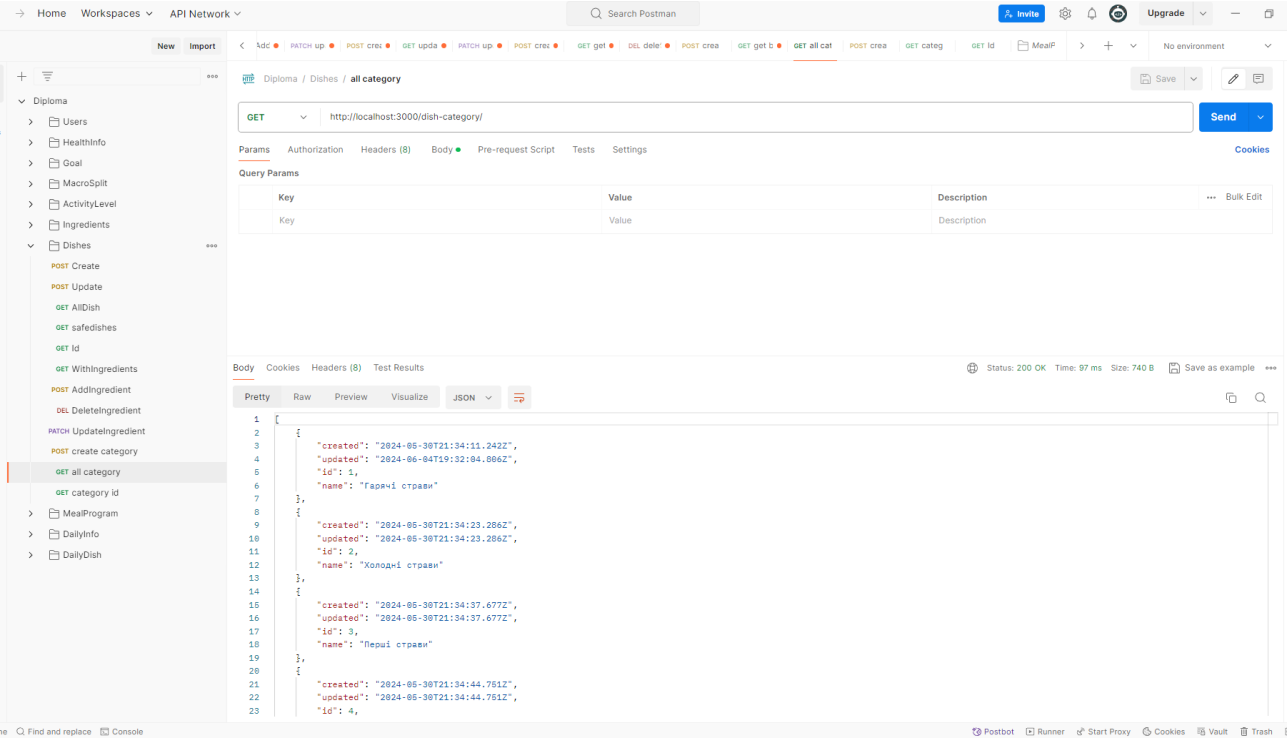


Рисунок 3.24. – Вікно Postman із виконаним запитом для усіх отримання категорій страв

API тестування є частиною інтеграційного тестування, що дозволяє виявляти помилки у взаємодії між різними модулями системи або між окремими системами. Цей вид тестування став особливо корисним під час проектування контролерів серверної частини.

Проведення регресійного тестування значно ускладнилось зі збільшенням кількості програмного коду та функціоналу. Для спрощення перевірки, чи нові зміни не вплинули на існуючий функціонал, та підвищення ефективності тестування, було використано популярну тестову бібліотеку для JavaScript - Jest. Вона надає можливість легко створювати і виконувати тести, забезпечуючи швидкий та надійний зворотний зв'язок щодо змін у коді. З її допомогою було створено сценарії для автоматичного запуску тестів і перевірки функціоналу деяких компонентів системи. Нижче наведено фрагмент коду тестування компоненту InputBoxForm за допомогою Jest та React Test Renderer (весь код наведено в Додатку Д):

```
it("Перемикає видимість пароля", () => {
  const component = renderer.create(
    <InputBoxForm name="password" placeholder="Password" />
  );
  const instance = component.root;
  const input = instance.findByType(TextInput);
  const toggleButton = instance.findByProps({ testID: "passwordToggle" });
  toggleButton.props.onPress();
  expect(input.props.secureTextEntry).toBe(false);
  toggleButton.props.onPress();
  expect(input.props.secureTextEntry).toBe(true);
});
```

Оскільки компонент InputBoxForm був створений як такий, що придатний до перевикористання, час від часу у нього доводилось вносити зміни з метою розширення його застосування. Щоб уникнути постійного повтору тестування одного і того ж функціоналу було створено тест, що перевіряє функціональність перемикання видимості пароля, зокрема, зміну значення secureTextEntry у компоненті TextInput.

Наступний приклад тесту був застосований вже для SubmitFormButton, а саме перевірки коректного застосування стилів до кнопки.

```
it("Коректно застосовує стилі тексту", () => {
  const component = renderer.create(
    <SubmitFormButton onPress={mockOnPress} text="Submit" />
  );
  const textStyles = component.root.findByType(Text).props.style;

  expect(textStyles).toEqual(
    expect.objectContaining({ fontSize: 26.5, color: "#ffff" })
  );
});
```

У результаті усі проведені тести були успішно пройдені (Рис. 3.25).

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Test Suites: 4 passed, 4 total
Tests:       11 passed, 11 total
Snapshots:   3 obsolete, 3 written, 2 passed, 5 total
Time:        4.32 s
Ran all test suites.

```

Рисунок 3.25. – Звіт виконання тестів

Завдяки пройденим тестам вдалося виявити дефекти у роботі системи і виправити їх ще на ранніх етапах розробки. У фінальному результаті вдалося реалізувати застосунок який відповідає усім поставленим вимогам.

Висновки до третього розділу

Третій розділ дипломного проектування був присвячений реалізації та тестуванню ПЗ. Було представлено діаграму розгортання застосунку та описано ключові елементи, зокрема Expo Server, Expo Client, React Native додаток і його взаємодію з Firebase Server та Backend.

Завершення розробки мобільного застосунку дало змогу представити інтерфейс і описати структуру екранів програми.

Для забезпечення високої якості програмного продукту було виконано мануальне тестування, інтеграційне тестування, створено ряд автоматичних тестів. Проведене тестування дозволило виявити вразливі місця програми та оперативно їх виправити.

ВИСНОВКИ

В процесі написання дипломної роботи було виконано аналіз теоретичних та функціональних вимог до програмного забезпечення, що дозволило окреслити ключові ідеї та етапи реалізації програмного продукту. Завдяки аналізу сучасних тенденцій було визначено, що найбільш зручним форматом для реалізації поставлених цілей є мобільний додаток, який дозволить забезпечити якісний доступ користувачів до сервісу.

Важливим етапом на шляху до визначення структури майбутнього програмного забезпечення став пошук та порівняльний аналіз аналогічних розробок. Було виділено найбільш важливі аспекти, необхідні для створення якісного та конкурентоспроможного застосунку.

Для забезпечення ефективності та масштабованості застосунку було обрано клієнт-серверну архітектуру. Для клієнтської частини використано React Native Expo. Для серверної частини Nest.js та TypeORM. В якості бази даних обрано PostgreSQL.

Були визначені основні сценарії використання системи, описано роботу ключових алгоритмів та обчислень. Виконано побудову та проектування усіх складових частин структури програмного продукту. Під час цього етапу особлива увага була звернута на зручність інтерфейсу, його інтуїтивність та відповідність вимогам.

По завершенню виконання основних етапів розробки, було представлено інтерфейс мобільного додатку, виконано тестування програмного комплексу з використанням різних інструментів та підходів, виправлено потенційні помилки.

Результатом завершення усіх завдань дипломної роботи став готовий до використання мобільний застосунок для підбору індивідуальних програм харчування з урахуванням особистих потреб користувачів.

Додаток може бути використаний як для особистих потреб, так і для впровадження в галузі медицини, спорту або фітнесу. Ці можливості відкривають широкі перспективи для подальшого розвитку програмного забезпечення.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

У майбутніх версіях програмного продукту є можливість реалізувати нові функції, які значно розширять його потенціал. Наприклад, можна вдосконалити алгоритми аналізу даних, забезпечити підтримку інших мов або інтеграцію зі сторонніми пристроями. Чудовим рішенням може стати впровадження алгоритму автоматичного підбору страв та додавання інших ролей користувачів, як от дієтолога, для можливості додавання нових страв або дієт у БД. Завдяки якісно підібраний та спроектованій архітектурі це можна зробити у легкий спосіб.

Найважчим у даній роботі виявилось використання нових технологій та складних інструментів, які були обрані для реалізації програмного продукту та вимагали багато часу для їх вивчення. Зокрема, використання React Native Expo, Nest.js та TypeORM виявилось викликом через їхню складність і обсяг можливостей, а також відсутність попереднього досвіду розробки. Не легкими видались пошук та створення алгоритмів обчислення кількості калорій, білків, жирів та вуглеводів (КБЖВ) для користувачів. Ці алгоритми потребували ретельного дослідження дієтологічних аспектів та правильного математичного підходу для точних розрахунків, що забезпечують коректні результати в програмі.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Все про мобільні застосунки [Електронний ресурс] // LinkedIn – Режим доступу до ресурсу: <https://www.linkedin.com/pulse/все-про-мобільні-застосунки-цифровий-світ-покупця-uployal-3eh8e/>.
2. Детальне пояснення розрахунку страв [Електронний ресурс] – Режим доступу до ресурсу: <https://calc.tablycjakalorijnosti.com.ua/explanation>.
3. Дєдх Т. А. Методи розробки мобільних додатків / Т. А. Дєдх – Одеса: Стан, досягнення та перспективи інформаційних систем і технологій: матеріали XXIII Всеукр. наук.-техн. конф. молодих вчених, аспірантів та студентів, 2023. – С. 231–233.
4. Діаграма розгортання: Підручник з UML [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/uk/deployment-diagram-uml-example.html>.
5. Здорове та оздоровче харчування осіб, які займаються фітнесом / О. І. Циганенко, І. М. Хоменко, О. В. Маслова, Я. В. Першегуба, Т. О. Терещенко, Н. В. Склярєва, Т. В. Коломієць. – Київ: Щек, 2021. – 104–106 с.
6. Каграманова Ю. Як будувати UML-діаграми [Електронний ресурс] / Юлія Каграманова. – 2022. – Режим доступу до ресурсу: <https://dou.ua/forums/topic/40575/>.
7. Кравчук Н. М. Проблеми культури харчування сучасного українця [Електронний ресурс] / Н. М. Кравчук – Режим доступу до ресурсу: <https://dspace.nuft.edu.ua/server/api/core/bitstreams/e80970b0-b90e-4dd8-8b09-a022f1bf02a4/content>.
8. Ознайомлення з Хуками [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.legacy.reactjs.org/docs/hooks-intro.html>.
9. Ічанська Н. В. Основні аспекти створення мобільних додатків та вибір інструментів їх розробки / Н. В. Ічанська, С. І. Улько // Системи управління, навігації та зв'язку. Збірник наукових праць. / Н. В. Ічанська, С. І. Улько. – Полтава, 2020. – С. 74–78.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

10. Основні характеристики та принципи роботи з Flutter [Електронний ресурс] // FoxmindEd. – 2024. – Режим доступу до ресурсу: <https://foxminded.ua/flutter-shcho-tse/>.
11. Павлоцька Л. Ф., Дуденко Н. В., Євлаш В. В. Фізіологія харчування / Підручник. – Харків: ХДУХТ, Світ книг, 2017. – С. 162–164.
12. Посібник зі зв'язків між таблицями [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/topic/40575/>.
13. Принципи сучасного раціонального харчування [Електронний ресурс] – Режим доступу до ресурсу: <https://www.bsmu.edu.ua/blog/2090-zdorove-harchuvanja/>.
14. Різниця між гібридними та нативними мобільними додатками [Електронний ресурс] – Режим доступу до ресурсу: <https://wezom.com.ua/ua/blog/chem-otlichajutsja-nativnoe-i-gibridnoe-mobilnye-prilozhenija>.
15. Розробка мобільних додатків від А до Я [Електронний ресурс] – Режим доступу до ресурсу: <https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatktiv-vid-a-do-ja-povnij-gajd/>.
16. Тесування React Native додатків [Електронний ресурс] – Режим доступу до ресурсу: <https://jestjs.io/uk/docs/tutorial-react-native>.
17. Типи мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/>.
18. Чим відрізняються мови програмування? [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/vidminnosti-mov-prohramuvannia/>.
19. Що таке фреймворк [Електронний ресурс] – Режим доступу до ресурсу: <https://brainlab.com.ua/uk/blog-uk/shho-take-frejmwork-poyasnyuyemo-prostymy-slovamy>.
20. Як використовувати JSON Web Tokens (JWT) для автентифікації [Електронний ресурс] // DevZone – Режим доступу до ресурсу: <https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi>.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

21. Як правильно розрахувати свою норму калорій [Електронний ресурс] // Медичний центр ОН Клінік – Режим доступу до ресурсу: <https://onclinic.ua/blog/shcho-take-kaloriyi-ta-yak-pravylny-yikh-rakhuvaty>.
22. 2/3 усіх хвороб в Україні є наслідком неправильного харчування [Електронний ресурс] // <https://moz.gov.ua/article/news/23-usih-hvorob-v-ukraini-e-naslidkom-nepravilnogo-harchuvannja>.
23. AsyncStorage [Електронний ресурс] // React Native. – Режим доступу до ресурсу: <https://reactnative.dev/docs/asyncstorage>
24. Borozenets M. React native init vs expo 2022: what are the differences? [Електронний ресурс] / Michael Borozenets // Fulcrum. – Режим доступу до ресурсу: <https://fulcrum.rocks/blog/react-native-init-vs-expo>.
25. Cardiovascular mortality attributable to dietary risk factors in 51 countries in the WHO European Region from 1990 to 2016: a systematic analysis of the Global Burden of Disease Study [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6325999/>.
26. Investigating the Healthiness of Internet-Sourced Recipes: Implications for Meal Planning and Recommender Systems [Електронний ресурс] – Режим доступу до ресурсу: <https://dl.acm.org/doi/10.1145/3038912.3052573>.
27. Use of controllers [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/controllers>.
28. Documentation | NestJS [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/security/authorization>.
29. Fast refresh [Електронний ресурс] // React Native. – Режим доступу до ресурсу: <https://reactnative.dev/docs/fast-refresh>.
30. Firebase authentication [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/auth>.
31. Global diet and health: old questions, fresh evidence, and new horizons [Електронний ресурс] – Режим доступу до ресурсу: [https://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(19\)30500-8/fulltext](https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(19)30500-8/fulltext).

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

32. Chris Griffith. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova / Chris Griffith., 2017.
33. Guards [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.nestjs.com/guards>.
34. Hariharan, Rohit, et al. The dietary inflammatory index, obesity, type 2 diabetes, and cardiovascular risk factors and diseases. // Obesity Reviews 23.1 // 2022: С. – 1–5.
35. Mastering Data Management: Exploring the Repository Pattern in NestJS [Электронный ресурс] – Режим доступа до ресурсу: <https://blog.stackademic.com/mastering-data-management-exploring-the-repository-pattern-in-nestjs-1f297942cab5>.
36. Mobile network subscriptions worldwide 2028 [Электронный ресурс] – Режим доступа до ресурсу: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>.
37. Mobile OS market share worldwide 2009-2024 [Электронный ресурс] – Режим доступа до ресурсу: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>.
38. Overview of trigger behavior. PostgreSQL Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://www.postgresql.org/docs/current/trigger-definition.html>.
39. Select using Query Builder [Электронный ресурс] – Режим доступа до ресурсу: <https://orkhan.gitbook.io/typeorm/docs/select-query-builder>.
40. Send API requests and get response data in postman. Postman Learning Center [Электронный ресурс] – Режим доступа до ресурсу: <https://learning.postman.com/docs/sending-requests/requests/>.
41. Serialization [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.nestjs.com/techniques/serialization>.
42. Start developing. Expo Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.expo.dev/get-started/start-developing/>.

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

- 43.Tab navigation [Электронный ресурс] – Режим доступа до ресурсу:
<https://reactnavigation.org/docs/tab-based-navigation>.
- 44.TypeORM [Электронный ресурс] – Режим доступа до ресурсу:
<https://typeorm.io/>.
- 45.Validation [Электронный ресурс] – Режим доступа до ресурсу:
<https://docs.nestjs.com/techniques/validation>.
- 46.What is Xamarin? [Электронный ресурс] – Режим доступа до ресурсу:
<https://learn.microsoft.com/en-gb/previous-versions/xamarin/get-started/what-is-xamarin>.

ДОДАТКИ

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

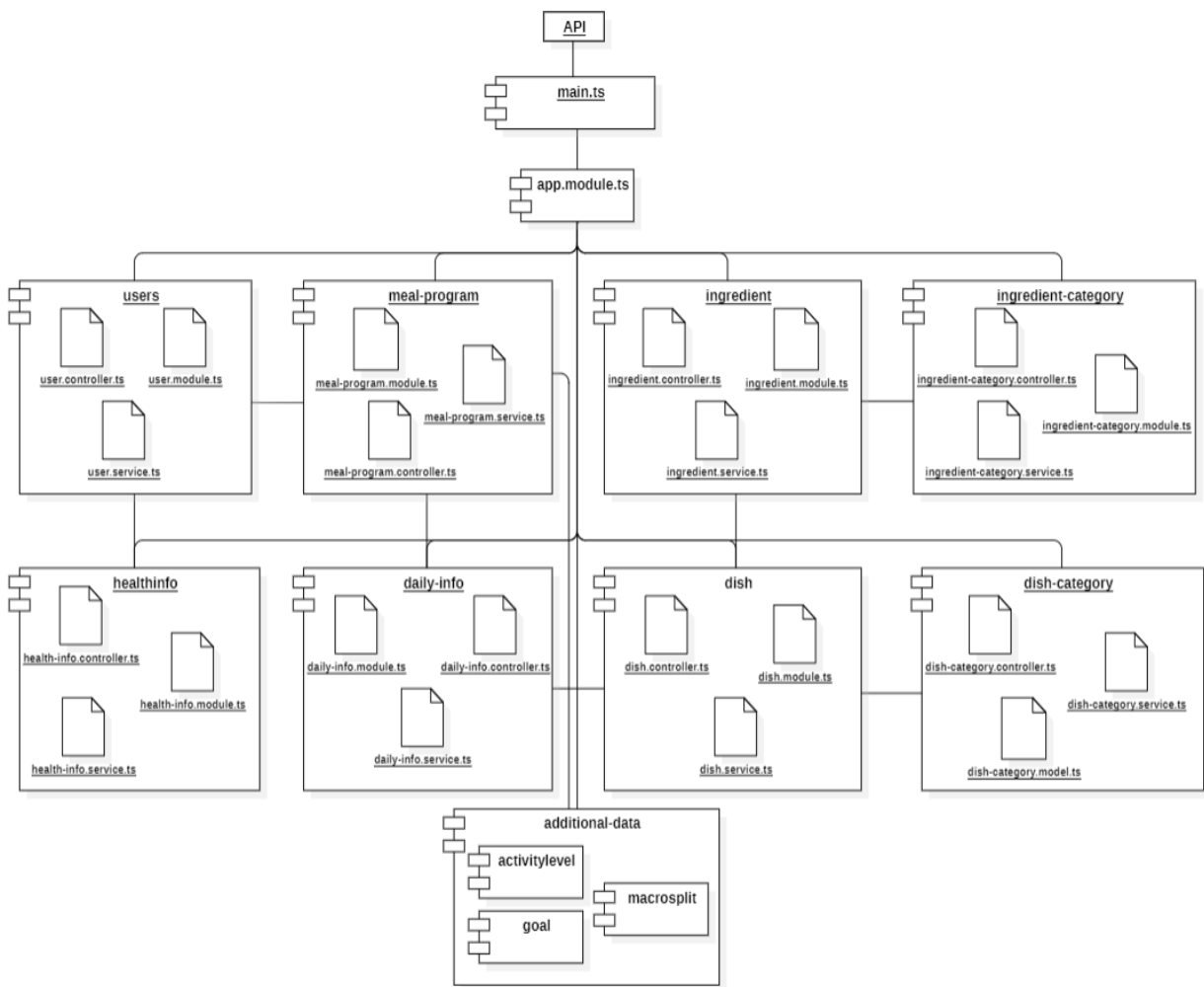


Рисунок А.1 – Діаграма компонентів серверної частини застосунку

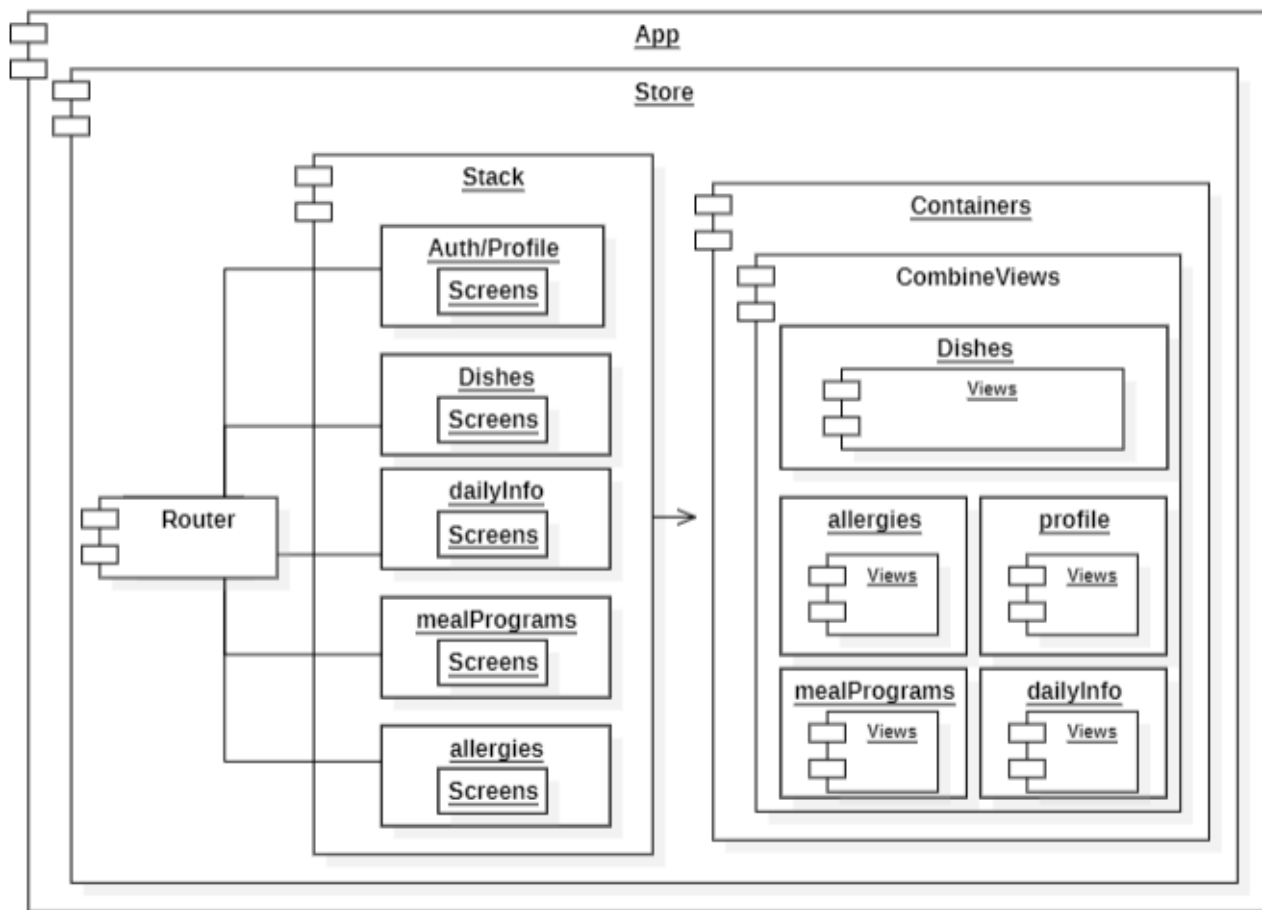


Рисунок А.2 – Діаграма компонентів клієнтської частини застосунку

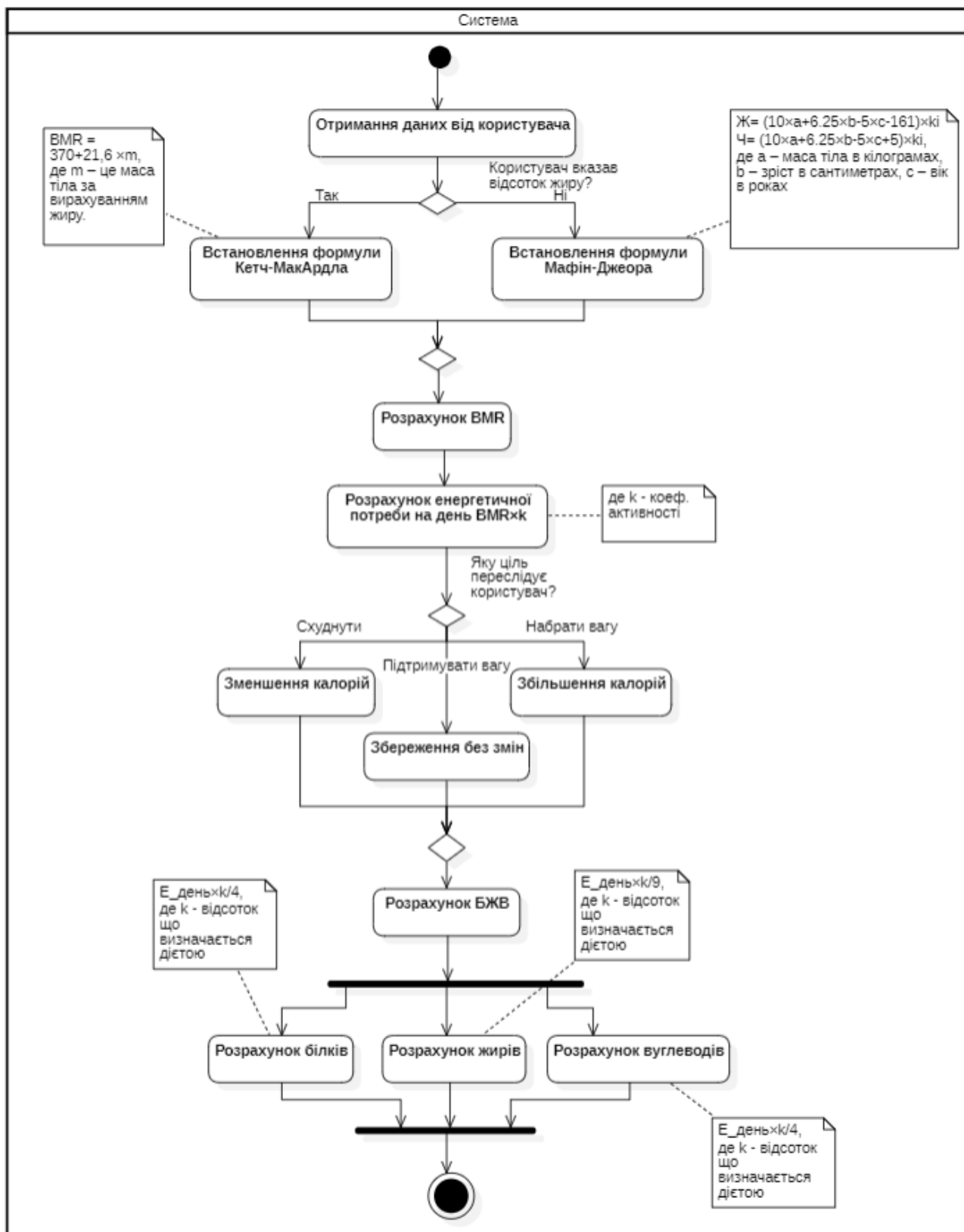


Рисунок Б.1 – Діаграма послідовності обрахунків

Лістинг коду обрахунків:

```

export const countBMR = (
  gender,
  numericWeight,
  numericAge,
  numericHeight,
  bodyFat
) => {
  let res = 0;
  if (!isNaN(bodyFat)) {
    // Katch-McArdle
    const lbm = (numericWeight * (100 - bodyFat)) / 100;
    res = 370 + 21.6 * lbm;
  } else {
    // Mifflin-St Jeor Equation
    if (gender === "female") {
      res = 10 * numericWeight + 6.25 * numericHeight - 5 * numericAge - 161;
    } else {
      res = 10 * numericWeight + 6.25 * numericHeight - 5 * numericAge + 5;
    }
  }
  return res;
};

export const countCaloriesWithGoal = (bmr, activity, goal) => {
  const bareCal = Math.round(bmr * activity.coefficient);
  if (goal?.value !== null && goal?.value !== undefined) {
    return bareCal + goal.value;
  } else if (goal?.coefficient !== null && goal?.coefficient !== undefined) {
    const caloriesReduction = bareCal * goal.coefficient;
    return bareCal - caloriesReduction;
  } else {
    return bareCal;
  }
};

export const countPCF = (
  cal,
  proteinPercentage,
  carbsPercentage,
  fatPercentage
) => {
  const proteinGrams = Math.round((cal * (proteinPercentage / 100)) / 4.0);
  const carbsGrams = Math.round((cal * (carbsPercentage / 100)) / 4.0);
  const fatGrams = Math.round((cal * (fatPercentage / 100)) / 9.0);
  return { proteinGrams, carbsGrams, fatGrams };
};

export const getMealCoefficientDistribution = (numberOfMeals) => {
  let mealDistribution;
  if (numberOfMeals == "3") {
    mealDistribution = {
      breakfast: { min: 0.3, max: 0.35 },
      lunch: { min: 0.35, max: 0.4 },
      dinner: { min: 0.25, max: 0.35 },
    };
  } else if (numberOfMeals == "4") {
    mealDistribution = {
      breakfast: { min: 0.25, max: 0.3 },
      morningSnack: { min: 0.05, max: 0.1 },
      lunch: { min: 0.35, max: 0.4 },
    };
  }
};

```

```

        dinner: { min: 0.25, max: 0.3 },
    };
} else if (numberOfMeals == "5") {
    mealDistribution = {
        breakfast: { min: 0.25, max: 0.3 },
        morningSnack: { min: 0.05, max: 0.1 },
        lunch: { min: 0.35, max: 0.4 },
        afternoonSnack: { min: 0.05, max: 0.1 },
        dinner: { min: 0.15, max: 0.2 },
    };
}
for (let meal in mealDistribution) {
    const { min, max } = mealDistribution[meal];
    mealDistribution[meal].average = (min + max) / 2;
}
return mealDistribution;
};

export const getMealDistribution = (totalInfo, distribution) => {
    const meals = {};

    Object.keys(distribution).forEach((meal) => {
        const { max, min, average } = distribution[meal];
        meals[meal] = {
            calories: {
                min: Math.round(totalInfo.calories * min),
                max: Math.round(totalInfo.calories * max),
                average: Math.round(totalInfo.calories * average),
            },
            carbohydrates: {
                min: Math.round(totalInfo.carbohydrates * min),
                max: Math.round(totalInfo.carbohydrates * max),
                average: Math.round(totalInfo.carbohydrates * average),
            },
            fat: {
                min: Math.round(totalInfo.fat * min),
                max: Math.round(totalInfo.fat * max),
                average: Math.round(totalInfo.fat * average),
            },
            proteins: {
                min: Math.round(totalInfo.proteins * min),
                max: Math.round(totalInfo.proteins * max),
                average: Math.round(totalInfo.proteins * average),
            },
        };
    });

    return meals;
};

export const countMealsCalories = (
    mealDistribution,
    totalCalories,
    totalProtein,
    totalCarbs,
    totalFat
) => {
    const mealCalories = {};

    Object.keys(mealDistribution).forEach((meal) => {
        const { min, max } = mealDistribution[meal];
        const minCalories = Math.round(min * totalCalories);

```

		Івченко Т.В.			ІІЗ.КР.Б – 121 – 11 – ІІЗ	Арк.
		Толстой І.А.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const maxCalories = Math.round(max * totalCalories);
const averageCalories = Math.round((minCalories + maxCalories) / 2);

const minProtein = Math.round(min * totalProtein);
const maxProtein = Math.round(max * totalProtein);
const averageProtein = Math.round((minProtein + maxProtein) / 2);

const minCarbs = Math.round(min * totalCarbs);
const maxCarbs = Math.round(max * totalCarbs);
const averageCarbs = Math.round((minCarbs + maxCarbs) / 2);

const minFat = Math.round(min * totalFat);
const maxFat = Math.round(max * totalFat);
const averageFat = Math.round((minFat + maxFat) / 2);

mealCalories[meal] = {
  minCalories,
  maxCalories,
  averageCalories,
  minProtein,
  maxProtein,
  averageProtein,
  minCarbs,
  maxCarbs,
  averageCarbs,
  minFat,
  maxFat,
  averageFat,
};
});

return mealCalories;
};

export const caloriesIntoKJ = (cal) => {
  return Math.round(cal * 4.187);
};

export const countAll = (
  gender,
  numericWeight,
  numericAge,
  numericHeight,
  bodyFat,
  activity,
  goal,
  expectedWeight,
  macros
) => {
  const BMR = countBMR(
    gender,
    numericWeight,
    numericAge,
    numericHeight,
    bodyFat
  );
  const calories = Math.round(countCaloriesWithGoal(BMR, activity, goal));
  const pcf = countPCF(calories, macros.protein, macros.carbs, macros.fat);
  const kJ = caloriesIntoKJ(calories);
  let weeks;
  if (expectedWeight !== numericWeight) {
    weeks = calculateTimeToTargetWeight(

```

		Івченко Т.В.			ІІЗ.КР.Б – 121 – 11 – ІІЗ	Арк.
		Толстой І.А.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        expectedWeight,
        numericWeight,
        goal.weightChangePerWeek
    );
} else {
    weeks = null;
}

return { BMR, calories, pcf, kJ, weeks };
};

const calculateTimeToTargetWeight = (
    currentWeight,
    targetWeight,
    weightChangePerWeek
) => {
    const weightDifference = Math.abs(currentWeight - targetWeight);
    const weeksNeeded = weightDifference / weightChangePerWeek;
    return isFinite(weeksNeeded) ? weeksNeeded : null;
};

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

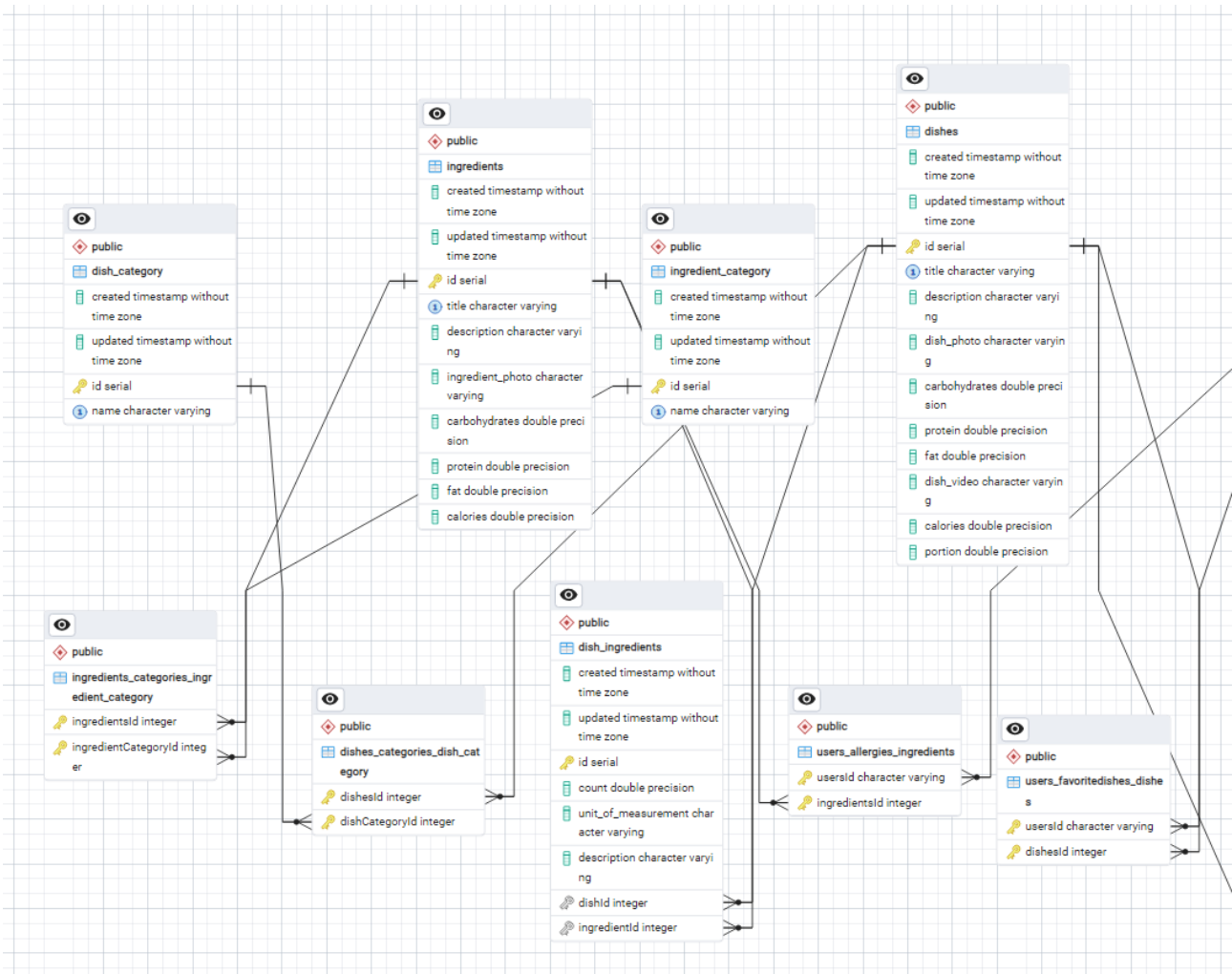


Рисунок В.1 – Діаграма бази даних ч.1

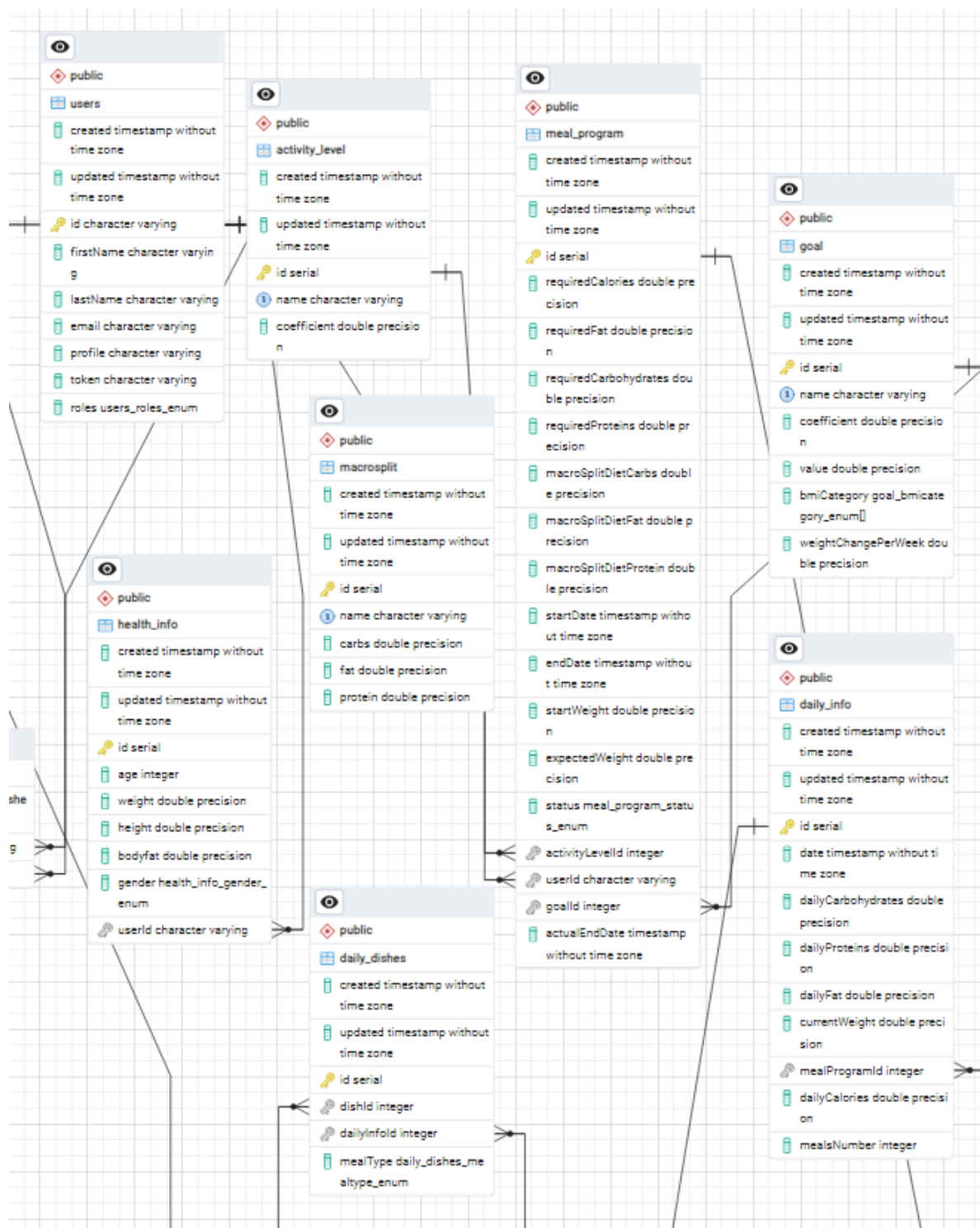


Рисунок В.2 – Діаграма бази даних ч.2

Лістинг UsersService:

```

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(Users) private userRepository: Repository<Users>,
    @InjectRepository(Ingredients)
    private ingredientsRepository: Repository<Ingredients>,
    @InjectRepository(Dishes)
    private dishRepository: Repository<Dishes>,
  ) {}

  async create(createUserDto: UsersCreateDto) {
    const user = await this.userRepository.save(createUserDto);
    const token = sign({ ...user }, process.env.JWT_CONSTANT);
    await this.userRepository.update(user.id, { token: token });

    return {
      ...user,
      token: token,
    };
  }

  async update(id: string, usersUpdateDto: Partial<UsersUpdateDto>) {
    const existingUser = await this.findUserById(id);
    Object.assign(existingUser, usersUpdateDto);
    return await this.userRepository.save(existingUser);
  }

  async findUserById(id: string) {
    const users = await this.userRepository.findOne({ where: { id: id } });
    return users;
  }

  async findUserByIdWithHealth(id: string) {
    const users = await this.userRepository
      .createQueryBuilder('user')
      .leftJoinAndSelect('user.healthInfo', 'healthInfo')
      .where('user.id = :id', { id })
      .getOne();
    return users;
  }

  async findUserByIdWithAllergies(id: string) {
    const users = await this.userRepository
      .createQueryBuilder('user')
      .leftJoinAndSelect('user.allergies', 'ingredients')
      .leftJoinAndSelect('ingredients.categories', 'categories')
      .where('user.id = :id', { id })
      .getOne();

    return users;
  }

  async findUserByIdWithFavDish(id: string) {
    const user = await this.userRepository
      .createQueryBuilder('users')
      .leftJoinAndSelect('users.favoritedishes', 'dishes')
      .where('users.id = :id', { id })

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .findOne();
        return user;
    }

    async findAll(): Promise<Users[]> {
        return this.userRepository.find();
    }

    async findOneUserByEmail(email: string) {
        return await this.userRepository.findOne({ where: { email: email } });
    }

    async remove(id: string) {
        const user = await this.userRepository.findOne({ where: { id } });
        await this.userRepository.remove(user);
        return {
            status: 200,
            message: `Користувача з id=${id} було видалено`,
        };
    }

    async addAllergies(id, addAllergiesDto: AllergiesDto): Promise<Users> {
        const user = await this.userRepository.findOne({
            where: { id: id },
            relations: ['allergies'],
        });

        if (!user) {
            throw new NotFoundException('Користувача не знайдено');
        }

        const allergy = await this.ingredientsRepository.findOne({
            where: { id: addAllergiesDto.ingredientsId },
        });

        if (!allergy) {
            throw new NotFoundException('Інгредієнту не знайдено');
        }

        const allergyExists = user.allergies.some(
            (a) => a.id === addAllergiesDto.ingredientsId,
        );
        if (allergyExists) {
            throw new HttpException(
                'Ця алергія вже додана користувачу',
                HttpStatus.CONFLICT,
            );
        }

        user.allergies = [...user.allergies, allergy];
        return this.userRepository.save(user);
    }

    async deleteAllergies(id, deleteAllergiesDto: AllergiesDto): Promise<Users> {
        const user = await this.userRepository.findOne({
            where: { id: id },
            relations: ['allergies'],
        });

        if (!user) {
            throw new NotFoundException('Користувача не знайдено');
        }
    }

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const allergyIndex = user.allergies.findIndex(
  (allergy) => deleteAllergiesDto.ingredientsId === allergy.id,
);

if (allergyIndex === -1) {
  throw new NotFoundException('Алергію не знайдено у користувача');
}

user.allergies.splice(allergyIndex, 1);

return this.userRepository.save(user);
}

async addFavDish(id, addfavdishDto: FavDishDto): Promise<Users> {
  const user = await this.userRepository.findOne({
    where: { id: id },
    relations: ['favoritedishes'],
  });

  if (!user) {
    throw new NotFoundException('Користувача не знайдено');
  }

  const dish = await this.dishRepository.findOne({
    where: { id: addfavdishDto.dishId },
  });

  if (!dish) {
    throw new NotFoundException('Страви не знайдено');
  }

  const dishExists = user.favoritedishes.some(
    (favDish) => favDish.id === addfavdishDto.dishId,
  );
  if (dishExists) {
    throw new HttpException(
      'Ця страва вже додана до улюблених користувача',
      HttpStatus.CONFLICT,
    );
  }

  user.favoritedishes = [...user.favoritedishes, dish];
  return this.userRepository.save(user);
}

async deleteFavDish(id, deleteFavDto: FavDishDto): Promise<Users> {
  const user = await this.userRepository.findOne({
    where: { id: id },
    relations: ['favoritedishes'],
  });

  if (!user) {
    throw new NotFoundException('Користувача не знайдено');
  }

  const dishExists = user.favoritedishes.some(
    (dish) => dish.id === deleteFavDto.dishId,
  );
  if (!dishExists) {
    throw new NotFoundException(
      'Страви не знайдено у списку улюблених страв користувача',
    );
  }

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		101

```

    });
}

user.favoritedishes = user.favoritedishes.filter(
  (dish) => deleteFavDto.dishId !== dish.id,
);

return this.userRepository.save(user);
}

async getUserAllergyIds(userId: string): Promise<number[]> {
  const user = await this.userRepository.findOne({
    where: { id: userId },
    relations: ['allergies'],
  });

  if (!user) {
    throw new Error('User not found');
  }

  return user.allergies.map((allergy) => allergy.id);
}

async findByIdWithFavDishAndAllergies(
  id: string,
  allergies: number[],
): Promise<Users> {
  // Знайти страви, що містять алергени
  const dishesWithAllergens = await this.dishRepository
    .createQueryBuilder('dish')
    .leftJoin('dish.dishingredients', 'dishingredients')
    .leftJoin('dishingredients.ingredient', 'ingredient')
    .where('ingredient.id IN (:...allergies)', { allergies })
    .select('dish.id')
    .getMany();

  const dishIdsWithAllergens = dishesWithAllergens.map((dish) => dish.id);

  // Знайти користувача з його улюбленими стравами, виключаючи страви з
  // алергенами
  const user = await this.userRepository
    .createQueryBuilder('user')
    .leftJoinAndSelect('user.favoritedishes', 'favoritedishes')
    .where('user.id = :id', { id })
    .andWhere('favoritedishes.id NOT IN (:...dishIdsWithAllergens)', {
      dishIdsWithAllergens:
        dishIdsWithAllergens.length > 0 ? dishIdsWithAllergens : [null],
    })
    .getOne();
  return user;
}
}

```

Лістинг MealProgramController:

```

@Controller('mealprogram')
export class MealProgramController {
  constructor(private readonly mealProgramService: MealProgramService) {}

  @Roles('user')
  @UseGuards(JwtAuthGuard, RoleGuard)

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@HttpCode(HttpStatus.OK)
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
@Post('create')
async create(
  @CurrentUser('sub') id: string,
  @Body() createMealProgramDto: CreateMealProgramDto,
): Promise<any> {
  try {
    const res = await this.mealProgramService.create(
      createMealProgramDto,
      id,
    );
    if (res) {
      return {
        status: true,
        result: res,
      };
    } else {
      return {
        status: false,
        result: [],
      };
    }
  } catch (error) {
    throw error;
  }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
@HttpCode(HttpStatus.OK)
@Patch('/update/:id')
async UpdateHealth(
  @Body() mealProgram: UpdateMealProgramDto,
  @Param('id') id: string,
) {
  try {
    const updatedMealProgram = await this.mealProgramService.update(
      id,
      mealProgram,
    );
    return {
      success: true,
      result: updatedMealProgram,
    };
  } catch (error) {
    console.log(error);
    throw error;
  }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
@Get('/allusermealprograms')

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІІЗ	Арк.
		Толстой І.А.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async getCurrentUserAllMealPrograms(@CurrentUser('sub') id: string) {
  try {
    const response =
      await this.mealProgramService.findAllUserMealPrograms(id);

    return {
      success: true,
      result: response,
    };
  } catch (error) {
    throw error;
  }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@Get('activemealprogram')
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
async getCurrentUserActiveMealProgram(
  @CurrentUser('sub') id: string,
): Promise<any> {
  const res = await this.mealProgramService.findActiveUserMealProgram(id);
  if (res) {
    return {
      success: true,
      result: res,
    };
  } else {
    return {
      success: false,
      result: [],
    };
  }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@Get('activemealprogramdayliInfo/:date')
@UseInterceptors(
  new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
async getCurrentUserActiveMealProgramByDate(
  @CurrentUser('sub') id: string,
  @Param('date') date: Date,
): Promise<any> {
  console.log(date);
  const res = await this.mealProgramService.findActiveUserMealProgramByDate(
    id,
    date,
  );
  if (res) {
    return {
      success: true,
      result: res,
    };
  } else {
    return {
      success: false,
      result: [],
    };
  }
}

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@Get('mealprogram/:status')
@UseInterceptors(
    new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
async getCurrentUserMealProgramsByStatus(
    @CurrentUser('sub') id: string,
    @Param('status') status: Status,
): Promise<any> {
    const res = await this.mealProgramService.findUserMealProgramByStatus(
        id,
        status,
    );
    if (res) {
        return {
            success: true,
            result: res,
        };
    } else {
        return {
            success: false,
            result: [],
        };
    }
}

@Roles('user')
@UseGuards(JwtAuthGuard, RoleGuard)
@Get('mealprogram-dailyinfo/:idProgr')
@UseInterceptors(
    new SerializeInterceptor<MealProgramGeneralDto>(MealProgramGeneralDto),
)
async getCurrentUserMealProgramByIdWithDailyInfo(
    @Param('idProgr') idProgr: number,
): Promise<any> {
    const res =
        await this.mealProgramService.findMealProgramByIdWithDailyInfo(idProgr);
    if (res) {
        return {
            success: true,
            result: res,
        };
    } else {
        return {
            success: false,
            result: [],
        };
    }
}
}

```

Лістинг коду основного модуля:

```

@Module({
    imports: [
        ConfigModule.forRoot(),
        TypeOrmModule.forRoot(configDB.getTypeOrmConfig()),
        TypeOrmModule.forFeature([Users]),
    ],
})

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    TypeOrmModule.forFeature([HealthInfo]),
    TypeOrmModule.forFeature([ActivityLevel]),
    TypeOrmModule.forFeature([Dishes]),
    TypeOrmModule.forFeature([Ingredients]),
    TypeOrmModule.forFeature([MealProgram]),
    TypeOrmModule.forFeature([DishIngredients]),
    TypeOrmModule.forFeature([Goal]),
    TypeOrmModule.forFeature([MacroSplit]),
    TypeOrmModule.forFeature([DailyInfo]),
    TypeOrmModule.forFeature([DailyDishes]),
    TypeOrmModule.forFeature([DishCategory]),
    TypeOrmModule.forFeature([IngredientCategory]),

    JwtModule.register({
      secret: process.env.JWT_CONSTANT,
      signOptions: { expiresIn: '1h' },
    }),
    PassportModule,
  ],
  controllers: [
    UsersController,
    HealthInfoController,
    DishController,
    IngredientController,
    MealProgramController,
    DishCategoryController,
    IngredientCategoryController,
    ActivityLevelController,
    GoalController,
    MacroSplitController,
    DishIngredientController,
    DailyInfoController,
    DailyDishesController,
  ],
  providers: [
    UsersService,
    HealthInfoService,
    ActivityLevelService,
    DishService,
    IngredientService,
    DishIngredientService,
    MealProgramService,
    DailyInfoService,
    JwtStrategy,
    AuthService,
    DailyDishesService,
    GoalService,
    MacroSplitService,
    DishCategoryService,
    IngredientCategoryService,
  ],
})
export class AppModule {}

```

Лістинг коду підключення БД:

```

require('dotenv').config();
import { TypeOrmModuleOptions } from '@nestjs/typeorm';

class ConfigDB {
  constructor(private env: { [key: string]: string | undefined }) {}

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private getValue(key: string, throwOnMissing = true): string {
  const value = this.env[key];
  if (!value && throwOnMissing) {
    throw new Error(`env missing ${key} value`);
  }
  return value;
}

public ensureValues(keys: string[]) {
  keys.forEach((key) => this.getValue(key, true));
  return this;
}

public getTypeOrmConfig(): TypeOrmModuleOptions {
  return {
    type: 'postgres',
    host: this.getValue('POSTGRES_HOST'),
    port: parseInt(this.getValue('POSTGRES_PORT')),
    username: this.getValue('POSTGRES_USER'),
    password: this.getValue('POSTGRES_PASSWORD'),
    database: this.getValue('POSTGRES_DATABASE'),
    entities: [__dirname + '/*.entity{.js, .ts}'],
    autoLoadEntities: true,
    synchronize: true,
  };
}
}

const configDB = new ConfigDB(process.env).ensureValues([
  'POSTGRES_HOST',
  'POSTGRES_PORT',
  'POSTGRES_USER',
  'POSTGRES_PASSWORD',
  'POSTGRES_DATABASE',
]);

export { configDB };

```

Лістинг коду хука useAxios

```

import { useState } from "react";
import axios from "axios";
import { useAppContext } from "../context/mealContext";

const useAxios = () => {
  const [data, setData] = useState();
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState();
  const { state } = useAppContext();

  const sendRequest = async (endPoint, body = null, method) => {
    let baseUrl = "http://192.168.50.124:3000/";
    let url;

    if (endPoint) {
      url = baseUrl + endPoint;
    }

    setLoading(true);
    setError(null);
    try {
      let axiosConfig = {

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        method: method ? method : body ? "POST" : "GET",
        url,
        headers: { Authorization: `Bearer ${state?.token}` },
    };
    if (body) {
        axiosConfig = {
            ...axiosConfig,
            data: body,
        };
    }
    const response = await axios(axiosConfig);
    setData(response?.data);
    return response;
} catch (error) {
    console.log(error.response.data.message);
    setError(error.response.data);
} finally {
    setLoading(false);
}
};

return { data, loading, error, sendRequest };
};

export default useAxios;

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ІЗ	Арк.
		Толстой І.А.				108
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг коду тестів:

```

import React from "react";
import { TextInput, TouchableOpacity, Text } from "react-native";
import renderer from "react-test-renderer";
import InputBoxForm from "../components/Form/InputBoxForm";

jest.mock("../help/namesInUkrainian", () => ({
  bodyFatInfo: "Mocked Body Fat Info",
}));

jest.mock("react-hook-form", () => ({
  ...jest.requireActual("react-hook-form"),
  useForm: jest.fn(() => ({
    control: {
      _names: {},
    },
  })),
  Controller: jest.fn(({ render }) =>
    render({
      field: {
        value: "",
        onChange: jest.fn(),
        onBlur: jest.fn(),
      },
      fieldState: {},
    })
  ),
}));

describe("InputBoxForm", () => {
  it("Рендериться коректно зі стандартними параметрами", () => {
    const component = renderer.create(
      <InputBoxForm name="test" placeholder="Test Input" />
    );
    const tree = component.toJSON();
    expect(tree).toMatchSnapshot();
  });
  it("Перемикає видимість пароля", () => {
    const component = renderer.create(
      <InputBoxForm name="password" placeholder="Password" />
    );
    const instance = component.root;
    const input = instance.findByType(TextInput);
    const toggleButton = instance.findByProps({ testID: "passwordToggle" });
    toggleButton.props.onPress();
    expect(input.props.secureTextEntry).toBe(false);
    toggleButton.props.onPress();
    expect(input.props.secureTextEntry).toBe(true);
  });
  it("Відкриває модальне вікно при натисканні на іконку інформації про відсоток жиру", () => {
    const component = renderer.create(
      <InputBoxForm name="bodyfat" placeholder="Body Fat" />
    );
    const instance = component.root;
    const infoButton = instance.findByProps({ testID: "infoButton" });
    infoButton.props.onPress();
  });
});

```

		Івченко Т.В.			ІПЗ.КР.Б – 121 – 11 – ПЗ	Арк.
		Толстой І.А.				109
Змн.	Арк.	№ докум.	Підпис	Дата		