МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ТА АНАЛІТИКИ ФІЗИЧНОЇ АКТИВНОСТІ КОРИСТУВАЧА»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.Ю. Демченко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Демченку Артему Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для відстеження та аналітики фізичної активності користувача

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та існуючих рішень.
2. Проєктування вебзастосунку.
3. Реалізація та тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	12.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	17.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	15.06.2026	

Студент _____ А.Ю. Демченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: «Розробка вебзастосунку для відстеження та аналітики фізичної активності користувача»: 75 с., 100 рис., 1 табл., 38 джерел.

АНАЛІТИКА ДАНИХ, БІБЛІОТЕКА, ВЕБЗАСТОСУНОК, ІНДЕКС МАСИ ТІЛА, ФІЗИЧНА АКТИВНІСТЬ, CLIENT-SERVER АРХІТЕКТУРА, NODE.JS, REACT, SQLITE.

Об’єкт дослідження – процес розробки вебзастосунку.

Предмет дослідження – бібліотека React, серверна платформа Node.js, засоби взаємодії клієнтської та серверної частин вебзастосунку, методи збереження, обробки й візуалізації даних фізичної активності користувачів.

Мета роботи – розробка вебзастосунку для відстеження та аналітики фізичної активності користувача засобами React, Node.js, Express та SQLite.

Методи дослідження – аналіз предметної області, проєктування клієнт-серверної архітектури вебзастосунку, модульне програмування, конструювання інтерфейсу користувача і тестування програмного забезпечення.

При розробці додатку був проведений аналіз предметної області, спроєктовано клієнтську та серверну архітектуру вебзастосунку для відстеження й аналізу фізичної активності користувачів. Клієнтську частину реалізовано з використанням React, серверну – на базі Node.js та Express, для збереження даних використано SQLite. У системі реалізовано функціонал авторизації користувачів, обліку фізичної активності, побудови статистики та розрахунку індексу маси тіла.

У результаті роботи створено функціональний вебзастосунок для моніторингу та аналізу фізичної активності користувачів.

SUMMARY

Bachelor's qualifying paper "Development of a Web Application for Tracking and Analyzing User Physical Activity": 75 pages, 100 figures, 1 table, 38 references.

DATA ANALYTICS, LIBRARY, WEB APPLICATION, BODY MASS INDEX, PHYSICAL ACTIVITY, CLIENT-SERVER ARCHITECTURE, NODE.JS, REACT, SQLITE.

The object of research is the process of developing a web application.

The subject of research is the React library, the Node.js server platform, means of interaction between the client and server parts of the web application, methods of storing, processing and visualizing user physical activity data.

The purpose of the work is to develop a web application for tracking and analytics of user physical activity using React, Node.js, Express and SQLite.

Research methods are domain analysis, designing the client-server architecture of the web application, modular programming, designing the user interface and testing the software.

When developing the application, an analysis of the subject area was conducted, the client and server architecture of the web application was designed to track and analyze the physical activity of users. The client part was implemented using React, the server part was based on Node.js and Express, SQLite was used to store data. The system implemented the functionality of user authorization, accounting for physical activity, building statistics and calculating the body mass index.

As a result of the work, a functional web application was created for monitoring and analyzing the physical activity of users.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та існуючих рішень..... 10
1.1 Аналіз предметної області фізичної активності користувача 10
1.2 Аналіз існуючих аналогів та програмних рішень у сфері відстеження фізичної активності..... 12
2 Проєктування вебзастосунку для відстеження та аналітики фізичної активності користувача..... 15
2.1 Архітектура та технологічне забезпечення вебзастосунку 15
2.2 Проєктування функціональної структури вебзастосунку..... 16
2.3 Проєктування бази даних та програмної моделі системи 19
3 Реалізація та тестування вебзастосунку для відстеження та аналітики фізичної активності користувача..... 23
3.1 Реалізація клієнтської частини вебзастосунку..... 23
3.2 Реалізація серверної частини вебзастосунку..... 40
3.3 Реалізація авторизації 52
3.4 Реалізація аналітики та графіків..... 54
3.5 Тестування вебзастосунку..... 57
Висновки 70
Перелік посилань..... 71

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій особливого значення набувають цифрові рішення, які спрямовані на контроль фізичної активності людини та підтримку здорового способу життя. Активне використання мережі Інтернет, мобільних пристроїв та вебтехнологій сприяє зростанню популярності систем, які дозволяють користувачам отримувати доступ до персональної інформації, аналізувати власні показники та вести облік щоденної діяльності [6]. Однією з важливих сфер застосування таких технологій є моніторинг фізичної активності, оскільки фізичний стан людини безпосередньо впливає на її працездатність, самопочуття та якість життя.

Сьогодні значна кількість людей веде малорухливий спосіб життя, що негативно позначається на загальному стані здоров’я. Недостатня фізична активність може призводити до погіршення загальної фізичної форми, зниження працездатності та ризику розвитку захворювань [25]. Тому актуальним є створення інформаційних систем, які дозволяють користувачам самостійно контролювати власну активність, вести статистику проведених тренувань та аналізувати отримані результати. Швидкий доступ до інформації, зручність використання та можливість роботи з системою на різних типах пристроїв забезпечує використання вебзастосунків у даній сфері.

Вебзастосунки є одним із найбільш популярних видів програмного забезпечення завдяки своїй універсальності та доступності, не потребують складного встановлення та можуть використовуватися через браузер незалежно від типу пристрою чи операційної системи. Сучасні вебтехнології дозволяють створювати інтерактивні та функціональні системи, що забезпечують високу швидкодію, зручний інтерфейс користувача та можливість обробки значних обсягів даних, а також можуть поєднувати функції збереження інформації, аналітики та візуалізації даних, що є важливим для систем контролю фізичної активності [17, 18, 21].

Актуальність теми полягає у необхідності створення сучасного вебзастосунку, який дозволить користувачам відстежувати власну фізичну активність, аналізувати статистичні показники та отримувати зручний інструмент для контролю результатів [17, 23].

Метою даної кваліфікаційної роботи є розробка вебзастосунку для відстеження та аналітики фізичної активності користувача з можливістю збереження даних, побудови статистики, візуалізації показників та авторизації користувачів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз предметної області та існуючих програмних рішень;
– обрати сучасні засоби та технології розробки;
– реалізувати систему реєстрації та авторизації користувачів;
– реалізувати функціонал додавання, редагування та видалення активностей;
– забезпечити можливість пошуку даних;
– реалізувати модулі аналітики та графічної візуалізації статистики;
– реалізувати функціонал для розрахунку індексу маси тіла;
– провести тестування та оцінити працездатність розробленого програмного продукту.

Об’єкт дослідження – процеси розробки вебзастосунку.

Предмет дослідження – бібліотека React, серверна платформа Node.js, засоби взаємодії клієнтської і серверної частин вебзастосунку, а також методи збереження, обробки, візуалізації даних фізичної активності користувачів.

Методи дослідження – аналіз предметної області, проєктування клієнт-серверної архітектури вебзастосунку, модульне програмування, конструювання інтерфейсу користувача і тестування програмного забезпечення.

Практичне значення роботи полягає у створенні функціонального вебзастосунку, який забезпечує збереження даних про види фізичної активності, обчислення показників, перегляд статистики та візуальне

представлення результатів у вигляді графіків. Крім того, до системи включено модуль розрахунку індексу маси тіла, що дозволяє користувачу отримати додаткову інформацію про свій фізичний стан.

У процесі розробки вебзастосунку особлива увага приділялася питанням зручності використання, швидкості дії та безпеки. Для реалізації системи використано сучасний підхід клієнт-серверної архітектури, який забезпечує розподіл логіки між серверною та клієнтською частинами [8, 9]. Інтерфейс користувача реалізовано з урахуванням принципів сучасного вебдизайну, що забезпечує простоту навігації та комфортне використання системи. Також важливим аспектом є забезпечення захисту даних користувачів шляхом використання механізмів авторизації та шифрування паролів [24].

Під час розробки вебзастосунку було використано сучасні вебтехнології для створення серверної та клієнтської частин системи, організації взаємодії з базою даних та реалізації аналітичних функцій, що дало змогу отримати програмний продукт, який має зручний інтерфейс, підтримує роботу з даними користувача і забезпечує наочне представлення статистичної інформації.

Таким чином, розробка вебзастосунку для відстеження та аналітики фізичної активності користувача є актуальним напрямом у сфері сучасних інформаційних технологій. Створений вебзастосунок поєднує функції обліку фізичних активностей, аналізу даних та візуалізації отриманих результатів, що дозволяє підвищити ефективність контролю за фізичною активністю користувачів і сприяє популяризації здорового способу життя.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз предметної області фізичної активності користувача

У сучасному суспільстві інформаційні технології активно впроваджуються у різні сфери діяльності людини, зокрема у сферу здоров’я. У теперішній час значна частина населення проводить багато часу за комп’ютерами та мобільними пристроями, що призводить до зменшення рівня рухової активності. Малорухливий спосіб життя негативно впливає на фізичний стан людини та може бути причиною розвитку серцево-судинних захворювань, ожиріння та інших проблем зі здоров’ям [25].

Фізична активність являє собою сукупність рухових дій людини, до яких відносяться як повсякденні дії, наприклад ходьба або підйом сходами, так і спеціалізовані тренування, заняття спортом або фітнесом. Для оцінки рівня фізичної активності використовуються різні показники, серед яких тривалість тренування, кількість витрачених калорій, тип фізичної активності, інтенсивність навантаження та інші параметри.

Підтримання фізичної активності є важливим фактором збереження здоров’я, працездатності та покращення якості життя людини, тому створення сучасних програмних засобів для контролю активності є актуальним напрямом розвитку інформаційних систем [36].

Сучасні інформаційні системи дозволяють автоматизувати процес збору та обробки даних про фізичну активність користувачів [27]. Для цього використовуються вебзастосунки, мобільні додатки, фітнес-платформи та різноманітні цифрові сервіси. Важливою перевагою вебзастосунків є можливість доступу до системи через браузер без необхідності встановлення додаткового програмного забезпечення, що робить їх універсальними та доступними для широкого кола користувачів незалежно від типу пристрою чи операційної системи [30, 38].

Однією з основних функцій систем відстеження фізичної активності є збереження інформації про активності користувача. Користувач повинен мати можливість редагувати існуючі дані, видаляти непотрібну інформацію та переглядати статистику своєї активності, а також додавати нові записи. Такий підхід дозволяє забезпечити повноцінний контроль над персональними даними та створює зручний механізм ведення історії фізичної активності.

Важливим елементом сучасних вебзастосунків є аналітика даних, оскільки просте накопичення інформації не забезпечує достатньої ефективності без можливості її аналізу та наступного візуального представлення. Тому сучасні системи повинні містити засоби обробки статистичних даних, побудови графіків та відображення основних показників фізичної активності користувача. Візуалізація даних дозволяє значно спростити сприйняття інформації та допомагає користувачу оцінити власний прогрес в динаміці занять [12]. У сучасних вебзастосунках для реалізації графічного представлення даних використовуються спеціалізовані бібліотеки для побудови діаграм та графіків, зокрема Recharts [30].

Практичну цінність системи підвищують додаткові функції: пошук фізичних активностей, фільтрація даних, обчислення статистичних показників, а також розрахунок індексу маси тіла користувача, що дозволяє зробити вебзастосунок більш функціональним та зручним у використанні.

Забезпечення безпеки персональних даних користувачів має суттєве значення, оскільки система працює з обліковими записами та персональною інформацією. Необхідно реалізувати механізми авторизації та автентифікації користувачів, а також захист паролів та доступу до даних [4, 33]. З цією метою у сучасних вебзастосунках використовуються JWT-токени та інші механізми захисту даних [4, 15, 16].

Під час розробки вебзастосунків важливу роль відіграє також зручність інтерфейсу користувача, який повинен бути зрозумілим, логічно структурованим та адаптивним до різних типів пристроїв. Зручна навігація та сучасний дизайн значно підвищують комфорт користування системою та покращують взаємодію користувача із застосунком [14].

Крім того, важливим аспектом сучасних вебсистем є швидкодія та ефективна взаємодія між клієнтською та серверною частинами. Для цього активно використовуються клієнт-серверна архітектура та REST API, які забезпечують стандартизований обмін даними у форматі JSON [13, 31, 32], що дозволяє забезпечити гнучкість системи, спрощує її масштабування та покращує продуктивність вебзастосунку.

Таким чином, аналіз предметної області показує, що існує потреба у створенні сучасного вебзастосунку для відстеження та аналітики фізичної активності користувача, який поєднуватиме функції збору, збереження, обробки та візуалізації даних, повинен забезпечувати зручний інтерфейс, реалізовувати механізми авторизації користувачів, підтримувати аналітичні функції та надавати можливість ефективного контролю фізичної активності.

1.2 Аналіз існуючих аналогів та програмних рішень у сфері відстеження фізичної активності

На сьогодні існує велика кількість програмних рішень для відстеження фізичної активності користувачів, що представлені мобільними додатками, вебзастосунками та комплексними платформами, які дозволяють користувачам контролювати власну фізичну активність, аналізувати статистику та зберігати інформацію про тренування. Проведення аналізу існуючих аналогів є важливим етапом розробки власного програмного продукту, оскільки дозволяє визначити основні функціональні можливості сучасних систем, їх переваги та недоліки [17, 18].

Серед найбільш популярних рішень у даній сфері можна виділити Google Fit, Samsung Health та Apple Health [5]. Дані системи забезпечують автоматичний збір інформації про фізичну активність користувача за допомогою сенсорів мобільних пристроїв та фітнес-гаджетів [27]. Основними їх функціями є підрахунок кроків, визначення витрачених калорій, облік тренувань, формування статистики та візуалізація результатів.

Google Fit є популярною платформою для моніторингу фізичної активності, яка підтримує синхронізацію з різними мобільними пристроями та фітнес-сервісами. Система дозволяє збирати статистику фізичної активності користувача, аналізувати показники та формувати рекомендації щодо підтримання активного способу життя.

Samsung Health – багатофункціональна система контролю фізичного стану користувача, яка підтримує облік тренувань, контроль сну, моніторинг серцевого ритму та інші функції, що пов’язані зі здоров’ям користувача.

Apple Health забезпечує інтеграцію з екосистемою пристроїв Apple та дозволяє централізовано зберігати інформацію про фізичну активність і медичні показники користувача.

Перевага даних систем – це високий рівень автоматизації збору даних та інтеграція з мобільними пристроями. Користувачеві не потрібно вручну вводити більшість показників, оскільки інформація збирається автоматично. Сервіси мають сучасний інтерфейс та підтримують графічне представлення статистичних даних.

Разом із тим, існуючі рішення мають і певні недоліки. Багато систем є прив’язаними до конкретної екосистеми або операційної системи, що обмежує можливість їх використання на різних пристроях; застосунки містять складний або перевантажений інтерфейс, що може ускладнювати взаємодію користувача із системою; окремі платформи мають обмежений функціонал безкоштовної версії або потребують підключення додаткових сервісів.

Окрім мобільних додатків, існують вебзастосунки для відстеження фізичної активності, які забезпечують доступ до системи через браузер. Основною перевагою їх є універсальність та незалежність від конкретного пристрою. Користувач може працювати із системою через персональний комп’ютер, планшет або смартфон без необхідності встановлення окремого програмного забезпечення.

Для створення сучасних вебзастосунків активно використовуються JavaScript-бібліотеки та серверні технології. Однією з найбільш популярних

бібліотек для створення інтерфейсу користувача є React, яка дозволяє реалізовувати компонентний підхід до побудови вебінтерфейсу [1, 28, 30, 35]. Серверна частина часто реалізовується за допомогою Node.js та Express.js, що дозволяє забезпечити швидку обробку HTTP-запитів та реалізацію REST API [11, 26, 32]. Основні принципи роботи HTTP, включаючи статус-коди та методи запитів, описані у документації MDN Web Docs [20].

Важливим аспектом сучасних програмних рішень є наявність функцій аналітики та візуалізації інформації, що дозволяє відображати дані у вигляді графіків, статистичних таблиць та діаграм, і дає змогу користувачеві аналізувати власну активність, оцінювати зміни показників та відстежувати прогрес за певний період часу [12].

Для зберігання інформації у вебзастосунках можуть використовуватися різні системи управління базами даних. У межах даної роботи використовується SQLite – легка реляційна система управління базами даних, яка не потребує окремого серверного середовища та є зручною для невеликих вебпроєктів [34].

На основі проведеного аналізу можна зробити висновок, що сучасні системи відстеження фізичної активності мають широкий функціонал, проте не завжди забезпечують оптимальне поєднання простоти використання, аналітики та доступності. Саме тому доцільною є розробка власного вебзастосунку, який забезпечуватиме зручний інтерфейс користувача, функції додавання, редагування та видалення активностей, пошук даних, аналітику та графічну візуалізацію статистики.

Отримані результати аналізу предметної області та існуючих аналогів дозволяють сформувати основні вимоги до майбутнього вебзастосунку та визначити ключові напрями його реалізації.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Архітектура та технологічне забезпечення вебзастосунку

Архітектура вебзастосунку для відстеження та аналітики фізичної активності користувача побудована за принципом клієнт-серверної взаємодії яка забезпечує збір, обробку, збереження, візуалізацію даних про рухову активність і включає frontend-частину, backend-частину і систему збереження даних [7].

Під час розробки вебзастосунку було здійснено аналіз сучасних технологічних рішень для реалізації фронтенд- та бекенд-частини, а також підходів до організації роботи з базами даних. Вибір технологічного стеку визначався з урахуванням складності системи, швидкості розробки, вимог до продуктивності та можливостей подальшого масштабування.

Для реалізації клієнтської частини було обрано бібліотеку React як основний інструмент побудови інтерфейсу користувача та забезпечує оптимальний баланс між гнучкістю, продуктивністю та можливістю масштабування. Компонентний підхід, розвинена екосистема бібліотек і висока підтримка спільноти дозволяють ефективно реалізовувати інтерфейси будь-якої складності та забезпечують зручність подальшого супроводу й розвитку проєкту.

Angular є потужним повноцінним фреймворком із широким набором вбудованих можливостей. Водночас він характеризується більш складною архітектурою та значним порогом входу, що може бути надлишковим для проєктів середнього масштабу. Vue.js відзначається простотою освоєння та швидким стартом розробки, однак його екосистема та рівень поширення в масштабних комерційних рішеннях є дещо меншими порівняно з React.

Клієнтська частина створена за допомогою бібліотеки React, серверна частина – на платформі Node.js із використанням фреймворку Express.js, а для

збереження даних використовується база даних SQLite [22, 29, 34]. Взаємодія між компонентами здійснюється через HTTP-запити до REST API. Такий підхід забезпечує розподіл функціональності між компонентами системи, підвищує масштабованість, зручність підтримки та ефективність роботи застосунку.

2.2 Проєктування функціональної структури вебзастосунку

Frontend-частина вебзастосунку відповідає за взаємодію користувача із системою, відображення інформації та забезпечення інтерфейсу користувача. Клієнтська частина реалізована на JavaScript із використанням бібліотеки React та функціональних компонентів. Основними компонентами frontend-частини є Login, Register, Dashboard, LineChart, AddActivity, BMICalculator.

Компонент Login.js реалізує функціонал авторизації користувача. Користувач вводить email та пароль, після чого дані передаються на сервер через API-запит. При успішній авторизації сервер повертає JWT-токен, який зберігається у localStorage для подальшої автентифікації [15]. Функції компонента Login – це введення облікових даних, перевірення заповнення полів, відправлення запиту авторизації, оброблення повідомлень про помилки, збереження токена доступу.

Компонент Register.js забезпечує реєстрацію нового користувача і основними його функціями є введення імені, email та пароля, перевірка коректності введених даних, надсилання даних на сервер, повідомлення про успішну реєстрацію. Компонент Dashboard.js є центральною частиною вебзастосунку та відповідає за отримання списку фізичних активностей, фільтрацію активностей, виведення статистики, редагування та видалення записів, побудову графіків з реалізацією відображення загальної кількості витрачених калорій, обчислення загальної тривалості тренувань, пошук активностей за назвою, побудову графіка калорій за допомогою бібліотеки

Recharts [30]. Для візуалізації аналітичних даних використовується компонент LineChart, що дозволяє відображати зміну витрачених калорій залежно від дати активності. Компонент AddActivity.js призначений для додавання нової фізичної активності користувача. Користувач може ввести тип активності, тривалість, кількість витрачених калорій та дату виконання. Після надсилання форми інформація передається до backend-частини через API.

Компонент BMICalculator.js реалізує калькулятор індексу маси тіла (body mass index (BMI)). Функціонал компонента базується на введенні основних антропометричних даних, зокрема маси тіла в кілограмах (m) та зросту користувача у сантиметрах (h), на основі яких автоматично обчислюється значення BMI. Після цього система автоматично виконує розрахунок BMI за формулою:

$$BMI = \frac{m}{h^2},$$

де m – маса тіла, кг; h – зріст, см.

Інтерпретацію отриманого результату визначено відповідної категорії фізичного стану користувача згідно з загальноприйнятими медичними критеріями: недостатня маса тіла – при значенні менше 18,5; нормальна маса тіла – 18,5 – 24,9; надлишкова вага – 25,0 – 29,0; ожиріння – 30 і більше [37].

Backend відповідає за обробку HTTP-запитів, реалізацію бізнес-логіки, взаємодію з базою даних, авторизацію користувачів, захист даних. Механізмом є JWT (JSON Web Token) дозволяє здійснювати перевірку користувача без зберігання сесій на сервері [15]. Такий підхід підвищує масштабованість системи та зменшує навантаження на сервер. Паролі користувачів зберігаються у захешованому вигляді, що відповідає вимогам інформаційної безпеки і мінімізує ризики несанкціонованого доступу. Додатково перевірка токена виконується на рівні кожного захищеного запиту, що забезпечує контроль доступу до персональних даних користувача.

Ключовим елементом серверної частини вебзастосунку є модуль server.js, у якому здійснюється налаштування серверного середовища, підключення middleware-компонентів, конфігурація API-маршрутів та запуск сервера на порту 5000. Для забезпечення модульності та спрощення супроводу програмного коду функціональність системи розподілена між окремими модулями. Зокрема, модуль auth.js відповідає за реєстрацію та автентифікацію користувачів, включаючи перевірку введених даних, хешування паролів і формування JWT-токенів доступу. Модуль activities.js реалізує CRUD-операції для управління даними фізичної активності користувача. Така організація серверної частини забезпечує зрозумілу структуру проєкту, підвищує читабельність програмного коду та спрощує подальше розширення функціональних можливостей системи.

Функціональну структуру вебзастосунку відображає діаграма варіантів використання (див. рис. 2.1) та описує взаємодію між користувачем і системою.

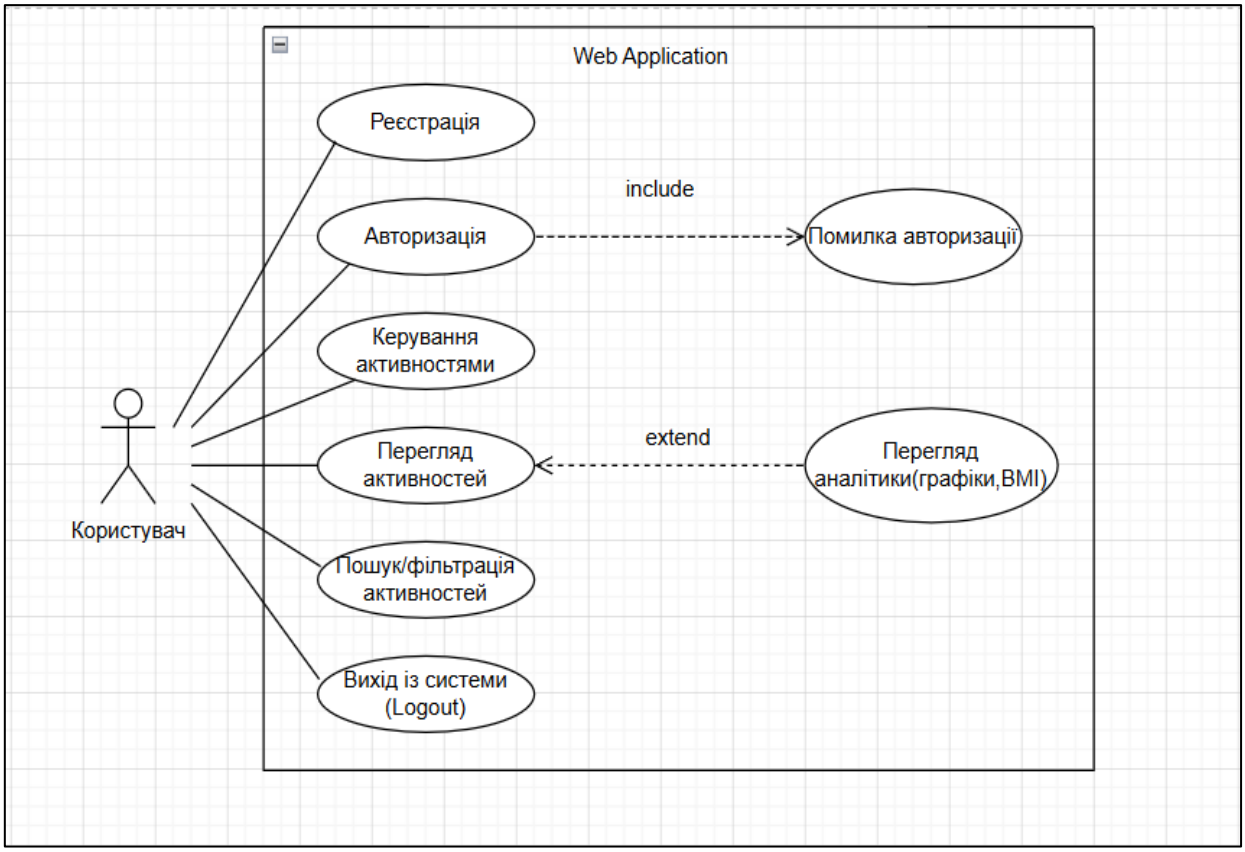


Рисунок 2.1 – Діаграма варіантів використання

Основним актором у системі є користувач, який після проходження процедури авторизації отримує доступ до всього передбаченого функціоналу. До основних сценаріїв використання належать реєстрація та вхід у систему, створення нових записів щодо фізичної активності, перегляд внесених даних, їх редагування та видалення, а також отримання аналітичної інформації у вигляді статистичних показників і графічних представлень.

Кожен із визначених варіантів використання відповідає окремій функціональній можливості системи та відображає логіку роботи з даними користувача. У сукупності вони формують повний цикл взаємодії користувача з вебзастосунком – від введення інформації до її аналізу.

Алгоритм роботи системи має наступний вигляд:

– користувач проходить реєстрацію або авторизацію;
– сервер перевіряє дані та повертає JWT-токен;
– frontend зберігає токен у localStorage;
– користувач додає фізичну активність;
– дані передаються через API до backend;
– backend зберігає інформацію у SQLite;
– frontend отримує оновлений список активностей;
– система формує статистику та графіки.

2.3 Проєктування бази даних та програмної моделі системи

Під час вибору системи керування базами даних було проаналізовано сучасні рішення з урахуванням вимог проєкту. PostgreSQL забезпечує високу продуктивність, підтримку складних SQL-запитів, багатокористувацький режим роботи та можливість масштабування, проте потребує окремого серверного середовища й додаткового адміністрування. На відміну від неї, SQLite є вбудованою системою керування базами даних, яка не потребує встановлення сервера та зберігає дані у локальному файлі, що спрощує процес розробки, тестування та використання [34].

У межах даної роботи обрано SQLite, оскільки вона повністю забезпечує потреби вебзастосунку щодо зберігання та обробки даних користувачів, а також спрощує розгортання і подальший супровід системи.

Для формалізації структури даних вебзастосунку визначено основні типи даних, які використовуються для збереження інформації про користувачів та їх фізичну активність і наведені у таблиці 2.1.

Таблиця 2.1 – Основні типи даних вебзастосунку

Сутність / модуль	**Поле / дані**	**Опис**
Користувач (User)	id	Унікальний ідентифікатор користувача
	name	Ім’я користувача
	email	Електронна пошта для авторизації
	password	Захешований пароль
Фізична активність	id	Унікальний ідентифікатор запису активності
	user_id	Зв’язок із користувачем
	type	Тип активності (біг, ходьба, тренування)
	duration	Тривалість виконання (у хвилинах)
	calories	Витрачені калорії
	date	Дата виконання активності
Аналітика (BMI)	weight	Вага користувача
	height	Зріст користувача
	bmi_result	Розрахований індекс маси тіла
Графіки	dataset	Набір даних для побудови графіків активності

Структурну модель системи відображає діаграма класів, яка визначає основні об’єкти предметної області, що використовуються для зберігання та опрацювання даних у вебзастосунку (див. рис. 2.2).

Ключовими сутностями є класи User та Activity. Клас User містить інформацію про користувача системи, зокрема унікальний ідентифікатор, ім’я, адресу електронної пошти та пароль, які зберігається у захешованому вигляді. Окрім цього, передбачено функціональність, пов’язану з реєстрацією, автентифікацією та отриманням даних профілю.

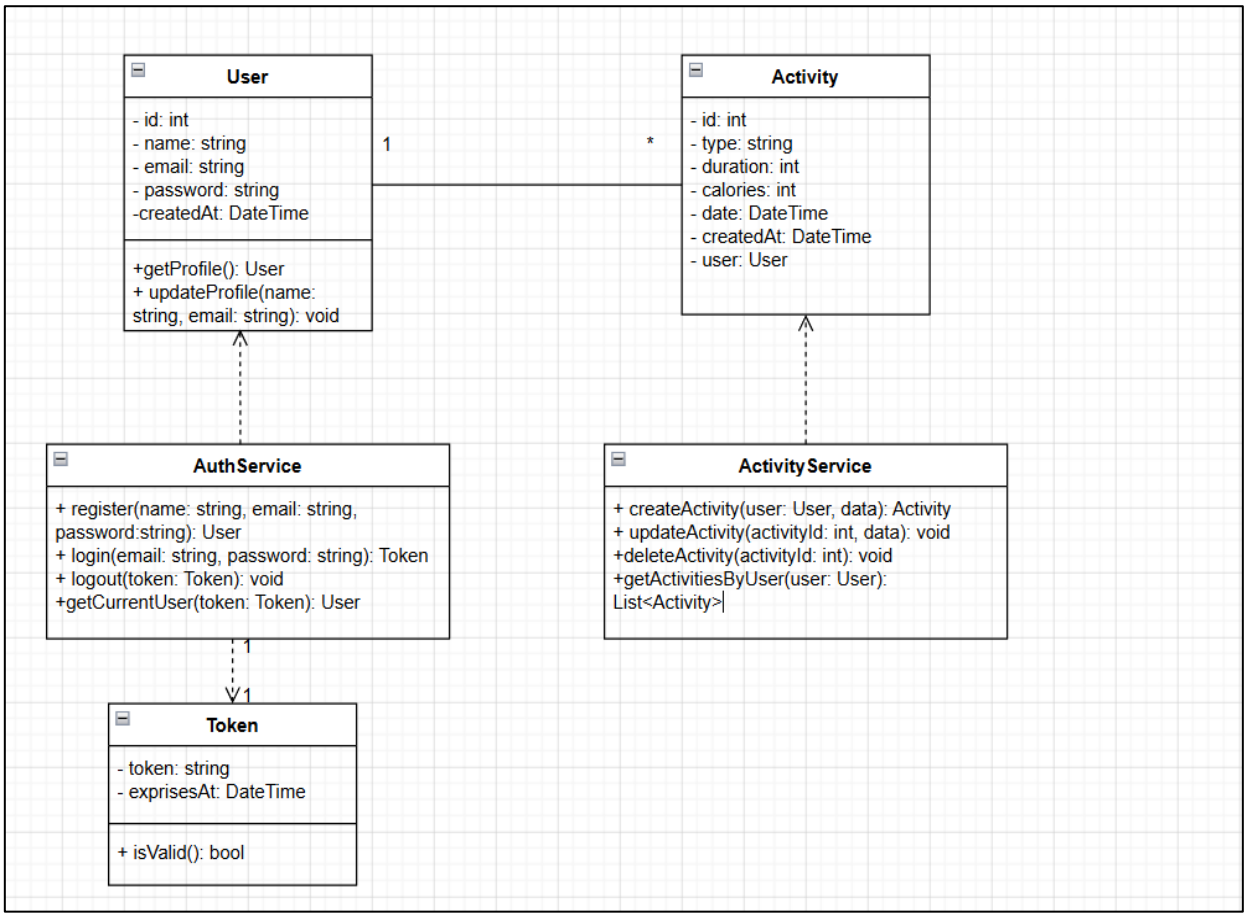


Рисунок 2.2 – Діаграма класів

Клас Activity моделює запис фізичної активності і включає наступні характеристики: тип активності, її тривалість, кількість витрачених калорій і дата виконання. Реалізовано набір операцій для створення, перегляду, редагування та видалення відповідних записів.

Між класами User та Activity встановлено асоціацію типу «один-до-багатьох», що відображає ситуацію, коли один користувач може створювати та зберігати декілька записів фізичної активності. Така структура відповідає логіці предметної області та забезпечує коректну організацію даних у системі.

У серверній частині вебзастосунку реалізовано моделі даних, які відповідають за збереження та обробку основної інформації системи – користувачів та їх фізичної активності. Модель User, реалізована у файлі user.js, що відповідає за роботу з даними користувачів системи, і забезпечує базові операції, які пов’язані з обліковими записами. Основні функції моделі: створення нового користувача в базі даних із попереднім хешуванням пароля

за допомогою bcrypt [2, 3, 10]; пошук користувача за адресою електронної пошти (email) для реалізації процесу авторизації та зберігання захищених облікових даних користувача.

Модель Activity, реалізована у файлі activity.js, відповідає за оброблення даних фізичної активності користувача. Основні функції моделі: створення нового запису про фізичну активність (тип, тривалість, калорії, дата); отримання списку всіх активностей конкретного користувача; оновлення вже існуючих записів активності; видалення активностей із бази даних.

Таким чином, архітектура розробленого вебзастосунку базується на сучасній клієнт-серверній моделі та забезпечує ефективну взаємодію між користувачем, сервером та базою даних. Використання React, Node.js, Express.js та SQLite дозволило реалізувати зручний, функціональний вебзастосунок для відстеження та аналітики фізичної активності користувача [1, 22, 29, 34].

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

У даному розділі розглядається процес практичної реалізації вебзастосунку для відстеження та аналізу фізичної активності користувача, описуються основні програмні модулі системи, особливості реалізації клієнтської та серверної частини, а також наведено результати тестування розробленого програмного продукту.

3.1 Реалізація клієнтської частини вебзастосунку

Клієнтська частина вебзастосунку відповідає за забезпечення взаємодії користувача із системою, відображення інтерфейсу та реалізацію основних функціональних можливостей вебзастосунку. Її основним завданням є забезпечення зручного, зрозумілого та швидкого доступу до функціоналу вебзастосунку з коректним відображенням інформації, отриманої із серверної частини системи.

Для реалізації клієнтської частини було використано бібліотеку React, яка дозволяє реалізувати компонентний підхід до побудови інтерфейсу користувача, при якому кожен елемент системи представляється окремим незалежним компонентом, що значно спрощує структуру програмного коду, покращує його читабельність, забезпечує повторне використання компонентів та спрощує подальшу підтримку і масштабування проєкту.

Однією з ключових переваг React є можливість динамічного оновлення інтерфейсу без повного перезавантаження сторінки, що дозволяє забезпечити більш швидку роботу вебзастосунку та покращити користувацький досвід. У процесі взаємодії користувача із системою оновлюються лише необхідні елементи сторінки, що зменшує навантаження на браузер та підвищує загальну продуктивність застосунку.

Структура клієнтської частини вебзастосунку побудована за модульним принципом. Кожен функціональний елемент реалізований у вигляді окремого компонента, який відповідає за виконання конкретної задачі.

До основних компонентів клієнтської частини належать:

– модуль авторизації та реєстрації користувача;
– головна панель керування (Dashboard);
– форма додавання фізичної активності;
– модуль перегляду статистики;
– система пошуку та фільтрації записів;
– модуль редагування та видалення активностей;
– графічна візуалізація статистичних даних;
– калькулятор індексу маси тіла (BMI).

Модуль авторизації забезпечує можливість входу користувача до системи та створення нового облікового запису. Для цього реалізовано форми введення електронної пошти та пароля, а також механізм перевірки коректності введених даних. Після успішної авторизації користувач отримує доступ до персонального функціоналу системи.

Головна панель керування є центральним елементом інтерфейсу користувача. На даній сторінці відображається список фізичних активностей користувача, статистична інформація, а також елементи керування записами. Саме через Dashboard користувач здійснює основну взаємодію із системою.

Особлива увага приділялася реалізації модуля додавання фізичної активності для подальшої аналітики даних. Користувач має можливість ввести тип фізичної активності, її тривалість, кількість витрачених калорій та дату виконання. Після підтвердження введені дані передаються на серверну частину за допомогою HTTP-запиту та зберігаються у базі даних.

Крім створення записів, у системі реалізовано можливість їх редагування та видалення, що дозволяє користувачу коригувати інформацію при помилковому введенні даних або оновленні показників фізичної активності.

Для підвищення зручності роботи з великою кількістю записів реалізовано механізм пошуку даних. Користувач може швидко знаходити необхідні записи за типом активності або іншими параметрами, що значно спрощує навігацію та роботу з інформацією.

Важливою складовою клієнтської частини є модуль аналітики та візуалізації даних. Графічне представлення інформації забезпечує більш наочне відображення результатів фізичної активності користувача та спрощує аналіз динаміки змін показників. Для побудови графіків використовується бібліотека Recharts, яка дозволяє створювати інтерактивні діаграми та графічні елементи статистики.

Окремо реалізовано BMI-калькулятор, який дозволяє користувачу розрахувати індекс маси тіла на основі введених параметрів зросту та ваги. Результат розрахунку відображається безпосередньо в інтерфейсі застосунку та дозволяє оцінити загальний фізичний стан користувача.

Інтерфейс вебзастосунку реалізований із дотриманням принципів мінімалізму та зручної навігації. Використання сучасного підходу до побудови інтерфейсу дозволяє забезпечити комфортну взаємодію користувача із системою навіть при тривалому використанні вебзастосунку.

На рисунку 3.1 представлена демонстрація коду з ініціалізацією локального стану компонента AddActivity за допомогою React Hook useState. У межах компонента створюються змінні стану для збереження введених користувачем даних: типу фізичної активності, тривалості, кількості витрачених калорій та дати виконання. Використання окремих станів для кожного поля дозволяє забезпечити контрольоване введення даних (controlled inputs) та оперативне оновлення значень у процесі взаємодії користувача з формою.

```
10    const [type, setType] = useState('');
11    const [duration, setDuration] = useState('');
12    const [calories, setCalories] = useState('');
13    const [date, setDate] = useState('');
```

Рисунок 3.1 – Оголошення стану компонента AddActivity (useState)

Фрагмент коду з реалізацією функції handleSubmit, яка відповідає за обробку події відправки форми, представлено на рисунку 3.2. Основним завданням цієї функції є перевірка введених користувачем даних, формування об’єкта активності та передача його на серверну частину системи. Спочатку виконується перевірка обов’язкових полів (type, duration, date). У разі відсутності даних користувач отримує повідомлення про необхідність заповнення всіх полів. Після цього формується запит на додавання активності, а введені значення очищуються для підготовки форми до наступного використання. Після успішного додавання викликається функція refresh(), яка оновлює список активностей користувача.

```
15  const handleSubmit = async (e) => {
16    e.preventDefault();
17
18    if (!type || !duration || !date) {
19      alert('Заповніть всі поля');
20      return;
21    }
22
23    await addActivity(token, {
24      type,
25      duration: parseInt(duration),
26      calories: parseInt(calories) || 0,
27      date
28    });
29
30    setType('');
31    setDuration('');
32    setCalories('');
33    setDate('');
34
35    refresh();
36  };
```

Рисунок 3.2 – Логіка обробки форми додавання фізичної активності

На рисунку 3.3 показано безпосередню взаємодію клієнтської частини із серверним API.

```
23      await addActivity(token, {
24        type,
25        duration: parseInt(duration),
26        calories: parseInt(calories) || 0,
27        date
28      });
```

Рисунок 3.3 – Відправка даних фізичної активності на сервер

Використовується функція addActivity, якій передається токен авторизації та об’єкт із даними фізичної активності. Перед відправкою числові значення (тривалість та калорії) приводяться до типу integer. Використання JWT-токена дозволяє ідентифікувати користувача та прив’язати запис до його облікового запису. Цей етап є ключовим у процесі збереження даних, тому що відбувається передача інформації з клієнтської частини до серверної.

Фрагмент коду з відображенням користувацького інтерфейсу форми додавання фізичної активності представлено на рисунку 3.4.

```
return (
  <form
    onSubmit={handleSubmit}
    style={{
      marginTop: '40px',
      background: '□#fff',
      padding: '20px',
      borderRadius: '10px'
    }}
  >

    <h3>Додати активність</h3>

    <input
      placeholder="Тип активності"
      value={type}
      onChange={e => setType(e.target.value)}
    />

    <input
      placeholder="Тривалість"
      value={duration}
      onChange={e => setDuration(e.target.value)}
    />

    <input
      placeholder="Калорії"
      value={calories}
      onChange={e => setCalories(e.target.value)}
    />

    <input
      type="date"
      value={date}
      onChange={e => setDate(e.target.value)}
    />

    <button
      type="submit"
      style={{
        backgroundColor: '■#2575fc',
        color: 'white',
        border: 'none',
        padding: '12px',
        borderRadius: '8px',
        cursor: 'pointer',
        marginTop: '10px'
      }}
    >
      Додати
    </button>

  </form>
);
}
```

Рисунок 3.4 – Інтерфейс компонента AddActivity (форма введення даних)

Даний інтерфейс реалізований у вигляді HTML-форм, стилізованих за допомогою inline-стилів, та містить поля для введення типу активності, тривалості, кількості калорій та дати виконання. Кнопка відправки форми ініціює виклик функції handleSubmit, яка забезпечує подальшу обробку введених даних.

Таким чином, інтерфейс компонента є важливою частиною клієнтської системи, оскільки забезпечує основний канал введення даних про фізичну активність користувача.

На рисунку 3.5 представлено фрагмент коду, де за допомогою хука useState створюються змінні стану компонента. Вони використовуються для збереження введених користувачем даних (вага, зріст), а також результату розрахунку індексу маси тіла (BMI) та його текстового статусу. Реалізація основної логіки калькулятора BMI представлена на рисунку 3.6.

```
5   const [weight, setWeight] = useState('');
6   const [height, setHeight] = useState('');
7   const [bmi, setBmi] = useState(null);
8   const [status, setStatus] = useState('');
```

Рисунок 3.5 – Оголошення стану компонента BMICalculator

```
10  const calculateBMI = () => {
11
12    if (!weight || !height) {
13      alert('Заповніть всі поля');
14      return;
15    }
16
17    const h = height / 100;
18
19    const result = (
20      weight / (h * h)
21    ).toFixed(1);
22
23    setBmi(result);
24
25    if (result < 18.5) {
26      setStatus('Недостатня вага');
27    }
28    else if (result < 25) {
29      setStatus('Нормальна вага');
30    }
31    else if (result < 30) {
32      setStatus('Надлишкова вага');
33    }
34    else {
35      setStatus('Ожиріння');
36    }
37  };
```

Рисунок 3.6 – Реалізація алгоритму розрахунку індексу маси тіла (BMI)

На рисунку 3.7 наведено фрагмент коду користувацького інтерфейсу калькулятора BMI.

```
return (
  <div style={{
    marginTop: '40px',
    background: '#fff',
    padding: '20px',
    borderRadius: '10px',
    boxShadow: '0 5px 15px rgba(0,0,0,0.1)'
  }}>

    <h3>Калькулятор BMI (ІМТ)</h3>

    <input
      type="number"
      placeholder="Вага (кг)"
      value={weight}
      onChange={e => setWeight(e.target.value)}
      style={{
        width: '100%',
        marginBottom: '10px',
        padding: '10px'
      }}
    />

    <input
      type="number"
      placeholder="Зріст (см)"
      value={height}
      onChange={e => setHeight(e.target.value)}
      style={{
        width: '100%',
        marginBottom: '10px',
        padding: '10px'
      }}
    />

    <button
      onClick={calculateBMI}
      style={{
        backgroundColor: '#6a11cb',
        color: 'white',
        border: 'none',
        padding: '12px',
        borderRadius: '8px',
        cursor: 'pointer',
        width: '100%'
      }}
    >
      Розрахувати BMI
    </button>

    {
      bmi && (
        <div style={{
          marginTop: '20px',
          padding: '15px',
          background: '#f5f5f5',
          borderRadius: '8px'
        }}>

          <h4>
            Ваш BMI: {bmi}
          </h4>

          <p>
            Статус: <strong>{status}</strong>
          </p>

        </div>
      )
    }

  </div>
);
}
```

Рисунок 3.7 – Інтерфейс компонента BMICalculator

Спочатку проводиться перевірка наявності введених даних, а далі користувач вводить параметри маси тіла та зросту, після чого натискає кнопку для виконання розрахунку. На основі отриманого результату визначається категорія фізичного стану користувача.

На рисунку 3.8 представлено фрагмент коду з використанням хуку useState для створення локального стану компонента Dashboard. Масив activities зберігає список фізичних активностей користувача, отриманих із серверної частини. Змінна filter використовується для реалізації пошуку та фільтрації даних.

```
25    const [activities, setActivities] = useState([]);
26    const [filter, setFilter] = useState('');
```

Рисунок 3.8 – Ініціалізація стану та збереження списку фізичних активностей

На рисунку 3.9 наведено фрагмент коду з використанням функції fetchActivities, яка виконує асинхронний запит до API для отримання списку активностей користувача. Отримані дані зберігаються у стані компонента через setActivities. Виклик useEffect забезпечує автоматичне завантаження даних при ініціалізації компонента.

```
28    const fetchActivities = async () => {
29      const data = await getActivities(token);
30      setActivities(data);
31    };
32
33    useEffect(() => {
34      fetchActivities();
35    }, [token]);
36
```

Рисунок 3.9 – Завантаження фізичних активностей користувача з серверної частини

Для реалізації видалення запису фізичної активності використана функція (див. рис. 3.10). Перед видаленням користувачу виводиться

підтвердження через window.confirm. Після успішного видалення викликається повторне завантаження даних для оновлення списку.

```
37  // DELETE
38  const handleDelete = async (id) => {
39    if (!window.confirm('Видалити активність?')) return;
40
41    await deleteActivity(token, id);
42    fetchActivities();
43  };
```

Рисунок 3.10 – Реалізація функції видалення фізичної активності

На рисунку 3.11 представлена реалізація функції handleEdit, що дозволяє редагувати існуючий запис активності. Для введення нових даних використовується prompt. Після оновлення дані передаються на сервер через updateActivity, після чого список оновлюється.

```
45  // EDIT
46  const handleEdit = async (activity) => {
47
48    const type = prompt('Тип активності', activity.type);
49
50    if (!type) return;
51
52    const duration = prompt(
53      'Тривалість',
54      activity.duration
55    );
56
57    const calories = prompt(
58      'Калорії',
59      activity.calories
60    );
61
62    await updateActivity(token, activity.id, {
63      type,
64      duration: parseInt(duration),
65      calories: parseInt(calories),
66      date: activity.date
67    });
68
69    fetchActivities();
70  };
71
```

Рисунок 3.11 – Реалізація редагування запису фізичної активності

Механізм пошуку та фільтрації фізичних активностей, реалізовано через метод filter() масиву activities (див. рис. 3.12). Пошук здійснюється за типом активності з використанням toLowerCase() для нечутливості до регістру. Результат зберігається у змінній filteredActivities.

```
// FILTER
const filteredActivities = activities.filter(a =>
  a.type.toLowerCase().includes(filter.toLowerCase())
);

```

Рисунок 3.12 – Механізм пошуку та фільтрації фізичних активностей

На рисунку 3.13 представлено фрагменті коду використання методу reduce() для обчислення сумарних показників. Розраховується загальна кількість витрачених калорій та загальна тривалість фізичних активностей користувача.

```
// STATS
const totalCalories = activities.reduce(
  (sum, a) => sum + a.calories,
  0
);

const totalDuration = activities.reduce(
  (sum, a) => sum + a.duration,
  0
);

```

Рисунок 3.13 – Розрахунок показників користувача

Формування даних для побудови графіка фізичної активності представлено на рисунку 3.14.

```
// CHART
const chartData = activities.reduce((acc, act) => {

  const existing = acc.find(
    a => a.date === act.date
  );

  if (existing) {
    existing.calories += act.calories;
  } else {
    acc.push({
      date: act.date,
      calories: act.calories
    });
  }

  return acc;

}, []).sort(
  (a, b) => new Date(a.date) - new Date(b.date)
);
```

Рисунок 3.14 – Формування даних для графіка фізичної активності

Дані для графіка формуються за допомогою reduce(), де відбувається групування активностей за датою. Якщо дата вже існує, значення калорій додаються, інакше створюється новий запис. Далі дані сортуються за датою у хронологічному порядку.

Для побудови графіка візуалізації витрачених калорій користувача використовується бібліотека Recharts. Компоненти LineChart, XAxis, YAxis та Line забезпечують відображення динаміки витрачених калорій у вигляді лінійного графіка (див. рис. 3.15). Графік є інтерактивним і підтримує підказки (Tooltip).

```
<LineChart data={chartData}>

  <CartesianGrid strokeDasharray="3 3" />

  <XAxis dataKey="date" />

  <YAxis />

  <Tooltip />

  <Legend />

  <Line
    type="monotone"
    dataKey="calories"
    stroke="#2575fc"
    strokeWidth={3}
  />

</LineChart>
```

Рисунок 3.15 – Графічна візуалізація витрачених калорій користувача

На рисунку 3.16 представлено фрагмент коду, що відповідає за основну розмітку сторінки Dashboard. Тут реалізовано структуру інтерфейсу, включаючи заголовок, кнопку виходу та основний контейнер для даних користувача.

У фрагменті коду, що наведено на рисунку 3.17, здійснюється підключення дочірніх компонентів системи. Компонент AddActivity відповідає за додавання нових записів, а BMICalculator – за розрахунок індексу маси тіла. Передача пропсів дозволяє забезпечити взаємодію між компонентами.

```
110   return (
111     <div style={{
112       maxWidth: '900px',
113       margin: '40px auto',
114       fontFamily: 'Arial'
115     }}>
116
117       <div style={{
118         display: 'flex',
119         justifyContent: 'space-between',
120         alignItems: 'center'
121       }}>
122         <h2>Мої активності</h2>
123
124         <button
125           onClick={onLogout}
126           style={{
127             backgroundColor: '#dc3545',
128             color: 'white',
129             border: 'none',
130             padding: '10px 15px',
131             borderRadius: '8px',
132             cursor: 'pointer'
133           }}
134         >
135           Вийти
136         </button>
137       </div>
```

Рисунок 3.16 – Інтерфейс головної панелі керування (Dashboard)

```
291         <AddActivity
292           token={token}
293           refresh={fetchActivities}
294         />
295
296         <BMICalculator />
297
298       </div>
299     );
300   }
```

Рисунок 3.17 – Інтеграція додаткових модулів у Dashboard

У компоненті Login використовується useState для збереження email, пароля та повідомлень системи, що дозволяє динамічно обробляти введені користувачем дані (рис. 3.18).

```
5   const [email, setEmail] = useState('');
6   const [password, setPassword] = useState('');
7   const [message, setMessage] = useState('');
```

Рисунок 3.18 – Ініціалізація стану форми авторизації користувача

На рисунку 3.19 представлено використання функції handleSubmit, яка виконує відправку даних авторизації на сервер. При успішному вході користувачу повертається JWT-токен, який зберігається у стані застосунку. Також реалізовано обробку помилок.

```
9   const handleSubmit = async (e) => {
10    e.preventDefault();
11    setMessage('');
12
13    if (!email || !password) {
14      setMessage("Заповніть всі поля");
15      return;
16    }
17
18    try {
19      const res = await login({ email, password });
20      setToken(res.token);
21    } catch (err) {
22      setMessage(err.message);
23    }
24  };
```

Рисунок 3.19 – Реалізація процесу автентифікації користувача через API

Фрагмент коду, що відповідає за відображення UI форми входу, представлено на рисунку 3.20.

```
26  return (
27    <div style={{
28      display: 'flex',
29      justifyContent: 'center',
30      alignItems: 'center',
31      minHeight: '100vh',
32      background: 'linear-gradient(135deg, #74ebd5 0%, #ACB6E5 100%)',
33      fontFamily: 'Arial, sans-serif'
34    }}>
35      <div style={{
36        background: 'rgba(255,255,255,0.95)',
37        padding: '40px 30px',
38        borderRadius: '15px',
39        boxShadow: '0 10px 30px rgba(0,0,0,0.2)',
40        width: '100%',
41        maxWidth: '400px',
42        textAlign: 'center'
43      }}>
```

Рисунок 3.20 – UI форми авторизації користувача

У наступному фрагменті коду продовжується реалізація UI компонента Login, включаючи стилізацію кнопки входу та базову структуру форми (див. рис. 3.21). Забезпечується зручність взаємодії користувача із системою.

```
<h2>Вхід</h2>
<form onSubmit={handleSubmit} style={{ display: 'flex', flexDirection: 'column', gap: '15px', marginTop: '20px' }}>
  <input
    type="email"
    placeholder="Email"
    value={email}
    onChange={e => setEmail(e.target.value)}
    style={{
      padding: '12px 15px',
      borderRadius: '8px',
      border: '1px solid #ccc',
      fontSize: '14px',
      outline: 'none'
    }}
  />
  <input
    type="password"
    placeholder="Пароль"
    value={password}
    onChange={e => setPassword(e.target.value)}
    style={{
      padding: '12px 15px',
      borderRadius: '8px',
      border: '1px solid #ccc',
      fontSize: '14px',
      outline: 'none'
    }}
  />
```

Рисунок 3.21 – Стилізація кнопки та структура форми авторизації

Фінальний вигляд форми авторизації користувача, який забезпечує повний цикл входу в систему представлено на рисунку 3.22. Таким чином, інтерфейс включає всі необхідні поля та елементи управління.

```
  <button type="submit" style={{
    padding: '12px',
    fontSize: '16px',
    color: 'white',
    border: 'none',
    borderRadius: '10px',
    background: 'linear-gradient(45deg, #6a11cb, #2575fc)',
    cursor: 'pointer',
    transition: 'all 0.3s ease'
  }}
  onMouseOver={e => e.currentTarget.style.transform = 'translateY(-3px)'}
  onMouseOut={e => e.currentTarget.style.transform = 'translateY(0)'}
  >Увійти</button>
</form>
```

Рисунок 3.22 – Фінальний вигляд форми авторизації користувача

На рисунку 3.23 представлено фрагмент коду, що відповідає за виведення повідомлень про помилки або успішну авторизацію. Стан message дозволяє динамічно відображати результат операції користувачу.

```
{message && <p style={{ color: message.includes('успішна') ? 'green' : 'red', marginTop: '10px' }}>{message}</p>}
```

Рисунок 3.23 – Відображення повідомлень про помилки авторизації

Кнопка setIsLogin(false) забезпечує перемикання між формами входу та реєстрації (див. рис. 3.24). Це дозволяє реалізувати єдиний компонент авторизації з двома режимами.

```
<button
  onClick={() => setIsLogin(false)}
  style={{
    marginTop: '5px',
    color: '#2575fc',
    cursor: 'pointer',
    border: 'none',
    background: 'none',
    textDecoration: 'underline'
  }}
  onMouseOver={e => e.currentTarget.style.color = '#6a11cb'}
  onMouseOut={e => e.currentTarget.style.color = '#2575fc'}
>
  Зареєструватися
</button>
```

Рисунок 3.24 – Реалізація перемикання між формами авторизації

У компоненті Register створюються змінні стану для імені, email, пароля та повідомлень, що представлено на рисунку 3.25 та дозволяє обробляти введені дані. Функція handleSubmit виконує відправку даних реєстрації на сервер. Після успішної реєстрації користувачу пропонується перейти до форми входу. Даний фрагмент коду представлено на рисунку 3.26.

```
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [message, setMessage] = useState('');
```

Рисунок 3.25 – Ініціалізація стану форми реєстрації користувача

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setMessage('');

  if (!name || !email || !password) {
    setMessage("Заповніть всі поля");
    return;
  }

  try {
    await register({ name, email, password });
    setMessage("Реєстрація успішна! Увійдіть у систему");
    setIsLogin(true);
  } catch (err) {
    setMessage(err.message);
  }
};
```

Рисунок 3.26 – Реалізація процесу реєстрації користувача через API

Інтерфейс компонента реєстрації користувача (Register) представлено на рисунку 3.27.

```
return (
  <div style={{
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    minHeight: '100vh',
    background: 'linear-gradient(135deg, #74ebd5 0%, #ACB6E5 100%)',
    fontFamily: 'Arial, sans-serif'
  }}>
    <div style={{
      background: 'rgba(255,255,255,0.95)',
      padding: '40px 30px',
      borderRadius: '15px',
      boxShadow: '0 10px 30px rgba(0,0,0,0.2)',
      width: '100%',
      maxWidth: '400px',
      textAlign: 'center'
    }}>
      <h2>Реєстрація</h2>
      <form onSubmit={handleSubmit} style={{ display: 'flex', flexDirection: 'column', gap: '15px', marginTop: '20px' }}>
        <input
          placeholder="Ім'я"
          value={name}
          onChange={e => setName(e.target.value)}
          style={{
            padding: '12px 15px',
            borderRadius: '8px',
            border: '1px solid #ccc',
            fontSize: '14px',
            outline: 'none'
          }}
        />
```

Рисунок 3.27 – Інтерфейс компонента реєстрації користувача (Register)

Продовження реалізації UI форми реєстрації з кнопкою відправки даних наведено на рисунку 3.28, що забезпечує базову стилізацію та адаптивність інтерфейсу.

```
        <input
          placeholder="Email"
          value={email}
          onChange={e => setEmail(e.target.value)}
          style={{
            padding: '12px 15px',
            borderRadius: '8px',
            border: '1px solid #ccc',
            fontSize: '14px',
            outline: 'none'
          }}
        />
        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={e => setPassword(e.target.value)}
          style={{
            padding: '12px 15px',
            borderRadius: '8px',
            border: '1px solid #ccc',
            fontSize: '14px',
            outline: 'none'
          }}
        />
```

Рисунок 3.28 – Реалізації UI-форми реєстрації з кнопкою відправки даних

Фінальна частина інтерфейсу реєстрації, що включає кнопку переходу до логіну і забезпечує зручну навігацію між формами, представлена на рисунку 3.29.

```
<button type="submit" style={{
  padding: '12px',
  fontSize: '16px',
  color: 'white',
  border: 'none',
  borderRadius: '10px',
  background: 'linear-gradient(45deg, #6a11cb, #2575fc)',
  cursor: 'pointer',
  transition: 'all 0.3s ease'
}}
onMouseOver={e => e.currentTarget.style.transform = 'translateY(-3px)'}
onMouseOut={e => e.currentTarget.style.transform = 'translateY(0)'}
>Зареєструватися</button>
</form>
```

Рисунок 3.29 – Фінальна частина інтерфейсу реєстрації користувача

На рисунку 3.30 представлено фрагмент коду з реалізацією виведення повідомлення про результат реєстрації.

```
{message && <p style={{ color: message.includes('успішна') ? 'green' : 'red', marginTop: '10px' }}>{message}</p>}
```

Рисунок 3.30 – Відображення повідомлень про стан реєстрації користувача

Перехід до форми авторизації користувача наведено на рисунку 3.31. Кнопка дозволяє повернути користувача до форми входу після реєстрації. Це завершує цикл авторизації в системі.

```
<button
  onClick={() => setIsLogin(true)}
  style={{
    marginTop: '5px',
    color: '#2575fc',
    cursor: 'pointer',
    border: 'none',
    background: 'none',
    textDecoration: 'underline'
  }}
  onMouseOver={e => e.currentTarget.style.color = '#6a11cb'}
  onMouseOut={e => e.currentTarget.style.color = '#2575fc'}
>
  Увійти
</button>
```

Рисунок 3.31 – Перехід до форми авторизації користувача

Таким чином, реалізація клієнтської частини вебзастосунку забезпечує ефективну взаємодію користувача із системою, зручне управління фізичними активностями, візуалізацію статистичних даних та підтримку основних функціональних можливостей, необхідних для повноцінної роботи вебзастосунку для відстеження та аналізу фізичної активності користувача.

3.2 Реалізація серверної частини вебзастосунку

Серверна частина розроблена на платформі Node.js із використанням фреймворку Express.js, що забезпечує обробку HTTP-запитів, виконання бізнес-логіки та взаємодію з базою даних. Реалізація серверної логіки включає обробку маршрутів API, перевірку вхідних даних, авторизацію користувачів та повернення структурованих JSON-відповідей клієнтській частині системи.

Підключення основних бібліотек та модулів серверної частини системи представлено на рисунку 3.32.

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3   const cors = require('cors');
4
5   const authRoutes = require('./routes/auth');
6   const activityRoutes = require('./routes/activities');
7
```

Рисунок 3.32 – Підключення бібліотек та модулів серверної частини вебзастосунку

Бібліотека Express використовується для створення HTTP-сервера та реалізації REST API. Middleware body-parser забезпечує обробку JSON-запитів, а cors дозволяє організувати взаємодію між клієнтською та серверною частинами, які можуть працювати на різних портах або доменах. Також підключаються окремі модулі маршрутів авторизації та управління фізичними активностями користувача.

Після підключення необхідних модулів виконується створення екземпляра Express-застосунку, що представлено на рисунку 3.33. Додатково підключаються middleware-компоненти, які забезпечують коректну обробку HTTP-запитів. Middleware cors() дозволяє серверу приймати запити від клієнтської частини вебзастосунку, а bodyParser.json() використовується для автоматичного перетворення JSON-даних у JavaScript-об'єкти, що спрощує подальшу обробку інформації на сервері.

```
8   const app = express();
9   app.use(cors());
10  app.use(bodyParser.json());
```

Рисунок 3.33 – Ініціалізація Express-сервера та middleware

Для забезпечення структурованої організації серверної логіки у вебзастосунку використовується механізм маршрутизації. Маршрут /api/auth відповідає за функціонал реєстрації та авторизації користувачів, тоді як /api/activities забезпечує обробку запитів, що пов'язані із фізичними активностями користувача (див. рис. 3.34). Такий підхід дозволяє логічно розділити функціональні модулі серверної частини та спрощує подальшу підтримку програмного коду.

```
12  app.use('/api/auth', authRoutes);
13  app.use('/api/activities', activityRoutes);
14
```

Рисунок 3.34 – Підключення маршрутів REST API серверної частини

На завершальному етапі виконується запуск серверної частини системи на локальному порту 5000, що представлено на рисунку 3.35. Після успішного запуску сервер починає обробляти HTTP-запити від клієнтської частини вебзастосунку та забезпечує взаємодію із базою даних.

```
15  app.listen(5000, () => console.log("Server running on port 5000"));
```

Рисунок 3.35 – Запуск серверної частини вебзастосунку

Використання локального сервера дозволяє реалізувати повноцінну взаємодію між клієнтською та серверною частинами системи під час розробки та тестування вебзастосунку.

На рисунку 3.36 представлено фрагмент коду, де виконується підключення до локальної реляційної бази даних SQLite. Для роботи із системою управління бази даних використовується бібліотека sqlite3, яка забезпечує взаємодію серверної частини вебзастосунку з базою даних. База даних зберігається у локальному файлі fitness.db, що спрощує розгортання системи та не потребує використання окремого серверу баз даних.

```
1   const sqlite3 = require('sqlite3').verbose();
2   const db = new sqlite3.Database('./fitness.db');
```

Рисунок 3.36 – Підключення до бази даних SQLite

Створення таблиці користувачів users, яка призначена для збереження інформації про зареєстрованих користувачів системи, представлена на рисунку 3.37. Таблиця містить унікальний ідентифікатор користувача, ім’я, електронну пошту та пароль.

```
5   db.run(`CREATE TABLE IF NOT EXISTS users (
6     id INTEGER PRIMARY KEY AUTOINCREMENT,
7     name TEXT,
8     email TEXT UNIQUE,
9     password TEXT
10  )`);
```

Рисунок 3.37 – Створення таблиці користувачів у базі даних

Поле id використовується як первинний ключ та автоматично збільшується при створенні нового запису. Поле email має обмеження UNIQUE, що унеможливлює створення декількох облікових записів із однаковою електронною адресою і дозволяє реалізувати механізми авторизації та персоналізації даних у вебзастосунку.

На рисунку 3.38 представлено фрагмент коду створення таблиці activities, яка використовується для збереження інформації про фізичну

активність користувачів системи і містить дані про тип активності, тривалість виконання, кількість витрачених калорій та дату проведення активності.

```
12    db.run(`CREATE TABLE IF NOT EXISTS activities (
13      id INTEGER PRIMARY KEY AUTOINCREMENT,
14      user_id INTEGER,
15      type TEXT,
16      duration INTEGER,
17      calories INTEGER,
18      date TEXT,
19      FOREIGN KEY(user_id) REFERENCES users(id)
20    )`);
```

Рисунок 3.38 – Створення таблиці фізичних активностей користувача

Поле user_id є зовнішнім ключем, який встановлює зв’язок між таблицями activities та users, що дозволяє реалізувати зв’язок типу «один до багатьох», при якому один користувач може мати декілька записів фізичної активності.

Для послідовного виконання SQL-запитів під час ініціалізації бази даних використовується метод serialize() (див. рис. 3.39). Це гарантує коректний порядок створення таблиць та забезпечує стабільну роботу системи під час запуску серверної частини вебзастосунку.

```
4   db.serialize(() => {
```

Рисунок 3.39 – Послідовна ініціалізація структури бази даних

Після створення та налаштування бази даних виконується експорт об’єкта підключення db, що дозволяє використовувати його в інших модулях серверної частини вебзастосунку для виконання SQL-запитів та роботи з даними (див. рис. 3.40).

```
23   module.exports = db;
```

Рисунок 3.40 – Експорт модуля бази даних

На наступному етапі виконується підключення необхідних модулів серверної частини. Бібліотека Express Router використовується для реалізації

маршрутизації API-запитів, а модуль jsonwebtoken забезпечує роботу з JWT-токенами для автентифікації користувачів. Також підключається модель Activity, яка відповідає за взаємодію з базою даних та виконання операцій над записами фізичної активності (див. рис. 3.41).

```
1   const express = require('express');
2   const router = express.Router();
3   const Activity = require('../models/activity');
4   const jwt = require('jsonwebtoken');
5
6   const SECRET = "supersecretkey";
7
```

Рисунок 3.41 – Підключення модулю Activity

Для забезпечення захисту API-запитів реалізовано middleware автентифікації користувача з представленням фрагменту коду на рис. 3.42. Під час виконання запиту сервер перевіряє наявність JWT-токена у заголовку Authorization.

```
8   // Middleware auth
9   const auth = (req, res, next) => {
10    const header = req.headers['authorization'];
11
12    if (!header) {
13      return res.status(401).json({
14        error: "Не авторизовано"
15      });
16    }
17
18    const token = header.split(' ')[1];
19
20    jwt.verify(token, SECRET, (err, decoded) => {
21      if (err) {
22        return res.status(401).json({
23          error: "Не авторизовано"
24        });
25      }
26
27      req.userId = decoded.id;
28      next();
29    });
30  };
```

Рисунок 3.42 – Реалізація middleware автентифікації користувача

У випадку успішної перевірки токена виконується декодування інформації про користувача та збереження його ідентифікатора у змінній

req.userId, що дозволяє обмежити доступ до персональних даних лише для авторизованих користувачів системи. При відсутності або некоректності токена сервер повертає помилку авторизації зі статусом 401 Unauthorized.

Для отримання списку фізичних активностей авторизованого користувача використовується маршрут GET (див. рис. 3.43).

```
32  // GET
33  router.get('/', auth, (req, res) => {
34    Activity.getAllByUser(req.userId, (err, rows) => {
35      if (err) {
36        return res.status(400).json({
37          error: "Помилка отримання"
38        });
39      }
40
41      res.json(rows);
42    });
43  });
```

Рисунок 3.43 – Реалізація отримання списку фізичних активностей користувача

Перед виконанням запиту застосовується middleware авторизації, що забезпечує перевірку прав доступу. Після успішної автентифікації виконується звернення до моделі Activity, яка отримує всі записи фізичної активності конкретного користувача із бази даних. Отримані дані передаються клієнтській частині у форматі JSON.

Для створення нового запису фізичної активності користувача використовується Маршрут POST (див. рис. 3.44). Дані про тип активності, тривалість, витрачені калорії та дату виконання надходять із клієнтської частини у тілі HTTP-запиту.

Після успішної перевірки авторизації виконується створення нового запису у базі даних. У відповідь сервер повертає інформацію про створену активність, що дозволяє автоматично оновити дані на клієнтській стороні вебзастосунку.

Механізм оновлення існуючих записів фізичної активності реалізовується через маршрут PUT з представленням на рисунку 3.45. Ідентифікатор запису передається через параметри URL-запиту, а нові

значення – у тілі HTTP-запиту. Після успішного оновлення сервер повертає повідомлення про успішне редагування запису. Реалізація даного функціоналу дозволяє користувачу коригувати інформацію про власну фізичну активність.

```
// POST
router.post('/', auth, (req, res) => {
  const { type, duration, calories, date } = req.body;

  Activity.create(
    req.userId,
    type,
    duration,
    calories,
    date,
    (err, id) => {
      if (err) {
        return res.status(400).json({
          error: "Помилка створення"
        });
      }

      res.json({
        id,
        type,
        duration,
        calories,
        date
      });
    }
  );
});
```

Рисунок 3.44 – Реалізація додавання фізичної активності

```
// PUT
router.put('/:id', auth, (req, res) => {
  const { type, duration, calories, date } = req.body;

  Activity.update(
    req.params.id,
    req.userId,
    type,
    duration,
    calories,
    date,
    (err) => {
      if (err) {
        return res.status(400).json({
          error: "Помилка оновлення"
        });
      }

      res.json({
        message: "Активність оновлено"
      });
    }
  );
});
```

Рисунок 3.45 – Реалізація редагування фізичної активності

Для видалення запису фізичної активності користувача із бази даних використовується маршрут DELETE (див. рис. 3.46). Перед виконанням операції виконується перевірка автентифікації користувача.

Після успішного видалення запису сервер повертає повідомлення про успішне виконання операції. Реалізація даного функціоналу забезпечує можливість повноцінного управління історією фізичної активності користувача.

```
98   // DELETE
99   router.delete('/:id', auth, (req, res) => {
100    Activity.delete(req.params.id, req.userId, (err) => {
101      if (err) {
102        return res.status(400).json({
103          error: "Помилка видалення"
104        });
105      }
106
107      res.json({
108        message: "Активність видалено"
109      });
110    });
111  });
```

Рисунок 3.46 – Реалізація видалення фізичної активності

На завершальному етапі виконується експорт створеного маршрутизатора (див. рис. 3.47), що дозволяє підключити модуль до основного серверного файлу вебзастосунку та забезпечити доступ до API фізичних активностей. Далі виконується підключення необхідних модулів серверної частини та створення маршрутизатора Express, що представлено на рисунку 3.48.

```
113  module.exports = router;
```

Рисунок 3.47 – Експорт маршрутизатора фізичних активностей

```
1  const express = require('express');
2  const router = express.Router();
3  const User = require('../models/user');
4  const bcrypt = require('bcrypt');
5  const jwt = require('jsonwebtoken');
6
7  const SECRET = "supersecretkey";
```

Рисунок 3.48 – Підключення модулів авторизації та створення маршрутизатора auth.js

Створення нового облікового запису користувача забезпечує маршрут /register (див. рис. 3.49). У процесі реєстрації сервер отримує ім'я користувача, електронну пошту та пароль із HTTP-запиту. Якщо будь-яке з полів не заповнене, система повертає повідомлення про помилку. При успішному створенні користувача формується відповідь про успішну реєстрацію.

```
// Реєстрація
router.post('/register', (req, res) => {
  const { name, email, password } = req.body;

  if (!name || !email || !password) {
    return res.status(400).json({ error: "Заповніть всі поля" });
  }

  User.create(name, email, password, (err, id) => {
    if (err) {
      return res.status(400).json({ error: "Користувач вже існує" });
    }
    res.json({ message: "Успішна реєстрація" });
  });
});
```

Рисунок 3.49 – Реалізація маршруту реєстрації користувача

Механізм входу користувача до системи реалізує маршрут/login (див. рис. 3.50). Для перевірки пароля використовується бібліотека bcrypt.

```
// Логін
router.post('/login', (req, res) => {
  const { email, password } = req.body;

  User.findByEmail(email, (err, user) => {
    if (err || !user) {
      return res.status(400).json({ error: "Невірний email або пароль" });
    }

    const isMatch = bcrypt.compareSync(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ error: "Невірний email або пароль" });
    }

    const token = jwt.sign({ id: user.id }, SECRET);
    res.json({ token });
  });
});
```

Рисунок 3.50 – Реалізація маршруту авторизації користувача

Під час авторизації система виконує пошук користувача за електронною поштою. Якщо користувача не знайдено або пароль введено неправильно, сервер повертає повідомлення про помилку. У випадку успішної перевірки

створюється JWT-токен, який використовується для подальшої автентифікації користувача під час роботи із захищеними маршрутами системи.

Після реалізації всіх маршрутів модуль експортується для подальшого підключення у головному файлі серверної частини вебзастосунку (див. рис. 3.51). Далі здійснюється підключення до модуля бази даних (див. рис. 3.52).

```
43
44  module.exports = router;
```

Рисунок 3.51 – Експорт маршрутизатора авторизації

```
1  const db = require('../db');
2
```

Рисунок 3.52 – Підключення модуля бази даних у activity.js

Фрагмент коду забезпечує доступ до структури бази даних та дозволяє виконувати SQL-запити до таблиці фізичних активностей, що представлено на рисунку 3.53. За створення нового запису фізичної активності користувача відповідає метод create().

```
3  const Activity = {
4    create: (user_id, type, duration, calories, date, callback) => {
```

Рисунок 3.53 – Створення об’єкта Activity та метод створення фізичної активності

Під час виконання функції у таблицю activities передаються такі параметри: ідентифікатор користувача, тип активності, тривалість, кількість витрачених калорій та дата виконання активності (див. рис. 3.54). Для додавання запису використовується SQL-запит INSERT INTO, який забезпечує збереження інформації у базі даних. Після успішного створення запису система повертає ідентифікатор нової активності через this.lastID, що дозволяє надалі працювати із конкретним записом.

```
5       db.run(
6         `INSERT INTO activities (user_id, type, duration, calories, date)
7         VALUES (?, ?, ?, ?, ?)`,
```

Рисунок 3.54 – SQL-запит додавання активності до бази даних

Отримання всіх фізичних активностей конкретного користувача реалізує метод getAllByUser(). Для цього використовується SQL-запит SELECT, який виконує вибірку записів за user_id. У межах запиту реалізовано сортування записів за датою у порядку спадання, що дозволяє відображати найновіші активності першими (див. рис. 3.55).

```
15    getAllByUser: (user_id, callback) => {
16      db.all(
17        `SELECT * FROM activities
18        WHERE user_id = ?
19        ORDER BY date DESC`,
```

Рисунок 3.55 – Метод отримання активностей користувача з SQL-запитом отримання списку активностей

Для редагування існуючих записів фізичної активності використовується метод update(). Користувач має можливість змінювати тип активності, тривалість, кількість калорій та дату виконання (див. рис. 3.56). Використання перевірки user_id забезпечує додатковий рівень безпеки та унеможливлює зміну чужих записів. Видалення запису фізичної активності з бази даних з реалізацією методом delete() представлено на рисунку 3.57. SQL-запит DELETE видаляє лише ті записи, які належать поточному користувачу.

```
25    update: (id, user_id, type, duration, calories, date, callback) => {
26      db.run(
27        `UPDATE activities
28        SET type = ?, duration = ?, calories = ?, date = ?
29        WHERE id = ? AND user_id = ?`,
```

Рисунок 3.56 – Оновлення даних фізичної активності

```
35    delete: (id, user_id, callback) => {
36      db.run(
37        `DELETE FROM activities
38        WHERE id = ? AND user_id = ?`,
```

Рисунок 3.57 – Видалення запису фізичної активності

Наприкінці файлу виконується експорт об’єкта Activity, що дозволяє використовувати реалізовані методи в інших частинах серверної логіки вебзастосунку (рис. 3.58). А далі підключення бази даних до бібліотеки bcrypt (див. рис. 3.59).

```
45   module.exports = Activity;
```

Рисунок 3.58 – Експорт модуля Activity

```
1   const db = require('../db');
2   const bcrypt = require('bcrypt');
```

Рисунок 3.59 – Підключення бази даних та бібліотеки bcrypt

Таким чином, файл activity.js реалізує основну логіку роботи з фізичними активностями користувача та забезпечує виконання всіх необхідних операцій із базою даних.

Для реєстрації нового користувача у системі знову використовуємо метод create(). Перед збереженням пароля у базі даних виконується його хешування за допомогою функції bcrypt.hashSync(). Використання хешування є важливим елементом безпеки, оскільки пароль не зберігається у відкритому вигляді. Навіть у випадку отримання доступу до бази даних сторонні особи не зможуть побачити реальні паролі користувачів. Після хешування виконується SQL-запит для додавання нового користувача до таблиці users (див. рис. 3.60). Після успішного виконання запиту система повертає ідентифікатор нового користувача через this.lastID.

```
7       db.run(
8         `INSERT INTO users (name, email, password) VALUES (?, ?, ?)`,
9         [name, email, hashedPassword],
10        function (err) {
11          callback(err, this.lastID);
12        }
13      );
14    },
```

Рисунок 3.60 – Додавання користувача до бази даних

Пошук користувача за адресою електронної пошти представлено на рисунку 3.61 і реалізовано методом findByEmail() з отриманням інформації про користувача SQL-запитом SELECT і експортом модуля User.

```
16    findByEmail: (email, callback) => {
17      db.get(`SELECT * FROM users WHERE email = ?`, [email], callback);
18    },
19  };
20
21  module.exports = User;
```

Рисунок 3.61 – Пошук користувача за email

Таким чином, серверна частина вебзастосунку забезпечує реалізацію основної бізнес-логіки системи, обробку даних користувачів, взаємодію з базою даних, захист інформації та підтримку функціональних можливостей вебзастосунку для відстеження та аналізу фізичної активності користувача.

3.3 Реалізація авторизації

Одним із ключових функціональних модулів розробленого вебзастосунку є система авторизації користувачів, яка забезпечує контроль доступу до персональних даних і функціоналу системи.

У межах проєкту авторизація реалізована на основі клієнт-серверної архітектури з використанням JWT (JSON Web Token). Процес включає основні етапи: реєстрацію користувача, перевірку введених даних, створення облікового запису, вхід у систему, генерацію JWT-токена, перевірку токена при доступі до захищених маршрутів та надання доступу до функціоналу.

На етапі реєстрації користувач вводить необхідні дані (ім’я, електронну пошту та пароль) (див. рис. 3.62), далі вони передаються на сервер через HTTP-запит, де виконуються перевірка коректності, заповнення обов’язкових полів та унікальності електронної пошти.

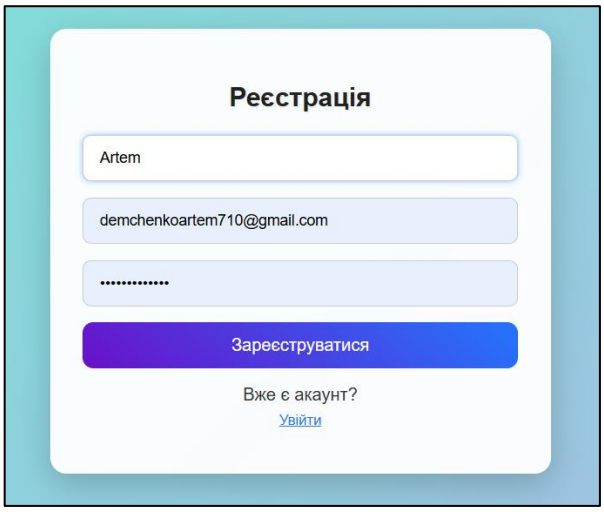


Рисунок 3.62 – Реєстрація користувача

Особлива увага приділяється захисту паролів користувачів. Перед збереженням у базі даних пароль проходить процедуру хешування, що дозволяє уникнути збереження конфіденційної інформації у відкритому вигляді. Такий підхід значно підвищує рівень безпеки системи та зменшує ризик несанкціонованого доступу до персональних даних користувачів.

Після успішної реєстрації користувач може виконати вхід до системи (див. рис. 3.63). Для цього на сторінці авторизації необхідно ввести електронну пошту та пароль. Клієнтська частина передає введені дані на сервер, де здійснюється перевірка їх відповідності інформації, що зберігається у базі даних. Якщо введені дані є коректними, сервер формує JWT-токен та надсилає його клієнтській частині.

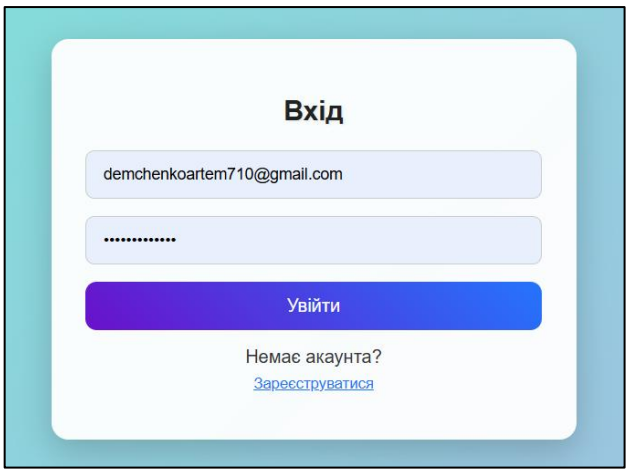


Рисунок 3.63 – Вхід до системи

У випадку помилки авторизації система відображає відповідне повідомлення користувачу, що дозволяє швидко виявити причину помилки та повторити спробу входу, що представлено на рисунку 3.64.

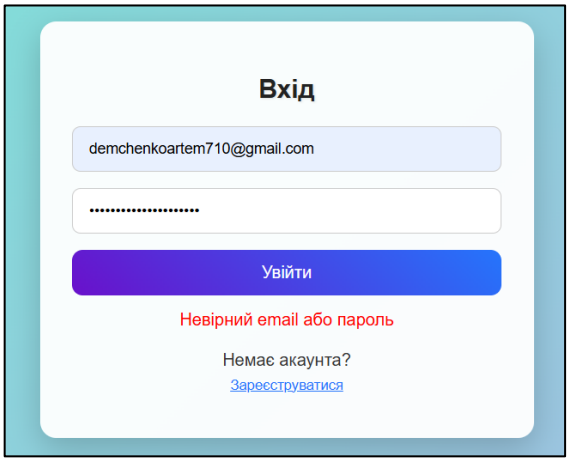


Рисунок 3.64 – Помилка авторизації

Такий підхід покращує зручність використання вебзастосунку та забезпечує більш комфортну взаємодію користувача із системою.

Важливим аспектом реалізації авторизації є також забезпечення безпеки передачі даних між клієнтською та серверною частинами. Для цього використовується формат JSON та HTTP-запити, що дозволяє стандартизувати процес взаємодії компонентів системи та забезпечити коректну передачу інформації.

Таким чином, реалізований модуль авторизації забезпечує безпечну взаємодію користувача із вебзастосунком, дозволяє контролювати доступ до функціональних можливостей системи та є важливою складовою загальної архітектури розробленого програмного продукту.

3.4 Реалізація аналітики та графіків

Калькулятор BMI – це зручний онлайн-інструмент відповідності маси тіла зросту і дозволяє отримати інформацію користувачу про відповідність

його ваги нормі або є відхилення. Користувач вводить значення маси тіла та зросту, після чого система автоматично виконує розрахунок та відображає результат разом із категорією (недостатня вага, норма або ожиріння), що представлено на рисунку 3.65.

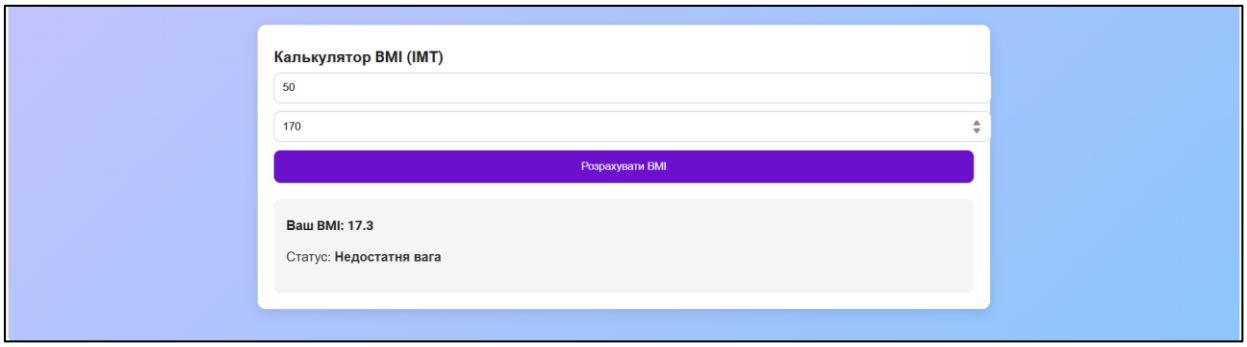


Рисунок 3.65 – Калькулятор BMI у вебзастосунку

Аналітичний модуль реалізований як окрема функціональна частина системи, яка взаємодіє із серверною частиною та базою даних. Основним джерелом інформації для побудови аналітики є записи про фізичну активність користувача, що містять тип активності, тривалість виконання, кількість витрачених калорій та дату виконання тренування. Відображення фізичної активності користувача представлено на рисунку 3.66.

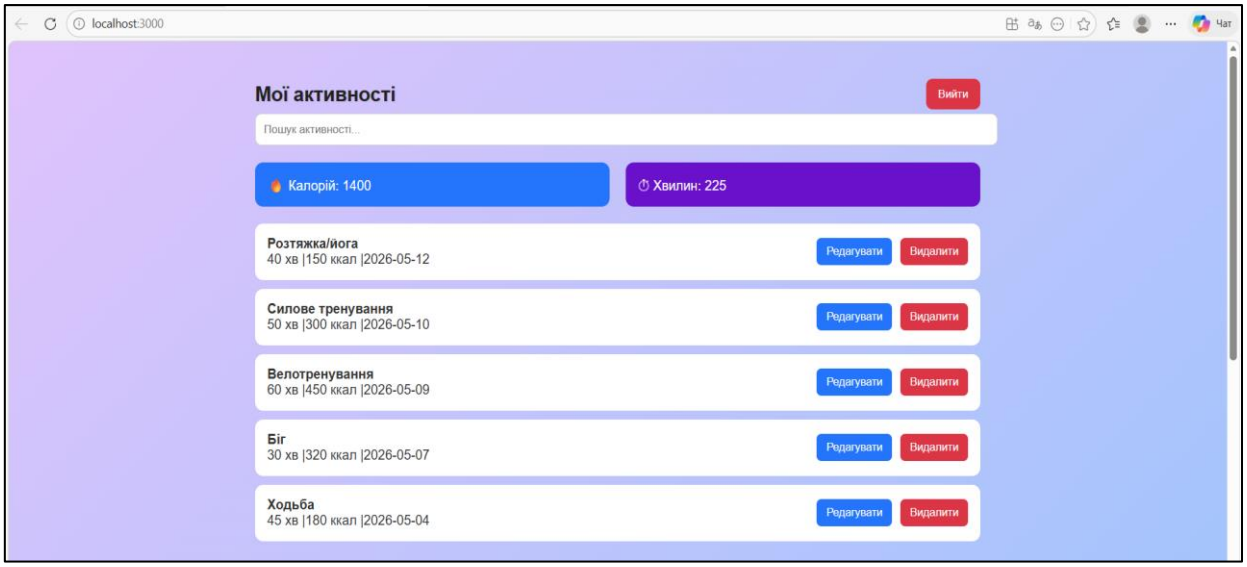


Рисунок 3.66 – Відображення фізичної активності користувача у вебзастосунку

На рисунку 3.67 наведено інтерфейс форми, призначеної для створення нового запису фізичної активності користувача. Форма містить поля для введення типу активності, її тривалості та кількості витрачених калорій. Після заповнення відповідних даних користувач має можливість зберегти запис, який автоматично додається до загального переліку активностей і враховується при формуванні статистичних показників системи.

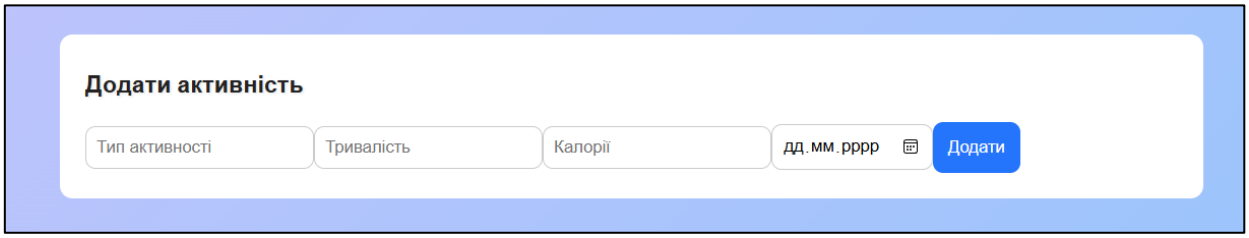


Рисунок 3.67 – Форма додавання фізичної активності користувача

Система аналітики фізичної активності є однією з функціональних можливостей розробленого вебзастосунку, яка забезпечує обробку, аналіз та візуалізацію показників у зручному для користувача вигляді і підвищує практичну цінність системи.

Після отримання даних із бази даних серверна частина виконує їх обробку та формує статистичну інформацію у форматі JSON, яка використовується для відображення статистики та побудови графіків у вебінтерфейсі (див. рис. 3.68).

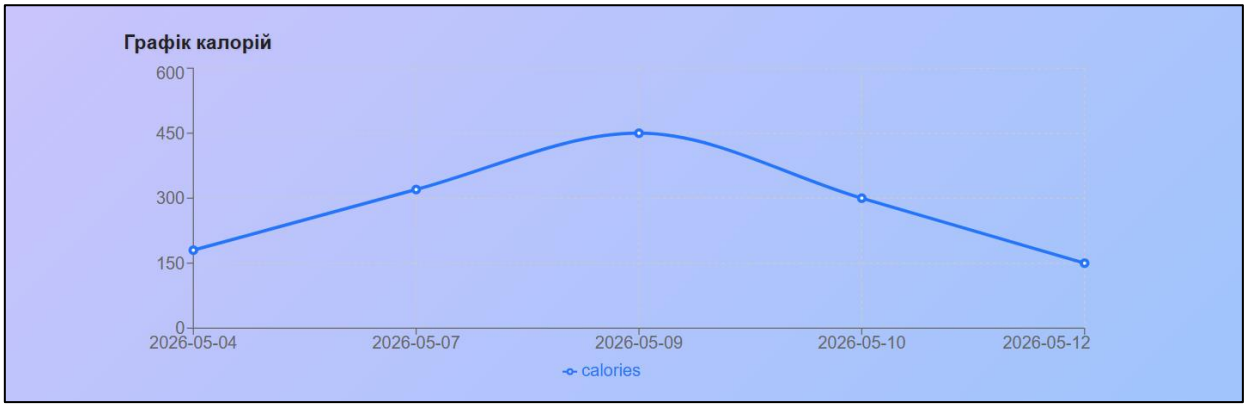


Рисунок 3.68 – Візуалізація активності витрачених калорій користувача у вигляді графіка

Таким чином, реалізований модуль аналітики та графічної візуалізації значно розширює функціональні можливості вебзастосунку, забезпечує користувачу зручний інструмент для аналізу своєї фізичної активності та підвищує загальну ефективність використання розробленої системи.

3.5 Тестування вебзастосунку

Після завершення етапу розробки вебзастосунку було проведено комплексне тестування системи з метою перевірки коректності роботи реалізованого функціоналу, стабільності взаємодії між клієнтською та серверною частинами і перевірки правильності обробки та збереження даних користувача. Проведення тестування є важливим етапом створення програмного забезпечення, оскільки дозволяє виявити можливі помилки, перевірити відповідність функціоналу поставленим вимогам та оцінити загальну працездатність системи.

Під час тестування модуля реєстрації користувача було виконано перевірку роботи REST API методом POST /api/auth/register, що представлено на рисунку 3.69. У запиті передавалися дані користувача, зокрема ім’я, електронна пошта та пароль.

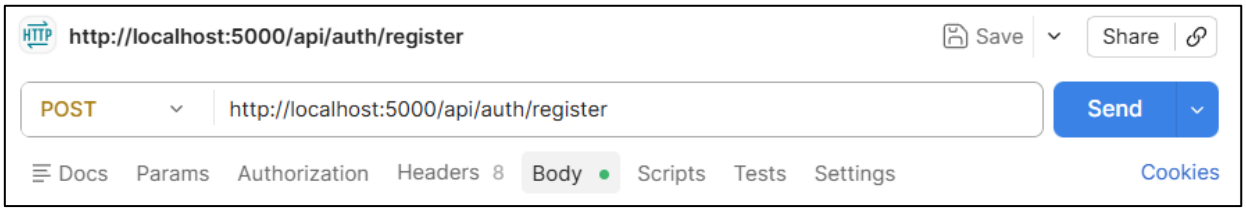


Рисунок 3.69 – Відправка POST-запиту на реєстрацію користувача через Postman (API /api/auth/register)

У ході тестування було використано наступний JSON-запит (див. рис. 3.70). Очікувалося, що система створить нового користувача та поверне повідомлення про успішну реєстрацію.

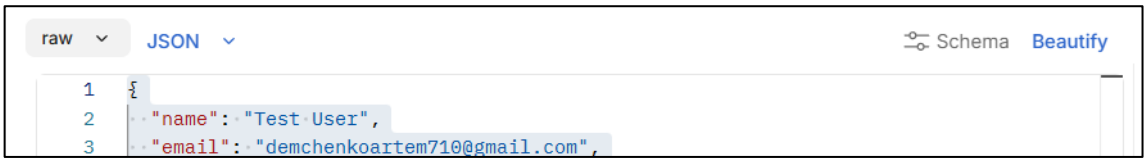


Рисунок 3.70 – Тіло запиту (JSON) для реєстрації користувача у системі

Однак фактичний результат показав, що сервер повернув відповідь, що представлена на рисунку 3.71.



Рисунок 3.71 – Відповідь сервера при повторній реєстрації користувача (помилка унікальності email)

Отримана відповідь свідчить про те, що система коректно виконує перевірку унікальності користувача за електронною адресою. Оскільки в базі даних поле email має обмеження UNIQUE, повторна реєстрація користувача з однаковою електронною адресою неможлива. Таким чином, система правильно відхиляє дубльований запит та забезпечує цілісність даних у базі. Це підтверджує коректну роботу механізму реєстрації користувачів.

Під час тестування модуля авторизації було виконано перевірку роботи REST API методу (див. рис. 3.72).

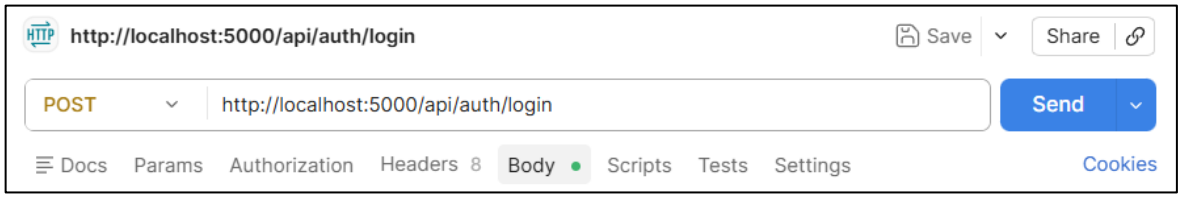


Рисунок 3.72 – Тестування REST API методу авторизації користувача (POST /api/auth/login) у Postman

У запиті передавалися дані користувача (електронна пошта та пароль), які були попередньо збережені під час реєстрації (див. рис. 3.73).

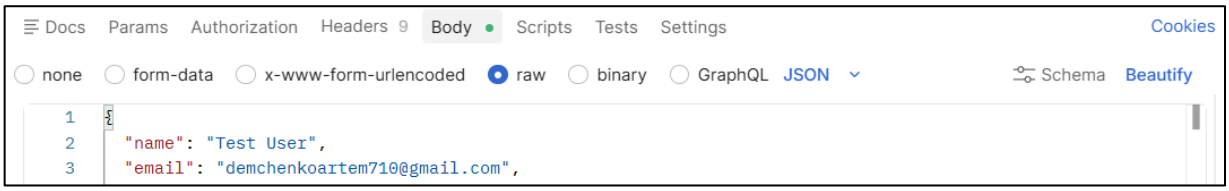


Рисунок 3.73 – Тіло запиту (JSON) для авторизації користувача в системі

У відповідь сервер повернув наступний результат, що представлений на рисунку 3.74.



Рисунок 3.74 – Відповідь сервера після успішної авторизації користувача (JWT-токен)

Отримана відповідь свідчить про успішну авторизацію користувача в системі. Після перевірки коректності введених облікових даних сервер формує та повертає JWT-токен, який використовується для доступу до захищених маршрутів API. Токен містить ідентифікатор користувача та час його створення, що дозволяє ідентифікувати користувача при подальших запитах до системи.

Під час тестування безпеки вебзастосунку було виконано перевірку доступу до захищеного REST API методу (див. рис. 3.75).

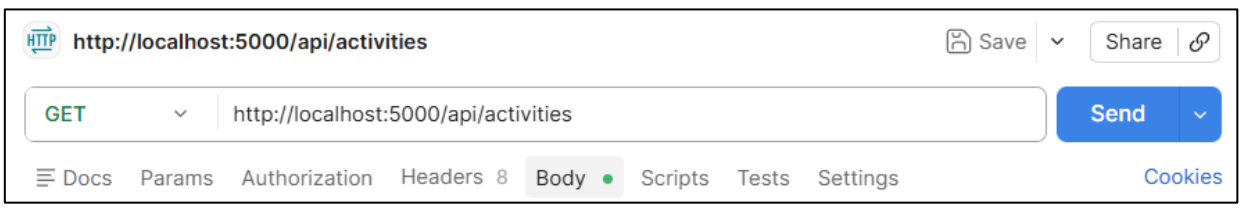


Рисунок 3.75 – Тестування REST API методу отримання списку фізичної активності користувача (/api/activities) у Postman без авторизації

Запит виконувався без передачі JWT-токена в заголовку Authorization, що дозволяє перевірити механізм захисту від неавторизованого доступу. Сервер повернув наступну відповідь, що показана на рисунку 3.76.

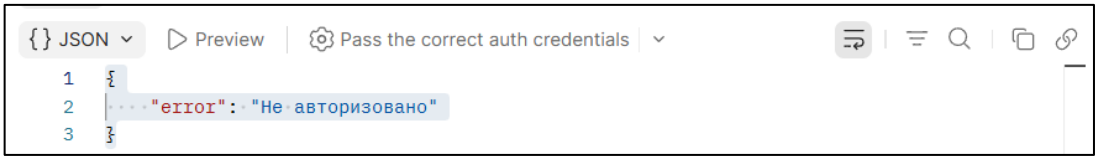


Рисунок 3.76 – Відповідь сервера при спробі доступу до захищеного ресурсу без JWT-токена (помилка авторизації)

Отримана відповідь підтверджує, що в системі коректно реалізовано механізм JWT-авторизації. При відсутності токена доступ до захищених маршрутів блокується за допомогою middleware, який перевіряє наявність заголовка Authorization. Таким чином, система запобігає несанкціонованому доступу до даних користувача та забезпечує базовий рівень безпеки API.

Під час тестування функціоналу вебзастосунку було перевірено роботу REST API методу додавання нової фізичної активності користувача (див. рис. 3.77).

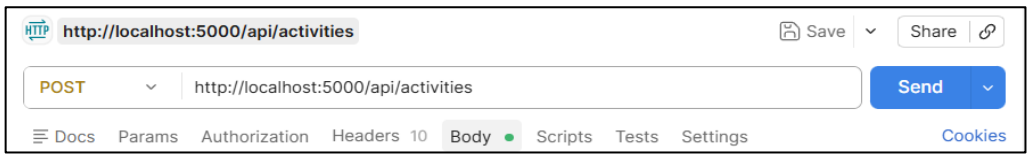


Рисунок 3.77 – Тестування REST API методу додавання фізичної активності користувача (/api/activities) у Postman

Запит виконувався через інструмент Postman із передачею JWT-токена в заголовку Authorization, що забезпечує доступ лише авторизованим користувачам системи. У тілі запиту передавалися наступні дані у форматі JSON (див. рис. 3.78).

Після виконання запиту сервер успішно обробив дані та створив новий запис у базі даних. У відповідь було повернуто об’єкт створеної активності разом з її унікальним ідентифікатором (див. рис. 3.79).



Рисунок 3.78 – Формування тіла POST-запиту у форматі JSON для створення запису фізичної активності користувача



Рисунок 3.79 – Відповідь сервера після успішного створення фізичної активності (повернення об’єкта з унікальним ідентифікатором)

Додана активність одразу відобразилася у вебзастосунку в розділі “Мої активності”, де користувач може переглядати список усіх записів, а також використовувати функції редагування та видалення (див. рис. 3.80). Під час тестування функціоналу вебзастосунку було перевірено роботу REST API методу отримання всіх фізичних активностей (див. рис. 3.81).



Рисунок 3.80 – Результат додавання фізичної активності та її відображення у вебзастосунку (інтерфейс користувача)

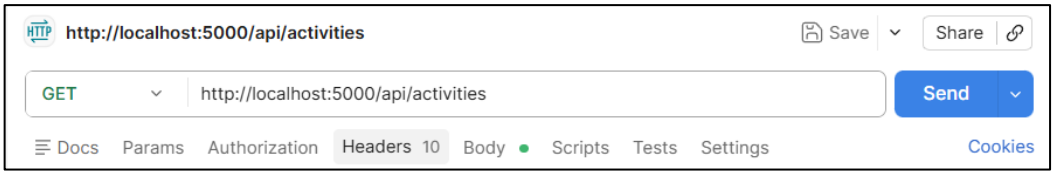


Рисунок 3.81 – Виконання GET-запиту до /api/activities із JWT-авторизацією

Даний запит є захищеним, тому для його виконання необхідно передати JWT-токен у заголовку Authorization. Токен отримується після успішної авторизації користувача в системі (див. рис. 3.82).

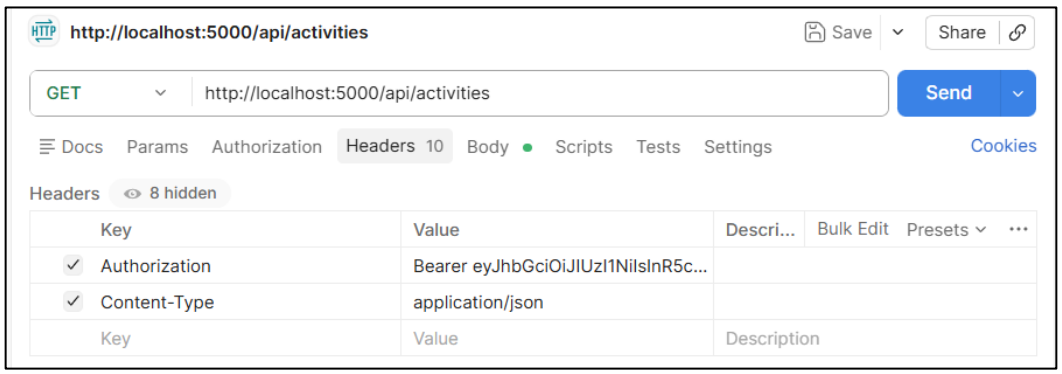


Рисунок 3.82 – Заголовки запиту з передачею токена

У результаті виконання запиту сервер повернув масив фізичних активностей користувача у форматі JSON, що представлено на рисунку 3.83.

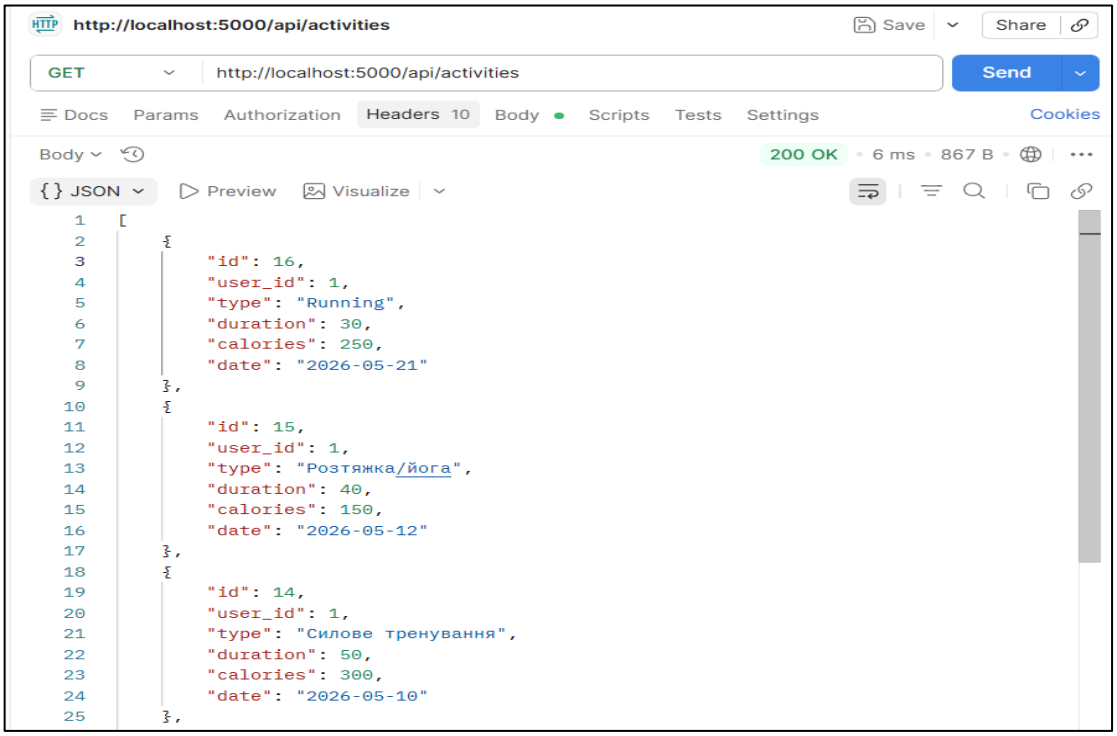


Рисунок 3.83 – Відповідь сервера у вигляді списку фізичних активностей

Під час тестування функціоналу вебзастосунку перевірено роботу REST API методу оновлення даних фізичної активності (див. рис. 3.84).

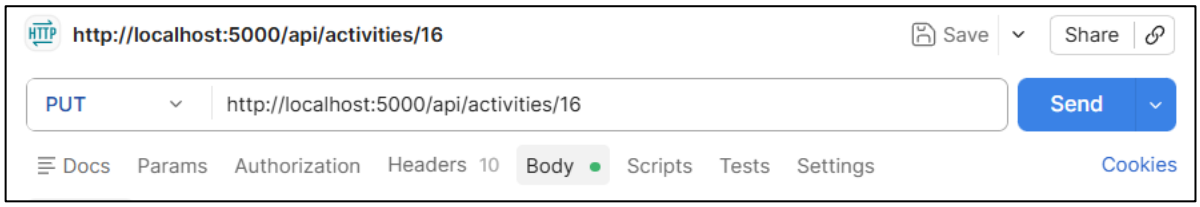


Рисунок 3.84 – Виконання PUT-запиту для оновлення фізичної активності

Даний запит використовується для редагування вже створеного запису про фізичну активність. Для виконання операції необхідно передати ідентифікатор запису (id) у URL-адресі, а також JWT-токен у заголовку Authorization, що забезпечує доступ лише авторизованим користувачам. У тілі запиту передаються оновлені дані у форматі JSON (див. рис. 3.85).



Рисунок 3.85 – Тіло запиту у форматі JSON для оновлення даних активності

Відповідь, яку повернув сервер у результаті виконання запиту, представлена на рисунку 3.86. Це підтверджує, що дані були успішно змінені та збережені у базі даних.

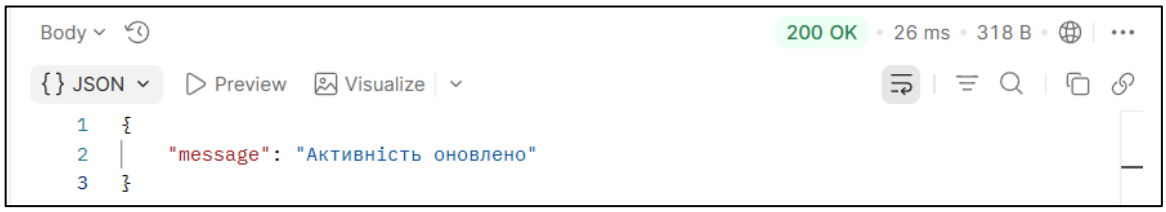


Рисунок 3.86 – Відповідь сервера після успішного оновлення запису

Після успішного виконання операції оновлення даних через REST API метод PUT /api/activities/:id зміни були автоматично відображені у вебінтерфейсі застосунку. Оновлена фізична активність з новими значеннями типу активності, тривалості, витрачених калорій та датою з’явилася у списку

активностей користувача (див. рис. 3.87), що підтверджує коректну роботу механізму синхронізації між серверною частиною та клієнтським інтерфейсом, правильне оновлення стану застосунку після виконання операції редагування.

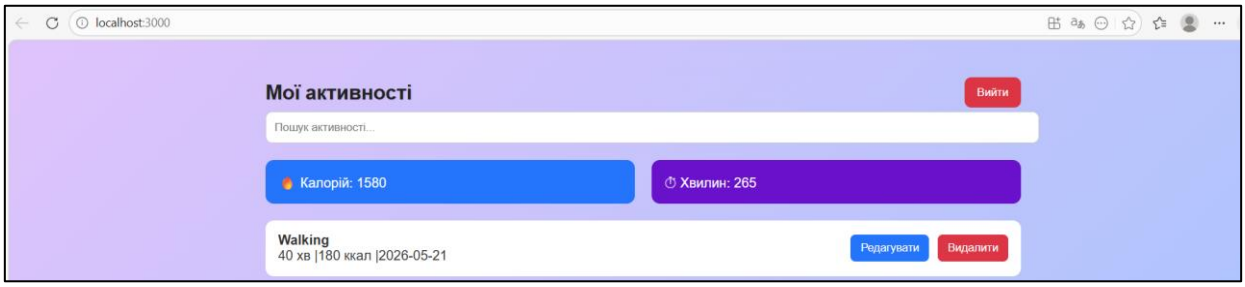


Рисунок 3.87 – Результат оновлення фізичної активності у вебзастосунку (відображення змінених даних в інтерфейсі користувача)

Під час тестування аналітичного модуля вебзастосунку було перевірено коректність обчислення та відображення статистичних даних фізичної активності користувача. На основі збережених записів система виконує автоматичне підсумовування ключових показників, зокрема загальної кількості витрачених калорій (totalCalories) та загальної тривалості активностей (totalDuration) (див. рис. 3.88).

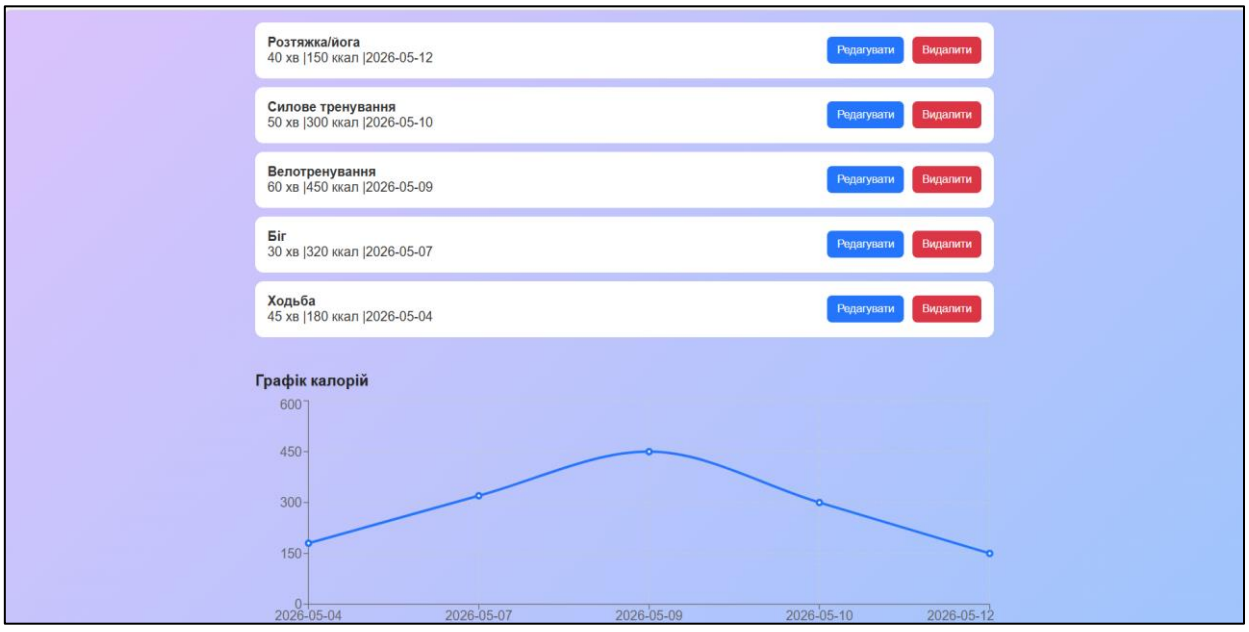


Рисунок 3.88 – Відображення аналітики фізичної активності користувача

Додатково було протестовано формування набору даних (chartData), який використовується для побудови графіка розподілу калорій за датами за допомогою бібліотеки Recharts. За результатами тестування отримано, що всі значення статистики обчислюються коректно, дані правильно групуються за датами, графік відображає динаміку фізичної активності користувача та інтерфейс оновлюється автоматично після змін у даних.

Таким чином, аналітичний модуль працює стабільно та забезпечує наочне відображення результатів фізичної активності користувача.

Під час негативного тестування вебзастосунку було перевірено поведінку системи при введенні некоректних або неповних даних при реєстрації та авторизації користувача. Основною метою даного етапу було перевірити коректність роботи серверної валідації та стійкість REST API до помилкових запитів. Протестовано три сценарії: реєстрація користувача без обов'язкового поля email (див. рис. 3.89, 3.90), авторизація без введення пароля (див. рис. 3.91, 3.92) і авторизація з неправильними обліковими даними (див. рис. 3.93, 3.94).

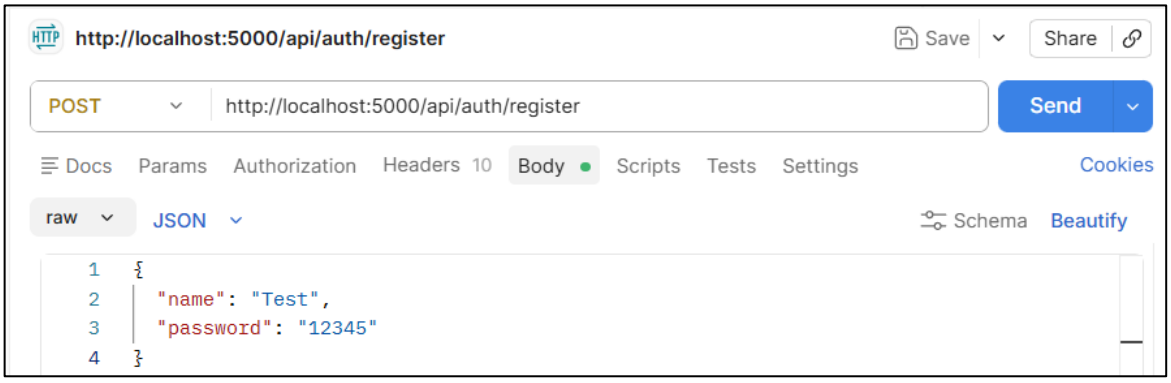


Рисунок 3.89 – Виконання POST-запиту на реєстрацію користувача без обов'язкового поля email



Рисунок 3.90 – Відповідь сервера на некоректні дані реєстрації

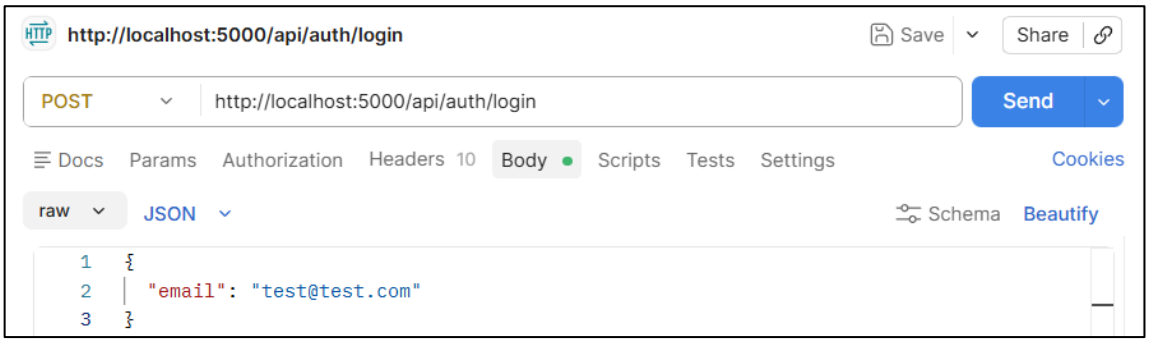


Рисунок 3.91 – Виконання POST-запиту на авторизацію без введення пароля

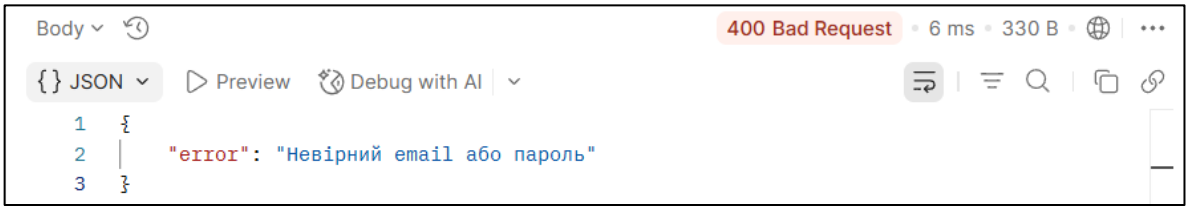


Рисунок 3.92 – Відповідь сервера при спробі авторизації з неповними даними



Рисунок 3.93 – Виконання POST-запиту на авторизацію
з некоректними обліковими даними

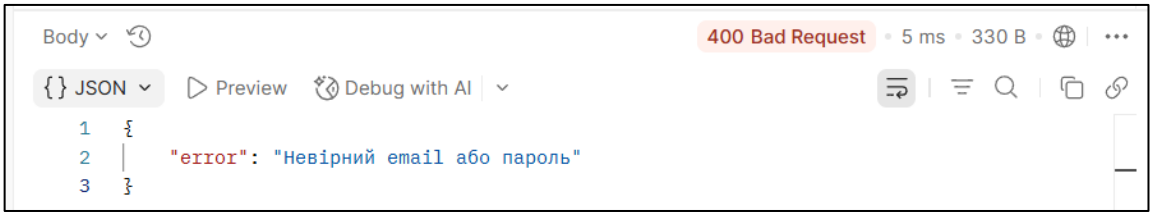


Рисунок 3.94 – Відповідь сервера “Невірний email або пароль”

У всіх випадках система відхиляла запити та повертала відповідні повідомлення про помилки, що підтверджує правильну реалізацію механізму валідації та захисту API від некоректного використання. Негативне тестування

підтвердило, що система коректно обробляє помилкові запити та забезпечує базовий рівень захисту даних користувача.

Наступним етапом перевірки якості розробленого вебзастосунку стало Unit-тестування, спрямоване на оцінку взаємодії між окремими модулями системи, з використанням бібліотеки Jest. Головна мета – перевірка коректності виконання базових обчислювальних операцій і правил валідації без розгортання повноцінного серверного середовища. Unit-тестування ізольовано оцінює роботу окремих функцій системи, що сприяє оперативному виявленню, усуненню помилок ще на початкових стадіях розробки.

Перший тест був спрямований на перевірку коректності реалізації функції розрахунку індексу маси тіла (BMI) і представлено на рисунку 3.95. Отримане значення порівнювалося з еталонним результатом із урахуванням допустимого відхилення, що є необхідним для обчислень із числами з плаваючою комою. Проведений тест підтвердив правильність роботи алгоритму визначення індексу маси тіла.

```
1  function calcBMI(weight, height) {
2    return weight / (height * height);
3  }
4
5  test('BMI calculation works correctly', () => {
6    const result = calcBMI(70, 1.75);
7    expect(result).toBeCloseTo(22.86);
8  });
```

Рисунок 3.95 – Unit-тест функції розрахунку індексу маси тіла (BMI)

Другий тест спрямований на перевірку базової валідації електронної пошти користувача. У його межах здійснюється контроль наявності символу “@”, який є обов’язковою ознакою коректного формату email-адреси. Проведене тестування підтвердило правильність реалізації базових правил перевірки користувацьких даних у системі і представлено на рисунку 3.96.

```
1  test('email should contain @', () => {
2    const email = "test@mail.com";
3    expect(email).toContain("@");
4  });
```

Рисунок 3.96 – Unit-тест для валідації електронної пошти

Третій тест спрямований на перевірку коректності логіки обробки даних фізичної активності користувача (див. рис. 3.97). Зокрема, контролюється значення тривалості тренування, що має бути більшим за нуль, та забезпечує недопущення некоректних або від'ємних значень, які б могли б спотворити результати аналітики в системі.

```
1  test('activity duration must be positive', () => {
2    const activity = { duration: 30 };
3
4    expect(activity.duration).toBeGreaterThan(0);
5  });
```

Рисунок 3.97 Unit-тест перевірки даних фізичної активності користувача

На рисунку 3.98 представлено результат запуску автоматизованих unit-тестів, реалізованих за допомогою фреймворку Jest. Після виконання команди тестування система успішно пройшла всі тестові сценарії без виявлення помилок.

```
> backend@1.0.0 test
> jest

 PASS  tests/activity object.test.js
 PASS  tests/login validation.test.js
 PASS  tests/bmi.test.js

Test Suites: 3 passed, 3 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.561 s, estimated 1 s
Ran all test suites.
```

Рисунок 3.98 – Результати виконання unit-тестів у середовище Jest

Зокрема, було перевірено три основні модулі:

– тестування об'єкта фізичної активності (activity_object.test.js);
– валідація даних авторизації користувача (login validation.test.js);
– перевірка функції розрахунку індексу маси тіла (bmi.test.js).

Усі тестові набори виконані успішно: 3 з 3 тестових suite пройдено, загалом виконано 3 з 3 тестів без помилок. Система не використовує snapshot-

тестування (0 snapshots), а загальний час виконання становив приблизно 0.56 секунди, що свідчить про високу швидкість та ефективність тестового середовища.

Таким чином, проведене unit-тестування засвідчило коректне функціонування основних компонентів вебзастосунку. Застосування фреймворку Jest дало змогу перевірити роботу обчислювальних і валідаційних модулів на ізольованому рівні, що сприяє підвищенню стабільності програмного продукту та зниженню ймовірності виникнення помилок у процесі його використання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено вебзастосунок для відстеження та аналітики фізичної активності користувача, що дозволяє здійснювати збір, збереження, обробку та візуалізацію даних про фізичну активність у зручному інтерактивному форматі.

Проведено аналіз предметної області, досліджено сучасні підходи до розробки веборієнтованих систем моніторингу фізичної активності та визначено ключові принципи побудови подібних інформаційних систем. На основі проведеного аналізу сформовано загальні функціональні та технічні вимоги до програмного продукту, які стали основою для подальшої реалізації системи.

Спроєктовано архітектуру вебзастосунку на основі клієнт-серверного підходу, розроблено структуру бази даних та визначено механізми взаємодії між компонентами системи через REST API. Це забезпечує чіткий розподіл функціональності між клієнтською та серверною частинами, ефективну обробку даних, а також масштабованість і зручність подальшого розвитку вебзастосунку.

Реалізовано функціональний вебзастосунок із використанням React на клієнтській стороні, Node.js та Express на серверній частині, а також SQLite для збереження даних. Додатково впроваджено модулі авторизації, аналітики та візуалізації фізичної активності користувача, а також функціональний модуль для розрахунку індексу маси тіла (BMI), що дозволяє користувачу отримувати базову оцінку свого фізичного стану.

Проведене тестування підтвердило коректність і стабільність роботи системи та відповідність реалізованого функціоналу поставленим завданням.

ПЕРЕЛІК ПОСИЛАНЬ

1. Безверхий О., Куценко О. Ефективність застосування бібліотеки React. *Інформаційні технології та суспільство.* 2022. № 2 (4). С. 13–19. URL: https://journals.maup.com.ua/index.php/it/article/view/2096 (дата звернення: 09.04.2026).

2. Arias D. Hashing in action: Understanding bcrypt. *arXiv.* 2021. P. 1–10. URL: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ (дата звернення: 09.04.2026).

3. Provos N., Mazieres D. A Future-Adaptable Password Scheme. *1999 USENIX Annual Technical Conference* : proceedings of the FREENIX Track (Monterey, California, USA, June 6–11, 1999). 13 p. URL: https://www.usenix.org/legacy/events/usenix99/provos/provos.pdf (дата звернення: 09.04.2026).

4. Bucko A., Vishi K., Krasniqi B., Rexha B. Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History. *Computers.* 2023. Vol. 12(4). URL: https://www.mdpi.com/2073-431X/12/4/78 (дата звернення: 27.04.2026).

5. Case M. A., Burwick H. A., Volpp K. G., Patel M. S. Accuracy of Smartphone Applications and Wearable Devices for Tracking Physical Activity Data. *JAMA.* 2015. Vol. 313(6). P. 625–626. URL: https://jamanetwork.com/journals/jama/fullarticle/2089355 (дата звернення: 27.03.2026).

6. Cho H., Slade P. OpenMetabolics: Estimating energy expenditure using a smartphone worn in a pocket. *Communications Engineering.* 2026. 5(1). P. 35. URL: https://pubmed.ncbi.nlm.nih.gov/41651983/ (дата звернення: 07.04.2026).

7. Client–Server / REST основа. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 07.04.2026).

8. Díaz-Gómez J. M., Quiceno-Rua E., David C. Correa-Álvarez Protocols, Reactive Architectures, and Computing Platforms for Low-Latency, High-Concurrency Web Applications: A Systematic Literature Review. *MDPI Future Internet*. 2026. 18(5). P. 254. URL: https://www.mdpi.com/1999-5903/18/5/254 (дата звернення: 09.04.2026).

9. Van Steen M., Tanenbaum A. S. A brief introduction to distributed systems. Computing. 2016. Vol. 98. P. 967–1009. URL: https://doi.org/10.1007/s00607-016-0508-7 (дата звернення: 09.04.2026).

10. Etese O., Adesina A. A Review and Comparative Analysis of Password Hashing Techniques: Evaluating Bcrypt and Argon2. *SSRN*. 2025. 8 p. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5363433 (дата звернення: 09.04.2026).

11. Express.js. Офіційна документація Express.js. URL: https://expressjs.com/ (дата звернення: 22.03.2026).

12. Few S. Data Visualization for Human Perception. *The Encyclopedia of Human-Computer Interaction.* 2013. URL: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception (дата звернення: 27.04.2026).

13. Grudniak M., Dzieńkowski M. REST API performance comparison of web applications based on JavaScript programming frameworks. *Journal of Computer Sciences Institute*. 2021. № 19. P. 121–125. URL: https://ph.pollub.pl/index.php/jcsi/article/view/2620 (дата звернення: 09.04.2026).

14. Het R., Masharu K. BMI Tracker: A Personalized Health Monitoring Web Application. *International Journal of Scientific Research and Engineering Development.* 2025. Vol. 8, Iss. 5. P. 1091–1096. URL: https://ijsred.com/volume8/issue5/IJSRED-V8I5P126.pdf (дата звернення 11.04.2026).

15. JWT стандарт. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 28.03.2026).

16. JWT.io. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення: 28.03.2026).

17. Kristoffersson A., Lindén M. Affiliations Expand A Systematic Review of Wearable Sensors for Monitoring Physical Activity. *Sensors (Basel)*. 2022. Vol. 22(2). P. 573. URL: https://www.mdpi.com/1424-8220/22/2/573 (дата звернення: 07.05.2026).

18. Miłosierny B., Dzieńkowski M. The comparative analysis of web applications frameworks in the Node.js Ecosystem. *Journal of Computer Sciences Institute*. 2021. № 18. P. 42–48. URL: https://ph.pollub.pl/index.php/jcsi/article/view/2423 (дата звернення: 09.04.2026).

19. Ming M., Buxi L., Ye Y., Peng K. et al. A machine learning framework for personalized exercise prescription based on BMI and physical fitness assessment. *Sci Rep.* 2026. Vol. 16(1). P. 13336. URL: https://www.nature.com/articles/s41598-026-42405-2 (дата звернення: 07.05.2026).

20. Mozilla Developer Network (MDN Web Docs). HTTP and Web APIs. URL: https://developer.mozilla.org/ (дата звернення: 05.04.2026).

21. Nadukuda N. A Cloud-Native Framework for Scalable Web Applications Using React, Node.js, and Event-Driven Microservices *International Journal of Computational and Experimental Science and ENgineering (IJCESEN)*. 2025. Vol. 11(4). P. 4266–4273. URL: https://ijcesen.com/index.php/ijcesen/article/view/5006 (дата звернення 11.04.2026).

22. Node.js. Офіційна документація Node.js. URL: https://nodejs.org/en/docs (дата звернення: 18.03.2026).

23. Ntantogian C., Malliaros S., Xenakis C. Evaluation of password hashing schemes in open source web platforms. *Computers & Security*. 2019. № 84. P. 206–224. URL: https://www.cyberwatching.eu/sites/default/files/Passwords-sub-v26.pdf (дата звернення: 18.03.2026).

24. Obiri-Tetteh J. A. et al. Bcrypt-Based Optimized Password Hashing Against Targeted Internet of Things Attacks. *Cureus Journal of Computer Science*. 2025. Article ID: 5167. URL: https://www.cureusjournals.com/articles/5167-bcrypt-based-optimized-password-hashing-against-targeted-internet-of-things-attacks#!/ (дата звернення: 11.04.2026).

25. Owen N., Healy G. N., Matthews C. E., Dunstan D. W. Too much sitting: the population-health science of sedentary behavior. *Exercise and Sport Sciences Reviews.* 2010. Vol. 38(3). P. 105–113. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3404815/ (дата звернення: 27.04.2026).

26. Pimentel V., Nickerson B. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing.* 2012. Vol. 16(4). P. 45–53. URL: https://ieeexplore.ieee.org/document/6201219 (дата звернення: 27.04.2026).

27. Piwek L., Ellis D. A., Andrews S., Joinson A. The Rise of Consumer Health Wearables: Promises and Barriers. *PLOS Medicine.* 2016. Vol. 13(2). URL: https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1001953 (дата звернення: 27.04.2026).

28. Putra F. P. E., Efendi R. W., Tamam A. B., Pramadi W.A. Trends and Best Practices in API-Based Web Development Using Laravel and React. *Brilliance: Research of Artificial Intelligence.* 2025. Vol. 5. No. 1. P. 223–233. URL: https://jurnal.itscience.org/index.php/brilliance/article/view/5971/4521 (дата звернення 11.04.2026).

29. React офіційна документація. URL: https://react.dev/learn (дата звернення: 15.03.2026).

30. Recharts. Documentation for React Chart Library. URL: https://recharts.org/en-US/ (дата звернення: 10.04.2026).

31. REST / web architecture (Fielding). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 10.04.2026).

32. REST API Tutorial. RESTful API Principles and Architecture. URL: https://restfulapi.net/ (дата звернення: 01.04.2026).

33. Skanda C., Srivatsa B., Premananda B. S. Secure Hashing using BCrypt for Cryptographic Applications. *IEEE NKCon Conference*. 2022. P. 1–6. URL: https://archive.cps-vo.org/node/97391. (дата звернення: 09.04.2026).

34. SQLite. Офіційна документація SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 25.03.2026).

35. Tekinerdogan B., Verdouw C. Systems Architecture Design Pattern Catalog for Developing Digital Twins. *Sensors*. 2020. Vol. 20(18). URL: https://www.mdpi.com/1424-8220/20/18/5103 (дата звернення: 25.04.2026).

36. Warburton D. E. R., Nicol C. W., Bredin S. S. D. Health benefits of physical activity: the evidence. *Canadian Medical Association Journal*. 2006. Vol. 174(6). P. 801–809. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1402378/ (дата звернення: 27.04.2026).

37. WHO BMI. URL: https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight (дата звернення: 27.04.2026).

38. Yiting G. Practical Approaches to Developing High-performance Web Applications Basedon React. *Frontiers in Science and Engineering*. 2025. № 2. P. 99–105. URL: https://doi.org/10.54691/5ccss512 (дата звернення: 09.04.2026).