Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Розробка інтерактивного застосунку для комплексного моніторингу та управління здоровим способом життя з використанням методів аналізу даних

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи
Канд. Фіз.-мат. наук, доцент
Афонін А. О. _____
(підпис)

«____» _____ 2025 р.

Виконав студент 3-го року навчання
на спеціальності «Комп'ютерні науки»
Молчанов О. К.

«____» _____ 2025 р.

Київ – 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики, доцент, кандидат наук _____ С. С. Гороховський
(підпис)
«_____» _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту 3-го курсу, факультету інформатики

Молчанову Олексію Костянтиновичу

Тема: «Розробка інтерактивного застосунку для комплексного моніторингу та управління здоровим способом життя з використанням методів аналізу даних»

Зміст ТЧ до курсової роботи:

Зміст
Анотація
Вступ
Розділ 1. Аналіз предметної області
Розділ 2. Теоретичні основи
Розділ 3. Програмна реалізація вебзастосунку
Висновки
Список літератури
Додатки (за потреби)

Дата видачі «_____» _____ 2025 р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Розробка інтерактивного застосунку для комплексного моніторингу та управління здоровим способом життя з використанням методів аналізу даних

Календарний план виконання роботи:

№ п/п	Назва етапу курсової робити	Термін виконання етапу	Примітка
1	Вибір та затвердження теми	06.11.2024	
2	Пошук відповідних джерел	08.12.2024	
3	Огляд джерел, виділення основною інформації	11.12.2024	
4	Створення плану	10.03.2025	
5	Написання, вступу, визначення актуальності, постановка завдань, написання розділу №1	11.03.2025	
6	Аналіз аналогів	16.03.2025	
7	Розробка дизайну	16.03.2025	
8	Розробка застосунку	29.03.2025	
9	Написання розділів № 2 і 3	20.04.2025	
10	Здача роботи науковому керівникові	08.05.2025	
11	Захист курсової робити	16.05.2025	

Студент: Молчанов О. К.

Керівник: Афонін А. О.

ЗМІСТ

АНОТАЦІЯ

У курсовій роботі описується процес розробки вебзастосунку для моніторингу здоров’я, що дозволяє користувачам вести свого здоров’я. Основна мета – створити зручний інструмент без реклами та платних обмежень.

У роботі досліджено предметну область, проаналізовано аналогічні застосунки, визначено вимоги, реалізовано авторизацію, персоналізований план, щоденний трекінг та статистику прогресу.

Практична перевага полягає в можливості моніторингу показників здоров’я у реальному часі.

ВСТУП

Актуальність теми. Здоровий спосіб життя та контроль за харчуванням стають сьогодні популярнішими. Багато людей прагнуть правильно харчуватися, бути активними та слідкувати за своїм здоров'ям. Проте часто складно самостійно контролювати свій раціон і фізичну активність. Сучасні технології і програми можуть допомогти вирішити цю проблему, даючи можливість слідкувати за показниками здоров'я. Також вебзастосунки, які допомагають користувачам слідкувати за своїм станом, стають все більш актуальними.

Мета дослідження полягає у розробці вебзастосунку для персонального моніторингу здоров'я з використанням методів аналізу даних.

Завдання дослідження:

1. Проаналізувати потреби користувачів у сфері моніторингу здоров'я.

2. Дослідити існуючі програми та їх можливості.

3. Розробити вебзастосунок з реєстрацією, персоналізованим планом і щоденним трекінгом.

4. Перевірити працездатність застосунку, протестувати його можливості.

Об'єкт дослідження – технології для цифрового моніторингу здоров'я.

Предмет дослідження – методи аналізу даних та клієнт-серверні технології для створення інтерактивних вебзастосунків.

Дослідження включає огляд предметної області, вивчення аналогів та розробку вебзастосунку на практиці. Для реалізації використано HTML, CSS, JavaScript, Node.js, Express.js та MongoDB. На ринку існує безліч мобільних застосунків для контролю здоров'я, подібних до MyFitnessPal. Вони дозволяють підраховувати калорії, контролювати фізичну активність та стан організму. Однак часто такі програми містять рекламу, платні

функції або складний інтерфейс. Вебзастосунки у цій сфері менш поширені, але мають переваги, зокрема зручність доступу з різних пристроїв. Враховуючи ці фактори, було вирішено створити вебзастосунок, який буде простим у використанні, без реклами та платних обмежень, а також доступним для україномовних користувачів.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Попит на здоровий спосіб життя та обґрунтування теми

За останнє десятиріччя помітно зріс попит на моніторинг і управління здоровим способом життя. Суспільство дедалі більше усвідомлює важливість профілактики здоров'я – це зумовило поширення глобального тренду на здоровий спосіб життя, і українці не є виключенням.

Однак, разом з тим, велика частина населення страждає через надмірну вагу та ожиріння. Згідно зі статтею Центру громадського здоров’я МОЗ України: “59,0% українців мають надлишкову масу тіла, а 24,8% – ожиріння” [1]. Передусім це зумовлено сучасною харчовою індустрією: висококалорійна їжа, надлишок цукру і солі, трансжирів та консервантів, доступність фаст-фуду, низька якість оброблених продуктів та сильний маркетинг, який маніпулює свідомістю. У поєднанні зі зростаючим малорухливим способом життя, у зв'язку із глобальною цифровізацією, це призводить до зростання серцево-судинних захворювань, діабету, порушення метаболізму тощо.

Для загального покращення самопочуття, усунення факторів ризику та перешкоджання розвитку захворювань на їхніх початкових стадіях, рекомендується щоденно відстежувати й дотримуватися норм, таких важливих показників здоров’я, як кількість спожитих калорій, співвідношення в організмі макронутрієнтів (БЖВ), об’єм випитою води й рівень фізичної активності.

У наслідок збігу двох факторів: тренду на здоров’я і проблеми надлишку ваги у значної частини населення – формується розуміння, що моніторинг здоров’я це не лише модно й сучасно, а й життєвонеобхідно. Таким, чином під цільову аудиторію різноманітних інструментів для відстеження здоров’я підпадають потенційні користувачі, як з метою їх використання для профілактики, так і з коригування ваги чи інших показників.

1.2 Стан цифрових рішень

У результаті широкої інтеграції технологій у повсякденне життя, стає можливим самостійний контроль і ведення обліку комплексних факторів здоров’я у режимі реального часу завдяки веб та мобільним застосункам. Для полегшення відстеження власного прогресу чи регресу користувача, спеціально налаштовані алгоритми збирають і синхронізують дані з носимих пристроїв з вбудованими датчиками, та динамічно візуалізують статистику у вигляді діаграм, графіків, тощо.

Абсолютне домінування на ринку належить мобільним рішенням, через зручність експлуатації смартфонів. При цьому, часто вебзастосунки є недооціненими, хоча й мають неабияку перспективу зростання, завдяки низці переваг: вебінтерфейси є зручнішими та ефективнішими у взаємодії із завданнями, які потребують роботу зі складною статистикою і великим обсягом інформації; робота в браузерах забезпечує використання будь-яких пристроїв (стаціонарні комп’ютери, ноутбуки, планшети, смартфони, тощо), незалежно від операційної системи.

Оскільки, у вебтехнологіях усі зміни та оновлення відбуваються лише на серверній стороні, без необхідності втручання з боку користувача, то обслуговування програмних продуктів стає простішим, а реагування на потреби користувача – швидшим, при цьому зберігається цілісність їх функціоналу.

1.3 Аналіз існуючих аналогів

У процесі роботи були досліджені як вебзастосунки, так і мобільні, проте основну увагу було зосереджено саме на мобільних, оскільки, нині, вебзастосунків у цій сфері значно менше, як зазначено у попередньому пункті. У результаті дослідження було зроблено висновок, що попри переваги, зазначені вище, більшість застосунків пропонують недостатньо зручний для всіх вікових категорій інтерфейс з багаторівневим меню, нав’язливою рекламою і низкою обмежень у функціоналі, які доступні

тільки в платних підписках. Також важливо зазначити, що більшість застосунків створюються для глобального ринку, тому, в більшості випадків, в них відсутня українська локалізація.

Усі наведені вище фактори демотивують і пригнічують зацікавленість користувачів, що негативне впливає на повсякденне використання інструменту. У результаті відсутності періодичних даних, статистика стає уривчастою і не правдивою, а потенціал та ефективність таких застосунків не розкривається до кінця.

Розглянемо на прикладі двох обраних застосунків: “MyFitnessPal” та “Таблиця Калорійності”. Їх об’єднує спільна концепція підрахунку та базовий трекінг активності користувачів, а також схожий набір елементів інтерфейсу:

- генерування персонального плану, згідно з даними про користувача;
- мета користувача (схуднення / набір ваги);
- лічильник спожитих калорій;
- співвідношення макронутрієнтів;
- кількість пройдених кроків;
- кількість випитої води;
- фізичні активності;
- додавання прийомів їжі (власні великі бази продуктів);
- статистика у вигляді графіків.

“MyFitnessPal” – мобільний застосунок, що вважається найпопулярнішим, серед наявних конкурентів. Згідно з інформацією з офіційного сайту, застосунок нараховує понад 200 мільйонів користувачів та містить базу понад 20 мільйонів харчових продуктів зі всього світу [2].

Головними перевагами застосунку є: продумана система харчових щоденників, підтримка голосового вводу для пошуку продуктів харчування, сканування штрихкодів для зчитування даних про продукт, а

також підтримка багатьох фітнестрекерів для синхронізації даних про фізичні активності, наприклад, Apple Health, Google Fit та ін.

Хоч сильні сторони, у MyFitnessPal прослідковується перевантаження інтерфейсу рекламними блоками і надмірною кількістю пропозицій щодо придбання преміум підписки, яка відкриває доступ до повного функціоналу, Таким чином фокус користувача на функціональних елементах інтерфейсу, конкурує з рекламними повідомленнями, які часто навіть не стосуються тематики здоров'я. Це супроводжується відчутним браком інтуїтивності взаємодії з інтерфейсом з погляду UX/UI, через складну ієрархію застосунку, при його використанні на етапі ознайомлення. (див. рис. 1.1)

Рисунок 1.1 – Інтерфейс застосунку "MyFitnessPal"

Більша частина можливостей інтерфейсу доступна лише за платною підпискою. Таке рішення позиціонує застосунок як преміум сегменту, який є недоступним для масового користувача, через можливі обмеження фінансових можливостей.

"Таблиця Калорійності" – мобільний застосунок, який, на відміну від MyFitnessPal, пропонує користувачеві простий, сучасний і мінімалістичний інтерфейс, що підходить під сучасні потреби користувачів різної вікової категорії. Перевагами даного застосунку є доступний в безплатній версії

сканер штрих-кодів продуктів харчування, що значно пришвидшує загальне використання застосунку. Також розробники доклали зусиль в UX/UI рішеннях, оскільки при тестуванні цього застосунку не виникло жодних проблем в інтуїтивності використання застосунку.

Попри переваги, описані вище, залишається спільна проблема для обох застосунків –обмежений доступ до повного функціоналу і наявність реклами.

Рисунок 1.2 – Інтерфейс застосунку "Таблиця Калорійності"

Після детального вивчення структур застосунків, що займають провідні місця на даному ринку, було прийнято рішення про розробку нового застосунку, у межах цієї курсової роботи, який врахує наявну проблематику існуючих додатків та запити користувачів, для охоплення більшої цільової аудиторії. Зважаючи на висвітлені раніше переваги вебзастосунків, у якості платформи було обрано саме веббраузер.

1.4 Функціональні вимоги до програмного забезпечення

Аналіз аналогів довів, що ринок інструментів для моніторингу здоров'я не є довершеним, тому виникає потреба у створенні нових рішень, які врахують ці помилки. Для того, щоб така система відповідала очікуванням потенційних користувачів, вона повинна бути інтуїтивно

простою у використанні, мати логічну ієрархію, обробляти щоденні показники та забезпечувати базову необхідну функціональність, без перенавантаження інтерфейсу.

Головними вимогами є:

- інтуїтивно зрозумілий інтерфейс, адаптований, включно для старшого покоління;
- прості механізми щоденного вводу даних;
- повна базова функціональність, з доступним, безплатним інтерфейсом і без нав'язливої реклами;
- підтримка української локалізації.

Розроблений в межах цієї курсової роботи інтерактивний вебзастосунок відповідає цим критеріям.

1.5 Постановка задачі

Основною метою розробки нового застосунку є створення простого інтерактивного застосунку, як інструменту для відстеження показників здоров'я і параметрів тіла, подолати дефіцит чи профіцит ваги. Користувач зможе моніторити свій прогрес та коригувати значення показників. На відміну від наявних рішень, новий застосунок зосереджений на базовому функціоналі без перевантаження інтерфейсу, орієнтований на щоденну взаємодію без вбудованої реклами, забезпечує підтримку української мови та містить у базі даних продукти виготовлені в Україні, має доступ до всіх можливостей без платних обмежень.

Важливою його відмінністю також є вираховування реального сценарію використання, адаптованого не тільки для молодого покоління, а й для старшого, яке часто зазнає труднощів з опануванням програмного забезпечення і частіше має розлади харчової поведінки на фоні пострадянського виховання.

Інтерактивний дашборд, простота у введенні даних, аналітична візуалізація показників та простий механізм обліку даних буде мотивувати

користувачів щоденно використовувати застосунок, що є основою для появи й підтримки здорових звичок.

Обрано основні задачі в межах функціональної структури вебзастосунка:

1. Сучасний, логічно структурований інтерфейс українською мовою, який динамічно оновлюється без потреби перезавантаження сторінки;
2. Швидка реєстрація:
    - введення персональних даних (ім’я, дата народження, вага, зріст, стать, спосіб життя);
    - вибір мети (схуднути або набрати вагу);
    - бажаний результат ваги
    - створення email і паролю з перевіркою його надійності;
3. Авторизація за email і паролем;
4. Автоматична генерація персонального плану:
    - розрахунок добової потреби у калоріях, воді, кроках та макронутрієнтах;
    - проста візуалізація всіх розрахунків.
5. Інтерактивний дашборд:
    - візуалізація прогресу користувача у вигляді кругових і лінійних індикаторів прогресу;
    - додавання і редагування спожитої їжі, води, пройдених кроків і поточної ваги користувача;
    - оцінка якості харчування;
    - вибір будь-якої дати в календарі, з можливістю переглядати та редагувати минулі або наступні дні;
6. Статистика у вигляді графіків за останній тиждень або місяць;
7. Створена локальна база продуктів з показниками їх харчової цінності;

8. Збереження інформації про користувача до окремих колекцій в базу даних про: реєстрацію, розрахований персональний план, прогрес кожен окремий день (вода, кроки, вага, спожита їжа та її оцінка якості).

Детальніше про реалізацію цих пунктів описано в розділах №2 і №3.

РОЗДІЛ 2 ТЕРЕОТИЧНІ ОСНОВИ

2.1 Підходи до побудови сучасних вебзастосунків

2.1.1 Загальні відомості про веб-розробку

Веброзробка – процес реалізації та підтримування вебсервісів, що включає розробку клієнтської і серверної частин, а також роботу з базами даних. Веброзробка є інтеграцією ряду технологій, інструментів і практик, спрямованих на забезпечення роботи цифрових продуктів у мережі Інтернет.

Front-end розробка стосується клієнтської частини та містить в собі все, що бачить і з чим взаємодіє користувач у вебзастосунку. До неї належать елементи інтерфейсу, такі як візуалізація оформлення а також механізми забезпечення інтерактивності. Основними технологіями, які використовуються для розробки front-end, є HTML для розмітки сторінок, CSS для стилізації та JavaScript для забезпечення динамічності. Front-end відповідає за те, щоб інформація була доступною, структурованою та представлена у зручному для користувача вигляді. При цьому акцент робиться не лише на функціональності, але й на збереженні високої якості користувацького досвіду (UX) та привабливого інтерфейсу (UI).

Back-end, є “двигуном” вебзастосунку і відповідає за обробку запитів, реалізацію бізнес-логіки, роботу з даними та забезпечення безпеки системи. Сюди входять сервери, додатки та програмні компоненти, які отримують запити від клієнта, обробляють їх і повертають результати у вигляді структурованої інформації. В використовує різноманітні мови програмування (такі як JavaScript/Node.js, Python, PHP) та фреймворки для створення RESTful-сервісів, що дозволяють забезпечити надійну комунікацію між різними частинами застосунку.

База даних є невід'ємною складовою сучасного вебзастосунку, оскільки вона відповідає за зберігання, обробку та видачу даних, необхідних для роботи програми. Головне завдання бази даних –

організувати ефективне зберігання інформації, забезпечити її надійний доступ і підтримувати високу продуктивність при поступанні великого обсягу запитів. Зазвичай бази даних поділяють на два основних типи:

- реляційні бази даних – структура даних у таких системах базується на таблицях, що складаються з рядків і стовпців. Дані пов’язані між собою за допомогою ключів, що дозволяє створювати складні зв’язки та виконувати розширені SQL-запити. Прикладами реляційних баз є PostgreSQL та MySQL, які широко використовуються завдяки своїй надійності та продуктивності.
- нереляційні бази даних (NoSQL) – системи , що зберігають дані у вигляді документів, широких таблиць і графіків, що дозволяє їмефективно обробляти великі обсяги інформації, що не вписуються у традиційну табличну модель. Серед найпопулярніших NoSQL - рішень варто зазначити MongoDB, яка зберігає дані у форматі, схожому на JSON, та Redis, що використовує пам’ять для особливо швидкого доступу до структур даних, таких як списки і хеші [3].

Ці аспекти є критично важливими для забезпечення привабливості та зручності використання вебзастосунку.

Таким чином, веброзробка являє собою багатокомпонентний процес, де кожен елемент – від створення зовнішнього вигляду до забезпечення внутрішньої логіки та збереження даних – працює у взаємодії для створення надійного, зручного та масштабованого продукту.

2.1.2 Клієнт-серверна архітектура

Клієнт-серверна архітектура поділяє вебзастосунок на дві частини: клієнт і сервер.

Звичайний пристрій, з якого надсилається запит на сервер для отримання повної інформації або з метою виконання конкретних дій – називають “клієнтом”. Якщо розглядати вебзастосунки з цієї точки зору, то

клієнтом зазвичай є веб-браузер, завдяки якому користувач взаємодіє із програмою.

Сервером називають систему, яка реагує на запит клієнта і призначена для повернення користувачу результатів. Сервер одночасно може обробляти декілька запитів, якщо таких запитів багато, він розподіляє ці запити в чергу, обробляє і відповідає послідовно по встановленим пріоритетам. Зазвичай найбільш пріоритетні запити обробляються швидше, або формується мережева група з декількох серверів, яка ефективно обробляє зростаючі навантаження при більшій кількості клієнтів [4].

HTTP (Hyper Text Transfer Protocol) – головний протокол для передачі даних у вебзастосунках. Він забезпечує стандартний спосіб обміну інформацією між клієнтами, зазвичай веб-браузерами, та серверними системами.

HTTP протокол працює за моделлю “запит-відповідь”, коли клієнт надсилає запит на сервер, а сервер відповідає потрібним ресурсом або даними. Проте, оскільки стандартний HTTP передає інформацію у незашифрованому вигляді, дані, що передаються через нього, можуть бути вразливими для перехоплення та змін зловмисниками. Саме тому з’явилась його покращена захищена версія.

HTTPS (Hyper Text Transfer Protocol Secure) використовує протокол шифрування TLS (натомість SSL) для шифрування даних, що забезпечує безпечну комунікацію між клієнтом і сервером. Це особливо важливо при передачі конфіденційної інформації, наприклад, особистих даних або платіжних реквізитів, оскільки шифрування гарантує, що навіть якщо з’єднання буде перехоплено, дані залишаться незрозумілими для зловмисників [5].

Щодо методів HTTP, вони визначають тип дії, яку потрібно виконати над ресурсом. Основні методи HTTP включають:

• GET використовується для отримання даних або ресурсів із сервера. Запити цього типу повинні бути безпечними, тобто не змінювати стан сервера.

• POST застосовується для надсилання даних до сервера, наприклад, при створенні нового запису або надсиланні форми. Цей метод може змінювати стан ресурсу.

• PUT використовується для повної заміни або оновлення існуючого ресурсу. На відміну від POST, PUT зазвичай використовується для заміни конкретного ресурсу за заданою URL-адресою.

• DELETE служить для видалення вказаного ресурсу з сервера.

Кожен з цих методів має свої семантичні особливості та застосування, що дозволяє розробникам чітко керувати операціями над даними. За допомогою правильного використання цих методів можна оптимізувати роботу застосунку, забезпечити коректну обробку запитів і, таким чином, покращити ефективність взаємодії між клієнтом і сервером [6].

2.1.3 Відмінності між SPA та MPA

SPA (Single-Page Application) – це метод розробки вебзастосунка, який розповсюджується виключно на використанні у веббраузері, що не потребує перезавантаження сторінки та динамічно оновлює контент в межах однієї HTML-сторінки. Основною перевагою односторінкових застосунків є швидкість переходів, що додає відчуття плавності користувацького досвіду. Основними варіантами технологій, які пропонують адаптивний і швидкий інтерфейс вебзастосунків є “Angular”, “React.js” і “Vue.js”. У приклад таких застосунків можна навести “Google Docs”, “Google Sheets” і “Gmail”.

Щодо недоліків можна відзначити:

- попри швидкість оновлення під час використання SPA-застосунку, початковий запуск може зайняти деякий час через завантаження фреймворків JavaScript;
- складністю розробки є керування маршрутизацією клієнтської сторони та обробки асинхронного потоку даних;
- через особливості пошукових систем в індексації окремих HTML-сторінок, такі застосунки можуть зіткнутися з труднощами з точки зору SEO (Search Engine Optimization).

SPA варто обирати, коли необхідно мати швидку реакцію сервера з миттєвим оновленням сторінки та багатофункціональний UI.

MPA (Multi-Page Aplication) – другий метод розробки, який передбачає собою наявність двох і більше статичних сторінок, у кожної з яких є своє персональне призначення. Такі сторінки мають зазвичай різні макети інтерфейсу, які відрізняються один від одного та окремий функціонал, адаптований під закладені в нього задачі. Відомими прикладами вебзастосунків, розроблених таким за таким методом є “eBay”, “Amazon” і “Facebook”.

Мінусами цього методу є повільніша взаємодія з користувачем і завантаження сторінок при слабкому інтернет-з’єднанні, а також навантаження сервера, надсиланням багатьох запитів з окремих сторінок.

MPA вартує уваги, якщо метою вебзастосунку є масштабування, зручна статистика й аналітика кожної сторінки, CEO-оптимізація, різні вхідні точки (швидке завантаження простих сторінок) [7].

2.2 Використаний стек технологій

2.2.1 Front-end (клієнтська частина)

HTML (Hyper Text Markup Language) – статична мова розмітки вебзастосунку, яка не вважається мовою програмування і не забезпечує створення інтерактивної функціональності. Її призначення – анотація

структури. Основною задачею HTML є повідомлення веббраузера про послідовність розміщення тексту, посилань, фотографій тощо [8].

CSS (Cascading Style Sheets) – мова, що відповідає за стилізацію структурних блоків HTML. Важливістю використання у вебзастосунках мови у проєкті є візуалізація розмірів, шрифтів, кольорів, розміщення елементів та відстань між ними, а також створення анімацій.

JavaScript – інтерпретована мова програмування, яка відповідає за динаміку створеного контенту на HTML і CSS. Це універсальна мова, яка працює як на стороні клієнта, так і сервера. На клієнтській стороні JavaScript забезпечує взаємодію користувача з кнопками, введенням даних, відображенням анімації, тощо. На вебсервері використовується для підключення до баз даних, безпечним отримуванням і поверненням даних браузерам та обробки даних [9].

2.2.2 Back-end (серверна частина)

Node.js – це середовище виконання JavaScript. Дозволяє запускати JavaScript за межами браузера та створювати високопродуктивні серверні застосунки завдяки неблокуючій, подієвій моделі вводу-виводу, що робить його чудовим вибором для розробки масштабованих мережевих додатків.

Express.js – це веб-фреймворк для Node.js, який значно спрощує розробку серверних застосунків. Він надає ізольований набір інструментів і модулів для організації маршрутизації HTTP-запитів, обробки даних та створення RESTful API, що дозволяє швидко та ефективно організувати серверну логіку без надмірної складності.

Mongoose – це бібліотека для Node.js, яка дає зручне управління даними при роботі з MongoDB. Вона дозволяє визначати схеми для колекцій у базі даних, автоматизувати процес валідації даних, а також виконувати операції створення, читання, оновлення й видалення даних, що сприяє підвищенню стабільності та структурованості серверного коду.

2.2.3 Додаткові інструменти

Chart.js – проста і безкоштовна бібліотека для JavaScript й усіх його фреймворків з технологією canvas HTML5 (остання версія мови), яка дозволяє відображати усі типи діаграм різними налаштуваннями [10].

Fetch API – технологія у JavaScript, яка дозволяє надсилати HTTP запити на зазначену URL-адресу, після чого цей запит обробляється і повертається. При цьому відповідь конвертується у JSON-форматі.
В межах розробки вебзастосунку під час курсової роботи було використано такі типи HTTP-запитів у Fetch API: GET – отримання даних з іншого сервера; POST – додавання даних на сервер.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1. Опис розробки

3.1.1 Реалізація авторизації та реєстрації користувача

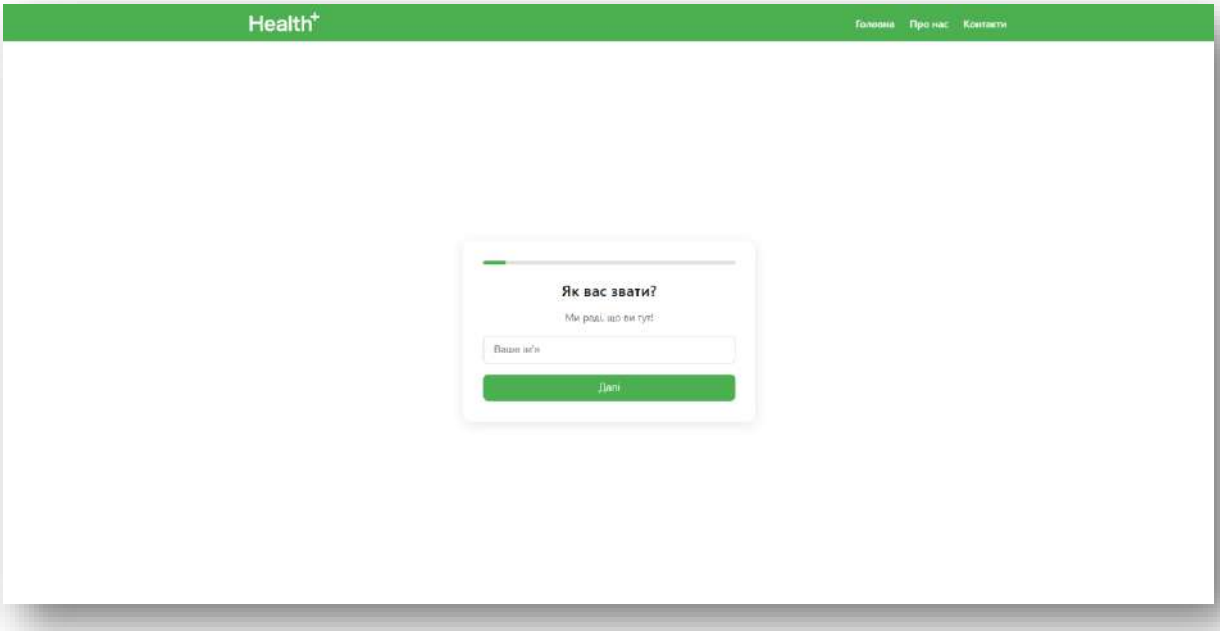

Рисунок 3.1 – Інтерфейс реєстраційного меню

Логіка реєстрації така, що користувач поступово проходить форму реєстрації: дата народження, вага, зріст, стать, кількість прийомів їжі на день, мета, бажана зміна ваги та скільки калорій користувач хоче спалювати або набирати за день. Кожен крок перевіряється на коректність допустимих значень. (див. рис. 3.2)

```
if (currentEl.querySelector("#weight")) {
  const weight = parseInt(document.getElementById("weight").value);
  if (weight < 30 || weight > 300) {
    showError(stepIndex, "Вага має бути від 30 до 300 кг.");
    return;
  }
}
```

Рисунок 3.2 – Приклад валідації для ваги (register.js)

Після успішного проходження усіх кроків форми, на стороні клієнта формується об'єкт “userData”, що містить усю зібрану інформацію про користувача: ім’я, електронну пошту, дату народження, фізіологічні

параметри (вага, зріст), стать, спосіб життя, кількість прийомів їжі на день, а також ціль (схуднення або набір ваги) та пов'язані з нею параметри – бажана зміна ваги й кількість калорій для досягнення мети. Цей об'єкт перетворюється у формат JSON та надсилається на сервер за допомогою методу fetch() через POST-запит. (див. рис. 3.3)

```
const response = await fetch("http://localhost:5000/api/auth/register", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(userData)
});
```

Рисунок 3.3 – POST-запит з об'єктом userData (register.js)

На сервері запит обробляється у функції "registerUser". Спершу на back-end-і є перевірка наявності користувача з таким email у базі даних. Для цього використовується метод User.findOne({ email }), який здійснює пошук по колекції users. Якщо користувач уже існує – сервер повертає відповідь зі статусом 400 та повідомленням про помилку. Такий механізм запобігає повторній реєстрації одного й того ж облікового запису. (див. рис. 3.4)

```
const existingUser = await User.findOne({ email });
if (existingUser) {
  return res.status(400).json({ message: "Користувач з такою поштою вже існує" });
}
```

Рисунок 3.4 – фрагмент функції registerUser для перевірки унікальності за email (authController.js)

Якщо email є коректним і унікальним, система переходить до наступного кроку – хешування пароля, який надійшов від клієнта. Для цього використовується бібліотека "bcrypt". (див. рис. 3.5)

```
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);
```

Рисунок 3.5 – Хешування пароля (authController.js)

Такий підхід забезпечує високий рівень захисту збережених паролів у разі потенційного витоку бази даних, адже хешоване значення неможливо зворотно декодувати до початкового.

Після обробки пароля формується новий об'єкт “User” згідно з попередньо визначеною схемою “Mongoose” у файлі User.js. В об'єкті зазначаються всі поля, необхідні для збереження: персональні дані, фізичні параметри, обрана ціль, кількість прийомів їжі тощо. Для полів, які є актуальними лише при цілях “схуднення” або “набір ваги” (target Weigt Change, Calories Per Day), встановлюється умова: якщо мета вимагає цих даних – вони зберігаються, інакше фіксуються як “null”. Після формування екземпляра моделі, запис зберігається у колекцію MongoDB за допомогою методу newUser.save(). (див. рис. 3.6)

```
const newUser = new User({
  nickname,
  email,
  password: hashedPassword,
  birthDate,
  weight,
  height,
  gender,
  mealsPerDay,
  lifestyle,
  goal,
  targetWeightChange: isGoalRequiringChange(goal) ? targetWeightChange : null,
  caloriesPerDay: isGoalRequiringChange(goal) ? caloriesPerDay : null
});

await newUser.save();
```

Рисунок 3.6 – Створення та збереження нового користувача у MongoDB (authController.js)

Функціональність входу до системи реалізована у функції “loginUser”, яка обробляє POST-запити за маршрутом /api/auth/login. Цей процес відбувається таким чином:

- на першому етапі сервер за допомогою методу “User.findOne” виконує пошук у базі даних за вказаним email. Якщо користувача не

знайдено – повертається помилка з повідомленням, що такого користувача не знайдено. (див. рис. 3.7)

- у випадку, якщо користувач знайдений, відбувається порівняння надісланого пароля з тим, що збережений у базі. Для цього використовується функція “bcrypt.compare”, яка порівнює введений пароль з хешованим значенням у полі “user.password”. Якщо паролі не збігаються – повертається повідомлення про те, що пароль не правильний. (див. рис. 3.7)
- після успішної автентифікації генерується JWT-токен. Він містить зашифрований об'єкт з ідентифікатором користувача (userId) і підписується за допомогою секретного ключа JWT_SECRET, що зберігається у файлі “.env”. Термін дії токена встановлено на 7 днів (expiresIn: "7d"). (див. рис. 3.7)
- згенерований токен надсилається у відповідь разом із додатковими полями (ім’я, email, стать користувача). Ці дані зберігаються у “localStorage” на клієнтській стороні та використовуються для автентифікації в подальших запитах. (див. рис. 3.7)

```
const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: "Користувача не знайдено" });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: "Невірний пароль" });
    }

    const token = jwt.sign(
      { userId: user._id },
      process.env.JWT_SECRET,
      { expiresIn: "7d" }
    );

    res.status(200).json({
      token,
      nickname: user.nickname,
      email: user.email,
      gender: user.gender
    });

  } catch (error) {
    res.status(500).json({ message: "Щось пішло не так..." });
  }
};
```

Рисунок 3.7 – функція loginUser для обробки логіну користувача (authController.js)

Такий спосіб забезпечує безпечну авторизацію, уникаючи збереження сесій на сервері, що є перевагою безстатусної архітектури REST API.

На стороні клієнта токен та інші дані (ім’я, email, стать) зберігаються в localStorage браузера. (див. рис. 3.8)

```
localStorage.setItem("token", token);
localStorage.setItem("userName", userData.nickname);
localStorage.setItem("userEmail", userData.email);
localStorage.setItem("userGender", userData.gender);
```

Рисунок 3.8 – Збереження токена та даних користувача у localStorage (register.js)

Це забезпечує збереження сесії користувача навіть після оновлення сторінки або повторного входу до застосунку. Це дозволяє front-end-у передавати токен з кожним запитом до серверної частини (включаючи план, прогрес, їжу, тощо).

3.1.2 Створення плану

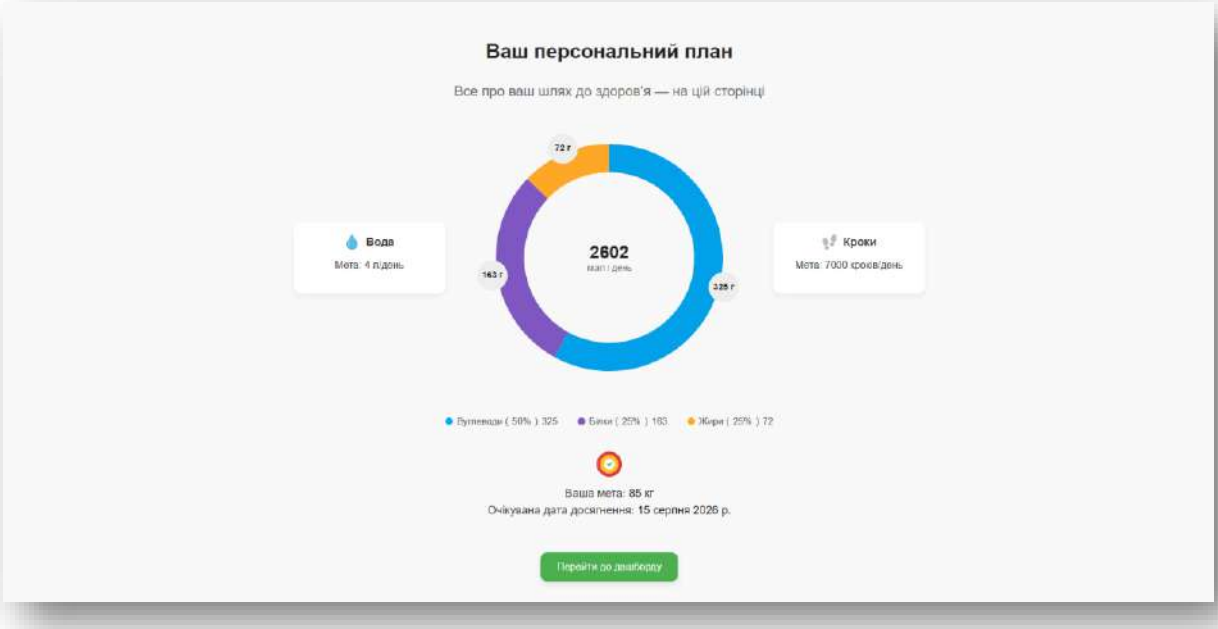

Рисунок 3.9 – Інтерфейс сторінки персонального плану користувача

Алгоритми розрахунку вибраних формул:

- розрахунок віку користувача. Щоб правильно сформувати персональний план – потрібно знати вік користувача,.

Вік (age) визначається як різниця між поточним роком та роком народження користувача. (див. рис. 3.10)

```
const currentYear = new Date().getFullYear();
const birthYear = new Date(birthDate).getFullYear();
const age = currentYear - birthYear;
```

Рисунок 3.10 – Розрахунок віку (plan.controller.js)

Це необхідно для подальших формул оцінки обміну речовин.

- розрахунок базової швидкості обміну речовин (BMR). Щоб зрозуміти скільки калорій людині необхідно, щоб просто підтримувати життя, навіть в пасивному стані, треба використовувати формулу BMR Міффліна-Сент-Джора:

1. для чоловіків: BMR = 10 * вага + 6.25 * зріст − 5 * вік + 5;

2. для жінок: BMR = 10 * вага + 6.25 * зріст − 5 * вік – 161 [11].

Тут: вага – у кг, зріст – у см, вік – у роках.

Ця формула вважається одною з найточніших для оцінки добової потреби організму в енергії. (див. рис. 3.11)

```
const BMR = gender === 'Чоловік'
  ? 10 * weight + 6.25 * height - 5 * age + 5
  : 10 * weight + 6.25 * height - 5 * age - 161;
```

Рисунок 3.11 – Розрахунок обміну речовин (plan.controller.js)

Цей метод показує мінімальну кількість калорій, яку людину має отримувати кожен день.

- після обчислення BMR наступним кроком розраховується коефіцієнт активності TDEE (Total Daily Energy Expenditure), у реальному житті – це кількість калорій, яку спалює людина на день, враховуючи її спосіб життя.

Формула така TDEE = BMR * activityFactor, зі значеннями коефіцієнтів – сидячий: 1.2, помірно активний: 1.375, активний: 1.55. (див. рис. 3.12)

```
const activityFactors = {
  'сидячий': 1.2,
  'помірно активний': 1.375,
  'активний': 1.55
};
const activityFactor = activityFactors[lifestyle] || 1.2;
const TDEE = BMR * activityFactor;
```

Рисунок 3.12 – Розрахунок TDEE (plan.controller.js)

- розрахунок цільової калорійності. Якщо ціль користувача – зміна ваги, калорійність коригується. (див. рис. 3.13)

```
let calorieGoal = TDEE;
let weeksToGoal = null;
let targetWeight = weight;
```

Рисунок 3.13 – Розрахунок цільової калорійності (plan.controller.js)

- коригування калорій для схуднення або набору ваги.

  3850 ккал ≈ 0.5 кг жиру, тому дефіцит/надлишок у 3850 ккал/тиждень ≈ 0.5 кг зміни ваги.

  Обмеження на 7000 ккал/тиждень ≈ 1 кг.

  Також обмеження у 1000 ккал/день як безпечна межа. (див. рис. 3.14)

```
if (goal === 'схуднення' || goal === 'набір ваги') {
  let kcalChangePerWeek = caloriesPerDay || 3850;

  if (kcalChangePerWeek > 7000) {
    kcalChangePerWeek = 7000;
  }

  const rawKcalPerDay = kcalChangePerWeek / 7;
  const kcalPerDay = Math.min(rawKcalPerDay, 1000);
```

Рисунок 3.14 – Коригування калорій (plan.controller.js)

- розрахунок калорійної цілі з урахуванням мети.

  При схудненні – зменшення TDEE, але не нижче мінімуму: чоловіки: 1500 ккал, жінки: 1200 ккал, при наборі ваги – надбавка до TDEE. (див. рис. 3.15).

```
calorieGoal = goal === 'схуднення'
  ? Math.max(TDEE - kcalPerDay, MIN_CALORIES)
  : TDEE + kcalPerDay;
```

Рисунок 3.15 – розрахунок калорій з урахуванням мети (plan.controller.js)

- розрахунок тривалості досягнення мети. Оскільки 0.45 кг ≈ 1 фунт, це – медичний стандарт зміни ваги за 1 тиждень. (див. рис. 3.16)

```
weeksToGoal = Math.ceil(Math.abs(targetWeightChange) / 0.45);
```

Рисунок 3.16 – розрахунок тривалості досягнення мети (plan.controller.js)

- розрахунок кінцевої ваги. Фіксація бажаної маси тіла після виконання плану. (див. рис. 3.17)

```
  targetWeight = goal === 'схуднення'
    ? weight - Math.abs(targetWeightChange)
    : weight + Math.abs(targetWeightChange);
}
```

Рисунок 3.17 – розрахунок кінцевої ваги (plan.controller.js)

- розрахунок макронутрієнтів (БЖВ).

  Білки = 25% калорій / 4 (1 г білка = 4 ккал)

  Жири = 25% калорій / 9 (1 г жиру = 9 ккал)

  Вуглеводи = 50% калорій / 4 (1 г вуглеводів = 4 ккал)

  Це класичне розподілення для збалансованого харчування: 50/25/25. (див. рис. 3.18)

```
const proteinGrams = Math.round((calorieGoal * 0.25) / 4);
const fatGrams = Math.round((calorieGoal * 0.25) / 9);
const carbGrams = Math.round((calorieGoal * 0.5) / 4);
```

Рисунок 3.18 – Розрахунок БЖВ (plan.controller.js)

- розрахунок норми води за формулою: норма води = вага * 0.035 (л) Встановлені під час розробки безпечні обмеження мінімум: 1.5 л, максимум: 4 л води. (див. рис. 3.19)

```
const baseWater = weight * 0.035;
const waterLiters = +Math.max(1.5, Math.min(baseWater, 4)).toFixed(1);
```

Рисунок 3.19 – Розрахунок норми води (plan.controller.js)

- розрахунок кількості кроків на день. Залежно від способу життя встановлюється норма кроків на день – це простий, але ефективний підхід до формування цільової активності. (див. рис. 3.19)

```
const stepsByLifestyle = {
  'сидячий': 4000,
  'помірно активний': 7000,
  'активний': 10000
};
const stepsPerDay = stepsByLifestyle[lifestyle] || 4000;
```

Рисунок 3.19 – Розрахунок кількості кроків на день (plan.controller.js)

- розрахунок очікуваної дати досягнення мети (рис. 3.19).

```
if (weeksToGoal) {
  targetDate = new Date();
  targetDate.setDate(targetDate.getDate() + weeksToGoal * 7);
}
```

Рисунок 3.19 – Розрахунок очікуваної дати досягнення мети (plan.controller.js)

Це проста проєкція у майбутнє – додаємо кількість тижнів до поточної дати.

3.3.3. Інтерфейс дашборду

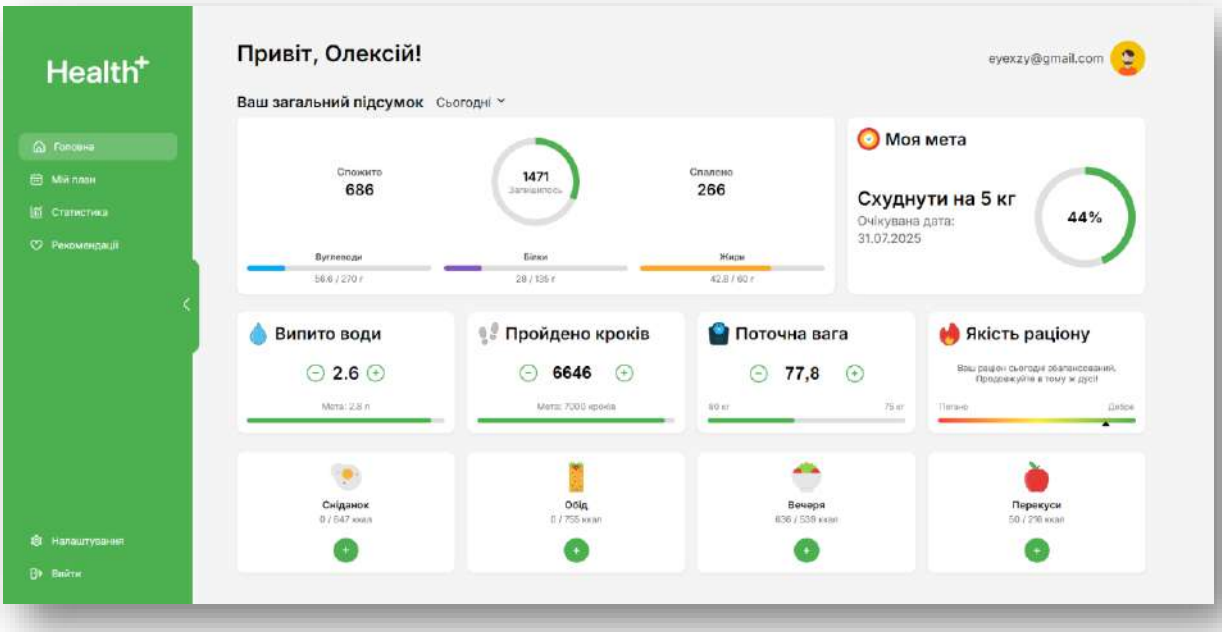

Рисунок 3.20 – Головна сторінка дашборду

- після завантаження сторінки з планом користувач потрапляє на дашборд.

Застосунок отримує ім’я користувача, його електронну пошту та стать через API-запит /api/dashboard/user. Дані зберігаються у змінну “user”, після чого автоматично оновлюється вітальний текст "Привіт, … !", а також підставляється email у правому верхньому куті. Крім того, аватар змінюється в залежності від статі: чоловікам – man.svg, жінкам – girl.svg. (див. рис. 3.21)

```
const userRes = await fetch(`http://localhost:5000/api/dashboard/user`, {
  headers: { Authorization: `Bearer ${token}` }
});
if (!userRes.ok) throw new Error("Помилка авторизації");
const user = await userRes.json();
await fetchMealCalories();
await updateFoodSummary();
await loadGoalInfo();
await updateNutritionQuality();
document.getElementById("greeting-name").textContent = user.name;
document.getElementById("profile-email").textContent = user.email;
const gender = user.gender.toLowerCase();
document.getElementById("avatar-icon").src =
gender === "ж" || gender === "female" ? "Icons/girl.svg" : "Icons/man.svg";
```

Рисунок 3.21 – Відображення імені та пошти користувача (dashboard.js)

- відображення дати дашборду. На панелі над основним блоком статистики відображається поточна або обрана користувачем дата.

За замовчуванням встановлюється сьогоднішній день. При натисканні на дату відкривається модальне вікно з полем <input type="date">, яке дозволяє перемикати дні. Дата зберігається у localStorage для подальшого використання та підвантаження відповідного прогресу. (див. рис. 3.22)

```
const todayStr = new Date().toISOString().split("T")[0];
const isToday = selectedDate === todayStr;
document.getElementById("currentDateText").innerHTML =
  (isToday ? "Сьогодні" : new Date(selectedDate).toLocaleDateString('uk-UA')) +
  '<i class="fi fi-rr-angle-small-down"></i>';
```

Рисунок 3.22 – Відображення та перемикання дати (dashboard.js)

- у центральній секції дашборду відображається загальний залишок калорій, а також інформація про спожиті калорії, білки, жири та вуглеводи.

Ці дані отримуються через запит /api/dashboard/summary, де API повертає caloriesGoal, caloriesEaten, macros, burned. Після цього заповнюються відповідні текстові поля та прогрес-бари. Для кругової діаграми використовуються налаштування Chart.js. (див. рис. 3.23)

```
async function getDailySummary() {
  try {
    const summaryRes = await fetch(`http://localhost:5000/api/dashboard/summary?date=${selectedDate}`, {
      headers: { Authorization: `Bearer ${token}` }
    });
    const summary = await summaryRes.json();
    const macroBars = document.querySelectorAll(".macro-bar-value");
    if (macroBars.length >= 3) {
      macroBars[0].textContent = `${summary.macros.carbs.eaten} / ${summary.macros.carbs.goal} г`;
      macroBars[1].textContent = `${summary.macros.protein.eaten} / ${summary.macros.protein.goal} г`;
      macroBars[2].textContent = `${summary.macros.fat.eaten} / ${summary.macros.fat.goal} г`;
    }
    document.querySelector(".macro-center-text div").textContent = summary.caloriesLeft;
    createDoughnutChart("macroRingChart", summary.caloriesEaten, summary.caloriesGoal, "#4caf50");
```

Рисунок 3.23 – Відображення калорій та макросів (dashboard.js)

- відображення блоку "Моя мета". Після входу в дашборд застосунок надсилає запит на /api/dashboard/goal, щоб отримати основну інформацію про ціль користувача. Повертаються дані: тип мети (goalType), бажана зміна ваги (goalKg), цільова дата (targetDate) та відсоток прогресу (progress). У блоці відображається текст типу “Схуднути на 5 кг”, прогнозована дата досягнення мети та кругова діаграма з поточним відсотком виконання. (див. рис. 3.24)

Рисунок 3.24 – Відображення цілі користувача (dashboard.js)

- розрахунок і відображення норми води.

Кількість випитої води за день показується в лічильнику, де користувач може вручну додавати/віднімати по 0.1 л за допомогою кнопок. Норма води розраховується на основі ваги користувача: норма = вага * 0.035, з обмеженнями: мінімум 1.5 л, максимум 4.0 л. Цей показник відображається під значенням, а також як прогресбар. Зміни надсилаються через API /api/dashboard/water. (див. рис. 3.25)

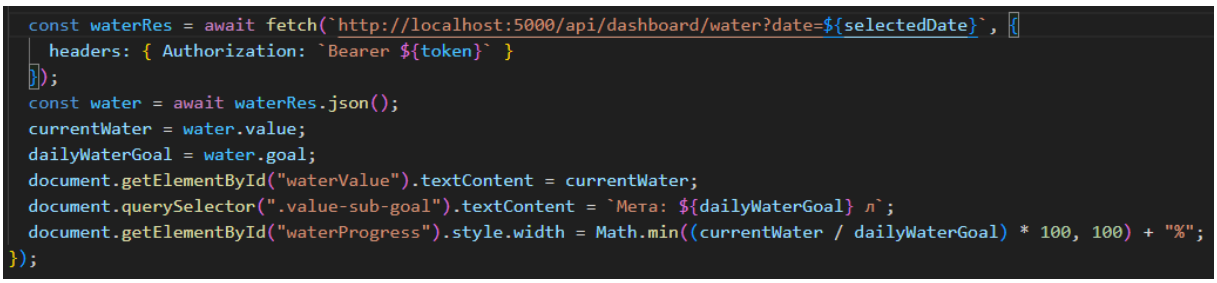

Рисунок 3.25 – Відображення випитої води (dashboard.js)

- розрахунок і відображення кількості кроків. Блок "Пройдено кроків" дозволяє користувачу вносити кількість пройдених кроків вручну, а також змінювати значення за допомогою кнопок ±100. Прогресбар оновлюється відповідно до виконаної норми.

Дані завантажуються з бекенду через /api/dashboard/steps, а зміни зберігаються через POST-запит з параметром delta. Крім того, автоматично перераховується кількість спалених калорій (приблизно 0.04 ккал на 1 крок). (див. рис. 3.26)

```
const stepsRes = await fetch(`http://localhost:5000/api/dashboard/steps?date=${selectedDate}`, {
  headers: { Authorization: `Bearer ${token}` }
});
const steps = await stepsRes.json();
currentSteps = steps.value;
goalSteps = steps.goal;
document.getElementById("stepsInput").value = currentSteps;
document.querySelector(".steps-block .value-sub-goal").textContent = `Мета: ${goalSteps} кроків`;
document.getElementById("stepsProgress").style.width = Math.min((currentSteps / goalSteps) * 100, 100) + "%";
```

Рисунок 3.26 – Відображення кроків та прогресу (dashboard.js)

- Відображення поточної ваги. Користувач може змінювати свою вагу вручну – кнопками по (+) та (-) 1 кг або ввівши значення безпосередньо. Значення ваги зберігається на кожен окремий день у базі даних через progress.weight. Прогрес візуалізується на лінійній шкалі: початкова вага – поточна вага – ціль. При зміні ваги автоматично оновлюється і відсоток досягнення мети. (див. рис. 3.27)

```
const weightRes = await fetch(`http://localhost:5000/api/dashboard/weight?date=${selectedDate}`, {
  headers: { Authorization: `Bearer ${token}` }
});
const weight = await weightRes.json();
currentWeight = weight.value;
startWeight = weight.start;
goalWeight = weight.goal;
updateWeightUI();
```

Рисунок 3.27 – Відображення прогресу зміни ваги (dashboard.js)

- оцінка якості раціону. У блоці “Якість раціону” виводиться оцінка збалансованості харчування користувача на основі індексу nutritionQuality. Цей індекс – середнє значення healthScore усіх продуктів, доданих протягом дня. Значення від 0 до 100 виводиться як положення стрілки на лінії (погано–добре) та відповідне текстове повідомлення. (див. рис. 3.28)

```
async function updateNutritionQuality() {
  try {
    const res = await fetch(`http://localhost:5000/api/dashboard/summary?date=${selectedDate}`, {
      headers: { Authorization: `Bearer ${token}` }
    });
    if (!res.ok) throw new Error("Не вдалося отримати зведення");

    const summary = await res.json();
    const quality = summary.nutritionQuality || 0;
    const indicator = document.getElementById("qualityIndicator");
    if (!indicator) return;

    indicator.style.left = `${Math.min(100, Math.max(0, quality))}%`;

    const feedback = document.getElementById("nutritionFeedback");
    if (quality >= 80) {
      feedback.textContent = "Ваш раціон сьогодні збалансований. Продовжуйте в тому ж дусі!";
    } else if (quality >= 50) {
      feedback.textContent = "Можна краще! Обирайте більше корисних продуктів.";
    } else {
      feedback.textContent = "Раціон незбалансований. Рекомендуємо звернути увагу на вибір продуктів.";
    }
  } catch (err) {
  }
}
```

Рисунок 3.28 – Оцінка якості раціону (dashboard.js)

- відображення прийомів їжі (сніданок, обід, вечеря, перекус).

  Кожен із блоків харчування показує, скільки ккал уже спожито у відповідний прийом їжі та яка максимальна норма калорій. При завантаженні сторінки ці значення автоматично розраховуються на основі загальної калорійної мети користувача. Розподіл такий: Сніданок – 30%, Обід — 35%, Вечеря – 25%, Перекус – 10%.
  Ці значення зберігаються в об’єкті mealCalories. (див. рис. 3.29)

```
mealCalories = {
  breakfast: Math.round(total * 0.3),
  lunch: Math.round(total * 0.35),
  dinner: Math.round(total * 0.25),
  snack: Math.round(total * 0.1),
};
```

Рисунок 3.29 – Розподіл добових калорій між прийомами їжі (dashboard.js)

- додавання продуктів через модальне вікно. Кнопка "+" у кожному блоці відкриває модальне вікно, де користувач може знайти продукт за назвою та додати його до певного прийому їжі. У результаті дані

про продукт (назва, калорійність, БЖВ, healthScore) записуються у колекцію foodentries. (див. рис. 3.30)

```
addBtn.addEventListener("click", async () => {
  await fetch(`http://localhost:5000/api/dashboard/food?date=${selectedDate}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`
    },
    body: JSON.stringify({
      date: today,
      meal: selectedMeal,
      query: food.name
    })
  });
```

Рисунок 3.30 – Додавання продукту в прийом їжі (dashboard.js)

- підрахунок сумарної кількості калорій та макросів.

Після додавання або видалення продукту система автоматично оновлює інформацію по: загальних калоріях, кількості калорій у кожному прийомі їжі. Це реалізовано у функції updateFoodSummary, яка обчислює значення і оновлює відповідні елементи на сторінці. (див. рис. 3.31)

```
  const plan = await totalPlan.json();
  const carbsPercent = Math.min((summary.carbs / plan.carbGrams) * 100, 100);
  const proteinPercent = Math.min((summ any rotein / plan.proteinGrams) * 100, 100);
  const fatPercent = Math.min((summary.fat / plan.fatGrams) * 100, 100);
  const macroBars = document.querySelectorAll(".macro-bar");
  const macroValues = document.querySelectorAll(".macro-bar-value");
  if (macroBars.length >= 3 && macroValues.length >= 3) {
    macroBars[0].firstElementChild.style.width = `${carbsPercent}%`;
    macroBars[1].firstElementChild.style.width = `${proteinPercent}%`;
    macroBars[2].firstElementChild.style.width = `${fatPercent}%`;
    macroValues[0].textContent = `${summary.carbs} / ${plan.carbGrams} г`;
    macroValues[1].textContent = `${summary.protein} / ${plan.proteinGrams} г`;
    macroValues[2].textContent = `${summary.fat} / ${plan.fatGrams} г`;
  }
```

Рисунок 3.31 – Підрахунок зведення за макросами (dashboard.js)

- вибір дати та збереження даних у БД. Всі зміни, які виконує користувач, прив’язані до конкретної дати. Коли обирається інша дата у випадаючому календарі, застосунок: оновлює значення selectedDate; перевіряє, чи існує запис у колекції progresses; якщо ні – створює новий запис; завантажує всі відповідні дані з бази. (див. рис. 3.32)

```
const todayStr = new Date().toISOString().split("T")[0];
const isToday = selectedDate === todayStr;

document.getElementById("currentDateText").innerHTML =
  (isToday ? "Сьогодні" : new Date(selectedDate).toLocaleDateString('uk-UA')) +
  '<i class="fi fi-rr-angle-small-down"></i>';

    await getDailySummary();
    await fetchMealCalories();
    await updateFoodSummary();
    await fetchAddedFood();
    await loadGoalInfo();
    await updateNutritionQuality();
```

Рисунок 3.32 – Вибір дати та оновлення даних (dashboard.js)

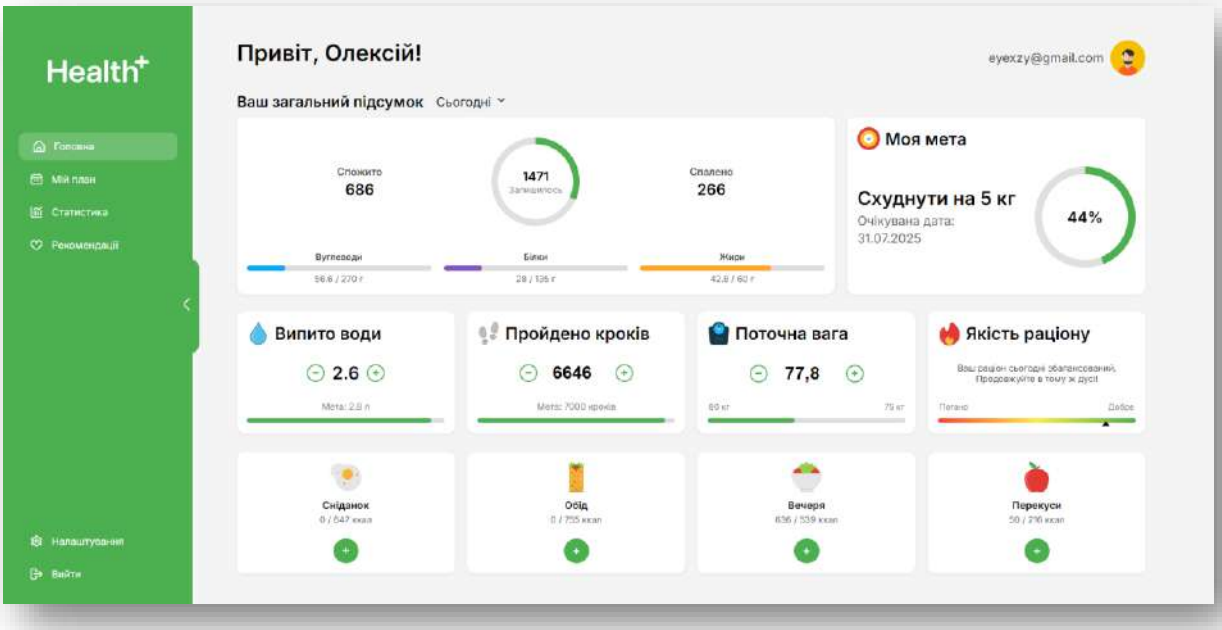

Рисунок 3.33 – головна сторінка дашборду під час використання

3.3.4. Сторінка статистики

Побудова статистики за водою, кроками, вагою, калоріями, якістю раціону та макросами.

На дашборді реалізовано розділ “Статистика”, де користувач може переглядати динаміку своїх показників за останній тиждень або місяць. Для цього використовуються лінійні графіки (Chart.js), які будуються на основі даних з колекції progresses.

При зміні режиму перегляду (week / month), запит надсилається до API /api/statistics, який повертає зведення по кожному дню.

Наприклад, побудова графіка води за тиждень виконується функцією createAreaChart, яка використовує дані з backend та виводить їх на канвас. (див. рис. 3.34)

```
.then((res) => res.json())
.then((data) => {
    createAreaChart("waterChart", "Вода", data.water, "#2196F3", statsMode);
    createAreaChart("stepsChart", "Кроки", data.steps, "#4CAF50", statsMode);
    createAreaChart("weightChart", "Вага", data.weight, "#9C27B0", statsMode);
    createAreaChart("caloriesChart", "Калорії", data.calories, "#FF9800", statsMode);
    createAreaChart("nutritionQualityChart", "Якість раціону", data.nutritionQuality, "#E91E63", statsMode);
```

Рисунок 3.34 – Побудова графіка споживання води (dashboard.js)

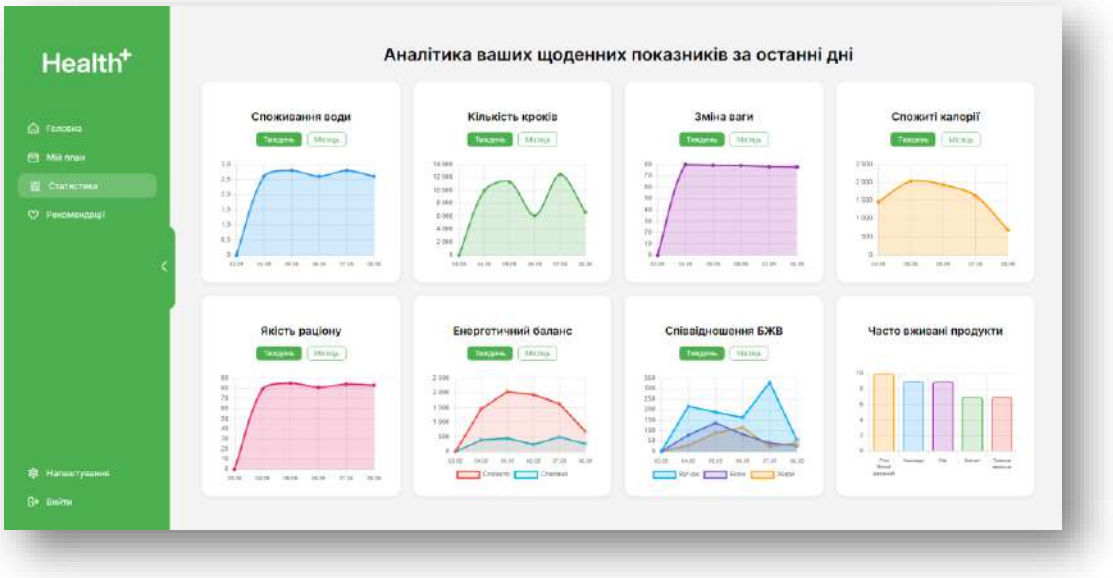

Рисунок 3.35 – статистика користувача під час прогресу

ВИСНОВКИ

У ході виконання курсової роботи був створений інтерактивний вебзастосунок для моніторингу та управління здоровим способом життя, що охоплює всі основні аспекти контролю показників здоров’я. Було сформульовано та виконано ключові завдання: досліджено предметну область, визначено вимоги до функціоналу, розроблено серверну та клієнтську частини застосунку, а також було проведене наглядне тестування користуванням застосунку протягом тижня.

Головними досягненнями є успішна реалізація реєстрації та авторизації користувачів, створення персоналізованого плану на основі введених даних та розрахунку особистих параметрів здоров’я. Також впроваджено щоденний трекінг активності, що дозволяє користувачу вносити дані про прийоми їжі, кількість випитої води, пройдені кроки та поточну вагу.

Практична цінність розробленого рішення полягає у можливості моніторингу змін стану здоров’я. Завдяки зручному дашборду, користувач отримує доступ до статистики, що дозволяє оцінювати ефективність своїх дій та коригувати поведінку відповідно до поставлених цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Всесвітній день боротьби з ожирінням: чому важливо дбати про здорову масу тіла | Центр громадського здоров'я. *Центр громадського здоров'я України | МОЗ*. URL: https://phc.org.ua/news/vsesvitniy-den-borotbi-z-ozhirinnyam-chomu-vazhlivo-dbati-pro-zdorovu-masu-tila.

2. Calorie tracker & BMR calculator to reach your goals | MyFitnessPal. URL: https://www.myfitnesspal.com/#howItWorks.

3. Web Development. *Geek for geeks*. 2023. URL: https://www.geeksforgeeks.org/web-development/.

4. Клієнт-серверна архітектура. *QA Test Lab*. 2020. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/.

5. HTTP vs. HTTPS: What are the differences? *Cloudflare*. URL: https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/.

6. HTTP request methods - HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods.

7. SPA vs MPA: which one is better for you? *GeeksforGeeks*. 2024. URL: https://www.geeksforgeeks.org/spa-vs-mpa-which-one-is-better-for-you/.

8. HTML Introduction. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/html-introduction/.

9. JavaScript tutorial. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/javascript/.

10. Chart.js. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/chart-js-tutorial/.

11. Mifflin-St Jeor Equation. *Medscape*. URL: https://reference.medscape.com/calculator/846/mifflin-st-jeor-equation.