НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
__інформаційних систем і технологій__
(назва кафедри)

_____/Швиденко М. З./
(підпис) (ПІБ)

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Рекомендаційна система підтримки здорового способу життя»

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми
_____д. ек. н, професор_____Руденський Р.А.
_____ _____ _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____д. пед. н., професор____ _____ ____Кузьмінська О. Г.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Сухомлин Олексій Олександрович_____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
інформаційних систем і технологій
(назва кафедри)

к.ек. н, доцент Швиденко М. З.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Сухомлин Олексій Олександрович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Рекомендаційна система підтримки здорового способу життя» затверджена наказом ректора НУБіП України від 26.02.2025 №238"С"
2. Термін подання завершеної роботи на кафедру 02.06.2025 р.
(рік, місяць, число)
3. Вихідні дані до роботи: створення Telegram-бота по веденню здорового способу життя.
4. Перелік питань, що розглядаються:
   - Системний аналіз предметної області
   - Інформаційне забезпечення
   - Прикладне програмне забезпечення
   - Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "26" лютого 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ / Кузьмінська О. Г. /
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання: _____ /Сухомлин О. О. /
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

У сучасному світі питання підтримки здорового способу життя набуває дедалі більшої актуальності. Ритм життя багатьох людей передбачає тривале перебування в офісі, низьку фізичну активність та несистемне харчування. Це зумовлює підвищений інтерес до цифрових засобів, що можуть допомогти у формуванні корисних звичок. Зокрема, зростає популярність рішень, які поєднують автоматизацію та персоналізований підхід — до таких належать рекомендаційні системи, реалізовані у формі чат-ботів.

Мета роботи полягає у розробці Telegram-бота, який надає індивідуальні рекомендації з фізичної активності та харчування на основі персональних параметрів користувача. Telegram обрано як платформу завдяки його широкому розповсюдженню, підтримці ботів, мобільності та зручності для щоденної взаємодії.

У процесі реалізації було використано сучасні програмні інструменти та методи, зокрема:

- мову програмування Python 3.10 [1];
- фреймворк Aiogram [2] для асинхронної обробки повідомлень у Telegram;
- Finite State Machine (FSM) для побудови логіки діалогів;
- SQLite [4] як вбудовану реляційну систему управління базами даних;
- алгоритм fuzzy-пошуку через бібліотеку rapidfuzz [3];
- побудова графіків та візуалізація статистики за допомогою matplotlib [16].
- базові підходи до розрахунку калорійності за формулою Mifflin-St Jeor [11].

Апробація результатів відбувалася під час індивідуального тестування з демонстрацією можливостей бота на передзахисті, а також під час внутрішніх консультацій на кафедрі комп’ютерних наук. Матеріали проєкту розглядались як основа для студентської науково-практичної доповіді на кафедральному семінарі.

Структура пояснювальної записки охоплює 81 сторінку тексту без додаткыв. У роботі використано 22 джерела, включено 2 додатки. Записка складається з чотирьох основних розділів:

- Розділ 1 містить системний аналіз предметної області, постановку завдання, огляд джерел і моделювання;
- Розділ 2 описує логічну модель даних, обґрунтування вибору системи управління базами даних і структуру бази;
- Розділ 3 присвячений реалізації Telegram-бота: архітектура, алгоритми, інтерфейс, програмні модулі;
- Розділ 4 включає тестування функціоналу, вимоги до середовища, інсталяційний пакет і діаграму розгортання.

Запропоноване рішення орієнтоване на повсякденне використання, поєднує функції моніторингу, рекомендацій та статистики, а також демонструє застосування сучасних технологій у сфері персонального цифрового здоров’я.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

У сучасному суспільстві питання підтримки здорового способу життя набуває все більшої актуальності. Проблеми недостатньої фізичної активності, незбалансованого харчування та низької мотивації до дотримання режиму дня характерні для великої кількості людей, зокрема студентів, офісних працівників та інших категорій населення. Водночас розвиток мобільних технологій відкриває нові можливості для цифрової підтримки здорових звичок, зокрема за допомогою месенджерів і чат-ботів.

Метою цієї бакалаврської кваліфікаційної роботи є розробка інформаційної системи у вигляді Telegram-бота, що надає персоналізовані рекомендації з фізичної активності та харчування. Система повинна працювати в інтерактивному режимі, враховувати індивідуальні параметри користувача (вік, вага, зріст, фізичні показники) та надавати адаптовані рекомендації з урахуванням поточного стану. Особливістю є підтримка двох паралельних компонентів — планування фізичної активності та харчування — із можливістю ручного коригування записів користувачем, а також збирання та візуалізація статистичних даних на основі щоденного використання.

У процесі розробки системи реалізовано наступну функціональність:

- реєстрація користувача з покроковим введенням особистих даних (ім’я, прізвище, вік, вага, зріст), із перевірками допустимості значень;
- введення показників максимальної фізичної підготовки (максимальна кількість відтискань, присідань, та тривалість планки), необхідних для подальшої генерації тренувального плану;

- автоматичне формування тренувального плану на день відповідно до рівня фізичної підготовки та обраної інтенсивності (легкий, середній, важкий), з можливістю оновлення плану за запитом користувача;
- фіксація факту виконання тренування з відображенням прогресу в модулі статистики;
- формування індивідуального плану харчування на основі добової калорійності, яка розраховується за формулою Mifflin-St Jeor [11] з урахуванням віку, ваги та зросту користувача;
- рекомендація конкретних продуктів, розрахованих за масою, калорійністю та вмістом білків, жирів і вуглеводів;
- оновлення харчового плану за запитом користувача;
- ведення харчового щоденника із можливістю ручного додавання продуктів на основі пошуку за назвою, із підрахунком фактично спожитих калорій та макронутрієнтів;
- можливість перегляду спожитих продуктів за останні 7 днів;
- модуль статистики, який дозволяє користувачу переглянути середні показники харчування за останній тиждень та кількість тренувань за останні 30 днів, а також отримати візуалізовану інформацію у вигляді графіків: лінійних діаграм для аналізу спожитих калорій і макронутрієнтів (білки, жири, вуглеводи) та календарної мапи для оцінки регулярності фізичної активності.
- профіль користувача з можливістю перегляду та повного редагування всіх особистих даних;
- зручний користувацький інтерфейс на основі інлайн-кнопок, що забезпечує вибір дій без текстового вводу;

- асинхронна обробка запитів з використанням Finite State Machine (FSM) для багатокрокових сценаріїв (реєстрація, редагування, щоденник тощо).

Програмна реалізація заснована на фреймворку Aiogram [2], який підтримує асинхронну обробку подій та побудову FSM-діалогів. Для зберігання даних використовується реляційна база даних SQLite [4], що реалізує повноцінну структуру користувача, вправ, продуктів, планів та історії активностей. Система модульна, масштабована, і може бути використана як інструмент для персонального моніторингу здоров'я.

Система належить до класу інформаційно-рекомендаційних, користувацьких програмних систем. За характером використання її можна віднести до мобільних та дистанційних сервісів. Вона орієнтована на широке коло користувачів, які мають доступ до месенджера Telegram і бажають автоматизувати контроль за здоровим способом життя.

1.2 Огляд інформаційних джерел та існуючих рішень

У процесі розробки інформаційної системи рекомендаційного типу було проаналізовано низку джерел, які висвітлюють підходи до автоматизації процесів підтримки здорового способу життя. Основну увагу приділено огляду існуючих програмних рішень, що мають схожий функціонал, а також методичній та науковій літературі, яка описує принципи формування раціону харчування та планів фізичної активності.

Серед найпоширеніших застосунків, що реалізують подібну функціональність, можна виділити:

- **MyFitnessPal** [17] — мобільний додаток для обліку харчування, тренувань і калорій. Має велику базу продуктів, підтримує ведення щоденника та

статистику. Недоліками є перевантаженість інтерфейсу, наявність реклами та обмеження в функціоналі без платної підписки.

- **Nike Training Club** [19] — фітнес-застосунок із широкою бібліотекою тренувальних програм. Програми побудовані за участі професійних тренерів, проте застосунок орієнтований виключно на фізичну активність, не включає блоків харчування, не враховує індивідуальні фізіологічні параметри користувача.

- **Yazio** [18] — популярний сервіс для розрахунку калорійності, макронутрієнтів та формування планів харчування. Дозволяє задавати цілі (схуднення, підтримка ваги, набір маси), однак вимагає значної ручної роботи — продукти вводяться вручну, тренування не враховуються, а сама система працює як окрема мобільна програма без підтримки бот-платформ.

На додаток до огляду програмних засобів було вивчено теоретичні основи моделювання харчової поведінки та фізичної активності. Зокрема, використано фізіологічну формулу Mifflin-St Jeor [11] для розрахунку базового обміну речовин, яка є однією з найточніших для осіб зі звичайним рівнем жирової маси. Розрахунок добової потреби в енергії здійснюється з урахуванням ваги, росту, віку та статі користувача. Добова норма поділяється на три категорії макронутрієнтів (білки, жири, вуглеводи) згідно з загальноприйнятими дієтологічними нормами (20/30/50%).

Також проаналізовано методики оцінки фізичної активності та визначення інтенсивності тренувань. Для адаптації планів до індивідуального рівня підготовки використано шкалу максимальних фізичних можливостей, що фіксується користувачем у вигляді кількості відтискань, присідань та часу утримання планки.

Аналіз показав, що більшість існуючих рішень або орієнтовані лише на одну складову (або тренування, або харчування), або вимагають встановлення окремого застосунку з авторизацією, оновленнями та рекламою. Водночас користувачі месенджера Telegram використовують його щоденно, що дає змогу реалізувати зручний та завжди доступний інтерфейс на основі чат-бота.

На відміну від розглянутих програм, запропонована система:

- не потребує встановлення додатків, працює прямо в Telegram;
- поєднує автоматично сформовані плани тренувань і харчування з можливістю ручного ведення щоденника;
- надає статистику за тиждень та місяць з візуалізацією фактів активності;
- підтримує збереження історії й оновлення планів у будь-який момент;
- реалізована з використанням асинхронної логіки обробки (FSM), що дозволяє гнучко керувати сценаріями взаємодії без втрати контексту.

Запропонована система поєднує в одному інтерфейсі блоки тренувань, харчування, щоденника, перегляду профілю та статистики. Вона орієнтована на повсякденне використання, автоматизує рутинні обчислення та підвищує рівень самоконтролю користувача без навантаження складними інтерфейсами. Таким чином, Telegram-бот є логічним, функціонально повним і зручним інструментом для формування та підтримки здорового способу життя.

1.3 Моделювання предметної області

У рамках проєктування програмного забезпечення було створено набір UML-діаграм, які відображають ключові аспекти функціонування системи. Метою моделювання є формалізація предметної області, визначення сценаріїв взаємодії користувача з Telegram-ботом, а також опис логіки обробки даних, що

зберігаються у внутрішній базі. Кожна діаграма ілюструє окремий рівень системи — від загальних сценаріїв до конкретних дій у межах окремих модулів.

1.3.1 Діаграма прецедентів. На рис. 1 представлено загальну діаграму прецедентів, яка відображає всі ключові функціональні можливості Telegram-бота з точки зору користувача. Основним актором є кінцевий користувач, який взаємодіє із системою через текстові команди та інлайн-кнопки.

Функціональні можливості згруповано в основні сценарії: реєстрація, профіль, тренування, харчування, ведення щоденника та статистика. Кожна з основних дій містить включені прецеденти (<<include>>), які реалізовані у вигляді вкладених опцій.

Зокрема:

- прецедент «Редагувати профіль» є частиною «Переглянути профіль», оскільки доступ до редагування можливий лише після перегляду (з інлайн-кнопки);
- «Оновити тренувальний план» включений до «Отримати тренувальний план» — користувач має можливість згенерувати новий план;
- аналогічно, «Оновити харчовий план» реалізований через кнопку в межах сценарію отримання харчування;
- «Видалити останній запис» та «Переглянути продукти за 7 днів» є допоміжними діями в межах «Вести харчовий щоденник»;
- у модулі статистики передбачені два додаткові прецеденти: «Переглянути деталі по харчуванню» і «Переглянути деталі по тренуваннях» — вони доступні лише після виклику основного сценарію.

Така структура забезпечує повне охоплення функціоналу бота, дозволяє зберігати послідовність та цілісність сценаріїв і прямо відображає архітектурну логіку системи.

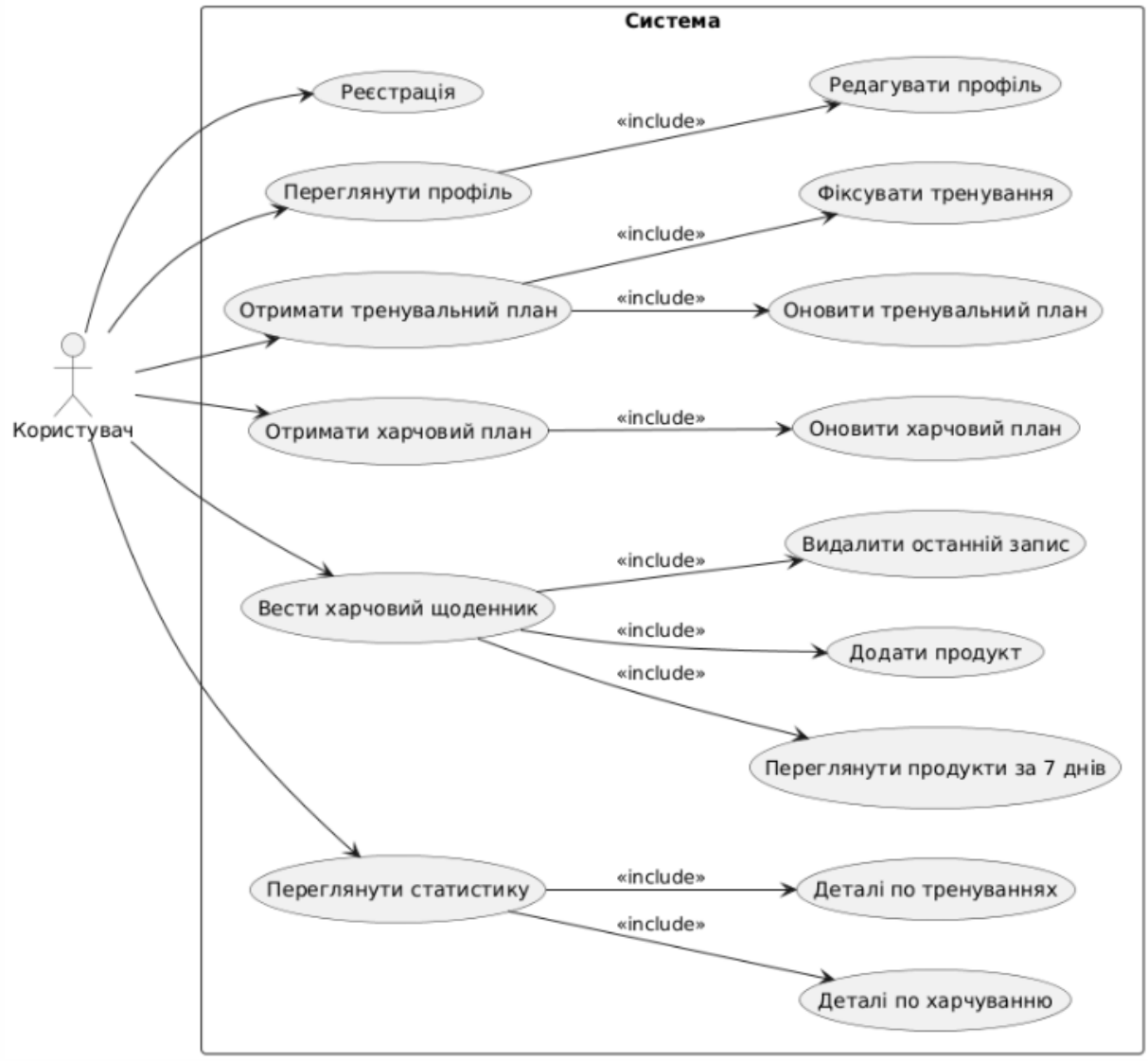

Рис. 1 – Діаграма прецедентів Telegram-бота для підтримки здорового способу життя

Після аналізу всієї системи як сукупності взаємопов’язаних сценаріїв, доцільно виділити два ключові модулі для глибшого моделювання їхньої поведінки та внутрішньої логіки. Це:

- Модуль “Тренування” — який включає багатоступеневу перевірку (наявність показників, плану), генерацію, оновлення та збереження даних.
- Модуль “Щоденник” — який є найбільш інтерактивним: включає введення даних, fuzzy-пошук [3] продуктів, збереження, перегляд і можливість видалення записів.

Обидва модулі мають розгалужену логіку, використовують FSM (машини станів), тісно взаємодіють із базою даних і реалізують поведінку користувача в динаміці. Тому в подальших підрозділах буде подано діаграми активності та послідовності саме для цих двох блоків, як найрепрезентативніших для оцінки архітектури та зручності взаємодії користувача із системою.

1.3.2 Діаграма активності: «Тренування». На рис. 2 представлено діаграму активності, що моделює сценарій використання Telegram-бота для формування тренувального плану. Діаграма побудована у вигляді двох лінійних послідовних перевірок — така структура точно відповідає реальній логіці реалізації функціоналу у програмному коді.

Першим кроком після обрання користувачем опції «Тренування» є перевірка, чи введено максимальні показники вправ (відтискання, присідання, планка). Якщо ці дані відсутні, система активує багатокроковий сценарій введення на основі FSM, після чого зберігає значення в базу даних. Незалежно від того, чи дані вже були, чи були щойно введені, система переходить до наступного етапу — перевірки наявності плану.

На другому етапі перевіряється, чи існує вже тренувальний план для поточної дати. Якщо план знайдено, бот витягує його з бази та виводить користувачеві. Якщо плану немає, користувач обирає рівень інтенсивності (легкий, середній, важкий), після чого система генерує новий план, зберігає його у базі даних та надсилає повідомлення з рекомендаціями.

Завершальним кроком, який користувач може виконати додатково, є фіксація факту завершення тренування. При підтвердженні система записує відповідний факт у таблицю workout_sessions, що впливає на формування статистики.

Такий підхід гарантує поетапність логіки, відповідність стану даних, зручність для користувача та гнучкість обробки сценаріїв.

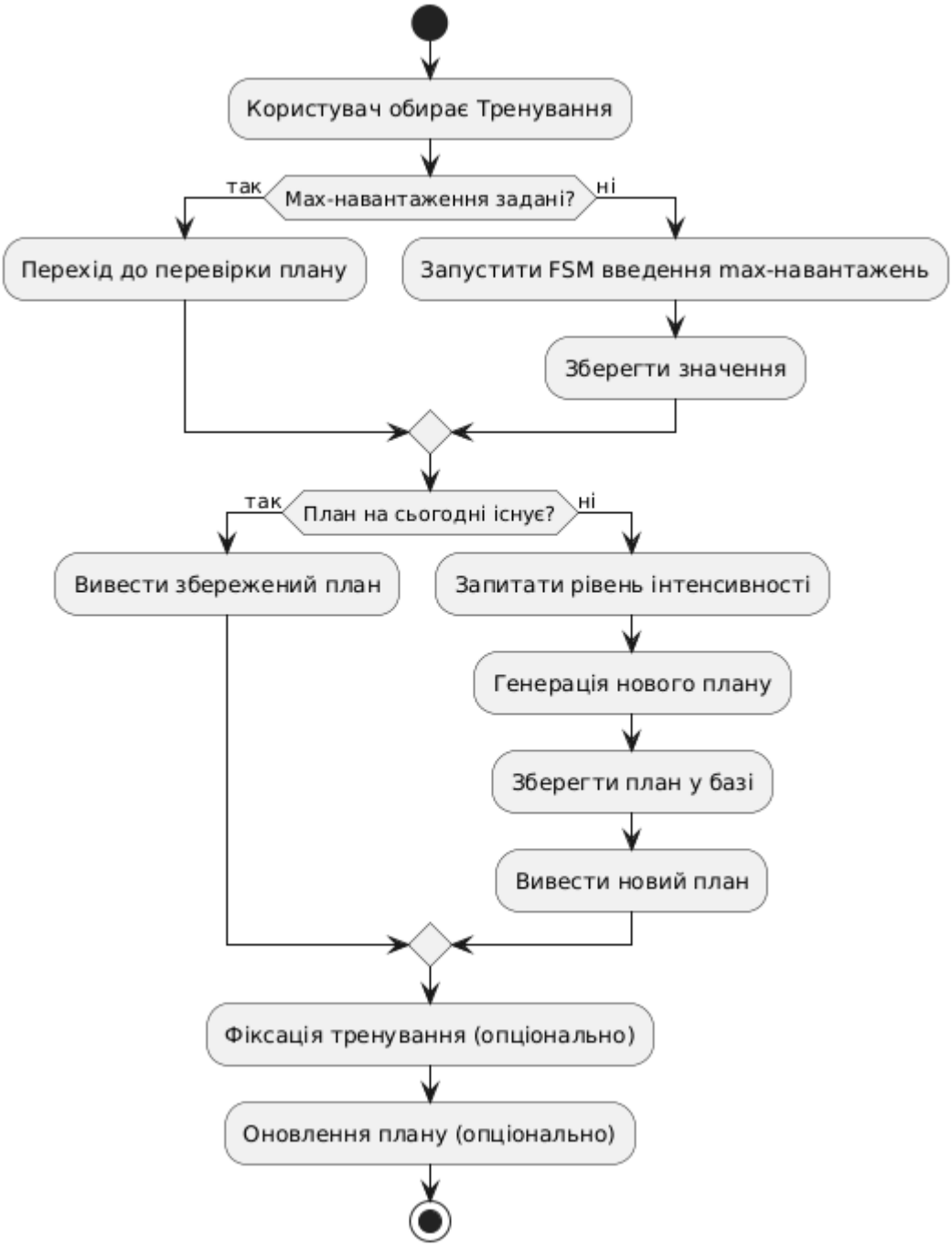

Рис. 2 – Діаграма активності взаємодії користувача з модулем «Тренування»

1.3.3 Діаграма послідовності: «Тренування». На рис. 3 представлено діаграму послідовності, яка описує порядок взаємодії між компонентами системи під час реалізації сценарію отримання тренувального плану. Ця діаграма дозволяє простежити покрокову взаємодію між користувачем, Telegram-ботом, внутрішнім сервісом генерації планів (planner.py) та базою даних.

Після обрання команди «Тренування» користувач ініціює діалог із ботом. Telegram-бот насамперед звертається до бази даних, щоб отримати інформацію про користувача, зокрема його максимальні фізичні показники. Якщо такі дані відсутні, бот запускає FSM-сценарій збору max-навантажень у форматі поетапного запиту (відтискання, присідання, планка). Після успішного збору даних — збереження в базу — відбувається повернення до початкової команди.

У разі наявності всіх необхідних параметрів Telegram-бот перевіряє, чи вже існує запис у таблиці workout_plans для поточної дати. Якщо запис знайдено, бот вибирає відповідні вправи з таблиці workout_plan_exercise та повертає користувачеві план.

Якщо ж плану немає, бот запитує рівень інтенсивності. Після відповіді користувача Telegram-бот передає особисті дані (вік, вага, зріст, max-навантаження) до модуля генерації планів, який повертає сформований перелік вправ з кількістю підходів і повторень. Отримані дані зберігаються у базу даних, після чого надсилаються користувачеві.

Окремим запитом користувач може зафіксувати факт виконання тренування. У цьому разі в таблицю workout_sessions додається запис із поточною датою. Цей етап є опціональним і виконується лише за прямим підтвердженням користувача.

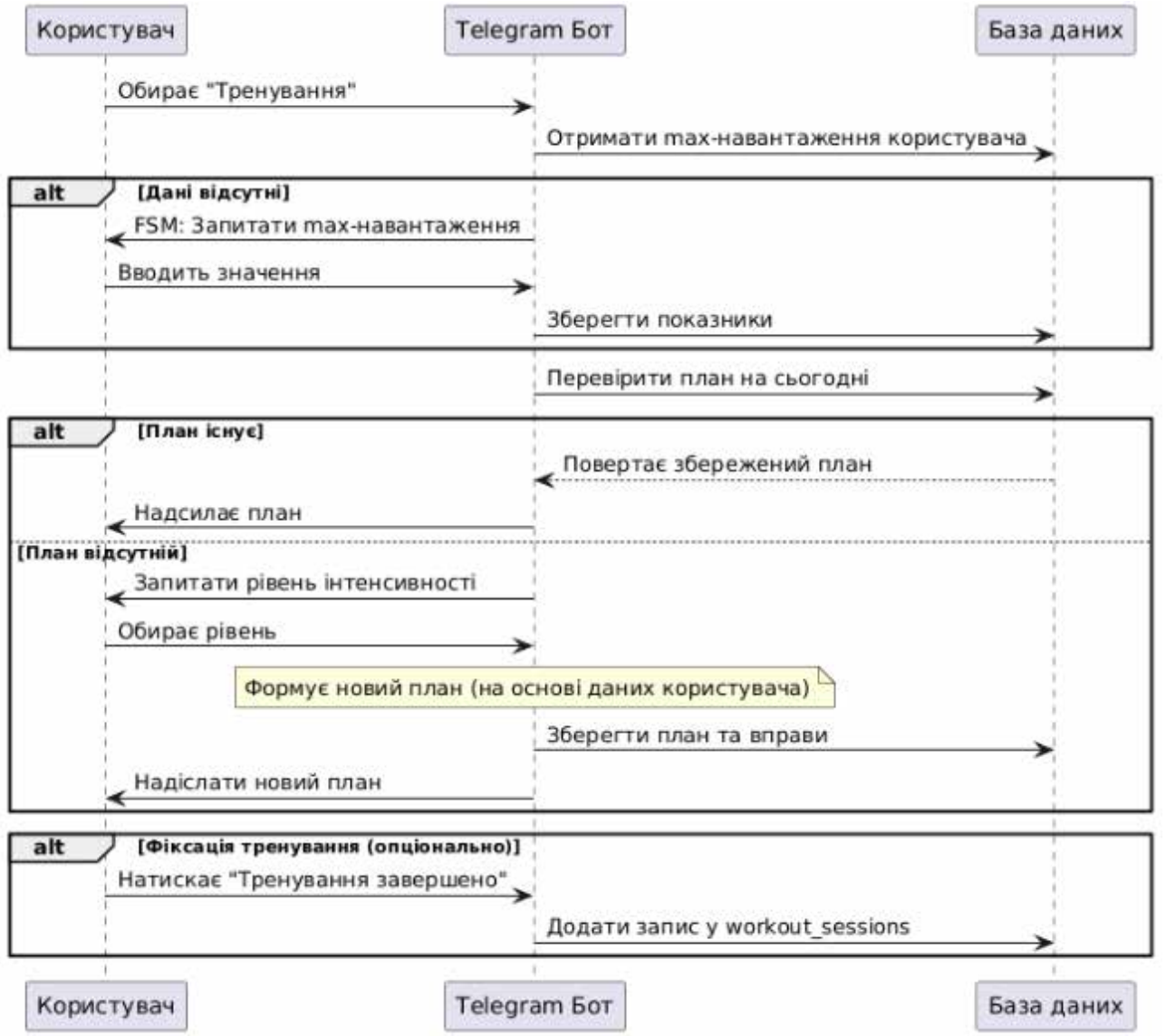

Рис. 3 – Діаграма послідовності взаємодії системи під час формування тренувального плану

1.3.4 Діаграма активності: «Щоденник». На рис. 4 представлено діаграму активності, яка моделює сценарій взаємодії користувача з Telegram-ботом у межах функціоналу «Щоденник». Цей модуль дозволяє вручну вести облік спожитих продуктів, переглядати щоденні та тижневі підсумки, а також редагувати останній запис.

Сценарій починається з обрання користувачем пункту меню «Щоденник». Далі користувачеві пропонується обрати одну з доступних дій: додати новий продукт, видалити останній запис або переглянути продукти за останні 7 днів.

Всі ці дії реалізовані як окремі гілки в межах одного FSM-сценарію з кнопками навігації.

У випадку додавання продукту запускається FSM-сценарій, який складається з двох кроків: введення назви продукту та введення кількості в грамах. Після введення назви бот проводить fuzzy-пошук [3] по базі та пропонує найбільш релевантний результат. Якщо користувач підтверджує вибір, система запитує масу продукту в грамах. У разі введення некоректного значення відбувається повторний запит. Успішно зібрані дані зберігаються у таблиці nutrition_plan_product, після чого користувачеві надсилається підтвердження з підрахунком калорійності та макронутрієнтів.

Операція видалення останнього запису дозволяє видалити останній внесений продукт за поточну дату. Ця дія не потребує введення додаткових даних — бот автоматично вибирає найновіший запис користувача.

Функція перегляду записів за 7 днів виконує запит до бази на предмет продуктів, спожитих за останній тиждень, і групує їх за датами, виводячи структурований список.

Завершальним етапом усіх сценаріїв є повернення користувача до меню дій з можливістю повернутись до головного меню або зробити повторний виклик дії через кнопки.

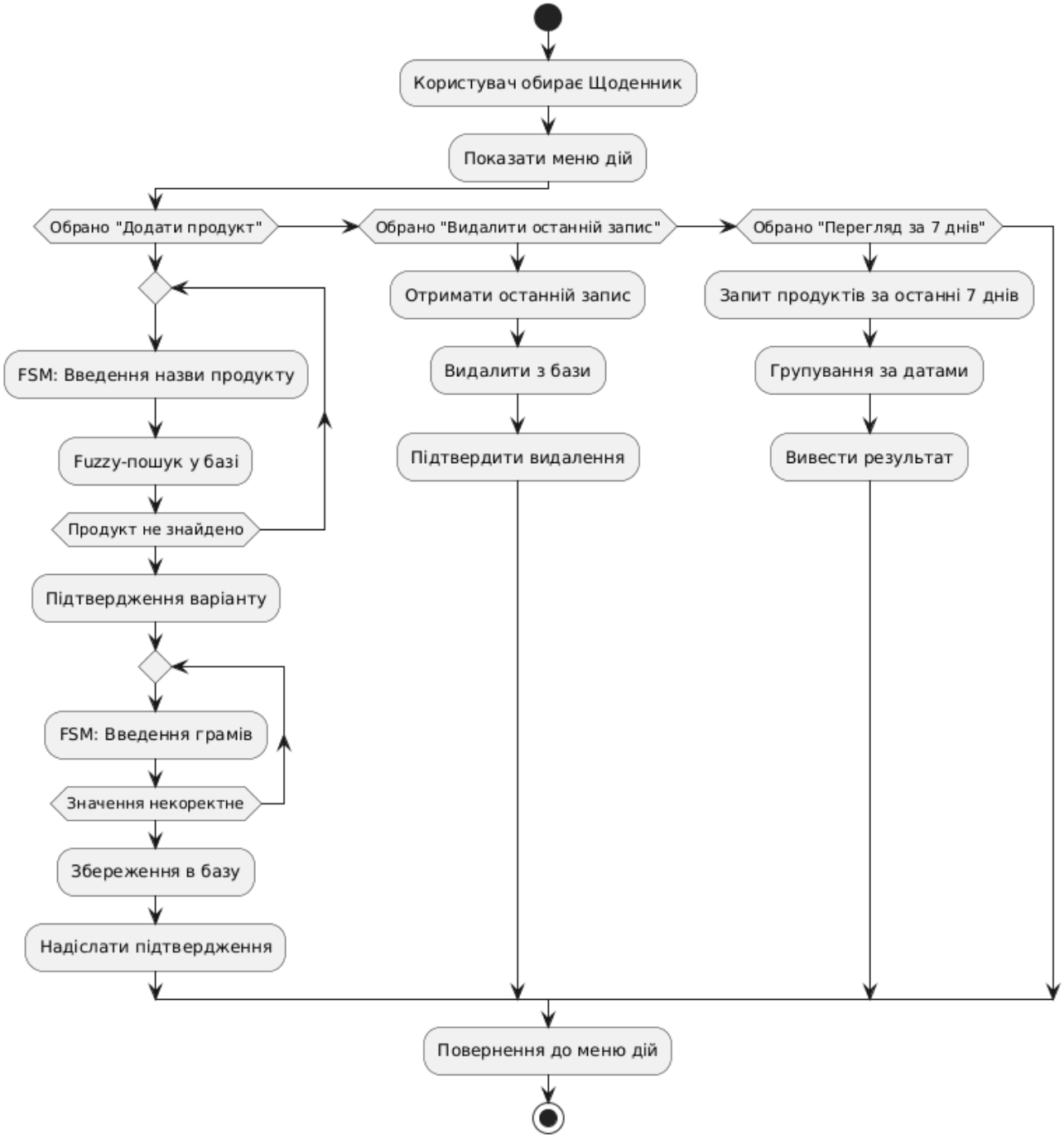

Рис. 4 – Діаграма активності взаємодії користувача з модулем «Щоденник»

1.3.5 Діаграма послідовності: «Щоденник». На рис. 5 представлено діаграму послідовності, що ілюструє логіку взаємодії між користувачем, Telegram-ботом та базою даних у процесі ведення харчового щоденника. Цей модуль є одним із найбільш інтерактивних у системі, оскільки передбачає введення текстових даних, fuzzy-пошук [3], перевірку введеної інформації та обробку кількох різних сценаріїв.

Сценарій починається з ініціації команди /diary або натискання відповідної кнопки. Telegram-бот показує користувачеві меню з варіантами дій. У випадку обрання пункту «Додати продукт» запускається FSM-послідовність, яка включає два основних етапи: введення назви продукту та введення кількості в грамах.

Після отримання текстової назви продукту бот виконує fuzzy-пошук [3] по локальній базі продуктів. Якщо результат не знайдено або рівень схожості нижчий за встановлений поріг, бот просить користувача повторити введення. У разі успішного підбору бот запитує вагу продукту. Якщо користувач вводить некоректне значення, повторюється запит із відповідним повідомленням про помилку.

Після успішного введення даних бот здійснює запис у таблицю nutrition_plan_product, використовуючи дату DATE('now'). Потім бот отримує назву продукту з бази для формування підтверджувального повідомлення, після чого надсилає користувачеві резюме з інформацією про калорійність і макронутрієнти.

Інші сценарії — видалення останнього запису або перегляд за 7 днів — реалізовані як окремі шляхи взаємодії, що також передбачають запити до бази та вивід даних. Ці дії виконуються без FSM, на основі кнопок, і не потребують введення тексту.

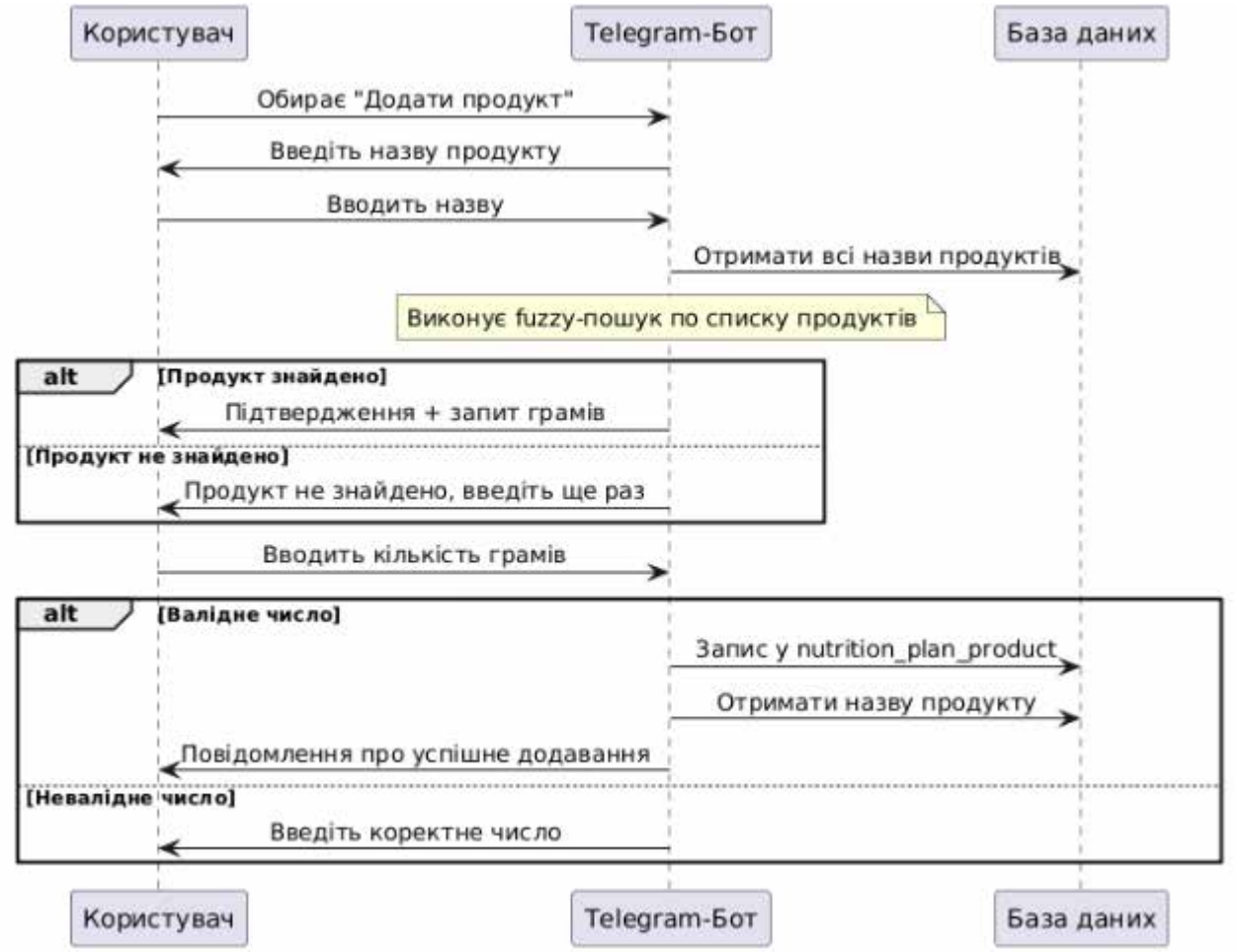

Рис. 5 – Діаграма послідовності взаємодії під час додавання продукту до щоденника

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Проєктування інформаційної бази програмної системи починається зі створення логічної моделі даних. Цей етап дозволяє формалізувати предметну область, виявити основні сутності, їхні атрибути, а також типи зв’язків між елементами системи. У контексті реалізації Telegram-бота для підтримки здорового способу життя було побудовано повноцінну логічну модель, яка охоплює персональні дані користувачів, набір вправ і продуктів, згенеровані плани тренувань та харчування, а також історію взаємодії з ботом.

На (рис. 6) подано ER-діаграму, яка наочно демонструє логічну структуру бази даних системи. Усі таблиці побудовані згідно з принципами реляційного моделювання та приведені до третьої нормальної форми для уникнення надмірності та забезпечення цілісності даних.

Основні таблиці та їхнє призначення:

- users — містить облікові дані користувачів, включаючи ім’я, прізвище, вік, вагу, зріст та максимальні показники фізичної підготовки (відтискання, присідання, планка). Поле id є первинним ключем і використовується як зовнішній ключ в інших таблицях.
- exercises — каталог вправ, який дозволяє зберігати як загальні, так і користувацькі вправи. Зв’язок із користувачем реалізується через поле user_id, що дозволяє додавати індивідуальні вправи.
- products — містить перелік харчових продуктів із зазначенням калорійності та складу макронутрієнтів (білки, жири, вуглеводи) на 100 г.

- workout_plans — таблиця для зберігання сформованих тренувальних планів. Кожен план містить дату створення (date), рівень інтенсивності (intensity) і зовнішній ключ на користувача (user_id).
- nutrition_plans — містить згенеровані харчові плани з добовою нормою калорій (total_calories) для конкретного користувача на певну дату. Використовується зовнішній ключ user_id.
- workout_plan_exercise — зв’язуюча таблиця, що реалізує зв’язок типу «багато до багатьох» між workout_plans та exercises. Містить також поля sets (підходи) і reps (повторення), які задають параметри виконання кожної вправи в межах плану.
- nutrition_plan_product — таблиця, яка фіксує продукти, додані користувачем до щоденника харчування. Містить зовнішні ключі на users та products, а також дату (date) і масу продукту в грамах (grams_quantity). Цей зв’язок не прив’язаний до nutrition_plans, оскільки формує фактичну спожиту інформацію, а не рекомендації.
- workout_sessions — таблиця фактів, яка зберігає дані про завершення тренувань. Містить посилання на user_id та дату завершеного заняття. Використовується для побудови статистики активності.

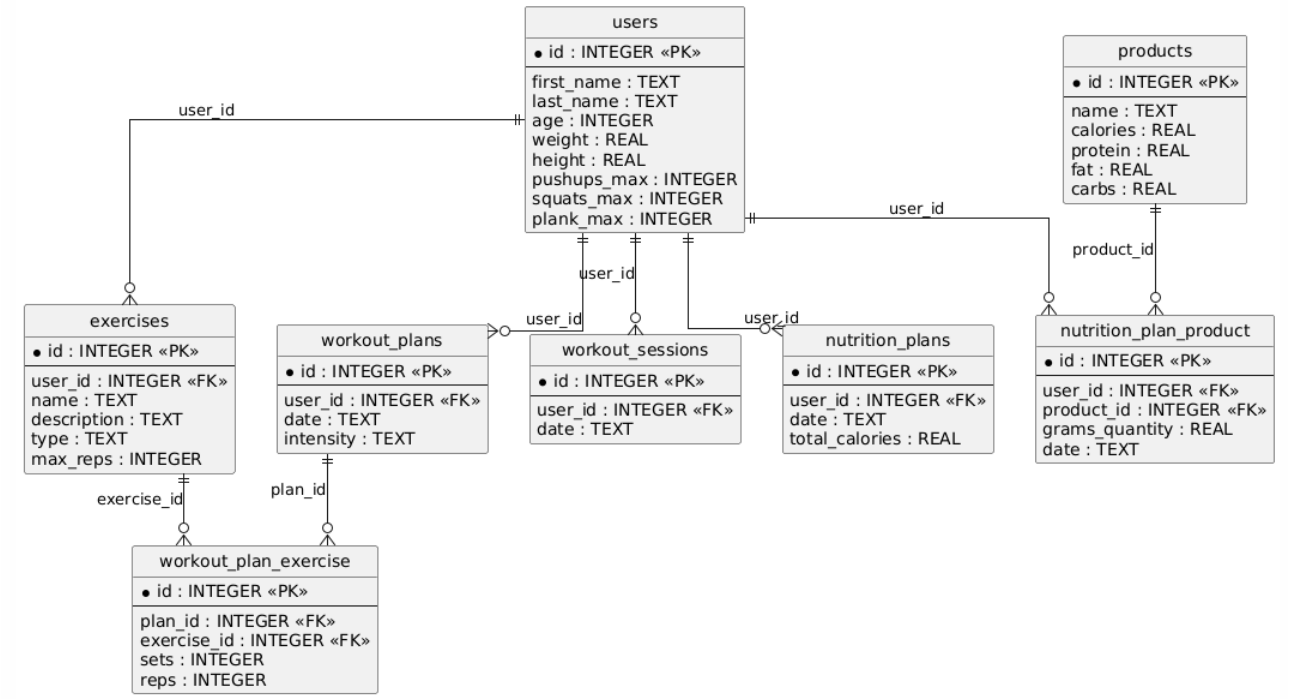

Рис. 6 – ER-діаграма логічної структури бази даних Telegram-бота для підтримки здорового способу життя

Усі зовнішні ключі реалізовані відповідно до вимог цілісності даних. У моделі використовуються як зв’язки типу «один до багатьох», так і «багато до багатьох», зокрема:

Один до багатьох:

- один користувач (users) може мати кілька тренувальних планів (workout_plans) або харчових планів (nutrition_plans);
- один користувач може створити кілька власних вправ (exercises);
- один користувач може додати кілька записів до щоденника харчування (nutrition_plan_product);
- один користувач може завершити кілька тренувань (workout_sessions).

Багато до багатьох (через проміжні таблиці):

- одна вправа (exercises) може входити до кількох тренувальних планів, і навпаки — один тренувальний план (workout_plans) може містити кілька вправ; зв’язок реалізовано через workout_plan_exercise;
- один продукт (products) може бути використаний у багатьох записах щоденника харчування, а користувач може вносити різні продукти — зв’язок реалізовано через nutrition_plan_product.

Модель є повністю реляційною, не містить вкладених структур або об’єктних атрибутів, що забезпечує прозору реалізацію в СУБД SQLite [4]. Такий підхід дозволяє ефективно організувати збереження даних, забезпечити гнучке масштабування, а також полегшує реалізацію запитів через Python-бібліотеку sqlite3.

Логічна модель даних є основою для подальшого фізичного проєктування, генерації таблиць SQL, формування обробників даних, а також забезпечує зв’язок між функціональними модулями системи та її внутрішньою структурою.

2.2 Вибір системи управління інформаційною базою даних

Одним із ключових рішень у процесі проєктування інформаційної системи є вибір системи управління базами даних (СУБД), яка забезпечить ефективне зберігання, обробку та доступ до інформації. Для розробки Telegram-бота, що виконує функції рекомендаційної системи щодо здорового способу життя, важливо враховувати не лише функціональні можливості СУБД, а й обмеження щодо складності розгортання, вимог до середовища виконання, обсягу даних і частоти звернень до бази.

У межах даного проєкту було розглянуто кілька популярних СУБД:

- **PostgreSQL** — потужна об'єктно-реляційна СУБД із широкими можливостями: підтримка транзакцій, розширень, реплікації та складних типів даних. Однак вимагає окремого встановлення, запуску серверного процесу, авторизації користувачів і адміністрування, що є надмірним для локального, компактного Telegram-бота.

- **MySQL / MariaDB** — популярні рішення для веб-застосунків, які потребують серверного компонента. Попри легшу конфігурацію порівняно з PostgreSQL, також не підходять для вбудованих сценаріїв, де потрібна автономність і простота.

- **Microsoft SQL Server** — потужна корпоративна платформа, яка надає розширений функціонал, проте її безкоштовна версія має обмеження, складна в розгортанні та переважно орієнтована на середовище Windows, що не відповідає кросплатформеній природі Python-застосунків.

Firebase Realtime Database — хмарна NoSQL-СУБД від Google, яка зберігає дані у форматі JSON. Підходить для мобільних застосунків, але не підтримує стандартні SQL-запити, не має реляційної моделі, вимагає авторизації через API та постійного інтернет-з’єднання, що суперечить автономній архітектурі Telegram-бота.

MongoDB — документно-орієнтована СУБД, що забезпечує гнучкість структури, однак не має вбудованої підтримки складних реляційних зв’язків типу «один до багатьох» і «багато до багатьох». Це ускладнює реалізацію структури даних, яку потребує бот (наприклад, списки вправ у планах тренувань).

З огляду на перелічені фактори, було обрано SQLite [4] як найбільш придатну систему керування базами даних для цього проєкту. SQLite — це вбудована реляційна СУБД, яка зберігає дані в одному .db-файлі та не вимагає запуску окремого серверного процесу. Вона інтегрується напряму з Python за

допомогою стандартної бібліотеки sqlite3, що виключає необхідність у сторонніх залежностях.

Основні причини вибору SQLite [4]:

- Простота інтеграції з Python — використовується вбудована бібліотека sqlite3, що не потребує встановлення драйверів чи сторонніх пакетів.
- Відсутність серверної частини — база даних функціонує як локальний файл, доступний напряму з коду Telegram-бота.
- Автономна робота — відсутність залежності від інтернету або зовнішніх серверів дає змогу використовувати бота без розгортання на хостингу.
- Портативність — база даних міститься в одному файлі, який легко переносити, копіювати, резервувати чи тестувати.
- Підтримка SQL — реалізовано основний функціонал реляційних СУБД: зовнішні ключі, агрегатні функції, індекси, JOIN-запити.
- Продуктивність — достатня для сценарію з одним активним користувачем, низькою частотою запитів і обмеженим обсягом даних.
- Ліцензія Public Domain — дозволяє вільне використання в навчальних, дослідницьких та комерційних проєктах без обмежень.

Враховуючи те, що Telegram-бот працює у вигляді локального застосунку з одним користувачем одночасно, без потреби масштабування або хмарного зберігання даних, SQLite [4] виявляється найраціональнішим вибором для реалізації інформаційної бази проєкту.

2.3 Створення інформаційної бази

Відповідно до попередньо спроєктованої логічної моделі даних було реалізовано створення фізичної моделі інформаційної бази даних, яка забезпечує надійне та структуроване зберігання всіх необхідних даних системи. Telegram-бот, створений у рамках даного проєкту, активно взаємодіє з базою даних — зчитує персональні параметри користувачів, формує плани тренувань і харчування, зберігає записи щоденника та оновлює статистику. Відтак, коректна організація бази даних і механізмів доступу до неї є критично важливими для стабільної роботи системи.

Для створення та ініціалізації бази даних реалізовано окремий модуль db.py, який містить логіку створення файлу bot.db і функцію init_db(). Ця функція автоматично створює структуру таблиць при першому запуску застосунку. Вона відкриває файл create_tables.sql, зчитує SQL-інструкції як текстовий скрипт і виконує їх за допомогою об’єкта cursor бібліотеки sqlite3. Після завершення всіх операцій з’єднання закривається, що дозволяє уникнути блокувань і втрати ресурсів. Логіка ініціалізації показана на (рис. 7).

```
def init_db():  2 usages
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()

    with open("database/create_tables.sql", "r", encoding="utf-8") as f:
        cursor.executescript(f.read())

    conn.commit()
    conn.close()
```

Рис. 7 – Функція init_db для автоматичного створення таблиць з SQL-файлу

Такий підхід дозволяє відокремити структуру бази (описану в SQL) від логіки самої програми (Python), що підвищує гнучкість і спрощує внесення змін. У разі потреби модифікації схеми достатньо змінити лише SQL-файл, без редагування коду. Всі таблиці створюються з використанням інструкції IF NOT

EXISTS, що забезпечує безпечне багаторазове виконання функції без помилок або дублювання структури.

SQL-структура бази охоплює вісім ключових таблиць: users, exercises, products, workout_plans, nutrition_plans, workout_plan_exercise, nutrition_plan_product, workout_sessions. Усі таблиці відповідають нормам реляційного моделювання, мають чітко визначені первинні ключі, коректні типи даних і зовнішні ключі для реалізації логічних зв’язків між сутностями. Приклад створення таблиці users наведено на рис. 8. З повним кодом create_tables.sql можна ознайомитись в ДОДАТКУ А.

```sql
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    age INTEGER,
    weight REAL,
    height REAL,
    pushups_max INTEGER,
    squats_max INTEGER,
    plank_max INTEGER
);
```

Рис. 8 – Фрагмент SQL-коду створення таблиці users у файлі create_tables.sql

Таблиці проєктувались згідно з вимогами третьої нормальної форми: кожен атрибут функціонально залежить від первинного ключа, відсутні надлишкові дані або вкладені структури. Для реалізації зв’язків типу «багато до багатьох» використовуються проміжні таблиці — workout_plan_exercise та nutrition_plan_product. Наприклад, одна вправа може входити до кількох планів, і навпаки — один план може містити кілька вправ. Аналогічно, один продукт може бути використаний у багатьох записах щоденника.

Крім функції init_db(), у модулі db.py реалізовано також функцію get_connection(), яка відкриває нове з’єднання з базою даних. Вона викликається

в усіх обробниках Telegram-бота і дозволяє централізовано керувати доступом до bot.db, що підвищує надійність та запобігає конфліктам транзакцій.

Однією з переваг використання SQLite [4] є повна автономність: база даних зберігається в одному .db-файлі, який легко переносити, резервувати чи перевіряти вручну.

Для супроводу, верифікації структури та ручного тестування використовувався графічний інструмент **DB Browser for SQLite** [7]. На рис. 9 наведено приклад роботи з інтерфейсом цього застосунку.

Його основні можливості:

- перегляд структури таблиць у візуальному представленні;
- виконання індивідуальних SQL-запитів;
- внесення тестових даних безпосередньо через графічний інтерфейс;
- перевірка зовнішніх ключів, типів даних і індексів;
- експорт і імпорт таблиць у форматі CSV.

Рис. 9 – Перегляд структури бази даних у DB Browser for SQLite

У підсумку, реалізація інформаційної бази відбувається повністю автоматизовано — без втручання користувача або потреби в окремих налаштуваннях. Архітектура БД є компактною, логічною, повністю

інтегрованою у програмну логіку Telegram-бота. Це забезпечує стабільність, повторюваність та простоту обслуговування системи, відповідаючи як вимогам академічного проєкту, так і реальним технічним потребам.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Telegram-бот, створений у межах цієї бакалаврської роботи, реалізований за модульною архітектурою, яка забезпечує чітке розділення функцій між окремими компонентами системи. Весь код упорядковано у вигляді структурованих каталогів і логічно відокремлених модулів, кожен з яких виконує визначену роль: обробка Telegram-команд, формування рекомендацій, робота з базою даних, взаємодія з користувачем тощо.

На (рис. 10) наведено фактичну структуру проєкту у середовищі розробки PyCharm [8]. Вона демонструє організацію вихідних файлів і каталогів згідно з логічним поділом за призначенням.

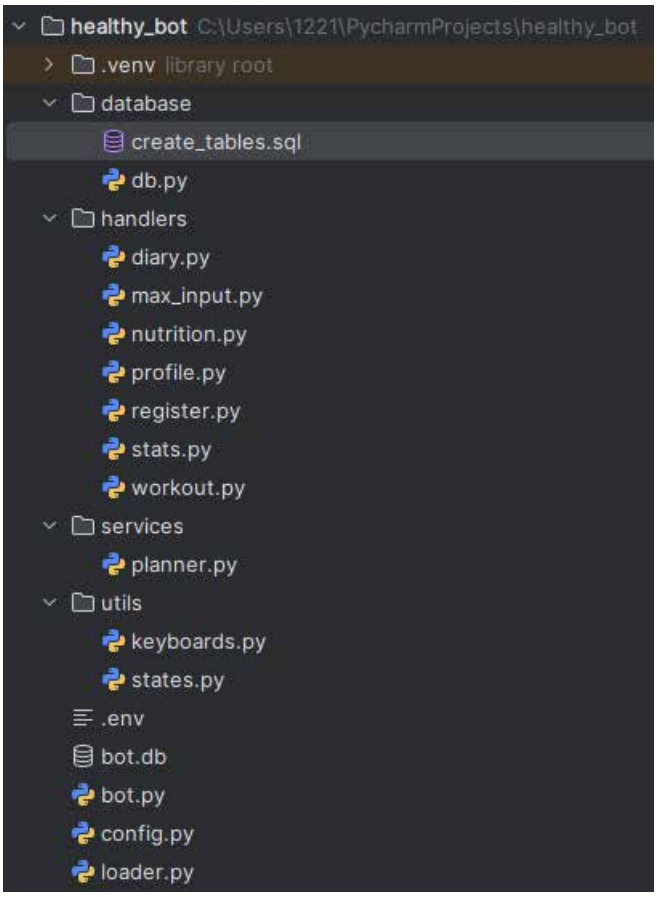

Рис. 10 – Структура проєкту Telegram-бота в середовищі PyCharm

Узагальнене представлення архітектури подано на рис. 11 у вигляді структурної діаграми пакетів. Вона відображає залежності між основними модулями системи, а також логічні рівні — основний запуск, обробники команд, база даних, сервіси генерації, утиліти.

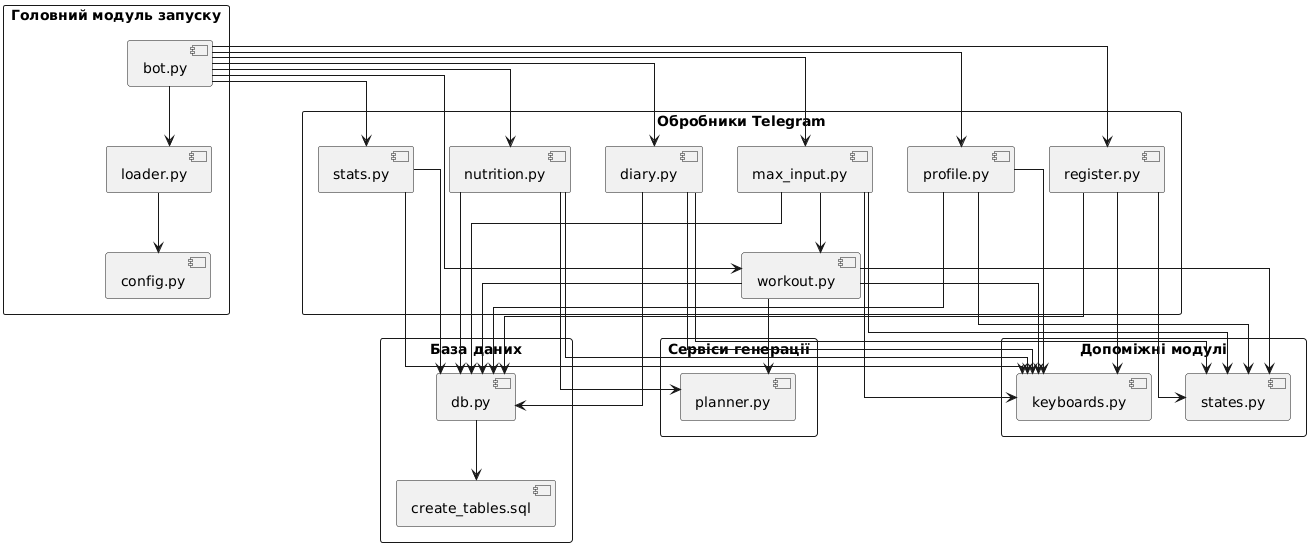

Рис. 11 – Діаграма пакетів організації програмного забезпечення Telegram-бота

Основні елементи структури:

- bot.py – головний модуль запуску. Відповідає за ініціалізацію бази даних (через init_db()), імпортує всі обробники та запускає цикл обробки повідомлень executor.start_polling(...). З повним кодом головного модуля можна ознайомитись в додатку Б.
- config.py – завантажує налаштування, зокрема токен бота з файлу .env, за допомогою бібліотеки dotenv [6].
- loader.py – створює глобальні об’єкти: bot, dp (dispatcher) та storage (для FSM), доступні в усіх модулях.
- .env – файл конфігурації середовища (наприклад, токен бота), не включається до репозиторію.
- bot.db – файл SQLite-бази даних, що містить усі таблиці, створені за SQL-скриптом create_tables.sql.

Каталоги:

- handlers/ – обробники Telegram-команд:
    - register.py – реєстрація нового користувача з використанням FSM;
    - profile.py – перегляд і редагування персонального профілю;
    - workout.py – отримання та оновлення тренувального плану, фіксація тренувань;
    - max_input.py – FSM-сценарій введення максимальних фізичних показників (відтискання, присідання, планка);
    - nutrition.py – створення харчового плану на день;
    - diary.py – додавання продуктів до харчового щоденника;
    - stats.py – перегляд статистики за харчуванням і тренуваннями.
- database/ – робота з базою даних:
    - create_tables.sql – SQL-інструкції для створення таблиць;
    - db.py – функції init_db() та get_connection().
- services/ – бізнес-логіка генерації планів:
    - planner.py – реалізація функцій generate_workout_plan() і generate_nutrition_plan() на основі параметрів користувача.
- utils/ – допоміжні модулі:
    - keyboards.py – створення інлайн-клавіатур Telegram;
    - states.py – визначення FSM-станів для обробки сценаріїв (реєстрація, профіль, щоденник, max-навантаження).

Така архітектура дозволяє:

- розділити бізнес-логіку, Telegram-інтерфейс, обробку даних та генерацію планів;
- забезпечити гнучкість підтримки та простоту масштабування;
- окремо тестувати кожен модуль без дублювання коду;
- інтегрувати нові сценарії без порушення існуючої структури.

Telegram-бот реалізований з використанням фреймворку Aiogram [2], який підтримує асинхронну обробку повідомлень, FSM-підхід для керування сценаріями взаємодії та зручне створення інлайн-клавіатур. Це забезпечує стабільність, модульність і можливість подальшого розширення функціоналу без порушення архітектурної цілісності.

3.2 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі розробки Telegram-бота для рекомендаційної підтримки здорового способу життя було обґрунтовано обрано набір інструментів, які відповідають сучасним вимогам до прикладного програмного забезпечення: висока швидкість створення прототипу, підтримка асинхронної обробки, простота розгортання, кросплатформеність, інтеграція зі сторонніми API та зручність роботи з базами даних.

Основною мовою програмування було обрано Python версії 3.10 [1]. Вона поєднує лаконічний синтаксис, швидкість розробки, широку підтримку бібліотек і потужну спільноту. Python ідеально підходить для створення Telegram-ботів завдяки легкості роботи з API, підтримці асинхронного програмування, а також наявності рішень для обробки даних і логіки взаємодії з користувачем.

Для реалізації Telegram-інтерфейсу було використано Aiogram [2]— сучасну асинхронну бібліотеку на базі asyncio, яка надає повний доступ до Telegram Bot API. На відміну від синхронних альтернатив, таких як python-telegram-bot, Aiogram [2] підтримує асинхронну обробку подій "з коробки", дозволяє масштабувати застосунок, знижує затримки та краще підходить для багатопотокових сценаріїв.

Aiogram [2] надає низку можливостей, критичних для реалізації логіки Telegram-бота:

- підтримка Finite State Machine (FSM) для побудови діалогових сценаріїв (реєстрація, введення фізичних показників, щоденник);
- повна інтеграція з inline- і callback-кнопками для побудови інтерактивного меню;
- зручне розділення обробників за модулями;
- асинхронність обробки запитів користувача.

У якості середовища розробки було використано PyCharm Community Edition [8] — повнофункціональна IDE, яка забезпечує автодоповнення, підсвічування синтаксису, роботу з віртуальними середовищами, інтеграцію з Git, вбудований дебагер та інші можливості. У цьому середовищі було реалізовано основну частину коду, проведено тестування та налагодження структури проєкту.

Для зберігання даних було обрано SQLite [4] — легковагову реляційну СУБД, яка зберігає всю інформацію в одному .db-файлі. Це рішення ідеально підходить для Telegram-бота, що працює локально без багатокористувацького доступу. SQLite не потребує встановлення окремого сервера, конфігурації клієнтів чи авторизації. База створюється автоматично при першому запуску застосунку за допомогою SQL-скрипту create_tables.sql.

У Python взаємодія з базою реалізована через стандартну бібліотеку sqlite3, яка дозволяє:

- створювати з’єднання;
- виконувати запити;
- керувати транзакціями;
- забезпечувати цілісність даних.

Окремий SQL-файл з описом таблиць (create_tables.sql) дозволяє зберігати структуру бази окремо від коду, що спрощує її модифікацію та повторне застосування.

Для ручного тестування, супроводу та візуальної перевірки структури БД використовувався DB Browser for SQLite [7] — безкоштовний графічний інструмент, який дозволяє:

- переглядати структуру таблиць;
- виконувати SQL-запити вручну;
- додавати тестові дані;
- перевіряти зовнішні ключі;
- експортувати та імпортувати дані у форматі CSV.

Для зберігання конфіденційних налаштувань, таких як токен бота, використовувався файл .env, зчитування з якого реалізовано через бібліотеку python-dotenv [6]. Такий підхід є рекомендованою практикою безпечної розробки: токен не зберігається у коді або публічному репозиторії, а параметри легко адаптуються до різного середовища (розробка, продакшн тощо).

У процесі розробки застосовувалися також стандартні модулі Python:

- random — для генерації псевдовипадкових тренувальних і харчових планів;
- collections — для зведення статистики (середні показники за тиждень);
- datetime — для роботи з датами, фільтрації записів;
- os — для взаємодії з операційним середовищем, зокрема з .env.

У підсумку, обраний стек технологій забезпечив швидку розробку, гнучку архітектуру, ефективне налагодження та повну автономність Telegram-бота. Усі використані компоненти є вільними для застосування, кросплатформеними,

добре документованими та мають широку підтримку. Це дозволяє масштабувати систему, інтегрувати її з іншими сервісами або адаптувати до змін без серйозних витрат часу та ресурсів.

3.3. Алгоритмізація та програмування програмних модулів

3.3.1 Реєстрація користувача. Реєстрація є першим кроком взаємодії користувача з Telegram-ботом. Її мета — зібрати ключові персональні параметри (ім’я, прізвище, вік, вага, зріст), необхідні для подальшої персоналізації тренувального та харчового плану.

У програмному коді алгоритм реалізовано як послідовну FSM-модель (Finite State Machine), описану у класі RegistrationFSM модуля states.py. Вся логіка сценарію розміщена в обробнику register.py, який реагує на команду /register та запускає покроковий збір даних через dp.message_handler(state=...).

Кожен етап верифікує коректність введених даних. Наприклад:

- вік має бути в межах 5–120 років;
- вага — 20–300 кг;
- ім’я й прізвище — лише літери (довжина до 30 символів).

У випадку некоректного вводу бот надає повідомлення з помилкою і повертає користувача до відповідного кроку FSM. Після проходження всіх етапів зібрані дані зберігаються у базу даних через модуль db.py. Для цього використовується функція get_connection(), яка відкриває з’єднання з bot.db, після чого виконується INSERT INTO users (...).

На (рис. 10) представлено блок-схему реєстрації, яка починається з перевірки наявності користувача в таблиці users, далі охоплює етапи збору інформації та завершується збереженням даних.

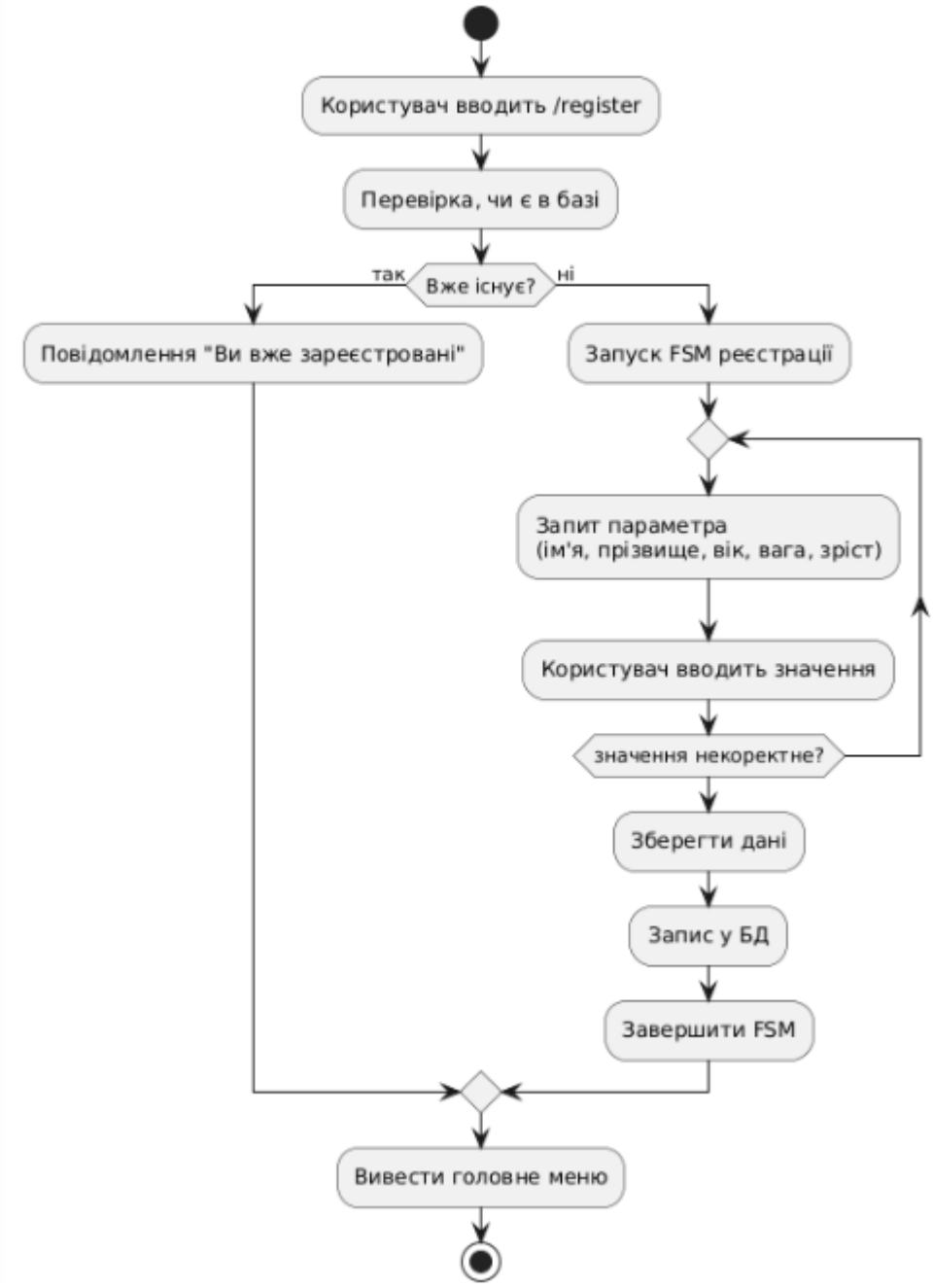

Рис. 10 – Блок-схема алгоритму реєстрації нового користувача у Telegram-боті

Після успішного завершення алгоритму користувач отримує повідомлення про завершення реєстрації, а також кнопки переходу до головного меню. Повторне проходження реєстрації не допускається — при повторному виклику команди /register користувач отримає відповідне попередження.

На рис. 11 наведено фрагмент коду, який реалізує фінальну частину сценарію: запис користувача до бази та повернення до головного меню.

```
data = await state.get_data()
conn = get_connection()
cursor = conn.cursor()
cursor.execute(
    """INSERT INTO users (id, first_name, last_name, age, weight, height)
    VALUES (?, ?, ?, ?, ?, ?)""",
    (
        message.from_user.id,
        data["first_name"], data["last_name"], data["age"],
        data["weight"], data["height"]
    )
)
conn.commit()
conn.close()

await state.finish()
await message.answer( text: "✅ Реєстрація завершена!", reply_markup=ReplyKeyboardRemove())
await bot.send_message(message.from_user.id, "Оберіть дію з меню 👇", reply_markup=get_inline_menu(True))
```

Рис. 11 – Завершення реєстрації: збереження даних у базу та повернення до головного меню

Таким чином, сценарій реєстрації реалізовано як стійкий до помилок багатокроковий процес із перевіркою кожного введеного значення. Це дозволяє забезпечити цілісність даних і коректну роботу усіх модулів, що залежать від персональних параметрів користувача.

3.3.2 Генерація тренувального плану. Генерація тренувального плану є одним із ключових функціональних сценаріїв Telegram-бота. Її мета — сформувати персоналізований набір вправ відповідно до фізичних показників користувача та обраного рівня інтенсивності (легкий, середній або важкий).

Уся логіка реалізована в модулі workout.py. Виклик здійснюється через команду /workout або натискання кнопки з головного меню. Основна функція send_workout_plan() спочатку перевіряє, чи вказані користувачем максимальні показники: кількість відтискань, присідань та тривалість планки. Якщо хоча б одне з цих значень відсутнє, бот активує FSM-сценарій ask_for_max_metrics() з модуля max_input.py, який у покроковому режимі запитує дані та зберігає їх до таблиці users.

Після завершення введення або якщо дані вже є, бот перевіряє, чи існує тренувальний план на поточну дату у таблиці workout_plans. Якщо план уже

збережений — його ID та рівень інтенсивності витягуються з бази, після чого бот формує повідомлення з детальним описом вправ, отриманих з таблиці workout_plan_exercise.

Якщо план відсутній, бот запитує рівень інтенсивності через інлайн-кнопки (light, medium, hard) і звертається до функції generate_and_send_new_plan(). Ця функція отримує особисті дані користувача з бази (age, weight, height, pushups_max, squats_max, plank_max), після чого викликає сервісну функцію generate_workout_plan() з модуля planner.py. На основі цих параметрів і обраного рівня інтенсивності функція обчислює кількість підходів (sets) та повторень (reps) для трьох базових вправ, використовуючи попередньо задані коефіцієнти (0.5, 0.7, 0.85).

Сформований план записується у таблицю workout_plans, а окремо кожна вправа — у таблицю workout_plan_exercise. Потім користувачу виводиться повідомлення з описом плану та кнопками дій: «завершити тренування», «оновити план», «повернутись до меню».

Окремий обробник workout_done_callback() фіксує факт виконання тренування, додаючи відповідний запис у таблицю workout_sessions. Цей механізм використовується в модулі статистики для обрахунку активності за останні 30 днів.

На рис. 12 представлено спрощену блок-схему, яка демонструє загальну логіку реалізації сценарію: від перевірки max-навантажень до збереження нового плану.

Рис. 12 – Алгоритм генерації тренувального плану для користувача

Фрагмент коду, що наведено на рис. 13, реалізує фінальну частину генерації: збереження згенерованого плану у таблицю workout_plans та запис конкретних вправ до workout_plan_exercise.

```
cursor.execute("""
    INSERT INTO workout_plans (user_id, date, intensity)
    VALUES (?, DATE('now'), ?)
""", (user_id, level))
plan_id = cursor.lastrowid

inserted_count = 0
for exercise in plan:
    cursor.execute("SELECT id FROM exercises WHERE name = ?", (exercise["name"],))
    row = cursor.fetchone()

    if not row:
        await bot.send_message(user_id, f"⚠ Вправа '{exercise['name']}' не знайдена в базі й пропущена.")
        continue

    exercise_id = row[0]
    cursor.execute("""
        INSERT INTO workout_plan_exercise (plan_id, exercise_id, sets, reps)
        VALUES (?, ?, ?, ?)
    """, (plan_id, exercise_id, exercise["sets"], exercise["reps"]))
```

Рис. 13 – Збереження згенерованих вправ тренувального плану в базу даних

Таким чином, модуль тренування реалізує багатоступеневу логіку перевірок, обробки введення, генерації й збереження результатів у базу. Він враховує індивідуальні особливості користувача та дозволяє формувати актуальні персоналізовані рекомендації для щоденних тренувань.

3.3.3 Формування плану харчування. Формування плану харчування є важливою функцією Telegram-бота, яка дозволяє користувачу отримати персоналізовану добову норму калорій та список рекомендованих продуктів на основі своїх фізичних параметрів. Цей модуль забезпечує швидке надання раціону без потреби у складних діях з боку користувача.

Основна логіка реалізована у модулі nutrition.py. Після виклику команди /nutrition або натискання відповідної кнопки, бот запускає функцію send_nutrition_plan(), яка виконує перевірку наявності вже створеного плану

харчування для поточної дати. Якщо такий запис існує, він попередньо видаляється, і генерується новий план, що гарантує його актуальність.

Для створення рекомендацій бот отримує з бази даних персональні параметри користувача (age, weight, height) і передає їх до функції generate_nutrition_plan() з модуля planner.py. Ця функція реалізує розрахунок добової потреби в калоріях за формулою Mifflin–St Jeor [11], з урахуванням сидячого стилю життя. На основі цієї енергетичної норми додатково визначаються цільові значення споживання білків, жирів і вуглеводів.

Далі з наперед визначеного набору шаблонів продуктів випадковим чином обирається один, після чого розраховується рекомендована кількість грамів кожного продукту так, щоб наближено покрити 100% добової потреби в калоріях. Якщо підсумкова калорійність виходить за межі ±10% від цілі, порції масштабуються. Результатом є зібраний список продуктів з розрахованою енергетичною цінністю та макронутрієнтами.

Важливо зазначити, що цей набір продуктів не зберігається у базі даних, а лише використовується для одноразового виведення повідомлення користувачеві і сам по собі є рекомендацією. У базі (nutrition_plans) зберігається лише обчислене значення total_calories на поточну дату.

На рис. 14 наведено блок-схему, яка відображає логіку генерації рекомендацій — від отримання параметрів до формування відповіді.

Рис. 14 – Алгоритм формування індивідуального плану харчування

Фрагмент коду на (рис. 15) демонструє, як саме записується значення total_calories у таблицю nutrition_plans після виклику функції генерації.

```
age, weight, height = user
plan = generate_nutrition_plan({"age": age, "weight": weight, "height": height})

cursor.execute("""
    INSERT INTO nutrition_plans (user_id, date, total_calories)
    VALUES (?, DATE('now'), ?)
""", (user_id, plan["total_calories"]))
conn.commit()
conn.close()
```

Рис. 15 – Збереження обчисленої добової калорійності у базу даних

Таким чином, модуль формування харчового плану працює як рекомендаційна підсистема: надає персоналізований набір продуктів без збереження в базі, але водночас забезпечує точну фіксацію енергетичної норми, що слугує орієнтиром для контролю споживання протягом дня.

3.3.4 Ведення щоденнику харчування. Модуль щоденника харчування реалізує механізм ручного обліку фактично спожитих продуктів протягом дня. На відміну від автоматично сформованого раціону, цей компонент дозволяє фіксувати реальні прийоми їжі, що суттєво підвищує точність та персоналізацію рекомендацій системи.

При виклику команди /diary або натисканні відповідної кнопки бот звертається до таблиці nutrition_plan_product, обираючи всі записи користувача за поточну дату. На основі приєднаних даних із таблиці products обчислюється сумарна калорійність, кількість білків, жирів та вуглеводів. Ці дані виводяться у вигляді зведеного повідомлення разом із кнопками дій.

Користувач може обрати одну з доступних дій:

- додати новий продукт;
- видалити останній запис;
- переглянути продукти за останні 7 днів;
- повернутися до головного меню.

При додаванні нового продукту запускається FSM-сценарій Diary (модуль states.py), який складається з двох послідовних етапів:

Введення назви продукту — бот виконує пошук найближчого збігу по назві, використовуючи алгоритм fuzzy matching (rapidfuzz.process.extractOne()) з порогом 60%.

Введення кількості в грамах — бот виконує перевірку, чи введене значення є числом, і лише тоді заносить його у таблицю nutrition_plan_product.

Успішно доданий продукт миттєво фіксується в базі даних із поточною датою. Користувач отримує підтвердження із зазначенням назви, ваги продукту та автоматично оновленими сумарними показниками.

На рис. 16 представлено блок-схему сценарію додавання нового запису до щоденника — від запуску FSM до запису в базу.

Рис. 16 – Алгоритм додавання продукту до щоденника харчування

Фрагмент коду, наведений на рис. 17, демонструє реалізацію другого кроку FSM — запису грамів та збереження у базу даних.

```
cursor.execute("""
    INSERT INTO nutrition_plan_product (product_id, grams_quantity, date, user_id)
    VALUES (?, ?, DATE('now'), ?)
""", (product_id, grams, user_id))
```

Рис. 17 – Додавання продукту користувачем до щоденника харчування

Опція «Видалити останній запис» виконує SQL-запит, який знаходить останній внесений продукт за сьогоднішню дату (сортування ORDER BY id DESC LIMIT 1), після чого видаляє його з таблиці nutrition_plan_product. Ця дія дозволяє швидко скасувати помилковий або зайвий запис. Вона не потребує додаткового введення — все виконується автоматично після натискання кнопки.

На рис. 18 показано алгоритм логіки видалення останнього запису — від запиту до підтвердження дії.

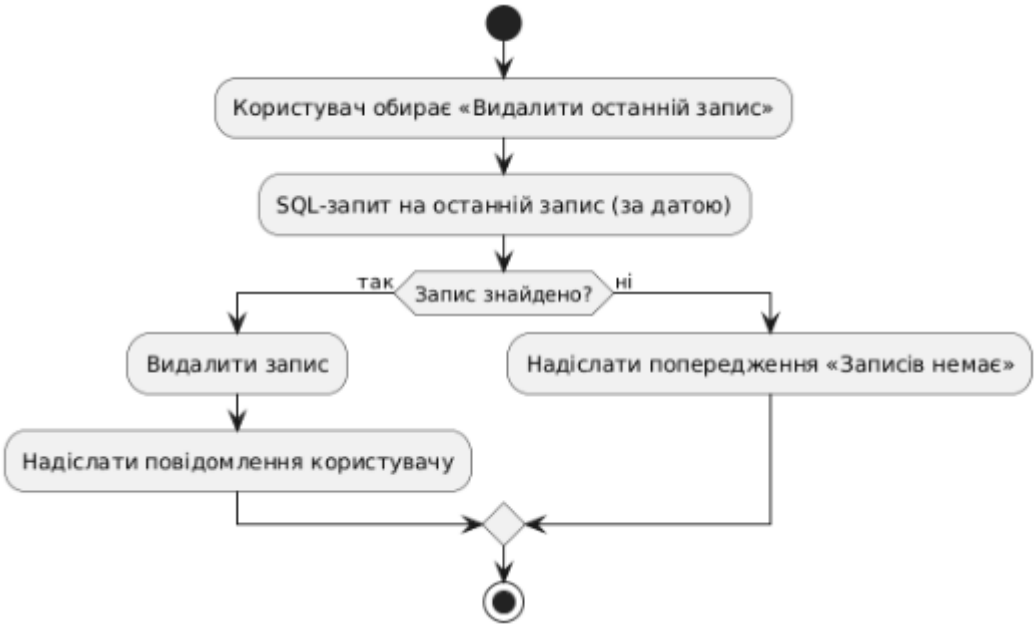

Рис. 18 – Алгоритм видалення останнього запису з щоденника харчування

Відповідний код, наведений на рис. 19, реалізує SQL-запит для вибору і видалення останнього запису, а також надсилає користувачу повідомлення про успішне виконання дії.

```
cursor.execute("""
    SELECT npp.id, p.name, npp.grams_quantity, p.calories
    FROM nutrition_plan_product npp
    JOIN products p ON p.id = npp.product_id
    WHERE npp.user_id = ? AND npp.date = DATE('now')
    ORDER BY npp.id DESC LIMIT 1
""", (user_id,))
row = cursor.fetchone()

if row:
    record_id, name, grams, calories_per_100 = row
    cursor.execute("DELETE FROM nutrition_plan_product WHERE id = ?", (record_id,))
    conn.commit()
    msg = f"🗑 Видалено останній запис: {grams} г {name}"
else:
    msg = "⚠️ У вас ще немає записів у щоденнику за сьогодні."
```

Рис. 19 – Видалення останнього запису з щоденника та оновлення виводу

Таким чином, модуль щоденника харчування дозволяє реалізувати точний контроль над споживанням їжі користувачем, а також динамічно переглядати й редагувати вміст за поточний день. Його реалізація базується на поєднанні FSM-сценаріїв, онлайн-інтерфейсу та роботи з базою даних, що забезпечує простоту користування та ефективність персоналізованого обліку.

3.3.5 Профіль користувача. Модуль роботи з профілем реалізує механізм перегляду й редагування персональних даних користувача. Саме на основі цієї інформації надалі формуються індивідуальні плани тренувань і харчування, тому її актуальність є критично важливою для коректного функціонування системи.

Після виклику команди /profile або натискання відповідної кнопки Telegram-бот звертається до бази даних (users) і перевіряє, чи існує запис про користувача. Якщо користувач ще не зареєстрований — бот надсилає відповідне

повідомлення з закликом пройти первинну реєстрацію. У протилежному випадку користувач отримує картку профілю з поточними значеннями: ім’я, прізвище, вік, вага, зріст, максимальна кількість відтискань, присідань та тривалість планки.

Нижче розміщуються інлайн-кнопки: «Редагувати профіль» та «Головне меню». У разі вибору першої запускається FSM-сценарій ProfileEditFSM, який реалізує послідовне заповнення полів з перевіркою на коректність введених даних, наприклад:

- ім’я та прізвище — тільки літери, до 30 символів;
- вік — від 5 до 120 років;
- вага — у межах 20–300 кг;

Після введення всіх значень у FSMContext, бот отримує поточний стан профілю і виконує оновлення. Якщо запис уже існує — дані оновлюються через UPDATE. Якщо користувача не було — створюється новий запис через INSERT. Завершується FSM-комунікація повідомленням про успішне оновлення профілю та кнопками для повернення.

Фрагмент коду, що наведено на рис. 20, демонструє частину логіки, яка безпосередньо відповідає за збереження нового або оновленого запису користувача.

```
cursor.execute("""
    UPDATE users SET
        first_name = ?, last_name = ?, age = ?, weight = ?, height = ?,
        pushups_max = ?, squats_max = ?, plank_max = ?
    WHERE id = ?
""", (
    data["first_name"], data["last_name"], data["age"], data["weight"], data["height"],
    data["pushups_max"], data["squats_max"], data["plank_max"], message.from_user.id
))
```

Рис. 20 – Оновлення персональних даних користувача під час редагування профілю

На рис. 21 подано блок-схему загальної алгоритм логіки редагування профілю — від запиту до підтвердження дії.

Рис. 21 – Алгоритм редагування профілю користувача

Завдяки структурованості FSM, точковим валідаціям і інтеграції з інлайн-інтерфейсом, модуль профілю дозволяє забезпечити точність, зручність і повний контроль користувача над персональними параметрами в системі.

3.3.6 Статистика користувача. Модуль статистики реалізовано як аналітичний інструмент, що дозволяє користувачу відстежувати динаміку своєї фізичної активності та харчування. Він об'єднує зведену інформацію у вигляді повідомлення та забезпечує графічну деталізацію, що візуалізує динаміку споживання макронутрієнтів і виконаних тренувань. Для побудови графіків використовується бібліотека matplotlib [16], яка дозволяє створювати інтерактивні діаграми безпосередньо в Telegram-інтерфейсі. Такий підхід підвищує обізнаність користувача та сприяє мотивації до підтримки здорового способу життя.

Після виклику команди /stats або натискання відповідної кнопки Telegram-бот виконує запити до таблиць nutrition_plan_product, products та workout_sessions. Основна логіка реалізована у функції show_stats(), яка:

- обчислює середнє споживання калорій, білків, жирів і вуглеводів за останні 7 днів;
- підраховує кількість завершених тренувань за останні 30 днів.

На основі отриманих даних формується підсумкове повідомлення, яке виводиться користувачу у структурованому вигляді разом із кнопками для отримання детальної інформації. На рис. 22 представлено алгоритм цієї логіки — від виконання запитів до виводу даних або обробки їхньої відсутності.

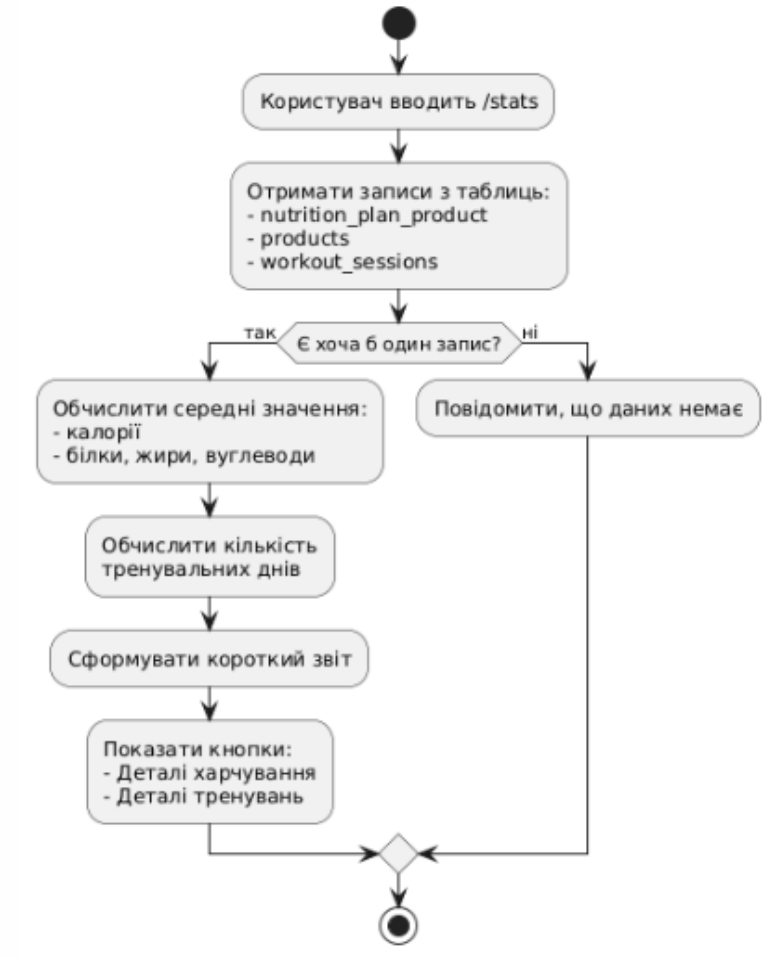

Рис. 22 – Алгоритм формування зведеної статистики користувача

Крім зведених даних, реалізовано також дві функції деталізації:

- по харчуванню — з графічною візуалізацією щоденного споживання калорій, білків, жирів і вуглеводів;

- по тренуваннях — із графічною індикацією днів, коли тренування було виконано або пропущено.

Оскільки обидва сценарії мають подібну структуру взаємодії (запит > групування > вивід), їх зручно узагальнити однією спільною блок-схемою. На рис. 23 представлено алгоритм деталізації, який описує послідовність дій для обох випадків.

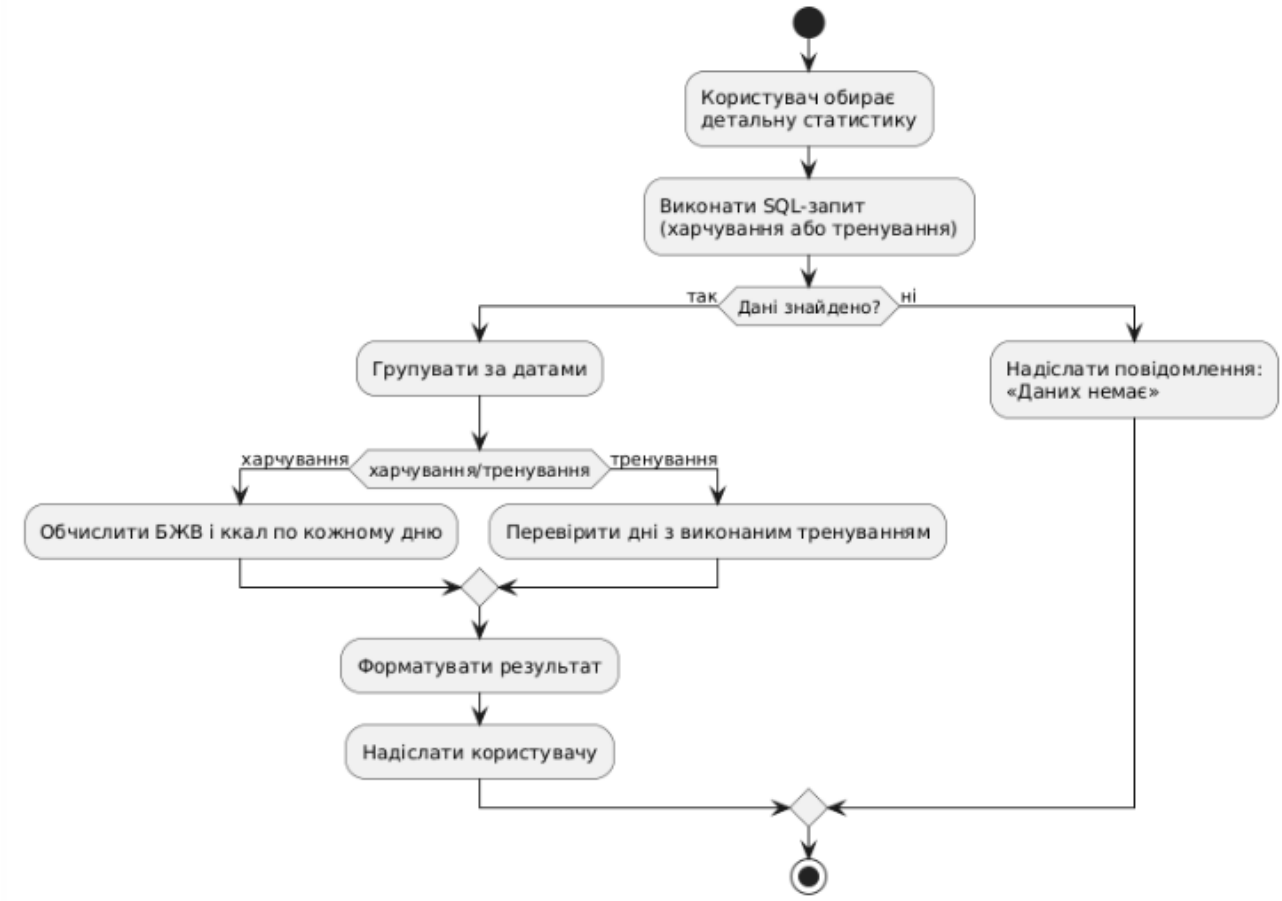

Рис. 23 – Алгоритм деталізації статистики користувача (харчування або тренування)

У разі відсутності даних користувач отримує повідомлення з попередженням, після чого може повернутись назад до зведеного звіту.

На рис. 24 показано фрагмент коду функції show_stats() — точки входу до блоку, яка здійснює агрегацію значень та формує короткий звіт.

```
cursor.execute("""
    SELECT date, calories, protein, fat, carbs, grams_quantity
    FROM nutrition_plan_product npp
    JOIN products p ON p.id = npp.product_id
    WHERE npp.user_id = ? AND date >= DATE('now', '-6 days')
""", (user_id,))
rows = cursor.fetchall()

nutrition_data_exists = bool(rows)

days_data = defaultdict(lambda: {"cal": 0, "prot": 0, "fat": 0, "carb": 0})
for date, cal, prot, fat, carb, grams in rows:
    factor = grams / 100
    days_data[date]["cal"] += cal * factor
    days_data[date]["prot"] += prot * factor
    days_data[date]["fat"] += fat * factor
    days_data[date]["carb"] += carb * factor

num_days = len(days_data)
total = {"cal": 0, "prot": 0, "fat": 0, "carb": 0}
for d in days_data.values():
    total["cal"] += d["cal"]
    total["prot"] += d["prot"]
    total["fat"] += d["fat"]
    total["carb"] += d["carb"]

avg_cal = round(total["cal"] / num_days) if num_days else 0
```

Рис. 24 – Формування зведеної статистики: підрахунок середніх значень

Функції show_detailed_nutrition() та show_detailed_workouts() наведено відповідно на рис. 25 та рис. 26. Перша формує графік на основі SQL-запиту до nutrition_plan_product, який ілюструє добове споживання калорій і макронутрієнтів за останні 7 днів. Друга формує графічну календарну візуалізацію тренувальної активності за останні 28 днів, відображаючи виконані та пропущені тренування у форматі кольорової сітки.

```
cursor.execute("""
    SELECT date, calories, protein, fat, carbs, grams_quantity
    FROM nutrition_plan_product npp
    JOIN products p ON p.id = npp.product_id
    WHERE npp.user_id = ? AND date >= DATE('now', '-6 days')
""", (user_id,))
rows = cursor.fetchall()
conn.close()

days = defaultdict(lambda: {"cal": 0, "prot": 0, "fat": 0, "carb": 0})
for date, cal, prot, fat, carb, grams in rows:
    factor = grams / 100
    days[date]["cal"] += cal * factor
    days[date]["prot"] += prot * factor
    days[date]["fat"] += fat * factor
    days[date]["carb"] += carb * factor
```

Рис. 25 – Деталізація харчування: статистика по днях

```
cursor.execute("""
    SELECT DISTINCT DATE(date) FROM workout_sessions
    WHERE user_id = ? AND DATE(date) <= DATE('now') AND DATE(date) >= DATE('now', '-29 days')
""", (user_id,))
rows = cursor.fetchall()
conn.close()

completed_days = {r[0] for r in rows}
today = datetime.today().date()
dates = [(today - timedelta(days=i)).isoformat() for i in range(30)]
```

Рис. 26 – Деталізація тренувань: календарна індикація активності

Таким чином, модуль статистики поєднує як агреговані, так і деталізовані дані. Завдяки цьому користувач отримує повну картину своєї активності та споживання, що є основою для персонального контролю й самокорекції у підтримці здорового способу життя.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Завершальним етапом життєвого циклу розробки програмного забезпечення є тестування, метою якого є перевірка відповідності функціоналу створеної системи поставленим вимогам. У межах даного проєкту тестування здійснювалось у форматі ручного функціонального контролю, орієнтованого на виявлення помилок у сценаріях взаємодії користувача з Telegram-ботом.

Особливістю застосування такого підходу є імітація дій кінцевого користувача без заглиблення у внутрішню реалізацію програмного коду (метод «чорної скриньки»). Це дозволяє перевірити логіку FSM-сценаріїв, коректність обробки введених даних, відповідність відповідей очікуваному формату, а також загальну стабільність роботи системи.

Тестування проводилося на базовому середовищі (Telegram Desktop, Python 3.10 [1], ОС Windows 10), з використанням реального токена Telegram-бота та локальної бази даних SQLite [4]. В якості результатів наведено покрокові інструкції, очікувану поведінку та фактичний результат виконання кожного ключового сценарію, доповнені скріншотами з інтерфейсу.

Усі тести виконувались вручну, без автоматизованих фреймворків, що є обґрунтованим підходом з огляду на інтерактивну природу Telegram-бота. Тестуванню підлягали модулі реєстрації, профілю, тренувань, харчування, ведення щоденника, перегляду статистики та навігації меню.

4.1.1 Реєстрація користувача. Сценарій реєстрації є першим кроком взаємодії користувача з Telegram-ботом і має вирішальне значення для подальшого використання системи. Його мета — отримати від користувача основні персональні дані, необхідні для формування індивідуальних

рекомендацій. Реалізація сценарію побудована на основі Finite State Machine (FSM) та активується через натискання інлайн-кнопки « Зареєструватися» після запуску бота.

Тестування цього модуля здійснювалося вручну з використанням Telegram Desktop. Був перевірений повний цикл взаємодії: запуск сценарію, поетапне введення коректних даних, реакція на помилки введення, успішне завершення процесу та відображення оновленого інтерфейсу.

Після запуску Telegram-бота користувач натискає кнопку « Почати», що відкриває головне меню з кнопкою « Зареєструватися». На рис. 26 показано зовнішній вигляд головного меню до реєстрації.

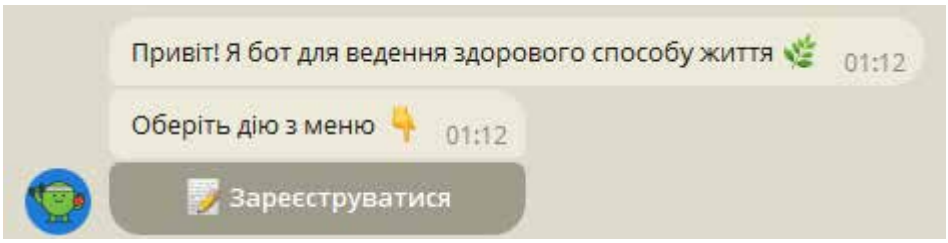

Рис. 26 – Головне меню Telegram-бота з кнопкою « Зареєструватися»

Після активації сценарію бот переходить у режим покрокового введення даних. Користувачу послідовно пропонується ввести ім’я (рис. 27), прізвище (рис. 28), вік (рис. 29), вагу (рис. 30) та зріст (рис. 31). На кожному кроці система перевіряє відповідність введених значень встановленим обмеженням: лексичним — для імені та прізвища, числовим — для віку, ваги та зросту.

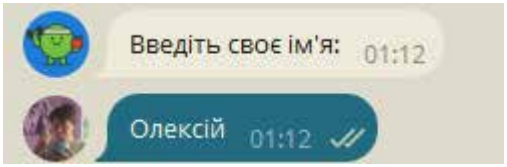

Рис. 27 – Запит Telegram-бота на введення імені користувача

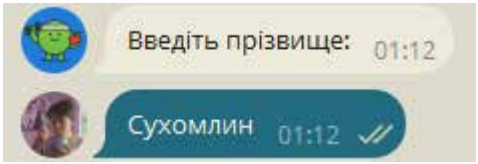

Рис. 28 – Запит Telegram-бота на введення прізвища користувача

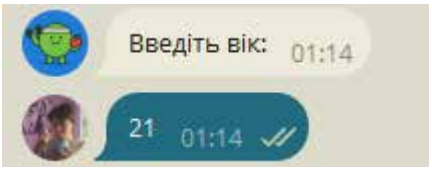

Рис. 29 – Введення віку користувачем

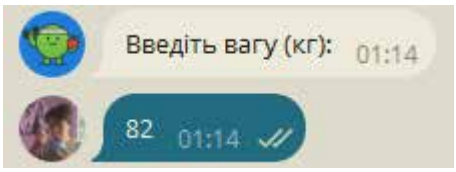

Рис. 30 – Введення ваги користувачем

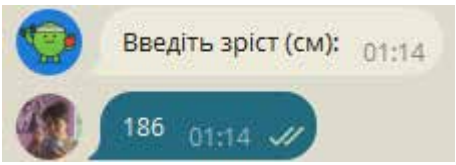

Рис. 31 – Введення зросту користувачем

Після введення всіх параметрів бот автоматично зберігає дані до бази та повідомляє про успішну реєстрацію. На рис. 32 показано підтвердження завершення сценарію. Одразу після цього головне меню оновлюється: з’являються кнопки доступу до інших функцій Telegram-бота (рис. 33).

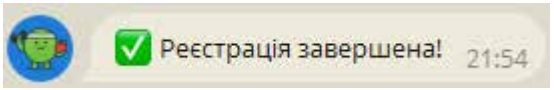

Рис. 32 – Повідомлення Telegram-бота про успішну реєстрацію користувача

Рис. 33 – Оновлене головне меню Telegram-бота після завершення реєстрації

Окремо було протестовано поведінку системи у разі введення недопустимих значень. Наприклад, при спробі ввести вік «200» бот вивів повідомлення про помилку та запропонував повторити введення (рис. 34). Аналогічна реакція спостерігалась при введенні недійсних значень у полі ваги та зросту.

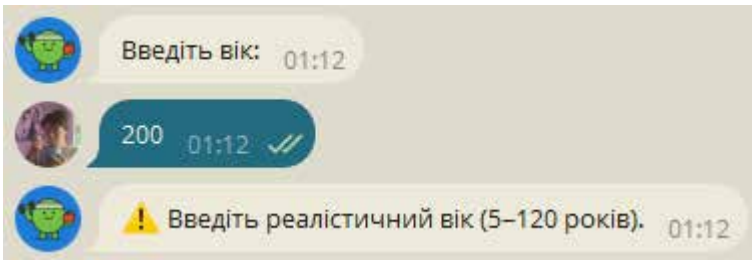

Рис. 34 – Повідомлення Telegram-бота про помилку при введенні недопустимого значення віку

У результаті тестування встановлено, що модуль реєстрації працює стабільно, забезпечує повноцінну обробку як правильних, так і помилкових введень, а також гарантує завершення сценарію лише за умови надання валідних даних. Повторна спроба реєстрації після успішного проходження блокується — користувач отримує повідомлення про те, що вже зареєстрований.

Таким чином, тестування підтвердило відповідність функціоналу реєстрації заданим вимогам. Сценарій повністю готовий до використання в продуктивному середовищі та слугує відправною точкою для активації інших модулів системи.

4.1.2 Профіль користувача. Модуль роботи з профілем реалізує можливість перегляду та редагування персональних даних, які використовуються системою для формування тренувальних та харчових рекомендацій. У межах тестування було перевірено дві основні дії: доступ до інформації профілю після реєстрації та оновлення даних за допомогою FSM-сценарію ProfileEditFSM.

Після завершення реєстрації користувач має доступ до функції «👤 Профіль», яка доступна через головне меню Telegram-бота. Натискання відповідної інлайн-кнопки ініціює виведення актуальної інформації з бази даних. На рис. 35 показано відображення профілю користувача одразу після реєстрації.

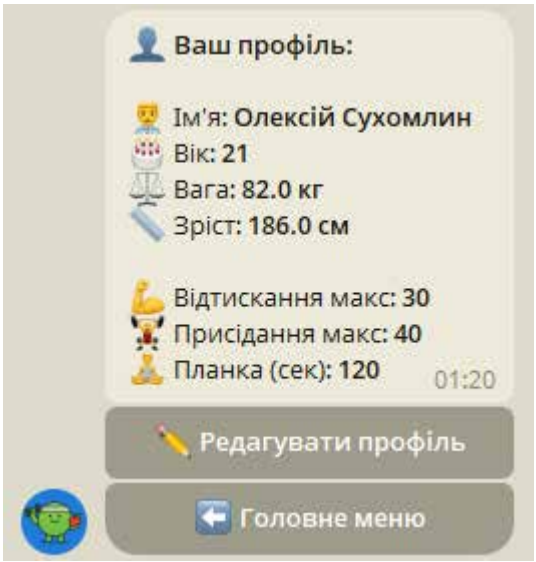

Рис. 35 – Відображення профілю користувача у Telegram-боті

Для редагування даних необхідно натиснути кнопку «✏️ Редагувати профіль», після чого запускається багатокроковий FSM-сценарій. Користувач по черзі вводить нові значення для кожного з параметрів. Кожен крок супроводжується перевіркою коректності введених даних: лексичні значення (ім’я, прізвище) обмежені символами та довжиною, числові параметри — діапазоном допустимих значень.

На рис. 36 зображено запит Telegram-бота на введення нового імені користувача. Після підтвердження значення сценарій переходить до прізвища (рис. 37), віку (рис. 38), ваги та зросту. Введення завершується блоком максимальних фізичних показників, необхідних для адаптації тренувального плану.

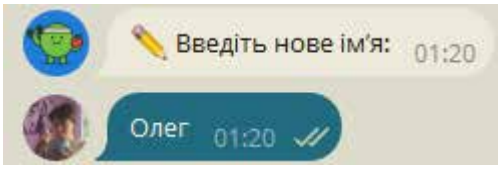

Рис. 36 – Запит Telegram-бота на введення нового імені користувача

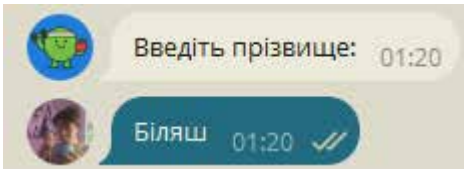

Рис. 37 – Запит Telegram-бота на введення нового прізвища користувача

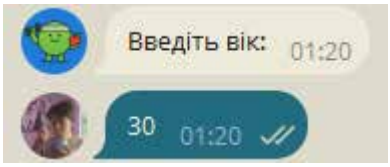

Рис. 38 – Введення нового віку користувача

Після завершення введення всіх параметрів Telegram-бот оновлює відповідні значення в базі даних і надсилає повідомлення про успішне збереження. На (рис. 39) показано приклад фінального повідомлення після завершення редагування профілю.

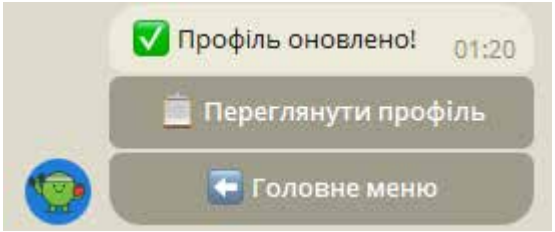

Рис. 39 – Повідомлення Telegram-бота про успішне оновлення профілю

Після цього користувач має змогу одразу перейти до оновленого профілю або повернутись у головне меню. При повторному відкритті профілю система відображає вже оновлені дані. Це підтверджує успішне збереження інформації в таблиці users та коректне функціонування всього FSM-сценарію. Приклад оновленої картки профілю наведено на рис. 40.

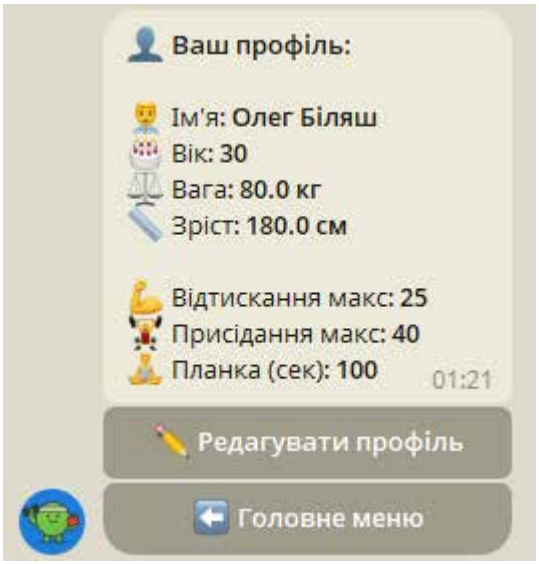

Рис. 40 – Оновлений профіль користувача після редагування

У межах тестування також було перевірено реакцію системи на помилкове введення (наприклад, використання цифр у полі імені або занадто великого значення віку). У всіх випадках бот коректно виводив повідомлення про помилку та повторно запитував дані, що підтверджує стійкість сценарію до помилок введення.

Таким чином, тестування модуля профілю підтвердило його працездатність, відповідність вимогам та надійність FSM-сценарію. Користувач має змогу в будь-який момент оновити свої персональні дані, що забезпечує актуальність рекомендаційної системи під час подальшого використання.

4.1.3 Тренування. Модуль тренування є ключовим елементом системи, що відповідає за генерацію індивідуального плану фізичних вправ, адаптованого до рівня підготовки користувача. У межах тестування було перевірено роботу

сценарію отримання та оновлення тренувального плану, а також фіксацію факту завершення тренування.

Після реєстрації та заповнення профілю користувач отримує доступ до функції «🏋 Тренування». Натискання відповідної кнопки з головного меню ініціює перевірку, чи вже існує тренувальний план на поточну дату. Якщо план відсутній, бот пропонує обрати рівень інтенсивності. На рис. 41 показано виведення відповідного повідомлення Telegram-бота.

Рис. 41 – Запит Telegram-бота на вибір рівня інтенсивності тренування

Після натискання на один із варіантів (наприклад, «🟡 Середній») система виконує генерацію плану на основі збережених максимальних фізичних показників користувача. Результатом є повідомлення з переліком вправ, кількістю підходів і повторень, а також кнопками дій. Приклад сформованого плану наведено на рис. 42.

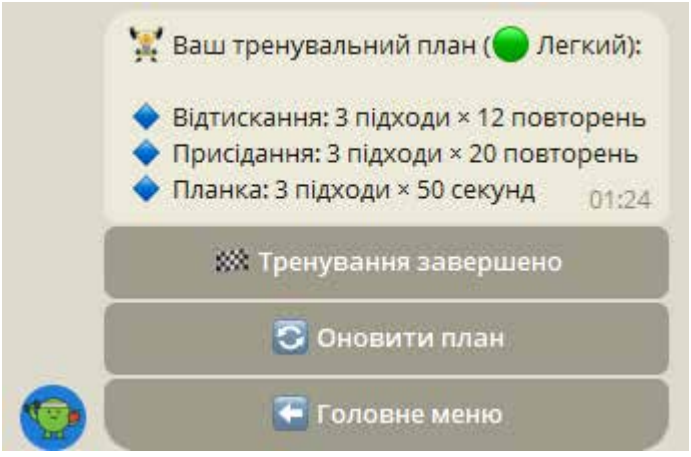

Рис. 42 – Згенерований індивідуальний тренувальний план

Користувач має змогу натиснути кнопку «🏁 Тренування завершено», що фіксує факт виконання в таблиці workout_sessions. Telegram-бот повідомляє про успішну реєстрацію активності. Цей запис використовується у модулі статистики для обліку кількості виконаних тренувань. Приклад відповідного повідомлення наведено на рис. 43.

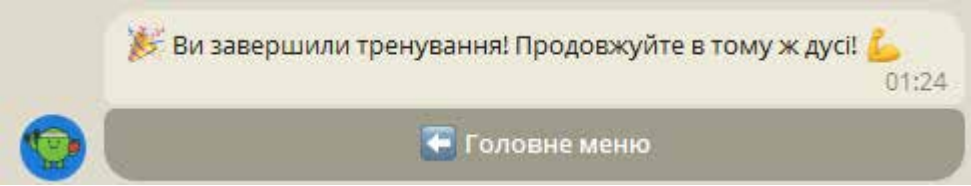

Рис. 43 – Повідомлення Telegram-бота про завершене тренування

Окрім фіксації, користувач також може натиснути кнопку «🔄 Оновити план» для повторного формування тренування на поточний день. У такому разі попередній запис видаляється, і бот знову пропонує вибір рівня інтенсивності. Після повторного вибору рівня (наприклад, «🔴 Важкий») генерується новий план, який також відображається у форматі, аналогічному початковому сценарію. Приклад оновленого плану подано на рис. 44.

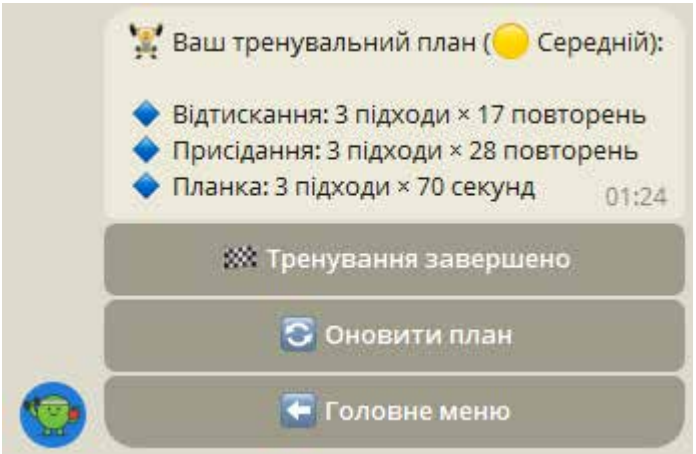

Рис. 44 – Новий тренувальний план після повторної генерації

У межах тестування окремо перевірявся сценарій повторного натискання кнопки «🏁 Тренування завершено» у той самий день. У такому випадку бот виводить повідомлення про те, що тренування вже було зареєстровано раніше,

без дублювання запису у базі. Приклад відповідної реакції системи наведено на рис. 45.

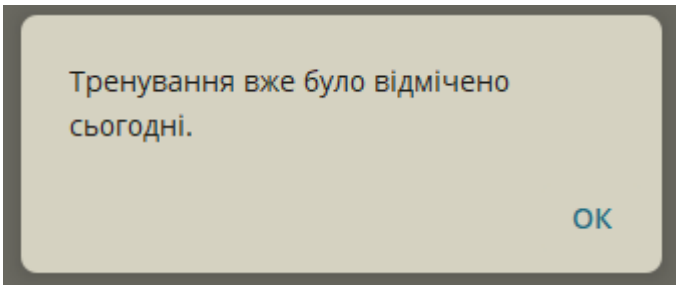

Рис. 45 – Повідомлення Telegram-бота про вже зареєстроване тренування

Тестування підтвердило, що модуль тренувань коректно реагує на стан системи (наявність або відсутність плану), забезпечує адаптивність до обраного рівня складності, стійкий до повторних викликів дій, та зберігає дані в базі згідно з логікою проєкту. Інтерфейс залишається зручним і послідовним для кінцевого користувача.

4.1.4 Харчовий план. Модуль харчування дозволяє користувачу отримати персоналізований денний раціон на основі віку, ваги та зросту. У межах тестування перевірялась коректність запуску сценарію генерації плану, оновлення рекомендацій та відповідність виводу встановленому формату.

Для активації сценарію користувач натискає кнопку «🥗 Харчування» з головного меню Telegram-бота. Система виконує генерацію нового плану за допомогою функції generate_nutrition_plan. Сформоване повідомлення містить загальну калорійність добового раціону, рекомендовану кількість білків, жирів і вуглеводів, а також список продуктів із розрахованою масою, калорійністю та макронутрієнтним складом. Усі дані виводяться у форматі структурованого блоку з графічним маркуванням. Приклад повного повідомлення харчового плану наведено на рис. 46.

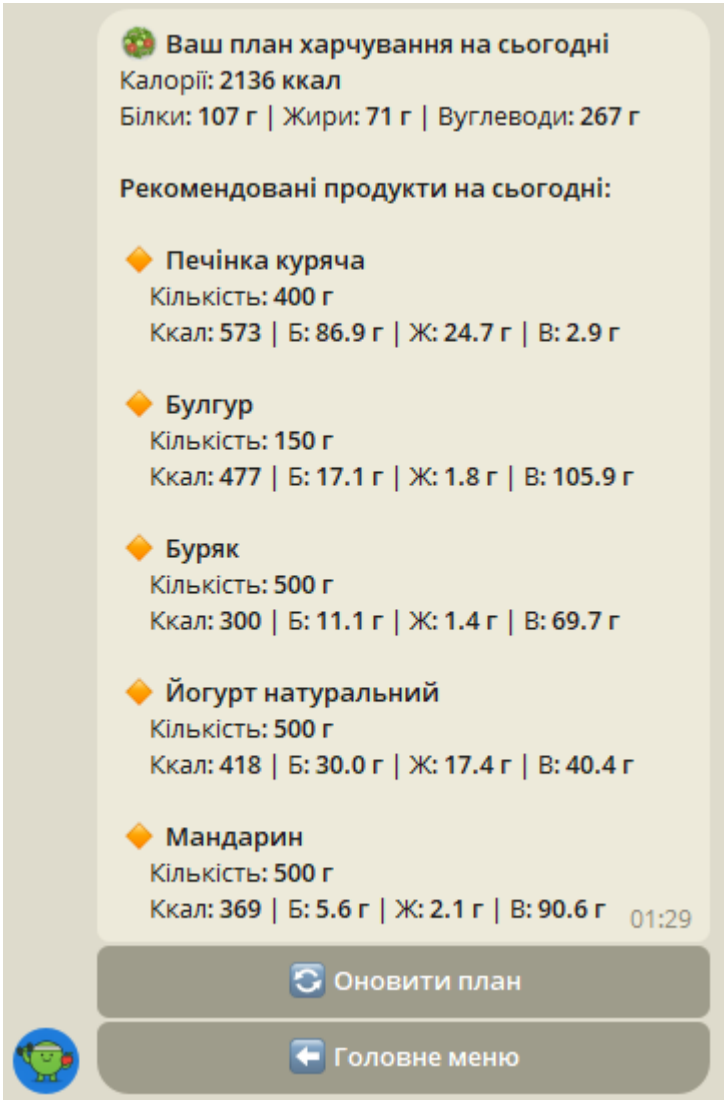

Рис. 46 – Структурований харчовий план з рекомендованими продуктами

Після виводу плану користувач отримує кнопки для подальших дій: « Оновити план» або « Головне меню». У процесі тестування було перевірено функціональність кнопки оновлення. Натискання « Оновити план» призводить до видалення попереднього запису та генерації нового раціону, що підтверджується відповідним повідомленням (рис. 47).

Рис. 47 – Повідомлення про оновлення харчового плану за запитом користувача

Порівняння двох окремих запусків сценарію продемонструвало варіативність вибору продуктів, що відповідає реалізованій логіці генерації на основі шаблонів. Результати кожного запуску містять індивідуальні набори продуктів із відповідно адаптованими порціями.

У ході тестування також було перевірено поведінку системи у разі запуску сценарію без попередньої реєстрації. У такому випадку Telegram-бот коректно виводить повідомлення про необхідність зареєструватися та не виконує жодних дій. Цей запобіжник виключає можливість генерації плану для анонімного користувача.

Таким чином, модуль харчування успішно пройшов функціональне тестування. Він демонструє стабільну роботу сценарію генерації, забезпечує

адаптивність до персональних параметрів та варіативність рекомендаційного виводу, що дозволяє формувати збалансований денний раціон користувача.

4.1.5 Щоденник харчування. Модуль щоденника харчування забезпечує облік фактично спожитих продуктів за поточну дату, а також перегляд історії за останній тиждень. Тестування цього сценарію охоплювало усі основні дії: додавання продукту вручну, fuzzy-пошук [3], обрахунок спожитих калорій, видалення останнього запису та перегляд історії.

Сценарій активується після натискання кнопки «📒 Щоденник» у головному меню Telegram-бота. При першому запуску бот формує підсумкове повідомлення з поточним балансом калорій, білків, жирів і вуглеводів, а також відображає кнопки для навігації. Приклад стартового повідомлення наведено на рис. 48.

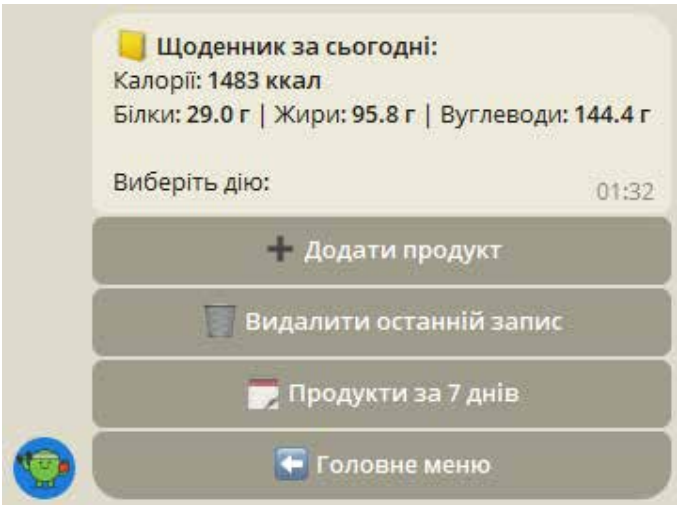

Рис. 48 – Початкове повідомлення Telegram-бота у модулі «Щоденник»

Після натискання кнопки «➕ Додати продукт» запускається FSM-сценарій з двох кроків. Спочатку бот запитує назву продукту. При введенні, наприклад, «гречка», бот виконує fuzzy-пошук [3] по базі даних і пропонує найбільш релевантний варіант. Припустимо, в базі є «Гречка», і результат схожості перевищує встановлений поріг. На рис. 49 показано відповідну взаємодію Telegram-бота з користувачем.

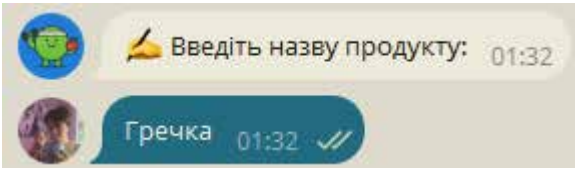

Рис. 49 – Пошук продукту в базі за частковим збігом назви

Після підтвердження знайденого продукту бот запитує кількість грамів. Введення значення, наприклад, «150», завершує сценарій. Telegram-бот додає запис до таблиці nutrition_plan_product із поточною датою, після чого оновлює повідомлення зі зведенням. На рис. 50 наведено результат успішного додавання продукту.

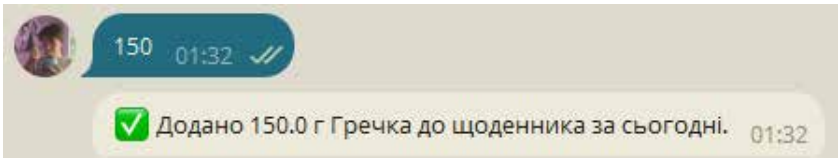

Рис. 50 – Підтвердження Telegram-ботом доданого продукту до щоденника

У межах тестування було також перевірено обробку помилок. Наприклад, при введенні неіснуючого продукту або результату з низьким рівнем схожості бот виводить повідомлення про помилку й просить повторити введення (рис. 51).

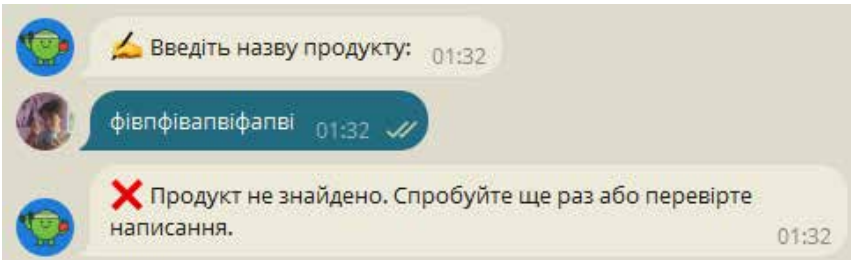

Рис. 51 – Повідомлення Telegram-бота при невдалому fuzzy-пошуку продукту

Додатково протестовано функцію «🗑 Видалити останній запис». При її активації бот виконує SQL-запит на вибір останнього доданого запису за поточну дату, після чого видаляє його з бази. У результаті виводиться повідомлення з деталями про видалений запис. На рис. 52 показано приклад реалізації цього сценарію.

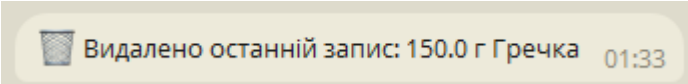

Рис. 52 – Повідомлення Telegram-бота про успішне видалення останнього запису

Нарешті, натискання кнопки «🗓 Продукти за 7 днів» викликає функцію, яка групує всі записи користувача за останні сім днів і виводить їх у вигляді структурованого списку із зазначенням дати та назви кожного продукту. Приклад відображення тижневої історії наведено на рис. 53.

Рис. 53 – Перегляд спожитих продуктів за останні 7 днів у Telegram-боті

Результати тестування підтвердили працездатність FSM-сценарію, правильну обробку пошукових запитів, захист від помилкових введень, а також стабільну роботу механізмів запису й видалення даних з бази. Всі ключові дії супроводжуються чіткими виводами й візуальним підтвердженням дій користувача.

Таким чином, модуль «Щоденник» забезпечує ефективний облік харчування, а реалізована логіка дозволяє користувачеві гнучко керувати введеними записами у зручному форматі Telegram-інтерфейсу.

4.1.6 Статистика. Модуль статистики забезпечує користувачу доступ до агрегованих і деталізованих показників активності та харчування. Він дозволяє відслідковувати прогрес у графічному вигляді, що формує основу для самоконтролю та мотивації. У межах тестування перевірялись основні сценарії: перегляд зведених даних, вивід графічної деталізації, коректність обчислень і відображення повідомлень у випадку відсутності записів.

Для запуску модуля користувач натискає кнопку «📊 Статистика» у головному меню. Якщо в системі збережені записи про виконані тренування або додані продукти до щоденника за останній період, бот формує підсумкове повідомлення зі зведенням. У ньому вказано середнє споживання калорій, білків, жирів і вуглеводів за останні 7 днів, а також кількість тренувальних днів за останні 30 днів. Приклад зведеної статистики показано на рис. 54.

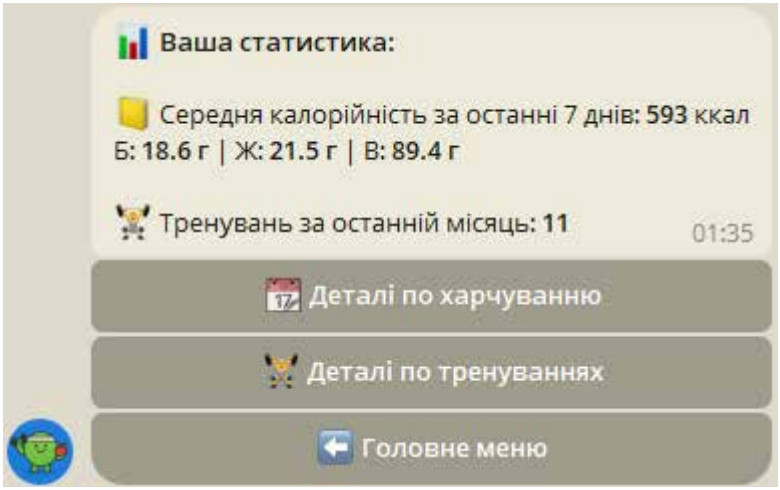

Рис. 54 – Зведене повідомлення Telegram-бота зі статистикою за тиждень

З цього повідомлення користувач може перейти до перегляду деталізованої інформації за допомогою кнопок «📅 Деталі по харчуванню» та «🏋 Деталі по тренуваннях». У першому випадку бот відображає графік щоденного споживання калорій і макронутрієнтів (білків, жирів, вуглеводів) за останні 7

днів. Візуалізація дозволяє швидко оцінити тенденції. На рис. 55 зображено приклад такої графічної деталізації.

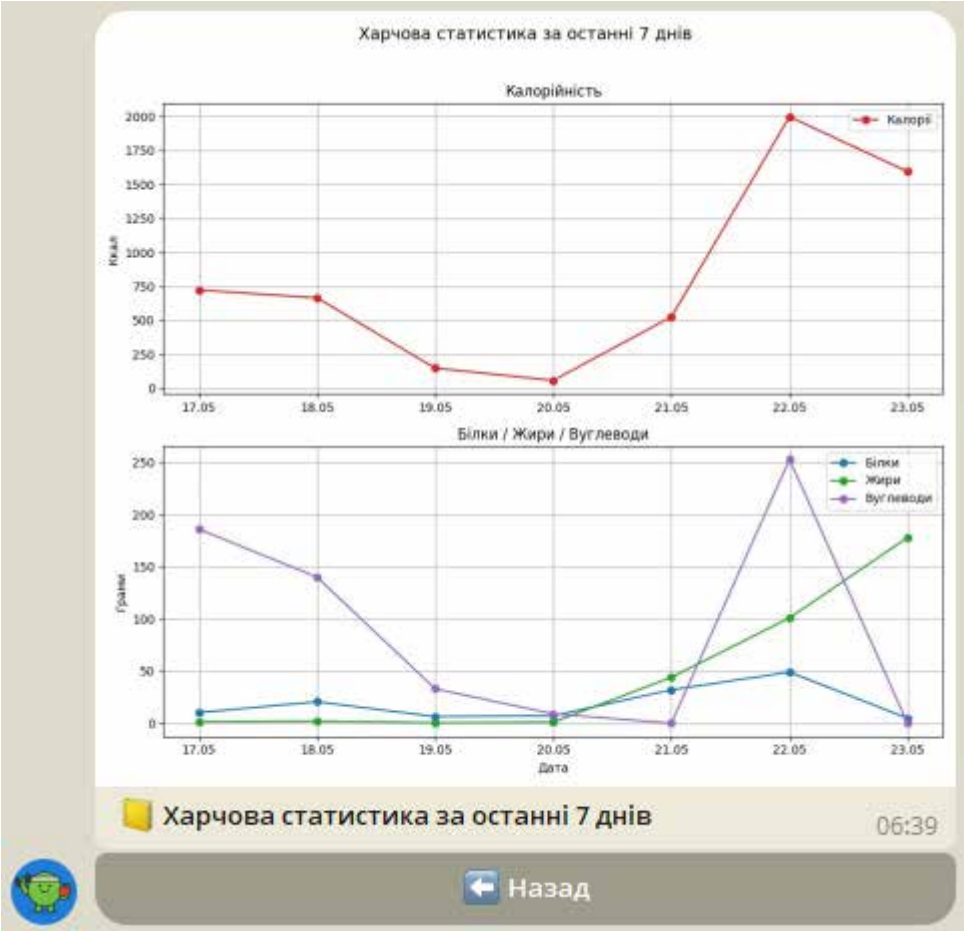

Рис. 55 – Деталізована статистика по спожитих продуктах за останні 7 днів

При виборі опції «🏋️ Деталі по тренуваннях» Telegram-бот відображає календарну візуалізацію за останні 28 днів із кольоровим маркуванням днів, коли тренування було виконано або пропущено. Зелений колір означає виконання, червоний — пропуск. Приклад календарної мапи наведено на рис. 56.

Рис. 56 – Деталізована активність користувача у розрізі тренувальних днів за останній місяць

Окремо було перевірено поведінку системи у випадку, коли в базі відсутні дані за останні дні. У такій ситуації Telegram-бот виводить повідомлення про відсутність записів та пропонує повернутися до головного меню. Цей сценарій був протестований на новому акаунті. Приклад відповідного повідомлення показано на рис. 57.

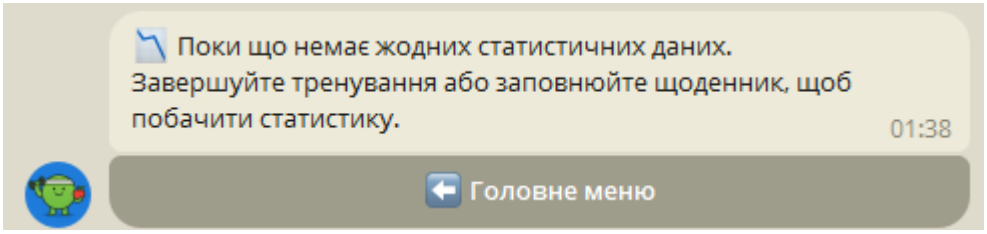

Рис. 57 – Повідомлення Telegram-бота про відсутність статистичних даних

Тестування підтвердило правильність розрахунків, стійкість до відсутності даних, а також зручність переходу між зведеним і деталізованим поданням. Формат виводу є зрозумілим для користувача, а логіка побудови звітів відповідає очікуванням.

Таким чином, модуль статистики виконує роль інформативного інструменту для самоспостереження й оцінки прогресу. Його реалізація є завершеною, стабільною та повністю відповідає вимогам до інтерактивного чат-бота рекомендаційного типу.

4.1.7 Головне меню та навігація. Головне меню Telegram-бота виконує функцію централізованого навігаційного інтерфейсу, що забезпечує доступ до всіх ключових модулів системи. Тестування включало перевірку стабільності переходів між модулями, повернення до меню, а також реакції на повторні запити.

Після успішної реєстрації користувачу стає доступне інлайн-меню з кнопками: «👤 Профіль», «🏋️ Тренування», «🥗 Харчування», «📒 Щоденник», «📊 Статистика». Кожен з модулів запускається коректно, а натискання кнопки «⬅️ Головне меню» в будь-якому сценарії повертає користувача до базового вікна.

Тестування показало, що меню динамічно адаптується: при відсутності реєстрації кнопки блокуються, а при оновленні статусу – оновлюється контекст. Навігація є інтуїтивно зрозумілою, без збоїв чи втрати контексту FSM-сценаріїв.

Таким чином, інтерфейс навігації працює стабільно, забезпечує повний доступ до функціональності бота та підтримує зручність використання на всіх етапах взаємодії.

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної роботи Telegram-бота рекомендаційної системи необхідно враховувати вимоги до середовища функціонування як з боку кінцевого користувача, так і на рівні реалізації серверної логіки. На рис. 59 подано діаграму розгортання системи, яка відображає взаємозв’язки між усіма

ключовими компонентами: користувачем, Telegram-сервером і локальним застосунком бота.

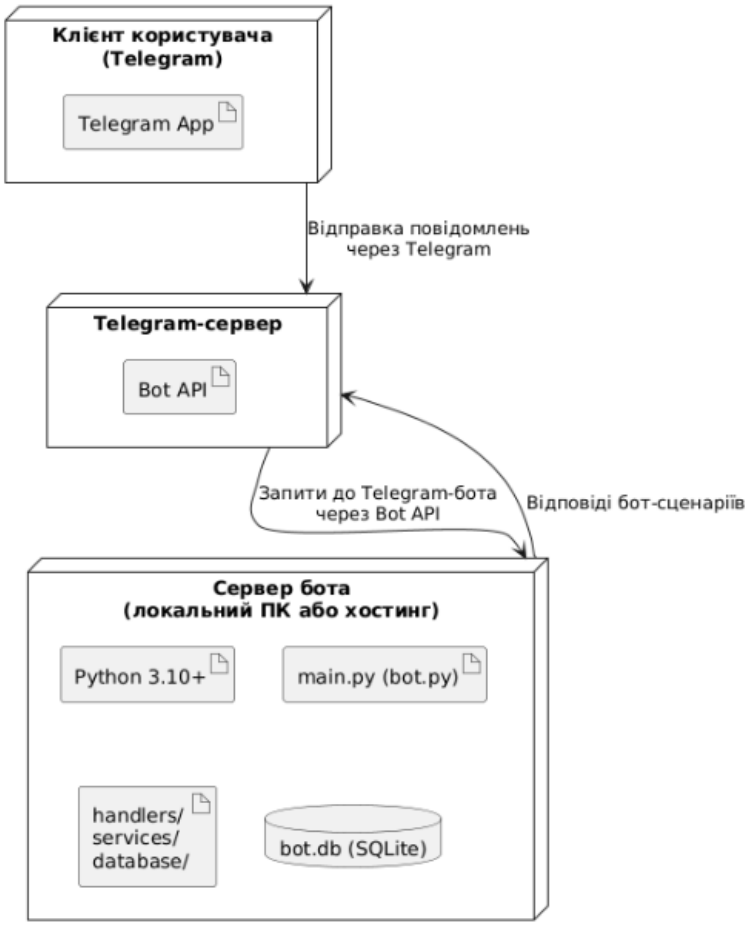

Рис. 59 – Діаграма розгортання Telegram-бота рекомендаційної системи

Вимоги до користувача:

- Telegram-бот не потребує встановлення окремого програмного забезпечення, а всі взаємодії відбуваються у межах стандартного Telegram-клієнта.
- Пристрій: смартфон або персональний комп’ютер із встановленим Telegram (мобільна або десктопна версія)
- Інтернет-з’єднання: обов’язкове
- Додаткове ПЗ: не потрібне

Вимоги до серверної частини:

- Telegram-бот є Python-застосунком, що може бути запущений локально або на сервері з підтримкою відповідного середовища виконання. Уся логіка обробки запитів, генерації планів, збереження історії та взаємодії з базою даних реалізується в межах одного процесу.
- Операційна система: Windows 10/11, Ubuntu 20.04+, macOS 11+
- Процесор: 2 ядра і більше
- Оперативна пам’ять: від 2 ГБ
- Вільне місце на диску: щонайменше 100 МБ
- Інтерпретатор мови: Python версії 3.10 [1] або вище
- Зовнішні бібліотеки:
- aiogram [2] – для асинхронної роботи з Telegram API
- python-dotenv [6] – для обробки середовищних змінних
- rapidfuzz [3] – для реалізації нечіткого пошуку продуктів
- matplotlib [16] – для побудови графічної візуалізації статистики
- Система керування базами даних: SQLite [4] (вбудована у Python; база bot.db)

Telegram-бот функціонує як автономна консольна програма без графічного інтерфейсу. Усі сценарії взаємодії реалізовано через Telegram-клієнт, що робить систему зручною для використання без залежності від операційної системи користувача або складних установчих процедур.

4.3 Склад інсталяційного пакету

Для забезпечення можливості інсталяції та запуску Telegram-бота на іншому пристрої або в середовищі розгортання сформовано інсталяційний пакет,

що включає всі необхідні компоненти системи. Пакет є самодостатнім, не потребує складної конфігурації або зовнішніх залежностей, окрім Python-інтерпретатора та базових бібліотек.

До складу інсталяційного пакету входять:

Основні файли запуску:

- bot.py — головний модуль запуску бота.
- config.py — файл конфігурації, що зчитує токен із середовищного файлу.
- .env.example — шаблон файлу з токеном бота (реальний .env не передається).

Каталоги з логікою системи:

- handlers/ — обробники FSM-сценаріїв:
    - register.py, profile.py, workout.py, nutrition.py, diary.py, stats.py, max_input.py
- services/ — бізнес-логіка генерації планів:
    - planner.py
- database/ — робота з базою даних:
    - create_tables.sql — SQL-інструкції для створення таблиць
    - db.py — функції ініціалізації та підключення
- utils/ — допоміжні модулі:
    - keyboards.py — інтерфейс кнопок
    - states.py — опис FSM-станів

Бібліотеки:

- requirements.txt — перелік зовнішніх бібліотек для встановлення через pip:
    - aiogram [2];
    - python-dotenv [6];
    - rapidfuzz [3;]

- matplotlib [16].

База даних:

- bot.db — SQLite-база даних, створюється автоматично при першому запуску (не обов’язково включати в дистрибутив).

Інсталяційний пакет зберігається у вигляді ZIP-архіву або каталогу, готового до розпакування на робочій машині. Усі шляхи відносні, структура проєкту дотримується відповідно до логіки Python-пакетів. Після налаштування токена у .env та встановлення залежностей (pip install -r requirements.txt) бот готовий до запуску без додаткової конфігурації.

ВИСНОВОК

У ході виконання бакалаврської кваліфікаційної роботи на тему «Рекомендаційна система підтримки здорового способу життя» було реалізовано повноцінний програмний продукт у вигляді Telegram-бота, який забезпечує надання персоналізованих рекомендацій із фізичної активності та харчування, фіксацію фактичного споживання продуктів, ведення статистики та взаємодію з користувачем у форматі зручного чат-інтерфейсу.

Була проведена системна декомпозиція предметної області, побудовано логічну модель даних, обрано оптимальну СУБД SQLite [4] та реалізовано багатомодульну архітектуру програмного забезпечення на основі Python та бібліотеки Aiogram [2].

Досягнуті результати повністю відповідають завданню, сформульованому у технічному завданні:

- реалізовано автоматичну генерацію тренувальних і харчових планів;
- реалізовано облік персональних параметрів користувача та їх редагування;
- впроваджено модуль харчового щоденника із ручним введенням і аналізом продуктів;
- створено модуль статистики з графічною візуалізацією динаміки фізичної активності та харчування за останні дні;
- забезпечено повну підтримку FSM-сценаріїв і зручного інтерфейсу взаємодії через Telegram.

Запропоноване технічне рішення має такі особливості:

- поєднання автоматичних та ручних компонентів у єдиному інтерфейсі Telegram;

- використання FSM для обробки діалогових сценаріїв;
- застосування fuzzy-пошуку [3] для підвищення зручності введення продуктів;
- модульність і кросплатформеність системи;
- використання лише вбудованих або відкритих інструментів без комерційних залежностей.

У порівнянні з існуючими аналогами (MyFitnessPal [17], Yazio [18], Nike Training Club [19]), реалізоване рішення:

- не потребує встановлення окремого додатку;
- поєднує одночасно тренування і харчування;
- має простий інтерфейс, доступний навіть для технічно необізнаного користувача;
- не нав’язує платних функцій і працює автономно.

Результати роботи можуть бути використані:

- для подальшого розширення функціональності в напрямі фітнес-моніторингу;
- як основа для мобільного застосунку;
- у навчальному процесі як приклад реалізації FSM-ботів на Python;
- як MVP (minimum viable product) для стартапів у сфері e-health.

До рекомендацій щодо подальшої роботи належать:

- реалізація авторизації та синхронізації з обліковими записами;
- інтеграція з зовнішніми джерелами харчових продуктів (наприклад, API калорійності);

- додавання нагадувань і push-повідомлень через Telegram;
- підтримка багатомовності інтерфейсу;
- адаптація для роботи з декількома користувачами на одному сервері.

Загалом, запропонована система є функціонально завершеною, технічно обґрунтованою та придатною до практичного застосування. Студентом було самостійно запропоновано структуру FSM-сценаріїв, логіку fuzzy-обробки [3] введення продуктів, а також реалізовано повний цикл генерації, збереження, перегляду та редагування даних через Telegram-інтерфейс, що свідчить про високий рівень володіння інструментарієм та глибоке розуміння предметної області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation. (2024). Python 3.10 documentation. https://docs.python.org/3.10/ [Технічна документація]
2. Aiogram Developers. (2024). Aiogram Telegram bot framework. https://docs.aiogram.dev/en/latest/ [Технічна документація]
3. Bachmann, M. (2024). RapidFuzz: Documentation for fuzzy matching. https://maxbachmann.github.io/RapidFuzz/ [Технічна документація]
4. SQLite Consortium. (2024). Official documentation. https://www.sqlite.org/docs.html [Технічна документація]
5. Telegram. (2024). Bot API reference. https://core.telegram.org/bots/api [Технічна документація]
6. Saurabh, K. (2023). python-dotenv documentation. https://saurabh-kumar.com/python-dotenv/ [Технічна документація]
7. DB Browser for SQLite. (2024). Graphical tool for SQLite databases. https://sqlitebrowser.org/ [Комп'ютерне програмне забезпечення]
8. JetBrains. (2024). PyCharm IDE. https://www.jetbrains.com/pycharm/ [Комп'ютерне програмне забезпечення]
9. GitHub. (2024). Aiogram examples. https://github.com/aiogram/aiogram
10. Mifflin, M. D., St Jeor, S. T., Hill, L. A., Scott, B. J., Daugherty, S. A., & Koh, Y. O. (1990). A new predictive equation for resting energy expenditure in healthy individuals. The American Journal of Clinical Nutrition, 51(2), 241–247.
11. U.S. Department of Agriculture. (2024). FoodData Central. https://fdc.nal.usda.gov/

12. National Health Service. (2023). How to eat a balanced diet. https://www.nhs.uk/live-well/eat-well/

13. World Health Organization. (2022). Physical activity factsheet. https://www.who.int/news-room/fact-sheets/detail/physical-activity

14. Centers for Disease Control and Prevention. (2023). How much physical activity do adults need? https://www.cdc.gov/physicalactivity/basics/adults/index.htm

15. Mayo Clinic. (2024). Healthy lifestyle: Nutrition and fitness. https://www.mayoclinic.org/healthy-lifestyle

16. Matplotlib Developers. (2024). Matplotlib: Visualization with Python. https://matplotlib.org/stable/contents.html [Технічна документація]

17. MyFitnessPal. (2024). Official website. https://www.myfitnesspal.com/

18. Yazio. (2024). Yazio nutrition & calorie tracker. https://www.yazio.com/

19. Nike, Inc. (2024). Nike Training Club: Workout app. https://www.nike.com/ntc-app

20. Zhang, Y., Wang, F., Liu, X., & Yang, J. (2020). Personalized nutrition recommendation system using hybrid approach. Journal of Healthcare Engineering, 2020, Article ID 8856643. https://doi.org/10.1155/2020/8856643

21. Sokolova, M., Mahmudova, I., & Ivchenko, A. (2021). Recommendation systems in e-health: Current trends. IEEE Access, 9, 75320–75331. https://doi.org/10.1109/ACCESS.2021.3081098

22. Healthline. (2024). Best apps for healthy eating and fitness in 2024. https://www.healthline.com/health/fitness-exercise/top-fitness-apps

ДОДАТОК А

Лістинг SQL-коду створення таблиць бази даних create_tables.sql

```
-- Таблиця користувачів
CREATE TABLE IF NOT EXISTS users (
   id INTEGER PRIMARY KEY,
   first_name TEXT,
   last_name TEXT,
   age INTEGER,
   weight REAL,
   height REAL,
   pushups_max INTEGER,
   squats_max INTEGER,
   plank_max INTEGER
);

-- Таблиця вправ
CREATE TABLE IF NOT EXISTS exercises (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   user_id INTEGER,
   name TEXT NOT NULL,
   description TEXT,
   type TEXT DEFAULT 'bodyweight',
   max_reps INTEGER,
   FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Таблиця продуктів
CREATE TABLE IF NOT EXISTS products (
```

```
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    calories REAL,
    protein REAL DEFAULT 0,
    fat REAL DEFAULT 0,
    carbs REAL DEFAULT 0
);

-- Таблиця планів тренувань
CREATE TABLE IF NOT EXISTS workout_plans (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    date TEXT,
    intensity TEXT DEFAULT 'medium',
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Таблиця планів харчування
CREATE TABLE IF NOT EXISTS nutrition_plans (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    date TEXT,
    total_calories REAL,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Таблиця зв’язку: План тренування — Вправи
CREATE TABLE IF NOT EXISTS workout_plan_exercise (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    plan_id INTEGER,
```

```
    exercise_id INTEGER,
    sets INTEGER,
    reps INTEGER,
    FOREIGN KEY (plan_id) REFERENCES workout_plans(id),
    FOREIGN KEY (exercise_id) REFERENCES exercises(id)
);

-- Таблиця щоденника
CREATE TABLE IF NOT EXISTS nutrition_plan_product (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    product_id INTEGER,
    grams_quantity REAL,
    date TEXT DEFAULT (DATE('now')),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);

-- Таблиця факту завершення тренувань
CREATE TABLE IF NOT EXISTS workout_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    date TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

ДОДАТОК Б

Лістинг коду головного модуля запуску Telegram-бота bot.py

```
from aiogram import executor, types
from loader import dp, bot
from database.db import init_db, get_connection
from utils.keyboards import main_menu, get_inline_menu, return_menu
from aiogram.utils.exceptions import MessageCantBeDeleted

from handlers.workout import send_workout_plan
from handlers.nutrition import send_nutrition_plan
from handlers.stats import show_stats
from handlers.profile import show_profile
from handlers.register import start_registration
from handlers.workout import send_workout_plan, workout_level_keyboard
from handlers.diary import diary_command

import handlers.register
import handlers.workout
import handlers.nutrition
import handlers.diary
import handlers.stats

from aiogram.types import CallbackQuery

@dp.message_handler(commands=['start'])
@dp.message_handler(lambda message: message.text == "▶️ Почати")

async def cmd_start(message: types.Message):
```

```
    user_id = message.from_user.id

    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
    is_registered = cursor.fetchone() is not None
    conn.close()

    await message.answer(
        "Привіт! Я бот для ведення здорового способу життя 🌿",
        reply_markup=main_menu
    )
    await message.answer("Оберіть дію з меню 👇",
reply_markup=get_inline_menu(is_registered))


@dp.callback_query_handler(lambda c: c.data == "register")
async def cb_register(call: CallbackQuery):
    await call.message.delete()
    await start_registration(call.from_user.id)


@dp.callback_query_handler(lambda c: c.data == "profile")
async def cb_profile(call: CallbackQuery):
    await call.message.delete()
    await show_profile(call.from_user.id)


@dp.callback_query_handler(lambda c: c.data == "workout")
async def cb_workout(call: CallbackQuery):
    await call.message.delete()
    user_id = call.from_user.id
```

```
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id FROM workout_plans WHERE user_id = ? AND
date = DATE('now')", (user_id,))
    plan_exists = cursor.fetchone()
    conn.close()

    if plan_exists:
        await send_workout_plan(user_id)
    else:
        await bot.send_message(user_id, "Оберіть рівень інтенсивності:",
reply_markup=workout_level_keyboard())

@dp.callback_query_handler(lambda c: c.data == "nutrition")
async def cb_nutrition(call: CallbackQuery):
    await call.message.delete()
    await send_nutrition_plan(call.from_user.id)

@dp.callback_query_handler(lambda c: c.data == "diary")
async def cb_diary(call: CallbackQuery):
    try:
        await call.message.delete()
    except MessageCantBeDeleted:
        pass

    await diary_command(call)

@dp.callback_query_handler(lambda c: c.data == "stats")
async def cb_stats(call: CallbackQuery):
    await call.message.delete()
    await show_stats(call.from_user.id)
```

```
@dp.callback_query_handler(lambda c: c.data == "menu")
async def cb_menu(call: CallbackQuery):
    user_id = call.from_user.id
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
    is_registered = cursor.fetchone() is not None
    conn.close()

    await call.message.delete()
    await bot.send_message(call.from_user.id, "🔄 Меню оновлено:",
reply_markup=get_inline_menu(is_registered))

if __name__ == "__main__":
    init_db()
    executor.start_polling(dp, skip_updates=True)
```