

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ПОГОДЖЕНО
Декан факультету

_____ **Ігор БОЛБОТ**
(підпис)

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ **Белла ГОЛУБ**
(підпис)

« ____ » _____ 2026 р. « ____ » _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи підтримки здорового способу життя»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
К.Т.Н., доцент

_____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської
кваліфікаційної роботи**

_____ **Іван КОРНІЛОВ**
(підпис)

**Консультант бакалаврської
кваліфікаційної роботи**
К.Т.Н., С.Н.С.

_____ **Юлія БОЯРІНОВА**
(підпис)

Виконав

_____ **Євген ВІНТОНЮК**
(підпис)

КИЇВ – 2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Вінтонюку Євгену Сергійовичу

(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення системи підтримки здорового способу життя»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «_22_» травня 2026 р.

Вихідні дані до роботи: опис системи моніторингу здорового способу життя; алгоритм Міффіна-Сан Жеора; бази даних продуктів і тренувань; стек технологій: Python, Aiogram, SQLAlchemy, SQLite.

Перелік питань, що підлягають дослідженню: системний аналіз предметної області; проєктування бази даних; розробка програмного забезпечення Telegram-бота; тестування та розгортання систем

Перелік графічних документів (за необхідності).

Дата видачі завдання «_22_» _____ грудня _____ 2025 р.

**Керівник бакалаврської
кваліфікаційної роботи**

_____ **Іван КОРНІЛОВ**
(підпис)

**Консультант бакалаврської
кваліфікаційної роботи**

_____ **Юлія БОЯРІНОВА**
(підпис)

Завдання взяв до виконання

_____ **Євген ВІНТОНЮК**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	4
ВСТУП.....	6
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ.....	9
1.1 Системний аналіз предметної області та актуальність розробки	9
1.2. Обґрунтування вибору теми розробки	11
1.3. Огляд інформаційних джерел та існуючих рішень за темою розробки....	12
1.4. Постановка завдання на розробку інтелектуальної системи.....	22
2 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ ТА ПРОЄКТУВАННЯ СИСТЕМИ.....	23
2.1. Логічна модель у вигляді ER-діаграми	23
2.2. Архітектура та структура програмного продукту	26
2.3. Фізична модель бази даних	31
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ	34
3.1. Структура програмного продукту	34
3.2. Алгоритмічне забезпечення системи.....	37
3.3. Опис інтерфейсу користувача та взаємодії.....	44
3.4. Інструкція користувача	48
4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ	50
4.1. Тестування функціоналу програмного продукту	50
4.2. Вимоги до апаратного та програмного середовища	56
ВИСНОВКИ	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	66
ДОДАТОК А	68
ДОДАТОК Б.....	90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- 3NF - Third Normal Form, третя нормальна форма, що забезпечує відсутність надмірності даних у реляційній базі.
- API - Application Programming Interface, програмний інтерфейс взаємодії веб-сервісів і модулів системи (зокрема Telegram Bot API).
- BMR - Basal Metabolic Rate, базовий рівень метаболізму або швидкість основного обміну речовин користувача.
- CPU - Central Processing Unit, центральний процесор серверного або локального пристрою.
- CRUD - Create, Read, Update, Delete, набір базових операцій управління даними (створення, читання, оновлення, видалення).
- ER - Entity-Relationship, модель "Сутність-зв'язок" для візуального проектування структури бази даних.
- FSM - Finite State Machine, скінченний автомат для управління станами діалогу та контекстом взаємодії бота.
- MVP - Minimum Viable Product, мінімально життєздатний продукт, базовий функціонал якого готовий до розгортання.
- ORM - Object-Relational Mapping, засіб взаємодії програмних об'єктів (наприклад, SQLAlchemy) із реляційною БД.
- PAL - Physical Activity Level, коефіцієнт фізичної активності, що використовується для розрахунку енерговитрат.
- RAM - Random Access Memory, оперативна пам'ять серверного пристрою або смартфона користувача.
- SSD - Solid-State Drive, твердотільний накопичувач сервера для швидкого доступу до даних.
- TDEE - Total Daily Energy Expenditure, загальна добова витрата енергії з урахуванням фізичних навантажень.
- UI - User Interface, розмовний або графічний інтерфейс взаємодії користувача із системою.

UML - Unified Modeling Language, уніфікована мова моделювання для візуалізації архітектури програмного продукту.

UX - User Experience, досвід взаємодії та зручність використання системи кінцевим споживачем.

VPS / VDS - Virtual Private Server / Virtual Dedicated Server, віртуальний приватний або виділений сервер для хостингу бота.

БД - База даних, структурований масив збережених профілів користувачів та логів харчування (наприклад, SQLite).

БЖВ - Білки, жири та вуглеводи, основні макронутрієнти продуктів харчування.

ЗСЖ - Здоровий спосіб життя, концепція підтримки фізичного та ментального здоров'я.

ІТ - Інформаційні технології.

ОЗП - Оперативний запам'ятовуючий пристрій (оперативна пам'ять).

ОС - Операційна система серверного або клієнтського середовища.

ПЗ - Програмне забезпечення.

СКБД - Система керування базами даних.

ВСТУП

У 2026 році цифрова трансформація сфери охорони здоров'я та персонального благополуччя досягла нового рівня, де користувачі віддають перевагу максимально легким та інтегрованим рішенням. Проблема підтримки здорового способу життя (ЗСЖ) залишається критичною через глобальні тренди на диджиталізацію робочих процесів, що призводить до хронічної гіподинамії та порушення культури харчування. Більшість існуючих комерційних рішень перевантажені рекламою, складними інтерфейсами та вимагають значних ресурсів пристрою.

Розробка інтелектуальної системи «OptiBody» на базі месенджера Telegram є актуальною відповіддю на ці виклики. Використання чат-бота як інтерфейсу дозволяє знизити бар'єр входу для користувача, забезпечуючи миттєвий доступ до інструментів контролю без необхідності встановлення додаткового програмного забезпечення. Науково-технічна цінність проєкту полягає у впровадженні суворого методу обліку нутрієнтів за «сирою» масою інгредієнтів, що є найбільш точним підходом у дієтології, та реалізації асинхронної архітектури обробки даних для забезпечення високої швидкості відгуку системи.

Мета розробки. Спроекувати та реалізувати інтелектуальну інформаційну систему «OptiBody», яка за допомогою асинхронних алгоритмів Telegram Bot API забезпечує персоналізований моніторинг енергетичного балансу, базуючись на індивідуальних антропометричних параметрах користувача та науково обґрунтованих формулах метаболічного обміну.

Завдання дослідження:

- Дослідити сучасні підходи до автоматизації контролю ЗСЖ та визначити переваги асинхронних інтерфейсів.
- Спроекувати архітектуру бази даних для збереження довідникової інформації про продукти та журналів активності користувачів.

- Реалізувати алгоритм розрахунку добової норми калорій за формулою Міффіна-Сан Жеора.
- Розробити модульну систему обробки вхідних даних (харчування, тренування) з використанням механізму станів (FSM).
- Провести комплексне тестування системи на відповідність функціональним вимогам щодо точності розрахунків та стабільності збереження даних.

Об'єкт дослідження – процес моніторингу та корекції показників енергетичного балансу людини за допомогою інтелектуальних цифрових сервісів.

Предмет дослідження – алгоритми, методи та програмні засоби реалізації асинхронної взаємодії в межах Telegram-інтерфейсу для контролю харчування та фізичних навантажень.

Методи та технології розробки. У роботі використано принципи об'єктно-орієнтованого проектування та асинхронного програмування. Технологічний стек включає:

- Python 3.10+ як основну мову розробки.
- Aiogram 3.28 для реалізації логіки взаємодії з Telegram Bot API.
- SQLAlchemy як інструмент ORM для взаємодії з реляційною базою даних.
- SQLite для локального та надійного зберігання даних.
- FSM для управління логічними ланцюжками діалогів.

Практичне значення. Створена система «OptiBody» є готовим до розгортання програмним продуктом, який дозволяє користувачу в режимі реального часу вести щоденник харчування, враховуючи калорійність продуктів у їх первинному (сирому) стані, що мінімізує статистичні похибки. Система надає гнучкі інструменти для оновлення фізичних параметрів та швидкого редагування історії записів, що робить її ефективним помічником у досягненні фітнес-цілей.

Структура записки. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг основної частини становить 65 сторінок. Список джерел містить 20 найменувань.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Системний аналіз предметної області та актуальність розробки

На сучасному етапі розвитку постіндустріального суспільства спостерігається фундаментальна трансформація способу життя людини, зумовлена тотальною цифровізацією виробничих та побутових процесів. Одним із найбільш критичних наслідків цього прогресу є феномен «цифрової сидячості» та хронічної гіподинамії, що особливо гостро проявляється серед фахівців інтелектуальної праці, зокрема представників ІТ-індустрії. В умовах, коли більша частина активного часу проводиться в статичному стані перед монітором, природні механізми регуляції енергетичного обміну в організмі людини починають давати збої, що призводить до поступового накопичення надлишкової маси тіла та розвитку супутніх метаболічних розладів.

Системний аналіз предметної області Digital Health (цифрової охорони здоров'я) дозволяє констатувати, що підтримка здорового способу життя (ЗСЖ) у 2026 році перестала бути просто модною тенденцією і перетворилася на стратегічну необхідність для збереження когнітивного ресурсу та працездатності індивіда. Ключовим викликом у цій сфері є не стільки відсутність знань про принципи раціонального харчування, скільки складність практичного застосування цих знань у повсякденній рутині. Користувач стикається з необхідністю постійного моніторингу великої кількості параметрів: калорійності продуктів, співвідношення білків, жирів та вуглеводів (БЖВ), рівня фізичної активності та водного балансу.

Особливої уваги заслуговує проблема точності вимірювань. Існуючі дослідження в галузі нутриціології вказують на те, що суб'єктивна оцінка спожитих калорій користувачем може відхилятися від реальних показників на

30–50%. Це зумовлено складністю оцінювання енергетичної цінності багатокомпонентних готових страв. Наприклад, калорійність термічно обробленої крупи чи м'яса суттєво залежить від способу приготування та кількості поглинутої вологи. Саме тому в системі «OptiBody» закладено фундаментальний принцип розрахунку за сировою (первинною) вагою інгредієнтів. Такий підхід дозволяє нівелювати похибки приготування та забезпечити лабораторну точність домашнього трекінгу, що є критично важливим для досягнення цілей щодо зниження або підтримки ваги.

Паралельно з цим, системний аналіз психології взаємодії користувача з програмним забезпеченням виявляє проблему «когнітивного перевантаження» від надмірної кількості самостійних додатків. Встановлення, налаштування та регулярне оновлення спеціалізованого софту для фітнесу часто стає бар'єром, через який користувачі припиняють ведення щоденника вже через кілька днів. Месенджер Telegram у цьому контексті виступає не просто як засіб зв'язку, а як універсальне середовище виконання асинхронних мікро-застосунків.

Розробка інтелектуальної системи «OptiBody» у форматі чат-бота дозволяє інтегрувати процес моніторингу ЗСЖ безпосередньо у потік щоденної комунікації користувача. Це вирішує низку системних задач:

- Зниження бар'єру входу: відсутність необхідності завантажувати стороннє ПЗ та проходити складні процедури авторизації.
- Асинхронність взаємодії: можливість внести дані про прийом їжі за лічені секунди, що підвищує ймовірність довгострокового дотримання режиму моніторингу.
- Кросплатформеність: миттєвий доступ до своїх даних як з мобільного пристрою, так і з робочої станції (desktop-версії), що є критичним для офісних працівників.
- Конфіденційність та централізація: збереження всіх персональних логів у межах єдиної безпечної екосистеми месенджера з використанням локальної бази даних на серверній частині бота.

Таким чином, розробка системи «OptiBody» є науково та практично обґрунтованою відповіддю на сучасні виклики автоматизації контролю здоров'я. Вона поєднує в собі строгі наукові методи розрахунку метаболізму з максимально спрощеним та ергономічним інтерфейсом користувача, що дозволяє перетворити складний процес дієтологічного обліку на просту щоденну звичку.

1.2. Обґрунтування вибору теми розробки

Вибір теми даної кваліфікаційної роботи обумовлений необхідністю розв'язання гострого соціально-технологічного протиріччя, що виникло в умовах сучасної постіндустріальної епохи. З одного боку, спостерігається стрімка деградація загальних показників здоров'я населення через поширення сидячого способу життя, що є прямим наслідком тотальної автоматизації робочих місць. З іншого боку, існуючий інструментарій для боротьби з цими наслідками часто є занадто складним або фінансово недоступним для масового користувача.

Варто акцентувати увагу на тому, що у 2026 році ринок персональних асистентів переживає етап «втоми від надмірності». Більшість користувачів відчують когнітивне навантаження від необхідності керувати десятками автономних додатків на своїх смартфонах. Саме тому вибір месенджера Telegram як платформи для розгортання системи «OptiBody» є стратегічно обґрунтованим рішенням. Telegram перестав бути лише засобом обміну повідомленнями, перетворившись на потужне операційне середовище для виконання інтелектуальних мікро-сервісів.

Тема розробки інтелектуального бота є актуальною також з інженерної точки зору. Вона дозволяє продемонструвати ефективність використання асинхронних архітектур та легких реляційних баз даних для вирішення завдань у реальному часі. Вибір на користь методики розрахунку за сировою масою інгредієнтів підкреслює наукову орієнтованість проекту, оскільки цей підхід є найбільш обґрунтованим у сучасній дієтології, але водночас найменш

представленим у популярному комерційному ПЗ. Таким чином, розробка системи «OptiBody» заповнює дефіцит ефективних, безкоштовних та високоточних засобів персонального моніторингу, інтегрованих у щоденний комунікаційний потік користувача.

1.3. Огляд інформаційних джерел та існуючих рішень за темою розробки

Еволюція програмного забезпечення для підтримки здорового способу життя пройшла довгий шлях від простих табличних калькуляторів до складних екосистем, що використовують алгоритми штучного інтелекту та великі дані. На сучасному етапі ринок мобільних технологій у сегменті Health & Fitness перенасичений різноманітними рішеннями, проте аналіз інформаційних джерел свідчить про наявність стійкого технологічного розриву між функціональною складністю професійних систем та повсякденними потребами пересічного користувача.

Для обґрунтування розробки системи «OptiBody» було проведено ґрунтовний аудит найбільш репрезентативних представників сучасного ринку програмних продуктів для контролю нутрієнтів та активності.

1. MyFitnessPal (Under Armour)

Ця система вважається світовим лідером за обсягом бази даних продуктів (понад 14 мільйонів найменувань) та кількістю активних користувачів. Проте, з точки зору системного аналізу, продукт має суттєві недоліки:

- **Функціональна надмірність:** Програма намагається охопити всі аспекти життя (сон, воду, кроки, тренування, соціальну мережу), що призводить до перевантаження інтерфейсу та ускладнює швидке введення даних.
- **Агресивна монетизація:** Більшість критично важливих функцій (наприклад, сканування штрих-кодів у деяких регіонах або детальний розподіл БЖВ за прийомами їжі) вимагають дорогої передплати.

- Ресурсна місткість: Додаток займає значний обсяг пам'яті мобільного пристрою та потребує постійних оновлень, що суперечить концепції «легкого» цифрового асистента.

2. FatSecret

Популярний застосунок, що фокусується виключно на підрахунку калорій. Він є більш лаконічним порівняно з попереднім аналогом, проте має свої обмеження:

- Орієнтація на готові страви: База даних FatSecret значною мірою наповнюється самими користувачами, що призводить до появи тисяч дублікатів одного й того самого продукту з кардинально різними показниками калорійності. Це створює проблему дезінформації та знижує довіру до системи.
- Відсутність гнучкого редагування: Процес виправлення випадково введеної помилкової інформації часто вимагає виконання кількох послідовних переходів по меню, що знижує оперативність взаємодії.

3. Спеціалізовані веб-сервіси та калькулятори

Існує велика кількість онлайн-ресурсів (наприклад, Nutritionix або локальні бази даних нутрієнтів), які надають точну інформацію про склад продуктів.

Переваги: Висока наукова точність та безкоштовність.

Недоліки: Повна відсутність персоналізації та збереження історії. Користувач змушений щоразу вручну вводити свої параметри, що робить системний моніторинг у довгостроковій перспективі практично неможливим.

Для наочної демонстрації переваг розроблюваного рішення нижче наведено порівняльну таблицю (таблиця 1.1) за ключовими критеріями, що визначають ефективність систем підтримки ЗСЖ.

Таблиця 1.1

Порівняльна таблиця існуючих підходів

Характеристика системи	MyFitnessPal / FatSecret	Спеціалізовані веб-сайти	OptiBody (Чат-бот)
Форм-фактор	Автономний мобільний додаток	Веб-сторінка (браузер)	Інтегрований бот у месенджері
Бар'єр входу	Високий (встановлення, реєстрація)	Середній (пошук у мережі)	Мінімальний (старт бота)
Методика розрахунку	Переважно готові страви	Теоретична база	Сувора (сира вага продуктів)
Вимоги до пристрою	Значні (ОЗП, вільне місце)	Наявність стабільного браузера	Мінімальні (вимоги Telegram)
Асинхронність	Обмежена (постійні нотифікації)	Відсутня	Повна (за запитом користувача)
Вартість	Freemium (з обмеженнями)	Безкоштовно	Повністю безкоштовно

Аналіз інформаційних джерел підтверджує, що сучасний користувач перебуває в ситуації «вибору без вибору»: або використовувати складні комерційні додатки з великою кількістю реклами та неточною базою готових страв, або витратити час на ручні розрахунки в таблицях.

Саме тому розробка «OptiBody» на базі Telegram є технологічно обґрунтованою та актуальною. Вона поєднує в собі:

1. Наукову строгість: Використання формули Міффіна-Сан Жеора як стандарту доказової дієтології.
2. Точність обліку: Філософія розрахунку за сирою масою інгредієнтів дозволяє користувачу самостійно контролювати складність страв та уникати прихованих калорій, що часто додаються у закладах громадського харчування.
3. Асинхронний UX: Архітектура чат-бота дозволяє реалізувати механізм «миттєвого запису», де взаємодія з базою даних SQLite відбувається

через прозорий інтерфейс месенджера, що значно підвищує дотримуваність користувача щодо обраної стратегії харчування.

Таким чином, огляд існуючих рішень демонструє, що інтелектуальна система «OptiBody» заповнює вільну нішу легких, високоточних та асинхронних інструментів персонального моніторингу здоров'я, пропонуючи користувачу якісно новий рівень контролю без додаткового когнітивного навантаження.

Розроблювана система за своєю архітектурою та функціональним наповненням належить до класу індивідуальних інформаційно-рекомендаційних систем із елементами підтримки прийняття рішень. Основним призначенням системи «OptiBody» є автоматизація процесу збору, накопичення та інтелектуальної обробки даних про енергетичний баланс користувача в режимі реального часу. Система орієнтована на забезпечення повного циклу моніторингу ЗСЖ: від первинної антропометричної ідентифікації до формування щоденних статистичних звітів про досягнення фітнес-цілей.

Інформаційний контур системи поділяється на три ключові масиви даних, кожен з яких відіграє критичну роль у функціонуванні бота:

1. Персональні дані профілю (Вхідна інформація):
 - Унікальний ідентифікатор користувача в месенджері Telegram.
 - Антропометричні показники: біологічна стать, актуальний вік (роки), точний зріст (см) та поточна вага (кг).
 - Коефіцієнт фізичної активності (PAL), що визначає інтенсивність щоденних навантажень.
2. Довідникова інформація (Нормативно-довідкові дані):
 - Реляційна база даних продуктів харчування, що містить понад 180 позицій.
 - Показники енергетичної цінності (ккал) та нутрієнтного складу на 100 грамів продукту в його сирому (необробленому) стані.
3. Журнали активності (Процесна інформація):
 - DailyLogs: записи про спожиті продукти із зазначенням точного часу та маси порції. Також цей масив містить записи про кастомні

продукти, яких немає в довіднику, із зазначенням користувацької назви та введеної вручну кількості калорій.

- WorkoutLogs: дані про виконані фізичні вправи, тривалість та розраховану кількість спалених калорій.

Для досягнення мети розробки, програмна система має забезпечувати автоматичне виконання таких операцій:

- Динамічний розрахунок BMR та TDEE: використання формули Міффіна-Сан Жеора для миттєвого визначення базового обміну речовин на основі введених параметрів профілю.
- Інтелектуальна фільтрація та пошук: реалізація алгоритмів вибірки даних із бази продуктів за частковим збігом назви.
- Обробка користувацького введення поза базою даних: забезпечення можливості ручного внесення назви невідомого продукту та його калорійності у випадках, коли алгоритм пошуку не знаходить збігів у довіднику.
- Автоматичне зведення енергетичного балансу: математичне порівняння сумарного обсягу спожитих калорій із витраченими (базовий метаболізм + тренування) для визначення дефіциту або профіциту енергії.
- Управління станами діалогу: автоматизація переходів між різними етапами введення даних за допомогою механізму Finite State Machine (FSM), що запобігає втраті контексту взаємодії.

Важливим етапом системного аналізу предметної області та проектування архітектури системи підтримки здорового способу життя є формалізація функціональних вимог до програмного продукту. Найбільш ефективним інструментом для об'єктивного опису взаємодії майбутнього користувача з компонентами системи на етапі передпроектного дослідження є використання апарату уніфікованої мови моделювання UML, зокрема побудова діаграми прецедентів (Use Case Diagram) [9].

Діаграма прецедентів є високорівневим абстрактним відображенням функціональних можливостей системи, що дозволяє чітко окреслити її межі, виявити суб'єктів взаємодії (акторів) та зафіксувати перелік завдань, які система виконує для задоволення кінцевих потреб. У контексті розробки чат-бота архітектурне моделювання через прецеденти набуває особливого значення. Оскільки розмовний інтерфейс за своєю природою базується на послідовному обміні текстовими повідомленнями та командами, декомпозиція бізнес-логіки на чіткі прецеденти дозволяє уникнути неоднозначності архітектури та закласти надійне підґрунтя для розробки керуючих алгоритмів.

Головним та єдиним прямим актором у розробленій системі виступає Користувач. Він є ініціатором усіх обчислювальних процесів та безпосереднім суб'єктом споживання інформаційних послуг, які надає програмний комплекс. Межі системи охоплюють фонову логіку роботи бота, включаючи модулі математичних розрахунків, управління станами діалогу та взаємодію з локальним реляційним сховищем даних.

У межах інформаційного контуру системи було виділено, детально структуровано та описано чотири основні мета-прецеденти (базові варіанти використання), які повністю покривають життєвий цикл експлуатації програмного продукту.

Варіант використання «Налаштування профілю». Цей прецедент відповідає за первинну ідентифікацію користувача, збір його індивідуальних параметрів та налаштування метаболічного профілю. Процес декомпонується на

кілька обов'язкових та опціональних підпроцесів за допомогою стандартних стереотипів зв'язків UML:

- Зв'язок включення (include) до прецеденту «Введення: стать, вік, зріст, вага». Це вказує на те, що базовий сценарій налаштування профілю фізіологічно не може бути виконаний без послідовного надання та програмної валідації цих антропометричних показників.
- Зв'язок включення (include) до прецеденту «Розрахунок BMR та TDEE». Після успішного збору фізичних параметрів система автоматично та без додаткових дій з боку користувача запускає обчислювальний модуль для визначення швидкості основного обміну речовин та сумарних добових енерговитрат.
- Зв'язок розширення (extend) від прецеденту «Оновлення антропометрії». Дана дія є опціональною і не виконується під час щоденної експлуатації системи, а активується лише у разі виявлення користувачем потреби у зміні поточних параметрів профілю (наприклад, при зміні фактичної ваги тіла).

Варіант використання «Щоденник харчування». Цей блок автоматизує щоденний облік спожитих нутрієнтів та енергії, забезпечуючи високу точність збору інформації. Він включає наступні взаємопов'язані операції:

- Зв'язок включення (include) до прецеденту «Пошук продукту в базі». При отриманні назви інгредієнта бот виконує внутрішній запит до довідкової таблиці, реалізуючи алгоритм фільтрації за частковим збігом буквених символів.
- Зв'язок включення (include) до прецеденту «Введення ваги сирової порції». Відображає суворе дотримання дієтологічної методології обліку продуктів у їх первинному стані для нівелювання похибок термічної обробки їжі.
- Зв'язок включення (include) до прецеденту «Розрахунок калорійності порції». На основі отриманої маси та довідникової цінності продукту на 100 грамів система автоматично обчислює фінальну калорійність.

- Зв'язок розширення (extend) від прецеденту «Видалення помилкового запису їжі». Дозволяє користувачу у разі випадкового або некоректного введення лога ініціювати процедуру редагування бази даних та динамічно вилучити конкретний рядок із транзакційної таблиці через інлайн-інтерфейс.

Варіант використання «Облік тренувань». Відповідає за фіксацію додаткових витрат енергії під час фізичної активності, що є обов'язковим для коректного моніторингу загального енергетичного балансу:

- Зв'язок включення (include) до прецеденту «Вибір інтенсивності навантаження». Користувач обирає один із фіксованих рівнів складності виконаного тренування за допомогою інтерактивного меню кнопок.
- Зв'язок включення (include) до прецеденту «Розрахунок спалених калорій». На основі обраної складності система вираховує обсяг спаленої енергії та фіксує запис у журналі фонової активності.

Варіант використання «Аналіз статистики». Забезпечує надання користувачу інтелектуального аналітичного зворотного зв'язку про стан його організму та прогрес у досягненні фітнес-цілей:

- Зв'язок включення (include) до прецеденту «Агрегація балансу за добу». Програмне ядро виконує кумулятивний підрахунок усіх спожитих калорій та віднімає від них суму базових енерговитрат профілю та енергії, витраченої на зафіксованих тренуваннях.
- Зв'язок включення (include) до прецеденту «Генерація Markdown-звіту». Результати математичного аналізу конвертуються у текстові повідомлення зі строгою ієрархічною структурою для полегшення сприйняття інформації.
- Зв'язок розширення (extend) від прецеденту «Формування звіту за період». Надає користувачу можливість вийти за межі стандартного добового моніторингу та зробити ретроспективний аналіз накопичених

часових рядів даних за останні 7 або 30 днів з метою оцінки загальної стратегії схуднення або набору маси.

Розроблена модель прецедентів є базовим функціональним каркасом системи. Вона дозволяє систематизувати вимоги до інтерфейсу та алгоритмів, що є передумовою для подальшого проектування структури бази даних та логічних станів скінченного автомата програмного продукту. Повну схему варіантів використання та архітектурних зв'язків системи відображено на рис. 1.1.

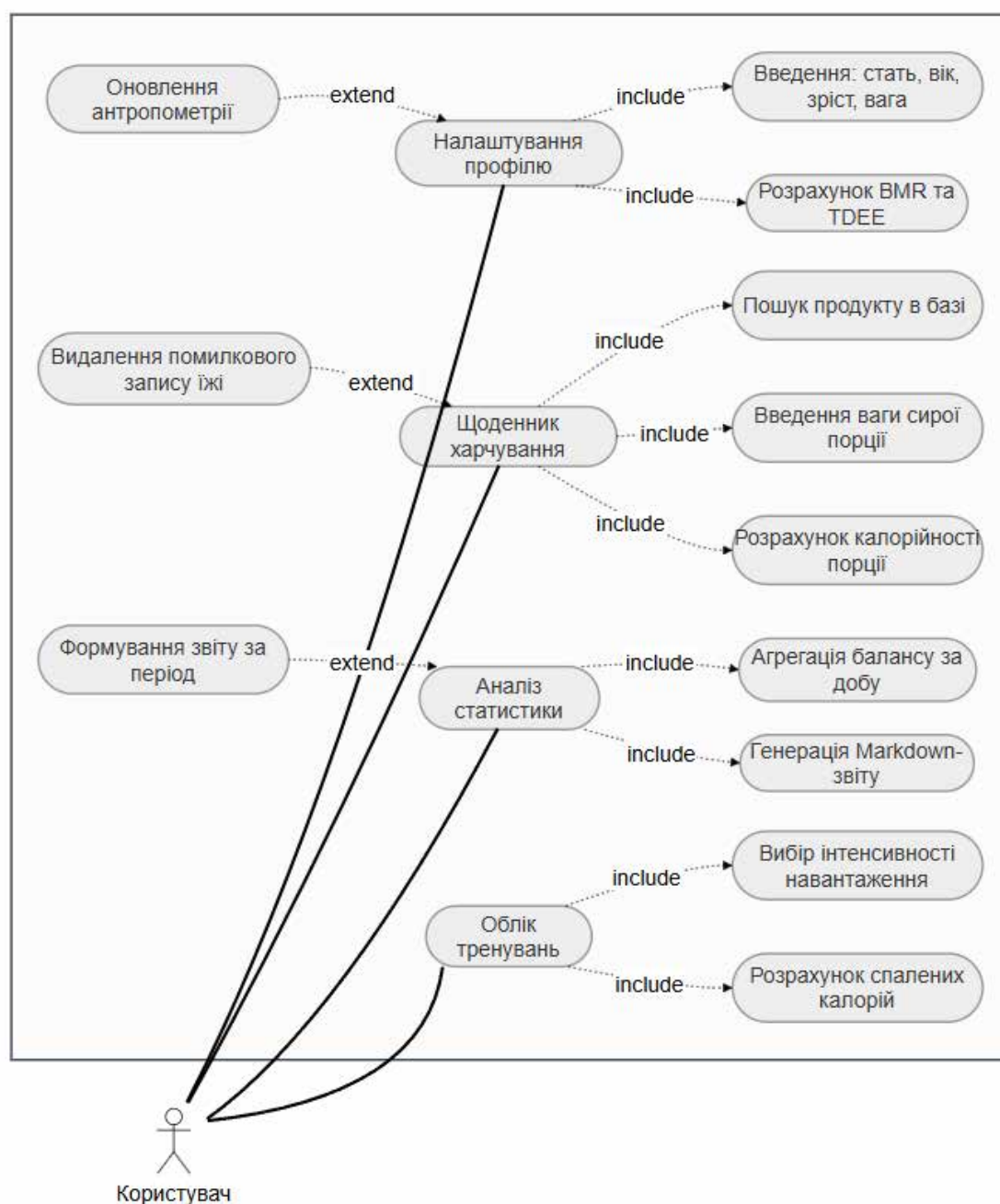


Рисунок 1.1 UML-діаграма прецедентів системи OptiBody

Згідно з вимогами до інженерії програмного забезпечення, система «OptiBody» повинна відповідати наступним критеріям:

Функціональні вимоги:

1. Реєстрація та модифікація профілю: можливість первинного створення аккаунту та подальшого оновлення фізичних параметрів у разі їх зміни.
2. Облік раціону: функціонал додавання продуктів із зазначенням їх ваги та автоматичним перерахунком калорійності.
3. Облік активності: можливість вибору типу навантаження та фіксації спаленої енергії.
4. Звітність: генерація щоденного підсумкового Markdown-документа, що містить структуровану інформацію про прогрес користувача за поточну добу.
5. CRUD-операції з логами: забезпечення можливості перегляду та видалення помилково введених записів для підтримки актуальності бази даних.

Нефункціональні вимоги:

1. Надійність та цілісність: гарантоване збереження даних у локальній базі SQLite навіть у разі перезавантаження серверної частини бота.
2. Продуктивність: час відгуку системи на запит користувача не повинен перевищувати 1-2 секунд (за умови стабільного Інтернет-з'єднання).
3. Доступність: робота системи у режимі 24/7/365 через асинхронну обробку запитів Telegram Bot API.
4. Масштабованість: архітектура системи повинна дозволяти легке розширення бази продуктів без необхідності зміни програмного коду ядра.

Кінцевим результатом функціонування системи є персоналізований статистичний звіт, що подається користувачу у формі структурованого повідомлення. Цей звіт слугує інструментом для прийняття рішень щодо подальшої стратегії харчування та фізичної активності на поточний день.

1.4. Постановка завдання на розробку інтелектуальної системи

На основі проведеного системного аналізу предметної області та критичного огляду існуючих аналогів, стає очевидною необхідність створення спеціалізованого програмного рішення, яке б поєднувало в собі наукову обґрунтованість розрахунків із максимальною ергономічністю інтерфейсу. Постановка завдання на розробку системи «OptiBody» передбачає трансформацію абстрактних потреб користувача у чітку сукупність технічних вимог, інформаційних моделей та алгоритмічних послідовностей.

2 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

Процес переходу від теоретичного аналізу до безпосередньої програмної реалізації інтелектуальної системи потребує детального етапу проєктування. На цьому етапі абстрактні вимоги трансформуються у конкретні моделі даних та архітектурні рішення. Центральне місце у цьому процесі посідає розробка логічної моделі даних, яка визначає структуру збереження інформації та забезпечує цілісність усіх бізнес-процесів системи «OptiBody».

2.1. Логічна модель у вигляді ER-діаграми

Логічне моделювання бази даних є критично важливим етапом інженерії програмного забезпечення, оскільки від правильності побудови схеми даних залежить продуктивність системи, відсутність аномалій при оновленні інформації та можливість подальшого розширення функціоналу. Для системи «OptiBody» було обрано реляційну модель, оскільки дані користувачів та їхні записи мають чітку структуру та зумовлені зв'язки.

Для візуалізації логічної структури було використано модель «сутність-зв'язок» (Entity-Relationship model, ER-модель) [10]. Вона дозволяє представити концептуальну схему бази даних у вигляді набору сутностей, їхніх атрибутів та типів взаємодії між ними. Логічну модель бази даних зображено на рис. 2.1. У розробленому програмному рішенні було виділено наступні ключові сутності.

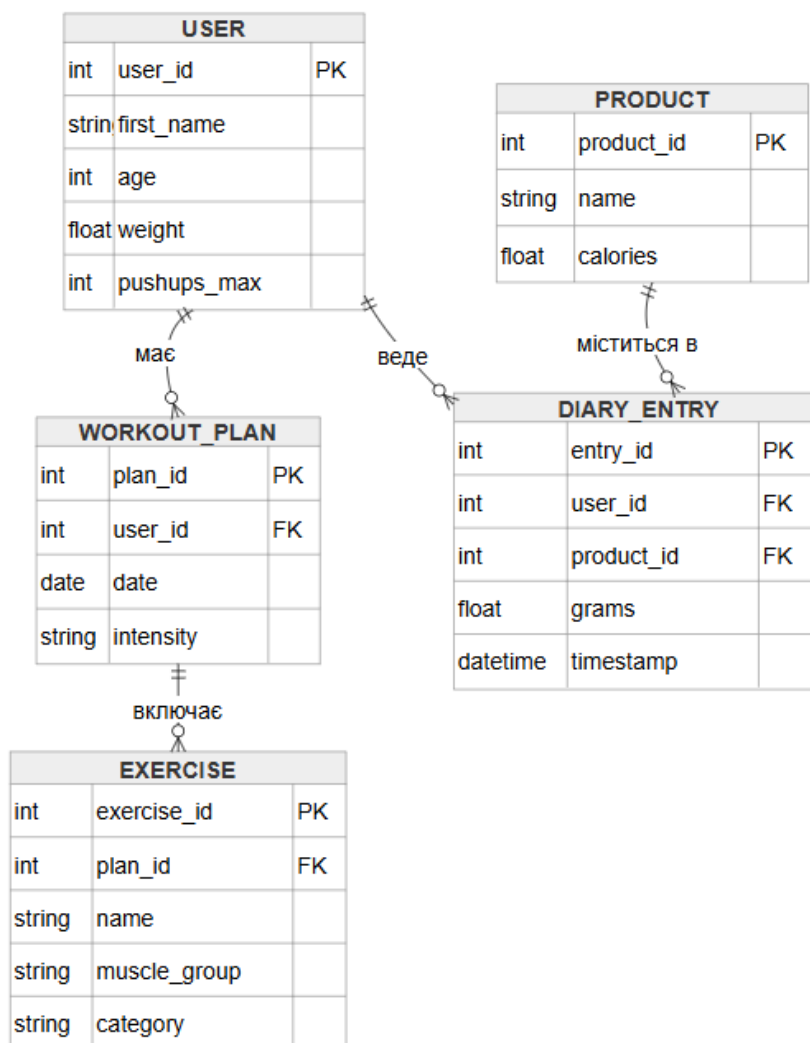


Рисунок 2.1 Логічна модель бази даних системи OptiBody у форматі ER-діаграми

Сутність «USER» (Користувач) – це центральна сутність системи, яка зберігає персональні параметри користувача, необхідні для автентифікації та роботи математичних алгоритмів розрахунку енергетичного балансу. Поля сутності:

- `user_id` (Primary Key) - унікальний ідентифікатор користувача в месенджері Telegram.
- `first_name` - ім'я користувача, яке використовується для персоналізації інтерфейсу.
- `age` - вік користувача у роках для розрахунку метаболічних показників.

- weight - актуальна вага користувача у кілограмах.
- pushups_max - індивідуальний показник фізичної підготовки (максимальна кількість віджимань) для планування тренувань.

Сутність «PRODUCT» (Продукт): Довідникова таблиця, що містить нормативно-довідкову інформацію про енергетичну цінність інгредієнтів. Поля сутності:

- product_id (Primary Key) - внутрішній унікальний ідентифікатор запису продукту.
- name - назва продукту харчування.
- calories - кількість калорій на 100 грамів продукту.

Сутність «DIARY_ENTRY» (Запис у щоденнику): Зберігає історію споживання їжі користувачем, пов'язуючи користувача з конкретним продуктом. Поля сутності:

- entry_id (Primary Key) - унікальний номер запису про прийом їжі.
- user_id (Foreign Key) - зовнішній ключ, що встановлює зв'язок із сутністю «USER».
- product_id (Foreign Key) - зовнішній ключ, що встановлює зв'язок із сутністю «PRODUCT» для отримання даних про калорійність.
- grams - фактична маса спожитої порції продукту у грамах.
- timestamp - дата та точний час внесення запису до щоденника.

Сутність «WORKOUT_PLAN» (План тренувань): Відображає заплановану або виконану фізичну активність користувача. Поля сутності:

- plan_id (Primary Key) - унікальний номер тренувального плану.
- user_id (Foreign Key) - зв'язок із сутністю «USER», що ініціював план.
- date - дата проведення тренування.
- intensity - рівень інтенсивності фізичного навантаження.

Сутність «EXERCISE» (Вправа): Деталізує конкретні вправи, що входять до складу тренувального плану. Поля сутності:

- exercise_id (Primary Key) - унікальний номер вправи.

- plan_id (Foreign Key) - зв'язок із сутністю «WORKOUT_PLAN».
- name - назва вправи.
- muscle_group - група м'язів, на яку спрямована вправа.
- category - категорія вправи (наприклад, силова чи кардіо).

Логічна модель була розроблена з урахуванням нормалізації даних до третьої нормальної форми [10, 17], що дозволяє уникнути надмірності та гарантує, що кожен атрибут залежить виключно від первинного ключа. Зв'язки між сутністю «USER» та журналами харчування («DIARY_ENTRY») і тренувань («WORKOUT_PLAN») мають тип «один-до-багатьох» (1:N), оскільки один користувач протягом часу створює множину записів про свою харчову поведінку та фізичну активність. Крім того, наявний зв'язок «один-до-багатьох» (1:N) між планом тренувань та переліком вправ, що дозволяє деталізувати структуру кожного окремого заняття.

Ця модель є фундаментом для подальшої фізичної реалізації бази даних.. Вона забезпечує стабільне зберігання історичних даних та дозволяє системі миттєво агрегувати статистику за будь-який обраний період, що є критично важливим для надання користувачу інтелектуального зворотного зв'язку щодо його способу життя.

2.2. Архітектура та структура програмного продукту

Проектування архітектури інтелектуальної системи «OptiBody» базується на принципах модульності, низької зв'язності та високої згуртованості [11, 16] . Оскільки розроблюваний продукт функціонує в асинхронному середовищі месенджера Telegram, його архітектура має забезпечувати не лише стабільну обробку запитів, а й прозору взаємодію між зовнішніми API, логікою бота та фізичним сховищем даних.

Для реалізації системи було обрано дворівневу архітектуру «клієнт-сервер» [7] з проміжною ланкою у вигляді Telegram Bot API. Така структура дозволяє делегувати завдання обробки мережових з'єднань та підтримки

інтерфейсу на боці месенджера, зосередивши основні потужності серверної частини на бізнес-логіці та управлінні даними.

Взаємодія компонентів системи «OptiBody» відбувається на трьох логічних рівнях (рис. 2.2):

1. Рівень представлення (Presentation Layer): Реалізується через офіційний додаток Telegram на пристрої користувача. Взаємодія здійснюється через графічні елементи (Inline та Reply клавіатури), що мінімізує потребу в ручному введенні команд та знижує імовірність помилок користувача.
2. Рівень бізнес-логіки (Application Layer): Це ядро системи, написане на мові Python з використанням фреймворку aiogram. На цьому рівні відбувається маршрутизація повідомлень, перевірка станів користувача через FSM та виконання математичних розрахунків за формулою Міффліна-Сан Жеора.
3. Рівень даних (Data Access Layer): Представлений базою даних SQLite та об'єктно-реляційним відображенням SQLAlchemy. Цей рівень відповідає за збереження транзакційної цілісності логів харчування та тренувань.

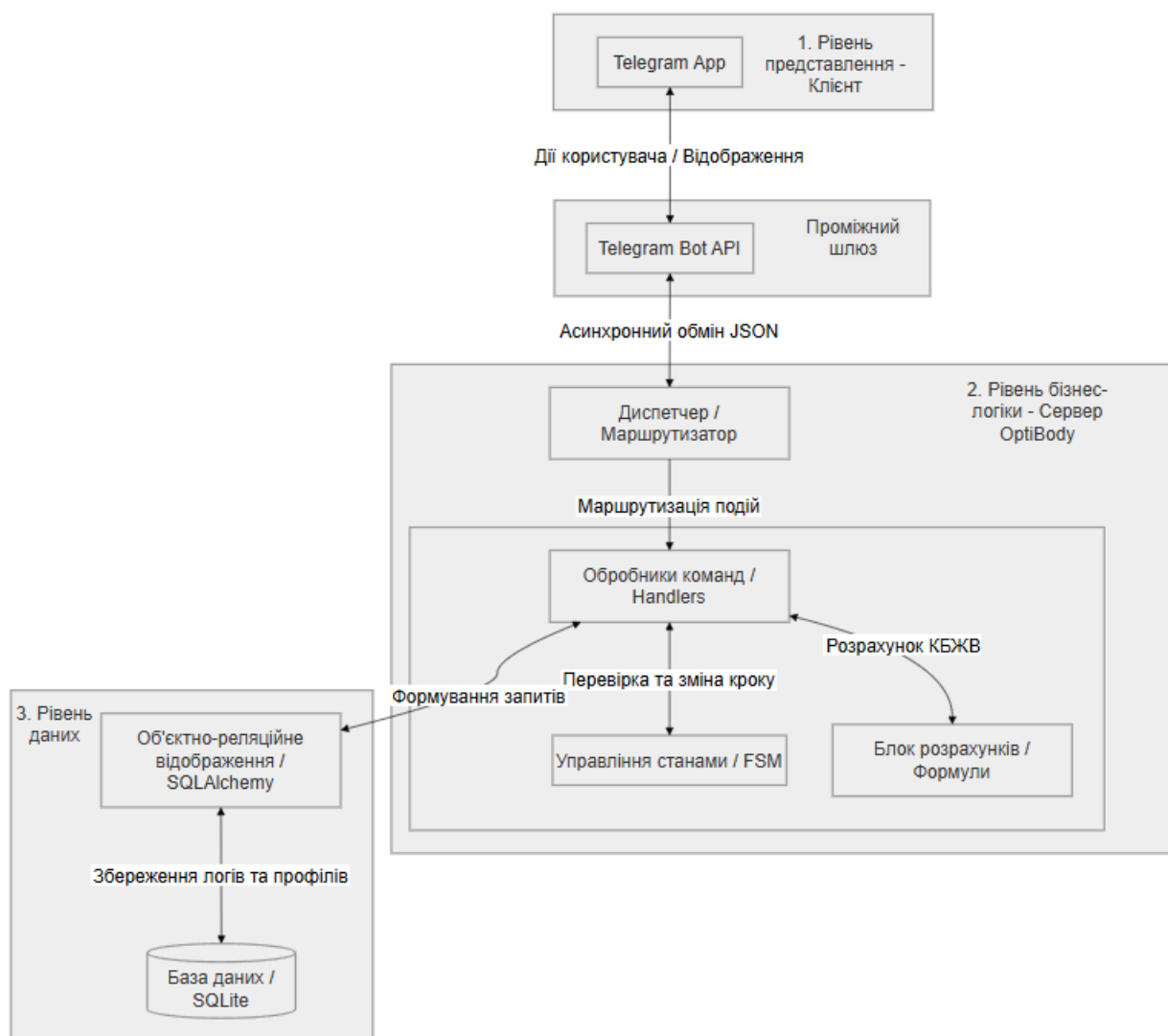


Рисунок 2.2 Архітектурна схема взаємодії компонентів системи OptiBody

На етапі об'єктно-орієнтованого аналізу та проєктування архітектури програмного забезпечення критично важливим є перехід від концептуальних моделей даних до безпосереднього опису статичної структури вихідного коду системи. Для вирішення цього завдання в уніфікованій мові моделювання UML використовується діаграма класів [9] (рис. 2.3). Вона дозволяє візуалізувати внутрішню архітектуру програмного продукту, визначити ключові класи, їхні внутрішні атрибути, методи обробки інформації, а також логічні зв'язки та залежності між окремими компонентами системи «OptiBody».

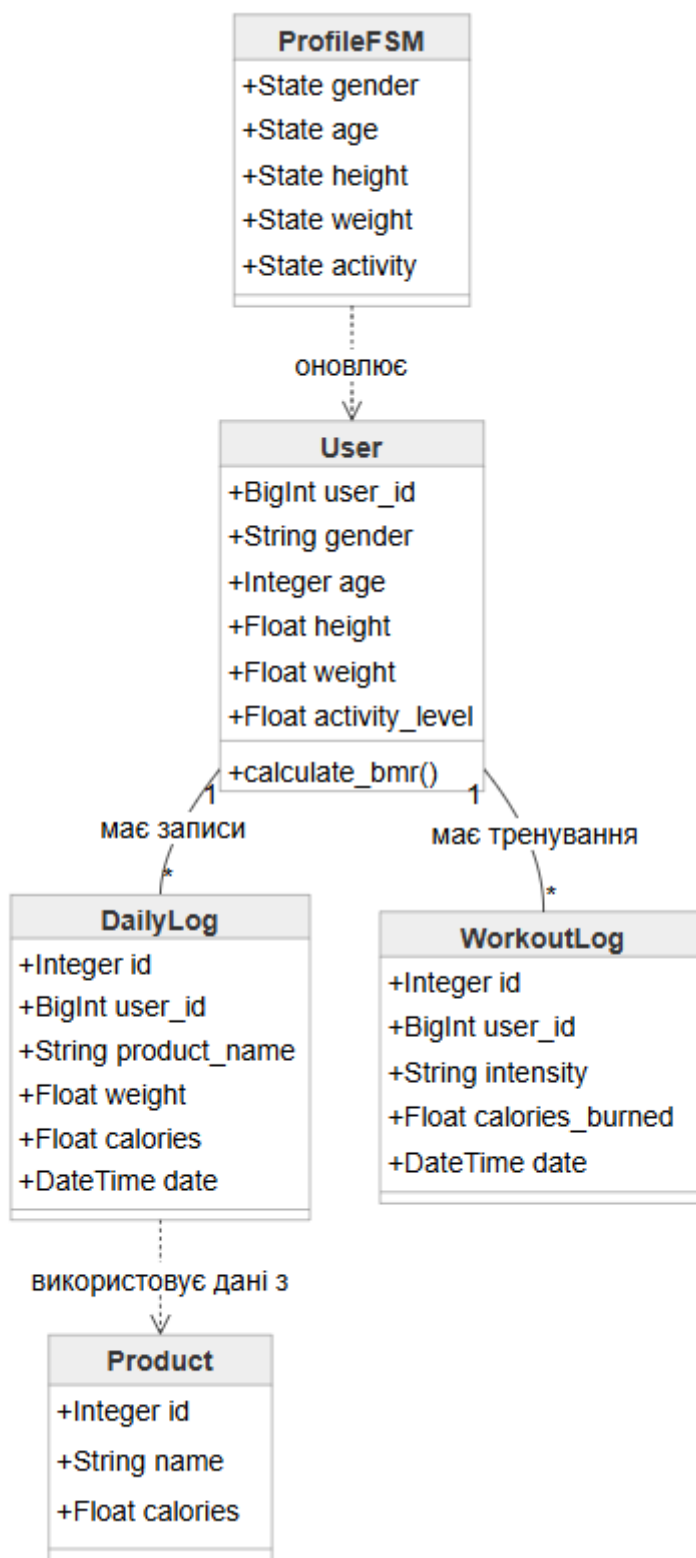


Рисунок 2.3 UML-діаграма класів об'єктної моделі системи OptiBody

У проєкті, що розробляється, побудова діаграми класів тісно пов'язана з концепцією об'єктно-реляційного відображення (ORM), яка реалізована за допомогою інструменту SQLAlchemy [4]. Це зумовлює специфіку архітектури,

де класи рівня даних безпосередньо відображають сутності реляційного сховища, але водночас виступають як повноцінні програмні об'єкти в середовищі виконання Python.

На розробленій діаграмі класів виділено три фундаментальні логічні блоки:

1. Класи моделей даних (ORM Models), що інкапсулюють стан об'єктів User, Product, DailyLog та WorkoutLog. Кожен клас відповідає за окрему сутність бази даних, забезпечуючи типізований доступ до атрибутів користувача, інгредієнтів харчування та записів активності.
2. Класи управління станами (FSM States), представлені класом ProfileFSM, що відповідають за логіку діалогів та послідовність кроків користувача. Цей модуль гарантує цілісність збору даних під час ініціалізації профілю.
3. Класи-маршрутизатори (Routers), які реалізують прикладну бізнес-логіку обробки вхідних запитів від Telegram Bot API.

Важливим елементом архітектури є відношення залежності між класом DailyLog та класом Product. Оскільки системі необхідно оперувати актуальними даними про калорійність, журнал харчування динамічно звертається до довідника продуктів. Такий підхід забезпечує єдине джерело істини для нормативно-довідкових даних та дозволяє уникнути надлишкового дублювання інформації в межах реляційної моделі. Зв'язки типу асоціації між класом User та журналами активності мають множинність «один-до-багатьох» (1:N), що відображає можливість накопичення необмеженої кількості записів користувачем протягом усього періоду експлуатації системи.

Завдяки чіткому розділенню відповідальності між класами моделей даних та класами керування станами досягається низька зв'язність модулів системи, що суттєво підвищує надійність коду, спрощує процес модульного тестування (Unit Testing) та забезпечує гнучкі можливості для подальшого масштабування програмного комплексу.

2.3. Фізична модель бази даних

Фізичне моделювання є завершальним етапом проєктування структури збереження даних, на якому абстрактна логічна схема адаптується до специфіки конкретної системи управління базами даних. Для інтелектуальної системи «OptiBody» як фізичне середовище збереження було обрано реляційну СКБД SQLite [3]. Вибір зумовлений її архітектурною легкістю, високою швидкістю виконання запитів до невеликих масивів даних та відсутністю необхідності розгортання окремого серверного процесу, що є ідеальним для асинхронних Telegram-ботів.

На етапі фізичного проєктування було визначено точні типи даних для кожного атрибута, встановлено обмеження цілісності (NOT NULL, UNIQUE) та визначено механізми забезпечення зв'язків між таблицями через первинні (Primary Key) та зовнішні ключі (Foreign Key). Фізична реалізація бази даних у файлі bot.db дозволяє системі миттєво обробляти транзакції та гарантує надійність збереження персональної статистики користувача.

Нижче наведено детальний опис структури основних фізичних таблиць системи «OptiBody» в таблицях 2.1, 2.2, 2.3, 2.4.

Таблиця 2.1

Опис структури таблиці users

Назва поля	Тип даних	Обмеження	Опис
user_id	BIGINT	PRIMARY KEY	Унікальний ID користувача в Telegram
gender	VARCHAR(10)	NOT NULL	Стать (male / female)
age	INTEGER	NOT NULL	Вік користувача (повних років)
height	REAL	NOT NULL	Зріст у сантиметрах
weight	REAL	NOT NULL	Актуальна вага в кілограмах
activity	VARCHAR(50)	NOT NULL	Коефіцієнт фізичної активності (PAL)
bmr	REAL	—	Розрахований базовий метаболізм
target_calories	REAL	—	Денна норма з урахуванням активності

Таблиця 2.2

Опис структури таблиці products

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AI	Унікальний номер продукту
name	VARCHAR(255)	NOT NULL, UNIQUE	Назва продукту харчування
calories	REAL	NOT NULL	Калорійність на 100г (сира вага)

Таблиця 2.3

Опис структури таблиці daily_logs

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AI	Унікальний номер запису
user_id	BIGINT	FOREIGN KEY	Зв'язок із таблицею users
product_name	VARCHAR(255)	NOT NULL	Назва спожитого продукту
weight	REAL	NOT NULL	Фактична маса порції
calories	REAL	NOT NULL	Розраховані калорії для цієї ваги
date	TEXT/DATE	NOT NULL	Дата та час внесення запису

Таблиця 2.4

Опис структури таблиці workout_logs

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AI	Унікальний номер запису тренування
user_id	BIGINT	FOREIGN KEY	Зв'язок із таблицею users
intensity	VARCHAR(50)	NOT NULL	Обраний рівень інтенсивності навантаження
calories_burned	REAL	NOT NULL	Розрахована кількість спалених калорій
date	TEXT	NOT NULL	Дата та час фіксації активності

Для забезпечення цілісності даних на фізичному рівні було впроваджено каскадне видалення записів (ON DELETE CASCADE). Це означає, що у разі видалення профілю користувача з таблиці users (наприклад, через функцію

«Видалити всі дані» у меню налаштувань), усі пов’язані з ним записи у журналах харчування та тренувань будуть автоматично видалені системою, що запобігає появі «сміттєвих» даних та підтримує оптимальний обсяг файлу бази даних.

Крім того, для підвищення продуктивності пошуку продуктів за назвою було створено індекси на полі name у таблиці products. Це критично важливо, оскільки база містить понад 180 найменувань, і миттєвий відгук бота під час фільтрації інгредієнтів є пріоритетною вимогою до UX.

Фізична модель, схема якої проілюстрована на рис. 2.4, повністю реалізує бізнес-вимоги до проєкту, забезпечуючи надійне та структуроване сховище для всіх інформаційних потоків. Використання SQLite як вбудованої бази даних дозволяє системі працювати автономно на будь-якому серверному середовищі без потреби у складних конфігураціях мережесх з’єднань для СКБД.

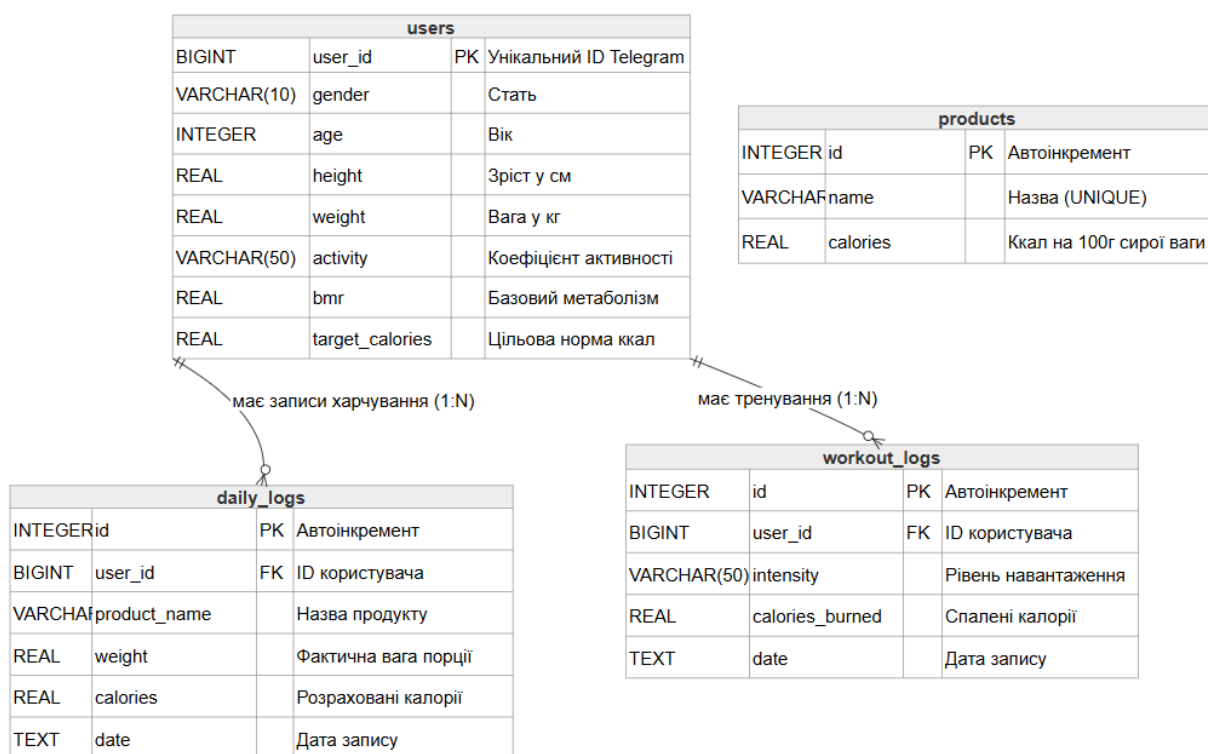


Рисунок 2.4 Фізична схема бази даних SQLite інтелектуальної системи OptiBody

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Третій розділ присвячений детальному опису практичного етапу розробки інтелектуальної системи «OptiBody». На цьому етапі архітектурні рішення, обрані у попередньому розділі, втілюються у вигляді конкретних програмних модулів, алгоритмів та інтерфейсних елементів. Особлива увага приділяється реалізації асинхронної логіки взаємодії, побудові гнучкої системи управління станами та інтеграції математичних моделей розрахунку метаболізму в програмний код.

3.1. Структура програмного продукту

Для забезпечення високої якості програмного забезпечення, зручності його тестування та можливості подальшої підтримки, структура проєкту «OptiBody» була спроектована з дотриманням принципів модульності та розділення відповідальності [6, 11]. Весь програмний комплекс розбитий на логічно незалежні блоки, кожен з яких відповідає за конкретний аспект функціонування системи: від низькорівневої взаємодії з базою даних до високорівневої обробки команд користувача.

Такий підхід дозволяє уникнути створення «громіздкого» коду та забезпечує легку масштабованість системи. У разі потреби розширення функціоналу (наприклад, додавання нових типів активності), розробнику достатньо внести зміни у відповідний модуль, не зачіпаючи при цьому ядро системи.

Нижче наведено детальний опис та таблицю 3.1 файлової структури та функціонального призначення основних компонентів проєкту:

1. Файл `main.py` (Точка входу): Це головний файл програми, який відповідає за ініціалізацію всіх системних компонентів. У ньому відбувається створення об'єктів бота та диспетчера, конфігурація

логування подій та реєстрація всіх обробників повідомлень (роутерів). Саме тут запускається асинхронний цикл опитування серверів Telegram (Polling), який забезпечує безперервну роботу системи.

2. Директорія database/ (Рівень даних): Даний модуль інкапсулює всю логіку роботи з фізичним сховищем SQLite.
 - models.py: містить опис класів SQLAlchemy, які відповідають таблицям у базі даних (User, Product, DailyLog). Використання декларативного стилю дозволяє системі автоматично генерувати необхідні SQL-запити на основі об'єктів Python.
 - requests.py: включає набір асинхронних функцій для виконання CRUD-операцій. Це дозволяє іншим частинам програми отримувати або зберігати дані, не вникаючи в деталі реалізації SQL-запитів.
3. Директорія handlers/ (Логіка взаємодії): Найбільший за обсягом модуль, що містить обробники подій.
 - register.py: логіка первинного опитування користувача (анкета) та збереження параметрів профілю.
 - products.py: реалізація пошуку продуктів у довіднику та додавання записів про споживання їжі за сировою вагою.
 - workout.py: обробка даних про фізичну активність та розрахунок витрат енергії.
 - reports.py: алгоритми агрегації даних за добу та формування підсумкових Markdown-звітів.
4. Директорія utils/ (Допоміжні інструменти):
 - states.py: опис класів станів для FSM (Finite State Machine). Кожна група станів відповідає за певний логічний ланцюжок діалогу, що гарантує послідовність дій користувача.
 - calculations.py: чиста математична реалізація формули Міффіна-Сан Жеора [12], ізольована від логіки бота для забезпечення точності та легкості тестування формул.

Таблиця 3.1

Склад програмних модулів та файлова структура системи OptiBody

Назва файлу / директорії	Функціональне призначення
main.py	Точка входу в програму: ініціалізація бота, диспетчера, реєстрація роутерів та запуск циклу асинхронного опитування.
database/models.py	Опис структури реляційних таблиць бази даних за допомогою ORM SQLAlchemy (User, Product, DailyLog).
database/requests.py	Реалізація набору асинхронних функцій для виконання CRUD-операцій та запитів до SQLite.
handlers/register.py	Модуль обробки процесу реєстрації користувача: збір антропометричних даних та налаштування профілю.
handlers/products.py	Реалізація логіки пошуку інгредієнтів у довіднику та додавання записів про споживання їжі за сировою вагою.
handlers/workout.py	Обробка даних про фізичну активність: вибір типу вправи та розрахунок спаленої енергії.
handlers/reports.py	Алгоритм агрегації добової статистики та генерація фінального звіту у форматі Markdown.
utils/states.py	Визначення ієрархії станів для механізму FSM (Finite State Machine), що керує діалогами.
utils/calculations.py	Програмна реалізація математичного алгоритму Міффліна-Сан Жеора для розрахунку метаболізму.
requirements.txt	Список зовнішніх бібліотек та залежностей, необхідних для розгортання та роботи системи.
.env	Конфігураційний файл для захищеного зберігання токена Telegram Bot API та налаштувань середовища.

Вибрана ієрархічна структура дозволяє ефективно використовувати асинхронні можливості бібліотеки aiogram 3.x. Завдяки використанню механізму Router, диспетчер може швидко розподіляти вхідні повідомлення між відповідними модулями, що забезпечує мінімальний час відгуку системи навіть при значній кількості активних сесій користувачів. Така організація коду повністю відповідає вимогам до сучасних інтелектуальних систем і закладає фундамент для надійної та продуктивної роботи «OptiBody».

3.2. Алгоритмічне забезпечення системи

Ефективність будь-якої інтелектуальної системи визначається якістю та надійністю її алгоритмічного забезпечення. У проєкті «OptiBody» алгоритмічний рівень виконує роль сполучної ланки між вхідними даними користувача та кінцевими статистичними результатами. Оскільки взаємодія з користувачем відбувається в асинхронному чаті, алгоритми мають бути спроектовані таким чином, щоб забезпечувати лінійність процесів (наприклад, заповнення анкети) та водночас дозволяти гнучке перемикання між функціями (наприклад, відміна введення або швидкий перехід до звітів).

Алгоритмічне забезпечення системи «OptiBody» можна розділити на два основні типи: керуючі алгоритми (логіка діалогів) та обчислювальні алгоритми (математична обробка даних).

Для забезпечення цілісності процесу збору даних було використано математичний апарат скінченних автоматів (FSM). Це дозволяє системі суворо контролювати кожен крок взаємодії. Наприклад, під час реєстрації користувача алгоритм не дозволить перейти до введення ваги, доки не буде успішно отримано та валідовано дані про зріст.

Основні ланцюжки станів у системі:

- Реєстраційний автомат: Керує послідовністю «Стать - Вік - Зріст - Вага - Рівень активності». На кожному кроці алгоритм виконує валідацію введених значень (перевірка на числовий формат та входження у фізіологічно можливі межі).
- Автомат обліку харчування: Реалізує пошук продукту, перевірку його наявності в базі та запит маси порції.
- Автомат тренувань: Забезпечує вибір типу навантаження та фіксацію витраченої енергії.

Використання FSM гарантує, що контекст розмови не буде втрачено, навіть якщо декілька користувачів звертаються до бота одночасно, оскільки стан

кожного користувача ізольовано зберігається в оперативній пам'яті сервера (або в сховищі Redis/MemoryStorage).

Для забезпечення цілісності процесу збору даних та гарантування послідовності дій користувача в асинхронному середовищі було застосовано математичний апарат скінченних автоматів (Finite State Machine, FSM). Це архітектурне рішення дозволяє системі суворо контролювати кожен крок взаємодії: алгоритм не дозволяє перейти до введення наступного параметра (наприклад, ваги), доки поточний (зріст) не пройде успішну валідацію.

Діаграма станів (рис. 3.1) наочно демонструє життєвий цикл автомата реєстрації користувача в системі «OptiBody». Перехід між станами (WaitingForGender, WaitingForAge тощо) ініціюється відповідними подіями від користувача після успішної перевірки введених даних на відповідність числовим типам та фізіологічним лімітам [9].

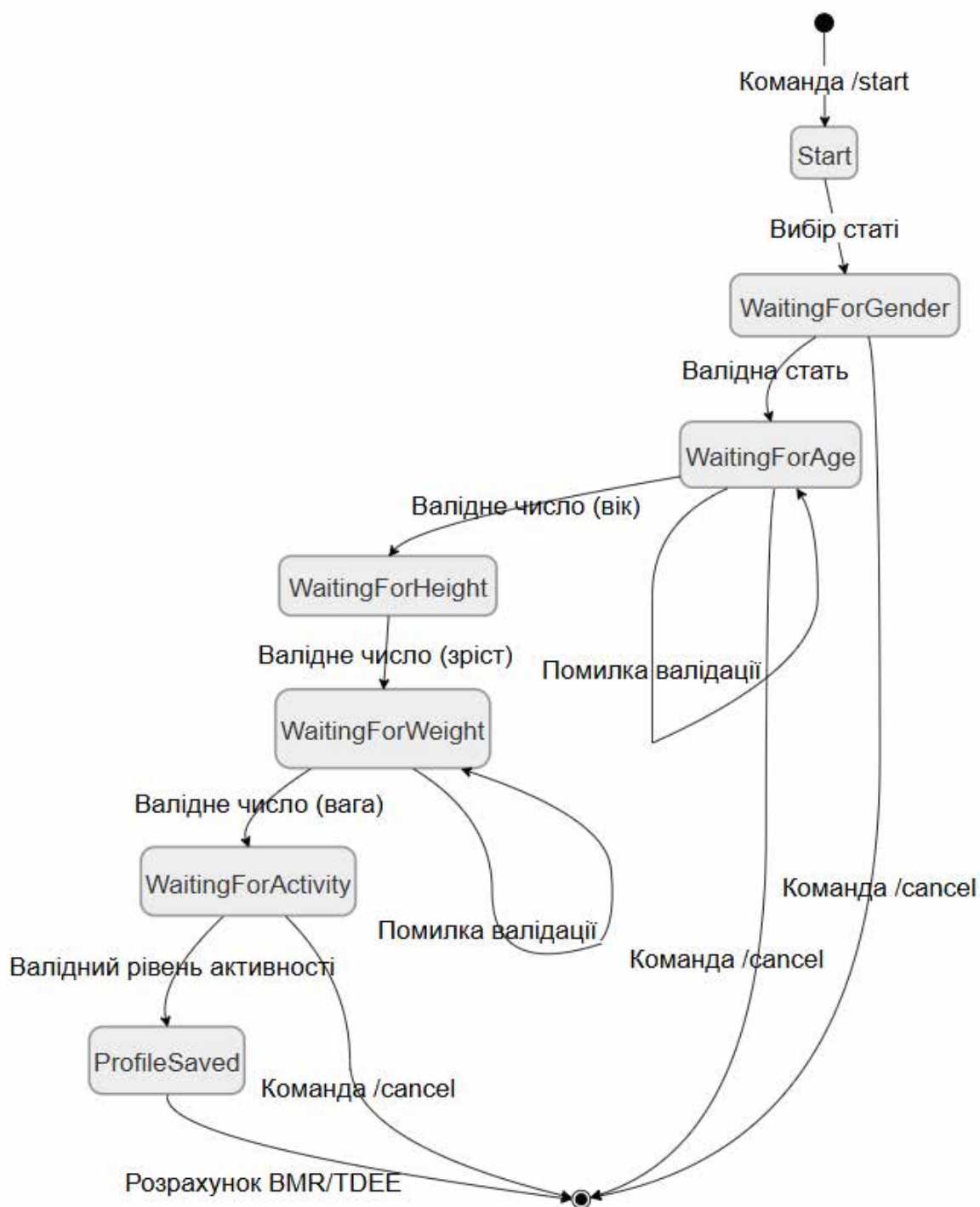


Рисунок 3.1 UML-діаграма станів процесу реєстрації користувача (FSM)

Критичними елементами цієї моделі є:

1. Стани (States): визначають етап, на якому знаходиться користувач під час заповнення анкети.

2. Переходи (Transitions): відбуваються при отриманні валідних даних; у разі помилки валідації система залишається в поточному стані та надсилає повідомлення про необхідність повторного введення.
3. Точки виходу: з будь-якого стану користувач має можливість перервати процес реєстрації через команду скасування, що автоматично очищає контекст FSM та повертає бота в початковий стан.

Використання FSM гарантує, що контекст розмови не буде втрачено, навіть якщо декілька користувачів звертаються до бота одночасно, оскільки стан кожного з них ізолювано зберігається в пам'яті сервера.

Центральним обчислювальним компонентом системи є алгоритм визначення добової потреби в енергії, заснований на формулі Міффіна-Сан Жеора. Алгоритм працює за наступним ієрархічним принципом:

1. Розрахунок базового метаболізму (BMR):

Для чоловіків:

$$BMR = (10 \times \text{вага}) + (6.25 \times \text{зріст}) - (5 \times \text{вік}) + 5$$

Для жінок:

$$BMR = (10 \times \text{вага}) + (6.25 \times \text{зріст}) - (5 \times \text{вік}) - 161$$

2. Визначення загальних добових витрат енергії (TDEE):

$$TDEE = BMR \times PAL$$

3. Алгоритм розрахунку калорійності порції:

$$E_{total} = \sum \frac{(C_{product} \times M_{raw})}{100}$$

Де $C_{product}$ - калорійність на 100г сирого продукту, M_{raw} - фактична маса інгредієнта. Даний підхід мінімізує похибку, пов'язану з термічною обробкою їжі.

На рис 3.2 зображено схему алгоритму обробки даних про харчування в системі OptiBody. Ця схема описує покрокове додавання запису в щоденник харчування.

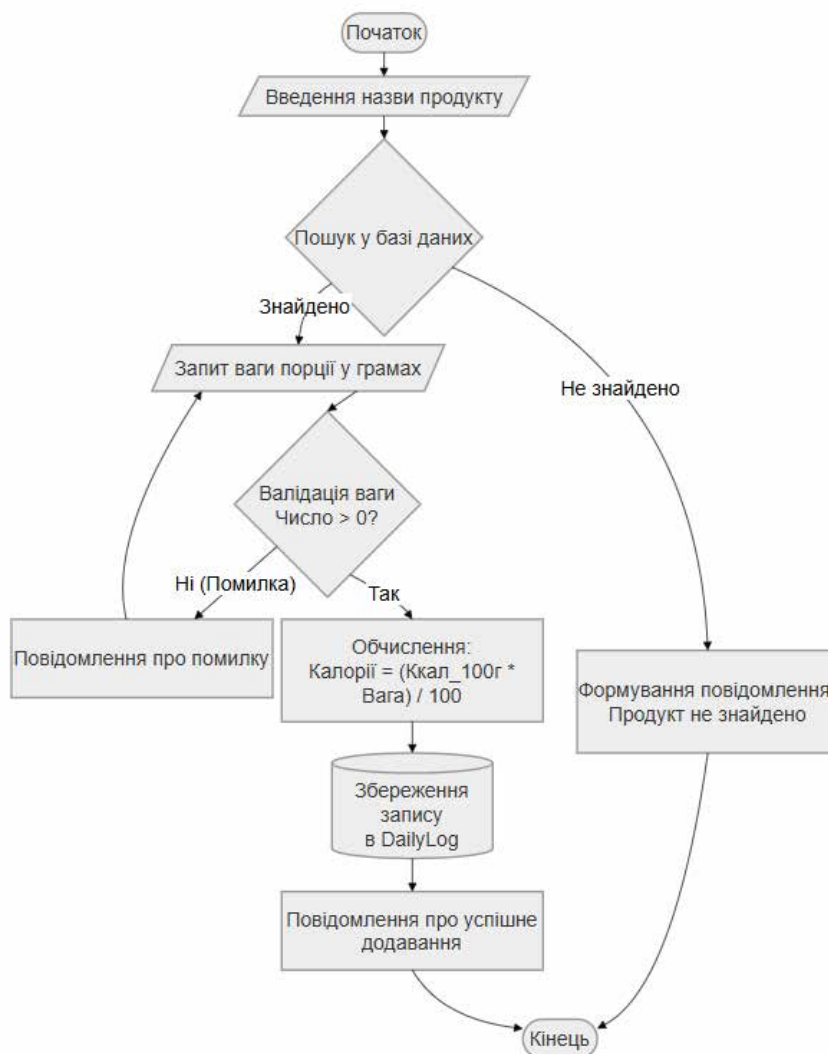


Рисунок 3.2 Схема алгоритму обробки даних про харчування в системі OptiBody

Програмна реалізація звітності базується на алгоритмі агрегації часових рядів. Система виконує SQL-запит до таблиць `daily_logs` та `workout_logs` за поточну дату, підсумовує значення калорій та формує Markdown-структуру повідомлення. Це дозволяє користувачу отримати цілісну картину енергообміну без необхідності самотійного аналізу окремих записів.

Одним із найважливіших аспектів проєктування розподілених програмних комплексів є моделювання динаміки взаємодії між їхніми внутрішніми та зовнішніми компонентами в часі. З цією метою в рамках об'єктно-орієнтованого підходу застосовується UML-діаграма послідовності (Sequence Diagram) [9]. На відміну від діаграми класів, яка демонструє виключно статичну архітектуру,

діаграма послідовності ілюструє життєвий цикл конкретного бізнес-процесу: від моменту ініціації запиту користувачем до збереження даних та отримання фінального результату.

У контексті розробки системи «OptiBody», яка функціонує на базі месенджера Telegram, критично важливо візуалізувати асинхронну природу обміну повідомленнями. На рис. 3.3 наведено діаграму послідовності, яка моделює один із ключових та найчастіше використовуваних прецедентів системи — процес обліку спожитих нутрієнтів (додавання запису до щоденника харчування).

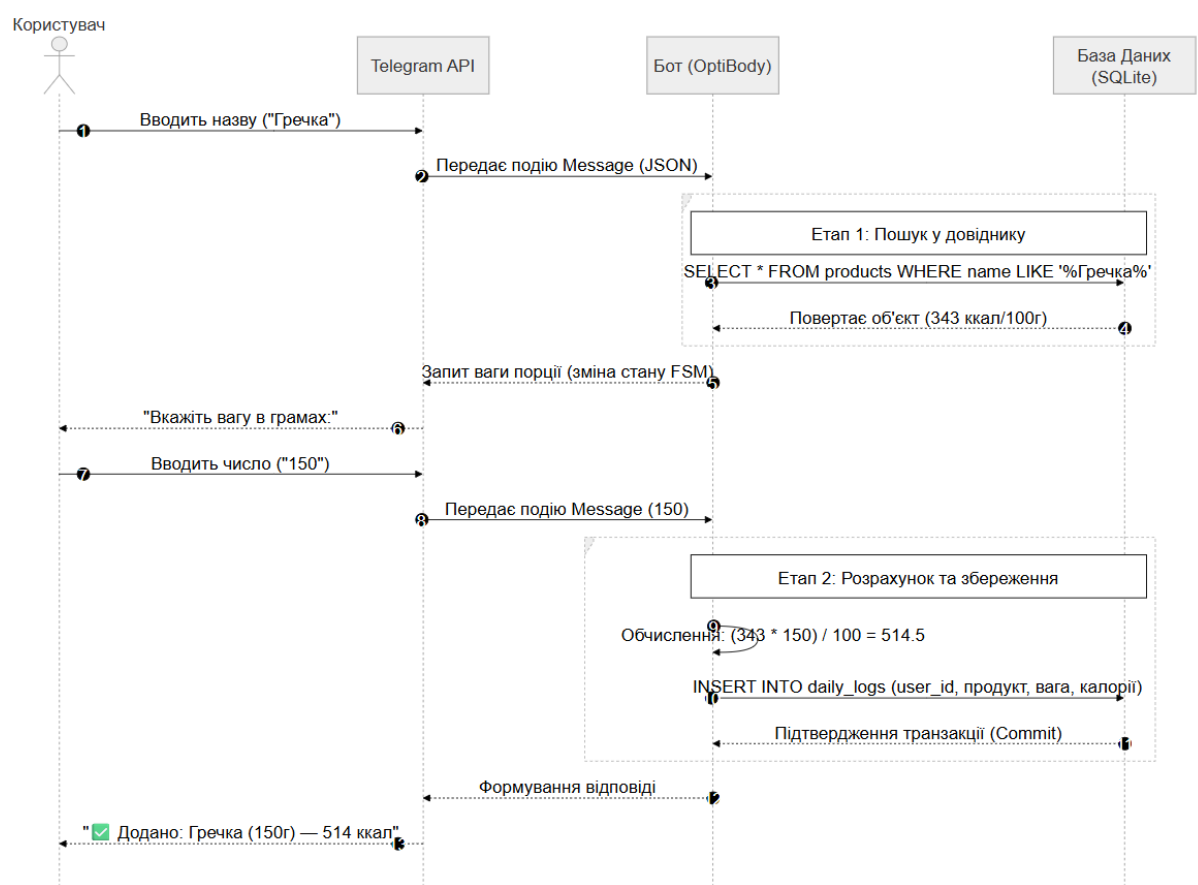


Рисунок 3.3 UML-діаграма послідовності додавання запису до щоденника харчування

На діаграмі виділено чотири основні лінії життя (Lifelines), які репрезентують активних учасників інформаційного обміну:

1. Користувач (Actor): ініціатор процесу, який взаємодіє з системою через графічний інтерфейс свого пристрою.
2. Telegram Bot API: зовнішній хмарний шлюз, який виконує роль маршрутизатора між клієнтським додатком та сервером розробника.
3. Бот OptiBody (Application Layer): серверне ядро системи на базі фреймворку aiogram, що інкапсулює бізнес-логіку, управління станами (FSM) та математичні розрахунки.
4. База даних SQLite (Data Layer): фізичне сховище, яке відповідає за пошук нормативно-довідкової інформації та транзакційне збереження логів.

Процес взаємодії розгортається у суворій хронологічній послідовності. Життєвий цикл розпочинається з відправки користувачем текстового запиту (назви продукту). Telegram API інкапсулює цей запит у формат JSON та асинхронно передає його на сервер бота. Отримавши подію, ядро системи формує SQL-запит до бази даних (SELECT) для пошуку відповідної позиції у довіднику продуктів.

У разі успішного знаходження інгредієнта, база даних повертає його еталонну калорійність на 100 грамів. Далі бот ініціює зустрічний запит до користувача з вимогою вказати точну масу порції у грамах. Після отримання числового значення від клієнта, програмний модуль виконує внутрішню операцію (Self-Message) - розрахунок фактичної енергетичної цінності за закладеним математичним алгоритмом (множення базової калорійності на введену вагу з подальшим діленням на 100).

Завершальним етапом транзакції є звернення ядра системи до бази даних із командою INSERT для збереження розрахованого лога у таблиці daily_logs. Після підтвердження успішного запису СКБД, бот формує фінальне повідомлення зі звітом про додану страву та транслює його через Telegram API на екран користувача.

Така модель наочно доводить, що обрана дворівнева архітектура з проміжним шлюзом забезпечує ефективне делегування відповідальності: месенджер бере на себе витрати на підтримку мережових з'єднань, тоді як серверна частина бота займається виключно корисною бізнес-логікою.

3.3. Опис інтерфейсу користувача та взаємодії

Проектування інтерфейсу користувача для інтелектуальних чат-ботів у 2026 році базується на парадигмі розмовного інтерфейсу. На відміну від традиційних графічних інтерфейсів, інтерфейс «OptiBody» інтегрований безпосередньо у середовище месенджера Telegram. Це дозволяє забезпечити безшовний досвід користувача, де процес моніторингу здоров'я виглядає як звичайна переписка, що значно знижує психологічний бар'єр перед веденням щоденних записів.

Фізична архітектура програмного забезпечення визначає спосіб розміщення компонентів системи в середовищі виконання, забезпечуючи надійну взаємодію між клієнтською частиною та сервером. Для візуалізації топології розгортання системи «OptiBody» було побудовано діаграму розгортання (Deployment Diagram), наведену на рис. 3.4.

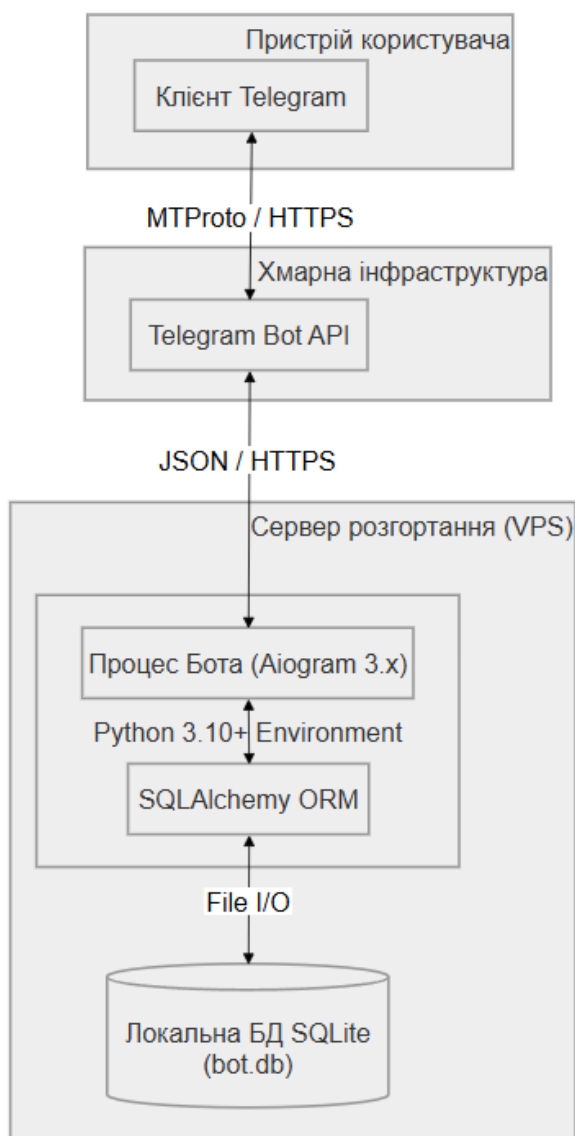


Рисунок 3.4 UML-діаграма розгортання системи OptiBody

На діаграмі виділено три основні вузли, що забезпечують безперебійне функціонування комплексу:

1. Пристрій користувача: на цьому вузлі функціонує клієнт месенджера Telegram. Він виступає як точка доступу до сервісу, забезпечуючи інтерфейс користувача та мережеву взаємодію за протоколом MTPROTO/HTTPS.
2. Хмарна інфраструктура: проміжний вузол, який інкапсулює логіку маршрутизації повідомлень. Він забезпечує асинхронну доставку даних між клієнтом та сервером, нівелюючи проблеми низької якості мережевого з'єднання на стороні користувача.

3. Сервер розгортання: ключовий обчислювальний вузол, на якому розгорнуто програмне ядро системи. Середовище виконання базується на інтерпретаторі Python 3.10+ та включає наступні компоненти:

- Процес бота: асинхронний застосунок на базі фреймворку aiogram 3.x, що керує бізнес-логікою.
- ORM SQLAlchemy: шар абстракції, що відповідає за об'єктно-реляційне відображення даних.
- Локальна СКБД SQLite: файлова база даних (bot.db), що забезпечує збереження станів користувачів, логів харчування та тренувань.

Взаємодія між вузлами реалізована через безпечні HTTPS-канали, що гарантує цілісність даних при передачі. Використання VPS-сервера для розміщення ядра системи дозволяє забезпечити цілодобову доступність бота, що є обов'язковою вимогою для систем моніторингу здорового способу життя. Такий підхід до розгортання забезпечує високу масштабованість проєкту та низьку затримку під час обробки транзакцій користувачів.

При розробці «OptiBody» було дотримано принципу «мобільного мінімалізму». Інтерфейс спроектовано таким чином, щоб користувач міг виконати будь-яку цільову дію за мінімальну кількість кліків. Основні акценти в UI зроблені на:

- Швидкість відгуку: Асинхронна архітектура дозволяє миттєво оновлювати елементи інтерфейсу.
- Контекстність: Бот надає лише ті інструменти керування, які необхідні в конкретний момент.
- Відсутність візуального шуму: Свідома відмова від графіків та зображень на користь тексту.

У системі «OptiBody» реалізовано три рівні взаємодії з користувачем:

1. Головне меню (ReplyKeyboardMarkup): Постійно доступна клавіатура в нижній частині екрана.
2. Інтерактивні кнопки (InlineKeyboardMarkup): Застосовуються для локальних дій, прив'язаних до конкретних повідомлень.
3. Текстове введення з валідацією: Використовується для введення назв продуктів та їхньої ваги.

Форматування інформації реалізовано за допомогою розмітки Markdown. Це дозволяє створювати візуальну ієрархію в тексті: жирний шрифт для виділення числових значень (калорії, вага), моноширинний шрифт для списків продуктів та емодзі як візуальні маркери розділів. Таблиця 3.1 показує функціональні елементи інтерфейсу.

Таблиця 3.1

Опис функціональних елементів інтерфейсу системи OptiBody

Елемент меню / Кнопка	Функціональне призначення
 Додати їжу	Запуск сценарію пошуку продукту в базі та фіксації ваги порції.
 Тренування	Відкриття меню вибору інтенсивності навантаження та запису активності.
 Звіт за сьогодні	Генерація та вивід підсумкового щоденного звіту з балансом калорій.
 Мій профіль	Відображення поточних антропометричних даних та розрахованої норми BMR/TDEE.
 Оновити дані	Запуск FSM-автомата для перереєстрації фізичних показників користувача.
 Видалити запис	Inline-кнопка в журналі для миттєвого видалення помилково введеної позиції.

3.4. Інструкція користувача

Ефективність впровадження інтелектуальної системи «OptiBody» безпосередньо залежить від розуміння користувачем принципів її функціонування та дотримання встановлених протоколів введення даних. Дана інструкція, структурована в таблиці 3.2 розроблена для забезпечення швидкої адаптації користувача до розмовного інтерфейсу бота та мінімізації помилок під час щоденного моніторингу енергетичного балансу.

Таблиця 3.2

Регламент взаємодії користувача з інтелектуальною системою OptiBody

Команда / Дія користувача	Очікуваний результат від системи
/start	Запуск бота, ініціалізація сесії та початок анкетування.
Введення "Гречка"	Пошук у БД та вивід списку знайдених інгредієнтів.
Введення ваги "150"	Розрахунок калорійності та збереження запису в щоденник.
Кнопка "  Звіт"	Вивід структурованого Markdown-повідомлення з балансом за добу.
Кнопка "  Оновити"	Скидання поточних параметрів профілю та перехід до нової анкети.
Команда /help	Вивід короткої довідки про основні функції та можливості бота.

Для стабільної роботи з системою «OptiBody» користувачу необхідно мати пристрій (смартфон, планшет або персональний комп'ютер) із встановленим клієнтом месенджера Telegram версії не нижче 8.0 та стабільне підключення до мережі Інтернет. Процедура ініціалізації передбачає знаходження бота за унікальним іменем та відправку команди /start.

Першим кроком є заповнення анкети. Користувач має послідовно відповісти на запити бота щодо статі, віку, зросту та ваги. Рівень активності обирається з п'яти запропонованих варіантів. Після завершення реєстрації система розраховує значення BMR та TDEE.

Основна робота з ботом полягає у фіксації нутрієнтів та фізичної активності. Додавання продукту здійснюється через пошук у базі за назвою та вказання маси у грамах (за сирою вагою). Облік тренувань передбачає вибір інтенсивності навантаження.

Команда « Звіт за сьогодні» генерує Markdown-повідомлення з деталізацією спожитих та спалених калорій, а також залишком до виконання норми. У разі помилкового введення даних, користувач може скористатися функцією видалення запису або оновити профіль через розділ налаштувань.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Завершальним етапом життєвого циклу розробки інтелектуальної системи «OptiBody» є проведення комплексних випробувань та підготовка до експлуатації. На цьому етапі основна увага приділяється верифікації відповідності реалізованого функціоналу раніше сформульованим вимогам. Процес тестування дозволяє виявити потенційні помилки в логіці роботи алгоритмів, перевірити стійкість системи до некоректного введення даних та переконатися в цілісності збереження інформації у базі даних SQLite.

4.1. Тестування функціоналу програмного продукту

Для перевірки працездатності системи «OptiBody» було обрано методику тестування за принципом «чорної скриньки» (Black Box Testing) [6]. Даний підхід фокусується на перевірці функціональних можливостей інтерфейсу без глибокого аналізу внутрішньої структури коду під час виконання тестів. Це дозволяє оцінити систему з точки зору кінцевого користувача, що є пріоритетним для розмовних інтерфейсів месенджера Telegram.

Об'єктом тестування виступає асинхронна взаємодія між клієнтом Telegram, сервером Bot API та серверною частиною «OptiBody». У процесі випробувань перевірялися наступні ключові аспекти:

- Коректність відпрацювання асинхронних хендлерів при різних типах вхідних повідомлень.
- Надійність збереження антропометричних даних у профілі користувача.
- Точність математичних розрахунків за формулою Міффліна-Сан Жеора.
- Робота механізму станів (FSM) при перериванні або зміні контексту діалогу.

Нижче наведено основні сценарії, за якими проводилася верифікація програмного продукту. Для кожного тесту було визначено вхідні дані, очікуваний результат та зафіксовано фактичний стан системи.

Тест-кейс №1: Реєстрація нового користувача

- Вхідні дані: Команда /start, стать: male, вік: 25, зріст: 180, вага: 85, активність: 1.55.
- Очікуваний результат: Успішне створення запису в таблиці users, розрахунок BMR та вивід головного меню.
- Фактичний результат: Система розрахувала BMR (1866.25 ккал) та TDEE (2875 ккал), вивела Reply-клавіатуру. Тест пройдено. На Рис. 4.1 зображено повне виконання тесту за допомогою інтерфейсу.

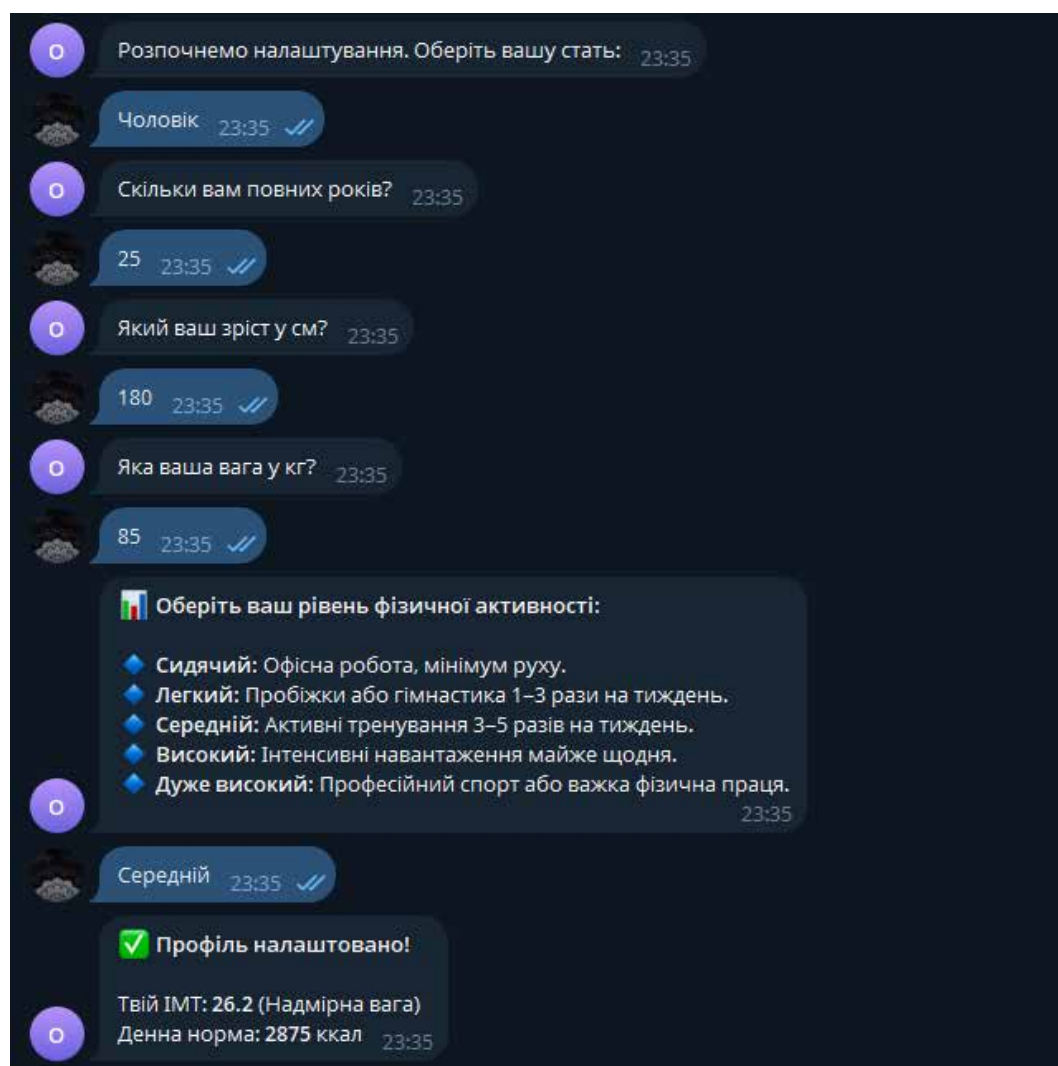


Рис. 4.1 Тест-кейс №1: Реєстрація нового користувача

Тест-кейс №2: Додавання продукту харчування

- Вхідні дані: Натискання «Додати їжу», пошук «Рис», вибір позиції, введення ваги «100».
- Очікуваний результат: Створення запису в daily_logs, автоматичний перерахунок калорій для вказаної ваги.
- Фактичний результат: Запис додано, система розрахувала енергетичну цінність на основі сирової ваги з бази даних. Тест пройдено. На рис. 4.2 зображено повне виконання тесту за допомогою інтерфейсу.

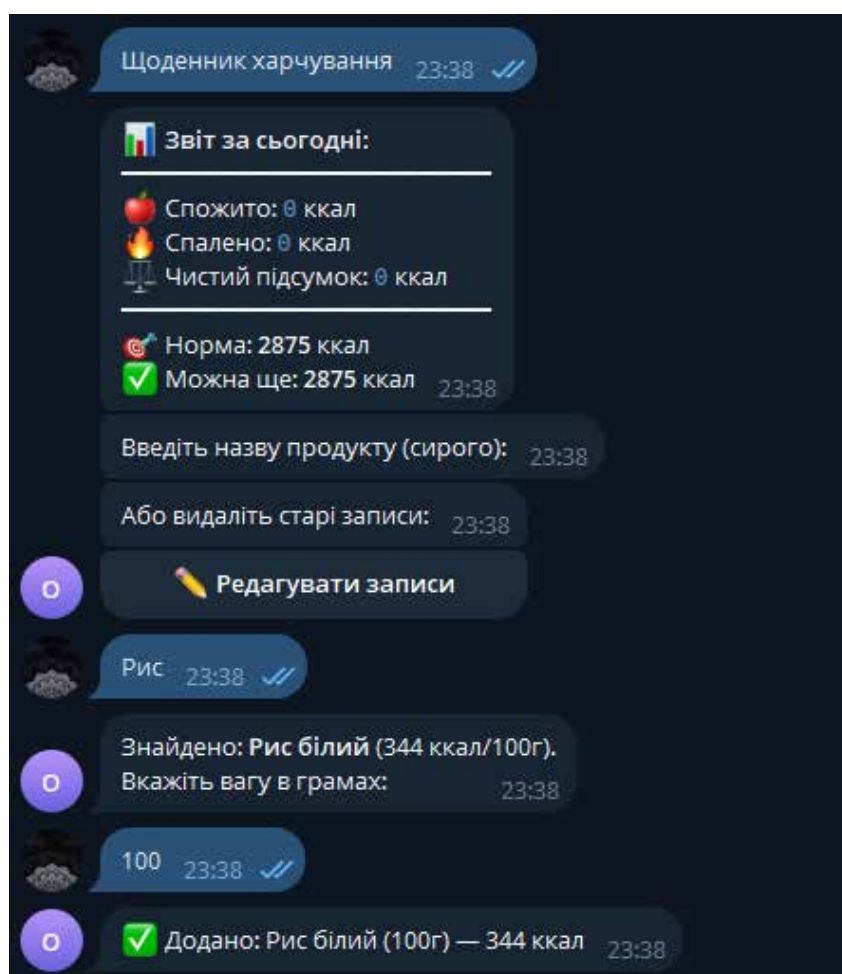


Рис. 4.2 Тест-кейс №2: Додавання продукту харчування

Тест-кейс №3: Валідація некоректного введення

- Вхідні дані: Введення тексту «сто грамів» замість числового значення «100».

- Очікуваний результат: Система має розпізнати некоректний формат та видати попередження, не змінюючи поточний стан FSM.
- Фактичний результат: Бот надіслав повідомлення: «Введіть число (грами).», очікування введення залишилося активним. Тест пройдено. На Рис. 4.3 зображено повне виконання тесту за допомогою інтерфейсу.



Рис. 4.3 Тест-кейс №3: Валідація некоректного введення

Тест-кейс №4: Формування звіту

- Вхідні дані: Натискання кнопки « Звіт» у головному меню (або відправка відповідної команди).
- Очікуваний результат: Система має агрегувати всі записи з таблиць `daily_logs` та `workout_logs` за обраний період (наприклад, за поточну

добу), порівняти їх із цільовою нормою калорій та вивести структуроване Markdown-повідомлення.

- Фактичний результат: Бот миттєво згенерував звіт, коректно підсумував спожиті та спалені калорії, а також розрахував поточний статус енергетичного балансу (дефіцит). Тест пройдено. На Рис. 4.4 зображено згенерований підсумковий звіт.

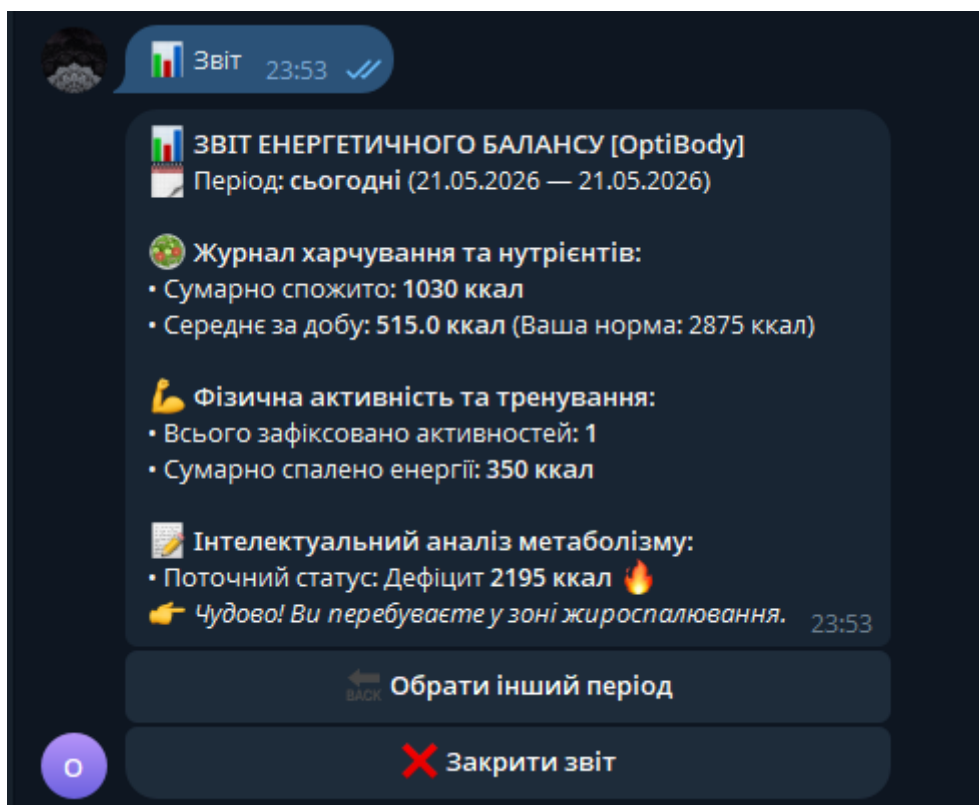


Рис. 4.4 Тест-кейс №4: Формування щоденного звіту

Тест-кейс №5: Видалення логів

- Вхідні дані: Натискання кнопки «✎ Редагувати записи» у розділі щоденника, після чого натискання на Inline-кнопку з конкретним продуктом (наприклад, «✗ Гречка»).
- Очікуваний результат: Система має виконати SQL-запит на видалення відповідного рядка з бази даних за його ідентифікатором, після чого динамічно оновити клавіатуру, прибравши видалений елемент.
- Фактичний результат: Запис успішно видалено з бази даних, Inline-клавіатура оновилася в режимі реального часу без надсилення нових

повідомлень. Тест пройдено. На Рис. 4.5 зображено інтерфейс редагування логів.

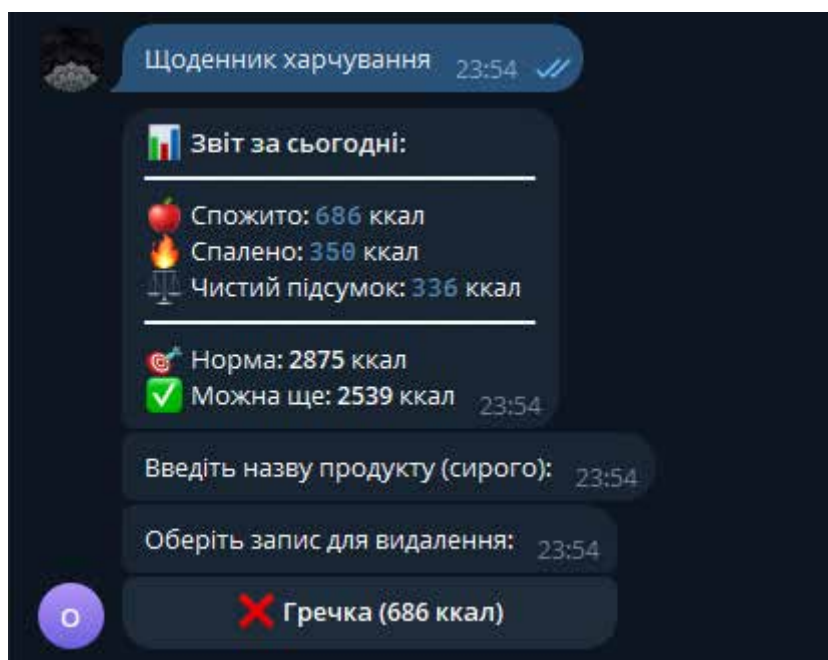


Рис. 4.5 Тест-кейс №5: Видалення записів із бази даних

Аналіз результатів проведеного тестування підтверджує високу стабільність системи. Використання асинхронної архітектури забезпечило відсутність затримок при обробці запитів, а механізм FSM успішно запобіг виникненню логічних аномалій у діалогах. Програмний продукт готовий до розгортання в робочому середовищі та повноцінної експлуатації користувачами. Це підтверджує Таблиця 4.1 - Результати функціонального тестування основних модулів OptiBody.

Таблиця 4.1

Результати функціонального тестування основних модулів OptiBody

Назва сценарію	Вхідні дані	Очікуваний результат	Статус
Реєстрація	/start, анкетні дані	Розрахунок норми, створення профілю	Пройдено
Пошук продукту	Введення назви	Вивід списку збігів із БД	Пройдено
Додавання запису	Назва + Вага	Запис у DailyLog, оновлення балансу	Пройдено
Валідація ваги	Текстове значення	Відмова, прохання ввести число	Пройдено
Формування звіту	Команда Звіт	Вивід Markdown-повідомлення	Пройдено
Видалення логів	Inline-кнопка	Видалення запису, каскадне оновлення	Пройдено

4.2. Вимоги до апаратного та програмного середовища

Етап впровадження будь-якого програмного забезпечення вимагає чіткого визначення мінімальних та рекомендованих умов для його стабільного функціонування. Оскільки інтелектуальна система «OptiBody» побудована за архітектурою «клієнт-сервер» з використанням Telegram Bot API як проміжного шлюзу, вимоги до середовища експлуатації чітко розділяються на дві категорії: вимоги до сторони кінцевого користувача (клієнта) та вимоги до обчислювальних потужностей хостингу (сервера).

Однією з ключових переваг розробки системи у форм-факторі чат-бота є концепція «нульового встановлення» (Zero Install footprint) для клієнта. Оскільки вся обчислювальна логіка, зберігання даних та рендеринг інтерфейсу відбуваються на сервері, вимоги до пристрою користувача є безпрецедентно низькими.

Апаратні вимоги користувача:

- Будь-який смарт-пристрій (смартфон, планшет, персональний комп'ютер), що здатний запускати сучасні додатки.
- Мінімальний обсяг оперативної пам'яті: від 1 ГБ.

- Наявність стабільного з'єднання з мережею Інтернет (достатньо покриття 3G/EDGE, оскільки бот передає виключно текстові дані без «важкого» медіаконтенту).

Програмні вимоги користувача:

- Встановлений офіційний клієнт месенджера Telegram (версія 8.0 або новіша) для операційних систем Android, iOS, Windows, macOS або Linux. Альтернативно можливе використання веб-версії Telegram (Telegram Web) через будь-який сучасний браузер (Chrome, Safari, Firefox).

Серверна частина «OptiBody» відповідає за безперервну обробку вхідних Webhook-запитів або Long Polling-з'єднань, виконання математичних розрахунків та транзакції до бази даних. Завдяки використанню асинхронного фреймворку aiogram та локальної реляційної СУБД SQLite, система оптимізована для роботи в умовах обмежених ресурсів.

Для забезпечення відмовостійкості, безпеки та легкої масштабованості, розгортання ядра системи доцільно проводити з використанням технологій контейнеризації (зокрема, Docker) [18]. Такий підхід гарантує повну ізоляцію програмного середовища та ідентичність умов роботи як на етапі локального тестування, так і в продакшені.

Мінімальні апаратні вимоги до сервера (VPS / VDS):

- Процесор (CPU): 1 віртуальне ядро (vCore) з частотою від 2.0 ГГц.
- Оперативна пам'ять (RAM): 512 МБ (рекомендовано 1 ГБ для забезпечення буфера під час пікових навантажень або оновлень).
- Дисковий простір: 10 ГБ SSD (база даних SQLite та вихідний код бота займають менше 50 МБ, основний обсяг необхідний для самої ОС).
- Програмні вимоги до сервера:
- Операційна система: Перевага надається дистрибутивам сімейства Linux (наприклад, Ubuntu 22.04 LTS або 24.04 LTS, Debian) завдяки їхній високій стабільності та зручності системного адміністрування через командний рядок.

- Інтерпретатор мови: Python версії 3.10 або вище.
- Залежності: Встановлені бібліотеки згідно з файлом requirements.txt (зокрема, aiogram 3.28, SQLAlchemy).
- СУБД: Вбудовані бібліотеки SQLite (не потребує встановлення окремого серверного процесу).

Таблиця 4.2 показує порівняння параметрів для Клієнта та Сервера.

Таблиця 4.2

Зведені вимоги до апаратного та програмного середовища системи OptiBody

Характеристика	Клієнтська частина (Користувач)	Серверна частина (Хостинг)
Платформа / ОС	Android, iOS, Windows, macOS, Linux	Linux (Ubuntu 22.04 / 24.04 LTS), Debian
Процесор (CPU)	Будь-який сучасний мобільний/десктопний	Мінімум 1 vCore (від 2.0 ГГц)
Оперативна пам'ять	Від 1 ГБ	Від 512 МБ (рекомендовано 1 ГБ)
Дисковий простір	До 100 МБ (кеш додатку)	Від 10 ГБ SSD
Програмне забезпечення	Telegram Client (v. 8.0+)	Python 3.10+, Docker (опціонально), SQLite
Інтернет-з'єднання	3G / 4G / Wi-Fi	Широкопasmовий доступ

Дослідження показало, що розроблена система здатна безперебійно функціонувати навіть на безкоштовних рівнях хмарних провайдерів, що робить її економічно вигідним рішенням як для стадії MVP (мінімально життєздатного продукту), так і для подальшого масштабування.

Процес програмної реалізації системи «OptiBody» базується на принципах модульності та чіткого розмежування відповідальності між шарами архітектури. Практична імплементація передбачає створення інтерактивного середовища, де користувач взаємодіє з програмним продуктом через інтуїтивно зрозумілий розмовний інтерфейс. Основна мета реалізації полягає у забезпеченні максимальної ергономічності діалогу при збереженні високої точності обробки вхідних даних.

Під час розробки інтерфейсної частини впроваджується концепція «керованого діалогу», де система на кожному етапі чітко визначає очікуваний формат вводу, мінімізуючи когнітивне навантаження на користувача. Використання механізмів Inline та Reply клавіатур дозволяє перетворити складні логічні запити на прості натискання інтерактивних елементів, що значно пришвидшує процес навігації та усуває можливість виникнення помилок введення.

Реалізація інтерфейсу бота передбачає динамічну генерацію повідомлень, що адаптуються під контекст стану користувача у скінченному автоматі. Наприклад, під час ініціалізації профілю система не лише очікує на введення антропометричних даних, а й здійснює первинну валідацію в реальному часі, надаючи миттєвий зворотний зв'язок у разі некоректного формату даних. Така архітектурна модель забезпечує стабільність роботи системи в асинхронному середовищі Telegram, де кожна сесія користувача ізольована та захищена від зовнішніх перешкод.

Функціональність модуля обліку харчування реалізується через механізм контекстного пошуку, що дозволяє користувачу швидко знаходити необхідні інгредієнти у вбудованому довіднику. Процес додавання страви автоматизується таким чином, що користувач взаємодіє лише з інтерфейсом вибору, тоді як усі складні математичні обчислення енергетичної цінності порції виконуються фоновими методами системи без залучення ручних розрахунків.

Для демонстрації процесу взаємодії користувача з основними функціональними модулями програмного продукту нижче наводяться відповідні ілюстрації.

Першим етапом роботи з системою є старт комунікації. На рис. 4.6 відображається стартове привітання бота та запуск головного Reply-меню, яке надає користувачу швидкий доступ до ключових функцій системи: перегляду профілю, щоденника харчування, тренувань та генерації звітів.

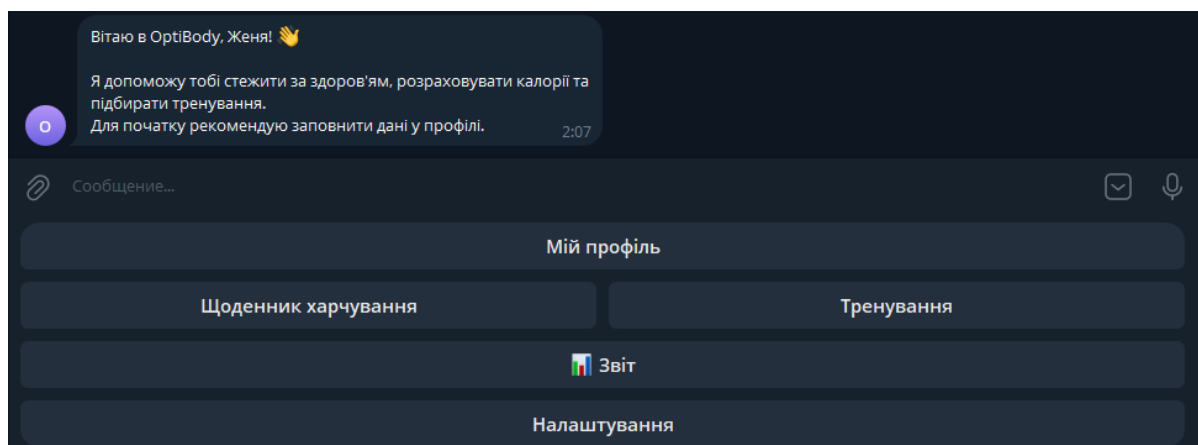


Рисунок 4.6 Відображення головного меню та ініціалізація діалогу

Для забезпечення точності математичних розрахунків система вимагає обов'язкового первинного налаштування. Процес послідовного збору фізіологічних показників користувача (зокрема віку, зросту, ваги та рівня активності) із використанням інтерактивних елементів управління ілюструється на рис. 4.7.

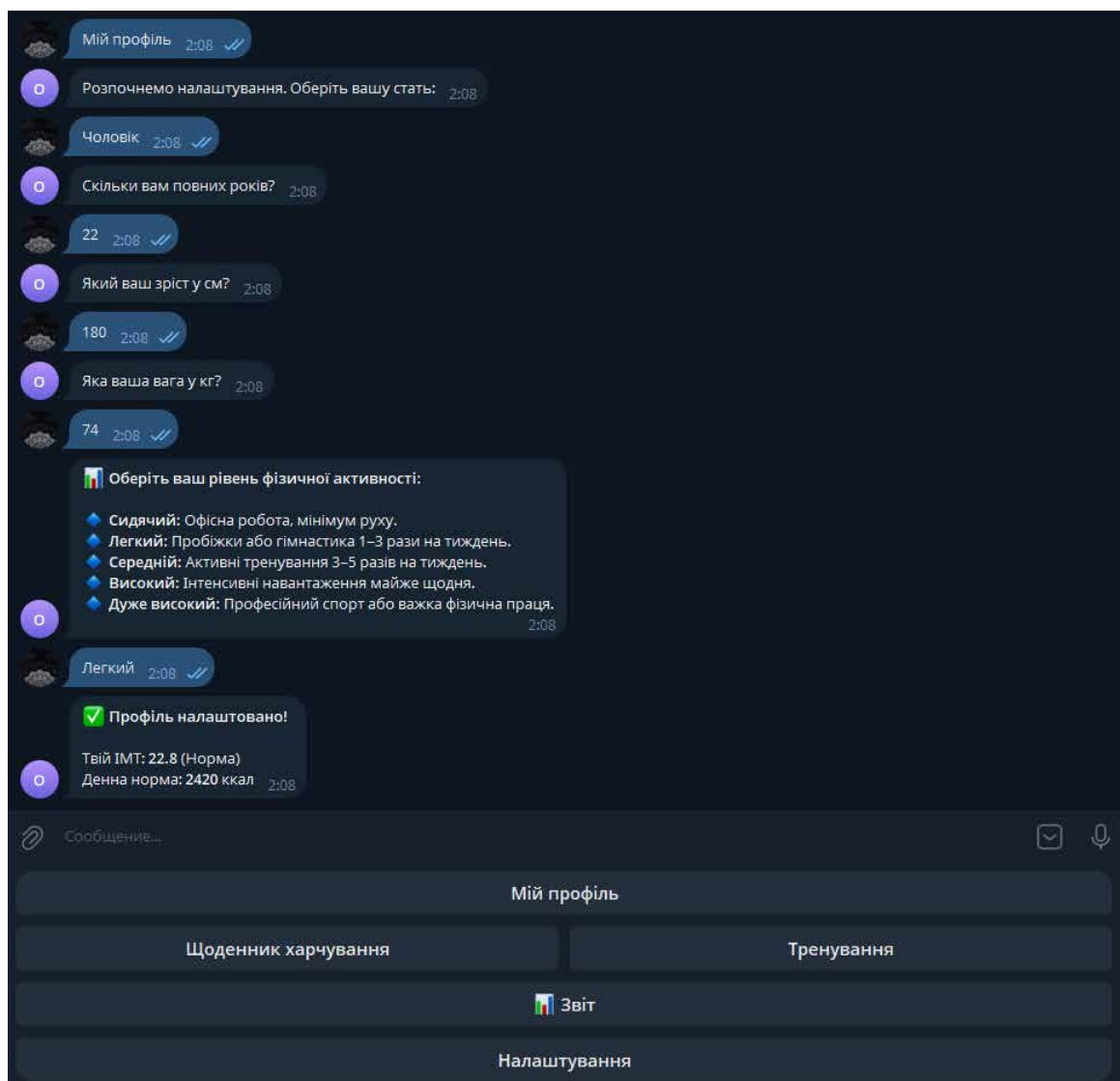


Рисунок 4.7 Етап збору фізіологічних показників користувача

Основний функціонал обліку раціону реалізується через покроковий алгоритм запити даних. На рис. 4.8 демонструється інтерфейс додавання запису про спожиту їжу до щоденника, що включає пошук продукту у базі даних за ключовим словом та підтвердження маси порції.



Рисунок 4.8 Інтерфейс додавання запису про спожиту їжу до щоденника

Кінцевим результатом агрегації введених користувачем даних є формування статистичного висновку. На рис. 4.9 відображається представлення результатів аналізу раціону в інтерфейсі бота у форматі деталізованого щоденного звіту, де чітко структуровано інформацію про спожиті та спалені калорії, а також підсумковий енергетичний баланс.

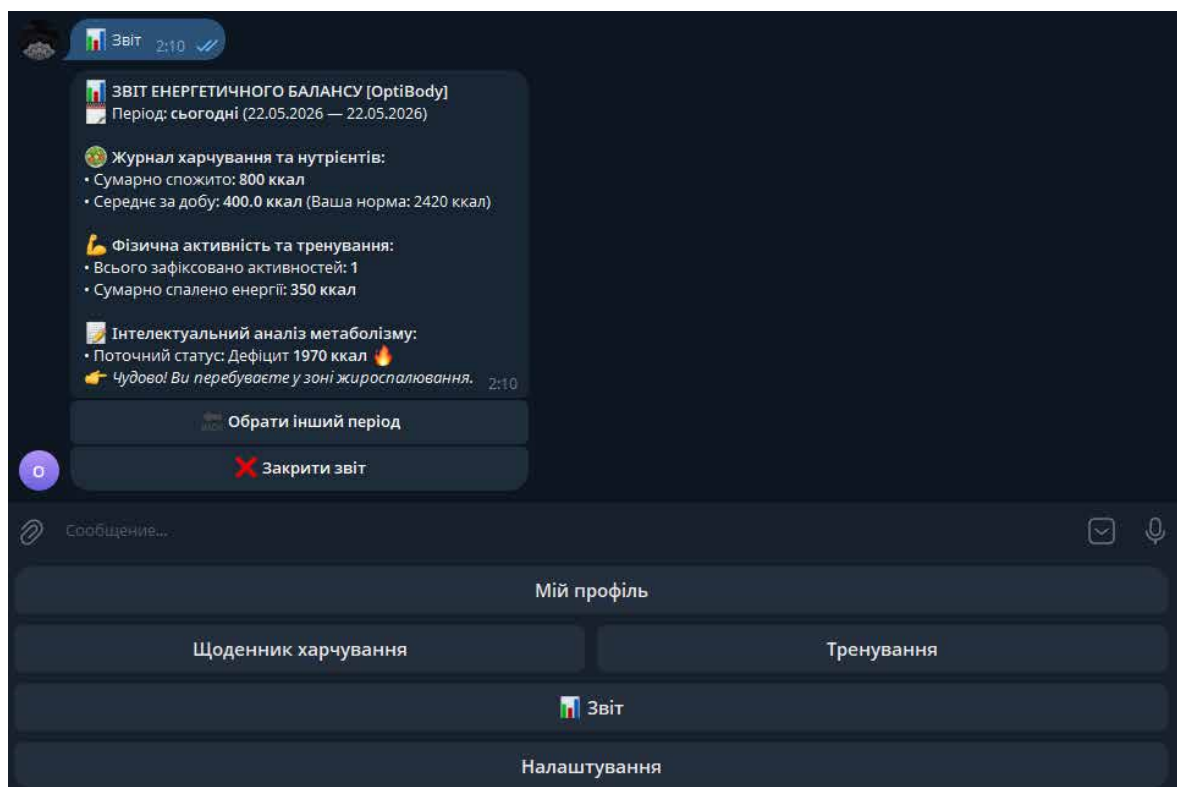


Рисунок 4.9 Представлення результатів аналізу раціону в інтерфейсі бота

Таким чином, реалізована система інтерфейсної взаємодії повністю відповідає поставленим вимогам щодо юзабіліті та функціональності. Кожен етап розробки спрямовується на створення цілісного інформаційного середовища, яке не лише виконує покладені на нього обчислювальні завдання, а й формує позитивний користувацький досвід через зрозумілу та швидку навігацію. Тестування на коректність роботи інтерфейсних модулів підтверджує стійкість системи до непередбачуваних сценаріїв поведінки користувача, що гарантує надійність експлуатації продукту в реальних умовах.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра вирішено актуальне науково-практичне завдання інженерії програмного забезпечення — розроблено інтелектуальну інформаційну систему «OptiBody» для моніторингу та корекції енергетичного балансу користувача, що функціонує в асинхронному середовищі месенджера Telegram.

За результатами виконання етапів проєктування, розробки та тестування системи можна зробити наступні висновки:

1. На основі проведеного системного аналізу предметної області доведено актуальність створення легких та доступних інструментів підтримки здорового способу життя. Аналіз існуючих аналогів (MyFitnessPal, FatSecret та ін.) виявив низку недоліків: перевантаженість інтерфейсу, агресивну монетизацію та неточність баз даних готових страв. Це обґрунтувало необхідність розробки системи з фокусом на розрахунок калорійності за сировою вагою інгредієнтів.

2. Здійснено проєктування архітектури програмного продукту. Побудовано логічну (ER-діаграма) та фізичну моделі реляційної бази даних. Обґрунтовано вибір технологічного стеку: мови Python 3.10+, асинхронного фреймворку aiogram 3.x, ORM SQLAlchemy та локальної СКБД SQLite. Обрана трирівнева архітектура забезпечує високу швидкість обробки запитів та легку масштабованість проєкту.

3. Програмно реалізовано алгоритмічне забезпечення системи. Інтегровано математичну модель Міффіна-Сан Жеора для високоточного розрахунку базового метаболізму (BMR) та добової норми витрат енергії (TDEE). Розроблено надійний алгоритм управління діалогами на базі механізму скінченних автоматів (FSM), що гарантує цілісність збору антропометричних даних. Створено мінімалістичний розмовний інтерфейс (Conversational UI) із використанням Inline та Reply клавіатур.

4. Проведено комплексне функціональне тестування розробленого програмного продукту за методом «чорної скриньки». Результати тестування

підтвердили коректність обробки вхідних даних, точність математичних розрахунків та стійкість системи до невалідного введення з боку користувача. Сформовано інструкцію користувача та визначено мінімальні вимоги до апаратного та програмного середовища.

Таким чином, мета дипломного проєкту повністю досягнута, а всі поставлені завдання виконані. Розроблена система «OptiBody» є готовим програмним продуктом, який ефективно автоматизує процеси щоденного трекінгу калорій, підвищуючи рівень обізнаності користувачів щодо власного здоров'я без створення додаткового когнітивного навантаження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація мови програмування Python [Електронний ресурс]: <https://docs.python.org/3/>
2. Документація асинхронного фреймворку aiogram 3.x [Електронний ресурс]: <https://docs.aiogram.dev/>
3. Офіційна документація системи керування базами даних SQLite [Електронний ресурс]: <https://www.sqlite.org/docs.html>
4. Документація інструменту об'єктно-реляційного відображення SQLAlchemy [Електронний ресурс]. – Режим доступу: <https://docs.sqlalchemy.org/>
5. Офіційна документація розробника Telegram Bot API [Електронний ресурс]: <https://core.telegram.org/bots/api>
6. Соммервілл І. Інженерія програмного забезпечення / Іан Соммервілл. – 10-те вид. – М.: Вільямс, 2018. – 736 с.
7. Фаулер М. Архітектура корпоративних програмних додатків / Мартін Фаулер. – М.: Вільямс, 2019. – 544 с.
8. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Прийоми об'єктно-орієнтованого проектування. Патерни проектування. – СПб.: Пітер, 2020. – 368 с.
9. Буч Г., Якобсон І., Рамбо Д. Уніфікована мова моделювання UML. Посібник користувача / Г. Буч, І. Якобсон, Д. Рамбо. – 2-ге вид. – М.: ДМК Прес, 2019. – 496 с.
10. Конноллі Т., Бегг К. Бази даних: проектування, реалізація та супроводження. Теорія та практика / Т. Конноллі, К. Бегг. – 3-тє вид. – М.: Вільямс, 2017. – 1440 с.
11. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / Роберт Мартін. – СПб.: Пітер, 2019. – 352 с.
12. Mifflin M. D., St Jeor S. T., Hill L. A., Lewis B. J., Koo S. A., Hamm M. P. A new predictive equation for resting energy expenditure in healthy individuals // The American Journal of Clinical Nutrition. – 1990. – Vol. 51. – No. 2. – P. 241-247.

13. Всесвітня організація охорони здоров'я (ВООЗ). Рекомендації щодо здорового харчування та фізичної активності [Електронний ресурс]: <https://www.who.int/news-room/fact-sheets/detail/healthy-diet>
14. Міністерство охорони здоров'я України. Національні рекомендації щодо здорового харчування дорослих [Електронний ресурс]: <https://moz.gov.ua/>
15. Любанович Б. Простий Python. Сучасний стиль програмування / Білл Любанович. – СПб.: Пітер, 2021. – 480 с.
16. Персіваль Х., Грегори Б. Патерни розробки архітектури програм на Python / Х. Персіваль, Б. Грегори. – М.: ДМК Прес, 2022. – 300 с.
17. Дейт К. Дж. Вступ до систем баз даних / Крістофер Дейт. – 8-ме вид. – М.: Вільямс, 2018. – 1328 с.
18. Офіційна документація інструменту контейнеризації Docker [Електронний ресурс]: <https://docs.docker.com/>
19. Шилдт Г. Патерни проектування на практиці / Герберт Шилдт. – М.: Вільямс, 2021. – 432 с.
20. Маккінні У. Python для аналізу даних / Уес Маккінні. – М.: ДМК Прес, 2020. – 544 с.

ДОДАТОК А

Програмний код застосунку

Лістинг models.py

```
from sqlalchemy import BigInteger, String, Float, Integer, ForeignKey, DateTime
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy.ext.asyncio import AsyncAttrs, async_sessionmaker, create_async_engine
from datetime import datetime
from config import DB_URL
from sqlalchemy.orm import Mapped, mapped_column
from sqlalchemy import String, Integer
```

```
engine = create_async_engine(url=DB_URL, echo=True)
async_session = async_sessionmaker(engine)
```

```
class Base(AsyncAttrs, DeclarativeBase):
    pass
```

1.Таблиця користувачів

```
class User(Base):
    __tablename__ = 'users'

    id: Mapped[int] = mapped_column(primary_key=True)
    tg_id = mapped_column(BigInteger, unique=True)
    name: Mapped[str] = mapped_column(String(50))
    age: Mapped[int] = mapped_column(Integer, nullable=True)
    height: Mapped[float] = mapped_column(Float, nullable=True)
    weight: Mapped[float] = mapped_column(Float, nullable=True)
    gender: Mapped[str] = mapped_column(String(10), nullable=True)
    activity_level: Mapped[float] = mapped_column(Float, nullable=True)
```

2.Таблиця щоденника харчування

```
class DailyLog(Base):
    __tablename__ = 'daily_logs'

    id: Mapped[int] = mapped_column(primary_key=True)
    tg_id = mapped_column(BigInteger, ForeignKey('users.tg_id'))
    product_name: Mapped[str] = mapped_column(String(100))
    calories: Mapped[int] = mapped_column(Integer)
    date: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
```

3.Щоденник тренувань

```
class WorkoutLog(Base):
    __tablename__ = 'workout_logs'

    id: Mapped[int] = mapped_column(primary_key=True)
```

```

tg_id = mapped_column(BigInteger, ForeignKey('users.tg_id'))
workout_type: Mapped[str] = mapped_column(String(50)) # Силові, Кардіо тощо
difficulty: Mapped[str] = mapped_column(String(20)) # Легко, Середньо...
calories_burned: Mapped[int] = mapped_column(Integer) # Спалені калорії
date: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)

# Функція створення таблиць
async def async_main():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)

class Product(Base):
    __tablename__ = 'products'

    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(100), unique=True)
    calories_per_100g: Mapped[int] = mapped_column(Integer)

```

Лістинг requests.py

```

from sqlalchemy import select, update, delete, func, between
from datetime import date, datetime
from database.models import async_session, User, Product, DailyLog, WorkoutLog

async def get_user(user_id: int):
    """Отримує дані користувача з бази за його Telegram ID (tg_id)"""
    async with async_session() as session:
        result = await session.execute(select(User).where(User.tg_id == user_id))
        return result.scalar()

async def set_user(user_id: int, first_name: str):
    """Зберігає користувача при натисканні /start, якщо його ще немає в базі"""
    async with async_session() as session:
        user = await session.execute(select(User).where(User.tg_id == user_id))

        if not user.scalar():
            new_user = User(tg_id=user_id, name=first_name)
            session.add(new_user)
            await session.commit()

async def update_user_profile(user_id: int, gender: str, age: int, height: float, weight: float, activity_level: float):
    """Оновлює антропометричні дані користувача"""
    async with async_session() as session:
        stmt = update(User).where(User.tg_id == user_id).values(
            gender=gender, age=age, height=height, weight=weight,
            activity_level=activity_level
        )

```

```
await session.execute(stmt)
await session.commit()
```

```
async def get_product_by_name(name: str):
    """Пошук продукту в базі за назвою (для health_diary.py)"""
    async with async_session() as session:
        result = await session.execute(select(Product).where(Product.name.icontains(name)))
        return result.scalars().first()
```

```
async def get_today_calories(user_id: int):
    """Отримує кількість спожитих калорій за поточну добу"""
    async with async_session() as session:
        today = date.today()
        query = select(func.sum(DailyLog.calories)).where(
            DailyLog.tg_id == user_id,
            func.date(DailyLog.date) == today
        )
        result = await session.execute(query)
        total = result.scalar()
        return total or 0
```

```
async def add_food_log(user_id: int, product_name: str, calories: int, log_date: datetime = None):
    """Зберігає запис про прийом їжі у щоденник"""
    if log_date is None:
        log_date = datetime.now()
    async with async_session() as session:
        new_log = DailyLog(
            tg_id=user_id, product_name=product_name,
            calories=calories, date=log_date
        )
        session.add(new_log)
        await session.commit()
```

```
async def get_today_logs(user_id: int):
    """Отримує всі записи їжі за сьогодні для меню редагування/видалення"""
    async with async_session() as session:
        today = date.today()
        query = select(DailyLog).where(
            DailyLog.tg_id == user_id,
            func.date(DailyLog.date) == today
        )
        result = await session.execute(query)
        return result.scalars().all()
```

```

async def delete_log(log_id: int, user_id: int):
    """Видаляє конкретний запис їжі з бази"""
    async with async_session() as session:
        # Перевіряємо tg_id, щоб користувач міг видалити тільки свої записи
        await session.execute(delete(DailyLog).where(DailyLog.id == log_id, DailyLog.tg_id == user_id))
        await session.commit()

```

```

async def get_today_burned(user_id: int):
    """Отримує кількість спалених калорій за поточну добу"""
    async with async_session() as session:
        today = date.today()
        query = select(func.sum(WorkoutLog.calories_burned)).where(
            WorkoutLog.tg_id == user_id,
            func.date(WorkoutLog.date) == today
        )
        result = await session.execute(query)
        total = result.scalar()
        return total or 0

```

```

async def add_workout_log(user_id: int, workout_type: str, difficulty: str, calories_burned: int,
                          log_date: datetime = None):
    """Зберігає запис про фізичну активність"""
    if log_date is None:
        log_date = datetime.now()
    async with async_session() as session:
        new_log = WorkoutLog(
            tg_id=user_id, workout_type=workout_type, difficulty=difficulty,
            calories_burned=calories_burned, date=log_date
        )
        session.add(new_log)
        await session.commit()

```

```

async def get_period_food_summary(user_id: int, start_date: date, end_date: date):
    """Отримує сумарну та середню калорійність харчування за вказаний період"""
    async with async_session() as session:
        query = select(
            func.sum(DailyLog.calories).label("total_calories"),
            func.avg(DailyLog.calories).label("avg_calories")
        ).where(
            DailyLog.tg_id == user_id,
            between(func.date(DailyLog.date), start_date, end_date)
        )

        result = await session.execute(query)
        summary = result.fetchone()

```

```

return {
    "total": round(summary.total_calories or 0, 2),
    "average": round(summary.avg_calories or 0, 2)
}

```

```

async def get_period_workout_summary(user_id: int, start_date: date, end_date: date):
    """Отримує загальну кількість тренувань та спалених калорій за період"""
    async with async_session() as session:
        query = select(
            func.sum(WorkoutLog.calories_burned).label("total_burned"),
            func.count(WorkoutLog.id).label("count")
        ).where(
            WorkoutLog.tg_id == user_id,
            between(func.date(WorkoutLog.date), start_date, end_date)
        )

        result = await session.execute(query)
        summary = result.fetchone()

        return {
            "total_burned": round(summary.total_burned or 0, 2),
            "count": summary.count or 0
        }

```

Лістинг health_diary.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from utils.states import FoodDiary
import database.requests as rq
from keyboards.reply import main_menu, cancel_kb
from keyboards.inline import get_edit_logs_keyboard
from utils.formulas import calculate_calories

diary_router = Router()

#СКАСУВАННЯ
@diary_router.message(F.text == "Скасувати ✖")
async def cmd_cancel(message: types.Message, state: FSMContext):
    await state.clear()
    await message.answer("Дію скасовано.", reply_markup=main_menu)

#ГОЛОВНИЙ ЕКРАН
@diary_router.message(F.text == "Щоденник харчування")
async def start_diary(message: types.Message, state: FSMContext):
    user = await rq.get_user(message.from_user.id)
    if not user or not user.activity_level:
        return await message.answer("Спочатку заповніть профіль!")

```

```
consumed = await rq.get_today_calories(message.from_user.id)
burned = await rq.get_today_burned(message.from_user.id)
norm = calculate_calories(user.weight, user.height, user.age, user.gender, user.activity_level)
```

```
net_calories = consumed - burned
remaining = norm - net_calories
```

```
summary = (f"📊 *Звіт за сьогодні:*\n"
            f"-----\n"
            f"🍏 Спожито: `{consumed}` ккал\n"
            f"🔥 Спалено: `{burned}` ккал\n"
            f"⚖️ Чистий підсумок: `{net_calories}` ккал\n"
            f"-----\n"
            f"🎯 Норма: `{norm}` ккал\n")
```

```
if remaining > 0:
    summary += f"✅ Можна ще: `{remaining}` ккал"
else:
    summary += f"⚠️ Перевищення: `{abs(remaining)}` ккал"
```

```
kb = types.InlineKeyboardMarkup(inline_keyboard=[
    [types.InlineKeyboardButton(text="✎ Редагувати записи", callback_data="manage_logs")]
])
```

```
await message.answer(summary, parse_mode="Markdown")
await message.answer("Введіть назву продукту (сирого):", reply_markup=cancel_kb)
await message.answer("Або видаліть старі записи:", reply_markup=kb)
await state.set_state(FoodDiary.waiting_for_product_name)
```

#ПОШУК ТА ДОДАВАННЯ

```
@diary_router.message(FoodDiary.waiting_for_product_name)
async def process_food_search(message: types.Message, state: FSMContext):
    print(f"Шукаю продукт: {message.text}")
    product = await rq.get_product_by_name(message.text)

    if product:
        await state.update_data(product_name=product.name, cal_100g=product.calories_per_100g)
        await message.answer(f"Знайдено: `{product.name}` ({product.calories_per_100g} ккал/100г).\n"
                              f"Вкажіть вагу в грамах:", parse_mode="Markdown")
        await state.set_state(FoodDiary.waiting_for_weight)
    else:
        await state.update_data(food_name=message.text)
        await message.answer(f"Продукту '{message.text}' немає в базі.\n"
                              f"Введіть кількість калорій для цієї порції вручну:")
        await state.set_state(FoodDiary.manual_calories)
```

```

@diary_router.message(FoodDiary.waiting_for_weight)
async def process_weight(message: types.Message, state: FSMContext):
    if not message.text.isdigit():
        return await message.answer("Введіть число (грами).")

    weight = int(message.text)
    if weight <= 0 or weight > 5000:
        return await message.answer("Введіть адекватну вагу (1-5000г).")

    data = await state.get_data()
    total_cal = int((weight * data['cal_100g']) / 100)

    await rq.add_food_log(message.from_user.id, data['product_name'], total_cal)
    await state.clear()
    await message.answer(f"✅ Додано: {data['product_name']} ({weight}г) — {total_cal} ккал",
                        reply_markup=main_menu)

```

```

@diary_router.message(FoodDiary.manual_calories)
async def process_manual_cal(message: types.Message, state: FSMContext):
    if not message.text.isdigit():
        return await message.answer("Введіть число калорій.")

    cal = int(message.text)
    if cal <= 0 or cal > 5000:
        return await message.answer("Введіть число від 1 до 5000.")

    data = await state.get_data()
    await rq.add_food_log(message.from_user.id, data['food_name'], cal)
    await state.clear()
    await message.answer(f"✅ Записано вручну: {data['food_name']}", reply_markup=main_menu)

```

#ВИДАЛЕННЯ

```

@diary_router.callback_query(F.data == "manage_logs")
async def manage_logs(callback: types.CallbackQuery):
    logs = await rq.get_today_logs(callback.from_user.id)
    if not logs:
        return await callback.answer("Записів немає!", show_alert=True)

    await callback.message.edit_text("Оберіть запис для видалення:",
                                    reply_markup=get_edit_logs_keyboard(logs))

```

```

@diary_router.callback_query(F.data.startswith("delete_log_"))
async def delete_log_handler(callback: types.CallbackQuery):
    log_id = int(callback.data.split("_")[2])
    await rq.delete_log(log_id, callback.from_user.id)

    logs = await rq.get_today_logs(callback.from_user.id)

```

```

if not logs:
    await callback.message.edit_text("Всі записи видалено.", reply_markup=None)
else:
    await callback.message.edit_text("Запис видалено. Щось ще?",
                                     reply_markup=get_edit_logs_keyboard(logs))
await callback.answer("Видалено")

```

Лістинг profile.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from utils.states import Profile
import database.requests as rq
from utils.formulas import calculate_bmi, get_bmi_category, calculate_calories
from keyboards.reply import main_menu, cancel_kb

profile_router = Router()

# ЛОГІКА СКАСУВАННЯ
@profile_router.message(F.text == "Скасувати ❌")
async def cmd_cancel(message: types.Message, state: FSMContext):
    await state.clear()
    await message.answer("Дію скасовано. Повертаємось у головне меню.", reply_markup=main_menu)

# ВІДОБРАЖЕННЯ ПРОФІЛЮ
@profile_router.message(F.text == "Мій профіль")
async def show_profile(message: types.Message, state: FSMContext):
    user = await rq.get_user(message.from_user.id)

    if user and user.weight and user.activity_level:
        bmi = calculate_bmi(user.weight, user.height)
        category = get_bmi_category(bmi)
        # Розрахунок з урахуванням активності
        calories = calculate_calories(user.weight, user.height, user.age, user.gender, user.activity_level)

        text = (f"👤 *Твій профіль:* \n\n"
                f"• Ім'я: {user.name} \n"
                f"• Стать: {user.gender} \n"
                f"• Вік: {user.age} \n"
                f"• Зріст: {user.height} см \n"
                f"• Вага: {user.weight} кг \n"
                f"• Рівень активності: {user.activity_level} \n\n"
                f"📊 *Показники:* \n"
                f"• ІМТ: {bmi} ({category}) \n"
                f"• Денна норма: {calories} ккал")

        kb = types.InlineKeyboardMarkup(inline_keyboard=[
            [types.InlineKeyboardButton(text="🔄 Оновити дані", callback_data="re_register")]
        ])
    )

```

```

    await message.answer(text, reply_markup=kb, parse_mode="Markdown")
else:
    await start_survey(message, state)

# ОБРОБКА КНОПКИ "ОНОВИТИ ДАНІ"
@profile_router.callback_query(F.data == "re_register")
async def re_register_callback(callback: types.CallbackQuery, state: FSMContext):
    await callback.answer()
    await callback.message.delete()
    await start_survey(callback.message, state)

# ПОЧАТОК АНКЕТУВАННЯ (FSM)
async def start_survey(message: types.Message, state: FSMContext):
    gender_kb = types.ReplyKeyboardMarkup(keyboard=[
        [types.KeyboardButton(text="Чоловік"), types.KeyboardButton(text="Жінка")],
        [types.KeyboardButton(text="Скасувати ✖")],
    ], resize_keyboard=True)

    await message.answer("Розпочнемо налаштування. Оберіть вашу стать:", reply_markup=gender_kb)
    await state.set_state(Profile.gender)

@profile_router.message(Profile.gender)
async def process_gender(message: types.Message, state: FSMContext):
    if message.text not in ["Чоловік", "Жінка"]:
        return await message.answer("Використовуйте кнопки для вибору статі.")
    await state.update_data(gender=message.text)
    await message.answer("Скільки вам повних років?", reply_markup=cancel_kb)
    await state.set_state(Profile.age)

@profile_router.message(Profile.age)
async def process_age(message: types.Message, state: FSMContext):
    if not message.text.isdigit():
        return await message.answer("Введіть число.")

    age = int(message.text)
    if not (10 <= age <= 100): # Ліміт: 10-100 років
        return await message.answer("Будь ласка, вкажіть реальний вік (від 10 до 100 років).")

    await state.update_data(age=age)
    await message.answer("Який ваш зріст у см?", reply_markup=cancel_kb)
    await state.set_state(Profile.height)

@profile_router.message(Profile.height)
async def process_height(message: types.Message, state: FSMContext):
    if not message.text.isdigit():

```

```

    return await message.answer("Введіть число.")

    height = int(message.text)
    if not (100 <= height <= 250): # Ліміт: 100-250 см
        return await message.answer("Зріст має бути в межах від 100 до 250 см.")

    await state.update_data(height=height)
    await message.answer("Яка ваша вага у кг?", reply_markup=cancel_kb)
    await state.set_state(Profile.weight)

@profile_router.message(Profile.weight)
async def process_weight(message: types.Message, state: FSMContext):
    weight_str = message.text.replace(',', '.')
    try:
        weight = float(weight_str)
    except ValueError:
        return await message.answer("Введіть число.")

    if not (30 <= weight <= 300): # Ліміт: 30-300 кг
        return await message.answer("Вага має бути в межах від 30 до 300 кг.")

    await state.update_data(weight=weight)

    activity_description = (
        "🇮🇹 *Оберіть ваш рівень фізичної активності:* \n\n"
        "💠 *Сидячий:* Офісна робота, мінімум руху. \n"
        "💠 *Легкий:* Пробіжки або гімнастика 1–3 рази на тиждень. \n"
        "💠 *Середній:* Активні тренування 3–5 разів на тиждень. \n"
        "💠 *Високий:* Інтенсивні навантаження майже щодня. \n"
        "💠 *Дуже високий:* Професійний спорт або важка фізична праця."
    )

    activity_kb = types.ReplyKeyboardMarkup(keyboard=[
        [types.KeyboardButton(text="Сидячий"), types.KeyboardButton(text="Легкий")],
        [types.KeyboardButton(text="Середній"), types.KeyboardButton(text="Високий")],
        [types.KeyboardButton(text="Дуже високий")],
        [types.KeyboardButton(text="Скасувати ✖")],
    ], resize_keyboard=True)

    await message.answer(activity_description, reply_markup=activity_kb, parse_mode="Markdown")
    await state.set_state(Profile.activity)

@profile_router.message(Profile.activity)
async def process_activity(message: types.Message, state: FSMContext):
    levels = {
        "Сидячий": 1.2,
        "Легкий": 1.375,
        "Середній": 1.55,
    }

```

```

    "Високий": 1.725,
    "Дуже високий": 1.9
}

if message.text not in levels:
    return await message.answer("Будь ласка, оберіть варіант за допомогою кнопок.")

coeff = levels[message.text]
await state.update_data(activity_level=coeff)

data = await state.get_data()

await rq.update_user_profile(
    user_id=message.from_user.id,
    gender=data.get('gender'),
    age=int(data.get('age')),
    height=float(data.get('height')),
    weight=float(data.get('weight')),
    activity_level=float(data.get('activity_level'))
)

bmi = calculate_bmi(data['weight'], data['height'])
category = get_bmi_category(bmi)
calories = calculate_calories(data['weight'], data['height'], data['age'], data['gender'], coeff)

await state.clear()
await message.answer(
    f"✅ *Профіль налаштовано!*\n\n"
    f"Твій ІМТ: *{bmi}* ({category})\n"
    f"Денна норма: *{calories}* ккал",
    reply_markup=main_menu,
    parse_mode="Markdown"
)

```

Лістинг reports.py

```

from aiogram import Router, F, types
from aiogram.filters import Command
from datetime import datetime, timedelta, date
from database.requests import get_user, get_period_food_summary, get_period_workout_summary
from keyboards.inline import get_report_period_keyboard, get_report_nav_keyboard

router = Router()

@router.message(F.text == "📊 Звіт за сьогодні")
@router.message(F.text == "📊 Звіт")
@router.message(Command("report"))
async def show_report_menu(message: types.Message):
    """Відображає інтерактивне inline-меню для вибору періоду аналітики"""
    await message.answer(

```

```

text="🌳 *Оберіть період*, за який ви хочете отримати аналітику енергетичного балансу:",
reply_markup=get_report_period_keyboard(),
parse_mode="Markdown"
)

```

```

@router.callback_query(F.data == "report_back")
async def back_to_report_menu(callback: types.CallbackQuery):
    """Повертає користувача назад до вибору періоду"""
    await callback.message.edit_text(
        text="🌳 *Оберіть період*, за який ви хочете отримати аналітику енергетичного балансу:",
        reply_markup=get_report_period_keyboard(),
        parse_mode="Markdown"
    )
    await callback.answer()

```

```

@router.callback_query(F.data == "report_close")
async def close_report_menu(callback: types.CallbackQuery):
    """Закриває (видаляє) повідомлення зі звітом"""
    await callback.message.delete()
    await callback.answer("Звіт закрито.")

```

```

@router.callback_query(F.data.startswith("report_"))
async def process_report_callback(callback: types.CallbackQuery):
    """Обробляє натискання кнопок вибору періоду та генерує фінальний звіт"""
    # Ігноруємо колбеки навігації, які обробляються вище
    if callback.data in ["report_back", "report_close"]:
        return

```

```

    user_id = callback.from_user.id
    user = await get_user(user_id)

```

```

    if not user or not user.weight or not user.height or not user.age:
        await callback.message.edit_text(
            text="⚠️ *Профіль не заповнено!* \nСпочатку пройдіть реєстрацію та налаштуйте свої антропометричні параметри."
        )
        await callback.answer()
        return

```

```

    today = date.today()
    period_text = ""

```

```

    if callback.data == "report_today":
        start_date = today
        end_date = today
        period_text = "сьогодні"
    elif callback.data == "report_week":

```

```

start_date = today - timedelta(days=6)
end_date = today
period_text = "останні 7 днів"
elif callback.data == "report_month":
    start_date = today - timedelta(days=29)
    end_date = today
    period_text = "останні 30 днів"
else:
    await callback.answer("Помилка вибору періоду.")
    return

food_data = await get_period_food_summary(user_id, start_date, end_date)
workout_data = await get_period_workout_summary(user_id, start_date, end_date)

days_count = (end_date - start_date).days + 1

if user.gender in ['male', 'Чоловіча', 'Чоловік']:
    bmr = (10 * user.weight) + (6.25 * user.height) - (5 * user.age) + 5
else:
    bmr = (10 * user.weight) + (6.25 * user.height) - (5 * user.age) - 161

target_calories = bmr * (user.activity_level or 1.2)
total_target_tdee = target_calories * days_count

report_lines = [
    f"📊 *ЗВІТ ЕНЕРГЕТИЧНОГО БАЛАНСУ [OptiBody]*",
    f"📅 Період: *{period_text}* ({start_date.strftime('%d.%m.%Y')} — {end_date.strftime('%d.%m.%Y')})\n",
    f"🍽️ *Журнал харчування та нутрієнтів:",
    f"• Сумарно спожито: *{food_data['total']} ккал*",
    f"• Середнє за добу: *{food_data['average']} ккал* (Ваша норма: {round(target_calories)} ккал)\n",
    f"🏃 *Фізична активність та тренування:",
    f"• Всього зафіксовано активностей: *{workout_data['count']}*",
    f"• Сумарно спалено енергії: *{workout_data['total_burned']} ккал*\n",
    f"🧠 *Інтелектуальний аналіз метаболізму:",
]

total_expenditure = total_target_tdee + workout_data['total_burned']
net_balance = food_data['total'] - total_expenditure

if days_count == 1:
    if net_balance < 0:
        report_lines.append(f"• Поточний статус: Дефіцит *{abs(round(net_balance))} ккал* 🔥")
        report_lines.append("👉 _Чудово! Ви перебуваєте у зоні жироспалювання._")
    else:
        report_lines.append(f"• Поточний статус: Профіцит *{round(net_balance)} ккал* 📈")
        report_lines.append("👉 _Увага! Профіцит енергії веде до накопичення маси тіла._")
else:
    avg_net_daily = net_balance / days_count
    if net_balance < 0:

```

```

report_lines.append(f"• Загальний дефіцит за період: *{abs(round(net_balance))} ккал* 🔥")
report_lines.append(f"• Середній добовий дефіцит: *{abs(round(avg_net_daily))} ккал*")
report_lines.append("👉 _Ви дотримуетесь правильної стратегії для стабільного схуднення! _")
else:
    report_lines.append(f"• Загальний профіцит за період: *{round(net_balance)} ккал* 📈")
    report_lines.append(f"• Середній добовий профіцит: *{round(avg_net_daily)} ккал*")
    report_lines.append("👉 _Рекомендується скоригувати раціон або збільшити обсяг тренувань._")

await callback.message.edit_text(
    text="\n".join(report_lines),
    parse_mode="Markdown",
    reply_markup=get_report_nav_keyboard()
)
await callback.answer()

```

Лістинг settings.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from utils.states import SettingsState
import database.requests as rq
from keyboards.reply import main_menu

settings_router = Router()

# Головне меню налаштувань
@settings_router.message(F.text == "Налаштування")
async def show_settings(message: types.Message):
    kb = types.InlineKeyboardMarkup(inline_keyboard=[
        [types.InlineKeyboardButton(text="👤 Змінити ім'я", callback_data="change_name")],
        [types.InlineKeyboardButton(text="📄 Про проєкт", callback_data="about_project")],
        [types.InlineKeyboardButton(text="⚠️ Видалити всі дані", callback_data="confirm_delete")]
    ])
    await message.answer("⚙️ *Налаштування профілю*\nОберіть потрібний пункт:",
        reply_markup=kb, parse_mode="Markdown")

# Зміна імені
@settings_router.callback_query(F.data == "change_name")
async def change_name_init(callback: types.CallbackQuery, state: FSMContext):
    await callback.answer()
    await callback.message.answer("Напишіть ваше нове ім'я:")
    await state.set_state(SettingsState.waiting_for_name)

@settings_router.message(SettingsState.waiting_for_name)
async def process_new_name(message: types.Message, state: FSMContext):
    if len(message.text) > 20:
        return await message.answer("Ім'я занадто довге (макс. 20 символів).")

```

```

await rq.update_user_name(message.from_user.id, message.text)
await state.clear()
await message.answer(f"Приємно познайомитись, *{message.text}*! Ім'я оновлено.",
                      reply_markup=main_menu, parse_mode="Markdown")

#Про проєкт
@settings_router.callback_query(F.data == "about_project")
async def about_info(callback: types.CallbackQuery):
    await callback.answer()
    about_text = (
        "🤖 *OptiBody Bot* v1.0\n\n"
        "Цей бот створений для підтримки здорового способу життя, "
        "контролю калорій та планування тренувань.\n\n"
        "🎓 *Розробник:* Вінтонюк Євген\n"
        "📍 *НУБіП України*\n"
        "Дипломний проєкт з розробки інтелектуальних систем."
    )
    await callback.message.answer(about_text, parse_mode="Markdown")

#Видалення даних
@settings_router.callback_query(F.data == "confirm_delete")
async def confirm_delete(callback: types.CallbackQuery):
    await callback.answer()
    kb = types.InlineKeyboardMarkup(inline_keyboard=[
        [types.InlineKeyboardButton(text="❌ ТАК, видалити все", callback_data="delete_all_final")],
        [types.InlineKeyboardButton(text="🔙 Скасувати", callback_data="cancel_settings")]
    ])
    await callback.message.edit_text(
        "🔴 *ВИ ВПЕВНЕНІ?*\n\nВсі ваші дані (профіль, історія харчування та тренувань) будуть видалені безповоротно.",
        reply_markup=kb, parse_mode="Markdown")

@settings_router.callback_query(F.data == "delete_all_final")
async def delete_all(callback: types.CallbackQuery):
    await rq.delete_user_complete(callback.from_user.id)
    await callback.answer("Дані видалено", show_alert=True)
    await callback.message.edit_text("🗑️ Всі ваші дані видалено. Натисніть /start, щоб почати знову.")

@settings_router.callback_query(F.data == "cancel_settings")
async def cancel_settings(callback: types.CallbackQuery):
    await callback.answer()
    await callback.message.delete()
    await callback.message.answer("Дію скасовано.", reply_markup=main_menu)

```

Лістинг start.py

```

from aiogram import Router, types
from aiogram.filters import CommandStart
import database.requests as rq
from keyboards.reply import main_menu

```

```

start_router = Router()

```

```

@start_router.message(CommandStart())

```

```

async def cmd_start(message: types.Message):

```

```

    await rq.set_user(message.from_user.id, message.from_user.first_name)

```

```

    await message.answer(

```

```

        f"Вітаю в OptiBody, {message.from_user.first_name}! 🤖\n\n"

```

```

        "Я допоможу тобі стежити за здоров'ям, розраховувати калорії та підбирати тренування.\n"

```

```

        "Для початку рекомендую заповнити дані у профілі.",

```

```

        reply_markup=main_menu

```

```

    )

```

Лістинг workout.py

```

from aiogram import Router, F, types

```

```

from aiogram.fsm.context import FSMContext

```

```

from utils.states import Workout

```

```

from utils.workout_data import WORKOUT_PLAN

```

```

from keyboards.reply import main_menu, cancel_kb

```

```

import database.requests as rq

```

```

workout_router = Router()

```

```

@workout_router.message(F.text == "Скасувати ✖")

```

```

async def cmd_cancel_workout(message: types.Message, state: FSMContext):

```

```

    await state.clear()

```

```

    await message.answer("Тренування скасовано.", reply_markup=main_menu)

```

```

@workout_router.message(F.text == "Тренування")

```

```

async def start_workout(message: types.Message, state: FSMContext):

```

```

    kb = types.InlineKeyboardMarkup(inline_keyboard=[

```

```

        [types.InlineKeyboardButton(text="🟢 Легкий", callback_data="set_diff_Легкий")],

```

```

        [types.InlineKeyboardButton(text="🟡 Середній", callback_data="set_diff_Середній")],

```

```

        [types.InlineKeyboardButton(text="🔴 Важкий", callback_data="set_diff_Важкий")]

```

```

    ])

```

```

    await message.answer("Оберіть рівень складності тренування:", reply_markup=cancel_kb)

```

```

    await message.answer("Рівні складності:", reply_markup=kb)

```

```

    await state.set_state(Workout.waiting_for_difficulty)

```

```

@workout_router.callback_query(F.data.startswith("set_diff_"))
async def show_workout_plan(callback: types.CallbackQuery, state: FSMContext):
    difficulty = callback.data.split("_")[2]
    plan = WORKOUT_PLAN[difficulty]

    await state.update_data(difficulty=difficulty, burned=plan['calories'])

    text = (f"💪 *Ваш план ({difficulty}):*\n\n"
            f"{plan['exercises']}\n\n"
            f"🔥 Спалиш приблизно: *{plan['calories']} ккал*")

    kb = types.InlineKeyboardMarkup(inline_keyboard=[
        [types.InlineKeyboardButton(text="✅ Завершити", callback_data="finish_workout")],
        [types.InlineKeyboardButton(text="❌ Скасувати", callback_data="cancel_workout_call")]
    ])

    await callback.message.edit_text(text, reply_markup=kb, parse_mode="Markdown")

@workout_router.callback_query(F.data == "finish_workout")
async def finish_workout(callback: types.CallbackQuery, state: FSMContext):
    data = await state.get_data()

    # Зберігаємо в базу
    await rq.add_workout_log(callback.from_user.id, "Загальне", data['difficulty'], data['burned'])

    await state.clear()
    await callback.message.delete()
    await callback.message.answer(f"🎉 Чудова робота! Ви спалили *{data['burned']} ккал*.",
                                reply_markup=main_menu, parse_mode="Markdown")

@workout_router.callback_query(F.data == "cancel_workout_call")
async def cancel_workout_inline(callback: types.CallbackQuery, state: FSMContext):
    await state.clear()
    await callback.message.delete()
    await callback.message.answer("Тренування скасовано.", reply_markup=main_menu)

```

Лістинг inline.py

```

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
from aiogram.utils.keyboard import InlineKeyboardBuilder

def get_gender_keyboard() -> InlineKeyboardMarkup:
    """Клавіатура для вибору статі під час реєстрації"""
    builder = InlineKeyboardBuilder()
    builder.button(text="👤 Чоловіча", callback_data="gender_male")
    builder.button(text="👤 Жіноча", callback_data="gender_female")
    builder.adjust(2)
    return builder.as_markup()

```

```

def get_activity_keyboard() -> InlineKeyboardMarkup:
    """Клавіатура для вибору коефіцієнта фізичної активності (PAL)"""
    builder = InlineKeyboardBuilder()
    builder.button(text="🛌 Малорухливий (1.2)", callback_data="act_1.2")
    builder.button(text="🚶 Легка активність (1.375)", callback_data="act_1.375")
    builder.button(text="🚶‍♀️ Середня активність (1.55)", callback_data="act_1.55")
    builder.button(text="🏃 Висока активність (1.725)", callback_data="act_1.725")
    builder.button(text="🔥 Екстремальна (1.9)", callback_data="act_1.9")
    builder.adjust(1)
    return builder.as_markup()

def get_workout_intensity_keyboard() -> InlineKeyboardMarkup:
    """Клавіатура вибору інтенсивності тренування"""
    builder = InlineKeyboardBuilder()
    builder.button(text="🟢 Низька (Йога, Ходьба)", callback_data="workout_low")
    builder.button(text="🟡 Середня (Біг, Фітнес)", callback_data="workout_mid")
    builder.button(text="🔴 Висока (Кросфіт, Вага)", callback_data="workout_high")
    builder.adjust(1)
    return builder.as_markup()

def get_report_period_keyboard() -> InlineKeyboardMarkup:
    """Клавіатура для вибору періоду звітності (Сьогодні/Тиждень/Місяць)"""
    builder = InlineKeyboardBuilder()
    builder.button(text="📅 За сьогодні", callback_data="report_today")
    builder.button(text="📅 31 За останні 7 днів", callback_data="report_week")
    builder.button(text="📅 30 За останні 30 днів", callback_data="report_month")
    builder.adjust(1)
    return builder.as_markup()

def get_report_nav_keyboard() -> InlineKeyboardMarkup:
    """Клавіатура навігації ПІСЛЯ відображення звіту"""
    builder = InlineKeyboardBuilder()
    builder.button(text="⬅️ Обрати інший період", callback_data="report_back")
    builder.button(text="❌ Закрити звіт", callback_data="report_close")
    builder.adjust(1)
    return builder.as_markup()

def get_edit_logs_keyboard(logs) -> InlineKeyboardMarkup:
    """
    Динамічна клавіатура, яка приймає список записів їжі
    і створює кнопку видалення для кожного з них.
    """

```

```
builder = InlineKeyboardBuilder()

for log in logs:
    btn_text = f"✖ {log.product_name} ({log.calories} ккал)"
    builder.button(text=btn_text, callback_data=f"delete_log_{log.id}")

builder.adjust(1)
return builder.as_markup()
```

```
def get_delete_log_keyboard(log_id: int, log_type: str) -> InlineKeyboardMarkup:
    """
    Клавіатура для видалення конкретного запису
    """
    builder = InlineKeyboardBuilder()
    builder.button(text="🗑 Видалити запис", callback_data=f"delete_{log_type}_{log_id}")
    return builder
```

Лістинг reply.py

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

# Головне меню
main_menu = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="Мій профіль")],
    [KeyboardButton(text="Щоденник харчування"), KeyboardButton(text="Тренування")],
    [KeyboardButton(text="📊 Звіт")],
    [KeyboardButton(text="Налаштування")]
], resize_keyboard=True)

# Кнопка для виходу з будь-якого стану
cancel_kb = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="Скасувати ✖")]
], resize_keyboard=True)
```

Лістинг formulas.py

```
def calculate_bmi(weight, height):
    # Формула: вага (кг) / (зріст (м) ^ 2)
    bmi = weight / ((height / 100) ** 2)
    return round(bmi, 1)

def get_bmi_category(bmi):
    if bmi < 18.5: return "Недостатня вага"
    if bmi < 25: return "Норма"
    if bmi < 30: return "Надмірна вага"
    return "Ожиріння"

def calculate_calories(weight, height, age, gender, activity_level):
    if gender == "Чоловік":
        bmr = 10 * weight + 6.25 * height - 5 * age + 5
```

```

else:
    bmr = 10 * weight + 6.25 * height - 5 * age - 161

    return int(bmr * activity_level)

```

Лістинг states.py

```

from aiogram.fsm.state import StatesGroup, State

class Profile(StatesGroup):
    gender = State()
    age = State()
    height = State()
    weight = State()
    activity = State()

class FoodDiary(StatesGroup):
    waiting_for_product_name = State() # Користувач пише назву
    waiting_for_weight = State()      # Користувач пише вагу
    manual_calories = State()         # Резервний варіант, якщо продукту немає

class Workout(StatesGroup):
    waiting_for_category = State()
    waiting_for_difficulty = State()

class SettingsState(StatesGroup):
    waiting_for_name = State()

```

Лістинг workout_data.py

```

WORKOUT_PLAN = {
    "Легкий": {
        "exercises": (
            "◆ Відтискання від підлоги: 10 разів\n"
            "◆ Присідання: 15 разів\n"
            "◆ Скручування на прес: 20 разів\n"
            "◆ Біг: 1 км (легкий темп)"
        ),
        "calories": 150
    },
    "Середній": {
        "exercises": (
            "◆ Відтискання від підлоги: 3 підходи по 15 разів\n"
            "◆ Присідання: 3 підходи по 20 разів\n"
            "◆ Скручування на прес: 3 підходи по 25 разів\n"
            "◆ Біг: 3 км"
        ),
        "calories": 350
    },
    "Важкий": {
        "exercises": (

```

```

        "◆ Відтискання від підлоги: 4 підходи по 25 разів\n"
        "◆ Присідання: 4 підходи по 30 разів\n"
        "◆ Скручування на прес: 4 підходи по 40 разів\n"
        "◆ Біг: 5 км"
    ),
    "calories": 600
}
}

```

Лістинг main.py

```

import asyncio
import logging
import sys
from aiogram import Bot, Dispatcher
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode

from config import BOT_TOKEN
from database.models import async_main

from handlers.start import start_router
from handlers.profile import profile_router
from handlers.health_diary import diary_router
from handlers.workout import workout_router
from handlers.settings import settings_router
from handlers.reports import router as reports_router

async def main():

    await async_main()

    logging.basicConfig(
        level=logging.INFO,
        format="%asctime)s - %(levelname)s - %(name)s - %(message)s",
        stream=sys.stdout
    )

    # Ініціалізація бота
    bot = Bot(
        token=BOT_TOKEN,
        default=DefaultBotProperties(parse_mode=ParseMode.MARKDOWN)
    )

    # Головний Диспетчер
    dp = Dispatcher()

    # Підключення роутерів
    dp.include_routers(

```

```
        start_router,
        profile_router,
        diary_router,
        workout_router,
        settings_router,
        reports_router
    )

    logging.info("Система OptiBody успішно ініціалізована та запущена.")

    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\nБот зупинений користувачем.")
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Вінтонюк Євген Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи підтримки здорового способу життя» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - о ☐ інструменти генеративного ШІ не використовувалися.
 - о ☐ інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____ інші (вказати назву)) були використані для:
 - ☐ перекладу іншомовних джерел;
 - ☐ стилістичного редагування та перевірки граматики;
 - ☐ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« _____ » 2026 р. _____

Євген ВІНТОНЮК