Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп`ютерних технологій
Наталія СТЕФУРАК/_____/
підпис
«___» _____ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: «Вебсайт мережі фітнес клубів та спортивного харчування»

Студент групи КН-41	Роман ЧИБИНЯК	_____ (підпис)
Керівник проєкту	Василь КУЗИК	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов Меленчук	_____ (підпис)
нормоконтролер	Наталія КУЛЬЧИНСЬКА	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК/_____/
підпис
«___» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»
студенту Чибиняку Роману Ярославовичу
(прізвище, ім’я та по-батькові студента)

1. Тема проєкту «Веб сайт мережі фітнес клубів та спортивного харчування» затверджена наказом по коледжу від “25 ”листопада 2024 р., №253а-н.
2. Термін здачі студентом завершеного проекту “25” червня 2025 р.
3. Вихідні дані до проекту _____
_____
4. Перелік питань, які повинні бути розроблені в проекті:
а) основна частина аналіз існуючих рішень та постановка завдань; проєктування системи; реалізація та тестування системи
б) техніко-економічне обгрунтування аналіз ринку; розрахункова частина; обгрунтування необхідності розробки
5. Перелік графічного матеріалу _____
_____

6. Консультанти проєкту:

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обгрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН

дипломного проєктування

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами до кваліфікаційної роботи		
2.	Огляд типових рішень та написання відповідного розділу ПЗ		
3.	Дослідження технологій реалізації та написання відповідного розділу ПЗ		
4.	Розробка функціональних вимог до проекту та робота над структурою програмного продукту. Написання відповідного розділу ПЗ		
5.	Встановлення на налаштування середовища реалізації та написання відповідного розділу ПЗ		
6.	Проєктування програмного засобу (функціоналу, інтерфейсу, бази даних продукту) та написання відповідного розділу ПЗ		
7.	Реалізація та налаштування програмного засобу та написання відповідного розділу ПЗ		
8.	Доопрацювання модулів		
9.	Тестування на налагодження програмного продукту та написання відповідного розділу ПЗ		
10.	Опрацювання економічного розділу дипломного проєкту та оформлення спеціального розділу		
11.	Робота над оформленням пояснювальної записки		
12.	Попередній захист кваліфікаційної роботи, доопрацювання		
13.	Підготовка до захисту кваліфікаційної роботи		
14.	Захист кваліфікаційної роботи		

7. Дата видачі завдання “25” травня 2025 р.

Керівник _____/Кузик В.М.

Завдання прийняв до виконання _____/Чибиняк Р.Я.

Реферат

Вебсайт для організації та автоматизації процесів мережі фітнес-клубів та спортивного харчування. Чибиняк Роман Ярославович. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. Спеціальність 122 «Комп’ютерні науки». ГФК, 2025. Сторінок – 111, Рисунків – 49, Додатків – 1

Об’єктом дослідження є методи та засоби автоматизації бізнес-процесів у сфері фітнес-індустрії, інструменти для розробки та впровадження інформаційних систем, принципи клієнт-серверної архітектури, а також методи та інструменти розробки адаптивних веб-інтерфейсів для ефективної взаємодії з базами даних.

Метою проєкту є реалізація вебсайту, який забезпечить комплексну автоматизацію процесів функціонування мережі фітнес-клубів та продажу спортивного харчування.

Система повинна бути реалізована у вигляді бази даних, що зберігатиме записи та подаватиме інформацію у структурованому вигляді, а також адаптивного веб-інтерфейсу для взаємодії з нею.

Для реалізації вебсайту було використано сучасні інструменти: JavaScript (JS) для розробки як клієнтської, так і серверної частини (Node.js з фреймворком Express.js), а також система управління базами даних Mongo (MongoDB) у поєднанні з бібліотекою Mongoose для зручного моделювання та взаємодії з даними

Результатом розробки стала завершений вебсайт "FitForge", який виконує всі запроєктовані функції.

ВЕБСАЙТ, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, БАЗИ ДАНИХ, JAVASCRIPT, ВЕБ-ДОДАТОК.

Abstract

A website for the organization and automation of processes for a network of fitness clubs and sports nutrition. Chybyniak Roman Yaroslavovych. Halytskyi Professional College named after Vyacheslav Chornovil, Department of Computer Technologies. Specialty 122 "Computer Science." GFK, 2025. Pages – 111, Figures – 49, Appendices – 1.

The object of study is the methods and means of automating business processes in the fitness industry, tools for the development and implementation of information systems, principles of client-server architecture, as well as methods and tools for developing adaptive web interfaces for effective interaction with databases. The goal of the project is to implement a website that will ensure comprehensive automation of the processes of functioning a network of fitness clubs and selling sports nutrition.

The system should be implemented in the form of a database that will store records and present information in a structured form, as well as an adaptive web interface for interacting with it.

For the implementation of the website, modern tools were used: JavaScript (JS) for both client-side and server-side development (Node.js with the Express.js framework), as well as the Mongo (MongoDB) database management system in conjunction with the Mongoose library for convenient modeling and interaction with data.

The result of the development was a completed website, "FitForge," which performs all the designed functions. Keywords: WEBSITE, BUSINESS PROCESS AUTOMATION, CLIENT-SERVER ARCHITECTURE, DATABASES, JAVASCRIPT, WEB APPLICATION.

Зміст

Вступ 6
1 Аналіз існуючих рішень та постановка завдання 7
1.1 Дослідження об’єкту інформатизації 7
1.2 Обґрунтування доцільності створення сайту 8
1.3 Аналіз існуючих аналогів 8
1.4 Постановка завдань 13
2. Проєктування системи 14
2.1. Опис архітектури системи та обґрунтування вибору технологій 14
2.2. Розробка моделей системи 15
2.3. Проєктування інтерфейсу користувача (User Interface Design) 21
2.4. Проєктування алгоритмів роботи основних функцій 25
2.5. Проєктування бази даних 27
3 Реалізація та тестування системи 31
3.1.Необхідне системне, прикладне та інструментальне програмне забезпечення 31
3.2.Встановлення та налаштування необхідного технічного та програмного забезпечення 32
3.3. Процес розробки програмного продукту. 36
3.4. Створення бази даних 66
3.5. Реалізація інтерфейсу користувача 70
3.6. Тестування системи 72
4 Техніко-економічне обґрунтування 82
4.1 Аналіз ринку 82
4.2 Розрахункова частина 83
4.3 Обґрунтування необхідності розробки 85
Висновки 86
Перелік джерел посилання 88
Додатки 89

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД – Система управління базами даних.

ОС – Операційна система.

API – Application Programming Interface.

CMS – Content Management System.

CRM – Customer Relationship Management.

CSS – Cascading Style Sheets.

DOM – Document Object Model.

HTML – HyperText Markup Language.

HTTP/HTTPS – HyperText Transfer Protocol / Secure.

IDE – Integrated Development Environment.

JSON – JavaScript Object Notation.

JWT – JSON Web Token L.

TS – Long Term Support.

NPM – Node Package Manager.

ODM – Object Data Modeling.

SaaS – Software as a Service.

SVG – Scalable Vector Graphics.

UI – User Interface.

UX – User Experience.

ВСТУП

Зал для тренувань — це не просто місце для фізичних вправ, а справжня арена для досягнення особистих цілей і розвитку. Сайт для мережі фітнес-клубів і спортивного харчування — це зручність як для клієнтів, так і для самого клубу. Такий ресурс представлений онлайн-платформою,яка об'єднує все необхідне для тренувань, харчування та здорового способу життя в одному місці.

Клієнти можуть отримувати актуальну інформацію про тренування, розклад занять,а також замовляти персональні тренування. Завдяки інтерактивним можливостям, відвідувачі можуть записатися на заняття, оплачувати абонементи.

Актуальність цієї роботи є високою у сучасному світі, через те, що багато користувачів потребують інформації про фітнес зали в онлайн режимі.

Метою кваліфікаційної роботи є створення сайту “Веб сайт мережі фітнес клубів та спортивного харчування”, який дозволить розповісти про заклад , зокрема графік його робити, допоможе потенційним користувачам зорієнтуватися серед немалої кількості подібних фітнес клубів , ознайомитися з послугами що надаються.

Для досягнення цілей проєкту необхідно виконати наступні завдання: розробити структуру сайту, вибрати середовище і засоби розробки, виконати його фізичну реалізацію.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Дослідження об’єкту інформатизації

Метою завдання є розробка сайту для фітнес клубу з продажею спортивного харчування. Даний сайт буде корисним для людей які прагнуть вдосконалити себе завдяки спорту.

Першим етапом є визначення тематики сайту, яка буде зацікавлювати аудиторію. Потрібно зацікавити людей, а саме добавити текст який буде надихати людей, розклад занять та різні види спортивного харчування. Важливо врахувати, що аудиторія — це люди різних різних вікових груп та рівня фізичної підготовки . Тому контент повинен відповідати їхнім різним потребам.

Наступним кроком є аналіз схожих сайтів за тематикою. Це дозволить зрозуміти, які функції та можливості приваблюють людей. Розгляд цих сайтів допоможе визначити ефективні методи для створення ресурсу. Також важливим є питання залучення нових відвідувачів сайту та утримання постійних.

Сайт буде надавати користувачам доступ до широкого спектра послуг: інформації про тренування, можливість придбати спортивні добавки та харчування, а також можливість онлайн-замовлення продукції. Дивлячись на те, що люди все більше звертаються до онлайн-платформ для отримання послуг, сайт повинен бути зручним і простим у використанні, з можливістю запису на заняття, оплатою абонементів, а також швидким доступом до інформації про спортивне харчування та його переваги. Для багатьох людей заняття спортом — це захоплююча подорож, в якій можна відкривати нові горизонти в особистих досягненнях та надавати своєму тілу необхідну увагу для підтримки здоров'я. Окрім фізичних переваг, спортивні тренування можуть значно змінити наше сприйняття здорового способу життя, адже кожен вид спорту має свої традиції та підходи, що пов’язані з культурою та історією тієї чи іншої практики.

Це також можливість навчитися новим технікам тренувань, дізнатися про спортивне харчування та добавки, що допомагають досягти найкращих результатів.

1.2 Обґрунтування доцільності створення сайту

Створення сайту для фітнес-клубів і продажу спортивного харчування є надзвичайно актуальним в сучасних умовах. Сьогодні, з розвитком здорового способу життя та популяризацією спорту, все більше людей звертається до онлайн-платформ для пошуку інформації про тренування, харчування та спортивні добавки. Водночас, зважаючи на величезну кількість доступних джерел, людям іноді важко знайти якісні ресурси, що пропонують перевірену інформацію та продукти, що відповідають їхнім потребам.

Інтернет перетворився на ключове джерело відомостей, де кожен користувач здатен швидко відшукати корисні поради та рекомендації. Розробка веб-ресурсу, присвяченого фітнесу та спортивному харчуванню, дасть змогу зібрати всю потрібну інформацію в одному місці. На такому веб-сайті відвідувачі зможуть дізнатися про тренування, а також придбати необхідні добавки та спортивне харчування.

Ще однією вагомою причиною для створення такого сайту є потенціал для розвитку бізнесу. Платформа може стати потужним інструментом для реклами та продажу спортивних товарів. Завдяки інтеграції онлайн-магазину, користувачі матимуть змогу зручно придбати необхідні продукти, що доповнюють їхні тренування. Окрім того, сайт може стати майданчиком для партнерства з виробниками спортивного харчування, обладнання або спортивних аксесуарів.

Отже, створення сайту для фітнес-клубів і продажу спортивного харчування є не лише актуальним і необхідним в умовах сучасного ринку, а й вигідним бізнес-проектом, що дозволяє задовольнити потреби широкої аудиторії.

1.3 Аналіз існуючих аналогів

Вивчення вже існуючих сайтів у тематиці фітнесу та спортивного харчування - це не просто формальність, а ключове рішення, чого хочуть користувачі й як їм це найкраще подати. Бо зрозуміти, що працює, а що ні – означає зекономити час і сили на власній розробці.

Типова структура таких ресурсів, здається, вже сформувалася сама собою. Головна сторінка - це вітрина магазину, де можна швидко побачити найсвіжіші пропозиції або топові товари.

Далі – категорії. Вони допомагають розкласти все по поличках: спортивне харчування, абонементи, аксесуари, тренування – щоб користувач міг легко знайти саме те, що шукає. Коли все заплутано – втрачена увага, і потенційний клієнт швидко йде.

Особливо цікавою деталлю є розділ відгуків. Тут користувачі діляться своїм досвідом, ставлять питання або радять щось іншим. Це – не просто додаткова опція, а фактичний елемент довіри. Якщо люди бачать, що інші задоволені, то й самі охоче починають взаємодіяти.

Контактна інформація, зрозуміло, має бути максимально доступною - бо іноді простий дзвінок або консультація вирішують набагато більше, ніж безкінечні сторінки з текстом.

Для конкретного аналізу взяли кілька відомих прикладів – Member24 зі Швеції, FitnessLife з їх простотою і зрозумілістю, а також український SportLife, який поєднує і магазин, і тренування, і корисну інформацію в одному місці. Ці сайти дають гарне уявлення про те, що вже працює на ринку і де можна шукати власні рішення.

Першим для аналізу обрано мережу у Швеції “Member24”(рисунок 1.1).

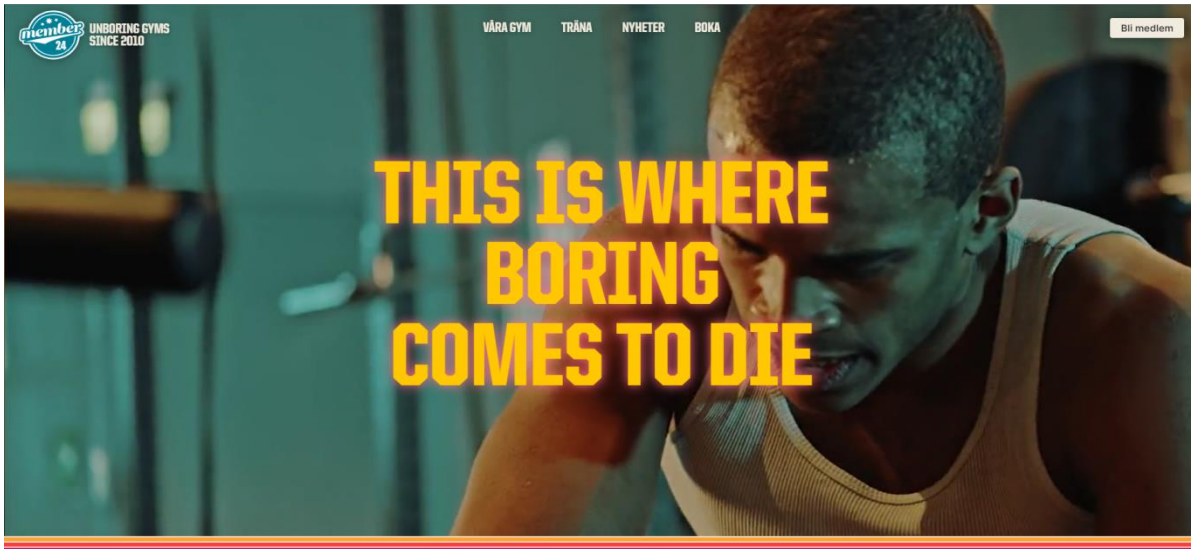

Рисунок 1.1 – Сайт мережі залів “Member24”

Платформа Member24 – це не просто сайт, а ідеальний екземпляр якісного продукьу . Вже на першій сторінці вони дають зрозуміти – тут усе інакше. Слоган “Прощай, нудний спортзал” не просто для краси - він чітко окреслює їхній стиль комунікації. Весело, просто і з натяком, що фітнес не мусить бути рутиною.

Навігація – проста. Запис на тренування всього в кілька кліків. Усе чітко структуровано. Немає заплутаних сторінок або складних форм, і це помітно знижує поріг входу, особливо для нових відвідувачів, які ще не зовсім “у темі”.

Розділ з новинами. Тут справді можна знайти корисне: оновлення графіку, тимчасові акції, зміни в клубах. Це створює ефект “живого” простору, де щось постійно відбувається, де є життя поза розкладом і гантелями.

Дизайн – яскравий та сучасний. В цьому й є його ключовий плюс. Він говорить мовою покоління, яке виросло на інстаграм-сторісах, швидких кліках і візуальних рішеннях. Просто, зручно. І саме така подача робить Member24 чимось більшим за “черговий фітнес-бренд”.

Другим прикладом для дослідження обрано сайт FitnessLife (рисунок 1.2).

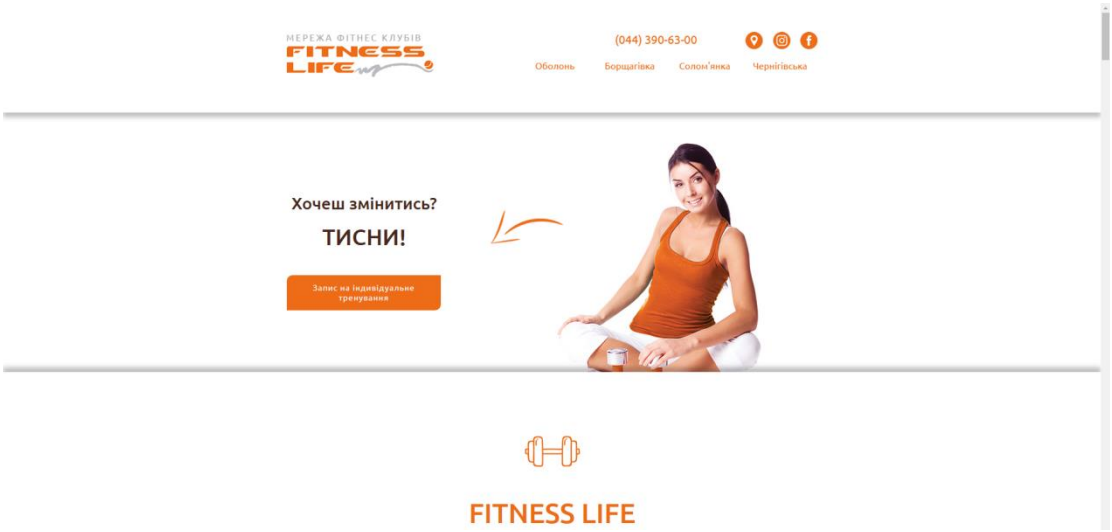

Рисунок 1.2 - Мережа “FitnessLife”

Сайт Fitness Life — це мережа фітнес-клубів по всій Україні. І не просто шаблонна сторінка з цінами та адресами, а повноцінна платформа, яка намагається бути зручною і для новачків, і для тих,хто шукає щось конкретне – наприклад сауну після тренування або бар із протеїновими коктейлями.

Головна сторінка – все як треба: коротко про мережу, кілька акцентів на тому, чому саме вони, й обов'язковий блок з локаціями. Тут важливо, що акцент робиться не на "величезному досвіді" чи "кращому обладнанні", а на зручності: доступність залів, вибір форматів, комфорт під час тренувань. Це те, що зачіпає не тільки фанатів спорту, а й звичайних відвідувачів.

Записатись на тренування можна досить просто, а саме індивідуально чи в групі – обираєш, що більше підходить по часу, по настрою чи банально по цінах. Система не змушує читати інструкцію, інтерфейс інтуїтивний.

Контентна частина розгалужена, але не перевантажена. Кардіозона, фітнес-бар, масаж, сауна – усе це подається не сухим списком, а з поясненнями, фото, деталями.

Візуальна складова сайту. Сучасний, чистий дизайн, без нав'язливих анімаційі. Розділи чітко структуровані, нічого не губиться, навіть з мобільного.

Загалом, платформа залишає по собі враження продуманої системи.

Наступним ресурсом для аналізу обрано SportLife (рисунок 1.3).

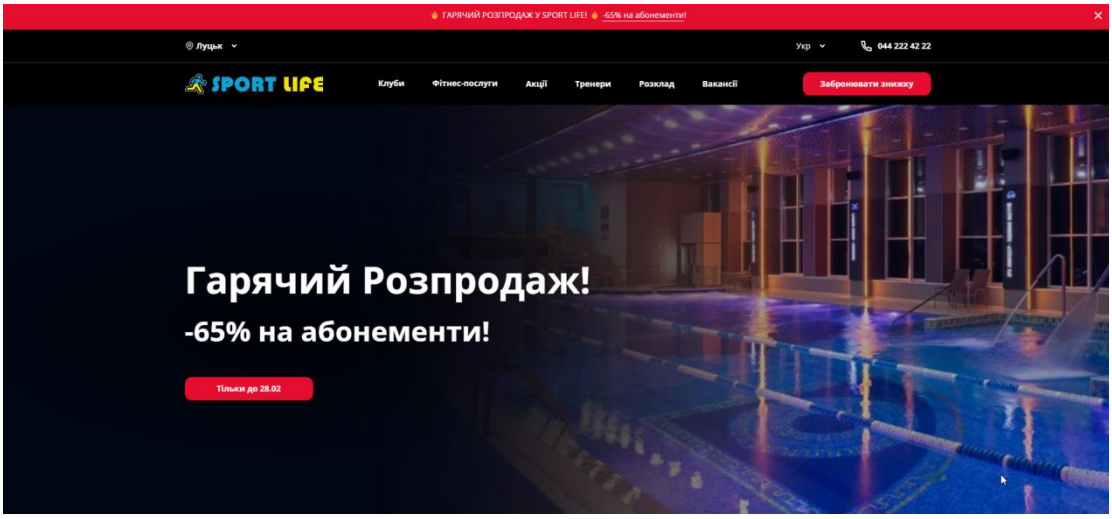

Рисунок 1.3- мережа "SportLife"

Sport Life – це вже давно не просто фітнес-клуб, а ціла мережа з широким набором послуг, які охоплюють найрізноманітніші потреби клієнтів. І перше це акцент на спеціальних пропозиціях і знижках, які не просто заманюють новачків, а дають стимул залишатися надовго. Бо в спорті, як і в житті, мотивація – це половина успіху.

Крім звичних тренажерних залів, тут пропонують басейни, дитячі програми, сауни, спа – перелік, який охоплює фактично всі можливі варіанти активного відпочинку і відновлення. Особливо цікаво, що дітям приділяється окрема увага: тренування та майстер-класи, що допомагають не лише фізично розвиватися, а й соціалізуватися.

Що стосується дизайну – він справді сучасний і яскравий, але при цьому не нав'язливий. Все зроблено так, щоб користувач швидко знайшов те, що шукає, і міг без проблем забронювати заняття або знайти інформацію про найближчий клуб. Чіткість і простота – ось що цінують відвідувачі.

Філії Sport Life розкидані по багатьох містах України, і сайт надає всю необхідну інформацію про кожен із них. Це важливо, адже доступність – ключова складова для великої аудиторії з різними потребами.

Кожен із трьох аналізованих сайтів – Member24, FitnessLife і Sport Life має свої сильні сторони. Member24 вражає своєю простотою й інформативністю, знайти інформацію про клуби в різних містах тут легко. Дизайн не просто сучасний, а ще й яскравий, що створює одразу приємну атмосферу. Відчувається, що сайт налаштований на позитивний настрій відвідувачів, і це важливо.

Fitness Life пропонує більш широкий спектр послуг: не лише стандартні тренування, а й фітнес-бар, масаж – речі, які додають комфорту і роблять перебування в клубі приємнішим.

Sport Life має свої плюси, особливо різні вікові категорії – від дорослих до дітей. І регулярні спеціальні пропозиції додають варіативності. Але навігація часом виглядає трохи заплутаною, а процес реєстрації чи запису на тренування не найпростіший, що може відлякати користувачів.

Member24 хоч і легкий у користуванні, але іноді здається, що функціонал трохи складно поданий. Fitness Life – це вже більше можливостей, але старенький дизайн і відсутність персоналізації знижують привабливість. А Sport Life попри багатство контенту і різноманіття, міг би бути більш лояльним до користувача, особливо для тих, хто не звик до онлайн-сервісів.

Отже, всі три сайти мають власні плюси, але водночас відкривають простір для вдосконалень. Можливо, саме баланс між простотою навігації та гнучкістю функцій – це те, що варто взяти за основу в майбутніх проєктах?

1.4 Постановка завдань

Основна мета - створити сайт, який буде дійсно зручним і корисним для відвідувачів. Потрібно зробити дизайн, що враховує потреби різних користувачів, з простою навігацією і приємним виглядом. Адже, якщо інтерфейс незрозумілий, чи важкий – люди швидко розчаровуються.

Важливо продумати систему, яка зручно показуватиме інформацію про фітнес-клуб: тренування, тренерів, а також можливість купити спортивне харчування прямо на сайті. Тут мають бути всі деталі – ціни, опис, характеристики товарів, щоб люди не шукали зайвих пояснень.

Обов’язковим є функціонал для реєстрації користувачів – це дозволить персоналізувати досвід, а також спростити оформлення замовлень і запис на тренування.

Без якісного тестування не обійтися. Перевірити сайт на помилки, недоліки – щоб усе працювало плавно і без збоїв. Користувачі не люблять, коли щось зависає чи ламається, тому комфорт і стабільність – на першому місці.

2. ПРОЄКТУВАННЯ СИСТЕМИ

На етапі аналізу було визначено основні вимоги та функціональні можливості майбутньої системи – веб-сайту фітнес-клубу "FitForge". Метою даного розділу є розробка детального проєкту системи, який дасть відповідь на питання: «як реалізувати поставлені завдання?». Результати, отримані на етапі аналізу, тут трансформуються у конкретні технічні рішення.

У цьому розділі буде описано архітектуру системи, обґрунтовано вибір технологій, розроблено інформаційні та структурні моделі, спроєктовано користувацький інтерфейс та основні алгоритми роботи системи, а також розглянуто питання проєктування бази даних.

2.1 Опис архітектури системи та обґрунтування вибору технологій

Для проєкту "FitForge" нічого нового вигадано не було – зупинились на класичній клієнт-серверній архітектурі. Це, по суті, стандарт для веб-додатків, і не просто так. Такий підхід дає чіткий поділ: є логіка відображення на фронтенді, і є бізнес-логіка з даними, що надійно заховані на бекенді. Це дозволяє масштабувати кожну частину незалежно і, що важливо, краще захищати дані. Вся комунікація між ними – звичайні HTTP/HTTPS запити. Клієнт просить, сервер відповідає.

Вибір технологій – це завжди компроміс між функціоналом, швидкістю розробки і планами на майбутнє. Був знайдений стек, який дозволить швидко рухатись і не зв'яже руки потім.

На клієнті все досить звично. В основі лежить семантичний HTML, а за вигляд відповідає CSS. Але ключовим рішенням тут став Tailwind CSS. Це утилітарний фреймворк, який дозволив зібрати кастомний інтерфейс з готових блоків, не пишучи гори власного CSS-коду. Неймовірна гнучкість і швидкість. Вся інтерактивність, звісно, на плечах JavaScript. А для візуального лоску – легенькі іконки Feather Icons, підключені прямо через CDN. Просто й ефективно.

Серверна частина – це територія JavaScript. Було обрано Node.js через його високу продуктивність в асинхронних операціях, та й можливість використовувати одну мову на фронті і беку – це просто та зручно. Поверх нього працює Express.js – мінімалістичний і гнучкий фреймворк для побудови API, обробки запитів і всього, що з цим пов'язано.

Інформація про користувачів, їхні кошики, товари, замовлення – все це не завжди вкладається в жорсткі реляційні таблиці. Тому вибір впав на MongoDB, документо-орієнтовану NoSQL базу. А щоб якось приборкати цю гнучкість і додати структуру, було використано Mongoose – бібліотеку, яка дає схеми, валідацію і значно спрощує роботу з базою. Безпека – це основа. Паролі хешуються за допомогою bcryptjs перед збереженням, а для автентифікації – JSON Web Tokens. Користувач логіниться, отримує токен і пред'являє його при кожному захищеному запиті. Класична і надійна схема.

Вся ця система спілкується між собою за допомогою JSON – легкої мови, зрозумілої і клієнту, і серверу.

Інтеграція з API Нової пошти сильно покращує користувацький досвід. На сторінці замовлення це дозволяє динамічно підтягувати міста і відділення. Вона зменшує кількість помилок при введенні адреси. Іноді саме такі фічі й відрізняють просто робочий сайт від зручного.

Обрана архітектура і технології створюють гнучку й масштабовану систему. Вона готова до поточних завдань.

2.2 Розробка моделей системи

Моделювання системи є важливим етапом проєктування, що дозволяє визначити ключові сутності, їх атрибути та взаємозв'язки. Для веб-сайту "FitForge" було розроблено інформаційну та структурну моделі.

2.2.1 Інформаційна модель

Інформаційна модель описує основні сутності даних, з якими працює система, та зв'язки між ними. Основними сутностями є: користувач, товар, кошик, замовлення, запис на абонемент.

Сутність "Користувач" (User) зображено у таблиці 2.1.

Таблиця 2.1 — Сутність користувача

Ім’я	Тип даних	Інформація
id	ObjectId	Унікальний ідентифікатор користувача.
username	String	Ім'я (логін) користувача.
email	String	Електронна пошта (унікальна, обов'язкова).
password	String	Хешований пароль.
cart	Array of Objects	Кожен об'єкт у масиві представляє товар у кошику.
productId	ObjectId	Посилання на товар.
quantity	Number	Кількість одиниць товару.

Зв'язки: один користувач може мати багато товарів у кошику, один користувач може зробити багато замовлень, один користувач може мати багато записів на абонементи.

Сутність "Товар" (Product) зображено у таблиці 2.2.

Таблиця 2.2 — сутність “Товар”

Ім’я	Тип даних	Інформація
id	ObjectId	Унікальний ідентифікатор товару.
name	String	Назва товару.
type	String	Тип/категорія товару.
price	Number	Ціна товару.
description	String	Опис товару.
imageUrl	String	Шлях до зображення товару.

Зв'язки: один товар може бути у багатьох кошиках користувачів та у багатьох замовленнях.

Сутність "Замовлення" (Order):

1) _id (ObjectId): Унікальний ідентифікатор замовлення;

2) userId (ObjectId, ref: 'User'): Посилання на користувача, що зробив замовлення (обов'язкове).

3) products (Array of Objects): Список замовлених товарів. Кожен об'єкт:

— productId (ObjectId, ref: 'Product'): Посилання на товар;

— name (String): Назва товару;

— quantity (Number): Замовлена кількість;

— price (Number): Ціна за одиницю на момент замовлення;

— totalPrice (Number): Загальна вартість замовлення (обов'язкове).

4) customerInfo (Object): Інформація про клієнта:

— fullName (String);

— email (String);

— phone (String);

— comment (String);

5) shippingInfo (Object): Інформація про доставку:

— cityRef (String);

— cityName (String);

— warehouseRef (String);

— warehouseName (String);

— createdAt (Date): Дата та час створення замовлення.

Зв'язки – одне замовлення належить одному користувачеві та може містити багато товарів.

Сутність "Запис на абонемент" (Enrollment). Ці дані зберігаються у файлі enrollments.json.

Структура кожного запису зображена у таблиці 2.3.

Таблиця 2.3 — сутність запису на абонемент

Ім’я	Тип даних	Інформація
id	String	Унікальний ідентифікатор запису.
userId	String	ID користувача системи, який зробив запис.
username	String	Логін користувача системи.
email	String	Email користувача системи.
productId	String	Ідентифікатор або назва обраного абонемента.
firstName	String	Ім'я особи, яка записується.
lastName	String	Прізвище особи, яка записується.
age	Number	Вік особи.
phoneNumber	String	Номер телефону особи.
timestamp	String ISO Date	Дата та час створення запису.

2.2.2 Математична модель

У рамках розробленого веб-додатку "FitForge" складні математичні моделі або алгоритми не застосовувалися. Основні математичні операції зводяться до простих арифметичних розрахунків:

Розрахунок загальної вартості товару в кошику:

сума_позиції = ціна_товару * кількість_товару.

Розрахунок загальної вартості кошика/замовлення: загальна_вартість = Σ (сума_кожної_позиції).

Розрахунок у грі "Вгадай число" (сторінка профілю):

Генерація випадкового цілого числа в діапазоні [1, 100] (Math.floor(Math.random() * 100) + 1), порівняння введеного користувачем числа із загаданим, підрахунок кількості спроб.

2.2.3 Структурна модель системи (Компонентна архітектура)

Якщо говорити про структурну модель "FitForge", то її можна описати доволі просто. Вся система, по суті, складається з двох великих компонентів, що постійно взаємодіють між собою.

Це клієнтська частина, яку ми називаємо фронтендом, і серверна частина наш бекенд (рисунок 2.1).

Рисунок 2.1 - Блок-схема архітектури системи

Модуль інтерфейсу користувача (UI) передбачає:

– Компонент навігації забезпечує навігацію по сайту, відображення логотипу, посилань на основні розділи, а також динамічні посилання "Вхід"/"Профіль", "Кошик". Реалізований як фіксований елемент на всіх сторінках.

– Кожна HTML сторінка (index.html, about.html тощо) є окремим компонентом представлення.

– Картки товарів, послуг, абонементів, замовлень, записів – уніфіковані елементи для відображення структурованої інформації.

– Форми реєстрації, входу, оформлення замовлення, запису на абонемент, додавання товару в адмін-панелі. Включають поля вводу, кнопки та механізми валідації.

– Компоненти для відображення сповіщень та запитів на підтвердження дій.

Модуль управління станом клієнта передбачає:

– Зберігання та управління токеном автентифікації (authToken) в localStorage.

– Динамічне оновлення інтерфейсу (наприклад, хедера) залежно від стану авторизації.

Модуль взаємодії з API (API Client) передбачає:

– Набір JavaScript функцій, що використовують Workspace API для асинхронної взаємодії з серверними ендпоінтами (отримання даних, відправка форм, оновлення, видалення).

– Обробка відповідей від сервера та можливих помилок.

Модуль специфічної логіки сторінок передбачає:

– Логіка кошика (cart.html) – завантаження, відображення, оновлення кількості, видалення товарів.

– Логіка каталогу спортхарчу (nutrition.html) – завантаження товарів, фільтрація на стороні клієнта, додавання до кошика.

– Логіка оформлення замовлення (pay.html) – взаємодія з API Нової Пошти, збір даних форми, відправка замовлення.

– Логіка профілю (profile.html) – завантаження даних користувача, логіка виходу, міні-гра.

– Логіка адмін-панелі (admin.html) – управління вкладками, завантаження та відображення списків товарів, замовлень, записів; обробка форм додавання та логіка видалення елементів.

Серверна частина, реалізована на Node.js та Express.js, включає:

Модуль маршрутизації (Routing):

– Визначення та обробка HTTP-запитів для кожного API ендпоінту.

Модуль автентифікації та авторизації:

– Реалізація реєстрації та входу користувачів.

– Генерація та перевірка JWT .

Модуль бізнес-логіки (Services/Controllers):

– Обробка даних, отриманих від клієнта.

– Взаємодія з моделями даних (Mongoose) для операцій з базою даних.

– Реалізація специфічної логіки для кожної операції (наприклад, розрахунок загальної суми замовлення, оновлення кількості товару в кошику).

Модуль взаємодії з базою даних (Data Access Layer):

– Визначення схем та моделей Mongoose (User, Product, Order).

– Виконання операцій – Create, Read, Update, Delete з колекціями MongoDB.

Модуль роботи з файловою системою:

– Читання та запис даних у файли orders.json та enrollments.json для дублювання або альтернативного зберігання інформації.

Взаємодія між клієнтом та сервером відбувається через чітко визначені API ендпоінти. Клієнт надсилає запити, сервер їх обробляє, взаємодіє з базою даних або файловою системою, та повертає відповідь (зазвичай у форматі JSON), яку клієнт потім використовує для оновлення інтерфейсу.

2.3 Проєктування інтерфейсу користувача (User Interface Design)

Проєктування інтерфейсу користувача (UI) для веб-сайту "FitForge" було спрямоване на створення зрозумілого, візуально привабливого та функціонального середовища для взаємодії користувачів з системою. Основна

увага приділялася забезпеченню легкості навігації, чіткості представлення інформації та узгодженості дизайну на всіх сторінках.

2.3.1 Загальні принципи проєктування UI

В основі дизайну цього інтерфейсу – користувач. Відштовхувались від потреб клієнтів фітнес-клубу, як існуючих, так і потенційних. Головне завдання – зробити все максимально просто, щоб людина могла виконати ключові дії без зайвих роздумів.

Звідси випливає і підхід до навігації. Вся структура сайту, всі ці меню та сторінки, спроєктовані так, аби користувач інтуїтивно розумів, куди йому рухатись далі. Ідея в тому, щоб йому взагалі не знадобились інструкції. Він зайшов і все зрозуміло.

А щоб досвід був цілісним, жорстко дотримувались єдиного стилю для "FitForge". Кольори, шрифти, іконки, навіть те, як виглядають картки з послугами чи кнопки- все підпорядковано одній логіці. Це створює впізнаваність, той самий образ бренду, який не сплутаєш ні з чим іншим. І цей образ- він важливий.

При цьому найважливіші елементи свідомо витягнуті на передній план. Заголовки, кнопки заклику до дії, якісь акційні пропозиції — все це одразу кидається в очі за рахунок розміру, кольору, розташування.

Візуально зупинились на сучасному "темному" дизайні. Яскраві жовті акценти, градієнтні фони і модний ефект "скла" на картках — все це створює приємну картинку і добре лягає на спортивну тематику. Вийшло динамічно, а не як черговий нудний корпоративний сайт. Основа є, а це головне.

2.3.2 Проєктування ключових сторінок

Головна сторінка (index.html):

– Верхня частина має динамічний перший екран з фоновим відео, затемненням для контрасту, великим заголовком "FitForge", слоганом та CTA-кнопкою "Дізнатись Більше".

– Секція "Розклад Занять" – це блок із заголовком та іконкою, розклад представлений картками днів тижня/типів занять; тематичне зображення.

– Секція "Чому саме FitForge?" має три візуально відокремлені картки з іконками та описом переваг клубу; підсилююче іміджеве зображення.

Сторінка "Про нас" (about.html):

– Верхня частина аналогічна головній, з тематичним фоновим зображенням та відповідним текстом.

– Ключові секції: "Наша Місія", "Наші Цінності" (грід-розкладка з трьома картками), "Наша Історія" (текст та зображення), "Чому Обирають Нас?" (грід-розкладка з трьома картками). Кожна секція з великим заголовком та тематичною іконкою.

Сторінка "Послуги" (services.html):

– Основні секції: "Фітнес-Тестування" (інформаційна картка), "Наші Послуги" (три картки: персональні тренування, групові заняття, тренажерний зал з іконками), "Наші Абонементи" (картки для типів абонементів з ціною, переліком послуг (іконки "галочка"/"хрестик") та CTA-кнопкою "Долучитися" з модальним вікном).

Сторінка "Спортхарч" (nutrition.html):

Структура: Двоколонковий макет.

– Ліва колонка – "Скляна" картка-панель фільтрів (пошук, ціна, тип).

– Права колонка – сітка товарів у "скляних" картках (зображення, назва, опис, ціна, CTA "Додати до кошика") з вертикальною прокруткою та кастомним скролбаром.

Сторінка "Контакти" (contact.html):

Структура: Двоколонковий макет .

– Ліва колонка – "Скляні" картки з контактною інформацією (адреса, телефон, email, графік) з іконками.

– Права колонк – "Скляна" картка з інтегрованою картою Google Maps та секцією "Як дістатися?".

– Додатково секція "Ми в Соціальних Мережах" (картка з іконками-посиланнями).

Сторінки "Вхід" (login.html) та "Реєстрація" (register.html):

– Дизайн має відцентровану по вертикалі та горизонталі "скляну" картку-форму на всю висоту видимої області.

– Заголовок з тематичною іконкою, поля вводу з іконками, кнопка відправки, місце для повідомлень про помилки, посилання для переходу між формами.

Сторінка "Профіль користувача" (profile.html):

– Доступ тільки для авторизованих користувачів.

– Основна картка з інформацією користувача (аватар-іконка, ім'я, email) та кнопкою "Вийти з акаунту". Окрема розгортна картка для міні-гри "Вгадай число".

Сторінка "Кошик" (cart.html):

– Сітка карток товарів у кошику (зображення, назва, ціна, керування кількістю, кнопка "Видалити").

– Кнопка "Оформити замовлення" (якщо кошик не порожній) або повідомлення про порожній кошик.

Сторінка "Оформлення замовлення" (pay.html):

– Дизайн це одна велика "скляна" картка-форма.

– "Ваші товари" (список з прокруткою, загальна сума), "Контактна інформація", "Доставка Новою Поштою" (випадаючі списки).

– Кнопка "Підтвердити замовлення", модальне вікно про успішне оформлення.

Сторінка "Адмін-панель" (admin.html):

– Три вкладки: "Управління Товарами", "Перегляд Замовлень", "Записи на Абонементи".

– Форма додавання, список існуючих товарів з кнопками видалення.

– Список карток замовлень з деталями та кнопкою видалення.

– Список карток записів з деталями та кнопкою видалення.

Ключові елементи інтерфейсу та їх дизайн (рисунок 2.2).

Рисунок 2.2 – Стилі елементів інтерфейсу

Такий підхід до проєктування UI спрямований на створення функціонального, зручного та візуально привабливого веб-додатку, який відповідає сучасним трендам веб-дизайну та потребам користувачів фітнес-клубу.

2.4 Проєктування алгоритмів роботи основних функцій

На цьому етапі детально розглядаються алгоритми, що лежать в основі ключових функціональних можливостей системи "FitForge". Опис алгоритмів буде представлено у текстовому форматі, що пояснює послідовність дій для кожної функції. У кваліфікаційній роботі доцільно було б доповнити цей опис блок-схемами або псевдокодом для кращої візуалізації.

2.4.1 Алгоритм реєстрації нового користувача

Розберемо потік даних у системі. Розгляд питання проходить навколо кількох ключових алгоритмів, що забезпечують взаємодію клієнта і сервера.

Все починається на фронтенді, на сторінці реєстрації. Користувач заповнює форму, і після кліку на кнопку спрацьовує JavaScript. Перший етап це валідація клієнта. Скрипт перевіряє, чи всі поля заповнені та чи має пароль хоча б 6 символів. Якщо щось не так юзер одразу бачить помилку, і ніякий запит на сервер не йде. Якщо все гаразд, дані пакуються в JSON і летять асинхронним POST-запитом на ендпоінт /register. Сервер обробляє дані, створює новий обліковий запис і повертає відповідь. Успіх чи помилка- користувач бачить відповідне повідомлення.

Процес входу – це подібний процес на реєстрацію. Знову форма, знову клієнтська валідація на заповненість полів. Далі такий самий POST-запит, але вже на /login. Сервер не просто перевіряє логін і пароль. Паролі в базі не зберігаються у відкритому вигляді – тільки хеші. Якщо хеш збігається, сервер генерує JWT-токен. Цей токен, що містить ID та ім'я користувача, повертається на клієнт і зберігається в localStorage. Саме цей токен і є ключем до всіх подальших дій в системі, він буде додаватись до заголовків усіх захищених запитів.

Коли користувач залогінений, він може, наприклад, додавати товари до кошика. Кожен такий запит спершу проходить через authenticateToken middleware. Цей проміжний обробник перевіряє валідність токена і, якщо все добре, додає дані користувача в об'єкт запиту req.user. Далі логіка: сервер отримує ID товару, знаходить кошик юзера в базі, перевіряє, чи є там вже такийтовар. Якщо є – просто збільшує кількість. Якщо немає – додає новий об'єкт. Все зберігається, і користувач отримує оновлений стан кошика.

Оформлення замовлення. Процес починається з authenticateToken. Система бере кошик користувача з БД, переконується, що він не порожній, і на його основі розраховує загальну суму та формує остаточний список товарів. Далі створюється новий документ в колекції Order в MongoDB. Процес замовлення

додатково дублюється в локальний orders.json. Після успішного збереження кошик користувача в базі очищується.

Далі адмін-панель. Адмін хоче подивитись замовлення. Він заходить на admin.html, клікає на відповідну вкладку. Скрипт робить GET-запит на ендпоінт /api/order, отримує список усіх замовлень, сортує їх від новіших до старіших і динамічно рендерить HTML-картки для кожної. Прямо звідси ж адмін може і видалити замовлення – на кнопці висить обробник, що після підтвердження відправляє DELETE-запит з потрібним ID. Після видалення список автоматично оновлюється.

2.5 Проєктування бази даних

Для ефективного зберігання та управління даними веб-додатку "FitForge" було обрано документо-орієнтовану NoSQL систему управління базами даних - MongoDB. Цей вибір обґрунтований гнучкістю схеми даних, яку надає MongoDB, що є особливо корисним для проектів, де структура даних може розвиватися, наприклад, при додаванні нових характеристик товарів або інформації про користувачів. MongoDB також добре масштабується та інтегрується з Node.js за допомогою бібліотеки Mongoose.

Проєктування бази даних включало визначення основних колекцій, їх структури (схем) та взаємозв'язків між ними.

Обґрунтування вибору MongoDB

Вибір MongoDB як основної СУБД для проекту "FitForge" був зумовлений наступними факторами:

– Гнучкість схеми – MongoDB є schemaless базою даних. Це означає, що документи в одній колекції можуть мати різний набір полів. Це корисно для таких сутностей, як "Товари", де різні типи спортивного харчування можуть мати різні набори характеристик, або для "Користувачів", де може з'явитися необхідність додати нові поля профілю без зміни структури всієї таблиці, як це було б у реляційних БД.

– Дані зберігаються у форматі, схожому на JSON (BSON – Binary JSON), що природно відображається на об'єкти в JavaScript. Це спрощує розробку на Node.js, оскільки не потребує складних перетворень між об'єктною моделлю додатку та реляційною моделлю бази даних.

– MongoDB дозволяє легко зберігати масиви та вбудовані документи всередині одного документа. Це було використано для реалізації кошика користувача (cart), який є масивом товарів, вбудованим безпосередньо в документ користувача.

– MongoDB підтримує горизонтальне масштабування, що може бути важливим для майбутнього росту додатку.

– Для багатьох типів запитів, особливо тих, що стосуються читання цілих документів або роботи з вбудованими даними, MongoDB може забезпечити високу продуктивність.

– Наявність потужної бібліотеки Mongoose для Node.js значно спрощує роботу з MongoDB, надаючи інструменти для визначення схем, валідації даних, побудови запитів та управління зв'язками між моделями.

Логічна структура бази даних (Колекції та Схеми)

База даних auth_db містить три основні колекції, що відповідають ключовим сутностям системи: users, products та orders. Колекція users зберігає інформацію про зареєстрованих користувачів. Схема зображена на рисунку 2.3.

Рисунок 2.3 – Схема колекції users

Колекція products зберігає каталог товарів спортивного харчування. Схема зображена на рисунку 2.4.

Рисунок 2.4 – Схема колекції продукту

Колекція ordes зберігає інформацію про оформлені замовлення. Схема зображена на рисунку 2.5.

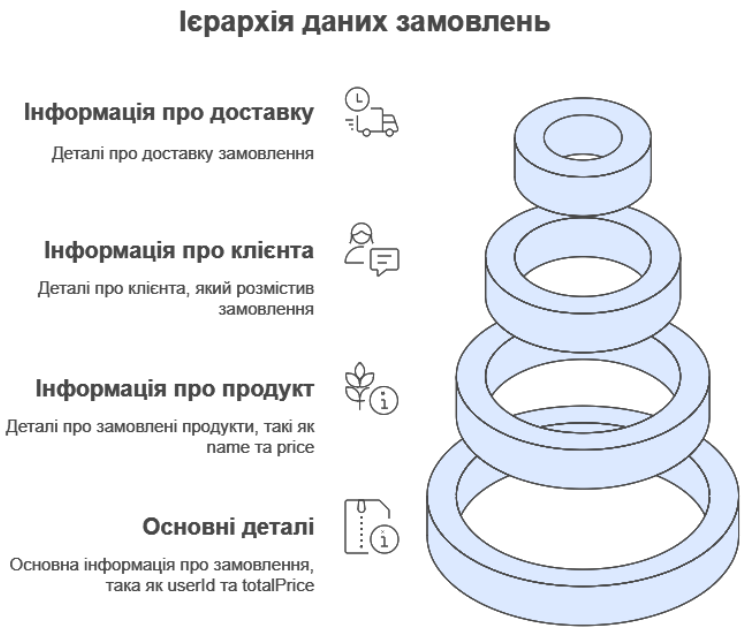

Рисунок 2.5 – Схема колекції замовлень

У MongoDB є, по суті, два головні способи пов'язувати дані між собою, і використовуються обидва.

Перший передбачає коли дані тісно пов'язані і майже завжди потрібні разом, просто вкладаються один в одного. Найкращий приклад – це кошик користувача. Замість того, щоб тримати його десь окремо, масив товарів з кошика вбудований прямо в документ користувача. Дуже швидко.

Але вбудовувати все підряд – не найкращий спосіб, особливо коли зв'язки складніші, як "один-до-багатьох". Тут використовуємо другий підхід – посилання. По суті, просто зберігаємо ID одного документа в іншому. Наприклад, у кожному замовленні є поле userId, яке посилається на конкретного користувача, а в списку товарів замовлення кожен товар посилається на відповідний документ у колекції продуктів. Mongoose полягає в методі populate(), який дозволяє легко "підтягнути" всі ці пов'язані дані одним запитом.

Також дублюються частини даних у звичайні JSON-файли на сервері. Це стосується замовлень та записів на абонементи. Тобто, окрім запису в MongoDB, створюється ще й копія у файлах orders.json та enrollments.json.

Зберігати транзакційні дані у файлах – це прямий шлях до проблем з одночасним доступом, цілісністю даних і продуктивністю, як тільки обсяги зростуть. Для цього проєкту це скоріше тимчасове рішення, механізм для резервування або просто спосіб спростити деякі адмінські задачі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

На даному етапі передбачено створення програмних засобів системи "FitForge" як повноцінного функціонально веб-додатку на основі розробленого на попередніх етапах проєкту. Описуємо використане програмне та технічне забезпечення, процес налаштування середовища розробки, конструювання програмного продукту з описом реалізації ключових функцій, створення бази даних, розробку користувацького інтерфейсу та проведення тестування.

3.1.Необхідне системне, прикладне та інструментальне програмне забезпечення

Для роботи над "FitForge" облаштували доволі стандартний, але ефективний робочий простір. Основним редактором коду — від HTML до серверного JavaScript — був Visual Studio Code. Коли процес доходив до тестування API, без Postman було не обійтись. Ну а вся графіка та дизайн-макети народжувались у Figma та Adobe Photoshop.

На бекенді основна мова програмування JavaScript. Вся логіка працює на Node.js, а поверх нього розгорнуто веб-сервер на Express.js. Він обробляє всі API-запити. Для зберігання даних було обрано MongoDB, а щоб зручно користуватися базою даних, перевіряти колекції та документи, використовується графічний клієнт MongoDB Compass.

Всю взаємодію з MongoDB з боку Node.js взяла на себе бібліотека Mongoose – вона дозволяє визначати схеми даних і значно спрощує запити. Безпека – це пріоритет, тому для хешування паролів використовується Bcryptjs, а для системи автентифікації та авторизації – jsonwebtoken (JWT).

На фронтенді все стандартно. Структура – це HTML5, а стилі – CSS3. Але основну роботу зі стилізації виконує утилітарний фреймворк Tailwind CSS, який дозволив швидко зібрати кастомний інтерфейс. Вся динаміка, інтерактивність та спілкування з сервером реалізовані на чистому JavaScript. Для візуального оформлення підключено легку бібліотеку іконок Feather Icons через CDN.

Інтегровано й зовнішній сервіс – API Нової Пошти. Він використовується на сторінці оформлення замовлення, щоб динамічно завантажувати списки міст та відділень.

3.2.Встановлення та налаштування необхідного технічного та програмного забезпечення

Процес розгортання робочого середовища для розробки та запуску системи "FitForge" включав наступні кроки:

1) Встановлення Node.js та npm.

Завантаження LTS (Long Term Support) версії Node.js з офіційного сайту для відповідної операційної системи (рисунок 3.1).

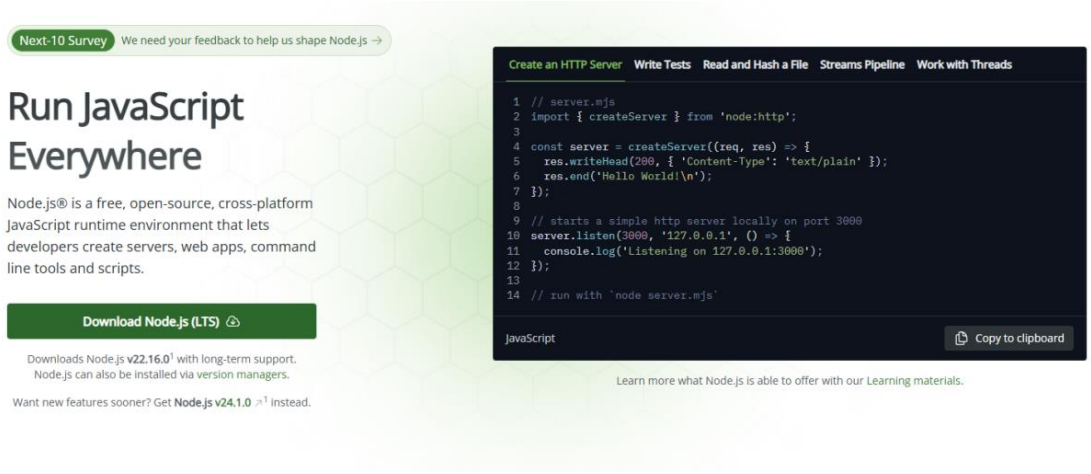

Рисунок 3.1 – Встановлення LTS

Встановлення Node.js, що автоматично включає npm. Щоб перевірити успішність встановлення Node.js потрібно ввести node -v та npm -v у командному рядку (рисунок 3.2).

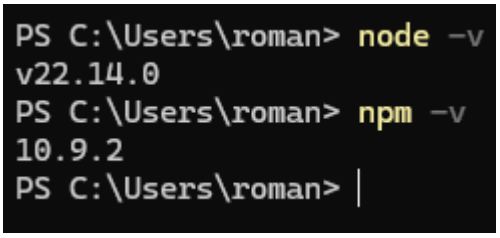

Рисунок 3.2 – Перевірка встановлення

2) Встановлення MongoDB.

Завантаження MongoDB Community Server з офіційного сайту. Спочатку нам потрібно увійти у профіль (рисунок 3.3).

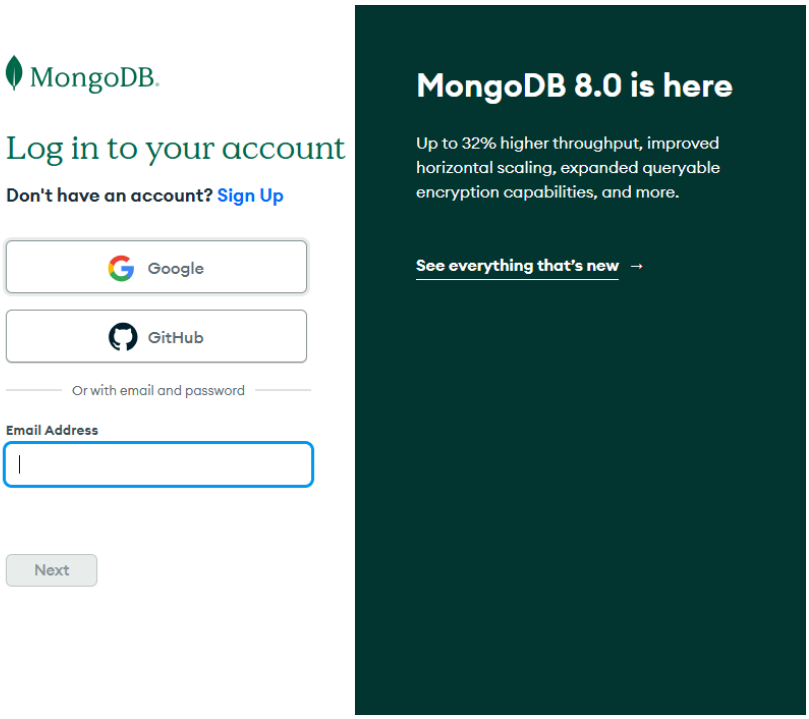

Рисунок 3.3 – Вхід у профіль

Скачуємо потрібну версію та вибираємо свою платформу (рисунок 3.4).

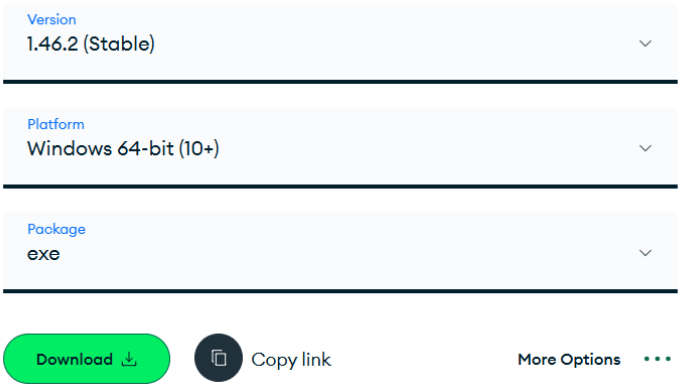

Рисунок 3.4 – Скачуємо MongoDB

Встановлення СУБД згідно з інструкціями для обраної ОС.

3) Налаштування проекту передбачає:

– Клонування репозиторію проекту або створення кореневої папки проекту.

– Створення файлу package.json за допомогою команди npm init -y (або вручну), який описує проект та його залежності.

– Встановлення серверних залежностей за допомогою npm (рисунок 3.5):

```
npm install express mongoose bcryptjs jsonwebtoken cors body-parser
```

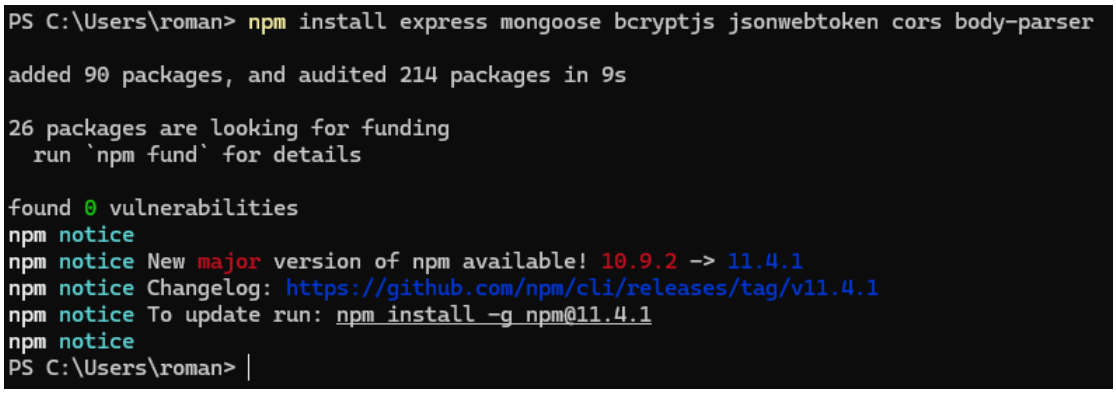

Рисунок 3.5 – Встановлення серверних залежностей

Ці залежності зберігаються у файлі package.json та завантажуються у папку node_modules.

4) Структура проекту.

Було організовано наступну структуру папок та файлів (приклад на основі наданого коду):

```
fff
├── public/
│   ├── images/ для зберігання зображень
│   │   ├── i1.png, a1.jpg, ...
│   ├── videos/
│   │   └── i1.mp4
│   ├── index.html
│   ├── about.html
│   ├── services.html
│   ├── nutrition.html
│   ├── contact.html
│   ├── login.html
│   ├── register.html
│   ├── profile.html
│   ├── cart.html
│   ├── pay.html
│   └── admin.html - адмінка
├── server.js - сервер сайту
├── orders.json - файл для зберігання замовлень
    ├── enrollments.json -файл для абонементів
├── package.json
└── package-lock.json
```

Клієнтські HTML, CSS (через Tailwind CDN) та JavaScript файли розміщуються у папці public, яка налаштована в server.js для віддачі статичного контенту.

5) Запуск серверної частини.

Сервер запускається командою node server.js з кореневої папки проекту. Після запуску сервер починає слухати запити на вказаному порту (за замовчуванням 5000) (рисунок 3.6).

```
PS C:\Users\roman> cd fff
PS C:\Users\roman\fff> node server.js
(node:38080) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.
js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:38080) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since
 Node.js Driver version 4.0.0 and will be removed in the next major version
Server running on port 5000
```

Рисунок 3.6 – Запуск сервера

6) Доступ до клієнтської частини.

Клієнтська частина доступна через веб-браузер за адресою http://localhost:5000(головна) (рисунок 3.7).

http://localhost:5000/назва_файлу.html для інших сторінок (рисунок 3.8).

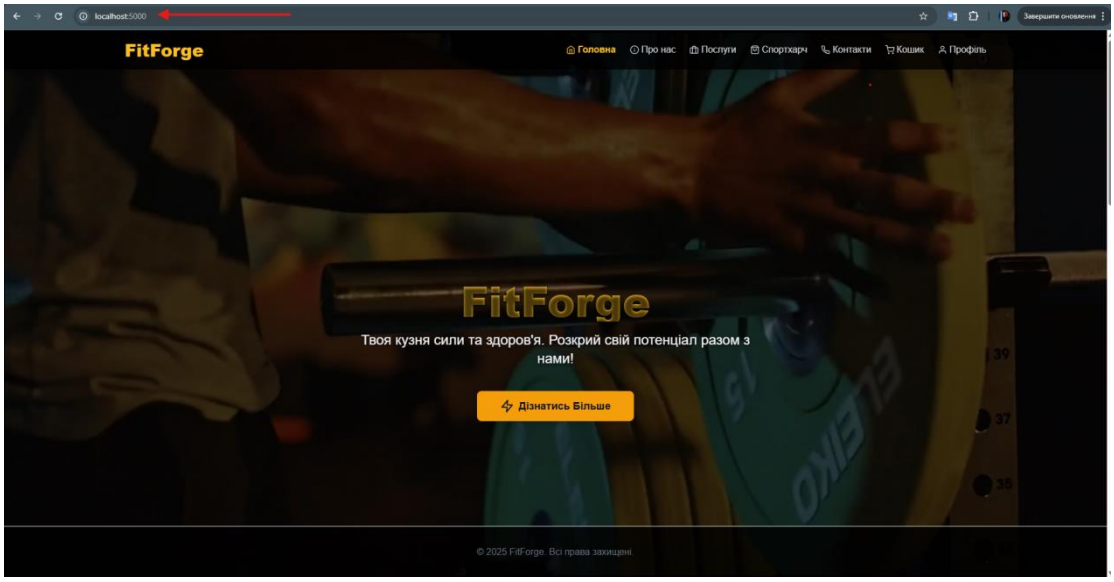

Рисунок 3.7 – Запуск сторінки http://localhost:5000/

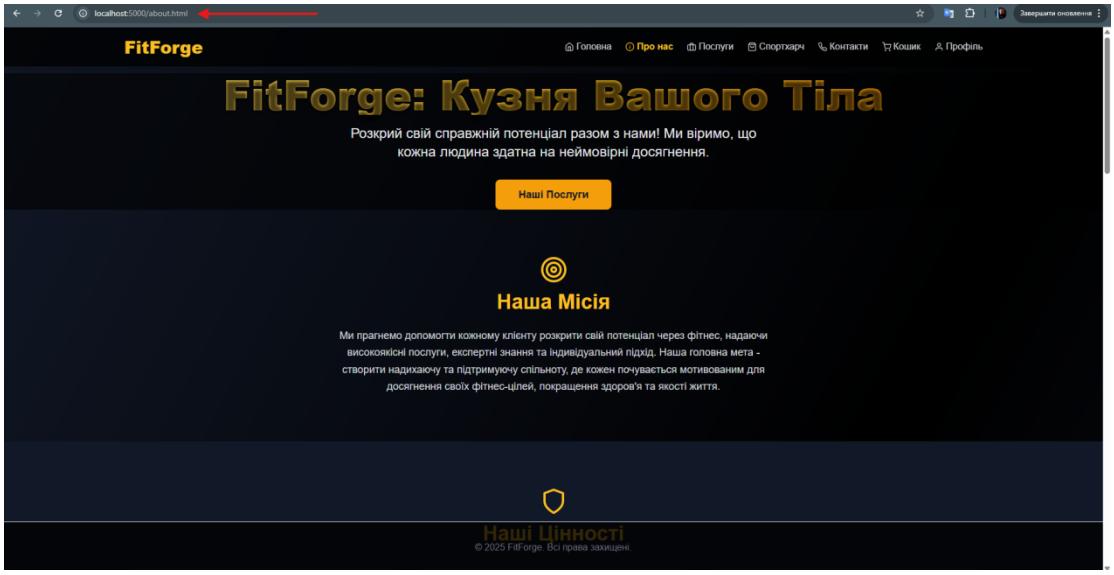

Рисунок 3.8 – Запуск сторінки http://localhost:5000/about.html

7) Налаштування API ключів.

Для роботи з API Нової Пошти, API ключ NP_API_KEY ("5175f23d5cbd7fc490599d5a25624413") було вказано безпосередньо у клієнтському JavaScript коді (pay.html). У реальному проекті такий ключ краще зберігати на сервері або в конфігураційних файлах, недоступних клієнту напряму, і робити запити до API Нової Пошти через бекенд, щоб захистити ключ. Однак, для цілей дипломного проекту та спрощення клієнтської логіки було обрано такий підхід.

8) Файли даних JSON.

Файли orders.json та enrollments.json створюються сервером автоматично при першому збереженні відповідних даних, якщо вони відсутні. Важливо забезпечити права на запис для сервера у директорію, де знаходиться server.js.

Це середовище дозволило ефективно розробляти, тестувати та демонструвати функціонал веб-додатку "FitForge".

3.3. Процес розробки програмного продукту.

Процес розробки веб-додатку "FitForge" був розділений на два основні напрямки:розробка серверної частини (back-end) та розробка клієнтської частини (front-end). Обидві частини розроблялися паралельно з регулярною інтеграцією для забезпечення коректної взаємодії.

3.3.1. Розробка серверної частини (Back-End)

Серверна частина додатку "FitForge" реалізована на платформі Node.js з використанням веб-фреймворку Express.js. Вона відповідає за обробку бізнес-логіки, взаємодію з базою даних MongoDB, автентифікацію користувачів, авторизацію доступу до ресурсів та надання API для клієнтської частини.

Налаштування сервера Express.js та Middleware передбачає наступне.

Основою серверної частини є файл server.js. На початку файлу відбувається підключення необхідних модулів:

```
JavaScript
const express = require('express');const mongoose =
require('mongoose');const bcrypt = require('bcryptjs');const jwt =
require('jsonwebtoken');const cors = require('cors');const
bodyParser = require('body-parser');const path =
require('path');const fs = require('fs');
const app = express(); // Створення екземпляру Express-
додатку
```

Це наступні модулі:

– Express - основний фреймворк для побудови веб-додатків та API.

– Mongoose – ODM-бібліотека для роботи з MongoDB, що забезпечує зручний інтерфейс для моделювання даних та взаємодії з базою.

– Bcryptjs — бібліотека для хешування паролів, що забезпечує їх безпечне зберігання.

– Jsonwebtoken – використовується для створення та перевірки JSON Web Tokens (JWT) для реалізації системи автентифікації.

– Cors – дозволяє обробляти запити з інших доменів, що важливо для взаємодії між фронтендом та бекендом, особливо під час розробки.

– body-parser – Middleware для розбору тіла HTTP-запитів.

– path та fs – вбудовані модулі Node.js для роботи зі шляхами до файлів та файловою системою відповідно. Вони використовуються для віддачі статичних файлів (клієнтська частина) та для роботи з файлами orders.json і enrollments.json.

Після ініціалізації додатку Express налаштовуються основні middleware:

```
JavaScript
app.use(cors()); // Дозволяє CORS-запити
app.use(bodyParser.json()); // Розбір JSON-тіла запитів
app.use(express.static(path.join(__dirname, 'public'))); //
Віддача статичних файлів з папки 'public'
```

express.static налаштовує Express на віддачу статичних файлів (HTML, CSS, клієнтський JavaScript, зображення, відео) з папки public. Це дозволяє клієнтській частині сайту бути доступною через браузер.

Middleware для автентифікації (authenticateToken) передбачає наступне.

Для захисту певних маршрутів API було створено middleware функцію authenticateToken. Вона перевіряє наявність та валідність JWT-токена в заголовку Authorization кожного запиту:

```
JavaScript
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader?.split(' ')[1]; // Очікується
формат "Bearer TOKEN"
    if (!token) return res.status(401).json({ error:
'Відсутній токен' });

    jwt.verify(token, 'secretkey', (err, user) => { //
'secretkey' - секретний ключ для підпису/перевірки
        if (err) {
            console.error("Помилка верифікації токена:",
err.message);
            return res.status(403).json({ error: 'Невірний
токен' });
        }
        req.user = user; // Додавання розшифрованих даних
користувача (наприклад, id) до об'єкту запиту
        next(); // Передача управління наступному обробнику
    });
}
```

Ця функція спочатку перевіряє, чи є заголовок Authorization. Якщо так, вона витягує токен. Якщо токен відсутній, повертається помилка 401 (Unauthorized). Якщо токен є, він перевіряється за допомогою jwt.verify. Використовується секретний ключ 'secretkey' (у реальному додатку цей ключ має бути складним та зберігатися безпечно, наприклад, у змінних середовища). У разі успішної перевірки, інформація про користувача (зазвичай його ID, який був закодований у токен при логіні) додається до об'єкта req, і керування передається далі. В іншому випадку повертається помилка 403 (Forbidden).

Підключення до бази даних MongoDB здійснюється за допомогою Mongoose:

```
JavaScript
mongoose.connect('mongodb://127.0.0.1:27017/auth_db', { // URL підключення до локальної БД
    useNewUrlParser: true,
    useUnifiedTopology: true,});
```

Auth_db – це назва бази даних. Параметри useNewUrlParser та useUnifiedTopology є стандартними для уникнення попереджень про застарілі функції драйвера.

Для взаємодії з колекціями в MongoDB визначаються схеми та моделі Mongoose:

User (Користувач) подано наступним чином:

```
JavaScript
const userSchema = new mongoose.Schema({
    username: String,
    email: { type: String, unique: true, required: true }, // Додано унікальність та обов'язковість
    password: { type: String, required: true }, // Додано обов'язковість
    cart: [        {            productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
                quantity: { type: Number, default: 1 }
            }
    ],    });const User = mongoose.model('User', userSchema);
```

Схема користувача включає ім'я, унікальний email, хешований пароль та кошик товарів, який є масивом об'єктів з посиланням на товар (Product) та його кількістю.

Product (Товар) подано наступним чином:

```
JavaScript
const productSchema = new mongoose.Schema({
    name: { type: String, required: true },
    type: { type: String, required: true }, // Категорія
товару
    price: { type: Number, required: true },
    description: String,
    imageUrl: String, // Шлях до зображення товару
    // Можна додати інші характеристики: вага, виробник,
наявність тощо.
});const Product = mongoose.model('Product', productSchema);
```

Схема товару містить назву, тип , ціну, опис та URL зображення.

Order (Замовлення) представлено:

```
JavaScript
const orderSchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, ref:
'User', required: true },
    products: [ // Змінено з 'items' на 'products' для
відповідності з JSON, але можна залишити 'items'
        {
            productId: { type:
mongoose.Schema.Types.ObjectId, ref: 'Product' },
            name: String, // Назва товару для зручності
            quantity: Number,
            price: Number // Ціна за одиницю на момент
замовлення
        }
    ],
    totalPrice: { type: Number, required: true },
    customerInfo: {
```

```
            fullName: String,
            email: String,
            phone: String,
            comment: String,
        },
        shippingInfo: {
            cityRef: String,
            cityName: String,
            warehouseRef: String,
            warehouseName: String,
        },
        createdAt: { type: Date, default: Date.now }
    });const Order = mongoose.model('Order', orderSchema);
```

Схема замовлення пов'язана з користувачем, містить масив замовлених товарів (з кількістю та ціною на момент замовлення), загальну вартість, контактну інформацію клієнта, інформацію про доставку та дату створення.

Реалізація API маршрутів (Endpoints) передбачає наступне.

Серверна частина надає наступні API маршрути:

– Реєстрація користувача (POST /register):
– Приймає username, email, password.
– Перевіряє, чи не існує вже користувач з таким email.
– Хешує пароль за допомогою bcrypt.hash.
– Зберігає нового користувача в базі даних MongoDB.
– Повертає повідомлення про успішну реєстрацію.

Процес реєстрації користувача подано наступним чином :

```
JavaScript
app.post('/register', async (req, res) => {
    const { username, email, password } = req.body;
    // ... валідація ...
    try {
        // ... перевірка існуючого користувача ...
        const hashedPassword = await bcrypt.hash(password,
10);
```

```
            const user = new User({ username, email, password:
hashedPassword, cart: [] }); // Ініціалізація порожнього кошика
            await user.save();
            res.status(201).json({ message: 'Користувача успішно
зареєстровано!' }); // Статус 201 Created
        } catch (err) {
            // ... обробка помилок ...
            res.status(500).json({ error: 'Помилка реєстрації на
сервері' });
        }
    });
```

Вхід користувача (POST /login) передпачає:

– Приймає email, password.

– Знаходить користувача за email.

– Порівнює наданий пароль із хешованим паролем у базі даних за допомогою bcrypt.compare.

– У разі успіху генерує JWT (з ID користувача та терміном дії 1 година) та повертає його клієнту.

Процес логіну користувача подано наступним чином :

```
JavaScript
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    // ... валідація ...
    try {
        const user = await User.findOne({ email });
        if (!user) return res.status(401).json({ error:
'Користувача з таким email не знайдено' });

        const isMatch = await bcrypt.compare(password,
user.password);
        if (!isMatch) return res.status(401).json({ error:
'Невірний пароль' });
```

```
            const token = jwt.sign({ id: user._id, username:
user.username }, 'secretkey', { expiresIn: '1h' });
            res.json({ message: 'Успішний вхід', token });
        } catch (err) {
            // ... обробка помилок ...
            res.status(500).json({ error: 'Помилка входу на
сервері' });
        }
    });
```

Отримання даних профілю (GET /profile) передбачає наступне:

– Захищений маршрут (authenticateToken).

– Знаходить користувача за ID (отриманим з токена).

– Повертає дані користувача (без пароля), включаючи інформацію про товари в кошику.

API для товарів (/api/products) передбачає наступне:

POST /api/products: Додавання нового товару. Приймає дані товару (name, type, price, description, imageUrl). Створює новий документ Product і зберігає його в MongoDB. Приклад реалізації в server.js:

```
JavaScript
app.post('/api/products', async (req, res) => {
    const { name, type, price, description, imageUrl } =
req.body;
    if (!name || !type || !price) { // Валідація основних
полів
        return res.status(400).json({ error: 'Відсутні
обов’язкові поля товару (назва, тип, ціна)' });
    }
    try {
        const product = new Product({ name, type, price,
description, imageUrl });
        await product.save();
        res.status(201).json({ message: 'Товар додано',
product: product }); // Повертаємо створений продукт
    } catch (error) {
```

```
            console.error("Помилка додавання товару:", error);
            res.status(500).json({ error: 'Помилка додавання
товару' });
        }
    });
```

GET /api/products: Отримання списку всіх товарів з MongoDB. Приклад реалізації в server.js:

```
JavaScript

app.get('/api/products', async (req, res) => {
    try {
        const products = await Product.find();
        res.json(products);
    } catch (error) {
        console.error("Помилка отримання товарів:", error);
        res.status(500).json({ error: 'Помилка отримання
товарів' });
    }
});
```

DELETE /api/products/:id - видалення товару за ID з MongoDB. Приклад реалізації в server.js:

```
JavaScript
app.delete('/api/products/:id', async (req, res) => {
    try {
        const deleted = await
Product.findByIdAndDelete(req.params.id);
        if (!deleted) {
            return res.status(404).json({ message: 'Товар не
знайдено' });
        }
        res.json({ message: 'Товар видалено' });
    } catch (error) {
        console.error("Помилка видалення товару:", error);
```

```
            res.status(500).json({ message: 'Помилка при
видаленні товару' });
        }
    });
```

API для кошика (/api/cart) передбачає наступне:

Усі маршрути захищені authenticateToken. POST /api/cart: Додавання товару до кошика поточного користувача. Приклад реалізації в server.js:

```
    JavaScript
    app.post('/api/cart', authenticateToken, async (req, res) =>
{
        const { productId, quantity = 1 } = req.body;
        if (!productId) return res.status(400).json({ error:
'Вкажіть productId' });

        try {
            const user = await User.findById(req.user.id);
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

            const productExists = await
Product.findById(productId);
            if (!productExists) {
                return res.status(404).json({ error: 'Товар не
знайдено' });
            }

            const itemIndex = user.cart.findIndex(item =>
item.productId.toString() === productId);
            if (itemIndex > -1) {
                user.cart[itemIndex].quantity += quantity;
            } else {
                user.cart.push({ productId, quantity });
            }
            await user.save();
```

```
            const updatedUser = await
User.findById(req.user.id).populate('cart.productId'); //
Повертаємо оновлений кошик
            res.json({ message: 'Товар додано в кошик', cart:
updatedUser.cart });
        } catch (error) {
            console.error("Помилка додавання в кошик:", error);
            res.status(500).json({ error: 'Помилка додавання в
кошик' });
        }
    });
```

GET /api/cart — отримання вмісту кошика поточного користувача. Приклад реалізації в server.js:

```
    JavaScript
    app.get('/api/cart', authenticateToken, async (req, res) => {
        try {
            const user = await
User.findById(req.user.id).populate('cart.productId');
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
            res.json(user.cart);
        } catch (error) {
            console.error("Помилка отримання кошика:", error);
            res.status(500).json({ error: 'Помилка отримання
кошика' });
        }
    });
```

PUT /api/cart/:productId — оновлення кількості товару в кошику. Приклад реалізації в server.js:

```
    JavaScript
    app.put('/api/cart/:productId', authenticateToken, async
(req, res) => {
        const { quantity } = req.body;
        const { productId } = req.params;
```

```
        if (!quantity || quantity < 1) {
            return res.status(400).json({ error: 'Кількість має
бути не менше 1' });
        }
        try {
            const user = await User.findById(req.user.id);
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

            const itemIndex = user.cart.findIndex(item =>
item.productId.toString() === productId);
            if (itemIndex === -1) {
                return res.status(404).json({ error: 'Товар не
знайдено в кошику' });
            }
            user.cart[itemIndex].quantity = quantity;
            await user.save();
            const updatedUser = await
User.findById(req.user.id).populate('cart.productId');
            res.json({ message: 'Кількість товару оновлена',
cart: updatedUser.cart });
        } catch (error) {
            console.error("Помилка оновлення кількості товару:",
error);
            res.status(500).json({ error: 'Помилка оновлення
кількості товару' });
        }
    });
```

DELETE /api/cart/:productId — видалення конкретного товару з кошика. Приклад реалізації в server.js:

```
    JavaScript
    app.delete('/api/cart/:productId', authenticateToken, async
(req, res) => {
        const { productId } = req.params;
        try {
```

```
            const user = await User.findById(req.user.id);
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

            user.cart = user.cart.filter(item =>
item.productId.toString() !== productId);
            await user.save();
            const updatedUser = await
User.findById(req.user.id).populate('cart.productId');
            res.json({ message: 'Товар видалено з кошика', cart:
updatedUser.cart });
        } catch (error) {
            console.error("Помилка видалення товару з кошика:",
error);
            res.status(500).json({ error: 'Помилка видалення
товару з кошика' });
        }
    });
```

DELETE /api/cart/clear — повне очищення кошика користувача. Приклад реалізації в server.js:

```
    JavaScript
    app.delete("/api/cart/clear", authenticateToken, async (req,
res) => {
        try {
            const user = await User.findById(req.user.id);
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
            user.cart = [];
            await user.save();
            res.json({ message: "Кошик очищено" });
        } catch (error) {
            console.error("Помилка при очищенні кошика:", error);
            res.status(500).json({ error: 'Помилка при очищенні
кошика' });
        }
```

```
});
```

API для запису на абонемент (POST /api/enroll) передбачає наступне:

Захищений маршрут (authenticateToken).Приймає дані з форми (productId, firstName, lastName, age, phoneNumber).Зберігає запис у файл enrollments.json. Приклад реалізації в server.js у додатку А.

API для оформлення замовлення (POST /api/order) передбачає наступне:

Захищений маршрут (authenticateToken).Зберігає замовлення в MongoDB (колекція orders) та дублює у orders.json. Очищує кошик користувача. Приклад реалізації в server.js:

```
JavaScript
app.post('/api/order', authenticateToken, async (req, res) =>
{
    const { fullName, email, phone, comment, cityRef,
cityName, warehouseRef, warehouseName } = req.body;
    if (!fullName || !email || !phone || !cityRef ||
!cityName || !warehouseRef || !warehouseName) {
        return res.status(400).json({ error: 'Будь ласка,
заповніть усі обов’язкові поля для замовлення та доставки.' });
    }
    try {
        const user = await
User.findById(req.user.id).populate('cart.productId');
        if (!user || user.cart.length === 0) {
            return res.status(400).json({ error: 'Кошик
порожній або користувача не знайдено.' });
        }

        let totalPrice = 0;
        let productsForOrder = [];
        for (const item of user.cart) {
            // ... (розрахунок totalPrice та формування
productsForOrder) ...
            if (!item.productId) continue; // Пропускаємо,
якщо товар не знайдено
```

```
                totalPrice += item.productId.price *
item.quantity;
                productsForOrder.push({
                    productId: item.productId._id,
                    name: item.productId.name,
                    quantity: item.quantity,
                    price: item.productId.price
                });
            }
```

API для перегляду замовлень в адмін-панелі (GET /api/order) передбачає наступне:

Цей маршрут є у server.js. Він має читати orders.json та повертати масив замовлень.

```
JavaScript
// Приклад для server.js:
app.get('/api/order', authenticateToken, (req, res) => { /*
... логіка читання orders.json ... */ });
```

API для видалення замовлення в адмін-панелі (DELETE /api/order/:orderId):

Цей маршрут є у server.js. Він видаляє замовлення з MongoDB та orders.json.

```
JavaScript
// Приклад для server.js:
app.delete('/api/order/:orderId', authenticateToken, async
(req, res) => { /* ... логіка видалення ... */ });
```

API для перегляду записів на абонементи в адмін-панелі (GET /api/enrollments):

```
JavaScript
// Приклад для server.js:
app.get('/api/enrollments', authenticateToken, (req, res) =>
{ /* ... логіка читання enrollments.json ... */ });
```

API для видалення запису на абонемент в адмін-панелі (DELETE /api/enrollments/:enrollmentId):

```
JavaScript
// Приклад для server.js:
app.delete('/api/enrollments/:enrollmentId',
authenticateToken, (req, res) => { /* ... логіка видалення ... */
});
```

Зберігання даних у JSON файлах (orders.json, enrollments.json) подано нижче.

Для спрощення та демонстраційних цілей, окрім збереження замовлень в MongoDB, вони також дублюються у файл orders.json. Аналогічно, записи на абонементи зберігаються у файл enrollments.json. Це реалізовано за допомогою вбудованого модуля fs Node.js:

– При створенні нового замовлення/запису, спочатку читається вміст відповідного JSON файлу (fs.readFileSync).

– Якщо файл не існує або порожній, створюється новий порожній масив.

– Новий об'єкт додається до масиву.

– Оновлений масив зберігається назад у файл (fs.writeFile), використовуючи JSON.stringify з форматуванням (null, 2) для кращої читабельності файлу.

Такий підхід має свої недоліки у багатокористувацькому середовищі (проблеми з одночасним доступом до файлу), але для даного проекту він слугує як простий механізм зберігання та перегляду даних без прямого доступу до MongoDB, особливо для адмін-панелі, якщо GET-маршрути налаштовані на читання саме з цих файлів.

3.3.2. Розробка клієнтської частини (Front-End)

Клієнтська частина "FitForge" — це обличчя всього додатку. Саме її бачить користувач, і саме вона відповідає за всю візуальну складову та інтерактивність. Реалізувано її на класичному тріо веб-технологій: HTML, CSS та JavaScript.

В основі всього лежить розмітка HTML5. Кожна сторінка сайту, від головної до адмін-панелі, має чітку структуру з використанням тегів <header>, <main>, <footer> та інших.

За вигляд відповідає Tailwind CSS. Було обрано цей утилітарний фреймворк, щоб не писати гори власного CSS-коду. Замість цього вбудовано інтерфейс з готових класів-утиліт, що значно прискорило розробку і допомогло досягти єдиного стилю на всіх сторінках. Навіть популярний ефект "розмитого скла" (Glassmorphism) для карток реалізовано прямо за допомогою класів Tailwind. І хоча повна мобільна адаптація не була пріоритетом на цьому етапі, "mobile-first" фреймворк вже заклав для цього міцний фундамент.

Невеликі нюанси, що довершують вигляд, — це іконки з бібліотеки Feather Icons. Вони використовуються всюди: в кнопках, заголовках, пунктах меню, роблячи інтерфейс більш зрозумілим і живим.

Кожна сторінка сайту виконує свою унікальну функцію, всі вони об'єднані спільними компонентами, такими як хедер та футер. Це створює цілісний і передбачуваний досвід для користувача, незалежно від того, в якому розділі сайту він знаходиться.

Головна сторінка(index.html) передбачає.

Призначення це привітання користувача, представлення фітнес-клубу, заклик до дій. Її реалізація відбувається наступним чином:

Головна-секція реалізована з фоновим відео на весь екран. На відео накладено напівпрозорий чорний шар для кращої читабельності тексту. Текст (назва клубу, слоган) та кнопка "Дізнатись Більше" (що веде до секції абонементів на сторінці послуг) (рисунок 3.9).

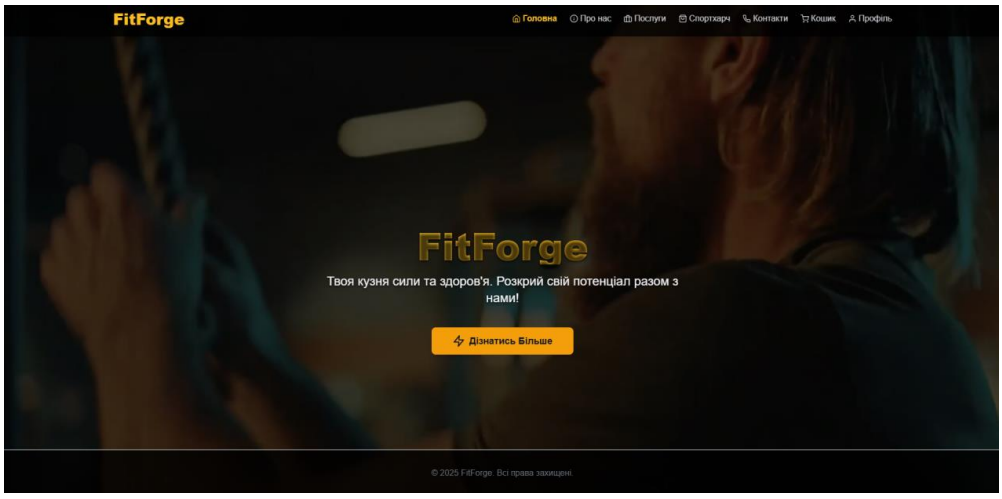

Рисунок 3.9 – Головна секція

Секція "Розклад Занять" відображає розклад групових занять по днях тижня. Кожен день або група занять представлена у вигляді картки з описом. Поруч розміщено зображення-заглушку (рисунок 3.10).

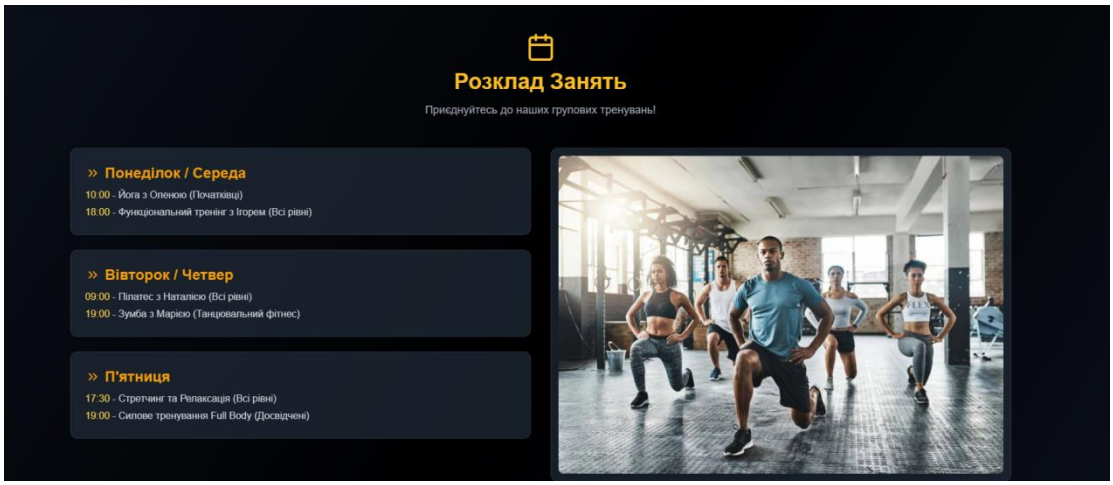

Рисунок 3.10 – С екція "Розклад занять"

Секція "Чому саме FitForge?" представляє ключові переваги клубу у вигляді трьох карток з іконками та описом. Також містить зображення-заглушку (рисунок 3.11).

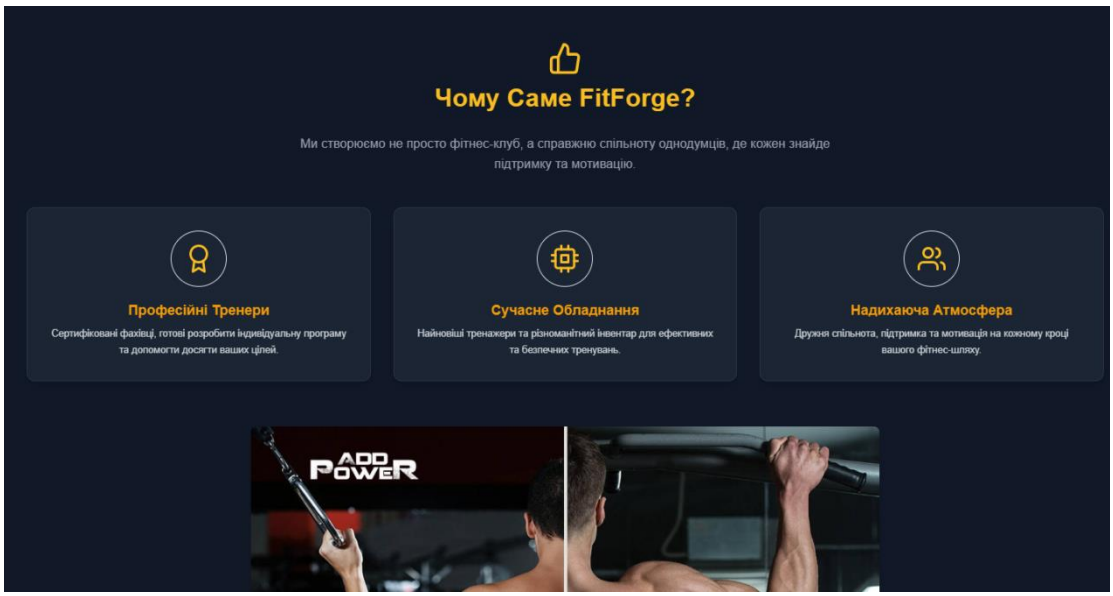

Рисунок 3.11 – Секція "Чому саме FitForge"?

Основний JavaScript на цій сторінці відповідає за уніфіковану навігацію в хедері та оновлення року в футері.

Про нас (about.html) передбачає розповісти про місію, цінності та історію фітнес-клубу.

Її реалізація відбувається наступним чином:

Головна-секція аналогічно до головної сторінки, але з іншим фоновим зображенням-заглушкою та текстом, що закликає дізнатися більше про послуги (рисунок 3.12).

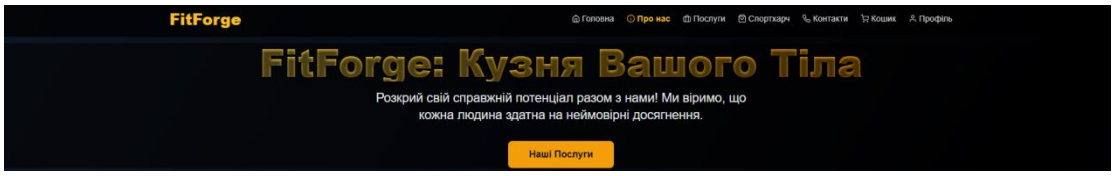

Рисунок 3.12 – Головна секція

"Наша Місія", "Наші Цінності", "Наша Історія", "Чому Обирають Нас?" кожна секція має заголовок з іконкою. Інформація представлена у текстових блоках та картках .Використовуються зображення-заглушки для ілюстрації (рисунок 3.13 та рисунок 3.14).

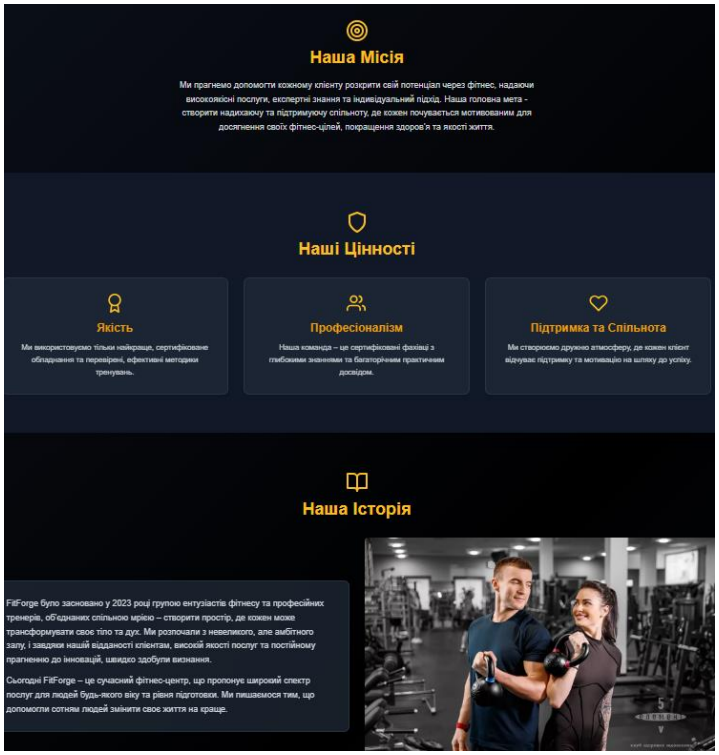

Рисунок 3.13 – Інші секції

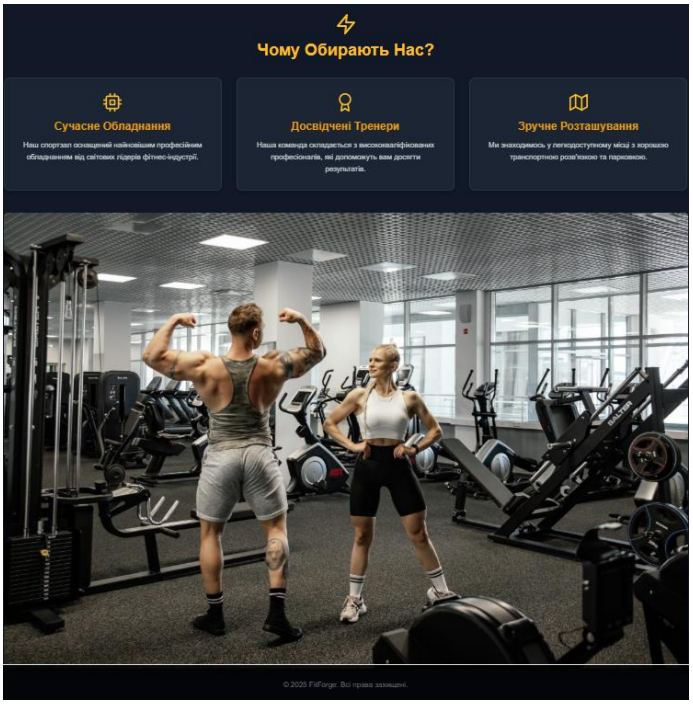

Рисунок 3.14 – Інші секції

JavaScript включає уніфіковану навігація та оновлення року.

Сервіси (services.html) передбачає детальний опис послуг клубу та абонементів, можливість записатися на абонемент.

Її реалізація відбувається наступним чином:

Головна секція з зображенням використовує CSS для адаптивного співвідношення сторін контейнера зображення. Текстовий оверлей містить заклик "Розпочати Зараз", який при авторизації користувача плавно прокручує до секції абонементів, а без авторизації – веде на сторінку входу (рисунок 3.15).

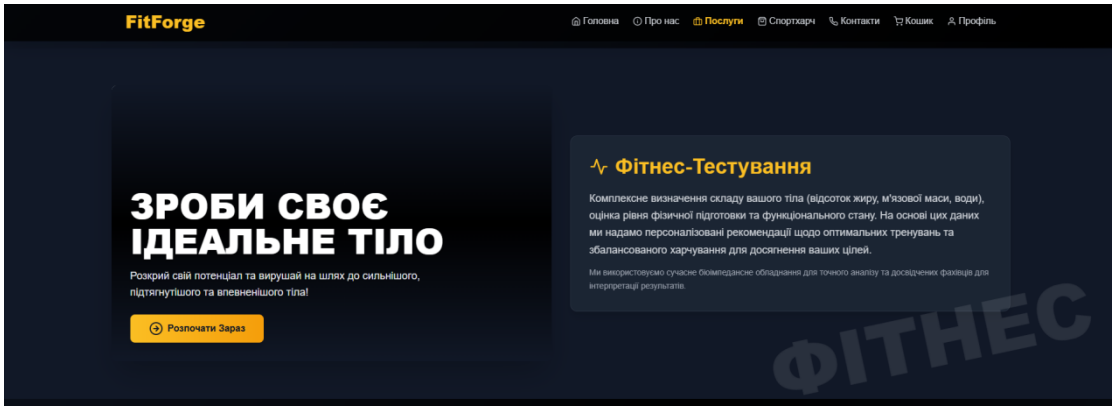

Рисунок 3.15 – Головна секція

Секція "Наші Послуги" це три картки з описом персональних тренувань, групових занять та залу, кожна з тематичною іконкою (рисунок 3.16).

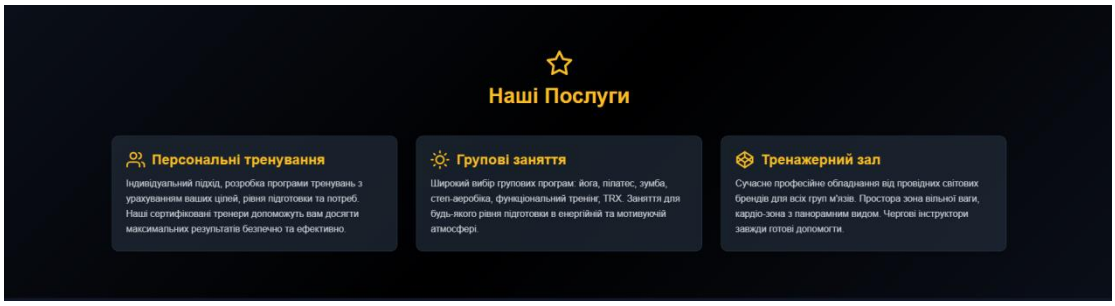

Рисунок 3.16 – Секція “Наші послуги”

Секція "Наші Абонементи" представлена трьома типами абонементів ("Старт", "Оптимум", "Преміум") у вигляді карток з цінами, переліком послуг (з іконками) та кнопками "Долучитися". Кнопка "Оптимум" має позначку "Популярний" (рисунок 3.17).

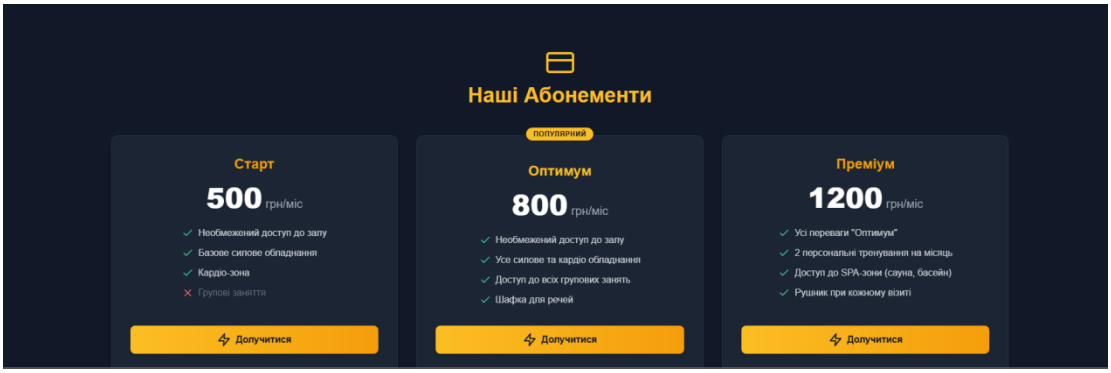

Рисунок 3.17 – Секція "Наші абонименти"

Модальне вікно запису реалізовано з використанням Tailwind CSS та JavaScript. Містить форму для введення імені, прізвища, віку, номера телефону та приховане поле для ID абонемента.

Назва обраного абонемента відображається у формі. Валідація полів та відправка даних на сервер здійснюється асинхронно (рисунок 3.18).

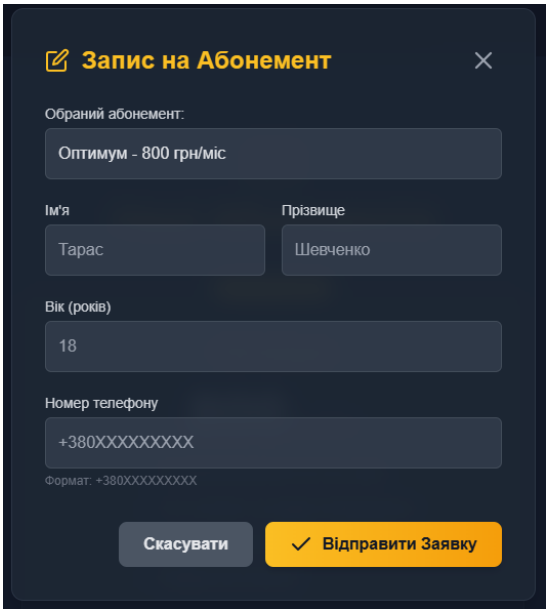

Рисунок 3.18 – Секція запису на абонимен

Модальне вікно сповіщень (notificationModal) — використовується для показу повідомлень про успішний запис або помилки.

Спортхарчування (nutrition.html) передбачає показ каталогу товарів спортивного харчування з можливістю фільтрації та додавання товарів до кошика. (рисунок 3.19).

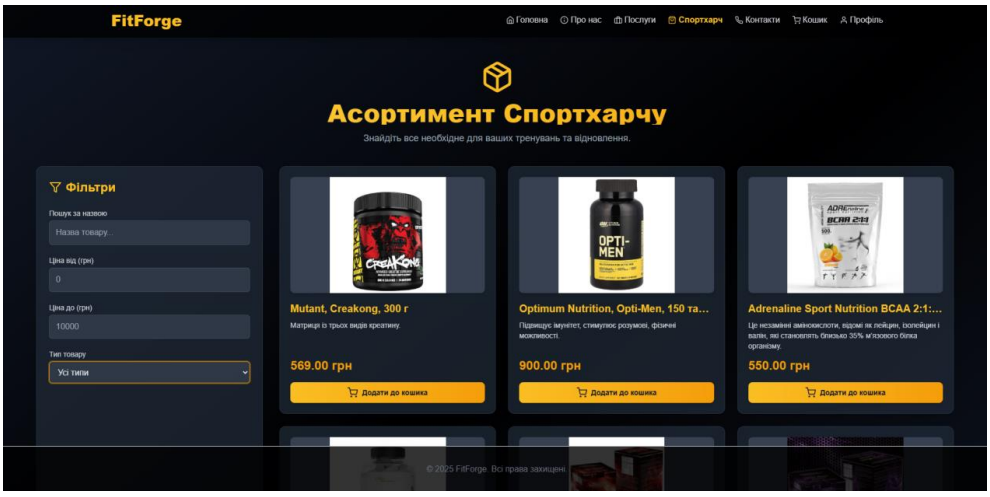

Рисунок 3.19 – Сторінка Спортхарчу

Її реалізація відбувається наступним чином:

– Заголовок сторінки "Асортимент Спортхарчу" з іконкою.

– Бічна панель фільтрів оформлена як .content-card. Містить поля для пошуку за назвою, фільтрації за мінімальною/максимальною ціною та типом товару. Зміна значень у фільтрах викликає функцію filterProducts().

– Сітка товарів відображає картки товарів. Область має обмежену висоту та кастомний скролбар.

– Кожен товар представлений карткою з зображенням, назвою, описом (обмеженим трьома рядками), ціною та кнопкою "Додати до кошика".

– Модальне вікно сповіщень для повідомлень про додавання до кошика або помилки.

– Контакти(contact.html) передбачає надання контактної інформації та можливості зв'язатися з клубом (рисунок 3.20).

Її реалізація відбувається наступним чином:

– Заголовок сторінки "Зв'яжіться з Нами" з іконкою.

– Блоки контактної інформації адреса (змінена на Тернопіль, вул. Крушельницької, 14), телефон, email, графік роботи представлені у вигляді окремих .content-card з відповідними іконками.

– Карта вбудовано iframe Google Maps, що показує вказану адресу.

– Секція "Як дістатися?" це текстовий опис.

– Секція "Ми в Соціальних Мережах" це іконки та посилання (заглушки) на соціальні мережі.

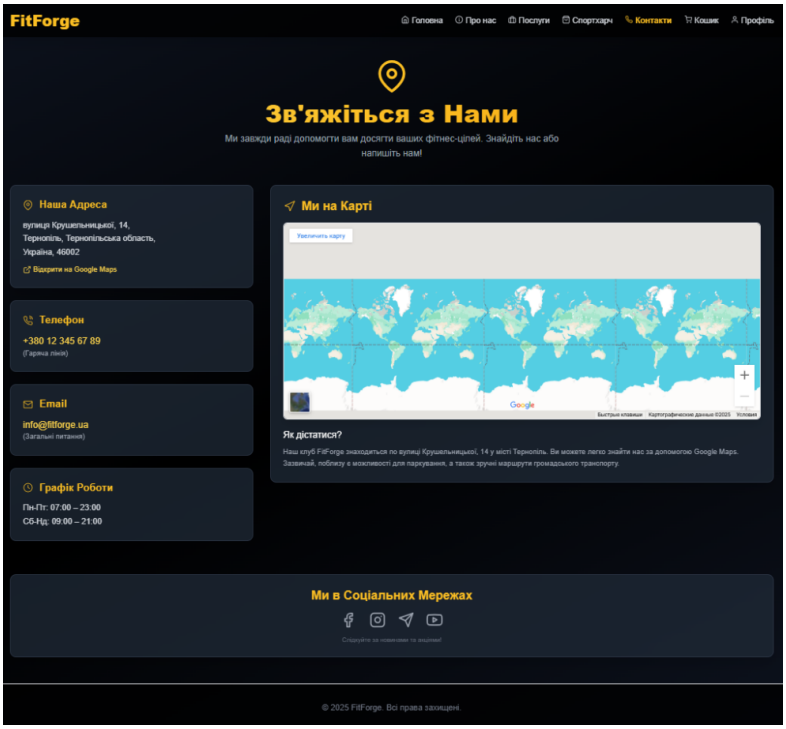

Рисунок 3.20 – Сторінка "Контакти"

Вхід (login.html) та Реєстрація (register.html) передбачає автентифікація та створення нових акаунтів користувачів (рисунок 3.21 та рисунок 3.22):

Її реалізація відбувається наступним чином:

– Обидві сторінки мають схожий дизайн - відцентрована картка форми на всю висоту видимої частини екрану (після хедера).

– Картка містить заголовок з іконкою (log-in або user-plus).

– Поля вводу для email, пароля (та імені користувача для реєстрації) мають іконки всередині та стилізацію.

– Кнопки "Увійти" / "Зареєструватися" стилізовані з градієнтом та іконками.

– Передбачено контейнер для відображення повідомлень про помилки зі стилізацією для помилок та успіху (для реєстрації).

– Посилання для переходу між сторінками входу та реєстрації.

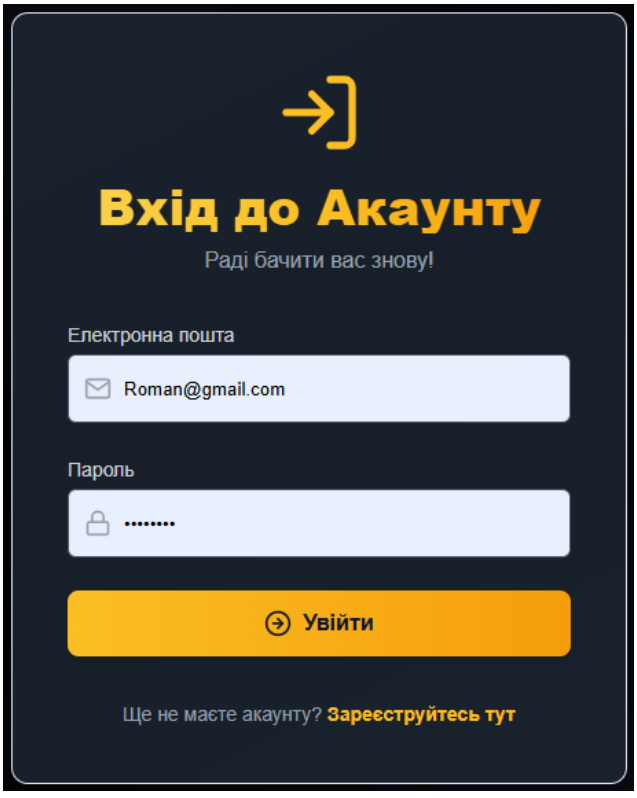

Рисунок 3.21 – Сторінка входу

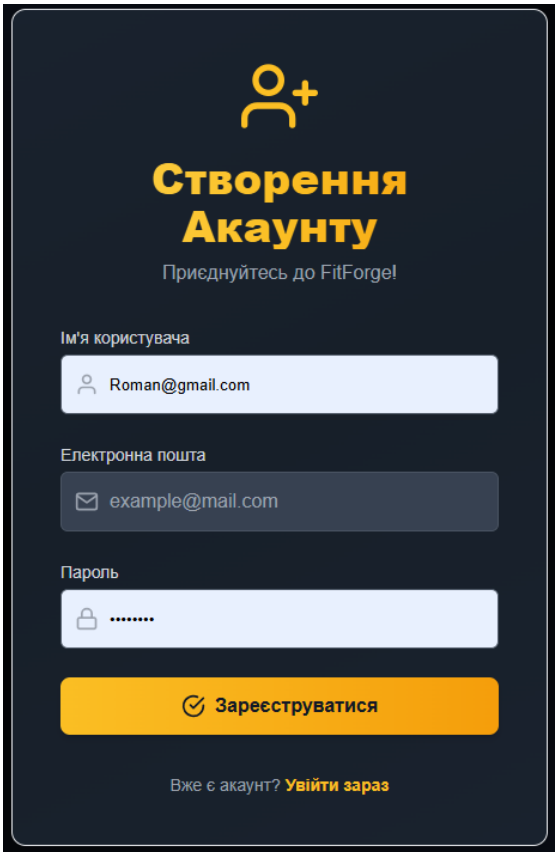

Рисунок 3.22 – Сторінка реєстрації

Профіль користувача(profile.html) передбачає відображення інформації про авторизованого користувача, надання можливості виходу міні-гра (рисунок 3.23):

Її реалізація відбувається наступним чином:

– Сторінка доступна лише авторизованим користувачам (перевірка токена, перенаправлення на login.html якщо токен відсутній).

– Картка профілю відображає аватар (SVG-іконка),ім'я користувача та email (завантажуються з /profile API). Поля мають ефект "завантаження".

– Кнопка "Вийти з акаунту" активна, видаляє токен з localStorage та перенаправляє на login.html.

– Картка з міні-грою "Вгадай число" містить кнопку для розгортання гри та саму гру з полем вводу, кнопкою та повідомленнями.

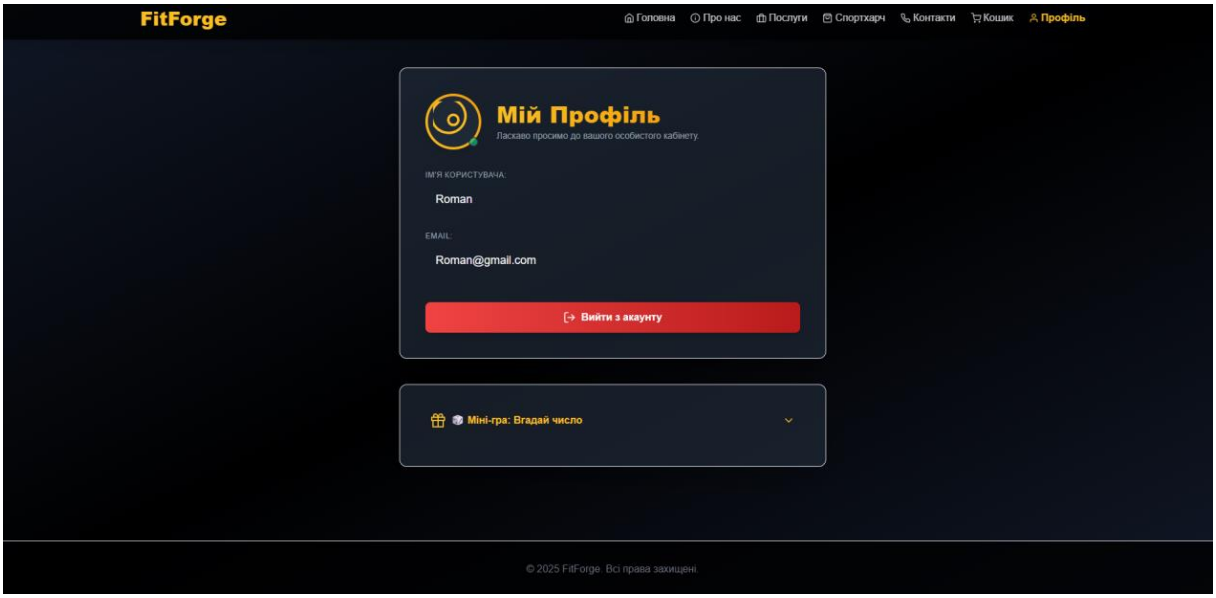

Рисунок 3.23 – Сторінка профілю

Кошик (cart.html) передбачає відображення товарів, доданих до кошика, можливість зміни їх кількості, видалення, перехід до оформлення замовлення (рисунок 3.24):

Її реалізація відбувається наступним чином:

– Сторінка доступна лише авторизованим користувачам.

– Заголовок "Ваш кошик".

– Кнопка "Оформити замовлення": Розміщена під заголовком, видима лише якщо кошик не порожній. Веде на pay.html.

– Відображає товари якщо кошик не порожній, товари відображаються у вигляді сітки карток. Кожна картка містить зображення товару (з плейсхолдером), назву, опис, ціну, елементи керування кількістю та кнопку "Видалити".

– Якщо кошик порожній, відображається відповідне повідомлення з іконкою та кнопкою "До каталогу спортхарчу".

– Модальні вікна використовуються для сповіщень про видалення товару, помилки, та для підтвердження видалення.

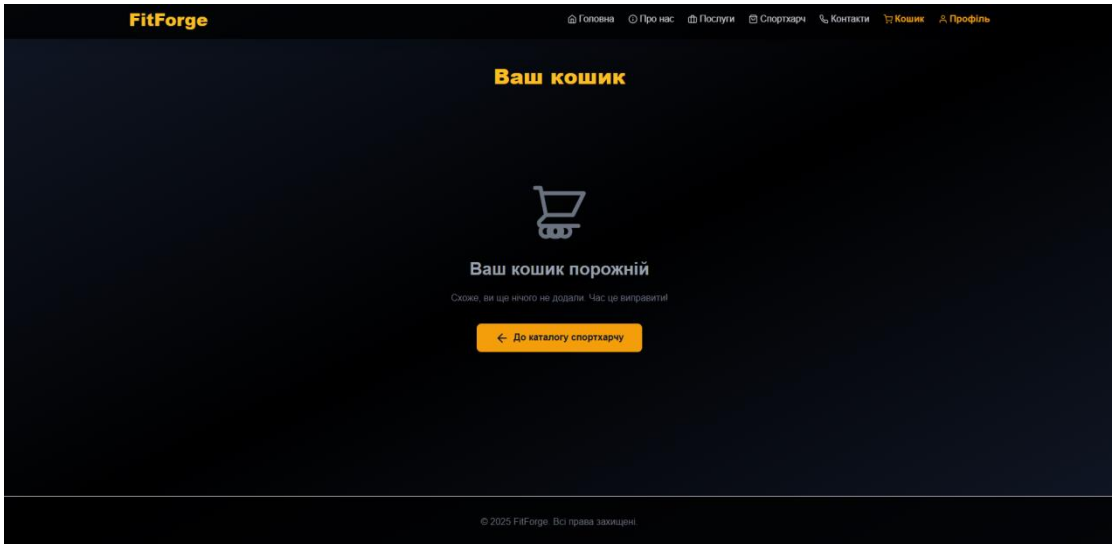

Рисунок 3.24 – Сторінка кошика

Оформлення замовлення (pay.html) передбачає збір контактної інформації клієнта та даних для доставки Новою Поштою, підтвердження замовлення (рисунок 3.25 та рисунок 3.26).

Її реалізація відбувається наступним чином:

– Сторінка доступна лише авторизованим користувачам.

– Заголовок "Оформлення Замовлення" з іконкою.

– Форма о формлена як .content-card.

– Блок "Ваші товари" відображає список товарів з кошика та загальну суму.

– Блок "Контактна інформація"- поля для ПІБ, телефону, email, коментаря. Поля мають іконки та стилізацію .form-field.has-icon.

– Блок "Доставка Новою Поштою" це два випадаючі списки (select) для вибору міста та відділення. Списки заповнюються динамічно за допомогою запитів до API Нової Пошти .

– Кнопка "Підтвердити замовлення" стилізована, з іконкою.

– Модальне вікно успіху відображається після успішного оформлення замовлення.

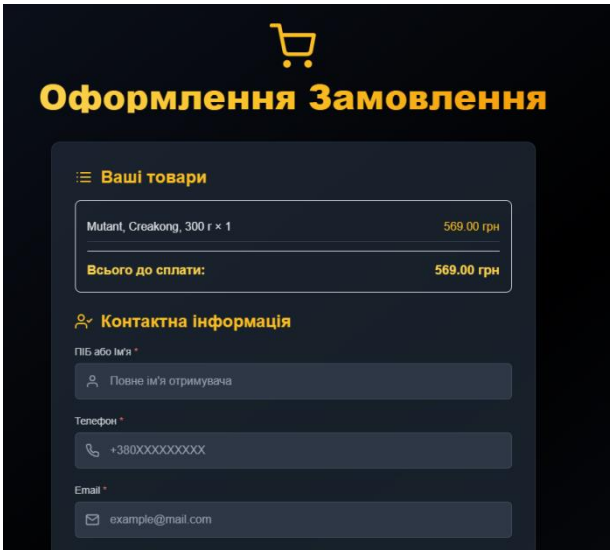

Рисунок 3.25 – Форма оформлення

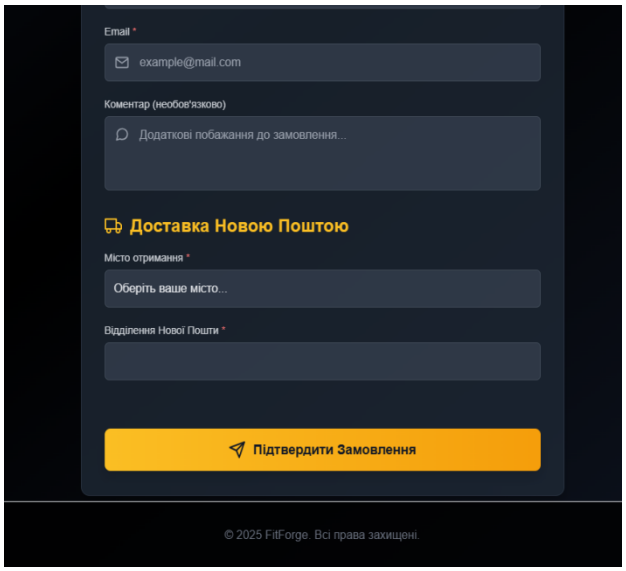

Рисунок 3.26 – Форма оформлення

Адмін панель(admin.html) передбачає управління товарами, перегляд замовлень та записів на абонементи.

Її реалізація відбувається наступним чином:

– Заголовок "Адмін-панель" з іконкою.

– Інтерфейс з вкладками реалізовано три вкладки ("Управління Товарами", "Перегляд Замовлень", "Записи на Абонементи") за допомогою кнопок та JavaScript для перемикання видимості контенту.

Вкладка "Управління Товарами" має формудля додавання нового товару (назва, тип, ціна, опис, URL зображення) та список існуючих товарів з можливістю видалення кожного товару (рисунок 3.27).

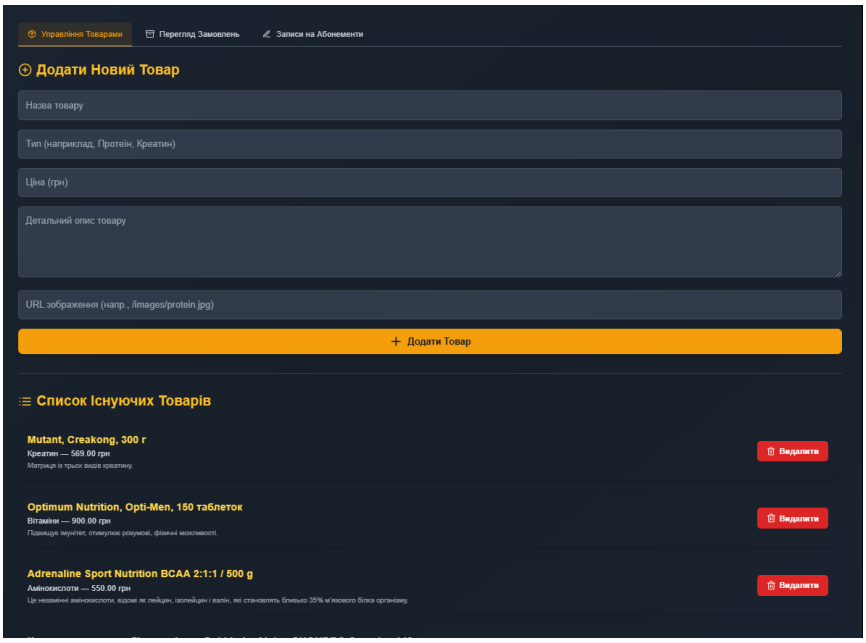

Рисунок 3.27 – Вкладка "Управління товарами)

Вкладка "Перегляд Замовлень" де кожне замовлення відображається у вигляді картки з детальною інформацією (клієнт, доставка, товари, сума, дата) та кнопкою "Видалити" (рисунок 3.28).

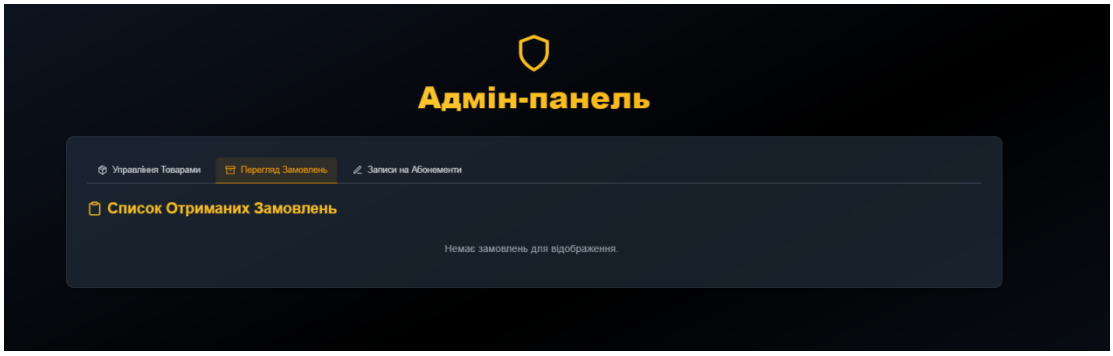

Рисунок 3.28 – Вкладка "Перегляд замовлень"

Вкладка "Записи на Абонементи" має список записів, отриманих з /api/enrollments (який читає enrollments.json). Кожен запис відображається у картці (ID, дата, користувач системи, дані клієнта з форми, обраний абонемент). Додано кнопку "Видалити" для записів (рисунок 3.29).

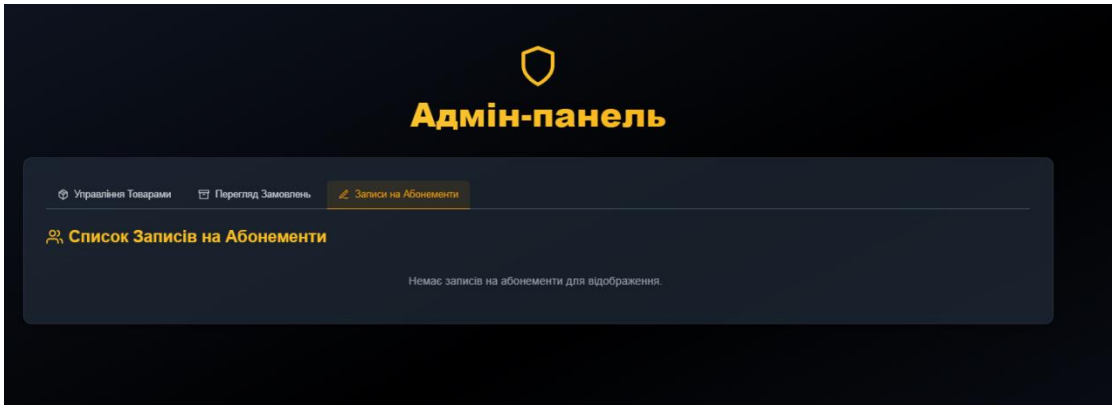

Рисунок 3.29 – Вкладка "Записи на абонименти"

Модальні вікна (notificationModal, confirmationModal): Використовуються для всіх операцій (додавання/видалення товарів, видалення замовлень, видалення записів).

Ключові JavaScript функції та механізми подані нижче.

Єдиний скрипт (DOMContentLoaded) на всіх сторінках відповідає за:

– Перевірку localStorage.getItem("authToken").

– Динамічну зміну посилання "Вхід" на "Профіль" та "Кошик" (з іконками), якщо користувач авторизований.

– Видалення посилання "Кошик",якщо користувач не авторизований.

Визначення поточної сторінки та підсвічування відповідного посилання в навігації жовтим кольором та жирним шрифтом. Інші посилання залишаються у стандартному стані.

Робота з API:

– Обробка відповідей включає перевірку response.ok та розбір JSON. Помилки сервера або мережі обробляються в блоках catch.

Динамічне оновлення DOM:

– Списки товарів, вміст кошика, списки замовлень та записів на абонементи генеруються динамічно на основі даних, отриманих з сервера. Створюються HTML-елементи, заповнюються даними та додаються до відповідних контейнерів.

– Інтеграція з API Нової Пошти (на pay.html):

– Функції loadNPCities та loadNPWarehouses роблять POST-запити до API Нової Пошти.

– Реалізовано залежне завантаження відділення завантажуються тільки після вибору міста.

– Обробляються можливі помилки від API.

3.4. Створення бази даних

Для зберігання всіх даних "FitForge" було обрано MongoDB. Для проєкту, де структура даних може і буде змінюватись, жорстка схема – це скоріше недолік, а MongoDB дозволяє легко редагувати таблиці даних.

Вся взаємодія на сервері відбувається через бібліотеку Mongoose – вона дає змогу моделювати об'єкти, валідувати дані та будувати зручні запити.

У проєкті є одна головна база даних – auth_db. Усередині неї створено основні колекції, які є серцем нашого проєкту: users (користувачі), products (товари) та orders (замовлення).

Модель User (Користувач) поданий нижче:

Колекція users зберігає інформацію про зареєстрованих користувачів сайту. Схема даних для користувача (userSchema) визначена в server.js і включає наступні поля:

– Username(String) – ім'я користувача (логін), яке він вказує при реєстрації.

– Email(String) – електронна пошта користувача. Це поле є унікальним (unique: true) та обов'язковим (required: true), використовується для входу в систему та комунікації.

– Password(String) – пароль користувача. В базі даних зберігається не сам пароль, а його хеш-сума, отримана за допомогою бібліотеки bcryptjs.

– Cart(Array) — кошик користувача. Це масив вбудованих документів, кожен з яких представляє товар у кошику:

– ProductId – ідентифікатор товару, що є посиланням на документ у колекції products.

– Quantity – кількість одиниць даного товару в кошику. За замовчуванням встановлюється в 1.

Приклад структури документа користувача в MongoDB:

```
JSON
{
  "_id": ObjectId("682c8d682a4f04d2fae6117b"),
  "username": "Roman",
  "email": "roman@example.com",
  "password": "$2a$10$...", // Хешований пароль
  "cart": [
    {
      "productId": ObjectId("68273168d35d9c447f349106"),
      "quantity": 2,
      "_id": ObjectId("...")
    }
  ],
  "__v": 0
}
```

Модель Product (Товар) передбчає наступне:

Колекція products містить інформацію про товари спортивного харчування, доступні на сайті. Схема productSchema включає:

– Name(String) – назва товару. Обов'язкове поле.

– Type(String) – категорія або тип товару (наприклад, "Протеїн", "Креатин", "Вітаміни"). Обов'язкове поле.

– Price — ціна товару в гривнях. Обов'язкове поле.

– Description(String) – детальний опис товару.

– imageUrl(String) – відносний або повний URL-шлях до зображення товару (наприклад, /images/q1.jpg).

Приклад структури документа товару в MongoDB:

```
JSON
{  "_id": ObjectId("68273168d35d9c447f349106"),
  "name": "Протеїн Ізолят Gold Standard",
  "type": "Протеїн",
```

```
    "price": 1250.00,
    "description": "Високоякісний сироватковий ізолят для
швидкого відновлення та росту м'язів.",
    "imageUrl": "/images/gold_standard_isolate.jpg",
    "__v": 0
  }
```


Модель Order (Замовлення) подано нижче:

Колекція orders зберігає інформацію про всі оформлені замовлення на сайті. Дані для цієї колекції також дублюються у файл orders.json. Схема orderSchema має наступні поля:

– userId – ідентифікатор користувача, який зробив замовлення. Є посиланням на документ у колекції users. Обов'язкове поле.

– Products(Array) – масив замовлених товарів. Кожен елемент масиву є об'єктом, що містить:

– productId – посилання на товар у колекції products.

– name – назва товару.

– quantity – замовлена кількість товару.

– price – ціна за одиницю товару на момент оформлення замовлення.

– totalPrice – загальна вартість замовлення. Обов'язкове поле.

– customerInfo – вбудований об'єкт з контактною інформацією клієнта, наданою при оформленні:

– fullName – повне ім'я отримувача.

– email – електронна пошта.

– phone – номер телефону.

– comment – коментар до замовлення.

– shippingInfot – вбудований об'єкт з інформацією про доставку через "Нову Пошту":

– cityRef – Ref ідентифікатор міста з API Нової Пошти.

– cityName – назва міста.

– warehouseRef – Ref ідентифікатор відділення Нової Пошти.

– warehouseName – повна назва та адреса відділення.

– createdAt – ата та час створення замовлення. За замовчуванням встановлюється поточний час (Date.now).

Приклад структури документа замовлення в MongoDB (схожий на структуру в orders.json):

```
JSON
{
  "_id": ObjectId("682c8dda2a4f04d2fae611b7"),
  "userId": ObjectId("682c8d682a4f04d2fae6117b"),
  "products": [
    {
      "productId": ObjectId("68273168d35d9c447f349106"),
      "name": "Gold",
      "quantity": 1,
      "price": 599
    }
  ],
  "totalPrice": 599,
  "customerInfo": {
    "fullName": "Roman",
    "email": "test@gmail.com",
    "phone": "+380687419836",
    "comment": ""
  },
  "shippingInfo": {
    "cityRef": "b06d9799-7534-11e8-9d95-005056b2fc3d",
    "cityName": "Плотича",
    "warehouseRef": "cdc706b9-6d1b-11ee-9eb1-d4f5ef0df2b8",
    "warehouseName": "Пункт приймання-видачі №2 (до 30 кг):
вул. Грушевського, 46/1"
  },
  "createdAt": ISODate("2025-05-20T14:12:42.358Z"),
  "__v": 0
}
```

Окрім цих колекцій, серверна логіка також взаємодіє з файлами enrollments.json (для зберігання записів на абонементи) та orders.json (як дублікат інформації про замовлення). Структура даних у цих файлах тісно пов'язана зі схемами MongoDB, але вони існують як окремі JSON-файли у файловій системі сервера. Це було реалізовано для демонстрації альтернативного способу зберігання та для потенційного спрощення доступу до даних без прямих запитів до БД для певних адміністративних завдань.

Використання Mongoose дозволяє визначати валідацію на рівні схеми, створювати віртуальні поля, методи екземплярів та статичні методи моделей, що значно полегшує розробку та підтримку серверної логіки, пов'язаної з даними.

3.5 Реалізація інтерфейсу користувача

Інтерфейс користувача веб-додатку "FitForge" розроблено з акцентом на сучасний вигляд, інтуїтивну навігацію та зручність використання на різних пристроях . Ключовими аспектами реалізації інтерфейсу єдиний візуальний стиль завдяки Tailwind CSS, всі сторінки сайту мають узгоджений дизайн. Це включає темну кольорову схему з яскравими жовтими акцентами, використання шрифту 'Inter', а також застосування ефекту "скла" (glassmorphism) для карток та фонових елементів. Це створює відчуття глибини та сучасності.

Структура сторінок відображається через:

Хедер зафіксований у верхній частині екрана, містить логотип "FitForge" (який є посиланням на головну сторінку) та основні навігаційні посилання (Головна, Про нас, Послуги, Спортхарч, Контакти). Також у хедері динамічно відображаються посилання "Вхід" або "Профіль" та "Кошик", залежно від стану авторизації користувача. Кожне посилання в навігації супроводжується відповідною іконкою Feather Icon. Поточна сторінка підсвічується жовтим кольором та жирним шрифтом.

Основний контент займає простір між хедером та футером. Для сторінок з великою кількістю контенту, що виходить за межі одного екрану, передбачена прокрутка. Сторінки, такі як "Вхід", "Реєстрація", "Оформлення замовлення",

мають контент, відцентрований у контейнері з обмеженою максимальною шириною для кращої читабельності форм. Сторінки каталогу або інформаційні сторінки можуть використовувати всю ширину контейнера.

Футер зафіксований у нижній частині екрана, містить інформацію про авторські права та динамічно оновлюваний рік. Має напівпрозорий фон з ефектом розмиття.

Інтерактивні елементи:

Усі основні кнопки дії (наприклад, "Додати до кошика", "Оформити замовлення", "Увійти", "Зареєструватися", "Долучитися") мають яскравий жовтий градієнтний фон, тінь та іконку, що візуально привертає увагу користувача. Додаткові або другорядні кнопки (наприклад, "Скасувати" в модальних вікнах, кнопки видалення) мають більш стриманий сірий або червоний дизайн. Усі кнопки мають ефекти при наведенні (hover) та фокусі.

Поля вводу та випадаючі списки стилізовані для відповідності темній темі: темний фон, світлий текст, жовтий акцент при фокусі. Для полів вводу на сторінках реєстрації, входу, контактів та оформлення замовлення реалізовано розміщення іконок всередині поля зліва, з відповідним відступом для тексту. Валідація на стороні клієнта та повідомлення про помилки відображаються безпосередньо біля форми.

Для сповіщень ("Товар додано", "Помилка") та підтвердження дій ("Ви впевнені, що хочете видалити?") використовуються кастомні модальні вікна. Вони з'являються поверх основного контенту з напівпрозорим затемненим фоном та мають анімацію появи/зникнення.

Для відображення товарів, послуг, абонементів, замовлень, записів на абонементи використовуються картки з ефектом "скла". Вони мають заокруглені кути, тінь та рамку, що створює відчуття глибини. При наведенні на картки товарів або послуг спрацьовує ефект підняття.

Для довгих списків (товари в адмін-панелі, замовлення, записи, товари в кошику на сторінці оформлення) використовується вертикальна прокрутка з кастомним стилем скролбару для кращої візуальної інтеграції.

Візуальні акценти та ієрархія:

Жовтий колір використовується для важливих елементів, заголовків, кнопок дії та активних посилань, що допомагає керувати увагою користувача.

Різні розміри та насиченість шрифту 'Inter' використовуються для створення чіткої візуальної ієрархії (заголовки сторінок, заголовки секцій, основний текст, підписи). Градієнтний текст застосовується для основних заголовків сторінок.

Feather Icons активно використовуються для покращення сприйняття інформації та навігації. Вони присутні в хедері, на кнопках, поруч із заголовками секцій та в інформаційних блоках.

Tailwind CSS дозволив гнучко керувати відступами) та простором між елементами,що сприяє кращій читабельності та загальному естетичному вигляду.

Загалом, інтерфейс користувача "FitForge" розроблено з метою бути сучасним, привабливим та функціональним, забезпечуючи позитивний досвід взаємодії для кінцевих користувачів та зручність управління для адміністратора.

3.6. Тестування системи

Тестування програмного продукту "FitForge" є невід'ємною частиною процесу розробки, спрямованою на виявлення та виправлення помилок, а також на перевірку відповідності реалізованого функціоналу поставленим вимогам. Було проведено декілька видів тестування:

Модульне тестування (Unit Testing) подано нижче.

Для даного проекту не використовувалися спеціалізовані фреймворки для автоматизованого модульного тестування JavaScript, окремі функції, особливо на серверній стороні (в server.js), перевірялися розробником вручну на коректність роботи.

Приклад для серверної частини - це перевірка функцій валідації даних, що надходять від клієнта, коректності запитів до бази даних MongoDB (наприклад,

пошук користувача, створення нового запису), правильності генерації JWT токенів та їх верифікації middleware-функцією authenticateToken.

Приклад для клієнтської частини – це перевірка функцій валідації форм (наприклад, формат email, номер телефону, заповненість полів), функцій розрахунку (наприклад, загальна сума в кошику), коректності формування об'єктів для відправки на сервер.

Інтеграційне тестування передбачає наступне.

Цей вид тестування був спрямований на перевірку коректності взаємодії між клієнтською та серверною частинами додатку, а також взаємодії з зовнішніми сервісами (API Нової Пошти).

Тестування API ендпоінтівю. За допомогою інструментів типу Postman (або безпосередньо через клієнтську частину) перевірялася робота всіх API маршрутів:

– Реєстрація та логін користувача: коректність створення запису в БД, генерація токена, обробка помилок (невірний пароль, існуючий email).

– Операції з товарами: додавання, отримання списку, видалення.

– Операції з кошиком: додавання, отримання, зміна кількості, видалення товару, очищення кошика.

– Оформлення замовлення: передача всіх даних, збереження в MongoDB та orders.json, очищення кошика.

– Запис на абонемент: передача даних, збереження в enrollments.json.

– Отримання списків замовлень та записів на абонементи для адмін-панелі.

Взаємодія Front-End та Back-End. Перевірялося, що дані, введені користувачем на клієнті, правильно надсилаються на сервер, обробляються ним, і відповідь сервера коректно інтерпретується та відображається на клієнті (наприклад, оновлення списків, показ повідомлень про успіх/помилку).

Інтеграція з API Нової Пошти.

Тестувалася коректність запитів до API для отримання списку міст та відділень, а також обробка можливих помилок від цього зовнішнього сервісу.

Функціональне тестування передбачає наступне.

Функціональне тестування проводилося вручну шляхом виконання типових сценаріїв використання системи з точки зору різних типів користувачів (незареєстрований користувач, зареєстрований користувач, адміністратор).

Основні тестові сценарії передбачають наступні кроки:

Відкрити сторінку реєстрації, ввести валідні дані, натиснути "Зареєструватися" (рисунок 3.30).

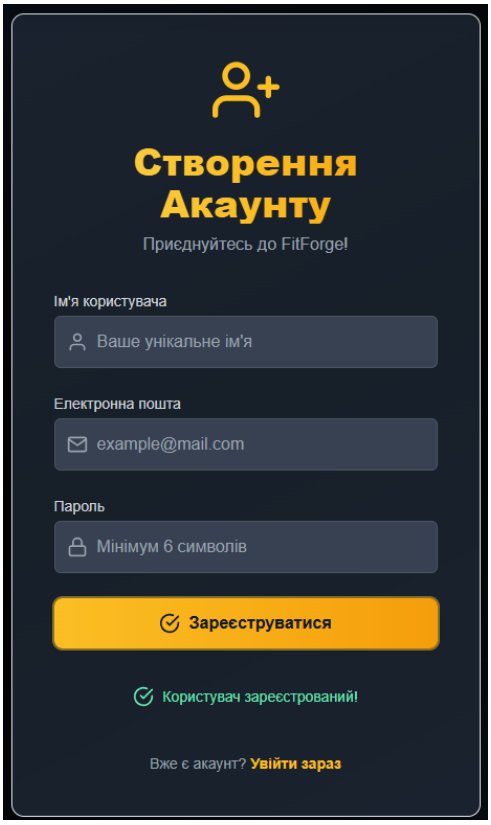

Рисунок 3.30 – Успішна реєстрація

Ввести невалідні дані (короткий пароль, некоректний email, порожні поля). Спробувати зареєструвати користувача з існуючим email (рисунок 3.31).

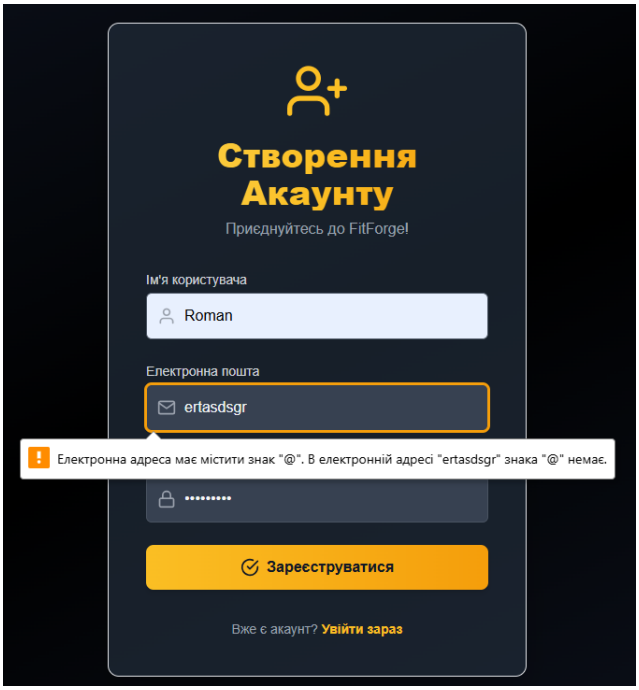

Рисунок 3.31 – Помилка реєстрації

Очікуваний результат – успішна реєстрація з валідними даними, відображення відповідних повідомлень про помилки при невалідних даних. Дані користувача з'являються в колекції users в MongoDB.

Сценарій входу користувача передбчає наступні кроки:

Відкрити сторінку входу, ввести правильні/неправильні логін та пароль.

Очікуваний результат – успішний вхід та перенаправлення на сторінку профілю при правильних даних (рисунок 3.32)

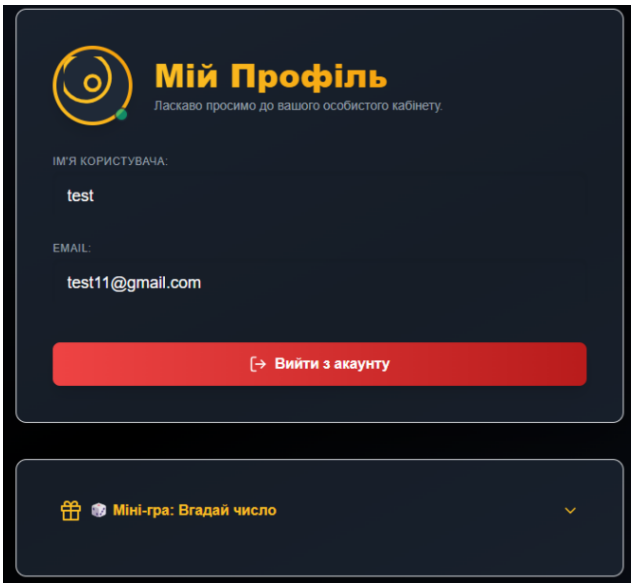

Рисунок 3.32 – Профіль користувача

Повідомлення про помилку при неправильних даних. authToken зберігається в localStorage (рисунок 3.33).

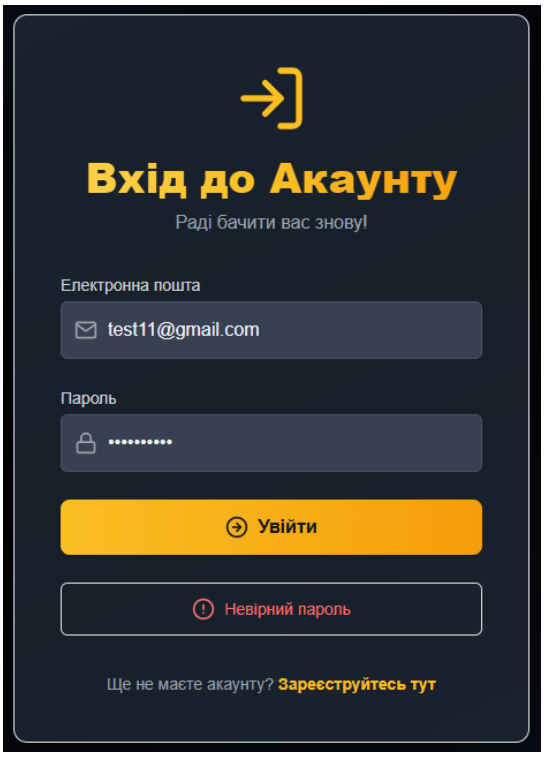

Рисунок 3.33 – Помилка при вході

Сценарій перегляду товарів та додавання до кошика (для авторизованого користувача) передбачає наступні кроки:

Увійти в систему. Перейти на сторінку "Спортхарч". Використати фільтри. Додати товар до кошика. Перевірити вміст кошика (рисунок 3.34 та рисунок 3.35).

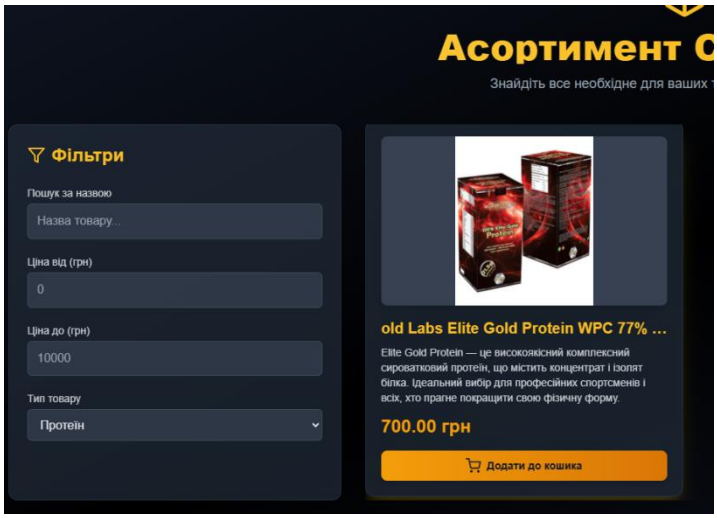

Рисунок 3.34 – Фільтр товару

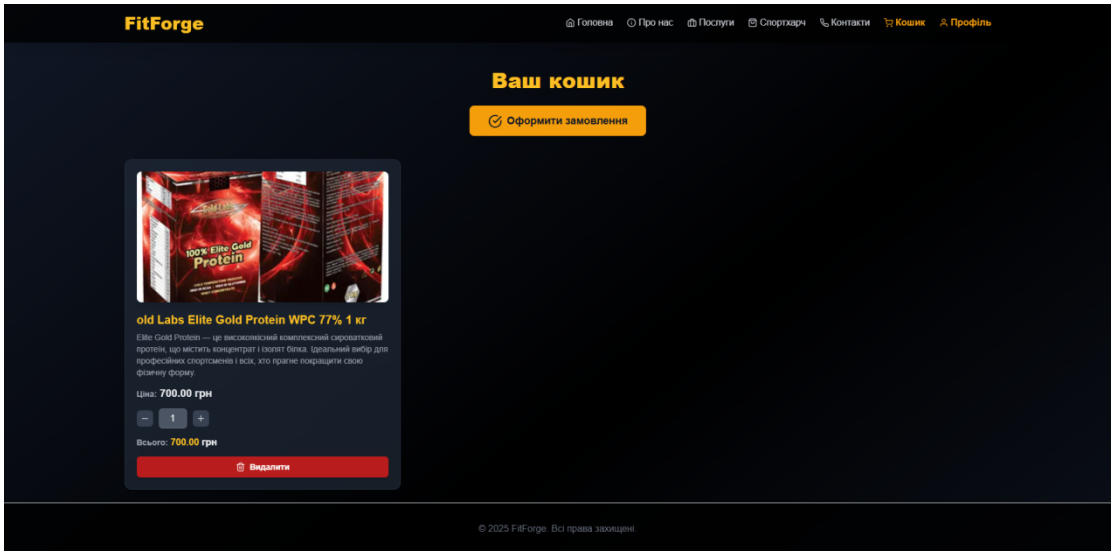

Рисунок 3.35 – Кошик

Відповідний запис з'являється в полі cart документа користувача в MongoDB.

Сценарій оформлення замовлення передбачає наступні кроки.

Додати товари до кошика. Перейти до оформлення замовлення. Заповнити контактні дані та дані доставки (вибрати місто/відділення НП). Натиснути "Підтвердити замовлення" (рисунок 3.36).

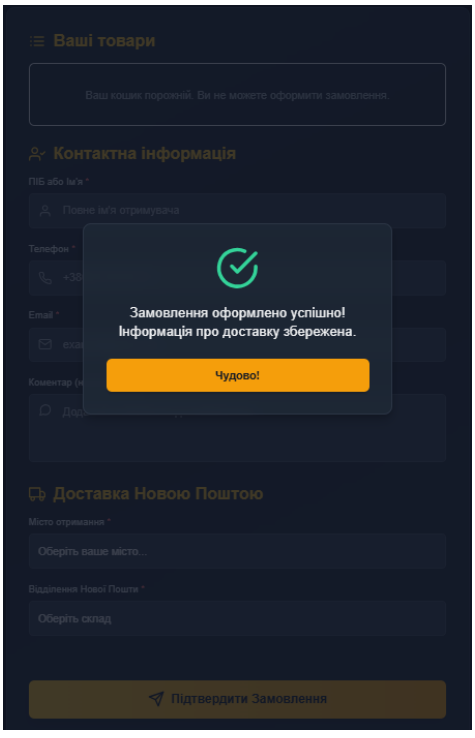

Рисунок 3.36 – Оформлення замовлення

Сценарій запису на абонемент (на сторінці "Послуги") передбачає наступні кроки. Авторизуватися. На сторінці послуг обрати абонемент, натиснути "Долучитися". Заповнити та відправити форму в модальному вікні (рисунок 3.37).

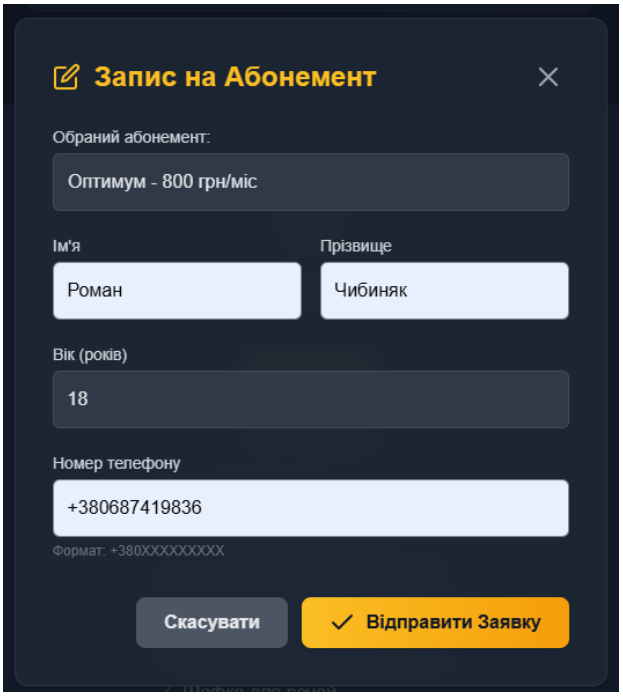

Рисунок 3.37 – Запис на абонимент

Сценарії для адмін-панелі (admin.html) передбачає наступні кроки.

Управління товарами. Додавання нового товару(з валідними/невалідними даними) (рисунок 3.38 та 3.39).

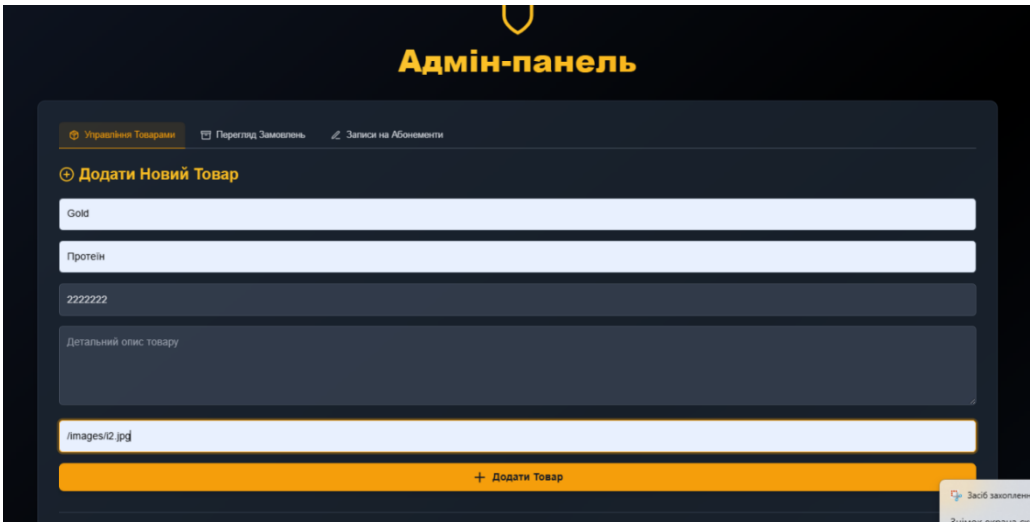

Рисунок 3.38 – Додавання товару з адмін панелі

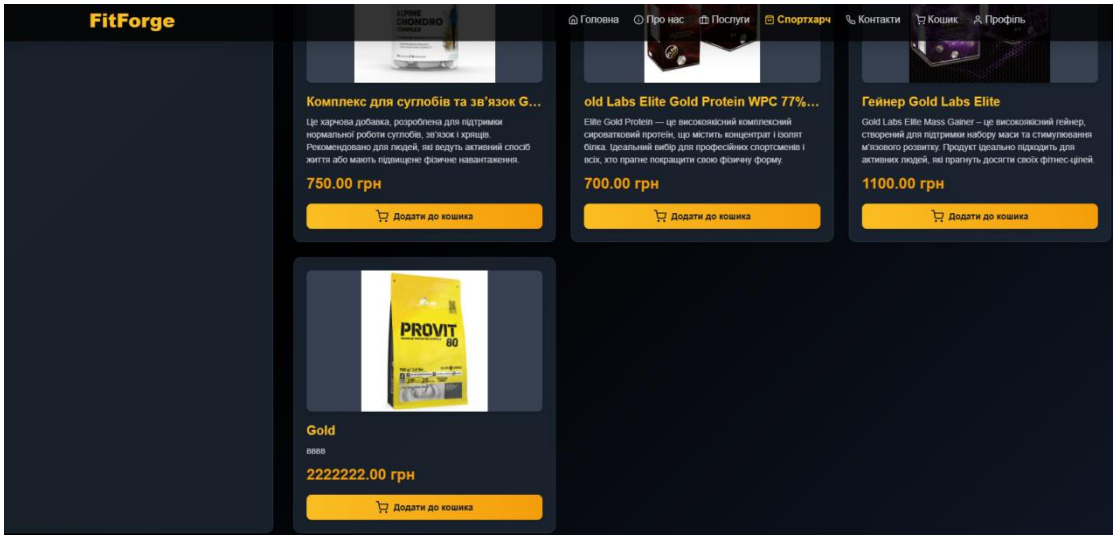

Рисунок 3.39 – Результат

Відображення списку товарів. Видалення товару (з підтвердженням) (рисунок 3.40).

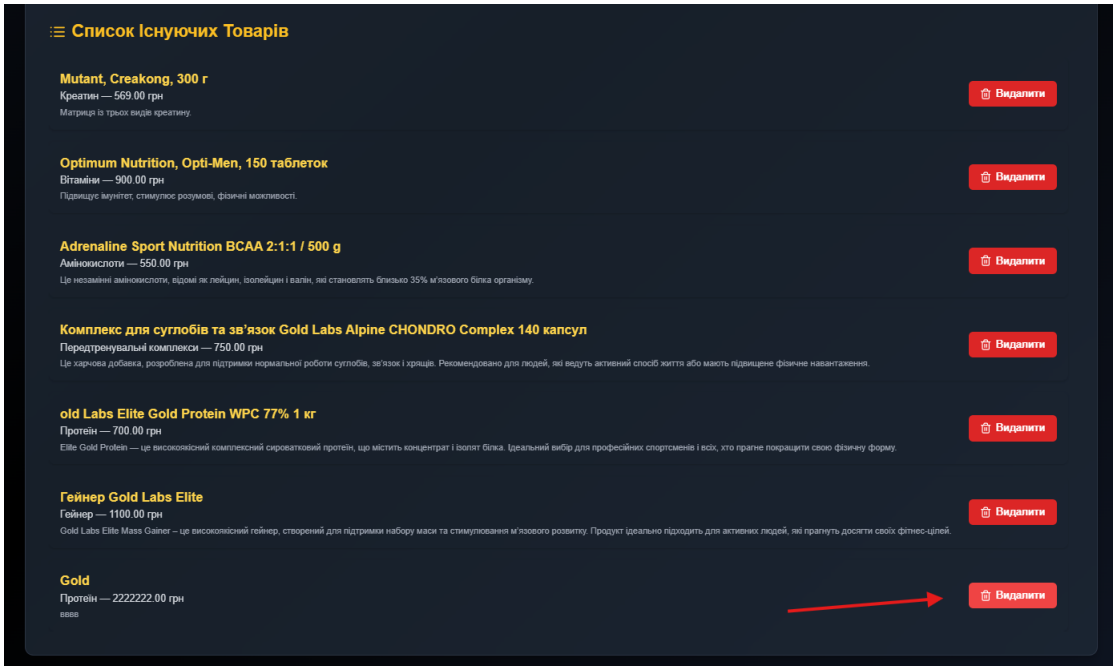

Рисунок 3.40 – Видалення товару

Перегляд замовлень.

Відображення списку замовлень з orders.json . Коректне відображення всіх деталей замовлення. Видалення замовлення(з підтвердженням) (рисунок 3.41).

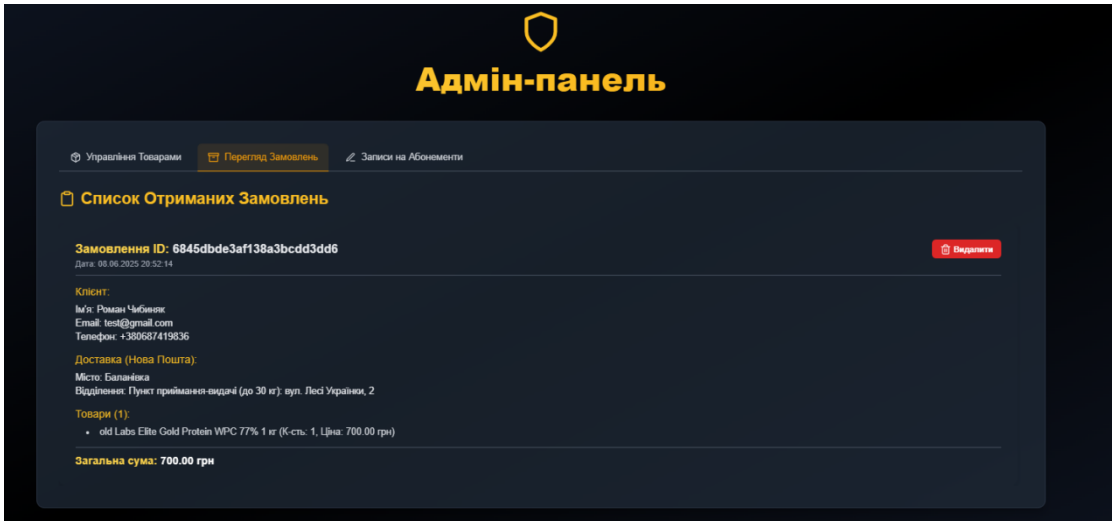

Рисунок 3.41 – Перегляд замовлень

Перегляд записів на абонементи.

Відображення списку записів з enrollments.json. Коректне відображення всіх деталей запису (рисунок 3.42).

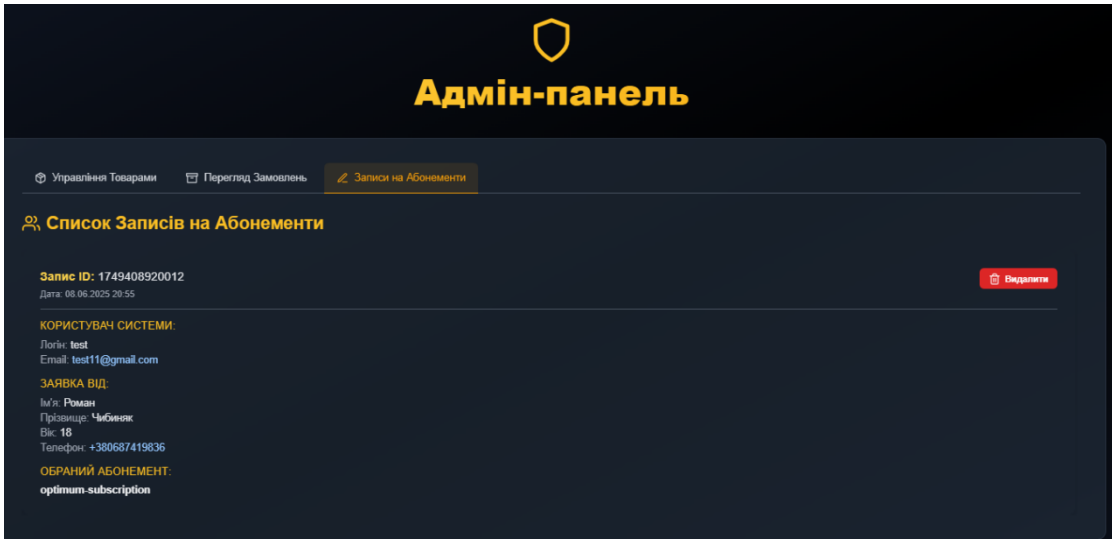

Рисунок 3.42 – Список записів

Всі операції виконуються коректно, дані оновлюються, відповідні повідомлення відображаються. Файли orders.json, enrollments.json та база даних MongoDB (колекції products, orders) оновлюються належним чином.

На етапі тестування було виявлено та виправлено ряд недоліків та помилок . Наприклад:

– Проблема з невидимим текстом у полях вводу через конфлікт стилів фону та тексту (виправлено зміною класів CSS для інпутів).

– Помилка ReferenceError ... is not defined при кліку на динамічно створені кнопки (виправлено переходом від onclick атрибутів до addEventListener).

– Некоректне завантаження списку міст Нової Пошти через неправильні параметри запиту (виправлено видаленням Limit та Page з methodProperties).

– Помилки авторизації ("Невірний токен", "Відсутній токен") через закінчення терміну дії токена або відсутність необхідних серверних маршрутів (додано маршрути, покращено обробку помилок на клієнті).

– Проблеми з відображенням зображень через неправильні шляхи (виправлено шляхи).

– Некоректне відображення даних у списках замовлень/записів через невідповідність структури JSON та логіки рендерингу (оновлено функції renderOrders, renderEnrollments).

– Відсутність HTML для модальних вікон на деяких сторінках (додано HTML).

Після виправлення виявлених помилок було проведено повторне тестування ключових функцій, яке підтвердило їх працездатність та відповідність основним вимогам. Система готова до демонстрації та подальшого можливого розширення.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Навіть якщо не дивитись на статистику, видно , що люди в Україні все більше захоплюються спортом. Незважаючи на всі економічні негаразди, тренд на здоровий спосіб життя лише набирає популярності. Це про повноцінні тренування в залах, про персональних тренерів, про правильне харчування. Попит на це все є, і він зростає.

Де є такий попит, там бізнес має йти в онлайн. Це вже аксіома. Створити сайт, який буде одночасно і вітриною фітнес-клубу, і інтернет-магазином спортивного харчування. Для клієнта це — зручність. Максимальна зручність. Все в єдиному цифровому просторі, де можна і записатись на тренування, і замовити протеїн.

Основні переваги вебплатформи представлено в таблиці 4.1.

Таблиця 4.1 – Основні переваги

Функція	Користь для бізнесу	Перевага для клієнтів
Онлайн-запис на тренування	Менше навантаження на персонал	Можливість самостійного бронювання
Онлайн-оплата абонементів	Проста інтеграція з CRM / бухгалтерією	Безпечна оплата в один клік
Каталог спортивного харчування	Додатковий дохід	Зручно замовити продукцію, не виходячи з дому
Акції та знижки	Підвищення продажів	Індивідуальні пропозиції
Персональний кабінет	Збір даних для аналітики	Контроль за досягненнями та витратами

Унікальність розробки полягає у повній адаптації до конкретного фітнес-бізнесу, який об’єднує тренувальні та торгові послуги. Створення єдиного цифрового середовища дозволяє уникнути розпорошення даних, об’єднати клієнтську базу та посилити бренд через комплексний онлайн-сервіс.

Конкуренцію створюють як міжнародні SaaS-рішення (Mindbody, Glofox), так і українські платформи на базі готових CMS, однак їх функціонал не є достатньо гнучким для поєднання з e-commerce. Більшість із них або орієнтовані лише на тренування, або лише на онлайн-магазини, тому наш продукт займає вигідну позицію на ринку як гібридна платформа.

Аналітика ринку спортивного харчування зображено у таблиці 4.2.

Таблиця 4.2 – Аналіз ринку спортивного харчування.

Показники	Значення (Україна, 2023 рік)
Обсяг ринку (оцінка)	=1.2 млрд. грн
Популярність онлайн-продажів	>60% замовлень через інтернет
Частота повторних покупок	1-2 рази на місяць

Таким чином, створення вебплатформи, яка об'єднує фітнес-послуги і продаж спортивного харчування, є доцільним, перспективним і стратегічно вигідним кроком.

4.2 Розрахункова частина

Розрахунок витрат на розробку подано нижче у таблиці 4.3.

Таблиця 4.3 – Розрахунок витрат на розробку.

Посада	Тариф, грн/год	Обсяг робіт, год	Вартість, грн
Frontend-розробник	640	20	12800
Backend-розробник	680	17	11560
UI/UX дизайнер	460	40	18400
QA інженер	450	30	13500
Project Manager	500	25	12500
Разом	-	-	68760

Сел – витрати на електроенергію:

Середнє споживання обладнання:

Комп’ютер+ноутбук+монітор=250Вт

Час роботи: 350 год → 87.5 кВт / 87.5 × 6.5 грн = 568.75 грн

Сінш – інші витрати продемонстровано у таблиці 4.4.

Таблиця 4.4 – Інші витрати

Товар	Вартість, грн
Канцелярія	300
Інтернет і мобільний зв’язок	1000
Комунальні витрати	1800
Ліцензії, домен, хостинг	2500
Разом	5600

Загальні витрати на розробку = 421890 + 568.75 + 5600 = 428058.75 грн

4.2.2 Економічний ефект

Джерела економії та прибутку продемонстровано у таблиці 4.5.

Таблиця 4.5 – Економія та прибуток

Джерело	Щомісячна економія/дохід, грн	Річна економія/дохід, грн
Автоматизація записів	3000	36000
Онлайн-оплата	1500	18000
Зменшення друку	700	8400
Дохід від продажу харчування (чистий прибуток)	8000	96000
Разом	13200	158400

4.2.3 Окупність

Ток = Срозр / Е = 428058.75 / 158400 ≈ 2.7 роки

Це досить короткий період для ІТ-рішення з мультифункціональним застосуванням.

Після окупності платформа починає приносити чистий прибуток.

4.3 Обґрунтування необхідності розробки

Сьогодні ніхто не хоче дзвонити, щоб дізнатись розклад, чи стояти в черзі, щоб оплатити абонемент. Сучасний клієнт фітнес-клубу очікує, що все це можна зробити онлайн – швидко, просто і в будь-який час. Саме тому веб-платформа, яка об'єднує в собі і систему керування залом, і магазин спортхарчу.

Це створення цілісної екосистеми для клієнта. Він не просто купує послугу, він отримує єдиний простір для свого здорового способу життя. Тут він планує тренування, тут же може замовити потрібні добавки для кращих результатів. Такий досвід створює лояльність і вигідно виділяє клуб на фоні конкурентів, особливо в очах молодої аудиторії, яка живе у смартфонах.

Для самого бізнесу переваги ще очевидніші. Система сама записує на заняття, надсилає нагадування, приймає оплату. Людський фактор – мінімальний, а це означає менше рутини для персоналу і більше часу на роботу з клієнтами. Продаж супутніх товарів відкриває додатковий потік доходу без потреби утримувати окремий магазин.

Така платформа – це ресурс цінних даних. Це дозволяє приймати рішення не навмання, а на основі цифр – адаптувати розклад, робити персоналізовані пропозиції, бачити, яка реклама працює. Правильна архітектура з самого початку - це ключ до майбутнього.

ВИСНОВКИ

У процесі розробки кваліфікаційної роботи вдалося створити повноцінну веб-платформу для фітнес-клубу “FitForge” – з усім, що може знадобитись як клієнту, так і власникам: від базового інформаційного наповнення до онлайн-запису на тренування і вбудованого магазину спортивного харчування. Усе в одному місці, без зайвих переходів і вікон.

На першому етапі розглядали аналіз, порівняння з конкурентами, такими як Member24 чи SportLife, пошук недоліків. Такі недоліки були найдені – більшість рішень або вузькі, або морально застарілі, або незграбні. У результаті сформувалась ідеальний сервіс для користувача.

Проєктування відбувалось стандартно, з ухилом у практичність. Взято за основу клієнт-серверну модель – тут без іновацій. На бекенді – Node.js з Express, а база MongoDB, бо з нею простіше працювати із структурами, які ще будуть змінюватись. Фронт був створений з Tailwind – зручна річ, не треба вигадувати стиль з нуля, але при цьому не обмежує. JavaScript – основа. Користувачі, замовлення, товари – усе це виведено в окремі логічні сутності з прописаними зв'язками.

Під час реалізації було створено серверну частину з API зроблене модульно: автентифікація через JWT, управління товарами, корзина, замовлення – все окремо. Клієнтська частина була реалізована у вигляді набору статичних та динамічних сторінок з інтерактивними елементами, формами та інтеграцією зі стороннім сервісом (API "Нової Пошти").

Тестування було проведено функціональне, перевірено інтеграцію, підключено UI-тестування. Сценарії “зареєструвався – замовив ” працюють добре. В адмін панелі можна подивитись усе, що потрібно: список замовлень, деталі по користувачах, змінити асортимент.

Техніко-економічне обґрунтування продемонструвало економічну доцільність проєкту. Коли прорахували витрати на команду – 68 760 грн, і порівняли з прогнозованим доходом/економією у 158 400 грн на рік, вийшло, що

проєкт окупається десь за 2,7 роки. І це без маркетингу. Як для ІТ-продукту з бізнес-складовою – це цілком адекватне рішення.

В результаті – маємо продукт, який справді можна впроваджувати. Він не просто існує – він вирішує реальні задачі, економить час, розширює можливості для користувачів і відкриває новий канал прибутку для клубу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Довідник з JavaScript. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript/Guide (дата звернення: 22.03.2025).
2. Документація бібліотеки bcrypt.js. npm : веб-сайт. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 04.04.2025).
3. Офіційна документація Express.js: веб-сайт. URL: https://expressjs.com/ (дата звернення: 15.04.2025).
4. Офіційна документація Tailwind CSS: веб-сайт. URL: https://tailwindcss.com/docs/ (дата звернення: 10.03.2025).
5. Робота з API «Нова Пошта». Документація для розробників: веб-сайт. URL: https://developers.novaposhta.ua/ (дата звернення: 20.03.2025).
6. JWT (JSON Web Tokens) Explained [відео] / Fireship. YouTube: веб-сайт. URL: https://www.youtube.com/watch?v=926mKnSW_vA (дата звернення: 19.05.2025).
7. Learn MERN Stack - Full Tutorial for Beginners (MongoDB, Express, React, Node.js) [/ freeCodeCamp.org. YouTube: веб-сайт. 12.10.2022. URL: https://www.youtube.com/watch?v=A63U_3-p6_Q (дата звернення: 11.05.2025).
8. Tailwind CSS Full Course for Beginners | Complete All-in-One Tutorial: веб-сайт. URL: https://www.youtube.com/watch?v=lCxcTsOHrjo (дата звернення: 25.05.2025).

ДОДАТКИ

Додаток А

Програмний код серверної частини

```
const express = require('express');
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const cors = require('cors');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const app = express();

app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
// ----- Middleware для авторизації //
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader?.split(' ')[1];
    if (!token) return res.status(401).json({ error: 'Відсутній
токен' });

    jwt.verify(token, 'secretkey', (err, user) => {
        if (err) return res.status(403).json({ error: 'Невірний
токен' });
        req.user = user;
        next();
    });
}
//Підключення до MongoDB//
mongoose.connect('mongodb://127.0.0.1:27017/auth_db', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
const userSchema = new mongoose.Schema({
    username: String,
    email: String,
    password: String,
    cart: [
        {
            productId: { type: mongoose.Schema.Types.ObjectId,
ref: 'Product' },
            quantity: { type: Number, default: 1 }
        }
    ],
```

```
    });
    const User = mongoose.model('User', userSchema);
    const productSchema = new mongoose.Schema({
        name: String,
        type: String,
        price: Number,
        description: String,
        imageUrl: String,
    });
    const Product = mongoose.model('Product', productSchema);

    app.get('/api/order', (req, res) => {
        const ordersFile = path.join(__dirname, 'orders.json');
        fs.readFile(ordersFile, 'utf8', (err, data) => {
            if (err) {
                if (err.code === 'ENOENT') {
                      console.warn("Файл orders.json не знайдено.
Повертається порожній масив.");
                     return res.json([]);
                }
               console.error("Помилка читання orders.json:", err);
                 return res.status(500).json({ error: 'Не вдалося
прочитати файл замовлень.' });
            }
            try {
                const orders = JSON.parse(data);
                res.json(orders);
            } catch (parseErr) {
                   console.error("Помилка парсингу orders.json:",
parseErr);
                res.status(500).json({ error: 'Не вдалося обробити
дані замовлень.' });
            }
        });
    });
    // Видалення замовлення
    app.delete('/api/order/:orderId',   authenticateToken,   async
(req, res) => {
        const { orderId } = req.params;

        if (!mongoose.Types.ObjectId.isValid(orderId)) {
              return  res.status(400).json({  error:  'Невірний  ID
замовлення' });
        }

        try {
            //Видалення з MongoDB
```

```
                        const    deletedOrderFromDB    =    await
Order.findByIdAndDelete(orderId);
            if (!deletedOrderFromDB) {

               console.warn(`Замовлення з ID ${orderId} не знайдено
в MongoDB, але спробуємо видалити з JSON.`);
            } else {
                  console.log(`✅ Замовлення ${orderId} успішно
видалено з MongoDB.`);
            }

            //Видалення з orders.json
            const ordersFile = path.join(__dirname, 'orders.json');
            let existingOrders = [];
            let orderFoundInJson = false;
            try {
               const fileData = fs.readFileSync(ordersFile, 'utf8');
                existingOrders = JSON.parse(fileData);
            } catch (readErr) {
                  console.error("Помилка читання orders.json при
видаленні:", readErr);
                if(deletedOrderFromDB) return res.json({ message:
'Замовлення видалено з бази даних, але виникла помилка при оновленні
файлу orders.json.' });
                return res.status(500).json({ error: 'Не вдалося
прочитати файл замовлень.' });
            }

            const updatedOrdersJson = existingOrders.filter(order
=> {
                if (order.orderId === orderId || order._id ===
orderId) {
                  orderFoundInJson = true;
                  return false;
                }
                return true;
            });

            if (!orderFoundInJson && !deletedOrderFromDB) {
                  return res.status(404).json({ error: 'Замовлення
з таким ID не знайдено ні в базі, ні у файлі.' });
            }

            if (orderFoundInJson) {
                                          fs.writeFile(ordersFile,
JSON.stringify(updatedOrdersJson, null, 2), (err) => {
                  if (err) {
```

```
                                console.error("Помилка при збереженні
оновленого orders.json після видалення:", err);
                                        if(deletedOrderFromDB)  return
res.json({ message: 'Замовлення видалено з бази даних, але виникла
помилка при оновленні файлу orders.json.' });
                                return res.status(500).json({ error: 'Не
вдалося оновити файл замовлень.' });
                            }
                             console.log(`✓ Замовлення ${orderId} успішно
видалено з orders.json`);
                                res.json({ message: 'Замовлення успішно
видалено!' });
                        });
                    } else if(deletedOrderFromDB) {
                         res.json({ message: 'Замовлення успішно видалено
з бази даних (у файлі не знайдено або не потребувало змін).' });
                    }

                } catch (error) {
                        console.error(`Помилка при видаленні замовлення
${orderId}:`, error);
                     res.status(500).json({ error: 'Помилка сервера при
видаленні замовлення.' });
                }
            });

            app.get('/api/enrollments', authenticateToken, (req, res) => {
                     const    enrollmentsFile    =    path.join(__dirname,
'enrollments.json');

                fs.readFile(enrollmentsFile, 'utf8', (err, data) => {
                    if (err) {
                        if (err.code === 'ENOENT') {
                           console.log("Файл enrollments.json не знайдено.
Повертається порожній масив.");
                            return res.json([]);
                        }
                        console.error("Помилка читання enrollments.json:",
err);
                         return res.status(500).json({ error: 'Не вдалося
прочитати файл записів на абонементи.' });
                    }
                    try {
                        const enrollments = JSON.parse(data);
                        res.json(enrollments);
                    } catch (parseErr) {
                       console.error("Помилка парсингу enrollments.json:",
parseErr);
```

```
                res.status(500).json({ error: 'Не вдалося обробити
дані записів на абонементи.' });
            }
        });
    });
    // Збереження даних запису на абонемент
    app.post('/api/enroll', authenticateToken, async (req, res) =>
{
        const { productId, firstName, lastName, age, phoneNumber }
= req.body;

          if (!productId || !firstName || !lastName || !age
|| !phoneNumber) {
              return res.status(400).json({ error: 'Будь ласка,
заповніть усі поля форми.' });
        }

        try {
            const user = await User.findById(req.user.id);
               if (!user) return res.status(404).json({ error:
'Користувача не знайдено.' });

                 const enrollmentsFile = path.join(__dirname,
'enrollments.json');
            let enrollments = [];

            try {
                   const data = fs.readFileSync(enrollmentsFile,
'utf8');
                enrollments = JSON.parse(data);
            } catch (readErr) {
               console.warn("Файл enrollments.json не знайдено або
він порожній. Створюємо новий.");
            }

            const newEnrollment = {
                id: Date.now().toString(),
                userId: user._id.toString(),
                username: user.username,
                email: user.email,
                productId: productId,
                firstName,
                lastName,
                age,
                phoneNumber,
                timestamp: new Date().toISOString()
            };
```

```
            enrollments.push(newEnrollment);

                                        fs.writeFile(enrollmentsFile,
JSON.stringify(enrollments, null, 2), (err) => {
                if (err) {
                    console.error("Помилка при збереженні запису на
абонемент у файл:", err);
                   return res.status(500).json({ error: 'Не вдалося
зберегти запис.' });
                   }
                 console.log(`✅Запис на абонемент успішно збережено
у ${enrollmentsFile}`);
                   res.json({ message: 'Ви успішно записалися на
абонемент!' });
            });

        } catch (error) {
            console.error("Помилка обробки запису на абонемент:",
error);
             res.status(500).json({ error: 'Помилка сервера при
обробці запису.' });
        }
    });
    const orderSchema = new mongoose.Schema({
          userId: { type: mongoose.Schema.Types.ObjectId, ref:
'User' },
        products: [
            {
                productId: { type: mongoose.Schema.Types.ObjectId,
ref: 'Product' },
                quantity: Number,
            }
        ],
        totalPrice: Number,
        customerInfo: {
            fullName: String,
            email: String,
            phone: String,
            comment: String,
        },
        shippingInfo: {
            cityRef: String,
            cityName: String,
            warehouseRef: String,
            warehouseName: String,

        },
        createdAt: { type: Date, default: Date.now }
```

```
    });
    const Order = mongoose.model('Order', orderSchema);

    // Очищення кошика
    app.delete("/api/cart/clear", authenticateToken, async (req,
res) => {
        try {
            const user = await User.findById(req.user.id);
                if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

            user.cart = [];
            await user.save();
            res.json({ message: "Кошик очищено" });
        } catch (error) {
            console.error("Помилка при очищенні кошика:", error);
             res.status(500).json({ error: 'Помилка при очищенні
кошика' });
        }
    });

    // Оформлення замовлення
    app.post('/api/order', authenticateToken, async (req, res) =>
{
        const { fullName, email, phone, comment, cityRef, cityName,
warehouseRef, warehouseName } = req.body;

          console.log("Отримані дані замовлення з фронтенду:",
{ fullName, email, phone, comment, cityRef, cityName, warehouseRef,
warehouseName });

        if (!fullName || !email || !phone || !cityRef || !cityName
|| !warehouseRef || !warehouseName) {
              return res.status(400).json({ error: 'Будь ласка,
заповніть усі обов’язкові поля для замовлення та доставки.' });
        }

        try {
                                const user = await
User.findById(req.user.id).populate('cart.productId');
                if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
                        if (user.cart.length === 0) return
res.status(400).json({ error: 'Кошик порожній, неможливо оформити
замовлення.' });

            let totalPrice = 0;
            let productsForOrder = [];
```

```
            let orderDescriptionItems = [];

            for (const item of user.cart) {
                const product = item.productId;
                if (!product) {
                    console.warn(`Товар з ID ${item.productId._id}
не знайдено в базі даних. Він буде пропущений.`);
                    continue;
                }
                totalPrice += product.price * item.quantity;
                productsForOrder.push({
                    productId: product._id,
                    name: product.name,
                    quantity: item.quantity,
                    price: product.price
                });
                        orderDescriptionItems.push(`${product.name}
(${item.quantity} шт.)`);
            }

            const orderDescription = orderDescriptionItems.join(',
');

            // Збереження замовлення в базу даних
            const newOrder = new Order({
                userId: user._id,
                products: productsForOrder,
                totalPrice: totalPrice,
                customerInfo: {
                    fullName,
                    email,
                    phone,
                    comment
                },
                shippingInfo: {
                    cityRef,
                    cityName,
                    warehouseRef,
                    warehouseName,
                },
                createdAt: new Date()
            });
            await newOrder.save();
           console.log('✅Замовлення успішно збережено в MongoDB:',
newOrder._id);

            // Збереження замовлення у файл orders.json
           const ordersFile = path.join(__dirname, "orders.json");
```

```
            let existingOrders = [];
            try {
               const fileData = fs.readFileSync(ordersFile, "utf8");
                existingOrders = JSON.parse(fileData);
            } catch (readErr) {
                console.warn("Файл orders.json не знайдено або він
порожній. Створюємо новий.");
            }

            const orderForJson = {
                orderId: newOrder._id.toString(),
                userId: user._id.toString(),
                customer: {
                    fullName,
                    email,
                    phone,
                    comment
                },
                items: productsForOrder,
                totalPrice: totalPrice,
                shipping: {
                    cityRef,
                    cityName,
                    warehouseRef,
                    warehouseName
                },
                description: orderDescription,
                createdAt: new Date().toISOString()
            };

            existingOrders.push(orderForJson);

           fs.writeFile(ordersFile, JSON.stringify(existingOrders,
null, 2), (err) => {
                if (err) {
                  console.error("Помилка при збереженні замовлення
у orders.json:", err);
                } else {
                    console.log("✅ Замовлення успішно збережено у
orders.json");
                }
            });

            // --- Очищення кошика користувача ---
            user.cart = [];
            await user.save();

            res.json({
```

```
                message: 'Замовлення оформлено успішно! Інформація
про доставку збережена.',
                orderId: newOrder._id,
                  // Більше не повертаємо novaPoshtaDocNumber та
novaPoshtaCost
            });
        } catch (err) {
                console.error("Помилка при оформленні замовлення
(сервер):", err.message);
             res.status(500).json({ error: 'Помилка при оформленні
замовлення: ' + err.message });
        }
    });

    // ----- Реєстрація (без змін) -----
    app.post('/register', async (req, res) => {
        const { username, email, password } = req.body;
        if (!username || !email || !password) {
                return  res.status(400).json({  error:  'Всі  поля
обов’язкові' });
        }
        try {
            const existingUser = await User.findOne({ email });
            if (existingUser) {
                return res.status(400).json({ error: 'Користувач з
таким email вже існує' });
            }
            const hashedPassword = await bcrypt.hash(password, 10);
              const user = new User({ username, email, password:
hashedPassword });
            await user.save();
            res.json({ message: 'Користувач зареєстрований!' });
        } catch (err) {
            console.error("Помилка реєстрації:", err);
            res.status(500).json({ error: 'Помилка реєстрації' });
        }
    });

    // ----- Логін (без змін) -----
    app.post('/login', async (req, res) => {
        const { email, password } = req.body;
        if (!email || !password) {
             return res.status(400).json({ error: 'Email і пароль
обов’язкові' });
        }
        try {
            const user = await User.findOne({ email });
```

```
            if (!user) return res.status(401).json({ error:
'Користувача не знайдено' });

            const isMatch = await bcrypt.compare(password,
user.password);
            if (!isMatch) return res.status(401).json({ error:
'Невірний пароль' });

            const token = jwt.sign({ id: user._id }, 'secretkey',
{ expiresIn: '1h' });
            res.json({ message: 'Успішний вхід', token });
        } catch (err) {
            console.error("Помилка входу:", err);
            res.status(500).json({ error: 'Помилка входу' });
        }
    });

    // ----- Профіль (без змін) -----
    app.get('/profile', authenticateToken, async (req, res) => {
        try {
            const user = await User.findById(req.user.id).select('-
password').populate('cart.productId');
            if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
            res.json(user);
        } catch (err) {
            console.error("Помилка профілю:", err);
            res.status(500).json({ error: 'Помилка сервера' });
        }
    });

    // ----- Продукти (без змін) -----
    app.post('/api/products', async (req, res) => {
        const { name, type, price, description, imageUrl } =
req.body;
        if (!name || !type || !price) {
            return res.status(400).json({ error: 'Відсутні
обов’язкові поля товару' });
        }
        try {
            const product = new Product({ name, type, price,
description, imageUrl });
            await product.save();
            res.status(201).json({ message: 'Товар додано' });
        } catch (error) {
            console.error("Помилка додавання товару:", error);
            res.status(500).json({ error: 'Помилка додавання
товару' });
```

```
        }
    });

    app.get('/api/products', async (req, res) => {
        try {
            const products = await Product.find();
            res.json(products);
        } catch (error) {
            console.error("Помилка отримання товарів:", error);
               res.status(500).json({ error: 'Помилка отримання
товарів' });
        }
    });

    app.delete('/api/products/:id', async (req, res) => {
        try {
                                const deleted = await
Product.findByIdAndDelete(req.params.id);
            if (!deleted) {
                 return res.status(404).json({ message: 'Товар не
знайдено' });
            }
            res.json({ message: 'Товар видалено' });
        } catch (error) {
            console.error("Помилка видалення товару:", error);
            res.status(500).json({ message: 'Помилка при видаленні
товару' });
        }
    });

    // ----- Кошик (без змін) -----
    app.post('/api/cart', authenticateToken, async (req, res) => {
        const { productId, quantity = 1 } = req.body;
          if (!productId) return res.status(400).json({ error:
'Вкажіть productId' });

        try {
            const user = await User.findById(req.user.id);
               if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

            const productExists = await Product.findById(productId);
            if (!productExists) {
                  return res.status(404).json({ error: 'Товар не
знайдено' });
            }
```

```
                const itemIndex = user.cart.findIndex(item =>
item.productId.toString() === productId);
            if (itemIndex > -1) {
                user.cart[itemIndex].quantity += quantity;
            } else {
                user.cart.push({ productId, quantity });
            }

            await user.save();
              res.json({ message: 'Товар додано в кошик', cart:
user.cart });
        } catch (error) {
            console.error("Помилка додавання в кошик:", error);
              res.status(500).json({ error: 'Помилка додавання в
кошик' });
        }
    });

    app.get('/api/cart', authenticateToken, async (req, res) => {
        try {
                                  const      user      =      await
User.findById(req.user.id).populate('cart.productId');
                if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
            res.json(user.cart);
        } catch (error) {
            console.error("Помилка отримання кошика:", error);
               res.status(500).json({ error: 'Помилка отримання
кошика' });
        }
    });

    app.put('/api/cart/:productId', authenticateToken, async (req,
res) => {
        const { quantity } = req.body;
        const { productId } = req.params;

        if (!quantity || quantity < 1) {
             return res.status(400).json({ error: 'Кількість має
бути не менше 1' });
        }

        try {
            const user = await User.findById(req.user.id);
               if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });
```

```
                const itemIndex = user.cart.findIndex(item =>
item.productId.toString() === productId);
            if (itemIndex === -1) {
                  return res.status(404).json({ error: 'Товар не
знайдено в кошику' });
            }

            user.cart[itemIndex].quantity = quantity;
            await user.save();
            res.json({ message: 'Кількість товару оновлена', cart:
user.cart });
        } catch (error) {
             console.error("Помилка оновлення кількості товару:",
error);
               res.status(500).json({ error: 'Помилка оновлення
кількості товару' });
        }
    });

    app.delete('/api/cart/:productId', authenticateToken, async
(req, res) => {
        const { productId } = req.params;
        try {
            const user = await User.findById(req.user.id);
               if (!user) return res.status(404).json({ error:
'Користувача не знайдено' });

                      user.cart = user.cart.filter(item =>
item.productId.toString() !== productId);
            await user.save();
            res.json({ message: 'Товар видалено з кошика' });
        } catch (error) {
            console.error("Помилка видалення товару:", error);
               res.status(500).json({ error: 'Помилка видалення
товару' });
        }
    });

    // ----- Видаляємо старий маршрут для orders.json, оскільки
новий /api/order його замінює -----
    // app.post("/api/orders", (req, res) => { /* ... */ });
    // Також видаляємо/коментуємо orderRouter, що конфліктував з
/api/order
    // const orderRouter = express.Router();
    // orderRouter.post('/', authenticateToken, async (req, res) =>
{ ... });
    // app.use('/api/order_old', orderRouter);
```

```
// ----- Статичні файли -----
app.get('/', (req, res) => {
        const indexPath = path.join(__dirname, 'public',
'index.html');
    if (fs.existsSync(indexPath)) {
        res.sendFile(indexPath);
    } else {
        res.send('<h1>Welcome to the server</h1>');
    }
});

// ----- Запуск сервера -----
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```