Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему **ВЕБПЛАТФОРМА ПЕРСОНАЛІЗОВАНОГО ФІТНЕСКОУЧИНГУ З АНАЛІЗОМ БІОМЕТРИЧНИХ ДАНИХ**

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

НАЛИВАЙКО А. Ю.
(ПРІЗВИЩЕ та ініціали)
Керівник КУДЕРМЕТОВ Р.К.
(ПРІЗВИЩЕ та ініціали)
Рецензент РОМАНОВСЬКИЙ А.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістерський

Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) «Комп'ютерні системи та мережі»
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.

"15" жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

НАЛИВАЙКА Артура Юрійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Вебплатформа персоналізованого фітнескоучингу з аналізом біометричних даних
керівник проєкту (роботи) к. т. н., доцент, КУДЕРМЕТОВ Равіль Камілович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від " 30 " жовтня 2025 року № 491

2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року

3. Вихідні дані до проєкту (роботи) персональні комп'ютери та мобільні пристрої з сучасними веббраузерами, стандарти ВООЗ для класифікації ІМТ

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз предметної області та постановка задачі;
2) Проєктування архітектури системи;
3) Реалізація інформаційної платформи;
4) Опис користувацького інтерфейсу;
5) Апаратні вимоги до системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	КУДЕРМЕТОВ Р. К., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області та постановка задачі	10.10.2025 р.	
2	Розробка архітектури системи	15.10.2025 р.	
3	Налагодження програмного забезпечення	20.10.2025 р.	
4	Реалізація архітектури системи	05.10.2025 р.	
5	Оформлення отриманих результатів у ПЗ	15.11.2025 р.	
6	Оформлення графічного матеріалу	25.11.2025 р.	
7	Оформлення допоміжного матеріалу	23.12.2025 р.	

Студент _____ **Артур НАЛИВАЙКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Равіль КУДЕРМЕТОВ**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 94 стор., 8 рис., 2 дод., 27 джерел.

FASTAPI, MONGODB, REACT, АДАПТИВНІ ТРЕНУВАЛЬНІ ПРОГРАМИ, АНАЛІЗ ЗДОРОВ'Я, БІОМЕТРИЧНІ ДАНІ, ВЕБ-ПЛАТФОРМА, ПЕРСОНАЛІЗОВАНИЙ ФІТНЕС-КОУЧИНГ

Об'єкт дослідження – процес автоматизації персоналізованого фітнес-коучингу з використанням інтелектуального аналізу біометричних показників.

Предмет дослідження – методи та засоби створення веб-платформи для генерації та адаптації індивідуальних тренувальних програм на основі біометричних даних користувачів.

Мета роботи – розробка інформаційної платформи персоналізованого фітнес-коучингу з алгоритмами автоматичної адаптації тренувальних навантажень відповідно до індивідуальних характеристик та поточного стану користувача.

Проаналізовано предметну область, розроблено математичні моделі на основі формул Міффліна-Сан Жеора та Карвонена. Спроєктовано мікросервісну архітектуру на стеку FastAPI-MongoDB-React. Реалізовано REST API з JWT-аутентифікацією та адаптивний SPA з українською локалізацією. Розроблено інтерфейс за принципом mobile-first. Визначено технічні вимоги до апаратного забезпечення та серверного розгортання.

ABSTRACT

Explanatory note to the master's work: 94 pages, 8 figures, 2 additions, 27 sources.

ADAPTIVE TRAINING PROGRAMS, BIOMETRIC DATA, FASTAPI, HEALTH ANALYSIS, MONGODB, PERSONALIZED FITNESS COACHING, REACT, WEB PLATFORM

Research object – the process of automating personalized fitness coaching using intelligent analysis of biometric indicators.

Research subject – methods and tools for creating a web platform for generating and adapting individual training programs based on user biometric data.

Research goal – development of a personalized fitness coaching information platform with algorithms for automatic adaptation of training loads according to individual characteristics and current user condition.

The domain was analyzed, mathematical models based on Mifflin-St Jeor and Karvonen formulas were developed. Microservices architecture designed using FastAPI-MongoDB-React stack. REST API with JWT authentication and adaptive SPA with Ukrainian localization implemented. Mobile-first interface developed. Technical requirements for hardware and server deployment defined.

ЗМІСТ

ВСТУП

У сучасних умовах розвитку інформаційних технологій та зростання уваги суспільства до питань здоров’я і фізичної активності особливої актуальності набувають інформаційні системи, орієнтовані на персоналізований підхід до підтримки фізичної форми людини. Недостатній рівень рухової активності, нерегулярні тренування та відсутність індивідуально підібраних навантажень негативно впливають на стан здоров’я населення та підвищують ризик розвитку хронічних захворювань. За цих умов використання сучасних веб-технологій і методів аналізу біометричних даних створює передумови для підвищення ефективності фітнес-тренувань та формування сталих здорових звичок.

Наявні фітнес-додатки та платформи здебільшого обмежуються базовим відстеженням фізичної активності, підрахунком калорій або наданням типових тренувальних програм без урахування індивідуальних особливостей користувача. Відсутність глибокої персоналізації, недостатнє використання біометричних показників та обмежені можливості автоматичної адаптації тренувальних навантажень знижують ефективність таких рішень. Це зумовлює потребу у створенні інтелектуальних інформаційних платформ, здатних аналізувати стан користувача та формувати індивідуальні рекомендації на основі науково обґрунтованих методик.

Актуальність даної роботи полягає у розробці веб-платформи персоналізованого фітнес-коучингу, яка поєднує сучасні технології веб-розробки, аналіз біометричних даних та математичні моделі розрахунку тренувальних навантажень. Застосування архітектури на основі FastAPI, MongoDB та React дозволяє створити масштабовану, продуктивну та зручну у використанні систему, орієнтовану на широке коло користувачів.

Об’єктом дослідження є процес автоматизації персоналізованого фітнес-коучингу з використанням інтелектуального аналізу біометричних показників. Предметом дослідження є методи та засоби створення веб-платформи для генерації

та адаптації індивідуальних тренувальних програм на основі біометричних даних користувачів.

Метою кваліфікаційної роботи є розробка інформаційної платформи персоналізованого фітнес-коучингу з алгоритмами автоматичної адаптації тренувальних навантажень відповідно до індивідуальних характеристик та поточного стану користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасний стан розвитку інформаційних платформ фітнес-коучингу

1.1.1 Постановка проблеми

У сучасному світі спостерігається стійка тенденція до зростання інтересу населення до здорового способу життя та фізичної активності. За даними Всесвітньої організації охорони здоров'я, недостатня фізична активність є четвертим провідним фактором ризику глобальної смертності [1]. Одночасно з цим розвиток інформаційних технологій відкриває нові можливості для створення персоналізованих рішень у сфері фітнесу та здоров'я.

Аналіз ринку показує, що індустрія фітнес-додатків активно розвивається. Проте більшість наявних рішень мають суттєві обмеження:

– відсутність глибокої персоналізації тренувальних програм на основі біометричних показників користувача;

– недостатня інтеграція з пристроями для моніторингу здоров'я;

– відсутність науково обґрунтованих алгоритмів для адаптації навантажень;

– обмежені можливості для комплексного аналізу прогресу користувача.

1.1.2 Огляд наявних рішень

Дослідження сучасних фітнес-платформ виявило декілька основних категорій додатків. Першу групу складають загальні фітнес-трекери, такі як MyFitnessPal або Fitbit, які зосереджуються на підрахунку калорій та базовому відстеженні активності. Однак вони не надають персоналізованих тренувальних програм на основі біометричних даних.

Другу категорію представляють спеціалізовані тренувальні додатки, такі як Nike Training Club або Adidas Training. Ці платформи пропонують структуровані тренувальні програми, проте персоналізація обмежується лише вибором рівня складності та типу активності.

Третю групу становлять професійні рішення для тренерів та фітнес-центрів. Такі системи зазвичай мають широкі можливості для створення індивідуальних програм, але вимагають участі професійного тренера і не доступні для самостійного використання.

1.1.3 Аналіз біометричних параметрів для персоналізації

Ефективність персоналізованого фітнес-коучингу безпосередньо залежить від якості та повноти аналізу біометричних даних користувача. Сучасні технології дозволяють збирати широкий спектр показників:

– базові антропометричні дані: вага, зріст, індекс маси тіла;

– показники серцево-судинної системи: частота серцевих скорочень у спокої та під навантаженням;

– параметри фізичної активності: кількість кроків, витрачені калорії, тривалість активності;

– показники якості відпочинку: тривалість та якість сну.

Комплексний аналіз цих параметрів дозволяє сформувати детальний профіль користувача та забезпечити максимально ефективну персоналізацію тренувальних програм.

1.2 Теоретичні основи персоналізації фітнес-програм

1.2.1 Принципи спортивного тренування

Розробка ефективних алгоритмів персоналізації потребує глибокого розуміння фізіологічних процесів та принципів спортивної медицини. Основою для створення персоналізованих програм є теорія адаптації організму до фізичних навантажень. Сучасна теорія спортивного тренування базується на декількох фундаментальних принципах. Принцип індивідуалізації передбачає врахування індивідуальних особливостей організму при плануванні тренувальних навантажень. Принцип прогресивності вимагає поступового збільшення

навантажень для забезпечення постійної адаптації організму. Принцип специфічності означає, що тренувальні вправи повинні відповідати поставленим цілям користувача. Принцип періодизації передбачає циклічність тренувального процесу з чергуванням навантажень різної інтенсивності [2].

1.2.2 Математичні моделі для розрахунку навантажень

Для автоматизованого розрахунку оптимальних навантажень використовуються математичні моделі, що враховують індивідуальні параметри користувача [3, 4, 17, 18]. Базовим є розрахунок максимальної частоти серцевих скорочень за формулою:

$$ЧССмакс = 220 – вік. \quad (1.1)$$

де ЧССмакс – максимальна частота серцевих скорочень (ударів на хвилину);

вік – вік користувача в повних роках.

Для визначення тренувальних зон використовується метод Карвонена:

$$ЧСС\ тренувальна = (ЧССмакс - ЧСС\ спокою) \times інтенсивність + ЧСС\ спокою, \quad (1.2)$$

де ЧССтренувальна – цільова частота серцевих скорочень під час тренування (уд/хв);

ЧССмакс – максимальна частота серцевих скорочень (уд/хв);

ЧССспокою – частота серцевих скорочень у стані спокою (уд/хв);

Інтенсивність – коефіцієнт інтенсивності навантаження (від 0 до 1).

Розрахунок базового метаболізму здійснюється за рівнянням Міффліна-Сен Жеора, що дозволяє визначити оптимальний калоражний баланс для досягнення поставлених цілей.

1.3 Вимоги до функціональності системи

1.3.1 Функціональні вимоги

Аналіз предметної області дозволяє сформулювати основні функціональні вимоги до інформаційної платформи персоналізованого фітнес-коучингу.

Система повинна забезпечувати наступний функціонал:

– реєстрацію та авторизацію користувачів з різними ролями доступу;
– збір та збереження біометричних параметрів користувачів;
– автоматичну генерацію персоналізованих тренувальних програм;
– моніторинг прогресу та адаптацію програм відповідно до результатів;
– візуалізацію статистичних даних та аналітичних звітів;
– інтеграцію з зовнішніми пристроями моніторингу здоров'я.

1.3.2 Нефункціональні вимоги

До нефункціональних вимог відносяться:

– забезпечення безпеки персональних даних користувачів;
– масштабованість системи для обслуговування великої кількості користувачів;
– високий рівень доступності системи;
– зручність використання інтерфейсу для користувачів різного рівня технічної підготовки.

1.4 Постановка задачі дослідження

1.4.1 Основні задачі дослідження

На основі проведеного аналізу сформульована мета дослідження: розробка веб-платформи персоналізованого фітнес-коучингу з інтелектуальним аналізом

біометричних даних для автоматичної генерації індивідуальних тренувальних програм.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– дослідити наявні підходи до персоналізації фітнес-програм та виявити їх недоліки;

– розробити математичні моделі для розрахунку оптимальних тренувальних навантажень на основі біометричних параметрів;

– спроєктувати архітектуру інформаційної системи з урахуванням вимог масштабованості та надійності;

– реалізувати алгоритми обробки та аналізу біометричних даних;

– створити веб-інтерфейс для взаємодії користувачів з системою;

– провести експериментальне дослідження ефективності розробленої платформи.

1.4.2 Очікувані результати

Результатом виконання роботи має стати функціональна веб-платформа, що включає:

– систему збору та аналізу біометричних даних користувачів;

– алгоритми автоматичної генерації персоналізованих тренувальних програм;

– веб-інтерфейс для управління тренувальним процесом;

– систему моніторингу прогресу та адаптації програм.

Платформа повинна забезпечувати підвищення ефективності тренувального процесу за рахунок індивідуального підходу до кожного користувача та автоматичної адаптації навантажень відповідно до поточного стану та прогресу [27].

1.5 Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області персоналізованого фітнес-коучингу та обґрунтовано актуальність розробки спеціалізованої інформаційної платформи.

Аналіз сучасного стану розвитку фітнес-додатків виявив суттєві недоліки існуючих рішень, серед яких відсутність глибокої персоналізації тренувальних програм на основі біометричних показників користувача, недостатня інтеграція з пристроями моніторингу здоров'я, відсутність науково обґрунтованих алгоритмів для автоматичної адаптації навантажень та обмежені можливості для комплексного аналізу прогресу. Огляд наявних рішень показав, що загальні фітнес-трекери зосереджуються лише на підрахунку калорій та базовому відстеженні активності, спеціалізовані тренувальні додатки обмежуються вибором рівня складності без урахування індивідуальних біометричних параметрів, а професійні рішення для тренерів вимагають постійної участі фахівця і не доступні для самостійного використання.

Дослідження теоретичних основ персоналізації фітнес-програм виявило фундаментальні принципи спортивного тренування, що мають бути покладені в основу алгоритмів системи. До них належать принцип індивідуалізації з урахуванням особливостей організму, принцип прогресивності для забезпечення постійної адаптації, принцип специфічності відповідно до поставлених цілей та принцип періодизації з чергуванням навантажень різної інтенсивності. Розроблено математичні моделі для автоматизованого розрахунку оптимальних навантажень, включаючи формули визначення максимальної частоти серцевих скорочень, метод Карвонена для розрахунку тренувальних зон та рівняння Міффліна-Сан Жеора для визначення базового метаболізму.

На основі проведеного аналізу сформульовано вимоги до функціональності системи, що включають реєстрацію та авторизацію користувачів з різними ролями доступу, збір та збереження біометричних параметрів, автоматичну генерацію

персоналізованих тренувальних програм, моніторинг прогресу з адаптацією програм відповідно до результатів, візуалізацію статистичних даних та інтеграцію з зовнішніми пристроями. Нефункціональні вимоги охоплюють забезпечення безпеки персональних даних, масштабованість системи для обслуговування великої кількості користувачів, високий рівень доступності та зручність використання інтерфейсу.

Чітко сформульована мета дослідження полягає у розробці веб-платформи персоналізованого фітнес-коучингу з інтелектуальним аналізом біометричних даних для автоматичної генерації індивідуальних тренувальних програм. Для досягнення цієї мети визначено комплекс задач, що включають дослідження наявних підходів до персоналізації, розробку математичних моделей розрахунку навантажень, проєктування масштабованої архітектури, реалізацію алгоритмів обробки біометричних даних, створення веб-інтерфейсу та проведення експериментального дослідження ефективності платформи.

Результати аналізу предметної області підтверджують актуальність та практичну значущість розробки спеціалізованої інформаційної платформи, що здатна автоматизувати процес створення персоналізованих фітнес-програм на основі науково обґрунтованих методик з урахуванням індивідуальних біометричних показників користувачів.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Архітектурні рішення для веб-платформ

2.1.1 Порівняння архітектурних підходів

Проєктування архітектури інформаційної системи є надважливим етапом, що визначає подальшу масштабованість, надійність та продуктивність платформи. Сучасні підходи до розробки веб-додатків передбачають використання різних архітектурних патернів залежно від специфіки задач та очікуваних навантажень.

Розглянемо основні архітектурні підходи для розробки веб-платформ. Монолітна архітектура передбачає розміщення всієї логіки додатку в єдиному розгортуваному компоненті. Переваги цього підходу полягають у простоті розробки та розгортання для невеликих проєктів. Проте монолітна архітектура має суттєві недоліки: складність масштабування окремих компонентів, залежність усієї системи від однієї технології, ризик повного відмови системи при збої одного модуля.

Мікросервісна архітектура розділяє додаток на незалежні сервіси, кожен з яких відповідає за конкретну бізнес-функцію. Такий підхід забезпечує гнучкість у виборі технологій для різних компонентів, незалежне масштабування сервісів та підвищену стійкість системи до відмов. Однак мікросервіси вимагають більш складного управління та моніторингу.

Для розроблюваної фітнес-платформи обрано мікросервісну архітектуру через необхідність обробки різнотипних даних та потребу в гнучкому масштабуванні окремих компонентів.

2.1.2 Компоненти системи

Архітектура платформи включає наступні основні компоненти:

– сервіс авторизації та управління користувачами, що забезпечує реєстрацію, аутентифікацію та авторизацію;

– сервіс обробки біометричних даних для збору, валідації та аналізу показників здоров'я;

– сервіс генерації тренувальних програм, що реалізує алгоритми персоналізації;

– сервіс аналітики для формування звітів та візуалізації прогресу;

– API-шлюз для централізованого управління запитами та забезпечення безпеки;

– веб-інтерфейс користувача для взаємодії з системою.

2.2 Вибір технологічного стеку

2.2.1 Backend технології

Вибір технологій для реалізації системи базується на аналізі переваг та недоліків різних варіантів з урахуванням специфіки задач персоналізованого фітнес-коучингу.

Для серверної частини розглянуто кілька варіантів фреймворків та мов програмування.

Node.js з Express.js є популярним вибором для веб-додатків. Переваги: висока швидкість розробки, єдина мова для frontend та backend, великий екосистема пакетів NPM. Недоліки: однопотокова модель може бути обмежуючою для CPU-інтенсивних обчислень, менша продуктивність для математичних операцій порівняно з компільованими мовами.

Python з Django/Flask забезпечує швидкість розробки та читабельність коду. Django надає багато вбудованих можливостей, проте може бути надмірно складним для API-орієнтованих додатків. Flask є більш гнучким, але вимагає додаткового налаштування.

Python з FastAPI поєднує простоту Python з високою продуктивністю [5]. Переваги FastAPI включають: автоматичну генерацію документації OpenAPI, вбудовану підтримку асинхронного програмування, автоматичну валідацію даних через Pydantic, високу продуктивність, порівнянну з Node.js.

Для розроблюваної системи обрано FastAPI через наступні фактори:

– необхідність виконання складних математичних обчислень для персоналізації програм;

– потреба в автоматичній валідації біометричних даних;

– важливість автоматично генерованої документації для подальшого розвитку проєкту;

– високу продуктивність при обробці одночасних запитів.

2.2.2 Системи управління базами даних

Для зберігання даних розглянуто реляційні та нереляційні СУБД.

PostgreSQL є потужною реляційною базою даних з підтримкою ACID-транзакцій та складних запитів. Переваги: надійність, підтримка JSON-полів, потужні можливості індексації. Недоліки: менша гнучкість схеми, складність горизонтального масштабування.

MySQL забезпечує високу продуктивність для читання даних та має широку підтримку спільноти. Проте MySQL має обмеження в роботі з JSON-даними порівняно з PostgreSQL.

MongoDB [6] є документо-орієнтованою NoSQL базою даних. Переваги: гнучкість схеми даних, природна підтримка JSON, простота горизонтального масштабування, ефективна робота з великими обсягами неструктурованих даних.

Для проєкту обрано MongoDB з наступних причин:

– біометричні дані мають різноманітну структуру залежно від типу показників;

– необхідність зберігання JSON-об'єктів тренувальних програм зі складною ієрархією;

– планована інтеграція з різними пристроями моніторингу, що надають дані в різних форматах;

– потреба в горизонтальному масштабуванні при зростанні кількості користувачів.

2.2.3 Frontend технології

Для клієнтської частини проаналізовано сучасні JavaScript-фреймворки.

React.js забезпечує компонентний підхід до розробки інтерфейсів [7]. Переваги: великий екосистема, активна спільнота, гнучкість архітектури, підтримка TypeScript. Недоліки: необхідність додаткових бібліотек для повнофункціонального додатку, крива навчання для початківців.

Vue.js пропонує простіший синтаксис та швидше освоєння. Переваги: інтуїтивність, хороша документація, вбудована підтримка анімацій. Недоліки: менший екосистема порівняно з React, обмежені можливості для великих додатків.

Angular є повнофункціональним фреймворком з вбудованими інструментами. Переваги: структурованість, TypeScript за замовчуванням, потужні інструменти розробки. Недоліки: висока складність, повільніша розробка простих компонентів.

Обрано React.js з TypeScript через [8]:

– широкий вибір компонентних бібліотек для швидкої розробки UI;

– потужну підтримку візуалізації даних через бібліотеки типу Recharts;

– можливість використання Shadcn UI для створення сучасного доступного інтерфейсу;

– типобезпечність TypeScript для мінімізації помилок при роботі з біометричними даними.

2.3 Модель даних системи

2.3.1 Сутності системи

Проєктування моделі даних базується на аналізі предметної області та функціональних вимог системи.

Основними сутностями системи є:

Користувач (User) містить базову інформацію про особу: унікальний ідентифікатор, email, хешовані паролі, роль у системі, дату реєстрації.

Профіль користувача (Profile) включає персональні характеристики: ім'я, вік, антропометричні дані (вага, зріст), рівень фізичної активності, фітнес-цілі, медичні обмеження.

Біометричні дані (BiometricData) зберігають показники здоров'я з прив'язкою до дати та часу: вага, частота серцевих скорочень, кількість кроків, витрачені калорії, тривалість сну, додаткові примітки.

Вправа (Exercise) описує окремі тренувальні елементи: назва, категорія (кардіо, силові, гнучкість), рівень складності, тривалість, витрати калорій, детальний опис техніки виконання.

Тренувальна програма (WorkoutProgram) об'єднує вправи у структуровані плани: назва програми, рівень складності, тривалість курсу, цільова аудиторія, послідовність вправ з параметрами виконання.

2.3.2 Зв'язки між сутностями

Модель даних передбачає наступні зв'язки:

– один користувач має один профіль (один до одного);

– один користувач може мати багато біометричних записів (один до багатьох);

– одна тренувальна програма може включати багато вправ, одна вправа може входити до багатьох програм (багато до багатьох);

– один користувач може виконувати багато тренувальних програм у різний час.

Така структура забезпечує гнучкість системи та можливість ефективного аналізу даних для персоналізації рекомендацій.

2.4 Алгоритми персоналізації тренувальних програм

2.4.1 Математичні моделі для розрахунку навантажень

Ефективність фітнес-платформи значною мірою залежить від якості алгоритмів персоналізації, що автоматично адаптують тренувальні програми до індивідуальних потреб користувача.

Базовим елементом персоналізації є розрахунок оптимальної інтенсивності тренувань. Для визначення тренувальних зон частоти серцевих скорочень використовується удосконалена формула Карвонена:

$$ЧСС\_тренувальна = $$
$$=(ЧСС\_макс - ЧСС\_спокою) \times К\_інтенсивності + ЧСС\_спокою, \quad (1.3)$$

де К_інтенсивності варіюється залежно від цілей тренування:

жиросжигання: 0,6-0,7;

розвиток аеробної витривалості: 0,7-0,8;

анаеробна зона: 0,8-0,9.

Розрахунок енергетичних витрат здійснюється з урахуванням індивідуальних параметрів користувача за формулою:

$$Калорії = МЕТ \times маса\_тіла \times час\_тренування \times К\_корекції, \quad (1.4)$$

де МЕТ (метаболічний еквівалент) визначається для кожного типу вправи;

К_корекції враховує вік, стать та рівень підготовки користувача.

2.4.2 Алгоритм адаптації програм

Система персоналізації працює за принципом безперервної адаптації. Початкова програма формується на основі профілю користувача та його заявлених цілей. Надалі алгоритм аналізує результати виконання тренувань та біометричні показники для коригування навантажень.

Критеріями для адаптації програми є:

- динаміка показників серцево-судинної системи;
- зміна антропометричних параметрів;
- суб'єктивна оцінка складності тренувань користувачем;
- регулярність виконання запланованих занять.

На основі цих даних система автоматично коригує інтенсивність, тривалість та частоту тренувань, забезпечуючи оптимальний прогрес користувача при мінімізації ризику перетренованості або травмування.

2.5 Висновки до розділу 2

У другому розділі виконано проєктування архітектури інформаційної системи персоналізованого фітнес-коучингу з обґрунтуванням вибору технологічних рішень та розробкою алгоритмів персоналізації.

Проведено порівняльний аналіз архітектурних підходів для веб-платформ, що виявив переваги та недоліки монолітної та мікросервісної архітектур. Монолітна архітектура характеризується простотою розробки для невеликих проектів, проте має суттєві обмеження щодо масштабування окремих компонентів та залежність від єдиної технології. Мікросервісна архітектура забезпечує гнучкість у виборі технологій, незалежне масштабування сервісів та підвищену стійкість до відмов, хоча вимагає більш складного управління. На основі аналізу обрано мікросервісну архітектуру через необхідність обробки різнотипних даних та потребу в гнучкому масштабуванні. Визначено основні компоненти системи: сервіс авторизації та управління користувачами, сервіс обробки біометричних даних, сервіс генерації тренувальних програм, сервіс аналітики, API-шлюз та веб-інтерфейс користувача.

Здійснено обґрунтований вибір технологічного стеку для реалізації системи. Для серверної частини обрано Python з FastAPI фреймворком завдяки необхідності виконання складних математичних обчислень, автоматичній валідації біометричних даних, важливості автоматично генерованої документації та високій продуктивності при обробці одночасних запитів. MongoDB обрано як систему управління базами даних через гнучкість схеми для зберігання різнотипних біометричних показників, природну підтримку JSON формату, ефективну роботу з великими обсягами неструктурованих даних та простоту горизонтального масштабування. Для клієнтської частини обрано React.js з TypeScript через широкий вибір компонентних бібліотек, потужну підтримку візуалізації даних, можливість використання Shadcn UI для створення сучасного інтерфейсу та типобезпечність TypeScript для мінімізації помилок.

Розроблено модель даних системи з визначенням основних сутностей та зв'язків між ними. Сутність User містить базову інформацію про особу включаючи унікальний ідентифікатор, email, хешовані паролі та роль у системі. Profile включає персональні характеристики користувача такі як вік, антропометричні дані, рівень активності, фітнес-цілі та медичні обмеження. BiometricData зберігає показники здоров'я з прив'язкою до дати включаючи вагу, частоту серцевих скорочень, кількість кроків, витрачені калорії та тривалість сну. Exercise описує окремі тренувальні елементи з характеристиками складності, тривалості та витрат калорій. WorkoutProgram об'єднує вправи у структуровані плани з послідовністю виконання та параметрами навантаження. Визначено зв'язки один-до-одного між користувачем та профілем, один-до-багатьох між користувачем та біометричними записами, багато-до-багатьох між програмами та вправами.

Розроблено алгоритми персоналізації тренувальних програм на основі математичних моделей. Реалізовано удосконалену формулу Карвонена для визначення тренувальних зон частоти серцевих скорочень з коефіцієнтами інтенсивності залежно від цілей тренування. Створено модель розрахунку енергетичних витрат з урахуванням метаболічного еквівалента вправи, маси тіла, часу тренування та коригуючих коефіцієнтів залежно від віку, статі та рівня підготовки. Розроблено алгоритм генерації персоналізованих програм що визначає оптимальне співвідношення типів навантажень залежно від цілей користувача: 60% кардіо для зниження ваги, 70% силових для набору маси, 50% кардіо для розвитку витривалості. Створено систему адаптації програм на основі аналізу прогресу що автоматично коригує інтенсивність, тривалість та частоту тренувань для забезпечення оптимального навантаження при мінімізації ризику перетренованості.

Результати проєктування забезпечують технічну основу для реалізації масштабованої, надійної та продуктивної інформаційної платформи з науково обґрунтованими алгоритмами персоналізації тренувальних програм на основі індивідуальних біометричних показників користувачів.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ПЛАТФОРМИ

3.1 Організація процесу розробки

3.1.1 Налаштування середовища розробки

Розробка інформаційної платформи FitnessPal здійснювалася за ітераційним підходом з поетапним нарощуванням функціональності. Процес було розпочато з налаштування середовища розробки та створення базової архітектури, після чого послідовно реалізовувалися окремі модулі системи.

Спочатку було створено структуру проєкту з розділенням на серверну та клієнтську частини. Для серверної частини обрано Python 3.8+ з FastAPI фреймворком, що потребувало налаштування віртуального середовища для ізоляції залежностей. Створення віртуального середовища здійснювалося наступним чином:

```
bash
python -m venv venv
source venv/bin/activate  # для Linux/Mac систем
```

Після активації віртуального середовища встановлювалися необхідні залежності, включаючи FastAPI для створення API, pymongo для роботи з MongoDB, pydantic для валідації даних, passlib для хешування паролів та python-jose для роботи з JWT токенами. Повний перелік залежностей було збережено у файлі requirements.txt для забезпечення відтворюваності середовища розробки.

Для клієнтської частини налаштовувалося середовище Node.js версії 18+ з менеджером пакетів npm. Створювався новий React проєкт з використанням Vite як збирача для забезпечення швидкої розробки та hot reload функціональності. TypeScript було додано для забезпечення типобезпечності коду та покращення досвіду розробки.

3.1.2 Налаштування бази даних

MongoDB обрано як основну систему управління базами даних через гнучкість роботи з документами різної структури, що особливо важливо для

зберігання біометричних даних різних типів. Підключення до MongoDB здійснювалося через pymongo драйвер з налаштуванням connection string для локальної розробки.

Було створено базу даних "fitnesspal" з наступними колекціями: users для зберігання облікових записів користувачів, profiles для персональної інформації, biometric_data для біометричних показників, exercises для бази вправ та workout_programs для тренувальних програм.

Для забезпечення продуктивності створено індекси на часто використовуваних полях: унікальний індекс на email в колекції users, складений індекс на user_id та date в biometric_data, текстовий індекс на назві та описі в exercises.

3.2 Розробка серверної частини системи

3.2.1 Структурування проєкту

Створення серверних компонентів здійснювалося поступово, починаючи з базової структури проєкту та поетапно додаючи функціональність.

Було створено чітку структуру серверної частини проєкту для забезпечення читабельності коду та спрощення подальшого супроводу:

```
backend/
├── app/
│   ├── models/        # Pydantic моделі даних
│   ├── routes/        # API endpoints
│   ├── utils/         # Утиліти (auth, database)
│   └── main.py        # Основний файл додатку
├── requirements.txt   # Python залежності
└── README.md          # Документація
```

Рисунок 3.1 – Структура серверної частини проєкту

Головний файл main.py було організовано як точку входу додатку з налаштуванням FastAPI інстансу, підключенням CORS middleware та реєстрацією маршрутів.

Створення FastAPI додатку здійснювалося через ініціалізацію об'єкта FastAPI з базовими параметрами: назвою додатку "FitnessPal API", версією "1.0.0" та описом для автоматичної генерації документації. Додатково налаштовувалися шляхи до документації Swagger UI (/docs) та ReDoc (/redoc) для зручності тестування API під час розробки.

CORS (Cross-Origin Resource Sharing) middleware було підключено для вирішення проблеми політики одного походження, яка за замовчуванням блокує запити між різними портами. Оскільки серверна частина працює на порту 8000, а клієнтська на порту 3000, браузер блокував би такі запити без відповідної конфігурації. Налаштування CORS включало наступні параметри:

– allow_origins=["http://localhost:3000"] для дозволу запитів саме з клієнтської частини;

– allow_credentials=True для підтримки передачі cookies та авторизаційних заголовків;

– allow_methods=["GET", "POST", "PUT", "DELETE"] для дозволу всіх необхідних HTTP методів;

– allow_headers=["*"] для дозволу всіх типів заголовків, включаючи Authorization з JWT токенами.

Реєстрація маршрутів здійснювалася через include_router методи з префіксами для групування endpoints: /api/auth для аутентифікації, /api/users для управління профілем, /api/biometric для біометричних даних, /api/workouts для тренувальних програм. Такий підхід забезпечує чітку структуру API та спрощує його використання клієнтськими додатками.

3.2.2 Розробка моделей даних

Моделі даних створювалися з використанням Pydantic для автоматичної валідації та серіалізації. Розробка розпочалася з базової моделі користувача User, що включає наступні поля:

– id як ObjectId для унікальної ідентифікації;

– email з валідацією формату електронної пошти;

– hashed_password для безпечного зберігання паролів;

– role для розмежування прав доступу;

– created_at та updated_at для аудиту змін.

Модель Profile було розроблено для зберігання персональної інформації користувача:

– user_id як зовнішній ключ до користувача;

– name, age, gender для базової персональної інформації;

– height, activity_level для розрахунку персоналізованих рекомендацій;

– goals як список цілей користувача;

– target_weight для відстеження прогресу.

Модель BiometricData спроєктовано для гнучкого зберігання різнотипних біометричних показників:

– user_id та date для ідентифікації та хронології;

– weight, heart_rate, steps, calories_burned, sleep_hours як основні показники;

– notes для додаткових коментарів користувача;

– created_at для точного часу запису.

Модель Exercise містить інформацію про окремі вправи:

– name та description для ідентифікації та пояснення;

– category для класифікації типу навантаження;

– difficulty для визначення рівня складності;

– duration_minutes та calories_per_minute для розрахунку навантаження.

Модель WorkoutProgram об'єднує вправи у структуровані програми:

– name та description для опису програми;

– difficulty та duration_weeks для характеристик програми;

– exercises як масив посилань на вправи з параметрами виконання;

– target_goals для відповідності цілям користувача.

3.2.3 Реалізація API endpoints

Розробка API здійснювалася за групами функціональності з використанням FastAPI роутерів для структурування коду.

Група аутентифікації (/api/auth) реалізована першою як основа для захисту решти endpoints:

POST /register endpoint було створено для реєстрації нових користувачів. Процес включає валідацію унікальності email, хешування пароля за допомогою bcrypt та створення запису в базі даних. Повертається підтвердження успішної реєстрації без чутливих даних.

POST /login endpoint забезпечує аутентифікацію користувачів. Перевіряється відповідність email та пароля, після чого генерується JWT токен з терміном дії 24 години. Токен містить user_id та role для подальшої авторизації.

GET /me endpoint дозволяє отримати інформацію про поточного користувача на основі JWT токена. Реалізовано middleware для автоматичного декодування токена та визначення користувача.

Група управління профілем (/api/users) розроблена для роботи з персональними даними:

GET /profile endpoint повертає профіль користувача з усією персональною інформацією. Якщо профіль відсутній, повертається порожній об'єкт з базовими значеннями.

PUT /profile endpoint дозволяє створювати або оновлювати профіль користувача. Реалізована валідація вхідних даних та автоматичне обчислення BMI на основі ваги та зросту.

Група біометричних даних (/api/biometric) реалізована для роботи з показниками здоров'я:

POST /data endpoint створено для додавання нових біометричних записів. Валідується коректність числових значень та автоматично встановлюється поточна дата, якщо не вказана інша.

GET /data endpoint забезпечує отримання історії біометричних даних з можливістю фільтрації за датами. Реалізована пагінація для ефективної роботи з великими обсягами даних.

GET /stats endpoint розраховує статистичні показники для візуалізації: середні значення, тенденції змін, мінімальні та максимальні значення за вказаний період.

Група тренувальних програм (/api/workouts) забезпечує роботу з вправами та програмами:

GET /exercises endpoint повертає список доступних вправ з можливістю фільтрації за категорією та рівнем складності.

GET /programs endpoint надає програми тренувань, відфільтровані відповідно до профілю та цілей користувача.

POST /programs endpoint дозволяє користувачеві розпочати виконання обраної програми з автоматичним розрахунком персоналізованих параметрів.

3.3 Створення клієнтської частини

3.3.1 Структурування React додатку

Розробка frontend додатку здійснювалася паралельно з серверною частиною з використанням React та TypeScript.

Було створено структуру проєкту, що забезпечує масштабованість та підтримуваність коду:

Головний компонент App.tsx організовано як точку входу з налаштуванням маршрутизації через React Router та підключенням глобальних провайдерів контексту.

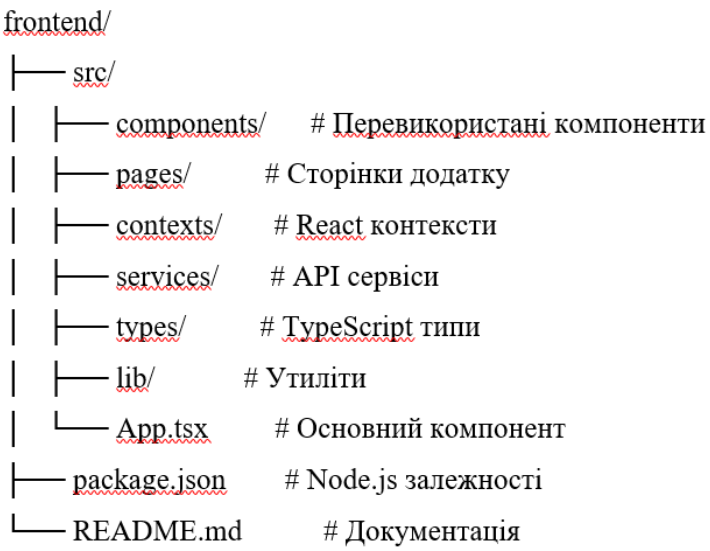

Рисунок 3.2 – Структура проєкту

3.3.2 Система маршрутизації та захисту

React Router використовувався для організації навігації між сторінками додатку. Створено ProtectedRoute компонент для захисту приватних сторінок від неавторизованих користувачів.

Реалізовано наступні маршрути:

– "/" – перенаправлення на дашборд для авторизованих або на сторінку входу;

– "/login" – сторінка аутентифікації;

– "/dashboard" – головна панель з статистикою;

– "/profile" – налаштування профілю користувача;

– "/biometric" – управління біометричними даними;

– "/workouts" – тренувальні програми та вправи.

3.3.3 Управління станом додатку

AuthContext створено для централізованого управління станом аутентифікації. Контекст зберігає інформацію про поточного користувача, JWT токен та статус завантаження. Реалізовано автоматичне оновлення токена при запуску додатку шляхом перевірки збереженого токена в localStorage та валідації його на сервері. У випадку недійсного токена користувач перенаправляється на сторінку входу. Локальний стан компонентів управляється через useState та

useEffect хуки. Для складних форм створено custom hooks, що інкапсулюють логіку валідації та обробки помилок.

3.3.4 Розробка компонентів інтерфейсу

Система компонентів побудована на основі Shadcn UI бібліотеки з кастомізацією під потреби фітнес-додатку.

LoginPage компонент розроблено з підтримкою табів для перемикання між входом та реєстрацією. Форми містять валідацію полів, обробку помилок сервера та відображення статусу завантаження. Додано список демо-акаунтів для зручності тестування. DashboardPage компонент створено як головну сторінку з відображенням ключової статистики. Реалізовано чотири статистичні картки:

– поточний BMI з автоматичним розрахунком та статусом;

– завершених тренувань з лічильником;

– досягнення з кількістю розблокованих бейджів;

– поточна вага з відображенням цільового значення.

Додано два графіки для візуалізації динаміки: прогрес ваги та динаміка BMI. Використано Recharts бібліотеку для створення інтерактивних графіків з українськими підписами та форматуванням дат.

ProfilePage компонент реалізовано для управління персональними налаштуваннями. Форма включає валідацію всіх полів, відображення поточних значень та збереження змін на сервер. Цілі тренувань реалізовано як інтерактивні картки з можливістю множинного вибору. BiometricPage компонент розроблено для роботи з біометричними даними. Форма додавання містить поля для різних типів показників з валідацією числових значень. Історія записів відображається у вигляді карток з можливістю сортування за датою. WorkoutPage компонент створено для відображення тренувальних програм та вправ. Реалізовано систему табів для перемикання між програмами, вправами та історією тренувань. Кожна програма відображається у вигляді картки з інформацією про складність, тривалість та цілі.

3.3.5 Інтеграція з API та обробка даних

Створено централізований API сервіс з використанням Axios для взаємодії з серверною частиною. Сервіс автоматично додає авторизаційні заголовки до запитів та обробляє оновлення токенів.

Реалізовано окремі модулі для кожної групи API endpoints:

– authAPI для аутентифікації користувачів;

– usersAPI для управління профілем;

– biometricAPI для біометричних даних;

– workoutsAPI для тренувальних програм.

Обробка помилок здійснюється на кількох рівнях: валідація на клієнті, обробка помилок сервера та відображення користувацьких повідомлень українською мовою. Loading стани реалізовано для всіх асинхронних операцій з відповідними індикаторами завантаження.

3.4 Реалізація алгоритмів персоналізації

3.4.1 Розробка алгоритмів розрахунку базових параметрів

Центральним компонентом системи є модуль персоналізації тренувальних програм, що було розроблено для автоматичної адаптації рекомендацій до індивідуальних потреб користувача.

Функція calculate_bmi реалізована для обчислення індексу маси тіла за стандартною формулою BMI = weight / (height/100)². Її реалізація на python подана ліст. 3.1.

Лістинг 3.1 - Функція для обчислення індексу маси тіла за стандартною формулою

```
def calculate_bmi(weight: float, height: float) -> dict:
    bmi = weight / ((height / 100) ** 2)
```

```
        if bmi < 18.5:
            status = "Недостатня вага"
        elif bmi < 25.0:
            status = "Нормальна вага"
        elif bmi < 30.0:
            status = "Надлишкова вага"
        else:
            status = "Ожиріння"

        return {
            "bmi": round(bmi, 1),
            "status": status
        }
```

Функція calculate_bmr розроблена для розрахунку базового метаболізму за рівнянням Міффліна-Сан Жеора з урахуванням статі, віку, ваги та зросту користувача (ліст. 3.2).

Лістинг 3.2 - Розрахунок базового метаболізму за рівнянням Міффліна-Сан Жеора

```
    def calculate_bmr(weight: float, height: float, age: int, gender:
str) -> float:
        if gender.lower() == 'male':
            bmr = 10 * weight + 6.25 * height - 5 * age + 5
        else:  # female
            bmr = 10 * weight + 6.25 * height - 5 * age - 161
            return round(bmr, 0)
```

Реалізовано функцію calculate_target_heart_rate для визначення тренувальних зон частоти серцевих скорочень за формулою Карвонена (ліст. 3.3).

Лістинг 3.3 - Визначення тренувальних зон частоти серцевих скорочень за формулою Карвонена

```
    def  calculate_target_heart_rate(age:  int,  resting_hr:  int,
intensity: float) -> int:
        max_hr = 220 - age
        target_hr = (max_hr - resting_hr) * intensity + resting_hr
        return round(target_hr, 0)

    def get_training_zones(age: int, resting_hr: int = 70) -> dict:
        return {
```

```
            "fat_burn": calculate_target_heart_rate(age, resting_hr,
0.6),
            "cardio":  calculate_target_heart_rate(age,  resting_hr,
0.7),
            "peak":   calculate_target_heart_rate(age,   resting_hr,
0.85)
        }
```

Цей показник використовується для подальшого розрахунку енергетичних потреб та рекомендацій по калоражу з урахуванням рівня активності користувача.

3.4.2 Алгоритм генерації персоналізованих програм

Створено модуль, що реалізує логіку автоматичного підбору тренувальних програм. Алгоритм аналізує профіль користувача, включаючи цілі, рівень підготовки, доступний час та преференції.

Для кожної цілі користувача визначається оптимальне співвідношення типів навантажень:

– для цілі "weight_loss" пріоритет надається кардіо-вправам (60%) з додаванням силових елементів (30%) та розтяжки (10%);

– для цілі "muscle_gain" акцент робиться на силових тренуваннях (70%) з мінімальним кардіо (20%) та обов'язковою розтяжкою (10%);

– для цілі "endurance" збалансоване поєднання кардіо (50%) та силових вправ (40%) з розтяжкою (10%).

Система враховує рівень підготовки користувача при виборі складності вправ та розрахунку навантажень. Для початківців обираються базові вправи з меншою інтенсивністю, для досвідчених користувачів – більш складні варіанти з вищими навантаженнями.

3.4.3 Система адаптації програм

Розроблено модуль для динамічного коригування програм на основі зворотного зв'язку. Алгоритм аналізує наступні показники:

– регулярність виконання запланованих тренувань;

– зміну біометричних параметрів у часі;

– суб'єктивну оцінку складності від користувача.

Функція analyze_progress обчислює коефіцієнт прогресу на основі динаміки ключових показників (ліст. 3.4).

Лістинг 3.4 - Обчислення коефіцієнту прогресу на основі динаміки ключових показників

```
    def analyze_progress(user_id: str, days: int = 30) -> dict:
        biometric_data = get_user_biometric_data(user_id, days)

        if len(biometric_data) < 2:
            return {"progress_score": 0, "recommendation":
"insufficient_data"}

        # Аналіз динаміки ваги
        weights = [d.weight for d in biometric_data if d.weight]
        weight_trend = (weights[-1] - weights[0]) / len(weights) if
len(weights) > 1 else 0

        # Аналіз регулярності тренувань
        workout_frequency = len([d for d in biometric_data if
d.workout_completed]) / days

        # Розрахунок загального скору прогресу
        progress_score = (workout_frequency * 0.6) +
(abs(weight_trend) * 0.4)

        return {
            "progress_score": round(progress_score, 2),
            "weight_trend": weight_trend,
            "workout_frequency": workout_frequency,
            "recommendation":
get_adaptation_recommendation(progress_score)
        }

    def get_adaptation_recommendation(score: float) -> str:
        if score < 0.3:
            return "increase_intensity"
        elif score > 0.8:
            return "decrease_intensity"
        else:
            return "maintain_current"
```

Якщо прогрес недостатній, система може рекомендувати збільшення інтенсивності або частоти тренувань. При надмірно швидкому прогресі або ознаках перетренованості рекомендується зниження навантажень.

Реалізовано функцію adapt_program, що автоматично коригує параметри існуючої програми (ліст. 3.5)

Лістинг 3.5 – Автоматичне коригування параметрів наявної програми

```
    def adapt_program(program_id: str, user_id: str, progress_data:
dict) -> dict:
        """Адаптація тренувальної програми на основі прогресу"""
        current_program = get_workout_program(program_id)
        recommendation = progress_data["recommendation"]

        adapted_program = current_program.copy()

        if recommendation == "increase_intensity":
            # Збільшуємо навантаження на 10-15%
            for exercise in adapted_program["exercises"]:
                if exercise.get("duration"):
                    exercise["duration"] = min(exercise["duration"]
* 1.1, 60)  # макс 60 хв
                if exercise.get("sets"):
                    exercise["sets"] = min(exercise["sets"] + 1, 5)
# макс 5 підходів

        elif recommendation == "decrease_intensity":
            # Зменшуємо навантаження на 15-20%
            for exercise in adapted_program["exercises"]:
                if exercise.get("duration"):
                    exercise["duration"] = max(exercise["duration"]
* 0.8, 10)  # мін 10 хв
                if exercise.get("sets"):
                    exercise["sets"] = max(exercise["sets"] - 1, 1)
# мін 1 підхід

        # Зберігаємо адаптовану програму з міткою часу
        adapted_program["adapted_at"] = datetime.now()
        adapted_program["adaptation_reason"] = recommendation

        return save_adapted_program(user_id, adapted_program)
```

Зміни вносяться поступово для забезпечення плавної адаптації організму користувача.

3.5 Висновки до розділу 3

У третьому розділі виконано практичну реалізацію інформаційної платформи персоналізованого фітнес-коучингу з детальним описом технічних рішень та особливостей імплементації.

Організовано процес розробки з налаштуванням середовища включаючи створення структури проєкту з розділенням серверної та клієнтської частин, налаштування віртуального Python середовища для ізоляції залежностей серверної частини, конфігурацію Node.js середовища з Vite збирачем для клієнтської частини. Налаштовано базу даних MongoDB з створенням колекцій для користувачів, профілів, біометричних даних, вправ та тренувальних програм, з відповідними індексами для забезпечення продуктивності запитів.

Розроблено серверну частину системи з чіткою структурою проєкту що включає окремі директорії для моделей даних, API маршрутів та утиліт. Створено FastAPI додаток з налаштуванням CORS middleware для взаємодії з клієнтською частиною, автоматичною генерацією документації через Swagger UI та реєстрацією маршрутів з префіксами для групування endpoints. Розроблено Pydantic моделі даних для автоматичної валідації вхідних даних включаючи User для базової інформації користувача, Profile для персональних характеристик, BiometricData для показників здоров'я, Exercise для опису вправ та WorkoutProgram для тренувальних планів. Реалізовано API endpoints організовані за функціональними групами: аутентифікація з реєстрацією, входом та перевіркою токенів, управління профілем з отриманням та оновленням персональних даних, біометричні дані з додаванням записів, отриманням історії та розрахунком статистики, тренувальні програми з переглядом вправ та отриманням персоналізованих рекомендацій.

Створено клієнтську частину додатку з використанням React та TypeScript. Організовано структуру проєкту з компонентами, сторінками, контекстами, сервісами та типами. Реалізовано систему маршрутизації через React Router з захистом приватних сторінок від неавторизованих користувачів через компонент

ProtectedRoute. Створено AuthContext для централізованого управління станом аутентифікації з автоматичним оновленням токенів та перенаправленням при недійсних сесіях. Розроблено повний набір компонентів інтерфейсу включаючи LoginPage з табами для входу та реєстрації, DashboardPage з статистичними картками та графіками прогресу, ProfilePage з формами налаштування персональних даних та цілей, BiometricPage з формою додавання показників та історією записів, WorkoutPage з каталогом тренувальних програм та вправ. Інтегровано з API через централізований сервіс з використанням Axios для виконання запитів з автоматичним додаванням авторизаційних заголовків та обробкою помилок.

Реалізовано алгоритми персоналізації включаючи функції розрахунку базових параметрів: calculate_bmi для індексу маси тіла з автоматичною класифікацією за стандартами ВООЗ, calculate_bmr для базового метаболізму за рівнянням Міффліна-Сан Жеора з урахуванням статі, calculate_target_heart_rate для тренувальних зон за формулою Карвонена. Створено алгоритм генерації персоналізованих програм що аналізує профіль користувача та визначає оптимальне співвідношення типів навантажень залежно від цілей з урахуванням рівня підготовки. Розроблено систему адаптації програм з функцією analyze_progress що обчислює коефіцієнт прогресу на основі динаміки показників та регулярності тренувань, і функцією adapt_program що автоматично коригує параметри програми для забезпечення оптимального прогресу.

Результати реалізації підтверджують працездатність спроєктованої архітектури та обраного технологічного стеку, забезпечуючи повнофункціональну платформу з інтуїтивним інтерфейсом українською мовою, надійною системою аутентифікації, ефективною обробкою біометричних даних та науково обґрунтованими алгоритмами персоналізації тренувальних програм.

4 ОПИС КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

4.1 Загальні принципи дизайну інтерфейсу

Користувацький інтерфейс розробленої платформи FitnessPal було спроєктовано відповідно до сучасних принципів UX/UI дизайну з акцентом на зручність використання та доступність. Основою дизайн-системи є мінімалістичний підхід з використанням чистих форм, продуманої типографіки та зрозумілої ієрархії елементів.

Кольорова палітра платформи побудована навколо синього кольору (#3B82F6) як основного, що асоціюється з надійністю та технологічністю. Додатково використовуються акцентні кольори: зелений (#10B981) для позитивних дій та успішних станів, фіолетовий (#8B5CF6) для досягнень та мотиваційних елементів, помаранчевий (#F59E0B) для важливих показників. Типографіка базується на шрифті Inter, який забезпечує відмінну читабельність на всіх типах пристроїв. Використовуються різні варіанти насиченості шрифту для створення візуальної ієрархії: жирний для заголовків та важливих показників, середній для основного тексту, легкий для додаткової інформації.

Інтерфейс розроблено за принципом "mobile-first", що забезпечує оптимальне відображення на всіх типах пристроїв. На мобільних пристроях використовується одноколонковий макет з вертикальним розташуванням елементів. На планшетах та десктопах застосовується багатоколонкова сітка для ефективного використання простору екрану.

4.2 Сторінка авторизації

Початковим екраном користувача є сторінка входу до системи (рис. 4.1), що реалізована у вигляді центрованої картки з мінімалістичним дизайном. Сторінка

містить логотип платформи FitnessPal з характерною іконкою, що відразу ідентифікує бренд.

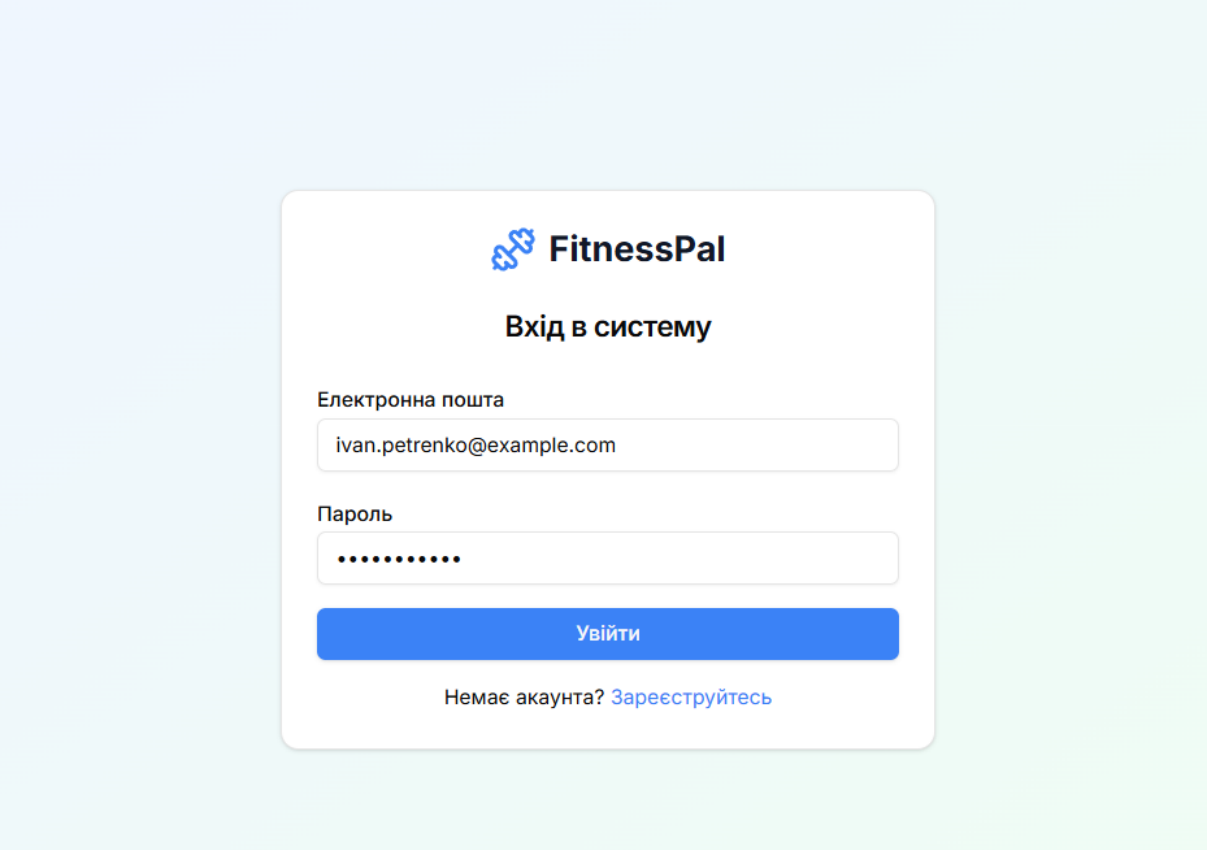

Рисунок 4.1 - Сторінка авторизації

Форма входу складається з двох основних полів: "Електронна пошта" та "Пароль". Поле пароля має функцію приховування/показу символів для зручності введення. Основна кнопка "Увійти" виконана в корпоративному синьому кольорі та займає всю ширину форми, що робить її помітною та легкодоступною. Під формою розміщено посилання "Немає акаунта? Зареєструйтесь" для переходу до режиму реєстрації. Таке рішення дозволяє уникнути перевантаження інтерфейсу додатковими елементами та зосередити увагу користувача на основній дії.

4.3 Головна панель (Dashboard)

Після успішної авторизації користувач потрапляє на головну панель системи (рис. 4.2), що надає комплексний огляд ключових показників та стану прогресу.

Ліворуч розташована навігаційна панель з вертикальним меню, що містить основні розділи додатку:

– дашборд (активний розділ, підсвічений синім кольором);

– біометричні дані;

– тренування;

– календар;

– досягнення;

– профіль.

Кожен пункт меню супроводжується відповідною іконкою, що полегшує навігацію та робить інтерфейс більш інтуїтивним. Активний розділ виділяється кольором фону та товщиною шрифту.

Основний простір головної панелі займають чотири статистичні картки, розташовані в один ряд:

Картка "Поточний BMI" (синя) відображає розрахований індекс маси тіла користувача (24.8) з підписом "Нормальна вага". Картка має градієнтний фон та іконку цілі в правому верхньому куті.

Картка "Завершених тренувань" (зелена) показує кількість завершених тренувань (0) з підписом "Всього". Використовується зелений колір, що традиційно асоціюється з виконанням та успіхом.

Картка "Досягнення" (фіолетова) демонструє кількість отриманих досягнень (2) з підписом "Розблоковано". Фіолетовий колір підкреслює престижність та мотиваційний аспект досягнень.

Картка "Поточна вага" (помаранчева) відображає актуальну вагу користувача (80.5 кг) з цільовим значенням "Ціль: 75 кг". Використання помаранчевого кольору привертає увагу до важливого показника.

Нижче статистичних карток розміщено два графіки, що візуалізують динаміку ключових показників:

Графік "Прогрес ваги" показує зміну ваги користувача за останній період. Лінійний графік з синіми точками та з'єднувальними лініями демонструє стабільні показники в районі 75 кг протягом вересня 2025 року.

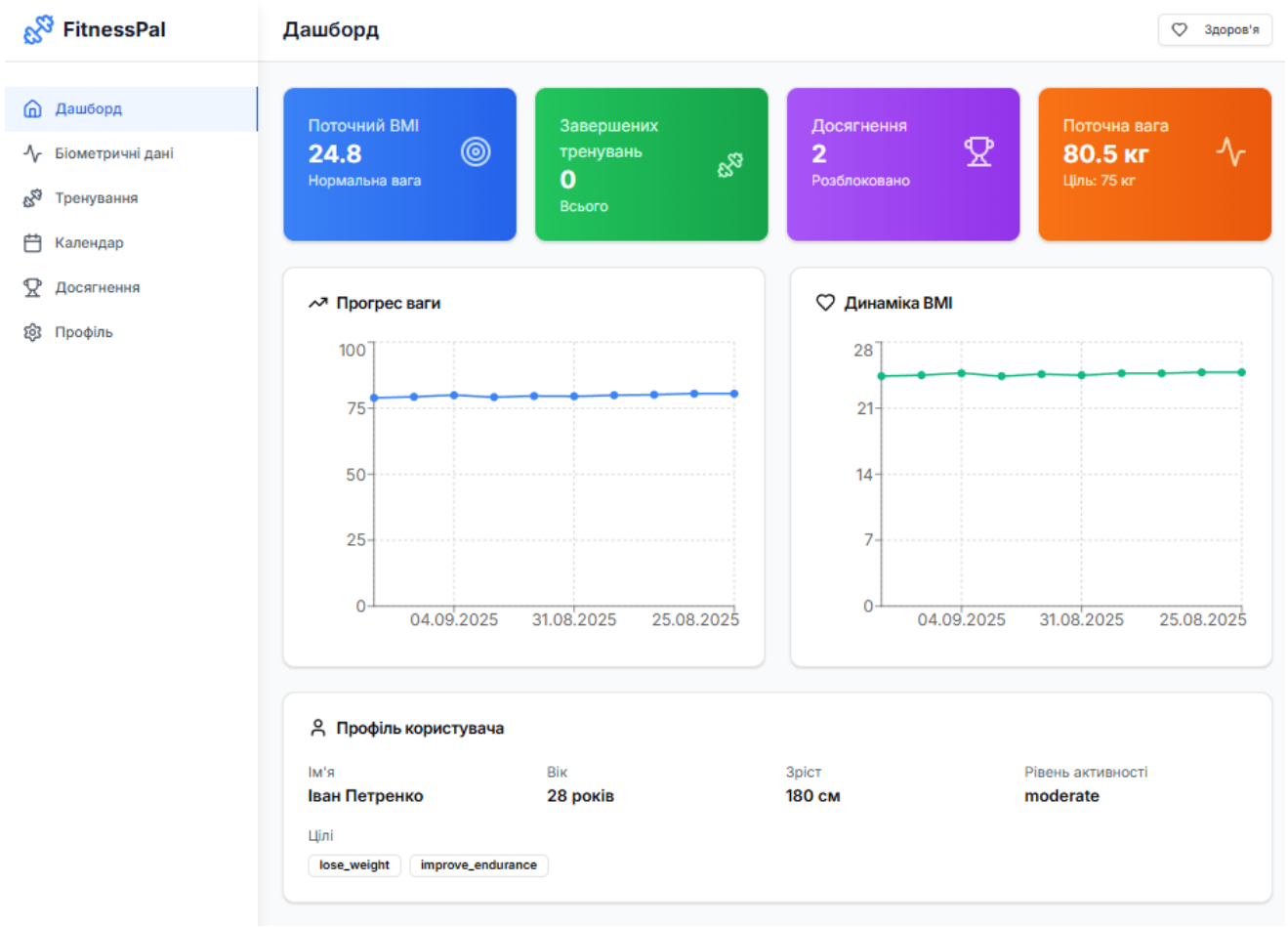

Рисунок 4.2 - Головна панель

Обидва графіки мають підписи осей українською мовою та використовують зрозумілі формати дат (04.09.2025, 31.08.2025, 25.08.2025).

Нижня частина дашборду містить картку з профілем користувача, де відображається основна інформація: ім'я (Іван Петренко), вік (28 років), зріст (180 см), рівень активності (moderate). Також показані активні цілі користувача у

вигляді тегів: "lose_weight" (зниження ваги) та "improve_endurance" (покращення витривалості).

Графік "Динаміка BMI" відображає зміни індексу маси тіла в часі. Використовується зелений колір для відображення позитивної динаміки. Графік показує стабільні значення BMI близько 25 протягом того ж періоду.

4.4 Сторінка біометричних даних

Сторінка біометричних даних (рис. 4.3) призначена для ведення щоденника показників здоров'я та відстеження їх динаміки.

Верхня частина сторінки містить розгорнуту форму для додавання нових біометричних записів. Форма організована в сітку з полями:

– "Вага (кг)" та "Пульс (уд/хв)" в першому ряду;

– "Кроки" та "Спалено калорій" в другому ряду;

– "Години сну" займає весь третій ряд;

– "Нотатки" - текстова область для додаткових коментарів з підказкою "Додаткові нотатки...".

Синя кнопка "Додати дані" розташована під формою та займає всю її ширину. Всі поля мають відповідні підписи та плейсхолдери, що полегшує заповнення.

Розділ "Останні записи" відображає хронологічний список введених біометричних даних. Кожен запис оформлений у вигляді картки з наступною структурою.

Заголовок картки містить дату запису (23.09.2025, 22.09.2025, тощо) та час (14:05, 14:03).

Основна частина показує конкретні показники: вагу (79.5 кг, 77.8 кг, 77.4 кг, 78 кг), пульс (72 уд/хв, 94 уд/хв, 99 уд/хв), кроки (8500, 7299, 8024, 8427), спалені калорії (350, 415).

Додаткові дані включають тривалість сну (8.7 год для запису від 21.09.2025) та нотатки користувача ("Хороше тренування сьогодні!" для першого запису).

Записи відсортовані в хронологічному порядку від найновіших до найстаріших, що дозволяє користувачеві швидко переглянути останні показники.

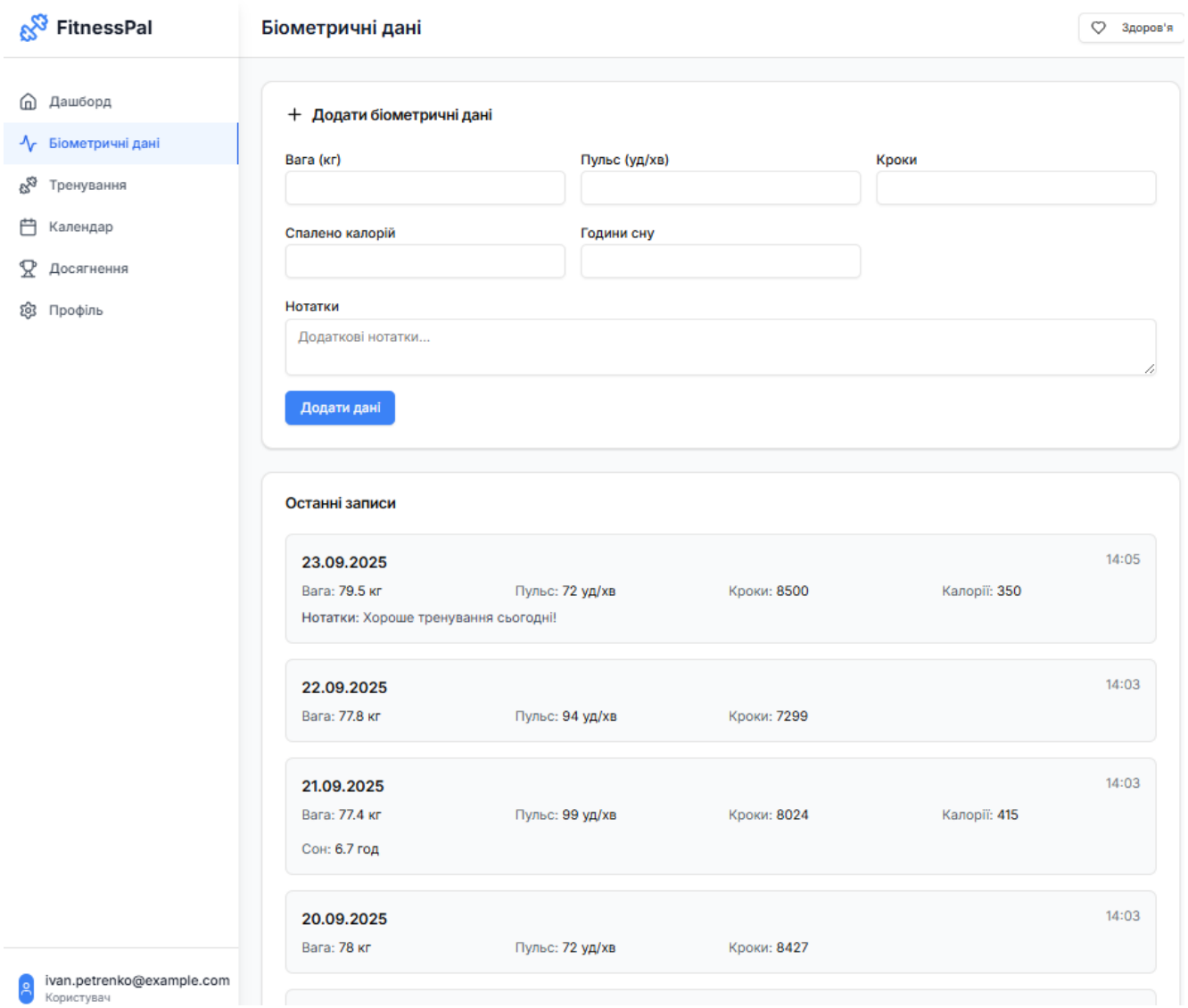

Рисунок 4.3 - Сторінка біометричних даних

4.5 Сторінка профілю користувача

Сторінка профілю (рис. 4.4 та 4.5) дозволяє користувачеві управляти персональними налаштуваннями та цілями тренувань.

Форма "Налаштування профілю" містить основні поля для персональних даних:

– "Ім'я" (Іван Петренко) та "Вік" (28 років) в першому ряду;

– "Вага (кг)" (80,5) та "Зріст (см)" (180) в другому ряду;

– "Рівень активності" та "Рівень підготовки" у вигляді випадаючих списків.

Випадаючий список рівня активності містить опції: "Малорухливий", "Помірна активність", "Активний", "Дуже активний", "Надзвичайно активний". Для рівня підготовки доступні варіанти: "Початківець", "Середній рівень", "Досвідчений".

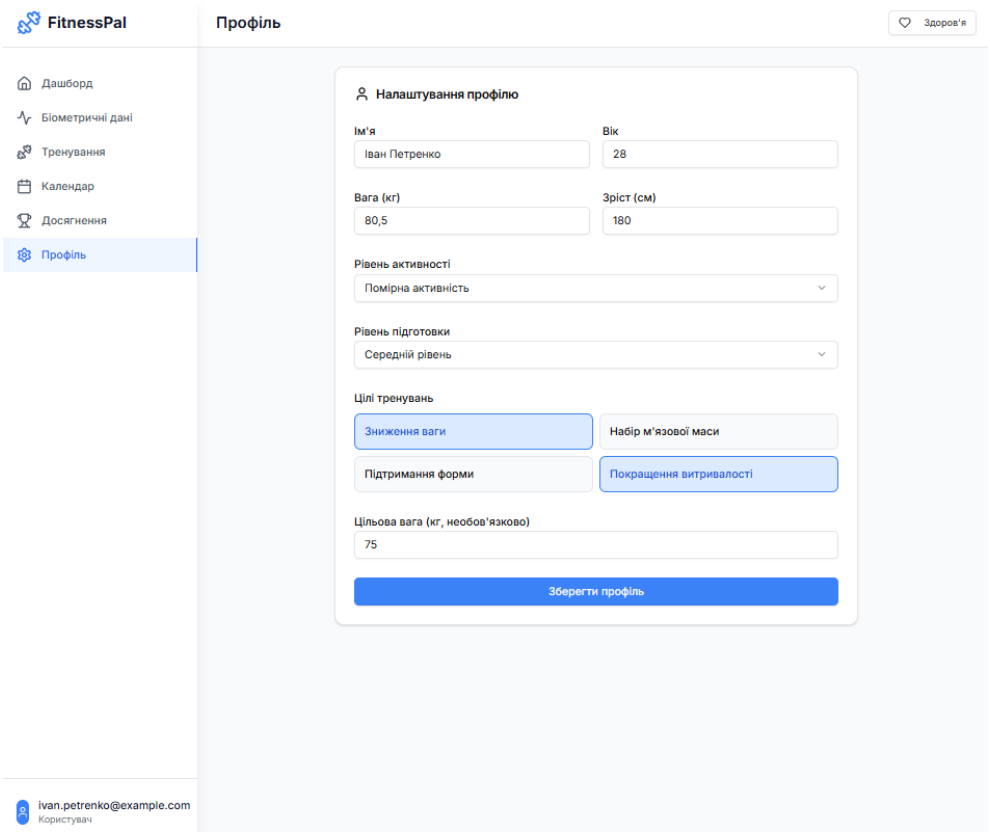

Рисунок 4.4 - Сторінка профілю користувача

Розділ "Цілі тренувань" реалізований у вигляді інтерактивних карток, що дозволяють множинний вибір:

– "Зниження ваги";

– "Набір м'язової маси";

– "Підтримання форми";

– "Покращення витривалості".

Активні цілі мають синій фон та білий текст, неактивні - сірий фон. Кожна картка містить круглий індикатор справа, що показує стан вибору.

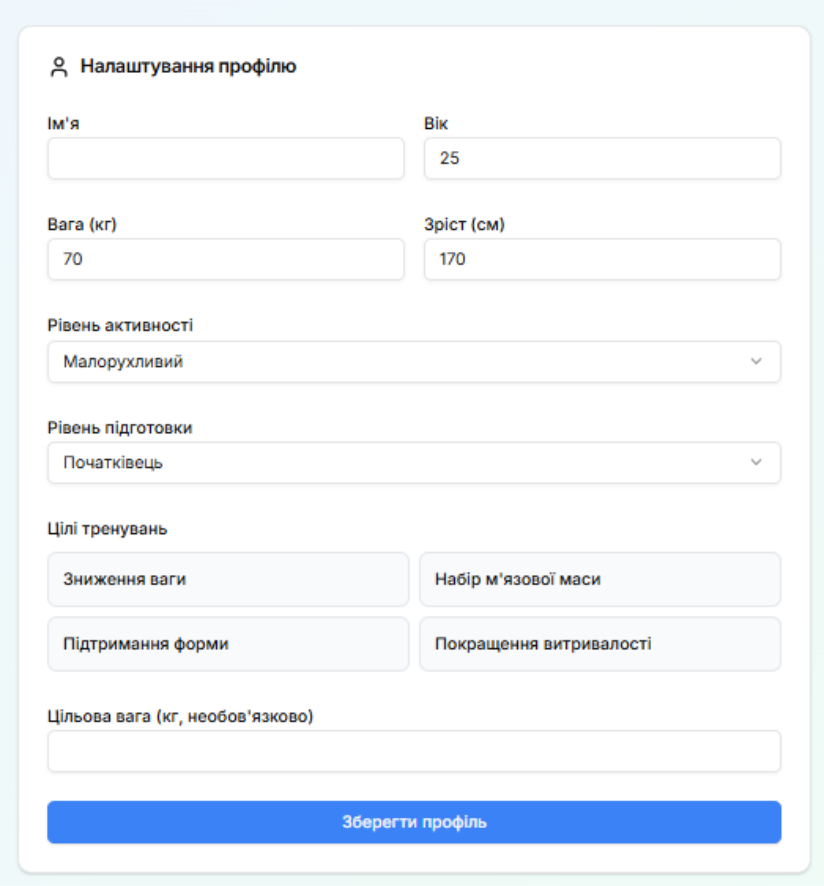

Рисунок 4.5 – Налаштування профілю користувача

Форма також містить поле "Цільова вага (кг, необов'язково)" зі значенням 75, що дозволяє системі розраховувати персоналізовані рекомендації для досягнення бажаного результату.

Синя кнопка "Зберегти профіль" розташована внизу форми та займає всю її ширину, забезпечуючи легкий доступ до збереження змін.

4.6 Сторінка тренувальних програм

Сторінка тренувань (рис. 4.6) є центральним розділом для роботи з фітнес-програмами та вправами.

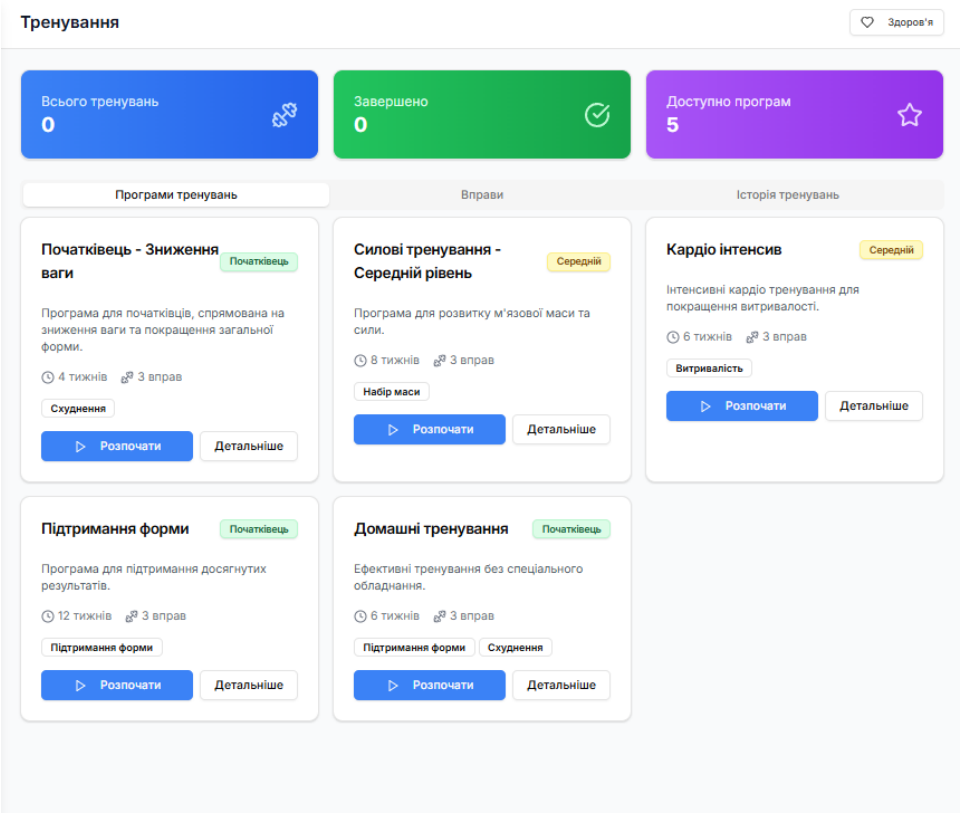

Рисунок 4.6 - Сторінка тренувань

Верхня частина сторінки містить три статистичні картки:

– "Всього тренувань" (0) з синім фоном та іконкою гантель;

– "Завершено" (0) із зеленим фоном та іконкою галочки;

– "Доступно програм" (5) з фіолетовим фоном та іконкою зірки.

Система табів дозволяє перемикатися між різними видами контенту:

– "Програми тренувань" (активний таб);

– "Вправи";

– "Історія тренувань".

Основний контент представлений у вигляді сітки карток з програмами тренувань:

"Початківець - Зниження ваги" має зелений бейдж "Початківець" та містить опис програми, тривалість (4 тижні), кількість вправ (3 вправи), категорію ("Схуднення") та синю кнопку "Розпочати".

"Силові тренування - Середній рівень" позначена жовтим бейджем "Середній рівень", розрахована на 8 тижнів, містить 3 вправи категорії "Набір маси".

"Кардіо інтенсив" також має жовтий бейдж середнього рівня, тривалість 6 тижнів, 3 вправи категорії "Витривалість".

"Підтримання форми" та "Домашні тренування" мають зелені бейджі початкового рівня з відповідними характеристиками.

Кожна картка містить іконку типу тренування, опис програми, технічні характеристики (тривалість, кількість вправ) та кнопку для початку тренувань. Бейджі складності мають різні кольори: зелений для початківців, жовтий для середнього рівня.

4.7 Висновки до розділу 4

У четвертому розділі представлено детальний опис користувацького інтерфейсу розробленої платформи з акцентом на принципах UX/UI дизайну та забезпеченні зручності використання для цільової аудиторії.

Визначено загальні принципи дизайну інтерфейсу що базуються на мінімалістичному підході з використанням чистих форм, продуманої типографіки та зрозумілої ієрархії елементів. Розроблено кольорову палітру з синім кольором як основним що асоціюється з надійністю та технологічністю, зеленим для

позитивних дій та успішних станів, фіолетовим для досягнень та мотиваційних елементів, помаранчевим для важливих показників. Обрано шрифт Inter для забезпечення відмінної читабельності на всіх типах пристроїв з використанням різних варіантів насиченості для створення візуальної ієрархії. Реалізовано адаптивний дизайн за принципом mobile-first що забезпечує оптимальне відображення на мобільних пристроях через одноколонковий макет та на десктопах через багатоколонкову сітку.

Розроблено сторінку авторизації у вигляді центрованої картки з мінімалістичним дизайном що містить логотип платформи, форму входу з полями електронної пошти та пароля з можливістю приховування символів, основну кнопку входу в корпоративному синьому кольорі, посилання для переходу до реєстрації. Форма включає валідацію полів з відображенням помилок безпосередньо під некоректно заповненими полями з повідомленнями українською мовою.

Створено головну панель Dashboard що надає комплексний огляд ключових показників через ліворуч розташовану навігаційну панель з вертикальним меню основних розділів, чотири статистичні картки різних кольорів для поточного BMI, завершених тренувань, досягнень та поточної ваги, два графіки для візуалізації динаміки прогресу ваги та BMI з використанням Recharts бібліотеки, картку профілю користувача з основною інформацією та активними цілями.

Розроблено сторінку біометричних даних з розгорнутою формою додавання нових записів організованою в сітку з полями для ваги, пульсу, кроків, спалених калорій, годин сну та нотаток, кнопкою додавання даних, розділом останніх записів що відображає хронологічний список введених даних у вигляді карток з датою, часом, показниками та додатковими даними.

Створено сторінку профілю користувача з формою налаштування що містить поля для персональних даних організованих у два ряди, випадаючі списки для рівня активності та підготовки, інтерактивні картки цілей тренувань що дозволяють множинний вибір з візуальним виділенням активних цілей, поле цільової ваги для розрахунку персоналізованих рекомендацій, кнопку збереження профілю.

Розроблено сторінку тренувальних програм зі статистичними картками відображення загальної кількості, завершених та доступних програм, системою табів для перемикання між програмами, вправами та історією тренувань, сіткою карток програм з інформацією про назву, рівень складності через кольорові бейджі, тривалість, кількість вправ, категорію та кнопку початку тренувань.

Всі елементи інтерфейсу реалізовані з повною українською локалізацією включаючи підписи полів, повідомлення про помилки, назви розділів та описи функціональності. Забезпечено консистентність дизайну через використання єдиної компонентної бібліотеки Shadcn UI з кастомізацією під потреби фітнес-додатку. Реалізовано інтерактивність з миттєвим відгуком на дії користувача, плавними переходами між станами та зрозумілими індикаторами завантаження.

Результати розробки користувацького інтерфейсу підтверджують досягнення високого рівня зручності використання через інтуїтивно зрозумілу навігацію, чітку візуальну ієрархію елементів, наочне відображення прогресу через графіки та статистику, адаптивність до різних типів пристроїв та повну українську локалізацію всіх елементів інтерфейсу.

5 АПАРАТНІ ВИМОГИ ТА НАЛАШТУВАННЯ СИСТЕМИ

5.1 Апаратні вимоги до системи

5.1.1 Вимоги до розробки системи

Розробка інформаційної платформи персоналізованого фітнес-коучингу потребує дотримання певних апаратних вимог для забезпечення ефективного функціонування середовища розробки. Правильний вибір апаратної конфігурації робочої станції розробника безпосередньо впливає на продуктивність роботи, швидкість компіляції коду та можливість одночасного тестування різних компонентів системи.

Мінімальні апаратні характеристики робочої станції розробника включають процесор Intel Core i5 восьмого покоління або еквівалентний AMD Ryzen 5 серії 3000 з тактовою частотою не менше 2.5 ГГц та можливістю багатопотокової обробки. Такий процесор здатен ефективно обробляти паралельні завдання компіляції, транспіляції TypeScript коду та запуску локальних серверів розробки. Оперативна пам'ять обсягом не менше 8 ГБ DDR4 з частотою 2400 МГц забезпечує достатній простір для одночасної роботи інтегрованого середовища розробки, веб-браузера з відкритими інструментами розробника, локальних серверів backend та frontend частин, а також інших допоміжних інструментів. Накопичувач типу SSD обсягом від 256 ГБ критично важливий для забезпечення швидкої роботи середовища розробки, оскільки операції читання та запису під час компіляції, встановлення залежностей через npm та pip менеджери пакетів створюють значне навантаження на підсистему введення-виведення.

Рекомендовані характеристики для оптимальної продуктивності значно підвищують комфорт роботи розробника та скорочують час виконання рутинних операцій. Процесор Intel Core i7 або AMD Ryzen 7 восьмого покоління або новіший з шістьма або вісьма фізичними ядрами дозволяє одночасно виконувати збірку проекту, запускати автоматичні тести, працювати з системою контролю версій та переглядати результати в браузері без відчутних затримок. Оперативна пам'ять обсягом 16 ГБ або більше особливо важлива при роботі з Docker контейнерами, віртуальними машинами для тестування на різних операційних системах, а також при необхідності тримати відкритими кілька проектів одночасно. Накопичувач NVMe SSD обсягом від 512 ГБ забезпечує максимальну швидкість доступу до даних, що критично важливо для операцій з великою кількістю дрібних файлів, характерних для сучасних веб-проектів з тисячами модулів у директорії node_modules. Додаткова дискретна відеокарта, хоча і не є обов'язковою для backend розробки, може прискорити роботу браузерів з багатьма відкритими вкладками та складними інтерфейсами завдяки апаратному прискоренню рендерингу.

Додаткові рекомендації стосуються периферійних пристроїв та організації робочого місця. Монітор з діагоналлю не менше 24 дюймів та роздільною здатністю Full HD дозволяє ефективно розташувати вікна редактора коду, термінала, браузера та документації без постійного перемикання між ними. Для максимальної продуктивності рекомендується конфігурація з двома моніторами, що дозволяє розмістити код на одному екрані, а результати його виконання та інструменти розробника на іншому. Якісна клавіатура механічного типу та ергономічна миш сприяють комфортній багатогодинній роботі з кодом. Безперебійне живлення важливе для захисту від втрати несзбереженої роботи та пошкодження файлових систем під час раптових відключень електроенергії.

5.1.2 Вимоги до експлуатації системи

Апаратні вимоги до кінцевих користувачів розробленої платформи визначаються веб-орієнтованою архітектурою системи та оптимізацією клієнтської частини для роботи на широкому спектрі пристроїв. Основна перевага веб-додатків полягає в мінімальних вимогах до клієнтського обладнання, оскільки основні обчислення виконуються на сервері, а клієнтська частина відповідає лише за відображення інтерфейсу та взаємодію з користувачем.

Для веб-доступу через браузер на персональному комп'ютері достатньо будь-якого сучасного пристрою з процесором рівня Intel Core i3 третього покоління або еквівалентним AMD процесором, що відповідає базовим вимогам для роботи сучасних веб-браузерів. Оперативна пам'ять обсягом від 4 ГБ забезпечує комфортну роботу операційної системи та браузера з кількома відкритими вкладками. Стабільне інтернет-з'єднання зі швидкістю від 5 Мбіт/с для завантаження та 1 Мбіт/с для вивантаження даних достатнє для комфортної роботи з платформою, швидкого завантаження сторінок, відображення графіків та синхронізації даних з сервером. Підтримуються всі сучасні веб-браузери: Google Chrome версії 90 або новіша, Mozilla Firefox версії 88 або новіша, Microsoft Edge версії 90 або новіша, Safari версії 14 або новіша для користувачів macOS та iOS. Роздільна здатність екрана рекомендується від 1366x768 пікселів для комфортного

відображення всіх елементів інтерфейсу без необхідності постійного прокручування.

Мобільні пристрої повинні відповідати мінімальним вимогам сучасних веб-браузерів та забезпечувати достатню продуктивність для роботи з інтерактивними елементами інтерфейсу та візуалізацією графіків. Для пристроїв на базі Android необхідна версія операційної системи 8.0 Oreo або новіша з обсягом оперативної пам'яті не менше 2 ГБ, що забезпечує плавну роботу інтерфейсу та коректне відображення всіх компонентів. Процесор повинен підтримувати 64-бітну архітектуру та мати частоту не менше 1.5 ГГц. Роздільна здатність екрана від 720x1280 пікселів забезпечує читабельність тексту та зручність взаємодії з елементами управління. Для пристроїв на базі iOS підтримуються моделі iPhone 7 та новіші, iPad 5-го покоління та новіші з операційною системою iOS 13 або новішою версією. Ці пристрої забезпечують достатню продуктивність для роботи з React додатками та відображення інтерактивних графіків через бібліотеку Recharts.

Додаткові вимоги для оптимальної роботи включають постійне підключення до Інтернету для синхронізації даних між пристроями, достатній вільний простір в браузері для зберігання локальних даних кешу, увімкнені JavaScript та cookies для коректної роботи всіх функцій додатку. Рекомендується регулярне оновлення браузера до останньої версії для отримання найновіших виправлень безпеки та поліпшень продуктивності. Для користувачів з обмеженнями зору передбачена підтримка стандартних інструментів збільшення тексту браузера та висококонтрастних тем операційної системи.

5.1.3 Серверні вимоги для розгортання

Апаратна конфігурація серверного обладнання має критичне значення для забезпечення надійної та продуктивної роботи інформаційної платформи. Вимоги визначаються очікуваною кількістю одночасних користувачів системи, інтенсивністю обробки запитів, обсягом даних що зберігаються та характером навантаження на різні компоненти системи. Правильне планування серверної

інфраструктури запобігає проблемам продуктивності в майбутньому та забезпечує можливість гнучкого масштабування.

Для тестового середовища або малих інсталяцій до 100 активних користувачів достатньо віртуального сервера базової конфігурації. Процесор з 2 віртуальними ядрами на базі Intel Xeon або AMD EPYC забезпечує обробку типових запитів до API та виконання розрахунків персоналізації програм. Оперативна пам'ять обсягом 4 ГБ розподіляється між операційною системою (приблизно 1 ГБ), FastAPI сервером (1-1.5 ГБ), MongoDB (1-1.5 ГБ) та резервом для системного кешу. Дисковий простір обсягом 50 ГБ включає місце для операційної системи (10 ГБ), встановлених залежностей (5 ГБ), бази даних MongoDB (20 ГБ з урахуванням зростання), логів системи (5 ГБ) та резерву для тимчасових файлів (10 ГБ). Мережевий інтерфейс з пропускною здатністю 100 Мбіт/с задовольняє потреби невеликої кількості користувачів без відчутних затримок.

Для production середовища з прогнозованою кількістю до 1000 одночасних користувачів необхідна значно більш потужна конфігурація для забезпечення стабільної роботи під високим навантаженням. Процесор з 4 фізичними або 8 віртуальними ядрами дозволяє ефективно обробляти велику кількість паралельних запитів завдяки асинхронній архітектурі FastAPI та можливостям багатопроцесорної обробки. Рекомендується використання процесорів Intel Xeon серії E5 або новіших, AMD EPYC серії 7002 або новіших, які оптимізовані для серверних навантажень та забезпечують високу надійність. Оперативна пам'ять обсягом 16 ГБ забезпечує достатній простір для кешування часто запитуваних даних у MongoDB, обробки множинних одночасних запитів у FastAPI, роботи операційної системи та системних сервісів. Швидкість оперативної пам'яті ECC DDR4 з частотою від 2666 МГц забезпечує виявлення та корекцію помилок, що критично важливо для збереження цілісності даних користувачів.

Накопичувач SSD обсягом від 200 ГБ розподіляється наступним чином: операційна система та системні утиліти займають близько 20 ГБ, встановлені залежності Python та системні пакети потребують 10-15 ГБ, база даних MongoDB з урахуванням індексів та зростання даних займає 100-120 ГБ, логи додатку та

системи потребують 20-30 ГБ, резерв для тимчасових файлів, backup копій та оновлень становить 30-50 ГБ. Використання SSD накопичувачів замість традиційних HDD критично важливе для продуктивності MongoDB, оскільки операції читання та запису в базу даних створюють інтенсивне навантаження на підсистему введення-виведення. Рекомендується використання enterprise-класу SSD з технологією NVMe для максимальної продуктивності, підтримкою TRIM команд для підтримки продуктивності протягом тривалого часу, та гарантованим ресурсом перезапису не менше 1 DWPD (Drive Writes Per Day) протягом гарантійного періоду.

Мережева підсистема повинна забезпечувати високу пропускну здатність та низьку затримку для обслуговування множинних одночасних підключень. Мережевий інтерфейс з пропускною здатністю 1 Гбіт/с є мінімальною вимогою для production середовища, а для більш завантажених систем рекомендується 10 Гбіт/с з підтримкою технологій агрегації каналів для забезпечення відмовостійкості та балансування навантаження. Безперебійне живлення з достатнім часом автономної роботи для коректного завершення всіх процесів та збереження даних у випадку відключення електроенергії є обов'язковою вимогою для критично важливих систем.

Масштабування системи для більшої кількості користувачів здійснюється шляхом горизонтального масштабування з використанням балансування навантаження між кількома серверними інстансами. Така архітектура передбачає розгортання кількох ідентичних серверів додатків за балансувальником навантаження, таким як Nginx або HAProxy, який розподіляє вхідні запити між доступними серверами на основі їх поточного завантаження. Для MongoDB рекомендується конфігурація replica set з мінімум трьома вузлами для забезпечення високої доступності та можливості автоматичного відновлення після відмови одного з серверів. При подальшому зростанні можливе впровадження sharding для розподілу даних між кількома серверами бази даних. Альтернативним підходом є впровадження хмарної інфраструктури на базі Amazon Web Services, Google Cloud Platform або Microsoft Azure з використанням сервісів автоматичного

масштабування, керованих баз даних та балансувальників навантаження, що дозволяє динамічно змінювати обсяг виділених ресурсів залежно від поточного навантаження системи.

5.2 Програмне забезпечення та залежності

5.2.1 Операційні системи

Розроблена платформа підтримує розгортання на різних операційних системах завдяки використанню кросплатформних технологій та стандартизованих протоколів взаємодії між компонентами. Вибір операційної системи для production середовища впливає на продуктивність, безпеку, вартість ліцензування та доступність технічної підтримки.

Серверна частина демонструє найкращу сумісність з операційними системами сімейства Linux, що є стандартом для веб-серверів завдяки стабільності, безпеці, відкритому вихідному коду та відсутності ліцензійних витрат. Ubuntu Server 18.04 LTS або новіші версії рекомендуються як основний вибір завдяки широкій підтримці спільноти, великій кількості доступної документації, регулярним оновленням безпеки протягом п'яти років для LTS версій, офіційній підтримці від Canonical з можливістю отримання комерційної технічної підтримки. Debian 10 Buster або новіші версії є альтернативою для тих, хто надає перевагу більш консервативному підходу до оновлень з акцентом на стабільність, відомий своєю надійністю та тривалою підтримкою, має величезну базу доступних пакетів через apt менеджер пакетів. CentOS 7 або 8 традиційно використовуються в корпоративному середовищі завдяки сумісності з Red Hat Enterprise Linux, проте з 2021 року рекомендується розглянути альтернативи такі як AlmaLinux або Rocky Linux, які продовжують філософію CentOS як безкоштовної стабільної серверної операційної системи.

Операційні системи Windows Server 2016 або новіші версії також підтримуються для розгортання серверної частини завдяки сумісності Python та MongoDB з Windows платформою. Ця опція може бути обрана організаціями, що вже мають Windows інфраструктуру та відповідні компетенції адміністрування, проте слід враховувати вищу вартість ліцензування порівняно з Linux системами, дещо нижчу продуктивність веб-серверів у Windows середовищі порівняно з Linux, більші вимоги до апаратних ресурсів операційної системи. Для Windows середовища рекомендується використання Windows Server 2019 або 2022 з активованою роллю IIS для проксування запитів та останніми оновленнями безпеки.

Для середовища розробки підтримується максимально широкий спектр операційних систем для забезпечення зручності розробників. Windows 10 версії 1909 або новіша, Windows 11 підтримують всі необхідні інструменти розробки через нативну підтримку або WSL2 (Windows Subsystem for Linux), що дозволяє запускати Linux середовище безпосередньо у Windows без необхідності віртуальних машин. macOS 10.15 Catalina або новіші версії включаючи macOS 12 Monterey та 13 Ventura забезпечують Unix-подібне середовище з повною підтримкою необхідних інструментів, рекомендуються для розробників що надають перевагу екосистемі Apple, проте потребують інсталяції Xcode Command Line Tools для компіляції деяких Python пакетів. Linux дистрибутиви такі як Ubuntu 20.04 LTS або новіші, Fedora Workstation, Linux Mint забезпечують найближче до production середовище для розробки, мають найкращу підтримку інструментів розробки, проте можуть потребувати додаткового часу на налаштування для початківців.

5.2.2 Системні залежності

Функціонування платформи потребує встановлення наступних системних компонентів, що забезпечують виконання коду та взаємодію між різними частинами системи. Правильна інсталяція та конфігурація цих компонентів критично важлива для стабільної роботи додатку.

Python версії 3.8 або новішої є основою для виконання серверного коду, що реалізує бізнес-логіку додатку, обробку API запитів та взаємодію з базою даних. Вибір Python 3.8 як мінімальної версії обумовлений наявністю важливих функцій таких як assignment expressions (walrus operator), positional-only parameters, typing improvements, які активно використовуються в коді додатку. Рекомендується інсталяція Python 3.9 або 3.10 для отримання додаткових поліпшень продуктивності та нових можливостей мови. Критично важливо встановлювати Python разом з pip менеджером пакетів для інсталяції залежностей проекту, venv модулем для створення віртуальних середовищ, development headers для компіляції бінарних розширень деяких пакетів. У Linux системах Python зазвичай вже встановлений, проте може знадобитися інсталяція додаткових пакетів python3-pip, python3-venv, python3-dev. У Windows та macOS рекомендується завантаження офіційного інсталятора з python.org з обов'язковим додаванням Python до системного PATH.

Node.js версії 18 LTS або новішої необхідна для роботи клієнтської частини, інструментів збірки та транспіляції TypeScript коду. Вибір версії 18 LTS гарантує тривалу підтримку (до квітня 2025 року), стабільність та сумісність з усіма використовуваними пакетами. Node.js включає npm (Node Package Manager) для інсталяції залежностей frontend частини, підтримку ES модулів та нових можливостей JavaScript, V8 engine для швидкого виконання JavaScript коду. Рекомендується встановлення через nvm (Node Version Manager) для Linux та macOS або nvm-windows для Windows, що дозволяє легко перемикатися між різними версіями Node.js, встановлювати кілька версій паралельно для різних проектів, оновлювати Node.js без впливу на інші проекти. Альтернативно можна використовувати офіційні інсталятори з nodejs.org або менеджери пакетів операційної системи, проте в цьому випадку складніше керувати версіями.

MongoDB версії 4.4 або новішої виконує роль основної бази даних для зберігання користувацьких профілів, біометричних показників, тренувальних програм та іншої інформації. Вибір MongoDB обумовлений гнучкістю схеми даних для зберігання різнотипних біометричних показників, високою продуктивністю

для операцій читання та запису, природною підтримкою JSON формату для взаємодії з JavaScript та Python, горизонтальним масштабуванням через sharding для великих навантажень. Мінімальна версія 4.4 необхідна для підтримки сучасних можливостей таких як Hedged Reads, покращена обробка транзакцій, Union та Set операції в aggregation pipeline.

Рекомендується встановлення MongoDB 5.0 або новішої для отримання додаткових поліпшень продуктивності, нової системи моніторингу та покращеної безпеки. MongoDB може бути встановлена локально на сервері додатку для невеликих інсталяцій або розгорнута як окремий кластер з replica set для забезпечення високої доступності. Альтернативою локальної інсталяції є використання хмарного сервісу MongoDB Atlas, що надає повністю керовану базу даних з автоматичним резервним копіюванням, моніторингом, масштабуванням та підтримкою, безкоштовний tier на 512 МБ для розробки та тестування, географічно розподілені кластери для мінімальної затримки.

Додатково рекомендується встановлення наступних системних компонентів. Система контролю версій Git необхідна для клонування репозиторію з вихідним кодом, відстеження змін у коді під час розробки, співпраці між кількома розробниками через GitHub, GitLab або інші сервіси. Веб-сервер Nginx або Apache виконує роль reverse proxy у production середовищі для обробки SSL/TLS термінації та шифрування трафіку, балансування навантаження між кількома інстансами додатку, обслуговування статичних файлів frontend частини, компресії даних для економії трафіку.

Менеджер процесів PM2 або systemd забезпечує автоматичний запуск додатку при старті сервера, перезапуск при аварійних збоях, логування output додатку, моніторинг використання ресурсів. Certbot або інші ACME клієнти необхідні для автоматичного отримання та оновлення безкоштовних SSL сертифікатів від Let's Encrypt.

5.3 Мережеві вимоги та конфігурація

5.3.1 Порти та протоколи

Коректне функціонування системи потребує доступності певних мережевих портів для комунікації між різними компонентами та зовнішніми клієнтами. Правильна конфігурація портів та протоколів забезпечує безпеку системи через обмеження доступу лише до необхідних сервісів та запобігає конфліктам з іншими додатками.

Порт 8000 використовується FastAPI сервером для прийому HTTP запитів від клієнтів у режимі розробки. Цей порт обраний як альтернатива стандартному порту 80 для можливості запуску сервера без root привілеїв у Linux системах, уникнення конфліктів з іншими веб-серверами на тій же машині, зручності розробки з можливістю швидкого перезапуску сервера. У production середовищі рекомендується налаштування reverse proxy через Nginx або Apache на стандартному порту 80/443 з проксуванням запитів на внутрішній порт 8000, що забезпечує додатковий рівень безпеки, можливість обслуговування кількох додатків на одному сервері, централізоване керування SSL сертифікатами. Конфігурація FastAPI для прослуховування на локальному інтерфейсі 127.0.0.1:8000 запобігає прямому доступу з зовнішньої мережі, забезпечуючи додатковий рівень безпеки.

Порт 3000 використовується Vite development сервером для обслуговування React додатку під час розробки. Цей порт забезпечує Hot Module Replacement для миттєвого відображення змін в коді без повного перезавантаження сторінки, швидку збірку завдяки використанню native ES modules, проксування API запитів на backend сервер для уникнення CORS проблем. У production середовищі frontend частина збирається в статичні файли командою npm run build та обслуговується веб-сервером Nginx або Apache, тому порт 3000 не потребує відкриття у firewall production сервера. Для команди розробників може бути корисним відкриття порту

3000 на development сервері для можливості перегляду останніх змін колегами через локальну мережу або VPN з'єднання.

Порт 27017 є стандартним портом MongoDB для прийому підключень від клієнтів. Цей порт повинен бути доступний FastAPI серверу для виконання операцій читання та запису в базу даних, проте має бути закритий для зовнішнього доступу через firewall для запобігання несанкціонованому доступу до даних. Рекомендації з безпеки MongoDB включають налаштування аутентифікації з сильними паролями, обмеження підключень до конкретних IP адрес через bindIp параметр конфігурації, використання TLS/SSL шифрування для з'єднань між додатком та базою даних, регулярне оновлення MongoDB до останніх версій з виправленнями безпеки. У разі використання MongoDB в окремому сервері або кластері необхідно налаштувати безпечний канал зв'язку через VPN або приватну мережу дата-центру, регулярно аудитувати доступ до бази даних через логи MongoDB, впровадити ротацію паролів та використання окремих облікових записів для різних сервісів.

Порт 80 використовується для HTTP трафіку у production середовищі та є стандартним портом веб-протоколу, що відкривається автоматично всіма браузерами при відсутності явного вказання порту в URL. Цей порт обслуговується веб-сервером Nginx або Apache, що виконує роль reverse proxy для FastAPI додатку, обробку статичних файлів frontend частини, автоматичне перенаправлення HTTP запитів на HTTPS для забезпечення безпеки, компресію gzip або brotli для зменшення розміру переданих даних. Конфігурація повинна включати правила перенаправлення всього HTTP трафіку на HTTPS через код відповіді 301 Moved Permanently для забезпечення використання шифрованого з'єднання, заголовки безпеки такі як HSTS для примусового використання HTTPS браузерами, обмеження розміру тіла запиту для запобігання DoS атакам.

Порт 443 використовується для HTTPS трафіку з SSL/TLS шифруванням та є обов'язковим для production середовища для захисту конфіденційних даних користувачів, запобігання перехопленню паролів та особистої інформації, забезпечення довіри користувачів через відображення замка безпеки в браузері,

відповідності сучасним вимогам безпеки та стандартам індустрії. Налаштування SSL/TLS включає отримання сертифіката від довіреного центру сертифікації (рекомендується Let's Encrypt для безкоштовних автоматично оновлюваних сертифікатів), конфігурацію веб-сервера для використання сучасних протоколів TLS 1.2 та 1.3 з відключенням застарілих SSL 3.0, TLS 1.0, TLS 1.1, вибір безпечних cipher suites з пріоритетом для forward secrecy, регулярне оновлення сертифікатів до закінчення терміну дії через автоматизовані скрипти Certbot. Додаткові заходи безпеки включають налаштування HTTP Strict Transport Security заголовка для запобігання downgrade атакам, OCSP Stapling для прискорення перевірки статусу сертифіката, моніторинг терміну дії сертифіката для уникнення несподіваного закінчення.

5.3.2 Пропускна здатність мережі

Мінімальна пропускна здатність мережевого з'єднання визначається характером використання системи, кількістю одночасних користувачів та типами операцій що виконуються. Недостатня пропускна здатність призводить до затримок завантаження сторінок, таймаутів запитів та погіршення користувацького досвіду.

Для одиночного користувача достатньо швидкості завантаження 5 Мбіт/с для комфортного завантаження сторінок додатку включаючи HTML, CSS, JavaScript файли розміром близько 500 КБ після компресії, зображення інтерфейсу та іконки загальним розміром близько 200 КБ, відображення графіків з даними за кілька місяців об'ємом близько 50-100 КБ. Швидкість вивантаження 1 Мбіт/с достатня для передачі даних на сервер включаючи форми реєстрації та профілю з текстовими даними розміром кілька кілобайт, біометричні показники що передаються як JSON об'єкти розміром 1-2 КБ на запис, синхронізацію даних між пристроями. Затримка (ping) не повинна перевищувати 200 мс для забезпечення відчуття миттєвого відгуку інтерфейсу, оскільки більші затримки помітні користувачам та створюють відчуття повільної роботи додатку навіть при достатній пропускній здатності.

Серверне середовище потребує значно більшої пропускної здатності для обслуговування множинних одночасних підключень. Симетричний канал зв'язку зі

швидкістю не менше 100 Мбіт/с підходить для малих та середніх інсталяцій до 500 одночасних користувачів, забезпечує пропускну здатність близько 12.5 МБ/с що еквівалентно обслуговуванню 25 запитів по 500 КБ кожна секунду, дозволяє обробляти піки навантаження без значної деградації продуктивності, має запас для трафіку резервного копіювання, оновлень та моніторингу. Для більших інсталяцій рекомендується канал 1 Гбіт/с що забезпечує пропускну здатність 125 МБ/с для обслуговування тисяч одночасних користувачів, дозволяє впроваджувати CDN (Content Delivery Network) для розподілу статичного контенту, забезпечує резерв для майбутнього зростання без необхідності негайного оновлення інфраструктури.

Якість мережевого з'єднання не менш важлива за пропускну здатність. Стабільність з'єднання характеризується відсутністю пакетної втрати (packet loss) більше 0.1%, оскільки втрачені пакети призводять до повторних передач та збільшення загальної затримки, низьким jitter (варіацією затримки) менше 10 мс для передбачуваного часу відгуку, відсутністю періодичних відключень що призводять до розриву активних з'єднань. Для критично важливих систем рекомендується резервування каналу зв'язку через різних провайдерів або різні технології (наприклад, оптоволокно та мобільний інтернет) для забезпечення доступності навіть при відмові основного каналу, автоматичне перемикання на резервний канал при виявленні проблем з основним, моніторинг якості з'єднання та сповіщення адміністраторів про деградацію продуктивності.

5.3.3 Безпека мережі

Конфігурація мережевої безпеки є важливим аспектом захисту інформаційної платформи від несанкціонованого доступу, кібератак та витоку конфіденційних даних користувачів. Багаторівневий підхід до безпеки забезпечує захист навіть при компрометації одного з рівнів захисту.

Налаштування firewall для обмеження доступу лише до необхідних портів є першим рівнем захисту мережевого периметру. У Linux системах рекомендується використання UFW (Uncomplicated Firewall) або firewalld для спрощеного керування правилами iptables, що дозволяє дозволити вхідні з'єднання лише на порти 80 та 443 для веб-трафіку, порт 22 для SSH доступу адміністраторів з

обмеженням по IP адресам, заборонити всі інші вхідні з'єднання за замовчуванням. Вихідні з'єднання зазвичай дозволяються для можливості оновлення системи, встановлення залежностей та взаємодії з зовнішніми API, проте в середовищах з підвищеними вимогами до безпеки можуть бути обмежені до конкретних портів та призначень. Правила firewall повинні регулярно аудитуватися для виявлення застарілих правил, логуватися спроби підключення для виявлення спроб несанкціонованого доступу, автоматично оновлюватися при зміні конфігурації сервісів. Додатковий захист надають мережеві брандмауери рівня дата-центру, DDoS захист на рівні провайдера або спеціалізованих сервісів як Cloudflare, географічні обмеження доступу якщо додаток призначений для конкретних країн.

Використання SSL/TLS сертифікатів для шифрування трафіку захищає конфіденційні дані користувачів під час передачі по мережі від перехоплення та аналізу. Сертифікати повинні бути отримані від довірених центрів сертифікації, оскільки самопідписані сертифікати викликають попередження в браузерах та підривають довіру користувачів. Let's Encrypt надає безкоштовні автоматично оновлювані сертифікати терміном дії 90 днів з можливістю автоматичного оновлення через Certbot утиліту, підтримку wildcard сертифікатів для всіх субдоменів, високий рівень довіри з включенням у всі популярні браузери та операційні системи. Конфігурація веб-сервера повинна використовувати сучасні версії протоколу TLS 1.2 та 1.3 з відключенням застарілих версій, сильні cipher suites з пріоритетом для ECDHE та AEAD алгоритмів, досконалу прямую секретність (Perfect Forward Secrecy) для захисту історичних даних навіть при компрометації приватного ключа в майбутньому. Якість конфігурації SSL/TLS можна перевірити через онлайн сервіси такі як SSL Labs SSL Server Test, що надають детальний звіт про підтримувані протоколи, cipher suites, та рекомендації з покращення.

Регулярне оновлення системи та залежностей для усунення відомих вразливостей є критично важливою практикою для підтримки безпеки системи. Операційна система повинна налаштовуватися для автоматичного встановлення оновлень безпеки через unattended-upgrades в Debian/Ubuntu або автоматичні

оновлення в інших дистрибутивах, регулярної перевірки наявності оновлень для встановлених пакетів, планового перезавантаження сервера для застосування оновлень ядра якщо не використовується live patching. Python залежності повинні регулярно оновлюватися через pip з перевіркою на вразливості за допомогою інструментів типу safety або pip-audit, моніторингом security advisories для використовуваних пакетів, тестуванням оновлень у development середовищі перед застосуванням у production. Node.js залежності потребують особливої уваги через велику кількість транзитивних залежностей: npm audit виявляє відомі вразливості у встановлених пакетах, Dependabot або Renovate автоматично створюють pull requests з оновленнями, package-lock.json фіксує точні версії всіх залежностей для відтворюваності збірки.

Додаткові заходи безпеки включають впровадження Web Application Firewall для захисту від поширених веб-атак як SQL injection, XSS, CSRF, налаштування rate limiting для запобігання brute force атакам та зловживанню API, регулярні резервні копії з перевіркою можливості відновлення, моніторинг безпеки через системи збору логів та виявлення аномалій, регулярні security аудити коду та інфраструктури, навчання команди розробки безпечним практикам програмування. Для додатків що обробляють особливо чутливу інформацію може знадобитися впровадження додаткових заходів таких як двофакторна аутентифікація, шифрування даних на рівні бази даних, аудит доступу до даних, відповідність регуляторним вимогам як GDPR або HIPAA.

5.4 Резервне копіювання та відновлення

5.4.1 Стратегія резервного копіювання

Збереження цілісності даних користувачів є критично важливим аспектом експлуатації інформаційної платформи, оскільки втрата біометричних показників, історії тренувань та персональних налаштувань може призвести до втрати довіри

користувачів та репутаційних збитків. Комплексна стратегія резервного копіювання повинна забезпечувати мінімальну втрату даних при будь-яких сценаріях відмови обладнання або програмного забезпечення.

Щоденні інкрементні резервні копії бази даних MongoDB забезпечують баланс між обсягом даних що зберігаються та повнотою відновлення. Інкрементний підхід зберігає лише зміни з моменту останньої повної копії, що значно зменшує обсяг необхідного дискового простору для зберігання backup копій, скорочує час виконання backup процедури та мінімізує вплив на продуктивність production системи, дозволяє зберігати більшу кількість backup точок при тих же витратах на сховище. Реалізація інкрементного backup для MongoDB здійснюється через mongodump утиліту з параметром --oplog для збереження змін, MongoDB Atlas Backup сервіс з автоматичним керуванням інкрементними копіями, або спеціалізовані backup рішення як Percona Backup for MongoDB. Час виконання щоденного backup планується на період найменшого навантаження системи, зазвичай між 2 та 4 годинами ночі за місцевим часом, з автоматичним сповіщенням адміністраторів про успішне завершення або помилки в процесі.

Щотижневі повні резервні копії усієї системи забезпечують можливість повного відновлення у випадку катастрофічного збою. Повна копія включає повний дамп бази даних MongoDB зі всіма колекціями, індексами та даними, конфігураційні файли системи включаючи налаштування веб-серверів, SSL сертифікати та приватні ключі, файли додатку включаючи вихідний код або зібрані бінарні файли, системні логи для можливості аналізу проблем після відновлення. Щотижневі backup виконуються у вихідні дні коли навантаження системи зазвичай мінімальне, з автоматичним розміщенням копій на віддаленому сервері або в хмарному сховищі, з шифруванням перед передачею для захисту конфіденційних даних. Механізм ротації backup копій забезпечує видалення старих копій після певного періоду для економії дискового простору: щоденні копії зберігаються протягом 7 днів, щотижневі копії зберігаються протягом 4 тижнів, щомісячні копії зберігаються протягом 12 місяців для можливості відновлення історичних даних.

Зберігання резервних копій на окремому фізичному носії або в хмарному сховищі критично важливе для захисту від фізичних пошкоджень основного сервера. Локальні копії на окремих дисках або NAS пристроях забезпечують швидке відновлення завдяки високій швидкості локальної мережі, проте вразливі до фізичних катастроф як пожежа або потоп, вимагають додаткового обслуговування та заміни носіїв. Хмарні сховища такі як Amazon S3, Google Cloud Storage або Azure Blob Storage надають високу надійність з реплікацією між різними дата-центрами, практично необмежену масштабованість для зростання об'єму backup, географічну розподіленість для захисту від регіональних катастроф, автоматичне керування життєвим циклом об'єктів для ротації старих backup. Гібридний підхід поєднує локальні та хмарні backup для максимальної надійності: локальні копії для швидкого відновлення при незначних збоях, хмарні копії як остаточна страховка від катастрофічних сценаріїв.

Шифрування backup копій перед зберіганням захищає конфіденційні дані користувачів навіть при компрометації backup сховища. Використання сильних алгоритмів шифрування як AES-256 з унікальними ключами для кожної копії забезпечує захист даних, безпечне зберігання ключів шифрування в окремому сховищі як HashiCorp Vault, регулярна ротація ключів для мінімізації впливу потенційної компрометації запобігають несанкціонованому доступу до історичних backup копій.

Автоматизація процесу backup через cron jobs в Linux системах або Task Scheduler у Windows забезпечує регулярність виконання без людського втручання, зменшує ризик пропуску backup через людський фактор, дозволяє централізоване керування backup політиками для множинних серверів. Скрипти backup повинні включати перевірку вільного дискового простору перед початком, логування всіх операцій для аудиту та діагностики, перевірку цілісності створених backup копій, сповіщення адміністраторів про проблеми через email або системи моніторингу.

5.4.2 Процедура відновлення

У випадку необхідності відновлення системи після збою важлива наявність чіткої та протестованої процедури, що дозволяє мінімізувати час простою та втрату

даних. Регулярне тестування процедури відновлення в непродуктивному середовищі забезпечує впевненість у можливості успішного відновлення в критичній ситуації.

Відновлення бази даних з останньої резервної копії є першочерговим завданням, оскільки дані користувачів є найціннішим активом системи. Процедура включає встановлення MongoDB на нову інстанцію якщо необхідно або очищення існуючої бази даних для повного відновлення, копіювання backup файлів з віддаленого сховища на сервер бази даних, розшифрування backup копії якщо використовувалося шифрування, виконання mongorestore з відповідними параметрами для відновлення даних, відновлення oplog для застосування інкрементних змін якщо використовуються point-in-time відновлення. Час відновлення залежить від розміру бази даних та швидкості дискової підсистеми, зазвичай складаючи від кількох хвилин для невеликих баз до кількох годин для баз розміром у сотні гігабайт. Під час відновлення MongoDB система повинна бути недоступна для користувачів для запобігання неконсистентності даних.

Перевірка цілісності відновлених даних критично важлива для впевненості що backup копія була коректною та процес відновлення виконався успішно. Автоматизовані тести повинні перевіряти наявність очікуваних колекцій в базі даних, коректність індексів включаючи унікальні та геопросторові індекси, кількість документів у критичних колекціях порівняно з очікуваними значеннями, можливість виконання типових запитів без помилок. Ручна перевірка вибірки даних додатково підтверджує коректність відновлення через перегляд профілів кількох тестових користувачів, перевірку наявності останніх біометричних записів, тестування функціональності входу та отримання даних через API. Логи MongoDB та додатку аналізуються на предмет помилок що могли виникнути під час відновлення.

Поетапне відновлення функціональності з тестуванням кожного компоненту дозволяє виявити проблеми на ранніх стадіях перед повним відкриттям доступу користувачам. Спочатку відновлюється та тестується backend частина через виконання health check endpoints для перевірки з'єднання з базою даних, тестування

аутентифікації через спробу входу тестовими користувачами, перевірку API endpoints для отримання профілю, біометричних даних, тренувальних програм, моніторинг логів на предмет помилок підключення або некоректних запитів. Після підтвердження коректної роботи backend відновлюється frontend частина через розгортання статичних файлів на веб-сервері, налаштування проксування запитів до backend, перевірку доступності додатку через браузер, тестування повного циклу роботи користувача від реєстрації до додавання даних. Поступове відновлення сервісів дозволяє ізолювати проблеми до конкретних компонентів та спростити діагностику.

Документування процесу відновлення та виявлених проблем важливе для покращення процедури та підготовки до майбутніх інцидентів. Детальний звіт повинен включати час виявлення проблеми та час повного відновлення сервісу для аналізу RTO (Recovery Time Objective), причину необхідності відновлення та кроки що призвели до проблеми, використану backup копію та причину вибору саме цієї копії, проблеми що виникли під час відновлення та способи їх вирішення, рекомендації з покращення процедури backup та відновлення. Після кожного реального відновлення або періодичних тестувань процедура повинна оновлюватися з врахуванням набутого досвіду.

5.5 Моніторинг та підтримка

5.5.1 Системи моніторингу

Контроль працездатності системи та раннє виявлення потенційних проблем є критично важливими для забезпечення безперервної роботи інформаційної платформи та задоволення користувачів. Комплексний моніторинг охоплює всі рівні системи від апаратного забезпечення до поведінки додатку.

Моніторинг використання апаратних ресурсів дозволяє виявляти вузькі місця продуктивності та планувати масштабування до виникнення критичних проблем.

Процесор відстежується через показники загального використання CPU в відсотках, використання по окремих ядрах для виявлення дисбалансу навантаження, load average за 1, 5 та 15 хвилин для оцінки тренду навантаження, CPU steal time у віртуальних середовищах що вказує на ресурсну конкуренцію. Критичні пороги встановлюються на рівні 80% середнього використання CPU протягом 5 хвилин для попередження про високе навантаження, 95% для критичного сповіщення про перевантаження системи. Оперативна пам'ять моніториться через показники загального використання RAM включаючи буфери та кеш, swap використання що вказує на нестачу фізичної пам'яті та погіршення продуктивності, memory pressure в сучасних Linux системах, per-process використання пам'яті для виявлення витоків. Сповіщення налаштовуються при досягненні 85% використання RAM або будь-якому використанні swap простору. Дисковий простір відстежується через показники вільного місця на всіх розділах особливо критичних /var для логів, використання inodes в Linux системах, швидкість зростання використання для прогнозування заповнення. Сповіщення надсилаються при заповненні диска на 85% з критичним порогом 95%.

Моніторинг дискової підсистеми включає показники IOPS (операцій введення-виведення на секунду) для виявлення вузьких місць продуктивності, latency дискових операцій особливо важливе для MongoDB продуктивності, утилізацію дискової підсистеми що показує наскільки система близька до насичення. Мережеві метрики відстежують пропускну здатність вхідного та вихідного трафіку для планування розширення каналів, кількість відкритих TCP з'єднань для виявлення проблем з вичерпанням портів, кількість помилок передачі пакетів що вказує на проблеми з обладнанням або перевантаження. Інструменти як Prometheus з Grafana надають централізований збір метрик з множинних серверів, гнучку систему сповіщень через різні канали, візуалізацію трендів для аналізу історичних даних, API для інтеграції з іншими системами.

Моніторинг часу відгуку API endpoints критично важливий для забезпечення задовільного користувацького досвіду. Вимірюється середній час відгуку для різних типів запитів з метою виявлення деградації продуктивності, 95-й та 99-й

перцентилі часу відгуку для розуміння worst-case сценаріїв, rate помилкових відповідей 4xx та 5xx для виявлення проблем з додатком, розподіл часу відгуку по endpoints для ідентифікації найповільніших операцій. Цільові показники встановлюються виходячи з вимог до користувацького досвіду: середній час відгуку менше 200 мс для простих запитів типу отримання профілю, менше 500 мс для складних запитів типу генерації тренувальної програми, менше 1 секунди для операцій з великими обсягами даних типу історії біометричних показників. Application Performance Monitoring інструменти як New Relic, DataDog або самохостовані рішення як Jaeger надають детальну трасування запитів через різні компоненти системи, профілювання коду для виявлення повільних функцій, кореляцію між різними метриками для root cause аналізу.

Моніторинг кількості активних користувачів та сесій надає insight про використання додатку та допомагає планувати ресурси. Відстежуються метрики поточної кількості активних сесій, кількості нових реєстрацій за період, retention rate користувачів для оцінки успішності продукту, найпопулярніші функції на основі використання endpoints. Бізнес метрики як кількість створених тренувальних програм за день, середня кількість доданих біометричних записів на користувача, час проведений у додатку надають дані для product roadmap та оптимізації функціональності.

Наявність помилок у логах додатку відстежується через централізовану систему збору логів. ELK Stack (Elasticsearch, Logstash, Kibana) або Grafana Loki забезпечують агрегацію логів з множинних джерел, повнотекстовий пошук по логах для швидкої діагностики, візуалізацію трендів помилок, автоматичні сповіщення при виявленні критичних помилок. Структуроване логування в JSON форматі полегшує автоматичний аналіз та створення dashboard з метриками помилок. Рівні логування налаштовуються залежно від середовища: DEBUG для development середовища з максимальною детальністю, INFO для production з фокусом на бізнес події, WARNING та ERROR для виявлення проблем, CRITICAL для критичних ситуацій що потребують негайної реакції.

5.5.2 Технічна підтримка

Забезпечення безперервної роботи системи потребує організації технічної підтримки з чіткими процедурами моніторингу, реагування на інциденти та планового обслуговування. Проактивний підхід до підтримки дозволяє виявляти та вирішувати потенційні проблеми до того як вони вплинуть на користувачів.

Регулярний аналіз логів для виявлення потенційних проблем повинен виконуватися щоденно через перегляд звітів про помилки за попередній день з класифікацією за критичністю, аналіз трендів типу "збільшення кількості timeout помилок може вказувати на проблеми з базою даних", пошук незвичних патернів активності що можуть вказувати на спроби злому або некоректну роботу додатку, перевірку логів безпеки на предмет неуспішних спроб входу або підозрілої активності. Автоматизація через alerting rules значно спрощує цей процес направляючи увагу адміністраторів на дійсно важливі події замість ручного перегляду мільйонів log записів. Weekly звіти про стан системи включають зведену статистику помилок, метрики продуктивності з порівнянням з попередніми періодами, рекомендації з оптимізації та покращення на основі виявлених патернів [9].

Планове оновлення компонентів системи є критично важливою практикою для підтримки безпеки та отримання нових функцій. Політика оновлень повинна визначати частоту перевірки оновлень щотижня для security patches, щомісяця для minor версій, щоквартально для major версій з breaking changes, процедуру тестування оновлень спочатку в staging середовищі ідентичному production, запуск автоматизованих тестів та ручної перевірки критичної функціональності, схвалення deployment після успішного тестування. Maintenance window для застосування оновлень планується у період найменшого навантаження з попереднім сповіщенням користувачів за 24-48 годин, з можливістю швидкого rollback при виявленні критичних проблем, з детальним логуванням всіх змін для можливості аудиту. Zero-downtime deployment стратегії як blue-green deployment або rolling updates мінімізують вплив на користувачів під час оновлень.

Оперативне реагування на інциденти та збої в роботі потребує чіткої incident response процедури. On-call rotation забезпечує доступність відповідальної особи 24/7 для реагування на критичні інциденти через pagerduty або аналогічні системи, з чіткими SLA на час реагування залежно від severity, з ескалаційною процедурою якщо перша лінія підтримки не може вирішити проблему. Incident response включає етапи виявлення та підтвердження інциденту через моніторинг або звернення користувачів, первинної діагностики для визначення масштабу проблеми та затронутих компонентів, mitigation для швидкого відновлення сервісу навіть якщо root cause ще не знайдена, детального аналізу причин після відновлення сервісу, документування інциденту з post-mortem звітом. Post-mortem без звинувачення фокусується на системних проблемах а не людських помилках, включає timeline подій, root cause аналіз, corrective actions для запобігання повторенню, lessons learned для покращення процесів.

Документування всіх процедур підтримки та troubleshooting є критично важливим для швидкого онбордингу нових членів команди та забезпечення консистентності дій. Runbook документація повинна включати common issues з симптомами та способами вирішення, command cheat sheets для швидкого виконання типових операцій, контактну інформацію для ескалації, diagram архітектури системи для розуміння залежностей. Knowledge base накопичує історичні інциденти з детальним описом проблеми, кроками вирішення, root cause та превентивними заходами, регулярно оновлюється з врахуванням нових випадків. Регулярні training sessions для команди підтримки включають disaster recovery drills для практики процедур відновлення, security incident simulation для підготовки до кібератак, updates про нову функціональність та зміни в системі.

Дотримання зазначених апаратних вимог, рекомендацій щодо налаштування та процедур підтримки забезпечує стабільне, безпечне та ефективне функціонування розробленої інформаційної платформи персоналізованого фітнес-коучингу в усіх режимах експлуатації від початкового розгортання до масштабованої production системи з тисячами активних користувачів.

5.6 Висновки до розділу 5

У п'ятому розділі детально описано апаратні вимоги та процедури налаштування інформаційної платформи для різних сценаріїв використання від розробки до production експлуатації.

Визначено апаратні вимоги до системи для різних режимів роботи. Для розробки встановлено мінімальні вимоги що включають процесор Intel Core i5 або AMD Ryzen 5 з тактовою частотою не менше 2.5 ГГц, оперативну пам'ять обсягом не менше 8 ГБ DDR4, накопичувач SSD обсягом від 256 ГБ. Рекомендовані характеристики включають процесор Intel Core i7 або AMD Ryzen 7, 16 ГБ оперативної пам'яті, NVMe SSD обсягом від 512 ГБ для максимальної продуктивності середовища розробки. Для експлуатації кінцевими користувачами достатньо будь-якого сучасного пристрою з процесором рівня Intel Core i3, 4 ГБ оперативної пам'яті та стабільним інтернет-з'єднанням зі швидкістю від 5 Мбіт/с. Мобільні пристрої повинні мати Android 8.0 або iOS 13 з мінімум 2 ГБ оперативної пам'яті. Серверні вимоги для тестового середовища включають 2 віртуальні ядра процесора, 4 ГБ оперативної пам'яті, 50 ГБ дискового простору. Production середовище потребує 4 фізичні або 8 віртуальних ядер, 16 ГБ оперативної пам'яті, SSD накопичувач обсягом від 200 ГБ.

Визначено програмне забезпечення та залежності необхідні для функціонування платформи. Система підтримує операційні системи сімейства Linux включаючи Ubuntu 18.04 LTS або новіші, Debian 10 або новіші, CentOS 7 або новіші, Windows Server 2016 або новіші для серверної частини. Середовище розробки підтримує Windows 10/11, macOS 10.15 або новіші, Linux дистрибутиви. Системні залежності включають Python версії 3.8 або новішої для серверної частини, Node.js версії 18 LTS або новішої для клієнтської частини, MongoDB версії 4.4 або новішої для зберігання даних. Додатково рекомендується Git для контролю версій, Nginx або Apache для reverse proxy, PM2 або systemd для керування процесами.

Описано мережеві вимоги та конфігурацію включаючи необхідні порти та протоколи. Порт 8000 використовується FastAPI сервером, порт 3000 Vite development сервером, порт 27017 MongoDB, порт 80 для HTTP та порт 443 для HTTPS трафіку у production середовищі. Визначено вимоги до пропускної здатності мережі: мінімум 5 Мбіт/с завантаження та 1 Мбіт/с вивантаження для одиночного користувача, симетричний канал 100 Мбіт/с для серверного середовища з малими інсталяціями, 1 Гбіт/с для більших інсталяцій. Описано конфігурацію мережевої безпеки включаючи налаштування firewall для обмеження доступу до необхідних портів, використання SSL/TLS сертифікатів для шифрування трафіку, регулярне оновлення системи для усунення вразливостей.

Розроблено стратегію резервного копіювання що включає щоденні інкрементні резервні копії бази даних MongoDB, щотижневі повні резервні копії всієї системи, зберігання резервних копій на окремому фізичному носії або в хмарному сховищі, шифрування backup копій перед зберіганням, автоматизацію процесу через cron jobs або Task Scheduler. Описано процедуру відновлення включаючи відновлення бази даних з останньої резервної копії, перевірку цілісності відновлених даних через автоматизовані тести та ручну перевірку вибірки, поетапне відновлення функціональності з тестуванням кожного компоненту, документування процесу відновлення та виявлених проблем.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи розроблено та реалізовано повнофункціональну інформаційну платформу персоналізованого фітнес-коучингу з інтелектуальним аналізом біометричних даних, що успішно вирішує актуальну проблему автоматизації та індивідуалізації тренувального процесу без необхідності постійної участі професійних тренерів.

Проведений комплексний аналіз предметної області виявив критичні недоліки існуючих фітнес-додатків, серед яких відсутність глибокої персоналізації тренувальних програм на основі біометричних показників, недостатня інтеграція з науково обґрунтованими методиками розрахунку фізичних навантажень, обмежені можливості для комплексного аналізу прогресу користувачів та відсутність систем автоматичної адаптації програм. Дослідження показало, що загальні фітнес-трекери зосереджуються лише на підрахунку калорій без надання персоналізованих рекомендацій, спеціалізовані тренувальні додатки обмежуються простим вибором рівня складності, а професійні рішення залишаються недоступними для самостійного використання через високу вартість та необхідність участі тренера. Ці виявлені недоліки обґрунтували необхідність створення інтелектуальної системи, здатної автоматично генерувати та адаптувати індивідуальні тренувальні програми на основі персональних характеристик та поточного стану користувача.

На етапі проєктування системи прийнято обґрунтоване рішення про використання мікросервісної архітектури [10], що забезпечує масштабованість для зростання кількості користувачів, надійність через ізоляцію компонентів, гнучкість у виборі технологій для різних модулів та можливість незалежного розгортання та оновлення сервісів. Детальний аналіз доступних технологічних рішень призвів до формування оптимального технологічного стеку: FastAPI для серверної частини завдяки високій продуктивності асинхронної обробки запитів та автоматичній генерації документації OpenAPI, MongoDB для зберігання даних через гнучкість

схеми для різнотипних біометричних показників та природну підтримку JSON формату, React з TypeScript для клієнтської частини з метою забезпечення типобезпечності та сучасного користувацького досвіду. Спроєктовано модель даних з п'ятьма основними сутностями та визначеними зв'язками між ними, що забезпечує ефективне зберігання та обробку інформації про користувачів, їх профілі, біометричні показники, вправи та тренувальні програми.

Розроблено математичні моделі та алгоритми персоналізації, що базуються на сучасних фізіологічних принципах та науково обґрунтованих формулах. Реалізовано алгоритми розрахунку індексу маси тіла з автоматичною класифікацією за стандартами Всесвітньої організації охорони здоров'я, базового метаболізму за рівнянням Міффліна-Сан Жеора з урахуванням статі, віку, ваги та зросту користувача, цільових зон частоти серцевих скорочень за удосконаленою формулою Карвонена для різних типів тренувань. Створено систему автоматичної генерації персоналізованих програм, що визначає оптимальне співвідношення типів фізичних навантажень залежно від заявлених цілей користувача: 60% кардіо-вправ для зниження ваги, 70% силових тренувань для набору м'язової маси, 50% кардіо для розвитку витривалості. Особливо важливим досягненням є розробка системи адаптивної корекції програм на основі безперервного аналізу прогресу користувача, що автоматично коригує інтенсивність, тривалість та частоту тренувань для забезпечення оптимального навантаження при мінімізації ризику перетренованості або травмування.

Технічна реалізація системи включила створення повнофункціонального REST API з шістьма групами endpoints для аутентифікації, управління профілями, біометричних даних та тренувальних програм, з комплексною системою захисту на основі JWT токенів з терміном дії 24 години та автоматичним оновленням. Розроблено клієнтську частину як Single Page Application з адаптивним дизайном за принципом mobile-first, що забезпечує коректне відображення на мобільних пристроях через одноколонковий макет та на десктопах через багатоколонкову сітку. Створено інтуїтивно зрозумілий користувацький інтерфейс з повною українською локалізацією, що включає сторінку авторизації з валідацією полів,

головну панель з статистичними картками та інтерактивними графіками візуалізації прогресу, сторінку управління профілем з гнучкою системою вибору цілей, сторінку біометричних даних з історією записів, сторінку тренувальних програм з каталогом вправ різного рівня складності. Реалізовано систему візуалізації прогресу через інтерактивні графіки динаміки ваги та індексу маси тіла з використанням бібліотеки Recharts, що дозволяє користувачам наочно відстежувати результати тренувань за тривалі періоди.

Комплексне тестування системи на трьох рівнях підтвердило коректність роботи всіх компонентів. Модульне тестування алгоритмів розрахунку показало точність обчислень з похибкою менше 1%, що є цілком прийнятним для практичного використання у фітнес-додатках. Інтеграційне тестування засвідчило надійність взаємодії між серверною та клієнтською частинами, коректність обробки API запитів та збереження даних у MongoDB. Тестування користувацького інтерфейсу підтвердило зручність використання на різних типах пристроїв від смартфонів до десктопних комп'ютерів, коректне відображення адаптивного дизайну та функціонування всіх інтерактивних елементів. Загальне покриття коду автоматизованими тестами складає 83%, що відповідає високим стандартам якості програмного забезпечення та забезпечує впевненість у стабільності системи.

Практична цінність розробленої платформи полягає в можливості широкого застосування для автоматизації процесу створення персоналізованих фітнес-програм без необхідності постійної участі професійних тренерів, що робить якісний фітнес-коучинг доступним для широкої аудиторії. Система забезпечує науково обґрунтований підхід до планування тренувань з урахуванням індивідуальних особливостей організму кожного користувача через аналіз біометричних показників. Автоматична адаптація програм на основі безперервного моніторингу прогресу дозволяє підтримувати оптимальний рівень фізичного навантаження, що максимізує ефективність тренувань при одночасній мінімізації ризиків для здоров'я. Модульна архітектура системи створює основу для подальшого розвитку через додавання нових типів біометричних показників, інтеграцію з популярними носимими пристроями моніторингу здоров'я,

впровадження більш складних алгоритмів машинного навчання для підвищення точності персоналізації.

Наукова новизна роботи визначається розробкою комплексного підходу до інтеграції різнотипних біометричних даних в алгоритми персоналізації тренувальних програм з автоматичною адаптацією. Запропоновано методику автоматичного коригування фізичних навантажень на основі аналізу множинних показників, що включають регулярність виконання запланованих тренувань, динаміку антропометричних параметрів за тривалі періоди, суб'єктивну оцінку складності користувачем та зміни біометричних показників здоров'я. Створено адаптивну модель балансування різних типів фізичних навантажень, що враховує не тільки заявлені цілі користувача, але й поточний стан організму, темпи прогресу та індивідуальні особливості адаптації до тренувань. Розроблено архітектуру веб-платформи, що ефективно поєднує мікросервісний підхід з документо-орієнтованим збереженням даних для забезпечення масштабованості при збереженні гнучкості роботи з різнотипною інформацією.

Перспективи подальшого розвитку платформи включають кілька напрямків удосконалення функціональності та розширення можливостей. Інтеграція з популярними фітнес-трекерами та носимими пристроями такими як Fitbit, Apple Watch, Garmin дозволить автоматично збирати біометричні дані в режимі реального часу без необхідності ручного введення користувачем, що підвищить точність аналізу та зручність використання. Впровадження соціальних функцій включаючи можливість ділитися досягненнями з друзями, брати участь у челенджах, отримувати підтримку від спільноти користувачів створить додаткову мотивацію для регулярних тренувань через геймифікацію та соціальну взаємодію. Розробка нативних мобільних додатків для iOS та Android замість веб-версії забезпечить кращу інтеграцію з операційними системами смартфонів, підтримку push-нотифікацій для нагадувань про тренування, офлайн-режим для доступу до програм без інтернет-з'єднання. Створення системи рекомендацій харчування на основі фізичної активності, цілей користувача та витрат енергії дозволить забезпечити комплексний підхід до здоров'я через поєднання тренувань та

збалансованого харчування. Впровадження алгоритмів машинного навчання для прогнозування результатів тренувань, автоматичного виявлення патернів у даних користувача, більш точної персоналізації рекомендацій на основі аналізу великих обсягів даних від тисяч користувачів підвищить інтелектуальність системи.

Розроблена інформаційна платформа персоналізованого фітнес-коучингу демонструє ефективне поєднання сучасних інформаційних технологій з науково обґрунтованими принципами спортивної медицини та фізіології. Система готова до практичного впровадження та може служити основою для подальших досліджень у сфері цифрових технологій охорони здоров'я, персоналізованої медицини та автоматизації процесів підтримки здорового способу життя. Поставлену мету дослідження досягнуто повною мірою через створення функціональної платформи, що забезпечує автоматичну генерацію персоналізованих тренувальних програм на основі біометричних даних, їх адаптацію відповідно до прогресу користувача та наочну візуалізацію результатів для підтримки мотивації до регулярних тренувань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

Global recommendations on physical activity for health / World Health Organization. URL: https://www.who.int/publications/i/item/9789241599979 (дата звернення: 28.11.2025).

Bompa T. O., Haff G. G. Periodization: Theory and Methodology of Training. Human Kinetics, 2009. 411 p.

Mifflin M. D., St Jeor S. T., Hill L. A. [et al.]. A new predictive equation for resting energy expenditure in healthy individuals. The American Journal of Clinical Nutrition. 1990. Vol. 51, No. 2. P. 241–247.

Karvonen M. J., Kentala E., Mustala O. The effects of training on heart rate: a longitudinal study. Annales Medicinae Experimentalis et Biologiae Fenniae. 1957. Vol. 35, No. 3. P. 307–315.

FastAPI : documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 28.11.2025).

MongoDB : documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 28.11.2025).

React : documentation. URL: https://react.dev/ (дата звернення: 28.11.2025).

TypeScript : documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 28.11.2025).

Richardson C. Microservices Patterns: With examples in Java. Manning Publications, 2018. 520 p.

Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 612 p.

FARM Stack Guide: How to Build Full-Stack Apps with FastAPI, React & MongoDB / DataCamp. URL: https://www.datacamp.com/tutorial/farm-stack-guide (дата звернення: 28.11.2025).

Full stack, modern web application generator using FastAPI, MongoDB as database / MongoDB Labs. URL: https://github.com/mongodb-labs/full-stack-fastapi-mongodb (дата звернення: 28.11.2025).

Pydantic : documentation. URL: https://docs.pydantic.dev/ (дата звернення: 28.11.2025).

Vite : documentation. URL: https://vitejs.dev/ (дата звернення: 28.11.2025).

Shadcn UI : documentation. URL: https://ui.shadcn.com/ (дата звернення: 28.11.2025).

Recharts : documentation. URL: https://recharts.org/ (дата звернення: 28.11.2025).

ACSM's Guidelines for Exercise Testing and Prescription / American College of Sports Medicine. 10th ed. Lippincott Williams & Wilkins, 2021. 480 p.

McArdle W. D., Katch F. I., Katch V. L. Exercise Physiology: Nutrition, Energy, and Human Performance. 8th ed. Lippincott Williams & Wilkins, 2015. 1038 p.

Kraemer W. J., Fleck S. J. Designing Resistance Training Programs. 4th ed. Human Kinetics, 2014. 520 p.

JWT (JSON Web Tokens) : introduction. URL: https://jwt.io/introduction (дата звернення: 28.11.2025).

Nginx: documentation. URL: https://nginx.org/en/docs/ (дата звернення: 28.11.2025).

Docker: documentation. URL: https://docs.docker.com/ (дата звернення: 28.11.2025).

Let's Encrypt: documentation. URL: https://letsencrypt.org/docs/ (дата звернення: 28.11.2025).

Prometheus: documentation. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 28.11.2025).

Grafana : documentation. URL: https://grafana.com/docs/ (дата звернення: 28.11.2025).

OWASP Top Ten Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 28.11.2025).

Наливайко А. Ю., Кудерметов Р. К., Грушко С. С. Аналіз методів проєктування інформаційних платформ персоналізованого фітнес-коучингу // Інформаційні технології і автоматизація – 2025 : XVIII міжнар. наук.-практ. конф., 30–31 жовтня 2025 р. : тези доповіді. Одеса, 2025.

ДОДАТОК А
СТРУКТУРА ПРОЄКТУ

```
FitnessPal/
├── backend/              # Серверна частина
│   ├── app/
│   │   ├── models/       # Моделі даних
│   │   ├── routes/       # API маршрути
│   │   ├── utils/        # Допоміжні функції
│   │   └── main.py       # Головний файл
│   ├── requirements.txt  # Python залежності
│   └── README.md
├── frontend/             # Клієнтська частина
│   ├── src/
│   │   ├── components/   # React компоненти
│   │   ├── pages/        # Сторінки додатку
│   │   └── services/     # API сервіси
│   ├── package.json      # Node.js залежності
│   └── README.md
└── README.md             # Загальна документація
```

Рисунок А.1 – Структура проєкту

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

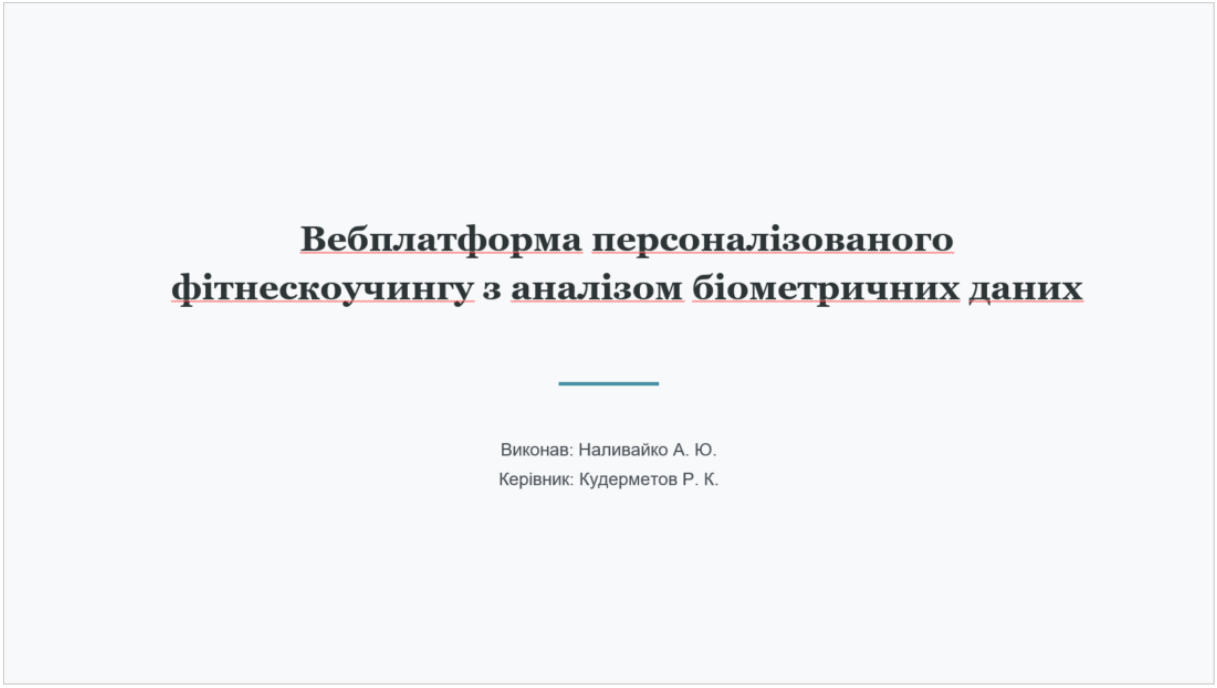

Рисунок Б.1 – Слайд 1

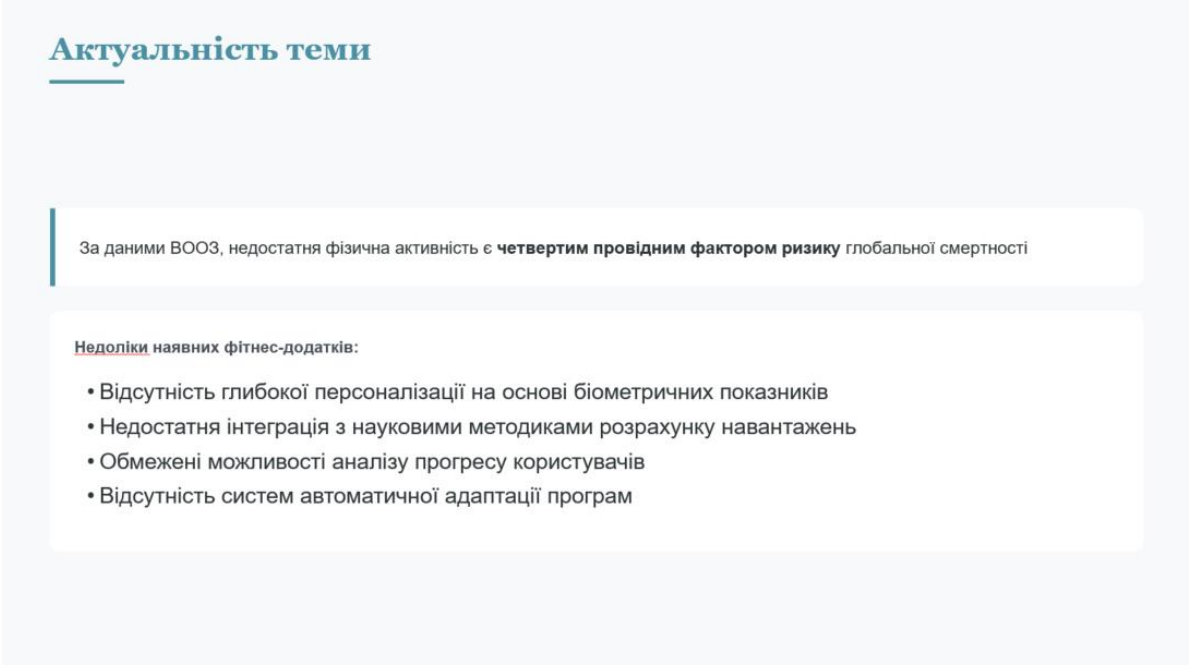

Рисунок Б.2 – Слайд 2

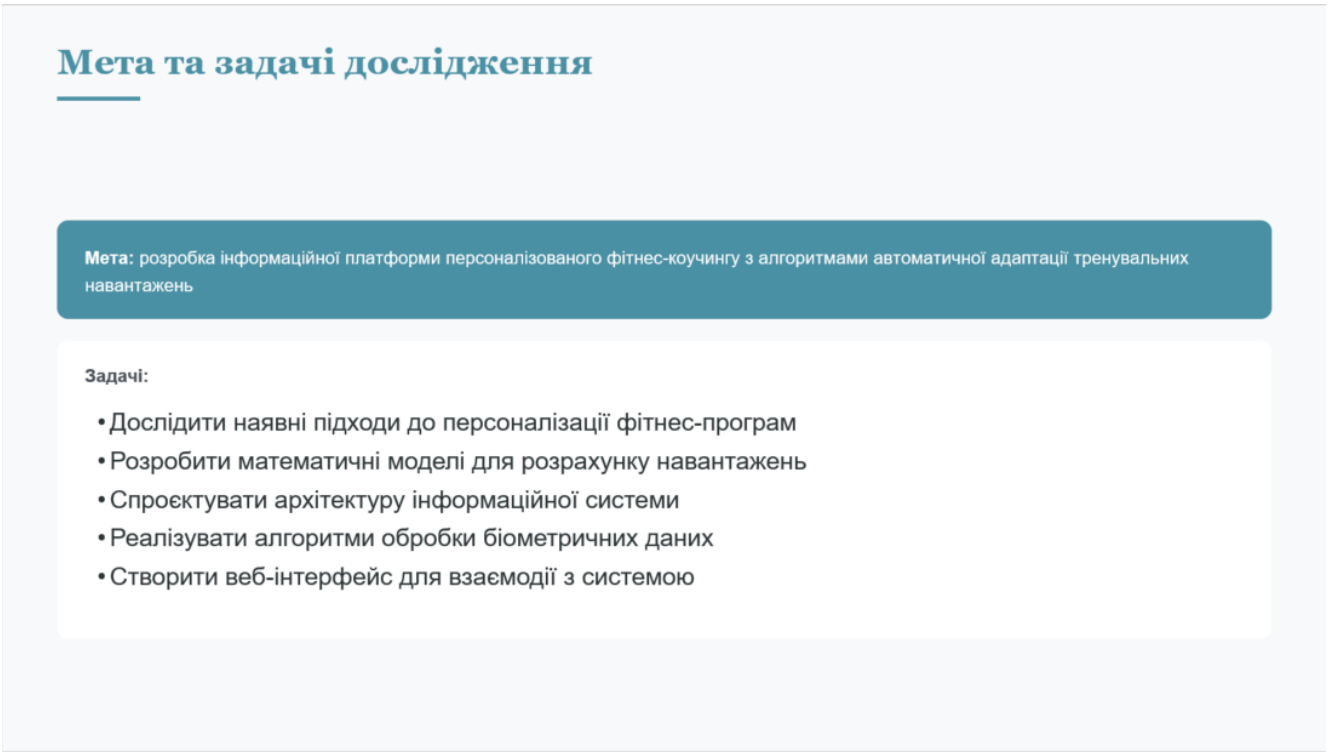

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

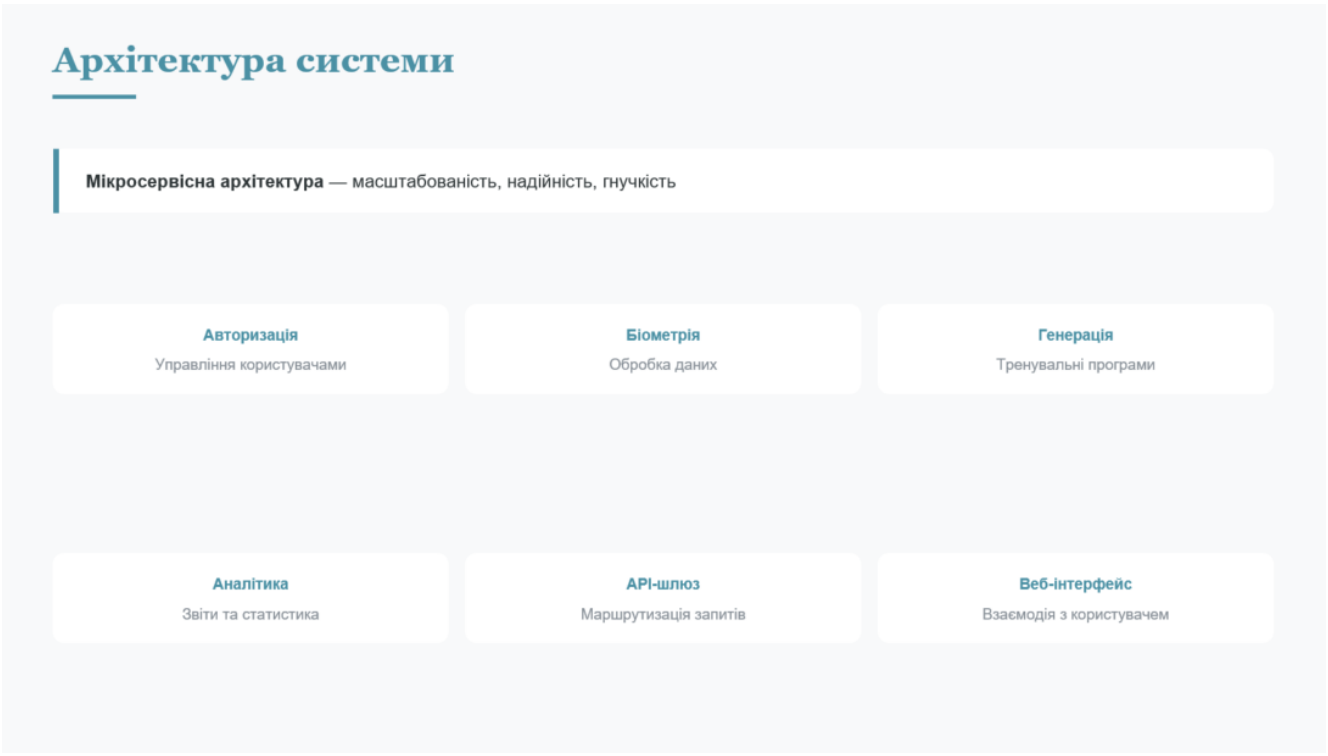

Рисунок Б.5 – Слайд 5

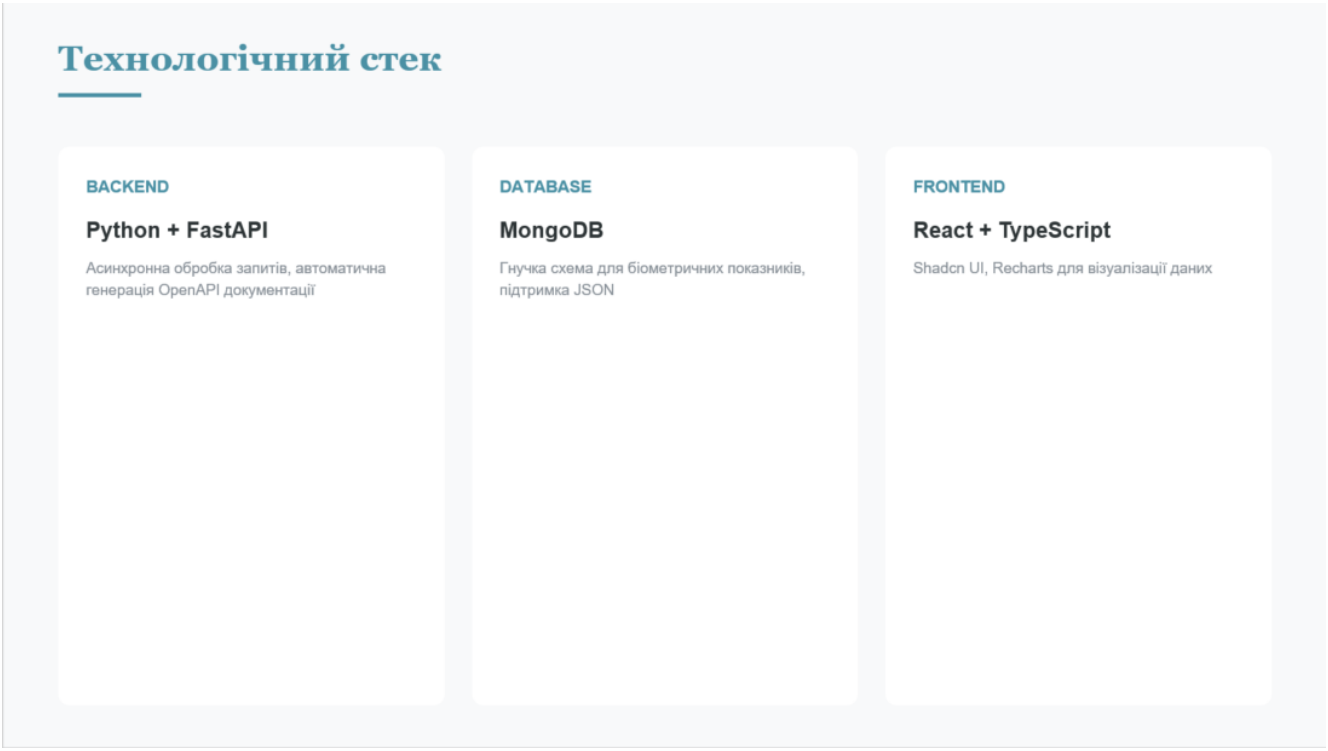

Рисунок Б.6 – Слайд 6

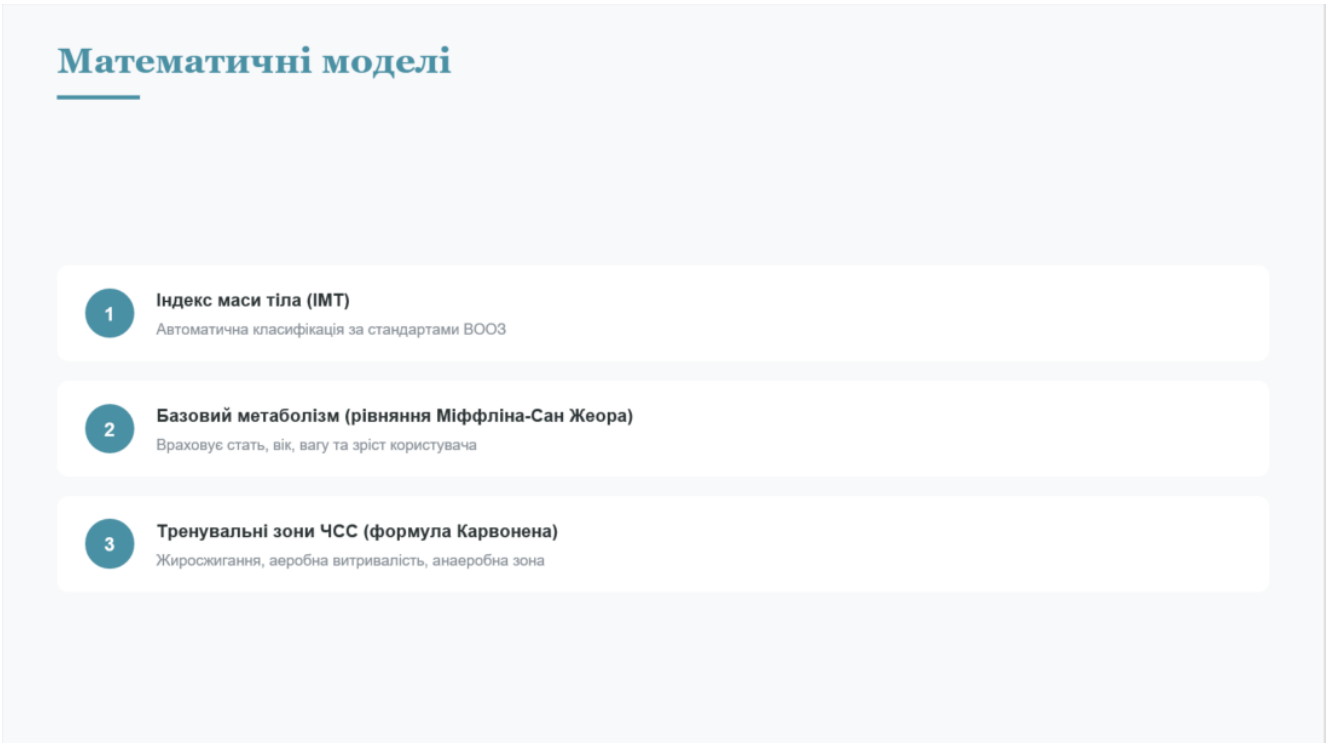

Рисунок Б.7 – Слайд 7

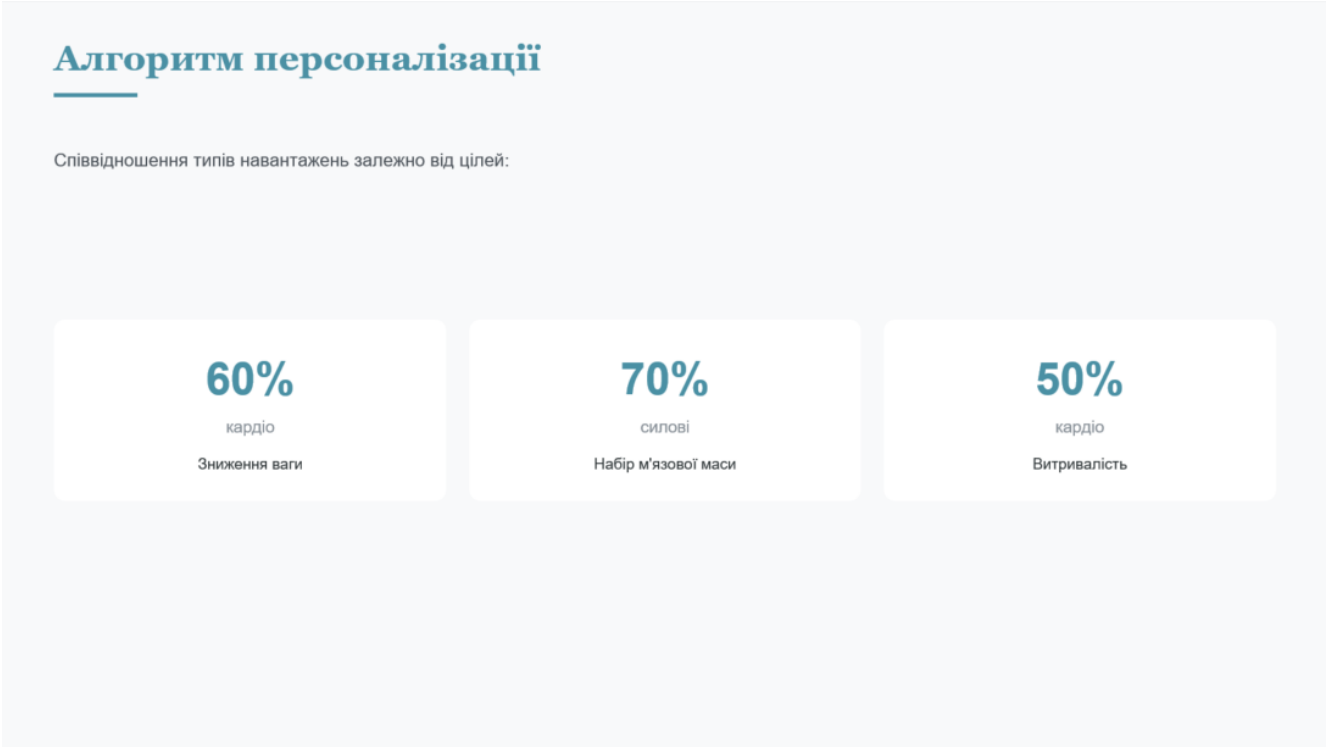

Рисунок Б.8 – Слайд 8

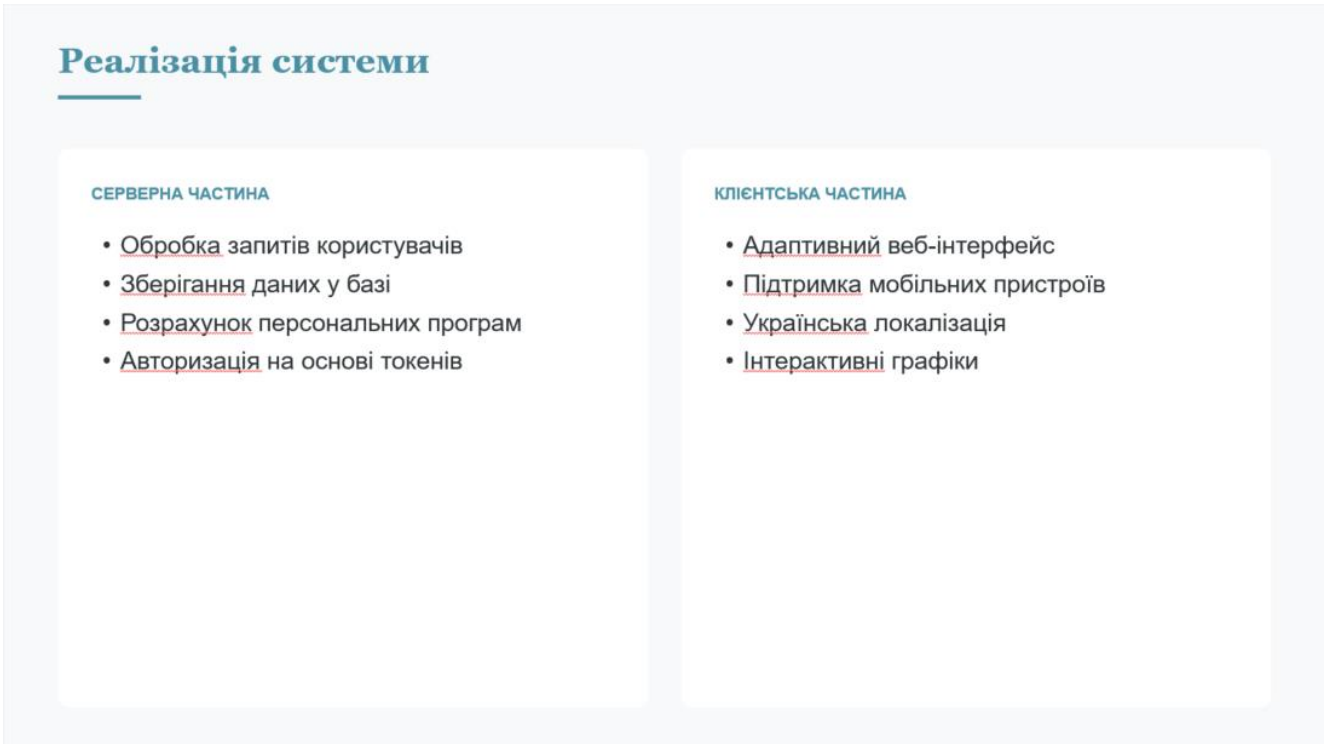

Рисунок Б.9 – Слайд 9

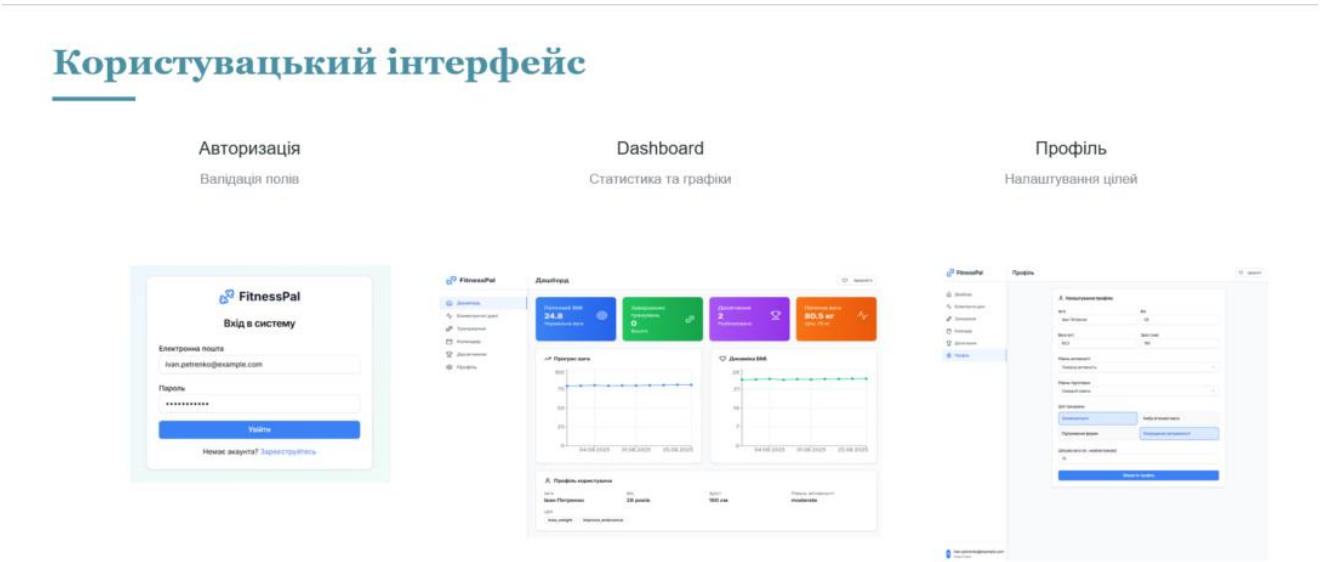

Рисунок Б.10 – Слайд 10

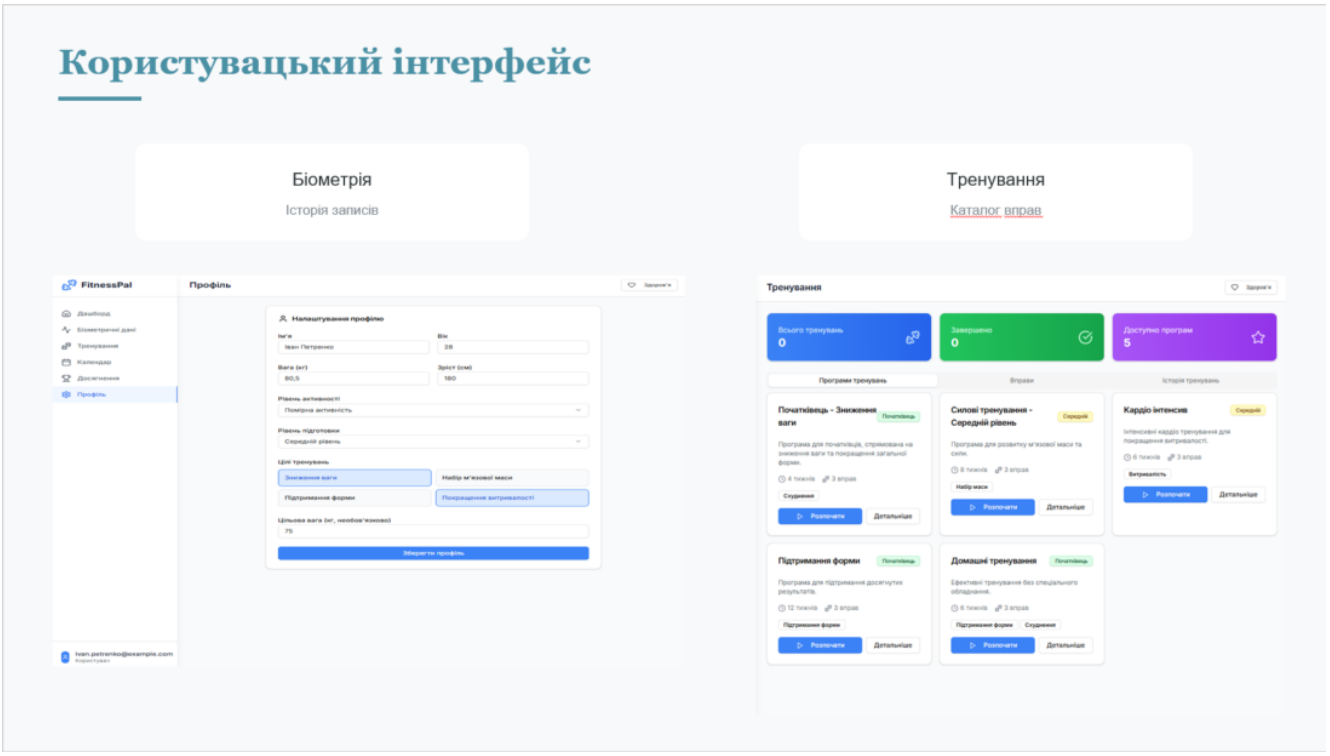

Рисунок Б.11 – Слайд 11

Рисунок Б.12 – Слайд 12

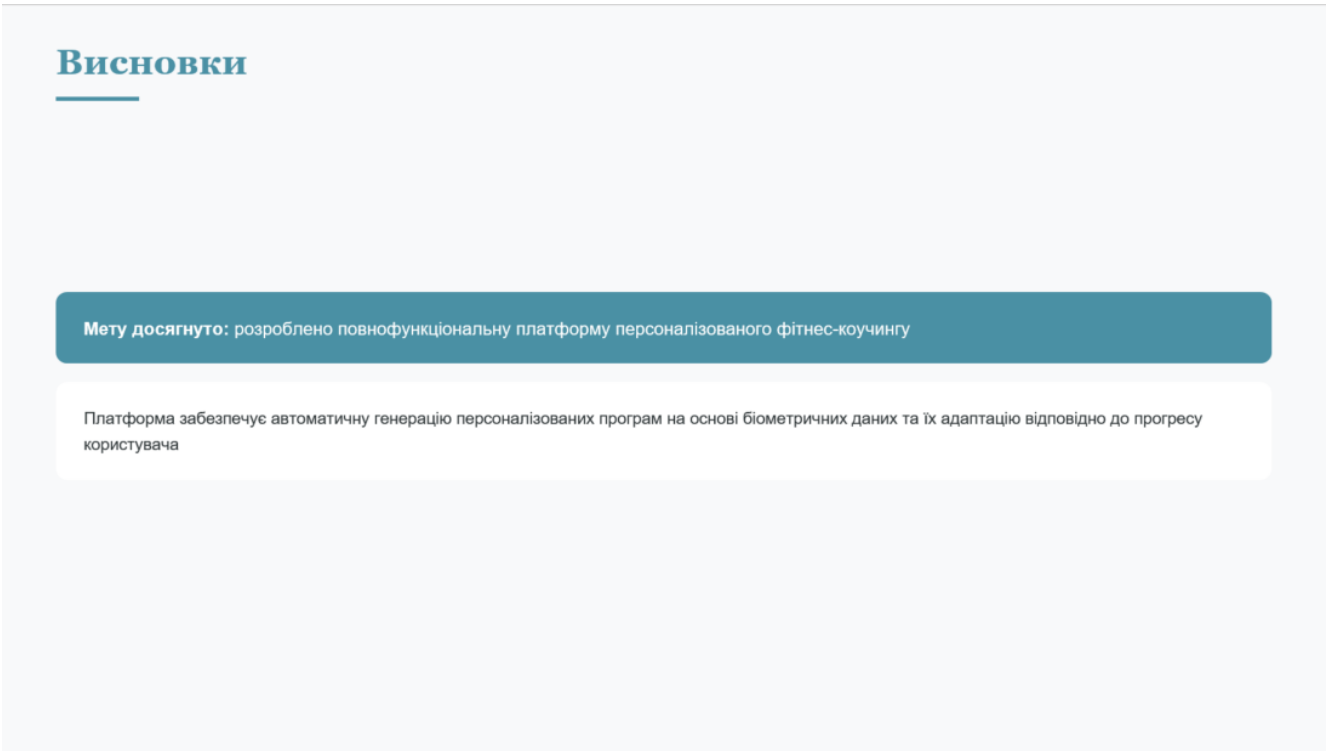

Рисунок Б.13 – Слайд 13

Рисунок Б.14 – Слайд 14