Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

**КОМПЛЕКСНА МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

# ВЕБ-ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТРЕНУВАНЬ ТА ДІЄТИ З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ. ЧАСТИНА 1. «РОЗПОДІЛЕНА АРХІТЕКТУРА ТА UI/UX ДИЗАЙН».

Виконав: студент 2 курсу, групи 1КІ-22м
спеціальності 123 — Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Никитюк В.О.
(прізвище та ініціали)
Керівник к.т.н., доцент каф. ОТ
Городецька О.С.
(прізвище та ініціали)
« 07 » 12 2023 р.
Опонент к.т.н., доцент каф. ПЗ
Майданюк В.П.
(прізвище та ініціали)
« 08 » 12 2023 р.

Допущено до захисту
Зав. кафедри д.т.н., проф. Азаров О.Д.
« 08 » 12 2023 р.

ВНТУ 2023

**ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
(шифр і назва)
Освітній рівень — магістр
Спеціальність — 123 «Комп'ютерна інженерія»
(шифр і назва)
Освітньо-професійна програма — Комп'ютерна інженерія
(Назва освітньо — професійної програми)

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ОТ, д.т.н, проф.

Азаров О.Д.

«26» 09 2023 р.

**З А В Д А Н Н Я**

**НА КОМПЛЕКСНУ МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ**

студенту Никитюку В'ячеславу Олександровичу

(прізвище, ім'я, по батькові)

1 Тема роботи Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» керівник роботи Городецька Оксана Степанівна, к.т.н., доцент, затверджені наказом вищого навчального закладу від **«26» 09 2023** року **№ 247**

2 Строк подання студентом роботи **9 грудня 2023** року.

3 Вихідні дані до роботи: Критерії вибору архітектури, текстові дані для створення UI/UX дизайну.

4 Зміст текстової частини: аналіз сучасних підходів для вибору архітектури веб-застосунку для організації тренувань та дієти, теоретичні основи проектування архітектури веб-застосунку та вибір архітектурного стилю розробки, вдосконалений метод клієнт-серверної взаємодії та проектування архітектури веб-застосунку, розробка UI/UX дизайну для веб-застосунку організації тренувань та дієти.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): дизайн сторінок авторизації, дизайн сторінки створення дієти, UML-діаграма розгортання веб-застосунку, загальна структура веб-застосунку, структурна схема архітектури веб-застосунку.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада Консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-4 | Городецька О.С., к.т.н., доцент каф. ОТ | | |
| 5 | Небава М.І. | | |

7 Дата видачі завдання **19.09.2023**.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

| № з/п | Назва етапів МКР | Термін виконання | | Прим |
|---|---|---|---|---|
| | | початок | закінчення | |
| 1 | Аналіз завдання | 26.09.23 | 27.09.23 | вик. |
| 2 | Аналіз предметної області | 28.09.23 | 01.10.23 | вик. |
| 3 | Аналіз критеріїв до вибору архітектури | 02.10.23 | 05.10.23 | вик. |
| 4 | Аналіз типів високорівневої архітектури | 06.10.23 | 12.10.23 | вик. |
| 5 | Обґрунтування вибору архітектурного стилю. Аналіз архітектурних шаблонів | 13.10.23 | 28.10.23 | вик. |
| 6 | Розробка загальної структури застосунку | 29.10.23 | 05.11.23 | вик. |
| 7 | Огляд технологій для розробки клієнтської та серверної частин | 06.11.23 | 11.11.23 | вик. |
| 8 | Вдосконалення методу клієнт-серверної взаємодії | 12.11.23 | 20.11.23 | вик. |
| 9 | Розробка інтерфейсу для формування запитів | 21.11.23 | 23.11.23 | вик. |
| 10 | Розробка UI/UX дизайну веб-застосунку | 24.11.23 | 28.11.23 | вик. |
| 11 | Оформлення пояснювальної записки та ілюстративного матеріалу | 29.11.23 | 05.12.23 | вик. |
| 12 | Перевірка якості виконання комплексної магістерської кваліфікаційної роботи та усунення недоліків | 05.12.23 | 08.12.23 | вик. |

Студент _____ Никитюк В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Городецька О.С.
(підпис) (прізвище та ініціали)

**АНОТАЦІЯ**

УДК 004.4

Никитюк В. О. Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн». Комплексна магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2023 — 103 с. На укр. мові. Бібліогр.: 22 назви; рис.: 20; табл. 13.

В комплексній магістерській кваліфікаційній роботі проаналізовано сучасні підходи для вибору архітектури веб-застосунку для організації тренувань та дієти, описано порівняльну характеристику аналогів та обґрунтовано вибір архітектурного стилю при розробці веб-застосунку. Розроблено загальну структуру веб-застосунку та вдосконалено метод клієнт-серверної взаємодії шляхом використання проміжного серверу. Розроблено UI/UX дизайн веб-застосунку, набір інтерфейсів користувача UI-KIT та інтерфейс для формування запитів до штучного інтелекту.

Ключові слова: веб-застосунок, дієта, тренування, розподілена архітектура, архітектурні стилі розробки веб-застосунків, клієнт-серверна взаємодія, Figma, веб-дизайн, методології розробки веб-застосунків.

**ABSTRACT**

УДК 004.4

Nykytiuk V.O. Web application for organizing training and diet with the integration of artificial intelligence. Part 1. "Distributed architecture and UI/UX design". Comprehensive master's qualification thesis on specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2023 — 103 p. In Ukrainian speech Bibliography: 22 titles; pic.: 20; tables 13.

In a comprehensive master's qualification work, modern approaches to choosing the architecture of a web application for training and diet organization are analyzed, the comparative characteristics of analogues are described, and the choice of architectural style in the development of a web application is substantiated. The general structure of the web application was developed and the method of client-server interaction by using an intermediate server was improved. The UI/UX design of the web application, a set of UI-KIT user interfaces and an interface for generating requests to artificial intelligence were developed.

Keywords: web application, diet, training, distributed architecture, architectural styles of web application development, client-server interaction, Figma, web design, web application development methodologies.

## ЗМІСТ

# ВСТУП

Сучасний світ веб-сайтів розв'язує безліч завдань для користувачів і бізнесу. Він включає навчальні платформи, надає можливість пошуку музики та здійснення зручних покупок різних товарів, усе це в рамках швидкого та зручного інтерфейсу.

Сучасний веб-застосунок для персональних тренувань та дієти з інтеграцією штучного інтелекту пропонує подібний підхід. Кожен користувач має доступ до веб-сайту, де він знаходить детальну інформацію про програми тренувань, дієт, рекомендації щодо здорового способу життя та багато іншої корисної інформації. Цей застосунок дозволяє людям вибирати та планувати свої тренувальні та харчові режими на основі власних потреб та цілей.

Завдяки інтеграції штучного інтелекту, застосунок може аналізувати та пропонувати індивідуальні підходи до тренувань та харчування, надаючи користувачам персональні рекомендації. Цей веб-застосунок особливо актуальний в умовах популярності здорового способу життя та фітнес-тренувань, коли доступ до персональної програми може бути вирішальним для досягнення поставлених цілей.

UI/UX дизайн та розподілена архітектура є важливими аспектами розробки веб-сайту. На перший погляд, красивий дизайн приваблює користувачів і забезпечує комфортне використання інтерфейсу (UI/UX).

Розподілена архітектура є концепцією розробки клієнт-серверного додатку, яка передбачає розташування різних компонентів застосунку на різних фізичних або логічних вузлах (комп'ютерах, серверах), які співпрацюють між собою через мережу (протокол HTTP). Ця архітектура розроблена з метою покращення масштабованості, надійності, продуктивності та ефективності додатків. В результаті користувачі отримують зручний та швидкий веб-застосунок для досягнення поставлених цілей здоров’я та тренування.

**Метою дослідження** є підвищення швидкодії обробки даних шляхом використання проміжного серверу в архітектурі веб-застосунку для організації тренувань та дієти з використанням штучного інтелекту.

**Задачі дослідження** комплексної магістерської кваліфікаційної роботи:

— проаналізувати сучасні підходи для вибору архітектури веб-застосунку;

— обґрунтувати вибір архітектурного стилю розробки;

— розробити загальну структуру веб-застосунку;

— вдосконалити метод клієнт-серверної взаємодії;

— проаналізувати трафік та швидкість завантаження сторінки;

— розробити UI/UX дизайн веб-застосунку.

**Об’єкт дослідження** — процес проектування розподіленої архітектури та розробки дизайну веб-застосунку для організації персональних тренувань та дієти з інтеграцією штучного інтелекту.

**Предметом дослідження** є методи клієнт-серверної взаємодії та сучасні інструменти розробки UI/UX дизайну при створенні веб-застосунку для організації персональних тренувань та дієти.

**Наукова новизна отриманих результатів** комплексної магістерської кваліфікаційної роботи полягає у вдосконаленні методу клієнт-серверної взаємодії шляхом використання проміжного серверу з можливістю виконання запитів до бази даних безпосередньо з клієнтського додатку, що дало можливість збільшити швидкодію обробки даних.

**Практичне значення одержаних результатів** комплексної магістерської кваліфікаційної роботи: розроблений веб-застосунок можна використовувати як персонального дієтолога, який опише план харчування та тренування згідно персональних даних.

**Апробація результатів** комплексної магістерської кваліфікаційної роботи — зроблено доповідь на LI науково-технічній конференції підрозділів Вінницького національного технічного університету та на Міжнародній

науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2024).

Матеріали роботи доповідались та опубліковувались [1, 2]:

В. О. Никитюк, О. В. Войцеховська / Аналіз методів клієнт-серверної взаємодії при розробці панелі адміністратора за допомогою реактивного фреймворка React // Тези доповіді. Матеріали LI наукової-технічної конференції підрозділів Вінницького національного технічного університету. Вінниця 2022 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15104/12730.

Городецька О. С., Войцеховська О. В., Никитюк В. О. / Сучасні підходи при розробці архітектури розподілених систем // Матеріали Міжнародної науково-практичної інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2024). 2024 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/19088/15838.

# 1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДЛЯ ВИБОРУ АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ТРЕНУВАНЬ ТА ДІЄТИ

При проектуванні розподіленої архітектури веб-застосунку для організації персональних тренувань та дієти з інтеграцією штучного інтелекту необхідно провести аналіз популярних і ефективних технологій для взаємодії між клієнтом і сервером.

Основним завданням проектування архітектури веб-сайту являються найбільш ефективно підібрані технології під конкретну ціль, що забезпечить швидкість розробки сайту та необхідний кінцевий результат.

UI/UX дизайн — це комплексна сфера, що охоплює всі аспекти програмного інтерфейсу. Серед них — кнопки, тексти, зображення, текстові поля та інші взаємодійні елементи, які користувач бачить і використовує. Крім цього, важливими є макет екранів, анімація та переходи між ними. Усі ці візуальні та інтерактивні елементи розробляються на етапі UI-дизайну з урахуванням візуальної привабливості та відповідності концепції додатку.

З іншого боку, UX дизайн концентрується на взаємодії користувача з додатком або сервісом. Він має створити інтуїтивно зрозумілу та логічну навігацію, щоб забезпечити комфортну взаємодію користувача з інтерфейсом. Ефективність взаємодії з елементами, створеними дизайнерами, визначається якістю UX дизайну. Тому обидві складові, UI та UX дизайн, є важливими для створення успішного та зручного програмного інтерфейсу.

## 1.1 Аналіз предметної області

Для розробки ефективного веб-застосунку для персональних тренувань та дієти необхідно проаналізувати ключові аспекти.

Перш за все, необхідно визначити основних користувачів веб-застосунку. Це можуть бути тренери, спортсмени, дієтологи, а також звичайні люди, які прагнуть покращити своє здоров’я та фізичну форму.

Другим кроком є визначення основних функцій веб-застосунку. Це може включати створення індивідуальних тренувальних програм, рекомендації щодо дієти та багато іншого.

Також важливо визначити, які технології будуть використовуватися для розробки веб-застосунку. Це може включати вибір мови програмування, фреймворків, бази даних, а також розгляд питань безпеки та приватності.

Нарешті, варто провести аналіз ринку, щоб зрозуміти, які подібні веб-застосунки вже існують, які їхні сильні та слабкі сторони, а також визначити унікальну цінність, яку пропонує ваш веб-застосунок.

Особливу увагу слід приділити створенню дизайну користувацького інтерфейсу (UI/UX). Для цього можна використовувати сучасні бібліотеки та фреймворки, наприклад, React або Qwik.

Важливо враховувати потреби користувачів у програмному інтерфейсі, зосереджуючись на оптимальному використанні елементів взаємодії, таких як кнопки, текстові поля та зображення.

Дизайн користувацького досвіду має забезпечити логічну та інтуїтивно зрозумілу навігацію. Застосування анімації та плавних переходів може підвищити привабливість та комфорт взаємодії з додатком.

Також важливо, щоб дизайн користувацького інтерфейсу та дизайн користувацького досвіду відповідали концепції застосунку та відображали його цільовий функціонал, забезпечуючи послідовність та зрозумілість для користувача.

У контексті розробки веб-застосунку з інтеграцією штучного інтелекту, дизайн повинен враховувати кілька ключових аспектів:

— адаптивність та респонсивність, дизайн має бути адаптивним до різних пристроїв та розмірів екранів для забезпечення оптимального вигляду та функціональності;

— забезпечення безпеки даних, врахування важливості безпеки в обробці та передачі конфіденційних даних користувачів;

— підтримка інтерактивності та реактивності, забезпечення швидкої та ефективної взаємодії користувача з додатком;

— візуальна привабливість, створення естетичного та привабливого вигляду додатку;

— легкість навігації, забезпечення логічної та інтуїтивно зрозумілої навігації в середині додатку;

— підтримка доступності, забезпечення доступності для користувачів з обмеженими можливостями.

Ефективний дизайн в веб-застосунку організації дієти та тренувань має враховувати технічні, безпекові та ергономічні вимоги, забезпечуючи не лише естетичний вигляд, але й оптимальну взаємодію з користувачем.

## 1.2 Порівняльна характеристика аналогів

Перед безпосередньою розробкою дизайну веб-застосунку були розглянуті і проаналізовані сайти подібної тематики. А саме це веб-сайти калькуляторів калорій та подібні.

На рисунку 1.1 подана головна сторінка веб-інтерфейсу калькулятора калорій [3].

Даний сайт має непогану задумку, включає сторінку з рецептами різних страв, які можна зручно відфільтрувати і знайти саме те, що потробно. Також є сторінка із активностями, які можуть бути впродовж дня, на якій вписані витрати калорій на кожну таку активність. І подібна сторінка з їжею та калоріями. Крім цього є можливість розрахувавши кількість необхідних калорій, потрібних людині для схуднення чи набору ваги, або ж утримування в нормі. Мінусом даного веб-додатку є те, що людина без хоча б базових знань в дієтології не зможе сама собі скласти дієту, і їй знадобиться купувати курс дієти або ж шукати в іншому місці. Щодо дизайну, то в цілому він зручний, але UX рішення виглядають застарілими на базі сучасних веб-додатків.

Проаналізувавши веб-інтерфейс додатку з статтями про харчування (рис 1.2), можна зробити висновок що дизайн цього застосунку теж являється застарілим [4]. Даний сайт є корисним доповненням, якщо користувача цікавить якась інформація, описана в статтях, але знову ж таки ми стикаємося з тою ж проблемою, а саме відсутності можливості автоматичного підбору дієти під конкретні потреби користувача.


Рисунок 1.1 — Вигляд головної сторінки веб-додатку з розрахунком калорій

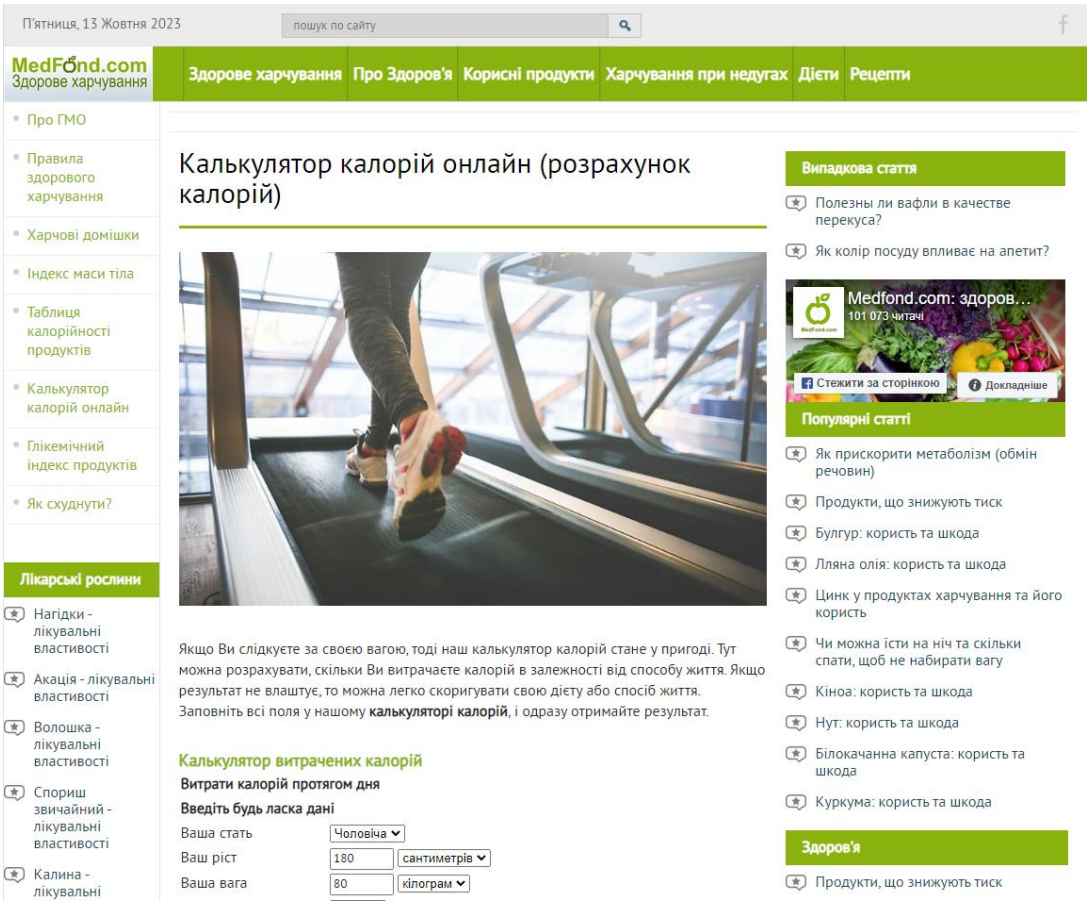


Рисунок 1.2 — Вигляд головної сторінки веб-додатку з статтями про харчування

Отже, проаналізувавши аналогічні ресурси, можна зробити висновок про необхідність застосування правильно підібраної колірної палітри, розмірів шрифтів в дизайні та на сайті, а також відповідати сучасним нормам UI/UX дизайну. Веб-інтерфейс повинен бути інтуїтивно-зрозумілим і відповідати сучасним вимогам, а основні елементи веб-сайту повинні розташовуватись в швидкому доступі.

## 1.3 Аналіз сучасних глобальних сховищ

Технологія глобального сховища — це дуже зручний підхід до зберігання та управління станом усього застосунку в одному місці.

Зручність такого зберігання можна реалізувати, наприклад, під час виконання завдання перемикання теми сайту (світлої або темної). Зазвичай для

виконання цього завдання можна використовувати лише props компонента. Однак реалізація цього методу потребує багато часу на розробку і є не динамічним рішенням. Під час створення нових компонентів ті самі props доводиться вручну реєструвати для кожного нового компонента.

Щоб заощадити час розробників, зменшити обсяг коду та покращити структуру проекту, існують бібліотеки, що виконують схожі завдання, наприклад, MobX, Redux та Qwik context, але принцип їхньої роботи значно відрізняється.

Глобальне сховище Redux — це контейнер для керування станом застосунку. Redux не прив'язаний безпосередньо до фреймворку чи бібліотеки і може використовуватися з різними JS-бібліотеками та фреймворками [5].

Ключові особливості Redux:

— сховище (store) для зберігання стану застосунку;

— дії (actions), тобто, це певний набір інформації, що надсилається із застосунку до сховища й вказує, що саме потрібно зробити. Щоб передати цю інформацію у сховище, використовується метод dispatch();

— творці дій (actions creators), тобто функції, що створюють дії;

— reducer, тобто функція (або функції), яка отримує дію і відповідним чином змінює стан сховища.

Загальну схему взаємодії елементів архітектури Redux представлено на рисунку 1.3.

Певна дія відправляється із представлення View (тобто компонента Qwik) через функцію dispatch, цю дію отримує функція reducer, яка оновлює стан сховища відповідно до зазначеної інформації. Після зміни стану сховища компонент Qwik застосовує уже оновлений стан. Таким чином, глобальний стан змінюється.

У застосунку на основі Flux є три основні складові: диспетчер (dispatcher), сховище даних (store) та представлення (view), які є стандартними компонентами Redux [6].

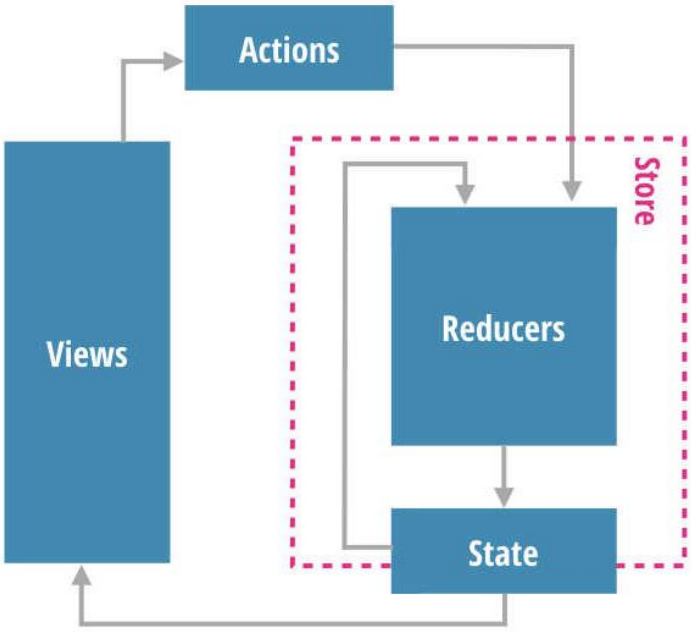


Рисунок 1.3 — Загальна схема взаємодії елементів архітектури Redux

Диспетчер — це центральна ланка схеми, що керує потоком даних у Flux. Диспетчер реєструє сховища та їхні зворотні виклики (callback). Коли диспетчер отримує дію ззовні, він використовує зворотні виклики сховища, щоб повідомити сховище, що дія відбулася.

Сховище містить стан і логіку застосунку, схожі на модель у патерні MVC, але керує групами об'єктів, а не окремими об'єктами. Кожне окреме сховище керує певною областю або доменом програми.

Кожне сховище реєструється в диспетчері разом з його зворотними викликами. Коли диспетчер отримує дію, він виконує зворотній виклик із цією дією як параметром. Залежно від типу дії, в сховищі викликається певний метод для оновлення стану. Коли сховище оновлюється, генерується подія, що вказує на те, що сховище було оновлено. За допомогою цієї події представлення дізнається, що сховище було оновлено, і оновлює свій стан.

Представлення формують візуальну частину застосунку. Особливий тип представлення, controller-view, є компонентом верхнього рівня, який містить всі інші компоненти. Controller-view прослуховує події зі сховища.

Отримавши подію, контролер передає дані, отримані зі сховища, іншим компонентам.

Коли controller-view отримує подію зі сховища, воно спочатку запитує всі необхідні дані зі сховища. Потім він викликає методи setState() і forceUpdate() і змушує компонент виконати метод render(). Це призводить до виклику методу render(), який оновлює дочірній компонент.

Дія — це функція, що містить дані, які передаються диспетчеру. Дії викликаються обробниками подій компонента. Дії також можуть бути ініційовані із зовнішнього джерела, наприклад, сервера. Сховище отримує дію через диспетчер і реагує на неї відповідним чином.

Глобальне сховище MobX — це бібліотека, яка робить управління станом простим і масштабованим завдяки прозорому використанню функціонального реактивного програмування та патерну Observer [7].

Паттерн Observer — це поведінковий патерн проектування, який створює механізм підписки, що дозволяє одному об'єкту відстежувати та реагувати на події, що відбуваються в інших об'єктах [8].

Мета цього патерну — визначити залежності між об'єктами типу "один-до-багатьох" так, щоб при зміні стану одного об'єкта всі об'єкти, які від нього залежать, отримували повідомлення та автоматично оновлювалися.

Цей патерн має наступні переваги:

— підтримує принцип вільної комунікації між взаємодіючими об’єктами;

— дозволяє ефективно передавати дані іншим об'єктам без модифікації класів Subject або Observer;

— спостерігачі можуть бути додані або видалені в будь-який час.

Схематично реалізацію патерна можна представити на рисунку 1.4.

При реалізації шаблону «Observer» зазвичай використовуються наступні класи:

— Subject, інтерфейс, який визначає методи для додавання, видалення та сповіщення спостерігачів;

— Observer, інтерфейс, через який об’єкт спостереження сповіщає спостерігачів;

— ConcreteSubject, конкретний клас, що реалізує інтерфейс Subject;

ConcreteObserver, конкретний клас, що реалізує інтерфейс Observer.

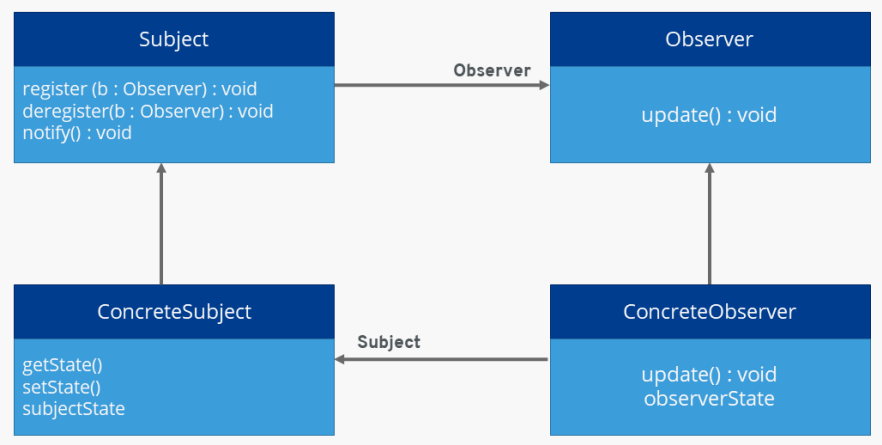


Рисунок 1.4 — Структура патерна Observer

Глобальне сховище MobX автоматично підписує компоненти на зміни, які використовуються під час рендерингу. Це означає, що компонент автоматично перемальовується лише тоді, коли змінюються відповідні спостережувані значення. Якщо ж відповідних змінних немає, компонент не перемальовується.

Практично це означає, що застосунки, які використовують сховище MobX, оптимізовані відразу після встановлення і простого налаштування. Їм, як правило, не потрібно додаткового коду для запобігання надмірному рендерингу. Механізм підписки в MobX є точнішим і ефективнішим у порівнянні з Redux, оскільки дані необхідно явно оголошувати або попередньо обчислювати.

Порівняльну характеристику глобальних сховищ наведено в таблиці 1.1.

Таблиця 1.1 — Порівняльна таблиця популярних глобальних сховищ

| Бібліотека | Кількість сховищ | Застосування | Масштабованість | Продуктивність |
|---|---|---|---|---|
| MobX | Можливість створення декілька глобальних сховищ | Для невеликих і простих додатків | Порівняно менш масштабований | Швидка зміна великої кількості даних |
| Redux | Можливість створення одного великого сховища | Для великих та комплексних сайтів | Створений для масштабованості в великих проектах | Можливі затримки при обробці великої кількості даних |
| Qwik контекст | Можливість створення необмеженої кількості | Для невеликих сайтів | Легко та швидко масштабується, але в рамках невеликих-середніх проектів | Реактивні зміни при зміні великої кількості даних |

Використання відразу декількох глобальних сховищ в MobX дозволяє логічно їх розділити, що в результаті забезпечує краще розуміння коду. Але власний контекст Qwik вирішує всі поставлені задачі в рамках даної роботи, тому в цьому випадку являється найбільш зручним рішенням.

Отже, контекст Qwik найзручніший в використанні і не потребує додаткових налаштувань, тому що відразу з пакетом Qwik його можна використовувати в компонентах і не потрібно завантажувати окремі бібліотеки, що дозволяє досягти кращої швидкодії додатку. Тому при проектуванні веб-застосунку організації дієти буде використовуватись контекст Qwik.

## 1.4 Критерії вибору архітектури

У процесі вибору архітектури для веб-застосунку, який орієнтований на управління персональними тренуваннями та дієтою з використанням штучного інтелекту, важливо враховувати ряд критеріїв. Ці критерії визначатимуть ефективність системи, забезпечуючи оптимальне поєднання функціональності штучного інтелекту та веб-застосунку. Обрана архітектура повинна відповідати конкретним потребам проекту, включаючи функціональність, швидкодію та вимоги до безпеки.

Питання масштабованості є важливим у веб-розробці. Необхідно розглядати можливості архітектури, які здатні ефективно працювати під збільшеним навантаженням та масштабуватись відповідно до зростаючого обсягу даних і користувачів.

Важливим критерієм кожного веб-застосунку є продуктивність, зважаючи на яку потрібно вибрати архітектуру, яка забезпечить швидкість відгуку системи на запити користувачів, зменшить час завантаження сторінок та забезпечить ефективну роботу веб-застосунку.

При виборі архітектури слід врахувати також аспекти безпеки. Це включає в себе механізми захисту даних користувачів, відповідність стандартам безпеки та заходи для запобігання можливим атакам.

Гнучкість та розширюваність архітектури грають важливу роль. Оптимальний вибір — це той, що легко дозволяє вносити зміни та розширювати функціонал без великих труднощів.

Важливо, щоб обрана архітектура відповідала сучасним стандартам розробки та забезпечувала можливість інтеграції з іншими технологіями, крім вибраних. Витрати на розробку та підтримку є також значущими. Архітектура має бути доступною для команди розробників та відповідати бюджетним обмеженням.

Не менш важливою є популярність вибраних для розробки технологій, так як чим популярніша технологія, тим краще вона документована та має

більшу кількість вирішених проблем і багів. Активна спільнота користувачів та підтримка розробників може стати важливим фактором для отримання доступу до різноманітних ресурсів, документації та вирішення проблем.

Тому важливо розглядати архітектуру, яка здатна відповісти майбутнім потребам та трендам в галузі веб-розробки для забезпечення актуальності застосунку з плином часу.

Описані критерії ляжуть в основу проектування архітектури, забезпечивши при цьому оптимальний вибір технологій, підходу до розробки та розроблений веб-застосунок в закладений термін.

# 2 ТЕОРЕТИЧНІ ОСНОВИ ПРОЕКТУВАННЯ АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКУ ТА ВИБІР АРХІТЕКТУРНОГО СТИЛЮ РОЗРОБКИ

## 2.1 Аналіз типів високорівневої архітектури

Побудова архітектури веб-застосунку — це складний процес, спрямований на чітке визначення структури системи.

Для проектування архітектури немає універсального рішення. Кожен конкретний архітектурний шаблон може підходити лише для конкретного рішення. Все залежить від особливостей кожного проекту та його мети. Хоча можна створити будь-що на будь-якій архітектурі, це не завжди буде обґрунтованим.

Для високорівневої архітектури зазвичай вистачає лише ідеї проекту. Розглянемо три типи високорівневої архітектури веб-застосунків: монолітну, мікросервісну та серверлес.

### 2.1.1 Монолітна архітектура

Монолітна архітектура — це традиційний підхід до розробки програмного забезпечення, при якому весь додаток розробляється як одна єдина технологічна система [9]. Всі компоненти додатку взаємодіють один з одним і розгортання відбувається на одному сервері або групі серверів. Графічно монолітна архітектура наведена на рис. 2.1.

Загалом прийнято вважати це застарілим архітектурним рішенням, але воно має свої переваги та чудово підходить для певних типів проєктів. Хоча монолітна архітектура не завжди є оптимальним рішенням, вона все ще може бути корисною для певних видів застосунків та встановлення базової архітектури на ранніх стадіях розвитку.

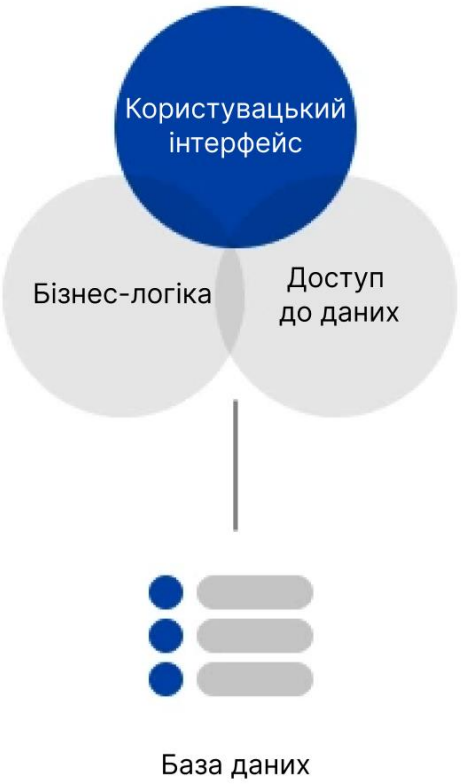


Рисунок 2.1 — Графічне зображення монолітної архітектури

Можна виділити наступні переваги даного підходу:

— простота розгортання, моноліти можна швидко та легко розгорнути, оскільки зазвичай у них є одна точка входу;

— розробка моноліту зазвичай відбувається швидко через їхню єдину кодову базу, яка завжди доступна;

— відладка моноліту спрощена через компактність — всі частини в одному місці, що дозволяє легше відстежувати код.

До недоліків відносяться:

— моноліти маштабуються як єдині цілісні системи, тому неможливо масштабувати окремі компоненти;

— надійність, у випадку неполадок в одному модулі виникають проблеми з усією системою;

— оновлення та зміни технологій у монолітах можуть бути надзвичайно складними;

— моноліти не дуже гнучкі, зміни в одному модулі майже завжди впливають на інші.

### 2.1.2 Мікросервісна архітектура

Мікросервісна архітектура — це концепція, що базується на розподілі модулів на індивідуальні системи, які взаємодіють за допомогою обміну повідомленнями [10].

Система в цілому складається з невеликих індивідуальних систем, які взаємодіють між собою. Графічне зображення мікросервісної архітектури наведене на рисунку 2.2.

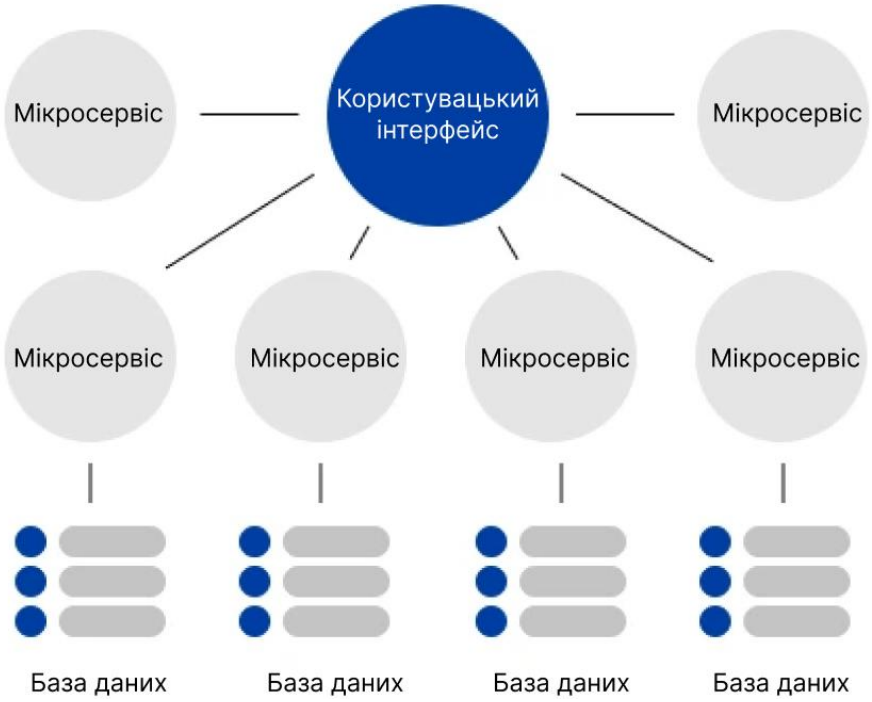


Рисунок 2.2 — Графічне зображення мікросервісної архітектури

Мікросервісна архітектура дуже популярна, але вона не завжди відповідає потребам малих та середніх веб-застосунків через свої недоліки.

В мікросервісній архітектури є такі переваги:

— мікросервіси демонструють велику гнучкість через ізольованість кожного сервісу, що означає, що зміни впливають тільки на конкретну частину системи;

— відмінність від монолітів полягає у можливості масштабувати лише певні складові системи, оскільки вони функціонують незалежно;

— гнучкість технологій, у мікросервісах кожна з частин системи може бути реалізована різними мовами програмування та технологіями;

— у випадку неполадок у одному з сервісів, загальна система зазвичай залишається працездатною, оскільки незруйнованість окремих частин важлива для системної стійкості.

До недоліків мікросервісної архітектури відносять:

— розробка мікросервісів складна через потребу створення та взаємодії декількох незалежних підсистем;

— комунікація між командами, часто виникає ситуація, коли різні команди працюють над різними підсистемами, що вимагає налагодження комунікації між ними для правильної взаємодії підсистем;

— відлагодження мікросервісів ускладнене, оскільки потрібно виявити, який саме сервіс відмовив і чому;

— початкове розгортання є складним, а додавання нових сервісів вимагає налаштування ключових частин проєкту.

### 2.1.3 Серверлес архітектура

Серверлесс — це архітектурне рішення, яке акцентується на процесі розробки, а не на налаштуванні та взаємодії між сервісами.

Серверлесс представляє собою альтернативу мікросервісам, де автоматизація розгортання відбувається завдяки хмарним технологіям [11]. Назва може ввести в оману, але варто зауважити, що тут присутня серверна частина, однак відсутність необхідності працювати з сервером як окремим середовищем. Приклад серверлес архітектури зображений на рис 2.3.

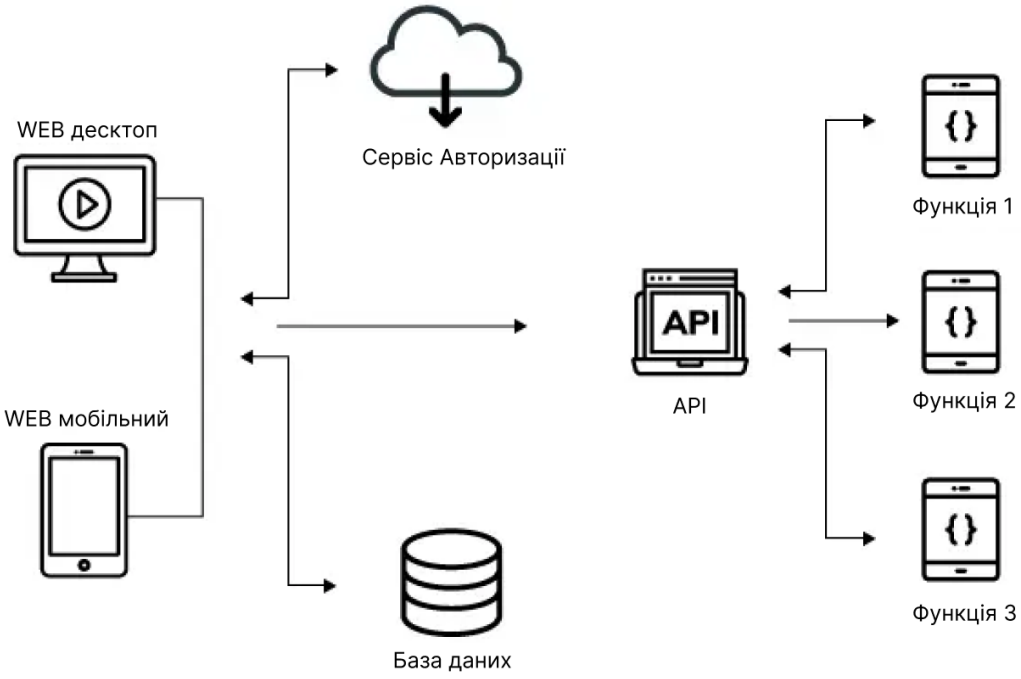


Рисунок 2.3 — Графічне зображення серверлес архітектури

Переваги серверлес архітектури:

— серверлесс відрізняється великою гнучкістю через відокремленість його модулів один від одного;

— абстракція від операційної системи (ОС), хмарна інфраструктура самостійно вирішує, яка операційна система потрібна та як її слід налаштувати, що дозволяє уникнути проблем з підтримкою ОС;

— легкий поріг входу, серверлесс зазвичай має простий та відокремлений код, що спрощує ознайомлення з проєктом;

— при правильному побудові проекту надійність серверлесс-рішення може бути такою ж, як і у випадку мікросервісів.

До недоліків серверлес архітектури відносять:

— гнучкість маштабування, хоча це перевага, розробник обмежений у впливі на маштабування, оскільки всі рішення приймає хмарна інфраструктура;

— подібно до мікросервісів, відлагодження ускладнюється взаємодією між компонентами;

— залежність від постачальника, кожна хмарова платформа має свої правила, що робить переїзд з однієї хмари на іншу майже неможливим;

— каскадне відмовлення, неправильна архітектура проекту може призвести до сильного впливу одних компонентів на інші, що може спричинити збій усього проєкту.

Проаналізувавши різні типи високорівневої архітектури, вибір був зроблений вибір в змішуванні монолітної та мікросервісної архітектур, що дало змогу спроектувати розподілену архітектуру, в якій клієнтська та серверна частини є окремими монолітами та пов’язані за допомогою клієнт-серверної взаємодії.

## 2.2 Обґрунтування вибору архітектурного стилю при розробці веб-застосунку

Стиль у програмній архітектурі — це набір принципів, які визначають шаблони та структуру для спрощення сприйняття. Це спосіб забезпечити спільне розуміння та комунікацію. Наприклад, коли одна команда розробляє архітектуру, а інша вносить зміни до функціоналу, стиль є тим зв'язком, який допомагає зрозуміти загальну структуру проекту.

В клієнт серверній взаємодії є декілька популярних архітектурних стилів, яким варто приділити увагу. Такі архітектурні стилі задають правила та навіть патерни використання запитів, їх формування та побудову і впливають на роботу всього додатку та внутрішньої архітектури сервера та клієнта. Ключова роль стилю — створити зрозумілу та цілісну архітектуру. Існують різні стилі, які використовуються у програмній архітектурі.

Архітектурний стиль клієнт-сервер — це концепція розробки веб-застосунків, що базується на розділенні функціоналу між двома основними компонентами: клієнтом і сервером [12]. У такому стилі клієнтська частина

відповідає за інтерфейс користувача та взаємодію з користувачем, тоді як серверна частина обробляє запити, надсилає необхідні дані та виконує бізнес-логіку. Цей стиль дозволяє розділити обов'язки між клієнтом і сервером, що полегшує масштабування, підтримку та розвиток системи. Клієнт та сервер можуть взаємодіяти через мережу, використовуючи протоколи комунікації, такі як HTTP, і забезпечують обмін даними для забезпечення функціональності веб-застосунку.

Архітектурний стиль компонентної архітектури розглядає систему як набір незалежних, самодостатніх компонентів, що мають чітко визначені функції та можуть бути використані повторно. У цій архітектурі програмне забезпечення розбивається на окремі логічні блоки, які можуть бути розвинуті, підтримані та перевикористані в різних місцях. Кожен компонент має свій власний інтерфейс, що дозволяє взаємодіяти з іншими компонентами системи. Це сприяє зменшенню залежності між компонентами та полегшує модифікацію або заміну окремих елементів без впливу на інші частини системи. Компонентна архітектура дозволяє створювати гнучкі та легко змінювані системи шляхом розподілу функціональності на чітко визначені компоненти з чіткими інтерфейсами спілкування між ними.

Також існує архітектурний стиль проблемно-орієнтованого дизайну, що фокусується на моделюванні та вирішенні конкретних проблем або бізнес-процесів, які виникають у веб-застосунку. Цей підхід ставить на перший план аналіз бізнес-потреб та створення програмних рішень, які оптимально відповідають цим потребам. Проблемно-орієнтований дизайн зосереджений на створенні моделей, що відображають бізнес-процеси, функції та взаємозв'язки між ними. Це дозволяє розробникам краще розуміти бізнес-логіку та потреби клієнтів для ефективного вирішення їхніх завдань через програмне забезпечення. Такий підхід сприяє розробці більш адаптивних та зорієнтованих на користувача систем, оскільки вони більш точно відображають специфіку бізнесу та його потреби.

Багатошаровий архітектурний стиль для веб-застосунку полягає в розділенні функціональності програми на відокремлені "шари" або рівні, кожен з яких відповідає за конкретний аспект роботи застосунку. Кожен шар має свої власні завдання та відповідальності, і вони взаємодіють між собою через чітко визначені інтерфейси. Це дозволяє розділити логіку програми на менші, керовані шарами, що полегшує розробку, тестування та підтримку веб-застосунку.

Шина повідомлень — це архітектурний стиль, що дозволяє взаємодіяти різним частинам веб-застосунку через спільний канал обміну повідомленнями, не прив'язуючи їх до конкретних компонентів. Повідомлення публікуються на цій шині, а модулі, зацікавлені певним типом повідомлень, можуть підписатися на них і отримувати відповідні дані. Це сприяє гнучкості застосунку, оскільки дозволяє додавати чи видаляти компоненти без змін у вже існуючих, сприяючи підвищенню масштабованості та зменшенню залежностей між її частинами.

Трирівневий архітектурний стиль веб-застосунку розподіляє функціональність застосунку на три відокремлені рівні, які розташовані на різних комп'ютерах чи серверах. Перший рівень — це представлення, відповідальне за відображення інформації користувачам та їх взаємодію з програмою через інтерфейс. Другий рівень, логіка бізнес-процесів, містить бізнес-логіку системи та обробку даних. Нарешті, рівень доступу до даних забезпечує роботу з базами даних чи іншими джерелами даних. Цей стиль сприяє розділенню відповідальностей та полегшує розробку, тестування та підтримку веб-застосунку та забезпечує взаємозв'язок та обмін даними між різними рівнями.

Об'єктно-орієнтований стиль архітектури відображає веб-застосунок як набір взаємодіючих об'єктів, кожен з яких має свій набір даних та методів, які визначають його поведінку. Ці об'єкти можуть бути повторно використані та взаємодіяти один з одним, створюючи більш складні функції та системи. Цей підхід сприяє модульності, гнучкості та легкості розширення застосунку,

оскільки кожен об'єкт відповідає за конкретну функціональність, і зміни в одному об'єкті мало впливають на інші.

Сервісно-орієнтований стиль архітектури полягає в організації веб-застосунку через використання окремих послуг (сервісів), які надають певні функції та можливості. Ці сервіси взаємодіють між собою, обмінюючись повідомленнями чи запитами, щоб забезпечити певну функціональність застосунку. Цей підхід дозволяє створити єдине незалежне середовище для функціональності, де кожен сервіс має чітко визначені межі та може бути розвинутий, масштабований чи замінений без впливу на інші частини застосунку. Сервісно-орієнтований підхід сприяє гнучкості та розширюваності системи шляхом створення модульних, легко масштабованих сервісів, що спільно працюють для забезпечення більшої функціональності.

Зазвичай архітектура та дизайн програмного забезпечення не обмежуються лише одним стилем. В більшості випадків застосовується комбінація різних підходів, що веде до створення повноцінної системи. Кожен стиль описується у технічній документації, щоб команда розробників могла чітко передати інформацію. Крім того, вимоги до безпеки системи змушують використовувати різні стилі, включаючи багаторівневу архітектуру.

Вибір архітектурного стилю залежить від багатьох факторів, таких як обмеження інфраструктури розробки, вибір технологічного стеку та рівень досвіду розробників.

Обираючи архітектурний стиль клієнт-сервер для веб-застосунку, вибір був обґрунтований важливістю розділення функцій та відповідальностей між клієнтською та серверною частинами. Цей підхід дозволяє чітко визначити, що клієнт, виконуючи запити, отримує доступ до ресурсів на сервері через стандартні протоколи. Такий розподіл функцій допомагає забезпечити ефективну обробку запитів, підвищити масштабованість та краще керувати ресурсами. Крім того, архітектурний стиль клієнт-сервер є одним з найпоширеніших та дозволяє забезпечити більшу гнучкість та швидкість в роботі, що є критичними для веб-застосунків.

## 2.3 Аналіз сучасних методологій розробки

Методологія розробки відіграє важливу роль в розробці будь-якого веб-застосунку. Правильно вибрана методологія дозволить швидше та продуктивніше закінчити роботу над застосунком, тому розглянемо основні з них.

Каскадну методологію відносять до класики, тому що це одна із перших методологій розробки, яка представлена на рисунку 2.4. Її суть полягає в послідовному виконанні етапів життєвого циклу: аналіз, дизайн, розробка, тестування, реліз і підтримка. Кожна стадія має бути завершена до початку нової [13]. Перевагою цієї моделі є те, що оцінку якості можна виконувати після кожного етапу. Але складно уявити веб-застосунок, який можна було б виконати строго послідовно. Тому модель часто вважають застарілою. Як до плюсів, так і до мінусів можна зарахувати те, що при використанні цього підходу терміни його реалізації та чітке технічне завдання точно визначені.

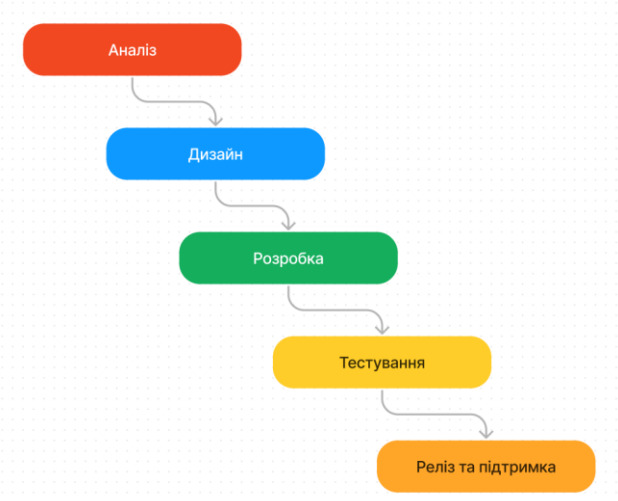


Рисунок 2.4 — Візуальне представлення каскадної методології

Серед переваг каскадної методології можна визначити:

— простота у розумінні та використанні;
— чітка специфікація вимог;
— контроль над витратами;

— якість результату.

Також каскадна методологія має наступні недоліки:

— жорстка послідовність;

— важкість виправлення помилок;

— довгий час до конкретних результатів;

— не підходить для складних та проектів, які можуть змінюватись в процесі.

Agile, або гнучка методологія розробки веб-застосунків має всі етапи життєвого циклу розробки, які можуть бути виконані за одну ітерацію і бути готовими до будь-яких змін (рисунок 2.5).

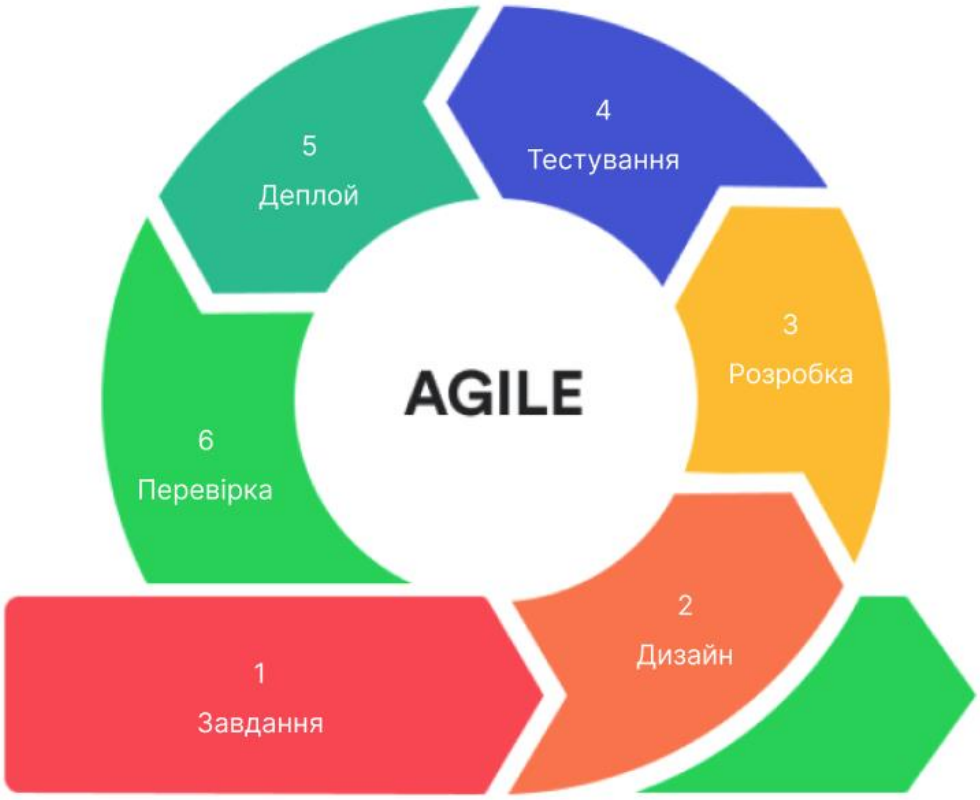


Рисунок 2.5 — Візуальне представлення методології Agile

Тобто розробка застосунку ділиться на спринти — відрізки часу, за які має бути отримано результат, зазвичай від одного до чотирьох тижнів. У

кожному спринті є свій список завдань, який має бути виконаний до кінця ітерації, кожне із завдань має свій рівень оцінки [14].

Крім цього, на початку спринту проводиться зустріч із планування завдань на ітерацію, а наприкінці — зустріч для обговорення результатів.

Дану методологію варто застосовувати, коли веб-застосунок постійно адаптується до умов ринку, має великий обсяг і довгий життєвий цикл.

Методологія розробки V-Model є послідовною методологією, що базується на послідовному виконанні етапів розробки та тестування [15].

Переваги методології розробки V-Model:

— чіткість та структурованість;

— аналіз вимог;

— фокус на тестуванні;

— прозорість процесу.

Недоліки методології розробки V-Model:

— жорстка послідовність;

— важко вносити зміни;

— ризик від затримок;

— не враховує змін;

— не підходить для складних проектів.

Хоча V-Model має свої переваги у вигляді чіткої структури та фокусу на тестуванні, вона може бути недостатньо гнучкою для сучасних веб-застосунків, де зміни є нормою.

В результаті аналізу сучасних методологій розробки, для розробки веб-застосунку була обрана каскадна методологія. Цей вибір обґрунтований знанням чітких термінів готовності веб-застосунку, його простота для невеликої команди розробки та безпосередній контроль над обсягом зробленої роботи.

## 2.4 Аналіз архітектурних шаблонів

Для безлічі завдань у розробці архітектури застосовують патерни, відомі також як шаблони. Вони дозволяють розв'язувати схожі проблеми більш ефективно, сприяючи скороченню часу розробки та спрощенню процесів. Коли паттерн описує певне завдання, його застосування у відповідному контексті дозволяє використовувати вже перевірені рішення замість того, щоб "винаходити велосипед" знову.

Шаблон представляє собою опис взаємодії різних елементів та класів, які складають архітектуру програмного рішення, зроблений у відповідності до контексту конкретної задачі та вимог проектування. Завдяки можливості ідентифікації ключових об'єктів структури та їхньому повторному використанню, використання патернів спрощує процес побудови архітектури.

Кожен шаблон має свої переваги та обмеження. Нижче наведено приклади готових архітектурних шаблонів та короткий опис принципів їх роботи:

— багаторівневий шаблон, цей підхід полягає в розбитті системи застосунку на рівні, що відображаються на діаграмі, кожен рівень може викликати лише той, що знаходиться нижче, що спрощує внесення змін до окремих компонентів без впливу на інші області, але така структура може бути складною та навантаженою, що впливає на продуктивність;

— шаблон посередника, при великій кількості модулів пряма взаємодія може стати заплутаною, посередник спрощує взаємодію, але втрата його працездатності може призвести до зупинки системи;

— модель подання-контролер, цей шаблон дозволяє відокремити інтерфейс від даних, спрощуючи зміну інтерфейсу без впливу на роботу системи;

— клієнт-серверний паттерн, застосування цього підходу дозволяє обмежити доступ користувачів до ресурсів та зробити систему більш доступною;

— паттерн «Фабричний метод», цей паттерн дозволяє додавати нові об'єкти різних типів, спрощуючи процес та зберігаючи систему незалежною.

Існує значна кількість шаблонів, що використовуються у розробці веб-застосунків. Проте це не означає, що всі вони мають однакову структуру або повторюють один одного. Вони ґрунтуються на загальних концепціях, а сама архітектура будується індивідуально, враховуючи контекст та особливості проекту.

Аналізуючи різноманітні архітектурні шаблони для розробки веб-застосунку, був обраний клієнт-серверний паттерн у зв’язку з тим, що цей шаблон дозволяє ефективно обмежити доступ користувачів до ресурсів та забезпечити більшу доступність системи, створюючи чітку модель взаємодії між клієнтом і сервером. Обраний паттерн спрощує архітектурну структуру застосунку, полегшуючи розвиток та його підтримку. Використання клієнт-серверного підходу дозволяє раціонально розподілити функції між клієнтом (взаємодія з користувачем) та сервером (забезпечення ресурсами), що сприяє покращенню продуктивності, масштабованості та безпеки системи.

## 2.5 Розробка загальної структури веб-застосунку

Для початку розробки веб-застосунку необхідне чітке технічне завдання, яке буде описувати процес розробки та який функціонал в цілому повинен бути в результаті роботи. До такого методу є альтернатива та, у кращому випадку, доповнення — структурна схема веб-застосунку.

Маючи структурну схему, розробники відразу можуть побачити весь об’єм роботи, кількість необхідного функціоналу та зв’язки між сутностями веб-застосунку.

В роботі розроблено загальну структуру веб-застосунку, яка наведена на рис. 2.6. Структура відображає, як користувач буде користуватися цим застосунком.

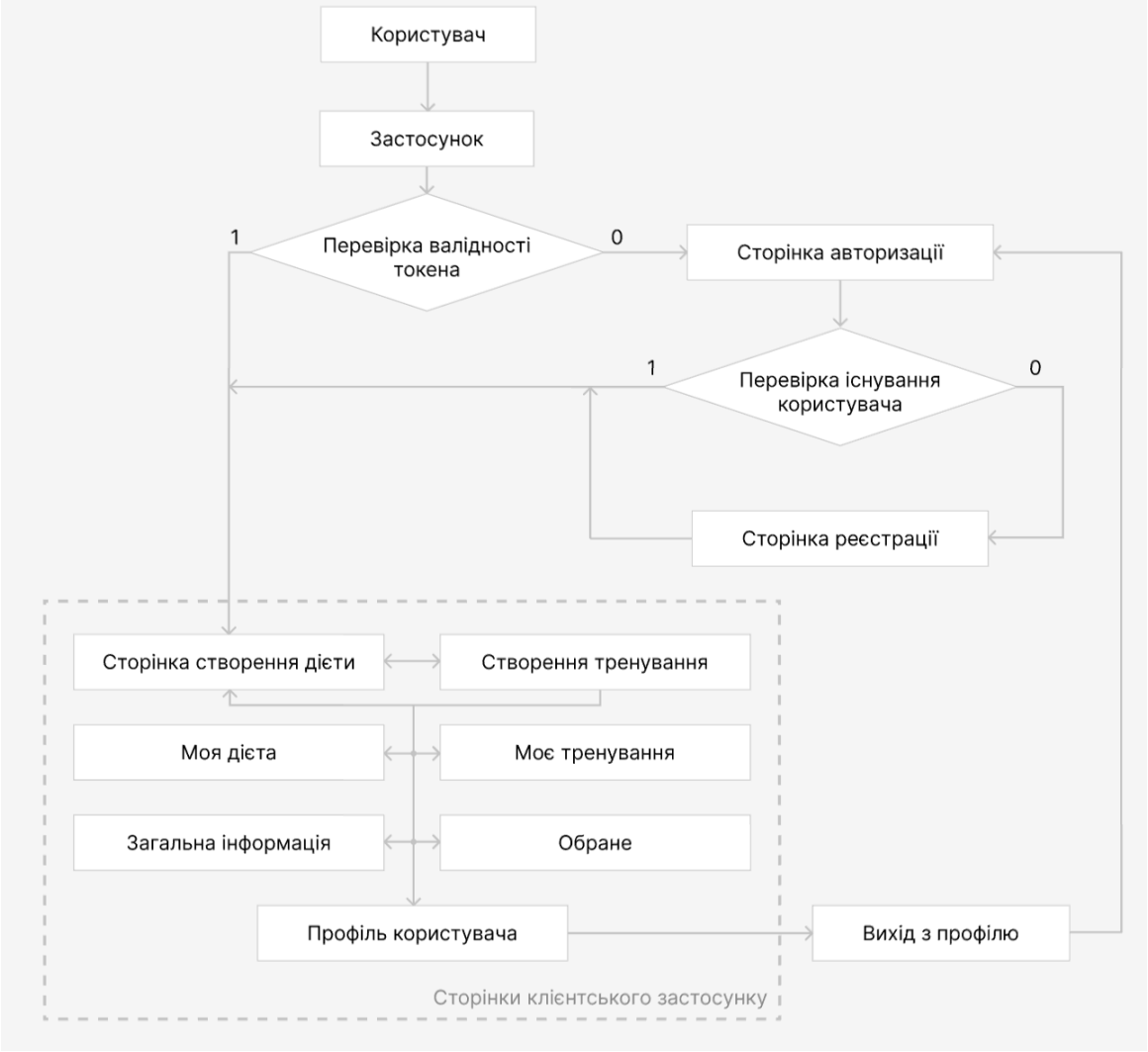


Рисунок 2.6 — Загальна структура веб-застосунку

Коли користувач заходить в додаток, здійснюється перевірка на валідність авторизаційного токену. Якщо токен невалідний, то здійснюється перехід на сторінку авторизації. Якщо користувач не був зареєстрований раніше, то на сторінці реєстрації він має можливість створити новий акаунт зі всіма необхідними даними.

Коли користувач знову входить у застосунок за допомогою форми авторизації, або створює нового користувача, він попадає на сторінку створення дієти.

Також взаємодію між всіма описаними компонентами можна представити у вигляді UML-діаграми розгортання (додаток Г).

Основні сторінки застосунку зв’язані між собою навігацією та користувач може переходити між цими сторінками без будь-яких обмежень. Лише сторінки створення дієти та створення тренування пов’язані напряму між собою, так як представляють приблизно однакове наповнення. Якщо користувач захоче змінити акаунт або створити новий, у нього є можливість зайти на сторінку профілю та вийти з акаунту, перейшовши таким чином на сторінку авторизації.

# 3 ВДОСКОНАЛЕНИЙ МЕТОД КЛІЄНТ-СЕРВЕРНОЇ ВЗАЄМОДІЇ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКУ

## 3.1 Огляд технологій для розробки серверної та клієнтської частин

Для розробки веб-застосунку є поставлені вимоги в масштабованості застосунку, його продуктивності, швидкодії та швидкості самої розробки. Щоб відповідати поставленим критеріям, необхідно зробити огляд найбільш популярних мов програмування, що будуть застосовані на серверній та клієнтській частинах. Також мови програмування мають власні фреймворки, що також достатньо збільшують як швидкодію застосунку, так і швидкість розробки.

Одними з популярних технологій розробки серверної частини веб-застосунку є ASP.NET та Node.js.

Node.js — це середовище виконання JavaScript, адаптивне до платформи, яке дозволяє розробникам створювати швидкі, масштабовані мережеві додатки. Він використовує неблокуючу модель вводу/виводу, керовану подіями, яка є легкою та ефективною. Додатки Node.js можуть працювати в браузері, на сервері, в хмарному середовищі або на мобільних пристроях [16].

Платформа ASP.NET з'явилася у 2002 році. Сьогодні це фреймворк веб-додатків з відкритим вихідним кодом від Microsoft. Він поєднує в собі ефективність і точність архітектури, новітні ідеї та гнучкі методи розробки програмного забезпечення [17].

Це комплексна платформа для створення веб-додатків. Платформа включає в себе набір компонентів та інструментів для створення, розгортання та управління веб-додатками. Платформа відома своєю продуктивністю та масштабованістю. Вона також має чітко визначену структуру і забезпечує чітко визначену модульність, що полегшує розробку і розгортання додатків.

При виборі .NET для серверної частини було приділено увагу наступним його перевагам над Node.js:

— потужність і гнучкість;

— кастомізація та масштабованість;

— функції безпеки;

— керованість;

— крос-платформенна міграція.

Дані переваги дозволять масштабувати веб-додаток на мобільний додатак або ж десктопний додаток, кардинально не змінюючи підхід.

Щодо клієнтської частини, для веб-застосунків використовується мова програмування JavaScript, що відверто домінує над всіма іншими варіантами, тому вибір очевидний.

Але в самій мові програмування JavaScript є безліч фреймворків та бібліотек, що збільшують швидкість розробки, покращують швидкодію застосунку та роблять його реактивним. Серед самих популярних є такі фреймворки як React, Angular та Vue. Також з’явився новий фреймворк Qwik, робота якого дещо відрізняється від інших популярних рішень.

Angular — це фреймворк, який розроблявся відразу з введенням типізації за допомогою TypeScript. Цей фреймворк є найбільш зручним для розробників серверної частини, тому що підходи до розробки клієнтської частини за допомогою Angular схожі до принципів розробки на ASP.NET.

Перевагами Angular є використання TypeScript «з коробки», детальна документація, одностороння прив’язка даних, що стандартизує поведінку веб-застосунків та призводить виникнення різних помилок до мінімуму. Його недоліками є низька продуктивність застосунків в порівнянні з іншими фреймворками та високий поріг входження для вивчення.

Vue — це легкий та реактивний фреймворк, що підходить для невеликих застосунків. Його перевагами є власні атрибути, що можна використовувати безпосередньо в HTML, хороша документація, масштабування та відносно невеликий розмір самого фреймворку, що впливає на швидкість першого завантаження сторінки. Серед недоліків відмічаються невелика поширеність

серед розробників відносно інших фреймворків та складність інтеграції в великі проекти.

Qwik — це новий тип веб-фреймворка, який може забезпечувати миттєве завантаження веб-додатків будь-якого розміру або складності [18]. Веб-сайти та додатки можуть запускатися лише з приблизно 1КБ JavaScript (незалежно від складності додатку) та досягати стабільної продуктивності в масштабуванні.

Qwik дуже схожий на інші веб-фреймворки. Qwik — це фреймворк, який рендерить дерево компонентів, створюючи інтерактивний додаток.

Унікальність Qwik полягає не в тому, що він робить, а в тому, як він досягає своїх цілей. Метою Qwik є надання миттєвого доступу до додатку, навіть на мобільних пристроях. Qwik досягає цього завдяки двом основним стратегіям:

— відкладення виконання та завантаження JavaScript настільки довго, наскільки це можливо;

— серіалізація стану виконання додатка та фреймворка на сервері та відновлення його на клієнті.

Метою Qwik є забезпечення можливості завантажувати та виконувати лише абсолютно необхідний мінімум додатка.

React — це найпопулярніший фреймворк, який вирізняється своєю зручністю, гнучкістю та відносною простотою в використанні для створення інтерфейсу користувача.

Однією із ключових переваг React перед іншими відомими фреймворками є можливість вибору лише необхідного набору бібліотек для включення в проект. Це дозволяє уникнути зайвого навантаження модулями та покращити продуктивність веб-додатку та його оптимізацію [19].

Компонентний підхід, який React використовує в розробці, розширює структурні можливості проекту. Тому що одні й ті самі компоненти можна використовувати в різних частинах проекту. За допомогою props (об'єкту з

даними) ви можете змінювати їх відображення або поведінку в залежності від потрібного результату.

Веб-додатки, розроблені за допомогою React, відомі як односторінкові додатки (SPA), тому що при переході між сторінками в браузері не відбувається повторного завантаження сторінки. Змінюється лише динамічна частина веб-інтерфейсу.

Для розробки був обраний фреймворк Qwik, що обґрунтовано його першою і найважливішою перевагою над React та іншими фреймворками є швидкість завантаження сторінки та виконання функціональних елементів веб-додатку. Це досягається новітнім підходом рендеру, який пропонує Qwik, та відкладеним виконанням скриптів. Тобто на момент завантаження сторінки, завантажуються лише ті скріпти, які необхідні в даний час. Коли, наприклад, натискається кнопка, що повинна виконати конкретну дію, лише після цього завантажується відповідний скрипт з серверу, що виконає функціональну частину після натискання.

Другою ж перевагою можна назвати велику кількість вбудованих технологій(хуків), які відразу можна використовувати в компонентах, наприклад useResource для виконання запитів до серверу та використання отриманих даних. Це спрощує та прискорює розробку, так як в React для подібних речей потрібно зробити попереднє налаштування.

## 3.2 Методи клієнт-серверної взаємодії

В розподіленій архітектурі важливу роль грає клієнт-серверна взаємодія, де сервер виступає джерелом обчислень, збереження даних та роботою з ними, більшої частини бізнес-логіки. Клієнт, в свою чергу, може спілкуватись з сервером, «просити» якісь дані в серверу. Це працює в форматі request-response — клієнт робить запит та отримує відповідь, яку вже відображає в залежності від деталей відповіді.

Запит зі сторони клієнта може не тільки бути направленим на отримання даних, а і на створення користувача чи іншої бізнес-моделі, проведення

оплати, редагування даних і т.д. З іншої сторони, сервер повертає відповідь клієнту з відповіддю чи запит був виконаний чи відхилений по якійсь конкретній причині. Схематичний приклад такої архітектури представлений на рисунку 3.1:

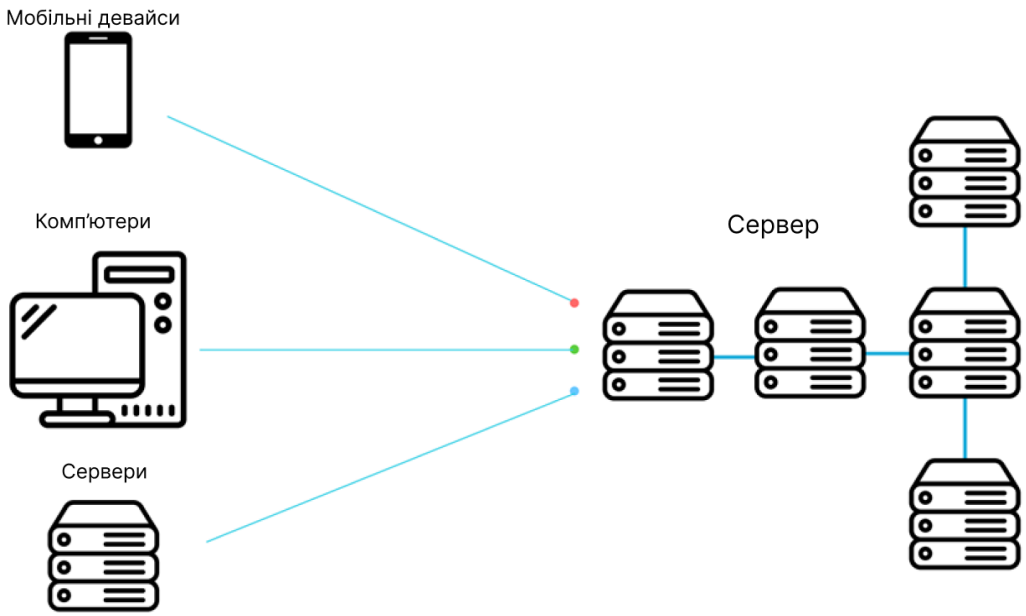


Рисунок 3.1 — Схематичне зображення клієнт-серверної взаємодії

Саме значення клієнт-серверної взаємодії не означає, що обмін даними може бути лише між браузером та сервером. Це означає що під клієнтом може виступати як браузер, так і телефон чи інший прилад, десктопний додаток чи навіть інший сервер. Як було сказано вище — сервер це джерело інформації та управління нею, а клієнт це споживач цих даних та спосіб відображення для звичайних користувачів.

Дана архітектура дозволяє використовувати різні клієнти, що являється дуже масштабованим рішенням. Також під сервером розуміється єдиний додаток, до якого можна звернутись, але влаштований він може бути як завгодно в плані виконання.

### 3.2.1 Архітектурний стиль REST API

REST API — це архітектурний стиль, набір правил для серверного додатку, який опису найбільш ефективне використання HTTP та розробляти API так, щоб додаток легко витримував навантаження та ефективно масштабувався.

Даний архітектурний стиль має декілька концепцій, перша із яких описує саме клієнт-серверну модель взаємодії.

Друга концепція описує багаторівневість (багатошаровість) системи. Це, знову ж таки, дозволяє масштабувати серверний додаток. Тобто сервер може складатись з багатьох серверів, може мати мікросервісну архітектуру, що в результаті для клієнта не грає великої ролі, тому що це незалежна частина архітектури.

Третя концепція вказує на відсутність стану сервера. Це означає, що при обміні запитами між клієнтом та сервером, ніякий проміжний стан клієнта з точки зору сервера не запам’ятовується. Тобто при кожному новому запиті від клієнта, клієнт та сервер обмінюються інформацією як наче в перший раз. Для того, щоб сервер ідентифікував клієнта, міг виконати необхідний запит — клієнту потрібно відправляти повну інформацію, з якою сервер зможе виконати такий запит.

Наступна концепція включає одноманітний уніфікований інтерфейс. До цієї концепції можна віднести схожу роботу з різними даними бізнес-моделей. Наприклад, робота з користувачами може включати в себе всі CRUD операції по типу створення, видалення, оновлення даних користувача. По такому ж принципу буде працювати робота з дієтами. Дієту можна створити, зберегти або ж видалити. Якщо авторизаційний заголовок називається Authorization, то він повинен так називатись у всіх місцях API без виключень.

В REST API є стиль версій, які можуть змінюватись. Тобто якщо застосунок вже являється комерційним і потрібно замінити якусь частину ендпоінтів в API, це не можна зробити напряму, так як такі зміни будуть не

обернено-сумісними і потрібно буде робити багато змін в клієнтському коді. В такому випадку до URL, який вказується в запиті додається назва версії конкретного ендпоінту, яка буде використовуватись на сервері. Наприклад: /api/users POST перетвориться на /api/v2/users POST.

Для документації API можна використовувати таку специфікацію, як OpenAPI або ж Swagger, який імплементує OpenAPI.

Також слід не забувати про кешування, яке може бути виконане як засобами HTTP за рахунок використання відповідних заголовків так і сторонніми засобами на сервері. GET та POST запити можуть кешуватись, а PUT та DELETE не кешуються. Це дозволяє досягти в першу чергу — швидкодії виконання запитів. Так як при виконанні запиту на сервер, якщо він кешується, то наступного разу запит виконуватись не буде, а просто візьме необхідні дані з браузерного кешу (або серверного). З іншої сторони це ще й оптимізує навантаження на сервер, так як кількість повторюваних запитів до нього в рази зменшиться.

В REST API можна обмінюватись різними типами даних, але найбільш поширеним серед них є JSON, хоча доволі часто використовується і XML. JSON — це формат обміну даними.

JSON не залежить від мови програмування, тобто його можна використовувати з будь-якою мовою програмування, а структура даних, що лежить в його основі, не залежить від платформи. Незалежність JSON від мови робить його ідеальним для використання у веб-розробці, де може знадобитися обмін даними з іншими мовами програмування, такими як Ruby або JavaScript.

XML — це відкритий стандарт для зберігання та обміну даними. Це мова розмітки для опису структури та вмісту будь-якого XML-файлу, наприклад, документів, веб-сторінок або баз даних. Можна вважати, що XML схожий на HTML, але краще: він дозволяє прикріплювати додаткову інформацію до вузлів документа, не змінюючи при цьому основний формат.

Дані формати мають наступні відмінності:

— синтаксис, XML являється текстовим форматом, який використовує теги для організації даних, в свою чергу JSON також текстовий формат, але використовує пари «ключ-значення» та розділювачі у вигляді фігурних дужок;

— читабельність, XML представляє більш читабельний варіант даних, оскільки на вигляд інформація більш структурована, JSON більш компактний та менш об’ємний, що додає йому плюс в плані ефективності обміну даними;

— підтримка типів даних, XML надає можливість визначення типів даних для елементів, а JSON такої підтримки не має;

— використання в мовах програмування, JSON є більш популярним форматом для обміну даних та має багато бібліотек для коректного парсингу таких даних, XML також підтримується багатьма мовами програмування, але JSON є більш зручним в використанні.

### 3.2.2 Протокол SOAP

SOAP — це протокол обміну структурованими повідомленнями. Якщо REST використовується з протоколом HTTP, то SOAP може використовуватись з любим протоколом прикладного рівня: SMTP, FTP, HTTP.

Для опису SOAP-сервісів використовується WSDL (Web Services Description Langiage) — мова опису веб-сервісів і доступу до них, основаних на форматі XML. В даному випадку немає поняття ендпоінтів, як в REST, тут використовуються операції, які також мають назву, якісь вхідні і вихідні дані.

Якщо REST являється набором правил, архітектурним стилем, то SOAP є вже протоколом, який представляє конкретну строгу структуру запитів, що будуть використовувати його, як показано на рисунку 2.3.

З рисунку видно, що повідомлення SOAP складається з трьох частин: Envelope, Header та Body.

Envelop — це кореневий елемент, що визначає початок та кінець повідомлення. Саме за допомогою цього клієнт розуміє коли повідомлення повністю отримане.

Header — щось схоже до заголовків в REST API. Дозволяє додати додаткові властивості до запиту, відправити якийсь токен, вказати формат повідомлення і любе інше допоміжне повідомлення.

Body — частина повідомлення, в якій передаються конкретні дані, що необхідні для повідомлення і успішного його виконання.

Знову ж таки, клієнт-серверна архітектура дозволяє використовувати на сервері відразу і REST контролери і SOAP контролери, які клієнт в разі необхідності буде використовувати.

### 3.2.3 Мова запитів GraphQL

Мова запитів GraphQL представляє собою основну можливість запитів необхідної кількості даних для клієнта. Це вирішує проблему непотрібності даних.

Наприклад, якщо продукт в інтернет-магазині має безліч полів, від характеристики до відгуків, то при попередньому перегляді всі ці дані непотрібні. Для реалізації такого в REST API необхідно було б створювати додатково ще один ендпоінт або ж завжди отримувати весь перелік даних, хоча 90% з них можуть не використовуватись. Це навантажує і сервер, і клієнт, але GraphQL вміло вирішує дану проблему.

В мові запитів GraphQL сервер представляє схему для конкретної бізнес-моделі. Така схема може включати безліч необхідних полів, які будуть необхідні при розробці додатка. В свою чергу клієнт, виконуючи запит до сервера, просто вписує необхідні поля, що будуть використані в результаті. Сервер віддає клієнту тільки ті поля, які він запросив, таким чином досягається оптимізація всіх запитів в додатку та зручне маніпулювання ними.

Аналогом GET запиту в REST API є Query в GraphQL. В свою чергу Mutation є аналогом до POST/DELETE/PUT запитів. Також в GraphQL є

додатковий вид запитів, який називається Subscription. Це підписка в реальному часі на зміни конкретних даних на сервері, що реалізована поверх веб-сокетів.

Ще до переваг GraphQL входить його частична самодокументація. Так як при створені схем відразу визначаються типи полів, які мають передаватись, так і відразу створюється їх документування, що являється досить зручним для розробників клієнтської частини. Також до переваг входить кодогенерація, тобто за допомогою допоміжних інструментів на основі схеми GraphQL можна створити відразу TypeScript інтерфейси, що досить скорочує час розробки. В тому числі можна генерувати цілі запити або ж додаткові ендпоінти для бібліотеки react-query або RTK Query, що являється частиною Redux Toolkit.

### 3.2.4 Протокол веб-сокетів

На відміну від HTTP, де запит виконується, отримується якась відповідь і на цьому все — веб-сокети виконують підписку на якісь події, що можуть відбутись на сервері. Наприклад — отримання нового повідомлення в чаті або графіки, в яких потрібно швидко показувати зміни. При такому з’єднанні клієнт та сервер безперервно обмінюються повідомленнями і в разі змін відповідних даних клієнт відразу буде про це знати і оновлювати інформацію.

Веб-сокети в більшості випадках не використовуються як єдиний протокол клієнт-серверної взаємодії. Частіше всього їх використання необхідне лише в деяких місцях додатку, як було сказано раніше, в чатах, різного типу повідомленнях, графіках і т.д.

### 3.2.5 Віддалений виклик процедур RPC

Віддалений виклик процедур дозволяє клієнту виконувати серверні методи. Тобто сервер реалізовує якусь конкретну процедуру, а клієнт може викликати цей метод як наче він реалізований на клієнті. Насправді в цей

момент відбувається мережевий запит на сервер від клієнта, але за цим не потрібно слідкувати, в коді це звичайний виклик методу.

gRPC — це технологія від Google, яку зараз використовують в тих же мікросервісних архітектурах. Далі перечислені переваги використання gRPC:

— використання HTTP 2 замість HTTP 1/1;

— куди швидше виконання запитів;

— простий і швидкий стрімінг даних (потокова передача);

— замість JSON використовується бінарний формат даних protobuf;

— багато реалізованих можливостей відразу «із коробки»;

— зручний виклик процедур.

По тестам HTTP 2 швидший за HTTP 1/1 на 15%. За допомогою бінарного формату, дані можна краще стиснути, швидше відправити та ефективніше опрацювати. З додаткових переваг формату даних protobuf можна виділити строгу типізацію даних, які відправляються.

На лістингу 3.1 описаний сервіс з двома процедурами, в яких на вхід очікуються об’єкти з відповідними до message полями. На виході ж ми отримуємо код на багатьох мовах програмування, що робить цей код універсальним після його компіляції.

Лістинг 3.1 — Сервіс з двома процедурами за допомогою gRPC

```
package helloworld;
service Greeter {
        rpc SayHello (HelloRequest) returns (HelloReply) {}
        rpc SayHelloStreamReply (HelloRequest) returns (stream HelloReply)
        {}
}
message HelloRequest {
        string name = 1;
}
message HelloReply {string message = 1;}
```

На лістингу 3.2 показаний приклад виклику процедури з лістингу 3.1.

Лістинг 3.2 — Виклик gRPC процедури в JavaScript

```
client.sayHello({name: user}, function(err, response) {
        console.log('Greeting:', response.message);
});
```

В результаті огляду різних методів клієнт-серверної взаємодії, було прийнято рішення використовувати REST API, як найбільш традиційний підхід без необхідності писати додаткові схеми, як в випадку з GraphQL. Також використання RPC дозволяє виконувати запити до бази даних відразу із клієнтської частини, що значно пришвидшує виконання запиту.

## 3.3 Вдосконалення методу клієнт-серверної взаємодії

Клієнт-серверна архітектура є одною з найбільш популярних при розробці веб-застосунків. Існують різні методи для зв’язку клієнтської та серверної частин. В цих методах база даних відноситься до серверної частини та безпосередній доступ до неї є лише в сервера. Щоб змінити якісь дані, або їх отримати на клієнтській частині, необхідно використовувати серверну частину та робити запити до неї, що різняться відносно методу взаємодії. Але для ще більшого пришвидшення взаємодії з базою даних, було вдосконалено метод клієнт-серверної взаємодії шляхом використання проміжного серверу в клієнтській частині, що дало змогу виконувати запити до бази даних безпосередньо з неї.

Схематично реалізацію методу клієнт-серверної взаємодії можна представити за допомогою структурної схеми, яка наведена на рисунку 3.2.

Веб-застосунок розділяється на клієнтську та серверну частину, які спілкуються між собою в першу чергу через проміжний сервер, що відноситься до клієнтської частини, та браузерні Cookies, до яких є доступ як в клієнта, так і в сервера.

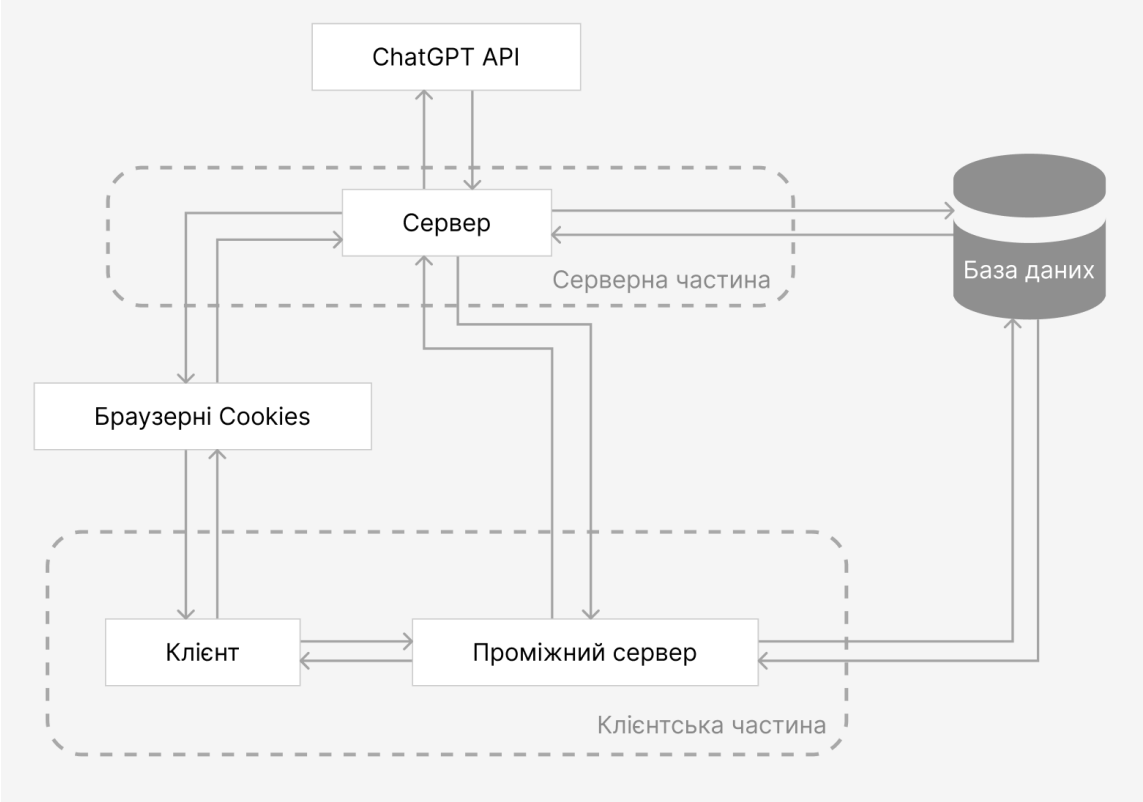


Рисунок 3.2 — Структурна схема реалізації методу клієнт-серверної взаємодії

Проміжний сервер виступає в ролі проксі між клієнтом та сервером, що надає можливість виконувати запити до бази даних відразу з клієнтської частини.

Крім того, сервер взаємодіє ще й з ChatGPT API та базою даних, що дозволяє зберігати різні дані та створити базовий CRUD (створення, читання, оновлення та видалення).

Такі можливості дозволяють нам мати доступ до бази даних в разі необхідності та, без використання віддаленого серверу, виконувати якісь прості дії з ним. Наприклад, коли потрібно зробити багато запитів до серверу для отримання інформації, можна просто використати даний підхід, який буде більш оптимізованим, ніж використання звичайного REST API.

Крім того, часто після виконання запиту необхідно виконати обробку даних, які повернув сервер. В такому випадку зазвичай на клієнті робляться цикли, в яких відбувається фільтрація. Але якщо обсяг даних є дуже великим,

то це може викликати проблеми з продуктивністю на клієнті та навіть «провисання» в момент обробки. Але використання проміжного серверу сильно полегшує даний процес, тому що вся обробка відбувається на сервері та не відноситься до клієнта. Все що потрібно знати користувачу в даний момент — це те, що процес відбувається, тобто показати прогрес або ж лоадер для кращого візуального сприйняття.

Для зв’язку між проміжним сервером та клієнтом використовується нативна функція для отримання даних fetch, але через розташування цього серверу безпосередньо в клієнтській частині, цей запит відбувається швидше ніж до віддаленого серверу.

В результаті вдосконалений метод клієнт-серверної взаємодії являється універсальним методом, що може комунікувати як з віддаленим сервером, так і напряму між клієнтською частиною та базою даних.

## 3.4 Реалізація роботи протоколу HTTP при клієнт-серверній взаємодії

Гіпертекстовий транспортний протокол знаходиться на найвищому рівні моделі OSI (рисунок 3.3), прикладному рівні — рівні додатків. Саме за допомогою HTTP більшість додатків обмінюються інформацією в інтернеті.

Першочерговою ідеєю HTTP був обмін гіпертекстовими документами, тобто HTML версткою. Але зараз по http можна передавати майже любі дані: текстові, файлові, html, xml, json.

Шаблонна структура HTTP запиту складається з таких частин:

— метод, семантика запиту, який ми хочемо зробити;

— url, запис, який означає куди саме ми відправляємо запит;

— версія http, запис, який вказує яка версія http використовується при запиті;

— заголовки, записи, що несуть важливу інформацію про запит;

— body, дані, які відправляються разом з запитом, це можуть бути авторизаційні дані, чи ідентифікаційний номер користувача і т.д.


Рисунок 3.3 — Декілька рівнів моделі OSI

Приклад http запиту представлений в лістингу 3.3.

Лістинг 3.3 — Шаблон HTTP запиту

```
POST /login HTTP/1.0

HOST: example.com
Content-Type: application/json
Content-Length: 26

{
        login: 'user'
        password: 'pass123'
}
```

Відповідь серверу на HTTP запит в більшості схожа, але з деякими відмінностями, як показано на лістингу 3.4. Наприклад в відповіді вказується код статусу відповіді, інші заголовки, які відносяться тільки до відповіді.

Лістинг 3.4 — Шаблон HTTP відповіді

```
HTTP/1.1 200 OK

Content-Type: application/json
Connection: Closed
{
        message: 'success login'
        user:{id: 1, username: …}
}
```

Всі види семантичних методів HTTP запитів представлені в таблиці 3.1.

Таблиця 3.1 — Семантичні методи HTTP запиту

| Назва методу | Застосування |
|---|---|
| GET | використовується для запиту даних із зазначеного ресурсу |
| POST | використовується для відправки даних на сервер для створення/оновлення ресурсу |
| PUT | використовується для відправки даних на сервер для створення/оновлення ресурсу |
| HEAD | майже ідентичний GET, але без тіла відповіді |
| DELETE | видаляє вказаний ресурс |
| PATCH | використовується для застосування часткових змін до ресурсу |
| OPTIONS | описує варіанти комунікації для цільового ресурсу |
| CONNECT | використовується для встановлення двостороннього зв'язку (тунелю) із запитуваним ресурсом |
| TRACE | використовується для виконання тесту зворотного циклу повідомлення, який перевіряє шлях до цільового ресурсу (корисний для налагодження) |

GET запити мають свої переваги на відміну від інших методів, а саме можливість кешування цих запитів, що дозволить їх повторне виконання без додаткових затрачених ресурсів.

Різниця між POST і PUT полягає в тому, що PUT-запити є ідемпотентними. Це означає, що виклик одного і того ж PUT-запиту кілька разів завжди призведе до одного і того ж результату. На противагу цьому, багаторазовий виклик POST-запиту має побічний ефект створення одного і того ж ресурсу кілька разів.

Щодо HEAD запитів, то вони майже ідентичні GET, але без тіла відповіді. Іншими словами, якщо GET /users повертає список користувачів, то HEAD /users зробить той самий запит, але не поверне список користувачів.

Запити HEAD корисні для перевірки того, що поверне GET-запит, перед тим, як зробити GET-запит — наприклад, перед завантаженням великого файлу або тіла відповіді.

Заголовки HTTP запитів і відповідей важливий аспект веб-розробки і взаємодії між клієнтом і сервером в інтернеті. Вони дозволяють обмінюватися інформацією та визначати параметри запитів і відповідей. Заголовки HTTP — це метадані, які супроводжують кожен запит і відповідь. Вони містять інформацію про тип даних, код статусу, кешування, мову, кодування і багато іншого.

Основні функції заголовків HTTP включають наступне:

— параметри запиту, дозволяють клієнту вказати параметри, які він бажає отримати від сервера, наприклад, заголовок "Accept" вказує типи даних, які клієнт підтримує, і сервер може відповісти відповідним чином;

— коди статусу, вказують на результат виконання запиту, наприклад, код 200 означає успішну відповідь, тоді як код 404 вказує на те, що запитана сторінка не знайдена;

— кешування, дозволяють керувати кешуванням даних, це дозволяє заощадити розмір бандлу і прискорити завантаження сторінок, які вже були відвідані користувачем;

— кодування та стиснення, вказують на необхідність стиснення або кодування даних для зменшення обсягу передачі даних і збільшення швидкості завантаження;

— мова і локалізація, включають інформацію про мову та локалізацію, що дозволяє серверу надсилати відповіді в певній мові або для певної регіональної аудиторії.

Заголовки HTTP дуже важливі для забезпечення правильної взаємодії між клієнтом і сервером. Вони допомагають забезпечити безпеку, ефективність і правильну інтерпретацію даних. Крім того, розуміння ролі заголовків HTTP є важливим аспектом веб-розробки та оптимізації продуктів для кінцевих користувачів.

На практиці використовуються різні засоби, такі як браузерні інструменти розробника або HTTP бібліотеки для мов програмування, щоб додавати і зчитувати заголовки у своїх HTTP запитах і відповідях. Заголовки HTTP дозволяють більш гнучко налаштовувати та контролювати взаємодію з серверами.

Далі описані кілька найпопулярніших заголовків, які використовуються в розробці:

— User-Agent, містить інформацію про веб-браузер або клієнтський додаток, який робить запит на сервер, він допомагає серверу зрозуміти як оптимально відповісти, орієнтуючись на особливості клієнта;

— Accept, вказує які типи контенту (медіа, мови, формати) клієнт готовий приймати від серверу;

— Content-Type, вказує на тип вмісту, що міститься в запиті до сервері чи в відповіді сервера, допомагаючи правильно інтерпретувати дані;

— Authorization, надає інформацію про ідентифікацію та авторизацію користувача, що дозволяє доступ до обмежених ресурсів на сервері зазвичай записаний в вигляді “Bearer token”, де на місці токена знаходиться JWT докен;

— Location, вказує на нову локацію, до якої клієнт повинен бути перенаправлений після виконання певної операції;

— Cache-Control, вказує браузеру або проксі-серверам, як довго можна зберігати вміст в кеші перед завантаженням нових даних з сервера;

— Cookie, містить інформацію про раніше встановлені cookies.

Коди статусу в HTTP відповіді допомагають клієнту уточнити відповідь сервера і слугує інформацією про результат відповіді, наприклад, про успішне виконання чи помилку.

Їх можна поділити на 5 основних груп:

— інформаційні, від 100 до199;

— успіх, від 200 до 299;

— перенаправлення, від 300 до 399;

— помилки клієнта, від 400 до 499;

— помилки сервера, від 500 до 599.

Статус кодів є доволі багато, але серед них можна виділити найбільш поширені, див. таблицю 3.2.

Таблиця 3.2 — Найбільш поширені коди HTTP відповіді

| HTTP код | Назва коду | Опис коду |
|---|---|---|
| 100 | Continue | Ця проміжна відповідь вказує на те, що клієнт повинен продовжити запит або проігнорувати відповідь, якщо запит вже завершено. |
| 200 | OK | Сервер успішно відповів на запит користувача, і вміст відповіді знаходиться в тілі відповіді. |
| 301 | Moved Permanently | URL-адресу запитуваного ресурсу було змінено назавжди. Нова URL-адреса буде вказана у відповіді. |
| 302 | Found | Цей код відповіді означає, що URI запитуваного ресурсу було тимчасово змінено. У майбутньому можуть бути внесені подальші зміни в URI. Тому в наступних запитах клієнт повинен використовувати цей самий URI. |
| 400 | Bad Request | Сервер не може або не хоче обробляти запит через те, що сприймається як помилка клієнта (наприклад, неправильний синтаксис запиту, невірне обрамлення повідомлення запиту або помилкова маршрутизація запиту). |
| 401 | Unauthorized | Хоча в стандарті HTTP вказано "неавторизований", семантично ця відповідь означає "неавтентифікований". Тобто клієнт повинен пройти автентифікацію, щоб отримати запитувану відповідь. |
| 403 | Forbidden | Клієнт не має прав доступу до контенту, тобто є неавторизованим, тому сервер відмовляє у наданні запитуваного ресурсу. На відміну від 401 Unauthorized, ідентифікатор клієнта відомий серверу. |

Продовження таблиці 3.2

| | | |
|---|---|---|
| 404 | Not Found | Сервер не може знайти запитуваний ресурс. У браузері це означає, що URL-адресу не розпізнано. В API це також може означати, що кінцева точка є дійсною, але самого ресурсу не існує. |
| 500 | Internal Server Error | Сервер зіткнувся з ситуацією, з якою не знає, як впоратися. |
| 503 | Service Unavailable | Сервер не готовий обробити запит. |

З описаних методів HTTP запитів в розробці використовуються найбільш популярні серед них, а саме це GET запити та POST запити. В деяких випадках використаний метод DELETE, що є семантично вірним в конкретному контексті. За допомогою описаних кодів статусу описуються відповідна реакція веб-інтерфейсу на конкретні випадки. У випадку статусів помилки — виведення помилки.

## 3.5 Розробка інтерфейсу для формування запиту до штучного інтелекту

Інтерфейс для формування запиту до штучного інтелекту є ключовим в контексті створеного веб-застосунку. Користувач повинен заповнити всі необхідні дані для подальшої генерації дієти за допомогою штучного інтелекту. Форма складається з двох частин: персональної інформації з обов’язковими полями, такими як вік користувача, його зріст, актуальна вага та рівень активності, та допоміжної частини з необов’язковими полями.

Всі необхідні поля для заповнення або вибору представлені в зручному та інтуїтивно зрозумілому інтерфейсі. В даній формі всі поля являються обов’язковими для заповнення та у випадку, якщо користувач залишить якесь поле пустим та натисне кнопку «Далі», буде виведена помилка про пусті обов’язкові поля. Кожне поле являється ключовим для створення

персональної дієти, тому при заповненні всіх даних, результат буде орієнтованим на конкретного користувача.

Необов’язкові поля для заповнення допоможуть штучному інтелекту створити найбільш персоналізовану дієту, з включенням вподобань користувача, його алергій та виключеної їжі з раціону.

Виведення помилок реалізовано в залежності від відповіді серверної частини на конкретний запит. Помилка буде відрізнятись в залежності від наповнення форми.

Також комфортним для користувача рішенням було зберегти введені дані в базі даних з прив’язкою до конкретного акаунта, що дало можливість при повторному завантаженні сторінки з формами — відразу заповнити поля з можливістю зміни інформації на нову.

## 3.6 Аналіз трафіку та швидкості завантаження сторінки

Перше завантаження сторінки повинне завжди бути настільки швидким, наскільки цього можна досягти в кожному конкретному випадку.

Швидкість завантаження застосунку була перевірена за допомогою сервісу Google PageSpeed Insights [20], результати представлені на рис. 3.4 та рис. 3.5.

PageSpeed Insights (PSI) звітує про користувацький досвід сторінки як на мобільних, так і на десктопних пристроях, а також надає пропозиції щодо покращення цієї сторінки. PSI надає як лабораторні, так і польові дані про сторінку. Лабораторні дані корисні для налагодження проблем, оскільки вони збираються в контрольованому середовищі. Однак вони можуть не відображати реальні проблеми. Польові дані корисні для фіксації справжнього, реального користувацького досвіду, але мають більш обмежений набір метрик.

З використанням сервісу PageSpeed Insights можна значно покращити швидкість завантаження сторінок та швидкодію веб-застосунку в цілому.

Будь-яка зелена оцінка (90+) вважається хорошим результатом, але знову ж таки ця оцінка не показує реальний досвід користування веб-застосунком, хоча і допомагає в його покращенні.

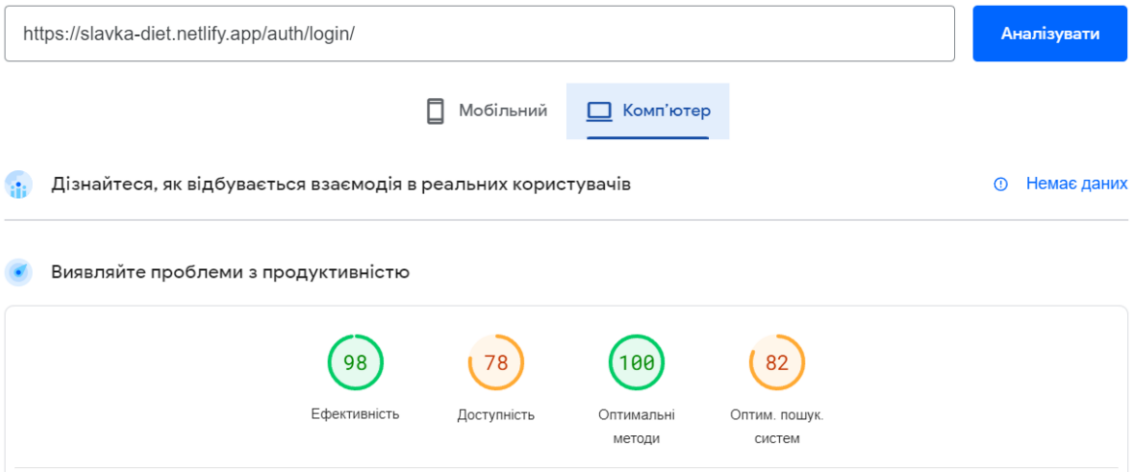


Рисунок 3.4 — Тестування швидкості завантаження на комп'ютері

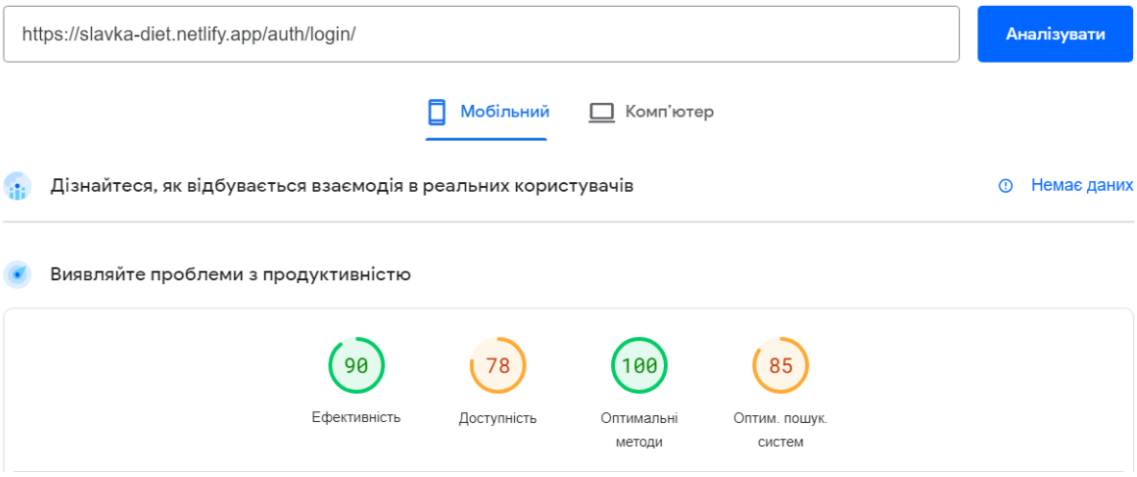


Рисунок 3.5 — Тестування швидкості завантаження на телефоні

Ефективність (швидкість) завантаження знаходиться в зеленій зоні, що відзначає дійсно високу швидкість першого завантаження, що сприяє чудовому UX. Таких значень досягнуто завдяки вибраному фреймворку для клієнтської частини та оптимізації великих картинок, стилів та коду в цілому.

Для комп'ютера при тестуванні було симульовано використання швидкості інтернету 40 мс TCP RTT, 10 240 кб/с, що являється мінімальною-

середньою швидкістю для більшості провайдерів. З такою швидкістю інтернету результат завантаження майже моментальний, та складає менше секунди реального часу.

У випадку з телефоном було симульовано швидкість 150 мс TCP RTT, 1 638,4 кб/с, що дорівнює 4G з низькою швидкістю. В цьому випадку швидкість завантаження склала більше двох секунд, що для мобільного інтернету являється також досить швидким результатом.

Тестування виконано для перших завантажень сторінок, а це означає, що сторінки та ресурси будуть кешуватись в подальших завантаженнях і швидкість відкриття веб-сайту збільшиться ще більше.

Окрім швидкості інтернету, для запуску застосунку зі сторони користувача немає високих системних вимог. Вимоги можна позначити тими, що дозволяють використовувати браузер. Тобто звичайної системи на телефонах або ж Windows/MacOS на комп’ютері буде цілком достатньо. Якщо комп’ютер підтримує встановлення Windows версії 10 чи 11, це вже є достатньою системою для користування застосунком.

Щодо хостингу веб-застосунку можна використати безкоштовний хостинг від сервісу Netlify, що представляє зручний інтерфейс завантаження нового застосунку, налаштування DNS. Також є можливість підключити моніторинг GitHub репозиторію та оновлювати застосунок автоматично в результаті нових оновлень в гілці main.

Для хостингу веб-застосунку з організації дієти та тренувань необхідно мати сервер з мінімальними технічними характеристиками, а саме це 10ГБ пам’яті, 2ГБ оперативної пам’яті, та процесор з одним ядром. Такі технічні характеристики представляє майже кожен хостинг провайдер за мінімальну ціну або ж безкоштовно, як у випадку з Netlify.

# 4 РОЗРОБКА UI/UX ДИЗАЙНУ ДЛЯ ВЕБ-ЗАСТОСУНКУ ОРГАНІЗАЦІЇ ТРЕНУВАНЬ ТА ДІЄТИ

## 4.1 Обґрунтування вибору інструментів для розробки UI/UX дизайну

Крім швидкодії сайту, наповнення його контентом та функціональними можливостями, які він виконує, не менш важливим фактором в сучасному світі є його дизайн. Щоб користувачам було зручно користуватись веб-додатком, повинні дотримуватись принципи UI та UX. Крім того, є різні застосунки для створення дизайну, чи його прототипу: Figma, Adobe Photoshop, Adobe XD та ін.

Додаток Adobe Photoshop — це графічний редактор, який є найпопулярнішим додатком для редагування растрових зображень, цифрового малювання. Велика кількість самих різноманітних інструментів дозволяє використовувати цей інструмент не тільки для редагування фотографій чи малювання, а і для створення макетів веб-сайтів.

Інтерфейс самого редактору є достатньо складним і різноманітним. Враховуючи велику наповненість функціями різного спрямування, цей застосунок використовує багато ресурсів комп’ютера, що є недоліком при виборі інструменту для створення дизайну.

Популярність Adobe Photoshop на ринку дизайну сайтів та мобільних додатків впала після випуску сучасніших інструментів для розробки дизайну.

Одним з таких інструментів є Figma — векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи та завантаження додатку на комп’ютер користувача. Принциповою різницею між Adobe Photoshop та Figma є направленість самого застосунку під конкретні задачі. В випадку Figma, це розробка додатків та веб-інтерфейсів.

Зручність полягає не тільки в створенні дизайну, але і в використанні цього дизайну уже в процесі розробки, що відсутнє в Adobe Photoshop. Також відносно мала кількість необхідних ресурсів комп’ютера для використання Figma впливає на швидкість розробки.

Також можливість створення анімацій прямо в дизайні, або ж повноформатний перегляд макету є ще одними перевагами онлайн-сервісу Figma.

При виборі інструменту для розробки дизайну були проаналізовані та випробувані ще такі додатки як Sketch та Adobe XD.

Перевагами Sketch можна зазначити його швидкодію, так як додаток працює на вбудованих засобах операційної системи, і зручність використання, що забезпечує високу продуктивність праці. Недоліками є можливість використання даного додатку лише на Mac OS та відсутність безкоштовної версії програми (тільки пробний період).

Додаток Adobe XD теж було порівняно з вищезазначеними варіантами. Суттєвою його перевагою є можливість створення та зручного керування станами компонентів дизайну, що допомагає показати розробнику необхідні анімації. Недоліками виступає відносно невеликий інструментарій по роботі з текстом та необхідність платної підписки для роботи [21].

Порівняння розглянутих інструментів для розробки макетів веб-інтерфейсів наведено таблиці 4.1.

Таблиця 4.1 — Порівняльна таблиця додатків для створення дизайну

| Назва інструменту | Використання ресурсів | Швидкість відповіді програми | Інтерфейс | Вартість |
|---|---|---|---|---|
| Figma | ~ 2GB+ оперативної пам’яті | Декілька мілісекунд | Інтуїтивний | Безкоштовно для одного користувача |
| Photoshop | 70%+ доступної оперативної пам’яті | Довга застримка | Складний | Підписка, близько 240$ на рік |

Продовження таблиці 4.1

| | | | | |
|---|---|---|---|---|
| Sketch | 4 GB пам’яті на диску;<br>2+ GB оперативної пам’яті | З затримкою | Інтуїтивний | Від 99$ перший рік, потім по 79$ за рік (пробний період 30 днів) |
| Adobe XD | 2+ GB пам’яті на диску;<br>4 GB оперативної пам’яті | Помірна продуктивність | Інтуїтивний | Безкоштовний стартовий пакет, 120$ підписка |

Отже, для розробки макету веб-застосунку для організації планування дієти прийнято рішення про використання інструменту Figma завдяки простоті вивчення, можливості колективної праці над макетом та наявності потрібних інструментів для дизайну.

## 4.2 Вибір палітри кольорів веб-застосунку

Колірна палітра веб-застосунку, що спрямований на здорове харчування та фізичну активність, має вирішальне значення для забезпечення комфортного користувацького досвіду. Кольори впливають на емоційний стан користувача, його сприйняття інформації та бажання взаємодіяти з продуктом.

Під час аналізування можливих варіантів кольорів для інтерфейсу веб-застосунку важливо враховувати психологічні аспекти та тематику веб-застосунку.

Також створення контрастних комбінацій кольорів є важливим для визначення ключових елементів веб-сайту. Правильне сполучення кольорів допомагає покращити читабельність, робить інформацію більш доступною та легкозасвоюваною для користувачів.

Важливо враховувати специфіку веб-застосунку при виборі кольорів. Наприклад, якщо продукт спрямований на здоровий спосіб життя, використання світлих та енергійних кольорів може підкреслити цінності та

мету застосунку. Також, важливо враховувати корпоративний стиль та брендові кольори, які відображають ідентичність застосунку.

Після вибору потенційних кольорів важливо провести тестування з маленькою аудиторією, щоб зрозуміти, як вони сприймають обрану палітру. Зворотний зв'язок допоможе виявити можливі недоліки та зробити необхідні коригування.

У зв’язку з цим була створена колірна палітра, ключовими кольорами в якій стали легкі в сприйнятті фіалковий колір та різні відтінки сірого та червоних кольорів (для контрасту), як показано на рис. 4.1.

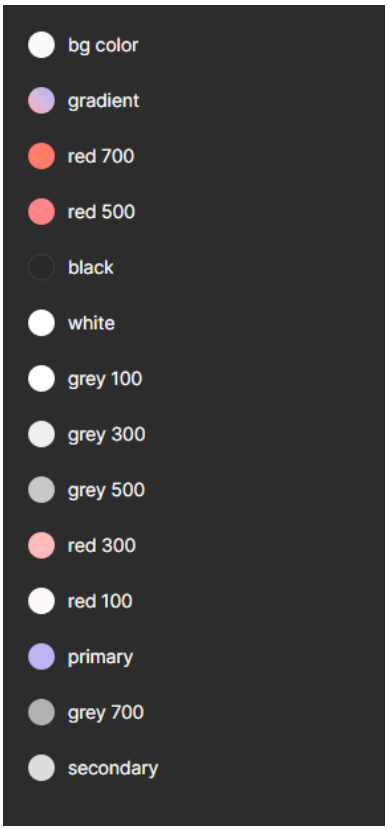


Рисунок 4.1 — Палітра вибраних кольорів для розробки дизайну

З рисунку видно, що всі кольори організовані як змінні, які можна легко перевикористовувати, що значно прискорює процес розробки дизайну та полегшує використання кольорів при безпосередній розробці клієнтської частини веб-застосунку.

## 4.3 Створення набору інтерфейсів користувача UI-KIT з необхідними елементами

UI-KIT (або набір інтерфейсів користувача) — це збірка готових елементів дизайну, таких як кнопки, поля вводу, панелі і т.д., які створені для спрощення процесу створення і забезпечення консистентності (цілісності) інтерфейсу веб-застосунків (рис. 4.2).

UI-KIT відіграє ключову роль у забезпеченні консистентності дизайну. Він дозволяє працювати з єдиною базою елементів, забезпечуючи однорідний вигляд та взаємодію всіх елементів веб-застосунку. Це сприяє покращенню користувацького досвіду та спрощує процес розробки.

Також UI-KIT дозволяє значно зекономити час, оскільки не потрібно створювати елементи інтерфейсу "з нуля" для кожної сторінки чи екрану. Це дозволяє фокусуватися на більш важливих аспектах дизайну та функціональності веб-застосунку.

Управлінням UI-KIT можна легко забезпечити стандартизацію дизайну в усьому веб-застосунку. Це дозволяє підтримувати єдиний стиль та забезпечує легкість у внесенні змін або оновлень у дизайні.

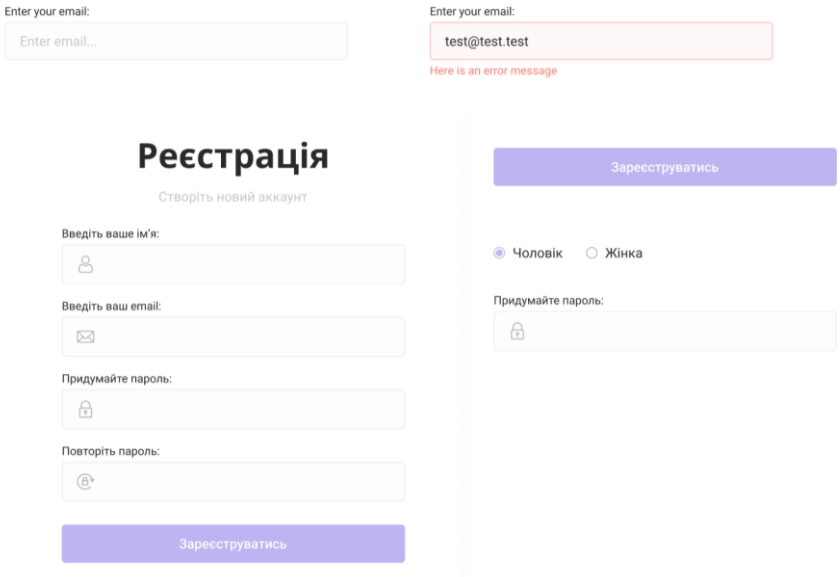


Рисунок 4.2 — Вигляд UI-KIT, що був розроблений в Figma

На рисунку можна побачити, що були створені такі компоненти UI-KIT, як: поля вводу, радіо кнопки, звичайні кнопки та інші елементи дизайну. Це дозволяє зберігати всі основні компоненти в одному місці та, при необхідності, замінити стилі лише одного компонента, що замінить стилі і всіх інших в дизайні та значно прискорить час розробки.

## 4.4 Розробка дизайну сторінок авторизації

В процесі розробки дизайну були створені сторінки авторизації (рис. 4.3), які відразу дають зрозуміти тематику веб-застосунку.

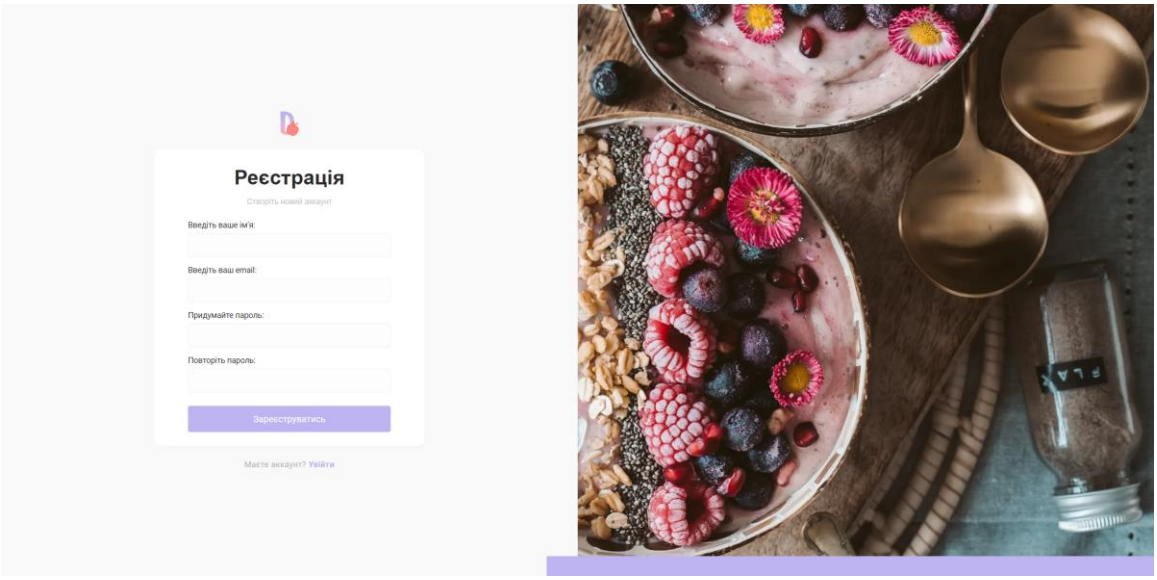


Рисунок 4.3 — Дизайн сторінки реєстрації

При створені сторінок авторизації використовувались приклади сучасного дизайну, що перевірені на практиці та вважаються найбільш зручними та інтуїтивно зрозумілими, що являється основним аспектом UX дизайну.

Також використовувались кольора з палітри кольорів, розробленої раніше та різні компоненти з UI-KIT.

Всі сторінки розроблялись зі стандартною шириною для дизайну — 1440 пікселів та висотою, що залежить від наповнення сторінки. Також

використовувались внутрішні інструменти Figma, такі як сітка чи автоматичний макет, що робить дизайн більш адаптивним до різних коригувань з контентом та розмірами.

## 4.5 Розробка дизайну основних сторінок

При розробці основних сторінок дизайну веб-застосунку також застосовувалась палітра кольорів та UI-KIT компоненти, так як дизайн повинен відповідати одному стилю на всіх сторінках веб-застосунку.

На основних сторінках дизайну присутні шапка та нижній блок з навігацією, так як це додає необхідну навігацію по всіх сторінках.

Також використані ілюстративні елементи в кутку основних форм для генерації дієти та тренувань (рис. 4.4) для відповідності тематиці веб-застосунку та покращення UI.

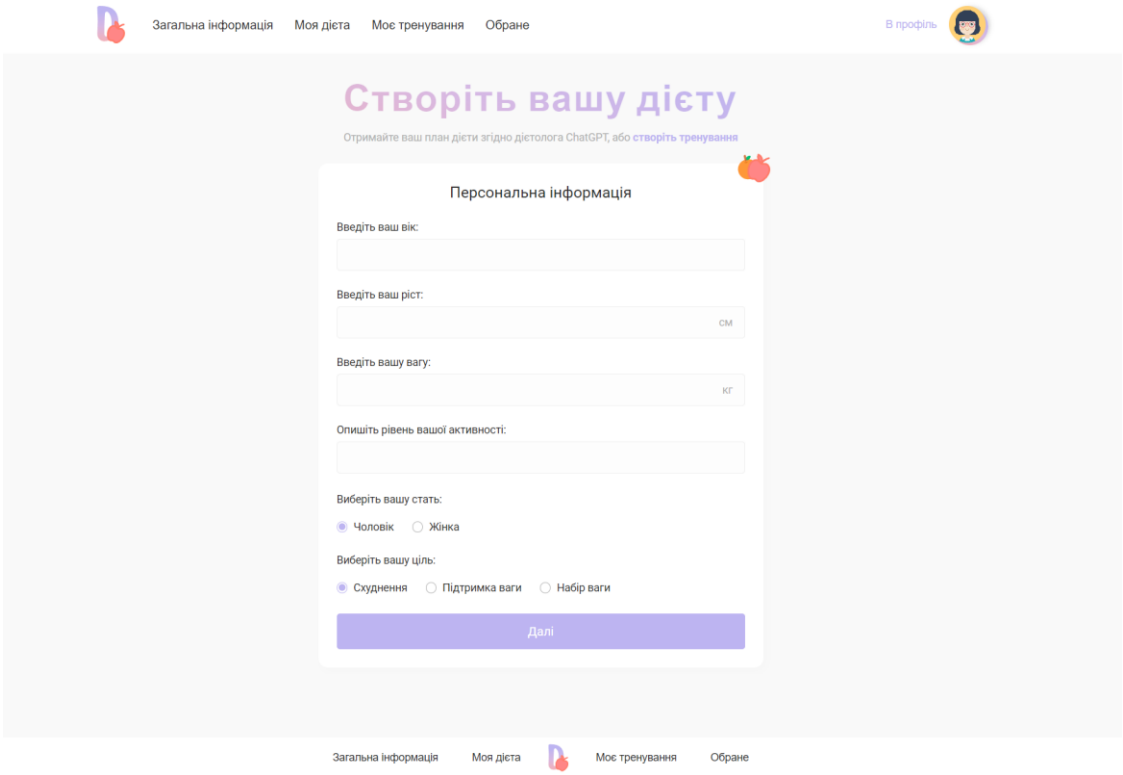


Рисунок 4.4 — Дизайн сторінки створення персональної дієти

На текстових сторінках (рис. 4.5) використаний зручний розмір шрифта, відстань між рядками та ширина контенту, щоб досягти можливості найбільш зручного читання без необхідності збільшувати чи зменшувати текст.

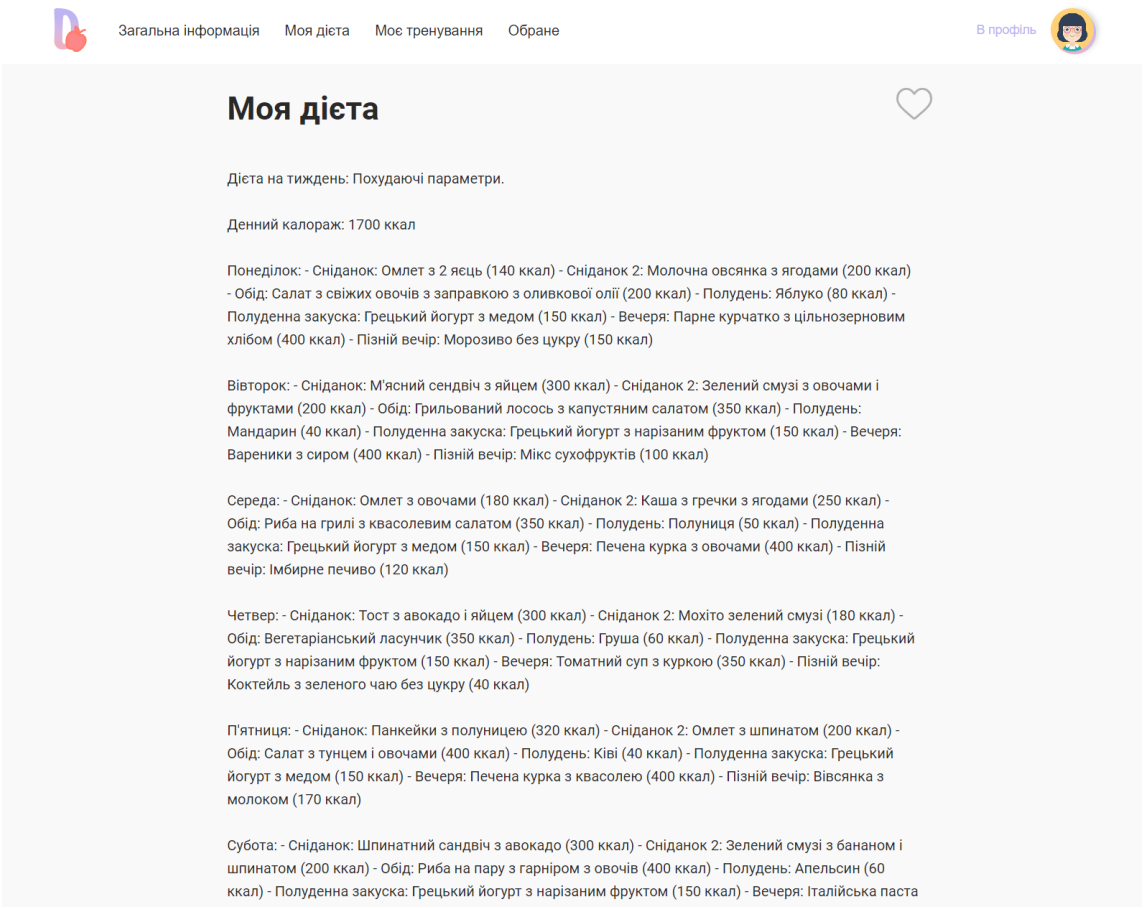


Рисунок 4.5 — Дизайн сторінки перегляду згенерованої дієти

Всі інші сторінки створені на основі представлених прикладів. Досягнуто основної задачі дизайну, а саме показати легкий та комфортний в користуванні веб-застосунок, підібрати всі необхідні кольори для відповідності тематиці та контрасту, і забезпечити можливість розробнику розробити застосунок на основі представленого дизайну.

## 4.6 Вимоги до апаратного забезпечення для розробки інтерфейсу веб-застосунку

Figma — це потужне програмне забезпечення, яке можна використовувати різними способами. Його легка хмарна природа дозволяє отримати до нього доступ практично з будь-якого пристрою.

Для комфортної роботи в Figma потрібен хороший процесор разом з достатнім об'ємом оперативної пам'яті. Мінімумом буде чотирьохядерний процесор з чотирма потоками, але бажано мати ще більше потужності.

Зазвичай, варто шукати процесори з високою продуктивністю одного ядра, а не високою багатоядерністю, щоб отримати найкращі враження при розробці дизайну.

Щодо відеокарти, Figma використовує WebGL (бібліотека веб-графіки) для обробки рендерингу. Це означає, що в нього дуже низькі вимоги до графіки — ви отримаєте відмінну продуктивність навіть на помірно потужній вбудованій графіці та навіть на відділеному графічному процесорі.

Є лише одна вбудована графіка, яка викликає проблеми та збої при рендерингу, це Intel HD Graphics 3000 у зв’язку з внутрішньою компоновкою.

Для комфортної розробки дизайну повинно бути хоча б 4ГБ оперативної пам’яті, але для більш комплексних проектів потрібно мати від 8ГБ мінімум.

В таблиці 4.2 представлені рекомендації для збірки ПК при роботі в Figma.

Таблиця 4.2 — рекомендації до збірки ПК для Figma

| Компонент ПК | Мінімальна збірка | Комфортна збірка | Найкраща збірка |
|---|---|---|---|
| Процесор | AMD Ryzen 3 3200G | Intel Core i5 12400 | AMD Ryzen 5 5600X |
| Відеокарта | AMD Radeon RX 560 4GB XFX | NVIDIA RTX 3050 8GB | NVIDIA 3060TI 8GB |
| Оперативна пам’ять | 4GB DDR4 3200 | 8GB DDR4 3200 | 16GB DDR4 3600 |

# 5 ЕКОНОМІЧНА ЧАСТИНА

Ефективне впровадження науково-технічної розробки стає можливим, якщо вона відповідає поточним вимогам науково-технічного прогресу та враховує економічні аспекти. Надання оцінки економічної ефективності отриманих результатів науково-дослідної роботи є важливою частиною цього процесу.

Комплексна магістерська кваліфікаційна робота, що присвячена розробці та дослідженню Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн», віднесена до науково-технічних робіт, спрямованих на введення на ринок. Рішення про комерціалізацію розробки може бути прийняте протягом самої роботи, дозволяючи реалізувати можливість виведення її на ринок. Цей напрямок розглядається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Однак для успішного втілення цього процесу важливо знайти зацікавленого інвестора, який був би зацікавлений у реалізації цього проекту, і переконати його в обґрунтованості таких інвестицій.

Для цього визначені наступні етапи виконання робіт:

1) проведено комерційний аудит науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу;

2) розраховані витрати на реалізацію науково-технічної розробки;

3) проведено розрахунок економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтовано економічну доцільність комерціалізації для інвестора.

## 5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1

Таблиця 5.1 — Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

| Бали (за 5-ти бальною шкалою) | | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| Технічна здійсненність концепції | | | | | |
| 1 | Достовірність концепції не підтверджена | Концепція підтверджена експертними висновками | Концепція підтверджена розрахунками | Концепція перевірена на практиці | Перевірено працездатність продукту в реальних умовах |
| Ринкові переваги (недоліки) | | | | | |
| 2 | Багато аналогів на малому ринку | Мало аналогів на малому ринку | Кілька аналогів на великому ринку | Один аналог на великому ринку | Продукт не має аналогів на великому ринку |
| 3 | Ціна продукту значно вища за ціни аналогів | Ціна продукту дещо вища за ціни аналогів | Ціна продукту приблизно дорівнює цінам аналогів | Ціна продукту дещо нижче за ціни аналогів | Ціна продукту значно нижче за ціни аналогів |
| 4 | Технічні та споживчі властивості продукту значно гірші, ніж в | Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів | Технічні та споживчі властивості продукту на рівні аналогів | Технічні та споживчі властивості продукту трохи кращі, ніж в | Технічні та споживчі властивості продукту значно кращі, ніж в |
| 5 | Експлуатаційні витрати значно вищі, ніж в аналогів | Експлуатаційні витрати дещо вищі, ніж в аналогів | Експлуатаційні витрати на рівні експлуатаційних витрат аналогів | Експлуатаційні витрати трохи нижчі, ніж в аналогів | Експлуатаційні витрати значно нижчі, ніж в аналогів |
| Ринкові перспективи | | | | | |

Продовження таблиці 5.1

| | | | | | |
|---|---|---|---|---|---|
| 6 | Ринок малий і не має позитивної динаміки | Ринок малий, але має позитивну динаміку | Середній ринок з позитивною динамікою | Великий стабільний ринок | Великий ринок з позитивною динамікою |
| 7 | Активна конкуренція великих компаній на | Активна конкуренція | Помірна конкуренція | Незначна конкуренція | Конкурентів немає |
| Практична здійсненність | | | | | |
| 8 | Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї | Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців | Необхідне незначне навчання фахівців та збільшення їх штату | Необхідне незначне навчання фахівців | Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї |
| 9 | Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні | Потрібні незначні фінансові ресурси. Джерела фінансування відсутні | Потрібні значні фінансові ресурси. Джерела фінансування є | Потрібні незначні фінансові ресурси. Джерела фінансування є | Не потребує додаткового фінансування |
| 10 | Необхідна розробка нових матеріалів | Потрібні матеріали, що використовують ся у військово промисловому комплексі | Потрібні дорогі матеріали | Потрібні досяжні та дешеві матеріали | Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві |
| 11 | Термін реалізації ідеї більший за 10 років | Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років | Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років |
| 12 | Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту | Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу | Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу | Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту | Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту |

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 5.2. Для опитування було залучені експерти: к.пед.н. доц. кафедри Войцеховська О.В., к.пед.н., доц. Добровольська Н.В., к.т.н, доц. Тарновський М.Г.

Таблиця 5.2 — Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

| Критерії | Експерт (ПІБ, посада) | | |
|---|---|---|---|
| | Войцеховська О.В., к.т.н, доц. | Добровольська Н.В., к.т.н, доц. | Тарновський М.Г., к.т.н, доц. |
| | Бали: | | |
| 1. Технічна здійсненність концепції | 4 | 4 | 4 |
| 2. Ринкові переваги (наявність аналогів) | 3 | 3 | 4 |
| 3. Ринкові переваги (ціна продукту) | 4 | 4 | 4 |
| 4. Ринкові переваги (технічні властивості) | 4 | 4 | 4 |
| 5. Ринкові переваги (експлуатаційні витрати) | 4 | 3 | 3 |
| 6. Ринкові перспективи (розмір ринку) | 3 | 3 | 3 |
| 7. Ринкові перспективи (конкуренція) | 3 | 3 | 4 |
| 8. Практична здійсненність (наявність фахівців) | 4 | 4 | 4 |
| 9. Практична здійсненність (наявність фінансів) | 3 | 3 | 3 |
| 10. Практична здійсненність (необхідність нових матеріалів) | 4 | 4 | 4 |
| 11. Практична здійсненність (термін реалізації) | 4 | 4 | 4 |
| 12. Практична здійсненність (розробка документів) | 3 | 3 | 3 |

Продовження таблиці 5.2

| Сума балів | $СБ_1$=43 | $СБ_2$=42 | $СБ_3$=44 |
|---|---|---|---|
| Середньоарифметична сума балів $СБ_с$ | (43+42+44)/3 = 43 | | |

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3

Таблиця 5.3 — Науково-технічні рівні та комерційні потенціали розробки

| Середньоарифметична сума балів СБ , розрахована на основі висновків експертів | Науково-технічний рівень та комерційний потенціал розробки |
|---|---|
| 41…48 | Високий |
| 31…40 | Вище середнього |
| 21…30 | Середній |
| 11…20 | Нижче середнього |
| 0…10 | Низький |

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» становить 43 бали, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий.

Результатом комплексної магістерської кваліфікаційної роботи Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» — веб-застосунок для надання користувачам дієт та програм тренувань під їх індивідуальні потреби.

## 5.2 Визначення рівня конкурентоспроможності розробки

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Одиничний параметричний індекс розраховуємо за формулою:

$$q_i = \frac{P_i}{P_{базі}}, \tag{5.1}$$

де $q_i$ — одиничний параметричний індекс, розрахований за $i$-м параметром;

$P_i$ — значення $i$-го параметра виробу;

$P_{базі}$ — аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Загальні технічні та економічні характеристики розробки представлено в таблиці 5.4.

Таблиця 5.4 — Основні техніко-економічні показники аналога та розробки, що проектується

| Показник | Варіанти | | Відносний показник якості | Коефіцієнт вагомості параметра |
|---|---|---|---|---|
| | Базовий (товар-конкурент) | Новий (інноваційне рішення) | | |
| 1 | 2 | 3 | 4 | 5 |
| Швидкодія, % | 90 | 94 | 1,02 | 20 |
| SEO-оптимізація, % | 92 | 95 | 1,03 | 30 |
| SSL-шифрування, % | 96 | 93 | 1,03 | 30 |
| Адаптивність, % | 90 | 95 | 1,06 | 20 |

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 — пристрій відповідає нормам і стандартам; 0 — не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховуємо як добуток частинних показників за кожним параметром за формулою:

$$I_{НП} = \prod_{i=1}^{n} q_i \ , \qquad (5.2)$$

де $I_{нп}$ — загальний показник конкурентоспроможності за нормативними параметрами;

$q_i$ — одиничний (частинний) показник за $i$-м нормативним параметром;

$n$ — кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \ , \qquad (5.3)$$

де $I_{тп}$ — груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

$q_i$ — одиничний параметричний показник $i$-го параметра;

$\alpha_i$ — вагомість $i$-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

$n$ — кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 5.4.

$$I_{тп} = 1{,}02 \cdot 0{,}2 + 1{,}03 \cdot 0{,}3 + 1{,}03 \cdot 0{,}3 + 1{,}06 \cdot 0{,}2 = 0{,}93.$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i \ , \qquad (5.4)$$

де $I_{ЕП}$ — груповий параметричний індекс за економічними показниками;

$q_i$ — економічний параметр $i$-го виду;

$\beta_i$ — частка $i$-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

$m$ — кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$$I_{ЕП}=0,76 \cdot 0,5 + 0,84 \cdot 0,5 = 0,80.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}}, \quad (5.5)$$

$$К_{ІНТ} = 1 \cdot 0,93 / 0,80 = 1,16.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

## 5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

### 5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам,

технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці [22].

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (5.6)$$

де $k$ — кількість посад дослідників залучених до процесу досліджень;

$М_{ni}$ — місячний посадовий оклад конкретного дослідника, грн;

$t_i$ — число днів роботи конкретного дослідника, дн.;

$Т_р$ — середнє число робочих днів в місяці, $Т_р$=21 дні.

$$З_о = 42000 \cdot 10 / 21 = 19091 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.5.

Таблиця 5.5 — Витрати на заробітну плату дослідників

| Найменування посади | Місячний посадовий оклад, грн | Оплата за робочий день, грн | Число днів роботи | Витрати на заробітну плату, грн |
|---|---|---|---|---|
| Керівник проекту | 42000 | 1909,1 | 10 | 19091 |
| Інженер-програміст | 39000 | 1772,7 | 55 | 97500 |
| Всього | | | | 116591 |

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» розраховуємо за формулою:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{5.7}$$

де $C_i$ — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_i$ — час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $C_i$ можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{5.8}$$

де $M_M$ — розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

$K_i$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$K_c$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_p$ — середнє число робочих днів в місяці, приблизно $T_p$ = 21 дн;

$t_{зм}$ — тривалість зміни, год.

$$C_1 = 6500{,}00 \cdot 1 \cdot 1{,}65 / (21 \cdot 8) = 65{,}8 \text{ грн.}$$

$$З_{р1} = 65{,}8 \cdot 2 = 131{,}6 \text{ грн.}$$

Результати приведено в таблиці 5.6

Таблиця 5.6 — Величина витрат на основну заробітну плату робітників

| Найменування робіт | Тривалість роботи, год | Розряд роботи | Погодинна тарифна ставка, грн | Величина оплати на робітника грн |
|---|---|---|---|---|
| 1.Підготовчі | 2 | 1 | 65,8 | 131,6 |
| 2.Налагоджувальні | 10 | 2 | 72,4 | 723,8 |

Продовження таблиці 5.6

| | | | | |
|---|---|---|---|---|
| 3.Випробувальні | 2 | 4 | 98,7 | 197,4 |
| Всього | | | | 1052,9 |

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.9)$$

де $Н_{дод}$ — норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (116591 + 1052,9) \cdot 11 / 100\% = 12940,81 \text{ грн.}$$

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.10)$$

де $Н_{зп}$ — норма нарахування на заробітну плату. Приймаємо 22%.

$$Зн = (116591 + 1052,9 + 12940,81) \cdot 22 / 100\% = 28728,61 \text{ грн.}$$

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн».

Витрати на матеріали (*М*) (таблиця 5.7), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj} , \quad (5.11)$$

де $H_j$ — норма витрат матеріалу *j*-го найменування, кг;

$n$ — кількість видів матеріалів;

$Ц_j$ — вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ — коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$B_j$ — маса відходів *j*-го найменування, кг;

$Ц_{вj}$ — вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.7 — Витрати на матеріали

| Найменування матеріалу, марка, тип, сорт | Ціна за 1 кг, грн | Норма витрат, кг | Вартість витраченого матеріалу, грн |
|---|---|---|---|
| Папір А 4 | 146 | 1 | 146 |
| Ручка | 14 | 1 | 14 |
| Диск оптичний OPTIMA CD | 15 | 1 | 15 |
| Flesh-пам'ять GOODRAM 64 C10A | 410 | 1 | 410 |
| Всього | | | 585 |
| З врахуванням коефіцієнта транспортування | | | 643,5 |

### 5.3.4 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} , \quad (5.12)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = (45000 \cdot 1) / (2 \cdot 12) = 1875 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8– Амортизаційні відрахування по кожному виду обладнання

| Найменування обладнання | Балансова вартість, грн | Строк корисного використання, років | Термін використання обладнання, місяців | Амортизаційні відрахування, грн |
|---|---|---|---|---|
| Комп'ютер | 45000 | 2 | 1 | 1875,00 |
| Приміщення лабораторії | 190000 | 20 | 1 | 791,67 |
| Всього | | | | 2666,67 |

5.3.5 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_e$) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i}, \quad (5.13)$$

де $W_{yi}$ — встановлена потужність обладнання на визначеному етапі розробки, кВт;

$t_i$ — тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,5 грн;

$K_{впі}$ — коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

$\eta_i$ — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$$В_е = 0,25 \cdot 250,0 \cdot 7,5 \cdot 0,5 / 0,8 = 292,97 \text{ грн.}$$

### 5.3.6 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \quad (5.14)$$

де $H_{св}$ — норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$$В_{св} = (116591 + 1052,9) \cdot 20 / 100\% = 23528,75 \text{ грн.}$$

### 5.3.7 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{H_{ів}}{100\%}, \quad (5.15)$$

де $H_{ів}$ — норма нарахування за статтею «Інші витрати», приймемо $Н_{ів} = 50\%$.

$$І_в = (116591 + 1052,9) \cdot 50 / 100\% = 58821,88 \text{ грн.}$$

### 5.3.8 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.16)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв} = 100\%$.

$$В_{нзв} = (116591 + 1052,9) \cdot 100 / 100\% = 117643,77 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.17)$$

$$В_{заг}=116591+1052,9+12940,81+28728,61+643,5+2666,67+292,97+23528,75+ 58821,88+117643,77= 363004,48 \text{ грн.}$$

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.18)$$

де $\eta$ — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо $\eta$=0,9.

ЗВ = 363004,48 / 0,9 = 403338,31 грн.

## 5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

$\Delta N$ — збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

$N$ – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ — вартість послуги у році до впровадження інформаційної системи, приймемо 2000,00 грн;

$\pm\Delta Ц_о$ — зміна вартості послуги від впровадження результатів, приймемо зростання на 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.19)$$

Де $\lambda$ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

$\rho$ — коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 40\%$;

$\vartheta$ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (1 \cdot 500 + 2000 \cdot 1500) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 640684,79 \text{ грн.}$$

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (1 \cdot 500 + 2000 \cdot (1500 + 1200)) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 1153578,9 \text{грн.}$$

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (1 \cdot 500 + 2000 \cdot (1500 + 1200 + 850)) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 1516585,2$$

грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.20)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

$T$ — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

$\tau$ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 18\%$;

$t$ — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 640684,79/(1+0,18)^1+1153578,9\ /(1+0,18)^2+1516585,2\ /(1+0,18)^3= \\ = 2216736,01 \text{ грн.}$$

Величина початкових інвестицій $PV$, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.21}$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 403338,31 грн.

$$PV = k_{інв} \cdot ЗВ = 2 \cdot 403338,31 = 806676,61 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.22}$$

де $ПП$ — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 2216736,01 грн;

$PV$ — теперішня вартість початкових інвестицій, 806676,61 грн.

$$E_{абс} = ПП - PV = 2216736,01 - 806676,61 = 1410059,39 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.23}$$

де $E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, грн;

$PV$ — теперішня вартість початкових інвестицій, грн;

$T_{ж}$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{в} = \sqrt[Tж]{1+\frac{E_{абс}}{PV}} - 1 = (1+1410059{,}39\ /806676{,}61)^{1/3} — 1 = 0{,}65.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ :

$$\tau_{мін} = d + f\ , \qquad (5.24)$$

де $d$ — середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні $d$ =0,1;

$f$ — показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 0,65 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}}\ , \qquad (5.25)$$

де $E_{в}$ — внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = 1 / 0{,}65 = 1{,}5 \text{ р.}$$

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» становить 43 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий. При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,16 рази.

Також термін окупності становить 1,5 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн».

# ВИСНОВКИ

У комплексній магістерській кваліфікаційній роботі спроектовано розподілену архітектуру веб-застосунку для організації дієти та тренувань з інтеграцією штучного інтелекту та розроблено UI/UX дизайн веб-застосунку.

Проведено аналіз сучасних підходів для вибору архітектури веб-застосунку. Визначені основні критерії для проектування архітектури та проаналізовані функціональні характеристики аналогічних застосунків. Також проведено аналіз предметної області, на основі якого була визначена категорія потенційних користувачів.

Проаналізовано різні типи високорівневої архітектури, а саме — монолітну, мікросервісну та серверлес архітектури, в результаті чого створено розподілену архітектуру шляхом змішування монолітної та мікросервісної архітектур. Обґрунтовано вибір архітектурного стилю при розробці веб-застосунку, в результаті чого був обраний клієнт-серверний архітектурний стиль, оскільки такий клієнт-серверний розподіл допомагає забезпечити ефективну обробку запитів, підвищити масштабованість та краще керувати ресурсами. Розроблено загальну структуру веб-застосунку та визначені основні елементи та зв’язки між ними.

Вдосконалено метод клієнт-серверної взаємодії за допомогою введення проміжного серверу, що дало змогу виконувати запити до бази даних безпосередньо з клієнтської частини, результатом чого стало підвищення швидкості виконання таких запитів. Спроектовано архітектуру веб-застосунку та реалізовано інтерфейс для формування запиту до штучного інтелекту. Проаналізовано трафік та швидкість завантаження сторінки за допомогою сервісу Google PageSpeed Insights. Зроблено висновок, що швидкість завантаження веб-застосунку знаходиться в зеленій зоні, що є досить високим результатом.

Розроблено UI/UX дизайн для веб-застосунку. Вибрано палітру основних кольорів, що відображатимуть тематику веб-застосунку. Створено

UI-KIT з необхідними для дизайну компонентами. Розроблено дизайн сторінок авторизації та основних сторінок. Визначено вимоги до апаратного забезпечення для розробки інтерфейсу веб-застосунку в Figma.

В роботі проведений економічний аналіз доцільності розробки. Абсолютний економічний ефект складає 1410059,39 грн, що свідчить про комерційну привабливість науково-технічної розробки. Термін окупності складає 1,5 роки.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Аналіз методів клієнт-серверної взаємодії при розробці панелі адміністратора за допомогою реактивного фреймворка React [Електронний ресурс] / В.О. Никитюк, О.В. Войцеховська — 2022. — https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15104.

2. Сучасні підходи при розробці архітектури розподілених систем [Електронний ресурс] / В.О. Никитюк, О.С. Городецька, О.В. Войцеховська — 2024. — https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/19088/15838.

3. Калькулятор калорій [Електронний ресурс]. Режим доступу: https://calc.tablycjakalorijnosti.com.ua/.

4. Веб-сайт зі статтями про харчування [Електронний ресурс]. Режим доступу: https://medfond.com/static/kalkulyator-kalorii.html.

5. Flux in-depth overview [Електронний ресурс]. Режим доступу: https://facebook.github.io/flux/docs/in-depth-overview/.

6. MobX introduction [Електронний ресурс]. Режим доступу: https://mobx.js.org/README.html.

7. Патерн Спостерігач [Електронний ресурс]. Режим доступу: https://refactoring.guru/uk/design-patterns/observer.

8. Redux vs MobX [Електронний ресурс]. Режим доступу: https://www.educba.com/mobx-vs-redux/.

9. Monolithic Architecture [Електронний ресурс]. Режим доступу: https://www.techtarget.com/whatis/definition/monolithic-architecture.

10. What are microservices? [Електронний ресурс]. Режим доступу: https://microservices.io/.

11. Serverless Architecture Overview [Електронний ресурс]. Режим доступу: https://www.datadoghq.com/knowledge-center/serverless-architecture/.

12. What is Client-Server Architecture? [Електронний ресурс]. Режим

доступу: https://www.simplilearn.com/what-is-client-server-architecture-article.

13. The Waterfall Methodology [Електронний ресурс]. Режим доступу: https://www.ejable.com/tech-corner/methodologies/waterfall-methodology/.

14. What is Agile Methodology? [Електронний ресурс]. Режим доступу: https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/.

15. The V-Model Methodology [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm.

16. Документація Node.js [Електронний ресурс]. Режим доступу: https://nodejs.org/uk/docs.

17. ASP.NET Official Website [Електронний ресурс]. Режим доступу: https://dotnet.microsoft.com/en-us/apps/aspnet.

18. Qwik framework overview [Електронний ресурс]. Режим доступу: https://qwik.builder.io/.

19. The benefits of ReactJS [Електронний ресурс]. Режим доступу: https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html.

20. Google PageSpeed Insights [Електронний ресурс]. Режим доступу: https://pagespeed.web.dev/.

21. Photoshop vs. Sketch vs. Adobe XD vs. Figma [Електронний ресурс]. Режим доступу: https://belovdigital.agency/blog/photoshop-vs-sketch-vs-adobe-xd-vs-figma-design-apps-comparison/.

22. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

# ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д..

" " 2023 р.

## ТЕХНІЧНЕ ЗАВДАННЯ

на виконання комплексної магістерської кваліфікаційної роботи

Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн»

08-54.КМКР.049.00.000 ТЗ

Науковий керівник: доцент к.т.н.

Городецька О.С.

Студент групи 1КІ-22м

Никитюк В.О.

ВНТУ 2023

1 Підстава для використання КМКР

1.1 Розподілена архітектура розроблена з метою покращення масштабованості, надійності, продуктивності та ефективності додатків. В результаті користувачі отримують зручний та швидкий веб-застосунок для досягнення поставлених цілей здоров’я та тренування.

1.2 наказ про затвердження теми комплексної магістерської кваліфікаційної роботи.

2 Мета і призначення КМКР

Метою комплексної магістерської кваліфікаційної роботи є підвищення швидкодії обробки даних шляхом використання проміжного серверу в архітектурі веб-застосунку для організації тренувань та дієти з використанням штучного інтелекту.

Призначення розробки — вдосконалення методу клієнт-серверної взаємодії шляхом використання проміжного серверу з можливістю виконання запитів до бази даних безпосередньо з клієнтського застосунку, що дало можливість збільшити швидкодію обробки даних.

3 Вихідні данні для виконання КМКР

3.1 Проведення аналізу сучасних підходів для вибору архітектури веб-застосунку.

3.2 Визначення теоретичних основ проектування архітектури веб-застосунку та вибір архітектурного стилю розробки.

3.3 Вдосконалення методу клієнт-серверної взаємодії.

3.4 Розробка UI/UX дизайну для веб-застосунку.

3.5 Виконання розрахунків для доведення доцільності нової розробки.

4 Технічні вимоги до виконання КМКР

Основними вимогами до виконання КМКР є:

— наявність спроектованої розподіленої архітектури із вдосконаленням методу клієнт-серверної взаємодії;

— наявність розробленого UI/UX дизайну веб-застосунку.

5 Етапи КМКР та очікувані результати

Робота виконується за п’ять етапів, що наведені в таблиці А.1.

Таблиця А.1 — Етапи виконання роботи

| № етапу | Назва етапу | Термін виконання | | Очікувані результати |
|---|---|---|---|---|
| | | початок | кінець | |
| 1 | Аналіз предметної області та визначення основних критеріїв для вибору архітектури | 26.09.2023 | 05.10.2023 | Аналітичний огляд літературних джерел |
| 2 | Проектування розподіленої архітектури, розробка загальної структури веб-застосунку, вдосконалення методу клієнт-серверної взаємодії | 06.10.2023 | 16.11.2023 | Вдосконалення методу, проектування архітектури, 2 і 3 розділи |
| 3 | Розробка UI/UX дизайну веб-застосунку | 17.11.2023 | 23.11.2023 | 4 розділ |
| 4 | Підготовка економічної частини | 24.11.2023 | 28.11.2023 | 5 розділ |
| 5 | Оформлення пояснювальної записки, графічного матеріалу і/або презентації | 29.11.2023 | 05.12.2023 | пояснювальна записка, графічний матеріал і/або презентація |

6 Матеріали, що подаються до захисту КМКР

До захисту КМКР подаються: пояснювальна записка КМКР, графічні і ілюстративні матеріали, протокол попереднього захисту КМКР на кафедрі, відзив наукового керівника, рецензія опонента, протоколи проходження перевірки на плагіат, анотації до КМКР українською та іноземною мовами, нормоконтроль про відповідність оформлення КМКР діючим вимогам.

## 7 Порядок контролю виконання та захисту КМКР

Виконання етапів графічної та розрахункової документації КМКР контролюється науковим керівником згідно зі встановленими термінами. Захист КМКР відбувається на засіданні екзаменаційної комісії, затверджено наказом ректора.

## 8 Вимоги до оформлення та проядок виконання КМКР

При оформлюванні КМКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

# ДОДАТОК Б

Дизайн сторінок авторизації

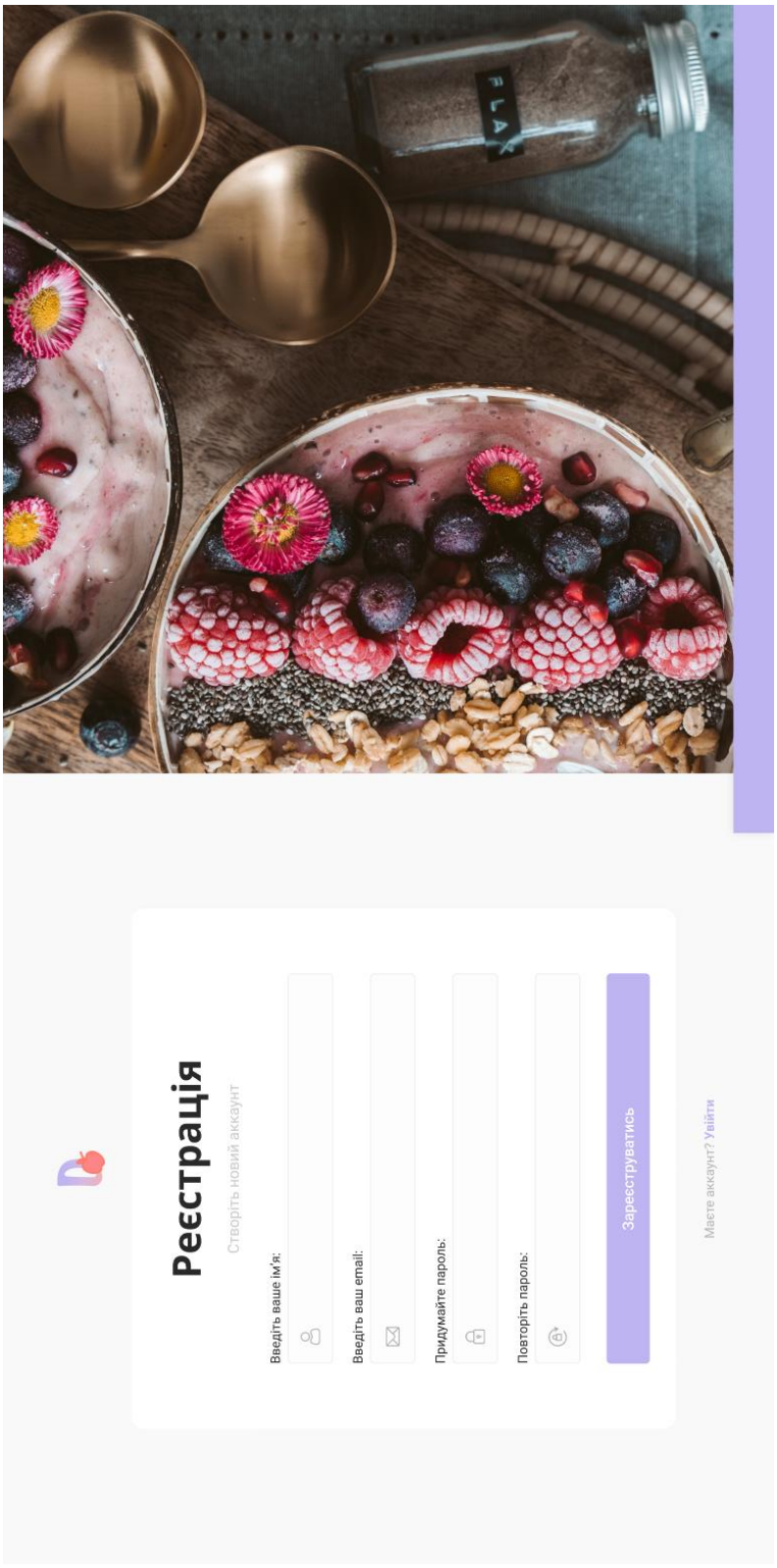


Рисунок Б.1 — Дизайн сторінки реєстрації

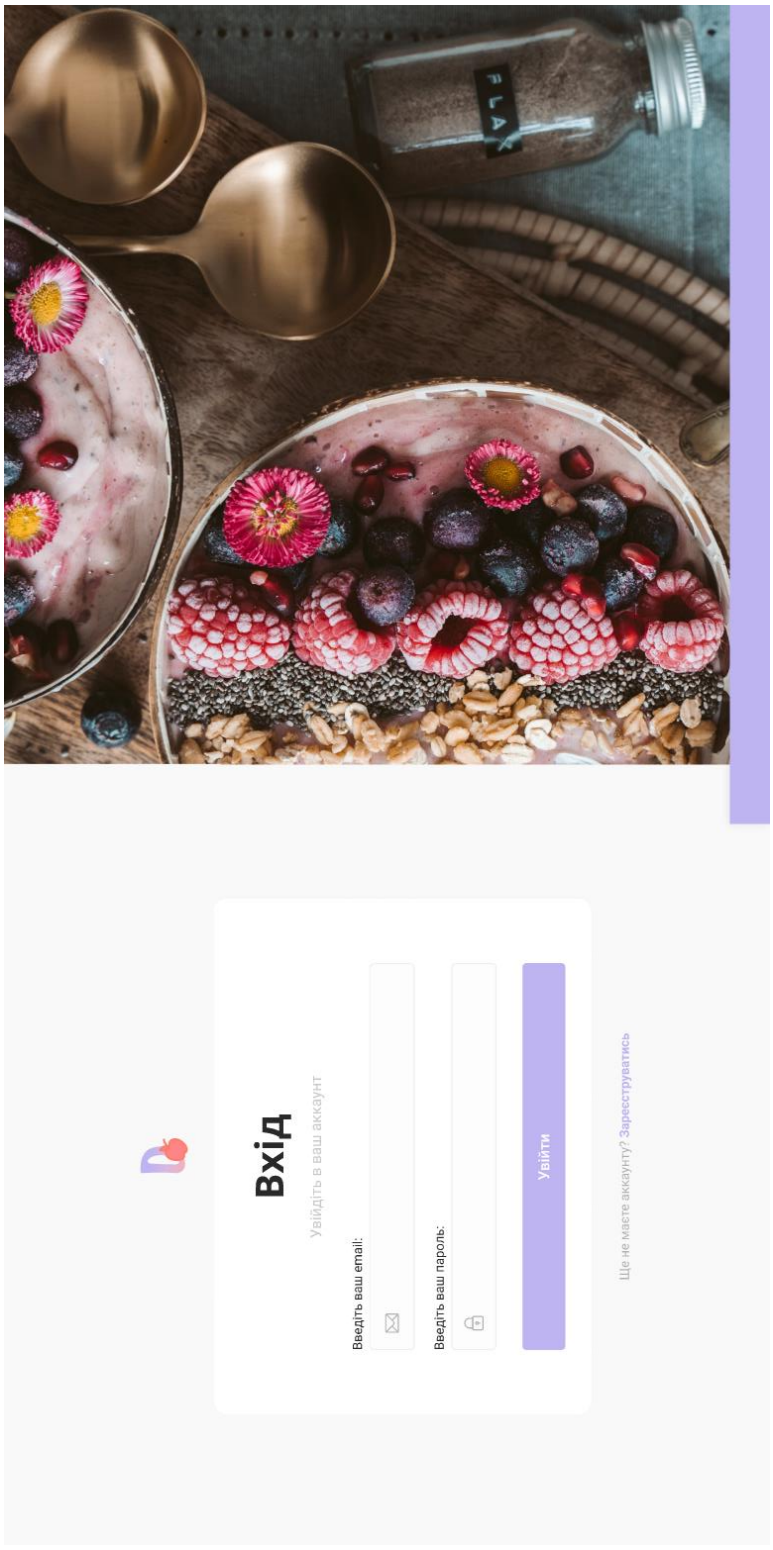


Рисунок Б.2 — Дизайн сторінки входу

# ДОДАТОК В

Дизайн сторінки створення дієти

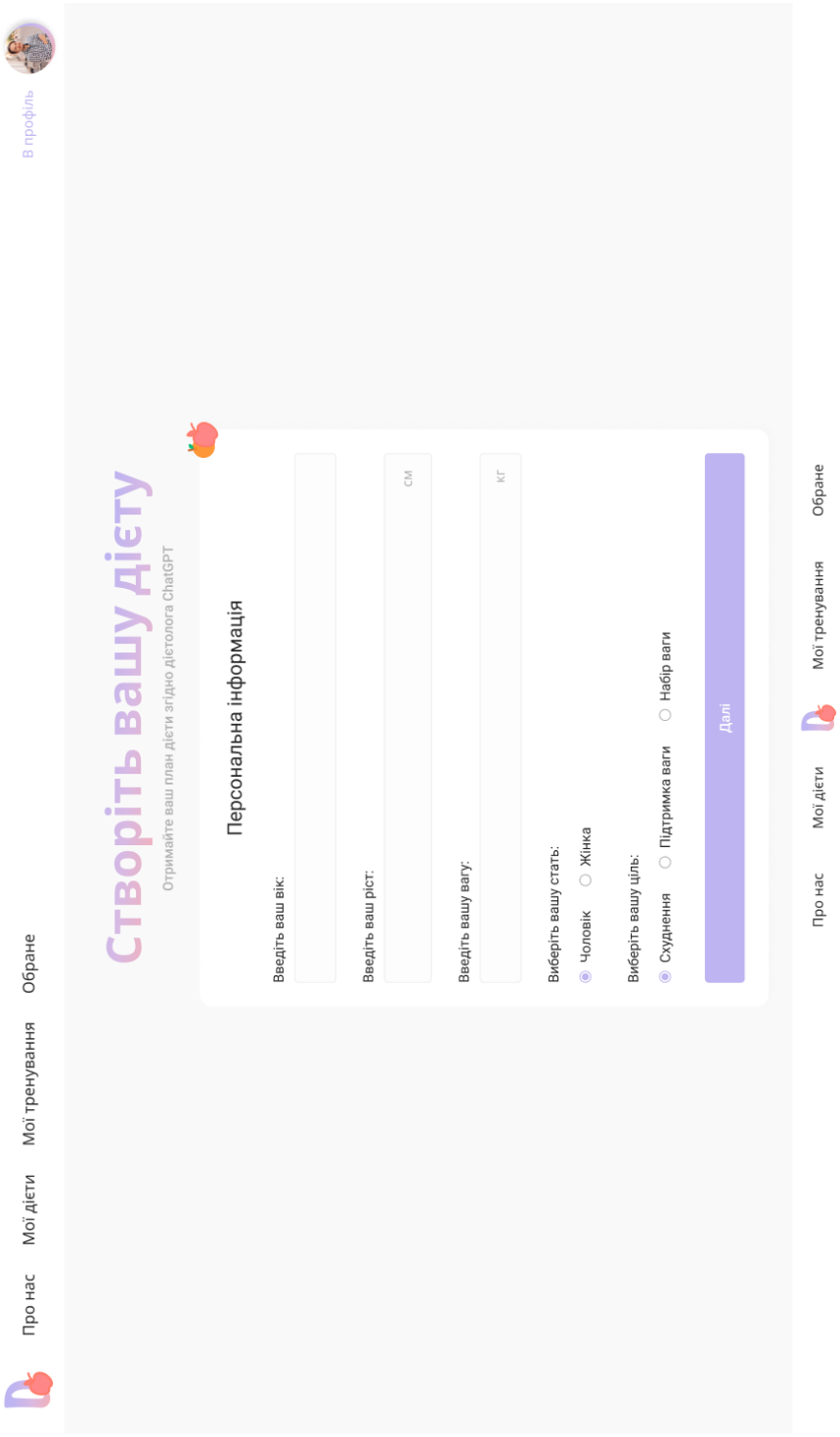


Рисунок В.1 — Дизайн сторінки створення дієти з формою персональної інформації

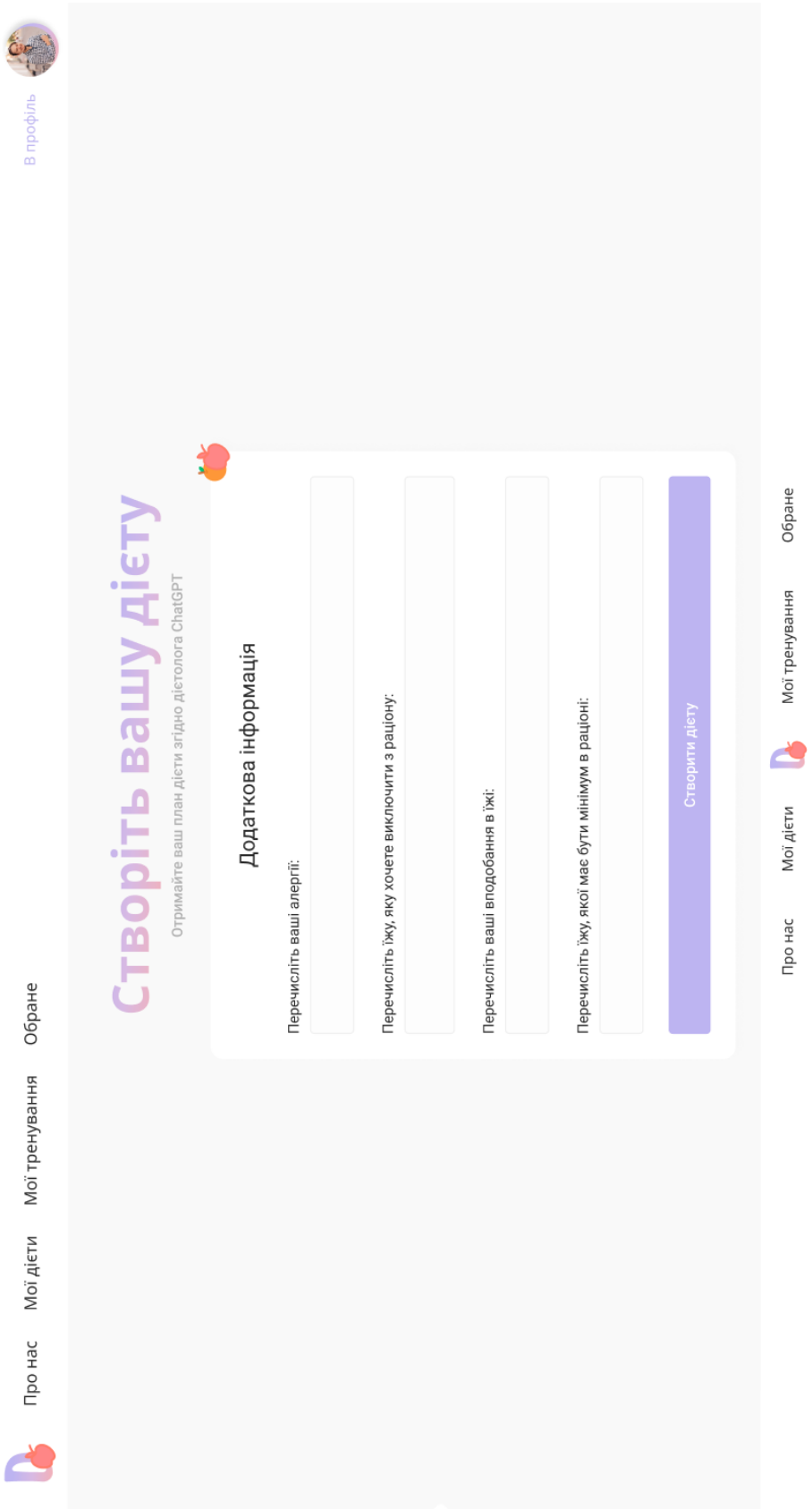


Рисунок В.2 — Дизайн сторінки створення дієти з формою додаткової інформації

# ДОДАТОК Г

UML-діаграма розгортання веб-застосунку

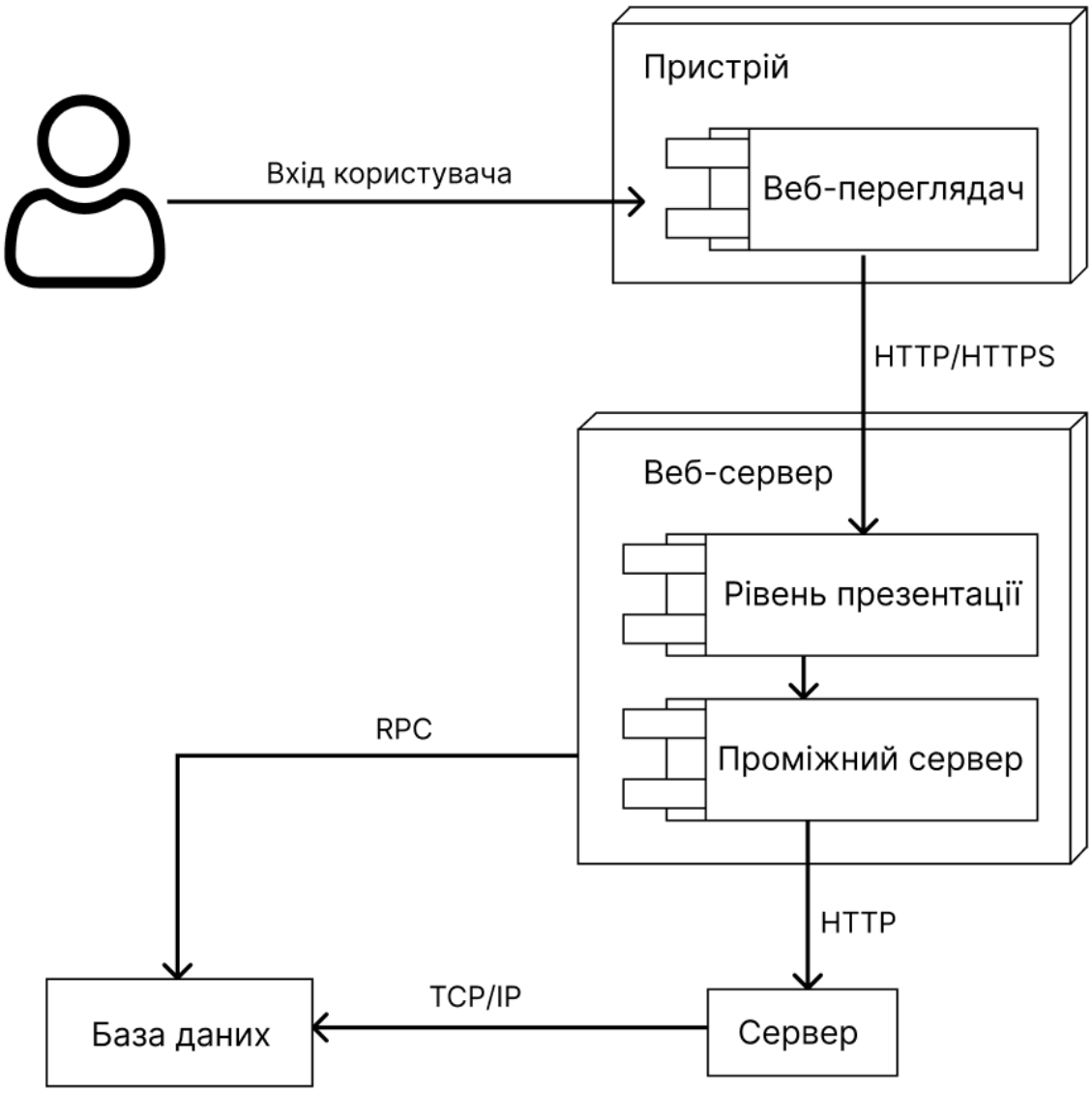


Рисунок Г.1 — UML-діаграма розгортання веб-застосунку

# ДОДАТОК Д

Загальна структура веб-застосунку

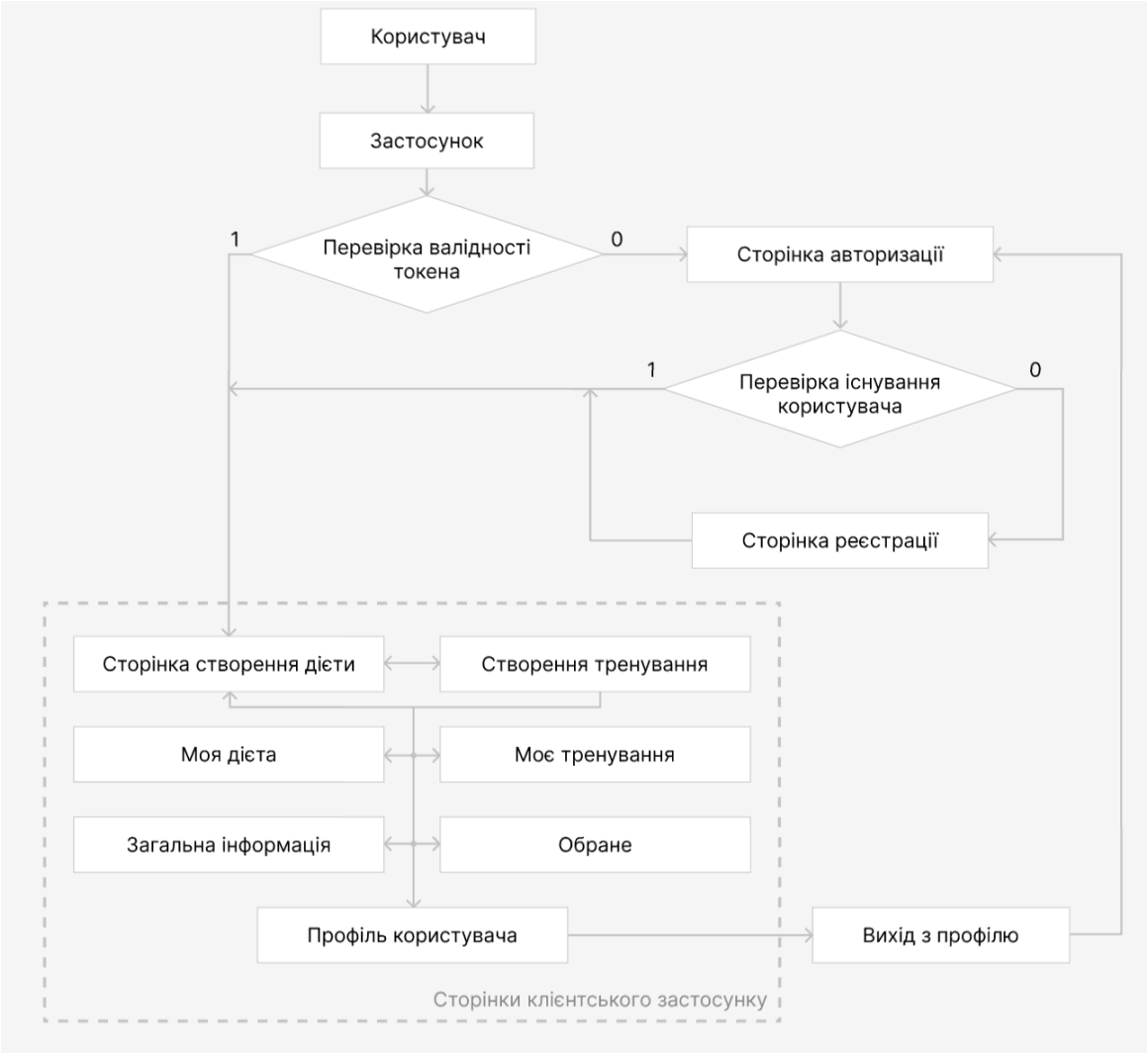


Рисунок Д.1 — Розподілена архітектура веб-застосунку

# ДОДАТОК Е

Структурна схема архітектури веб-застосунку

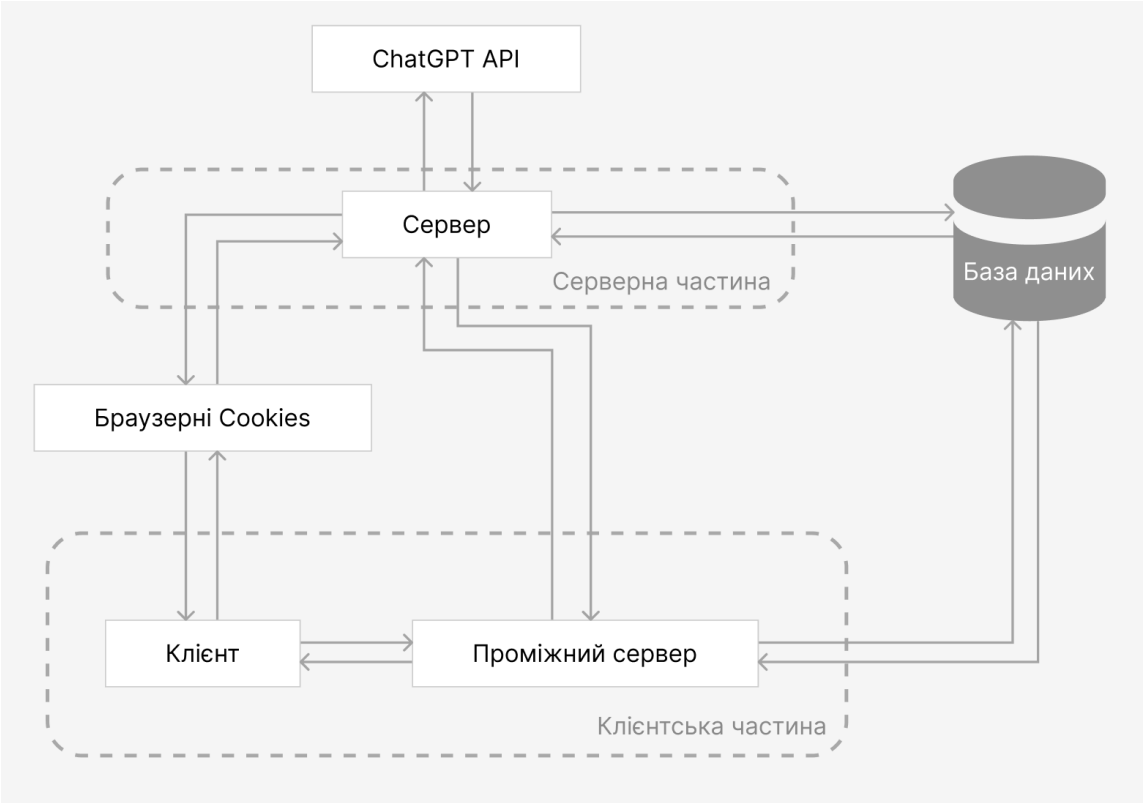


Рисунок Е.1 — Загальна структурна схема веб-застосунку

# ДОДАТОК Ж

Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн»

Тип роботи: комплексна магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

**Показники звіту подібності Unicheck**

Оригінальність 90.2% Схожість 9.8%

Аналіз звіту подібності (відмітити потрібне):

✓ **Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Никитюк В.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Городецька О. С.
(підпис) (прізвище, ініціали)