

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа супроводу клієнтів у сфері фітнесу»

Виконав студент групи ПЗ-22-4

ГОЛЬЦЕВ Костянтин Олександрович

Керівник роботи:

САВІЦЬКИЙ Роман Святославович

Рецензент:

ЄФРЕМОВ Юрій Миколайович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГОЛЬЦЕВ Костянтин Олександрович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Вебплатформа супроводу клієнтів у сфері фітнесу»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: Об'єктом дослідження є використання інформаційних технологій для організації роботи персональних тренерів та супроводу клієнтів у сфері фітнесу. Предметом дослідження є використання технологій розробки вебдодатків, побудова чистої та підтримуваної архітектури з використанням відповідних паттернів та принципів програмування.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	06.03.2026	06.03.2026
2	Савіцький Р.С.	06.03.2026	06.03.2026
3	Савіцький Р.С.	06.03.2026	06.03.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з розробленої онлайн-платформи та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 94 сторінки, 49 ілюстрацій та 11 таблиць.

Головною метою є створення продукту, який зможе стати альтернативою застарілим методам супроводу клієнтів у сфері фітнесу, зайнявши нішу на українському ринку.

У роботі визначено ключові завдання проєкту, проведено аналіз існуючих рішень на українському та закордонному ринках, наведено сценарії використання продукту. Було обґрунтовано вибір технологій та архітектури, описані модулі системи й основні алгоритми їх взаємодії.

Вебплатформу реалізовано з поділом на серверну та клієнтську частини. Серверну частину побудовано на основі ASP.NET Core з архітектурою модульного моноліту. Для зберігання даних використано PostgreSQL, а контейнеризацію компонентів виконано за допомогою Docker. Клієнтську частину реалізовано на основі Next.js, React і TypeScript, що дало змогу поєднати серверне завантаження даних з інтерактивністю вебінтерфейсу. Для автоматизації перевірки змін, запуску тестів і підготовки контейнерних образів налаштовано GitHub Actions.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, СУПРОВІД, ОНЛАЙН-ТРЕНУВАННЯ, ЧИСТА АРХІТЕКТУРА, .NET, NEXT.JS, ТЕСТУВАННЯ, CI/CD.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Гольцев К.О.			Вебплатформа супроводу клієнтів у сфері фітнесу	Літ.	Арк.	Аркушів
Керівник		Савіцький Р.С.					3	110
Рецензент		Єфремов Ю. М.				Житомирська політехніка, група ІПЗ-22-4		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's qualification work consists of the developed online platform and an explanatory note. The explanatory note contains 94 pages, 49 illustrations, and 11 tables.

The main goal is to create a product that can become an alternative to outdated methods of client support in online training and occupy a niche in the Ukrainian market.

The work defines the key objectives of the project, analyzes existing solutions in the Ukrainian and foreign markets, and presents the main use cases of the product. The choice of technologies and architecture is substantiated, and the system modules and main algorithms of their interaction are described.

The web platform is implemented with a division into server-side and client-side parts. The server-side part is built using ASP.NET Core and a modular monolith architecture. PostgreSQL is used for data storage, and Docker is used for component containerization. The client-side part is implemented using Next.js, React, and TypeScript, which made it possible to combine server-side data loading with the interactivity of the web interface. GitHub Actions is configured to automate change verification, test execution, and preparation of container images.

KEYWORDS: WEB PLATFORM, SUPPORT, ONLINE TRAINING, CLEAN ARCHITECTURE, .NET, NEXT.JS, TESTING, CI/CD.

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	9
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	12
1.1. Аналіз предметної області та аналогів програмного продукту	12
1.2. Постановка задачі та план розробки.....	17
1.3. Вибір архітектури системи	20
1.4. Обґрунтування вибору технологій та засобів розробки	24
Висновки до першого розділу	27
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ СУПРОВОДУ КЛІЄНТІВ У СФЕРІ ФІТНЕСУ «FITLEAD»	29
2.1. Визначення варіантів використання та проектування об'єктно- орієнтованої структури системи	29
2.2. Розробка бази даних системи	40
2.3. Проектування алгоритмів системи	43
2.4. Реалізація вебплатформи супроводу клієнтів у сфері фітнесу	54
Висновки до другого розділу.....	67
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ	69
3.1. Розгортання вебплатформи та налаштування неперервної інтеграції	69
3.2. Опис інтерфейсу вебплатформи.....	73
3.3. Тестування роботи програмного продукту	84
Висновки до третього розділу	87
ВИСНОВКИ	89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	91

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ.....	95
Додаток А	96
Додаток Б.....	107

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіицький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД (база даних) – організована сукупність даних, що зберігається та обробляється в інформаційній системі.

ПЗ (програмне забезпечення) – сукупність програм, даних і супровідних матеріалів, які забезпечують роботу комп'ютерної системи.

API (Application Programming Interface) – інтерфейс прикладного програмування, який визначає спосіб взаємодії між програмними компонентами.

BFF (Backend for Frontend) – серверний проміжний шар, який адаптує запити клієнтської частини до backend-сервісів.

CA (Clean Architecture) – чиста архітектура, підхід до побудови програмного забезпечення з розділенням відповідальності між шарами системи.

CI/CD (Continuous Integration / Continuous Delivery) – безперервна інтеграція та безперервне постачання або розгортання програмного забезпечення.

CQRS (Command Query Responsibility Segregation) – підхід, за якого операції зміни стану системи відокремлюються від операцій читання даних.

CSRF (Cross-Site Request Forgery) – тип атаки, за якого зловмисник змушує браузер користувача виконати небажаний запит до вебзастосунку.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, що використовуються для опису зовнішнього вигляду вебсторінок.

DDD (Domain-Driven Design) – підхід до проєктування програмного забезпечення, за якого основна увага приділяється моделі предметної області та бізнес-правилам.

DTO (Data Transfer Object) – об'єкт передачі даних між частинами системи.

EF (Entity Framework) – фреймворк для роботи з базами даних у .NET через об'єктно-реляційне відображення.

Endpoint – конкретна URL-адреса або маршрут, за яким API приймає запити для доступу до ресурсів системи або виконання певних операцій.

HTML (HyperText Markup Language) – мова розмітки гіпертексту, що використовується для створення структури вебсторінок.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

HTTP/HTTPS (HyperText Transfer Protocol / HyperText Transfer Protocol Secure) – протокол передавання гіпертексту та його захищена версія.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на об’єктній нотації JavaScript.

LCP (Largest Contentful Paint) – метрика продуктивності вебсторінки, що показує час завантаження найбільшого видимого елемента сторінки.

ORM (Object-Relational Mapping) – об’єктно-реляційне відображення, підхід до роботи з реляційною базою даних через об’єкти мови програмування.

REST (Representational State Transfer) – архітектурний стиль побудови мережових сервісів.

SEO (Search Engine Optimization) – пошукова оптимізація, набір підходів для покращення видимості вебсторінок у пошукових системах.

TBT (Total Blocking Time) – метрика продуктивності вебсторінки, що показує загальний час блокування основного потоку виконання JavaScript.

UX/UI (User Experience / User Interface) – користувацький досвід і користувацький інтерфейс, що поєднують функціональну зручність та візуальне оформлення програмного продукту.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

Актуальність розробки платформи FitLead пов'язана з тим, що фітнес-послуги дедалі частіше використовують цифрові канали взаємодії. За оцінкою Allied Market Research, світовий ринок онлайн-фітнесу у 2022 році становив 14,9 млрд доларів США, а до 2032 року прогнозується його зростання до 250,7 млрд доларів США із середньорічним темпом зростання 32,7% [1]. Це не є прямим доказом зростання попиту саме на українському ринку, однак свідчить про економічну значущість наряду онлайн-фітнесу та про загальну тенденцію до перенесення частини фітнес-послуг у цифрове середовище.

Для українського ринку можна констатувати наявність онлайн-пропозиції фітнес-послуг. В основному, поширення послуг тренерами відбувається через соціальні мережі, тому масштаби важко оцінити кількісно. Проте на платформі BUKI є окрема категорія онлайн-занять із фітнес-тренерами, що налічує 215 тренерів [2]. Це підтверджує, що онлайн-формат роботи з тренером уже використовується в Україні.

Додатковим чинником актуальності є зміна умов комунікації між людьми після початку повномасштабного вторгнення. Війна спричинила масштабне переміщення населення як усередині країни, так і за її межі. За даними ІОМ, станом на грудень 2025 року в Україні залишалося близько 3,7 млн внутрішньо переміщених осіб [3]. За таких умов дистанційна взаємодія між тренером і клієнтом стає практично важливою, оскільки клієнт може змінити місце проживання, але зберігати потребу в регулярних тренуваннях, консультаціях і контролі прогресу.

Відповідно, в Україні за останні роки зріс рівень цифрової залученості населення і можливість користування онлайн-сервісами. За даними DataReportal, на початок 2024 року частка населення України, яке користувалося інтернетом, становила 79,2%. У звіті Digital 2026 зазначено, що наприкінці 2025 року цей показник становив 89,6% [4, 5].

Окрему проблему становить фрагментованість інструментів, які використовуються для онлайн-супроводу клієнтів. На практиці тренер може

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

використовувати месенджери для комунікації, таблиці для складання тренувальних програм та окремі застосунки для фіксації прогресу і збору статистики. Такий підхід ускладнює системне ведення клієнта, оскільки інформація про програми, виконані тренування, рекомендації та комунікацію зберігається в різних середовищах. Для клієнта це також створює незручності, оскільки процес тренувань не має єдиного цифрового простору.

З огляду на це розробка платформи FitLead є актуальною як спроба централізувати основні процеси онлайн-взаємодії тренера й клієнта. Система орієнтована на об'єднання функцій створення тренувальних програм, комунікації, ведення журналу тренувань та відстеження прогресу в межах одного вебсередовища. Такий підхід дозволяє зменшити залежність тренера від набору сервісів і створити більш структурований процес супроводу клієнта незалежно від місця його перебування.

Метою кваліфікаційної роботи є проектування та розробка вебплатформи, що дозволяє мінімізувати бар'єр переходу з існуючих інструментів і сприяє швидкому впровадженню сервісу в роботу тренерів.

Для досягнення поставленої мети визначено наступні **завдання**:

- проаналізувати предметну область онлайн-супроводу клієнтів у сфері фітнесу та наявні програмні рішення для взаємодії між тренером і клієнтом;
- сформулювати функціональні вимоги до вебплатформи та визначити основні сценарії роботи тренера і клієнта;
- спроектувати архітектуру системи, модель предметної області та структуру бази даних;
- реалізувати серверну та клієнтську частини FitLead;
- налаштувати автоматизовану перевірку, тестування, контейнеризацію та розгорнути вебплатформу в серверному середовищі.

Об'єктом дослідження є процес онлайн-супроводу клієнтів персональними тренерами, що включає створення тренувальних програм, комунікацію з клієнтами та відстеження прогресу клієнта.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є методи та засоби розробки програмного забезпечення для організації процесу онлайн-взаємодії між персональним тренером і клієнтом, архітектура та внутрішня організація коду для реалізації підтримуваного та масштабованого ПЗ.

Практичне застосування результатів роботи полягає у використанні розробленої вебплатформи «FitLead» персональними тренерами для організації онлайн-роботи з клієнтами. Використання єдиної системи дозволяє зменшити кількість сторонніх інструментів, підвищити зручність взаємодії між тренером і клієнтом та оптимізувати процес супроводу клієнтів. Розроблений програмний продукт може бути запущений як пробну версію з доступом обмеженій кількості користувачів для збору відгуків та подальших рішень щодо розвитку продукту.

У тезах було розглянуто вибір оптимальної технології розробки та способи побудови програмного забезпечення з використанням принципів чистої архітектури та шаблонів програмування [6, 7].

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Аналіз предметної області та аналогів програмного продукту

Для визначення місця платформи FitLead на ринку було проведено аналіз існуючих програмних рішень, що використовуються для онлайн-супроводу клієнтів у сфері фітнесу. Розгляд охоплює міжнародні програмні продукти, які формують сучасні підходи до організації дистанційної взаємодії між тренером і клієнтом, а також рішення, доступні користувачам на українському ринку.

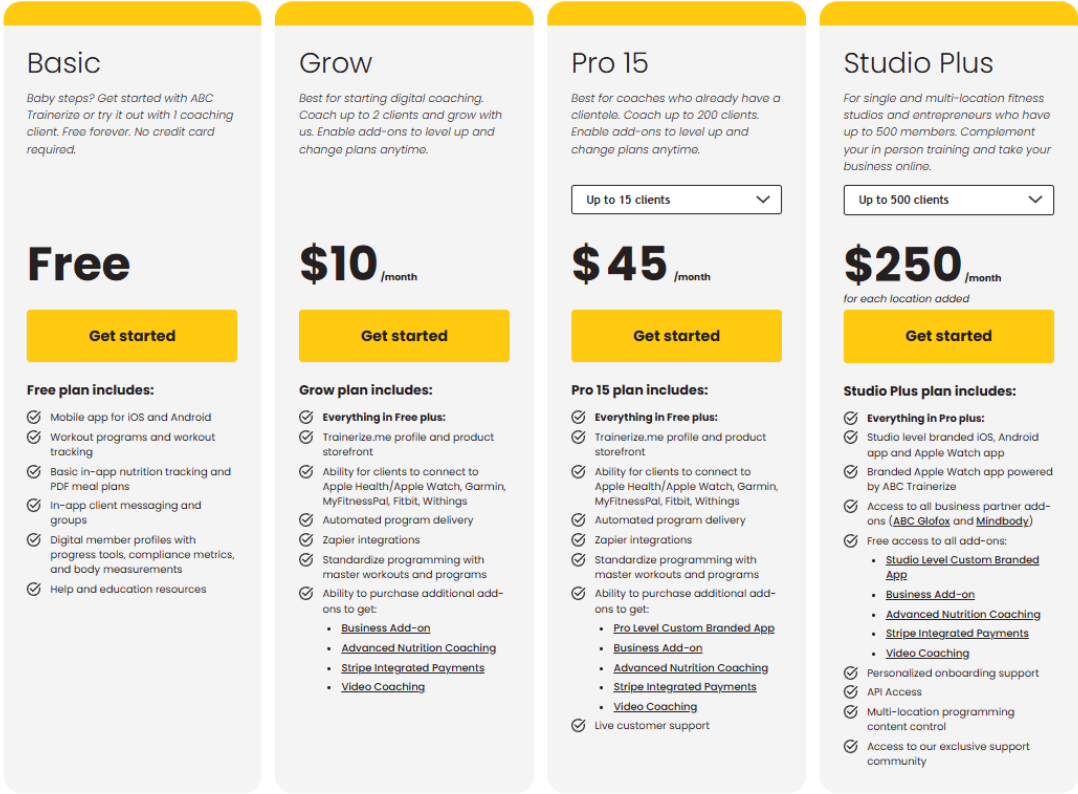
Результати огляду показують, що більшість розвинених платформ орієнтовані на глобальний ринок і враховують потреби користувачів із високим рівнем цифрової інтеграції сервісів. Такі рішення передбачають використання платіжних систем, які не є поширеними в Україні, а також містять значну кількість функціональних можливостей, орієнтованих на масштабовані бізнес-моделі або командну роботу. Це може ускладнювати використання подібних продуктів персональними тренерами, які лише починають впроваджувати онлайн-супровід клієнтів.

Водночас програмні рішення, представлені на українському ринку, здебільшого орієнтовані на автоматизацію роботи фітнес-клубів, студій або інших організацій, що надають послуги офлайн. Такі системи фокусуються на управлінні розкладом занять, бронюванням та обліком відвідувань, що не повністю відповідає специфіці індивідуальної роботи персонального тренера з клієнтами онлайн. Також існують окремі застосунки, що частково підтримують взаємодію між тренером і клієнтом, проте вони, як правило, покривають лише окремі аспекти процесу онлайн-супроводу та не забезпечують комплексної підтримки формування програм тренувань, комунікації та відстеження прогресу у межах єдиного середовища.

ABC Trainerize є однією з найбільш відомих міжнародних платформ для онлайн-тренувань та супроводу, яка використовується персональними тренерами для управління клієнтами, створення тренувальних програм та монетизації послуг [8]. Система поєднує інструменти планування тренувань,

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

комунікації, відстеження результатів та управління бізнес-процесами у межах єдиного цифрового середовища. На рисунку 1.1 зображено ціни та переваги, що надає застосунок у залежності від обраної підписки.



All prices are in USD and subject to applicable taxes. Annual plans are billed once per year.

Рис. 1.1. Цінова політика та переваги кожної підписки ABC Trainerize

Сервіс орієнтований на глобальний ринок і підтримує інтеграції з різними фітнес-додатками та сторонніми сервісами, що дозволяє автоматично отримувати дані про активність користувачів.

З точки зору позиціонування, Trainerize є повнофункціональною платформою, що охоплює не лише процес створення тренувань, але і бізнес-аспекти роботи тренера, включаючи продаж послуг, брендові мобільні додатки та управління клієнтською базою. Водночас значна кількість функцій та орієнтація на глобальний ринок можуть ускладнювати освоєння системи для індивідуальних тренерів, які працюють у локальному контексті або лише починають онлайн-супровід.

Trainero – це комплексна платформа призначена для персональних тренерів, фітнес-тренерів та спортивних студій [9]. Система поєднує інструменти планування тренувань, управління клієнтами, комунікації та монетизації послуг

у межах єдиного програмного середовища. Платформа складається з окремих додатків для тренера і клієнта, а також CRM-модуля для управління бізнесом. Тренер може створювати індивідуальні програми тренувань і харчування, вести базу клієнтів, організовувати групові заняття та відстежувати результати тренувального процесу. Клієнти отримують доступ до персонального кабінету, де можуть переглядати тренування, фіксувати прогрес і взаємодіяти з тренером (рис. 1.2).

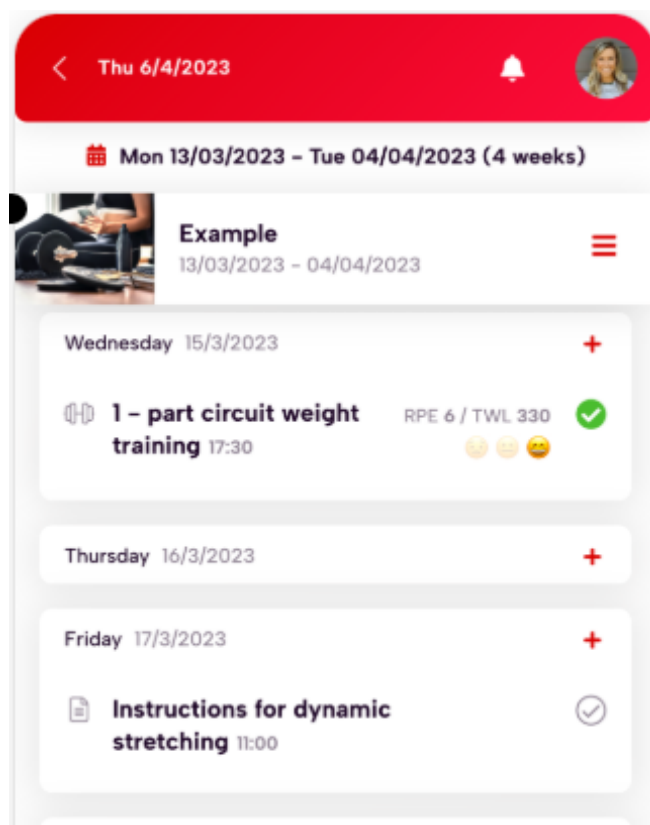


Рис. 1.2. Вигляд клієнтського додатку Trainero

Сервіс орієнтований на глобальний ринок і підтримує роботу тренерів різних напрямків фітнесу, включаючи онлайн-тренування і роботу з групами клієнтів. Також доступні рішення, що дозволяють використовувати платформу під власним брендом.

З точки зору позиціонування, Trainero є універсальною платформою, що поєднує функціонал планування тренувань, комунікації та управління бізнесом тренера. Водночас система орієнтована на міжнародний ринок і містить широкий набір функцій, що може ускладнювати початкове освоєння для індивідуальних тренерів на українському ринку.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

Trainerfu – це програмна платформа для онлайн-персональних тренерів, яка надає базовий набір інструментів для створення тренувальних програм, комунікації з клієнтами та відстеження їх прогресу [10]. Система орієнтована на індивідуальних тренерів і невеликі команди, які надають послуги дистанційного супроводу клієнтів.

Платформа дозволяє формувати індивідуальні та групові програми тренувань із використанням бібліотеки вправ, контролювати виконання тренувань і фіксувати результати клієнтів. Trainerfu також включає інструменти взаємодії між тренером і клієнтом, зокрема повідомлення та стрічку активності, що дозволяє підтримувати регулярний зворотний зв'язок у процесі тренування.

Додатково сервіс підтримує можливість поширення навчальних матеріалів і продажу онлайн-програм, що дозволяє тренерам масштабувати власні послуги. Платформа доступна у вигляді веб-застосунку та мобільного додатку, що забезпечує зручний доступ до тренувальних планів і результатів.

Trainerfu є прикладом відносно простішого рішення у порівнянні з більш комплексними системами, такими як Trainerize. Trainerfu підтверджує наявність базового стандарту функціоналу для систем онлайн-супроводу клієнтів, що включає конструктор тренувань, відстеження прогресу та інструменти комунікації (рис. 1.3).

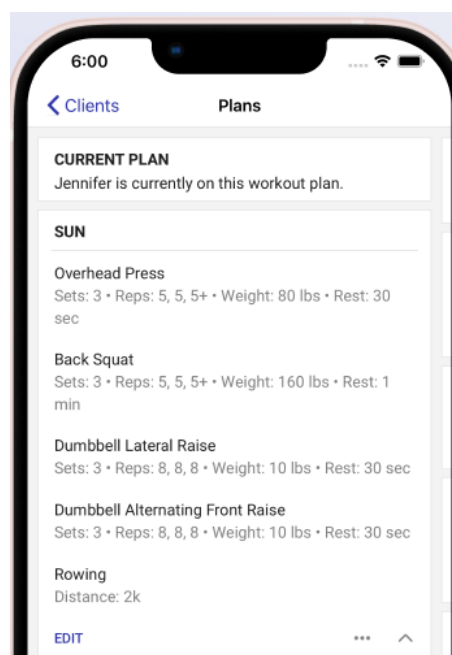


Рис. 1.3. Вигляд програми тренувань у Trainerfu

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

У порівняльній табл. 1.1 наведено критерії та функції описаних вище аналогів. Порівняння виконано за функціональними та користувацькими характеристиками, які є важливими для організації онлайн-взаємодії між персональним тренером і клієнтом. Оцінювання враховує наявність ключових модулів, локалізацію, підтримку комунікації, можливість відстеження прогресу та складність початкового освоєння системи.

Таблиця 1.1

Порівняння з аналогами

Характеристика	Майбутня розробка	Trainerize	Trainero	Trainerfu
Локалізація під український ринок	+	-	-	-
Фокус на персональних тренерів	Так	Частково	Частково	Так
Конструктор тренувальних програм	+	+	+	+
База вправ тренера	+	+	+	+
Відстеження прогресу клієнта	+	+	+	+
Комунікація тренер-клієнт (чат)	+	+	+	+
Публікація матеріалів тренером	-	+	+	-
Відеозвіти клієнта з відгуком про техніку	+	Частково	Частково	+
Групові програми	-	+	+	+
Орієнтована складність освоєння	Легко	Важко	Середньо	Середньо
Адаптація до мобільних пристроїв	+	+	+	+
Окремі мобільні додатки	-	+	+	+

Під складністю освоєння розуміється орієнтовна кількість зусиль, необхідна користувачу для початку роботи із системою. Оцінка враховує

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

кількість модулів, кількість налаштувань, складність навігації та час, потрібний для виконання базових сценаріїв, таких як створення програми, додавання клієнта й перегляд прогресу.

Під фокусом на персональних тренерах у порівнянні розуміється те, наскільки система орієнтована саме на індивідуальну роботу тренера з клієнтом, а не на ширше управління фітнес-бізнесом, студією або груповими програмами. Оцінка “Так” використовується для систем, у яких основні сценарії побудовані навколо персонального тренера, його клієнтів, тренувальних програм, комунікації та контролю прогресу. Оцінка “Частково” означає, що система підтримує персональних тренерів, але також значною мірою орієнтована на ширші сценарії: роботу фітнес-студій, групові програми, CRM, платежі, маркетинг або управління командою.

Відеозвіти оцінювались не лише за можливістю надсилання відеофайлів, а й за наявністю окремого сценарію перевірки техніки виконання вправ і надання відгуку тренером.

Результати аналізу аналогів дозволили сформулювати перелік ключових функціональних вимог до системи. До них належать конструктор тренувальних програм, засоби комунікації між тренером і клієнтом та відстеження прогресу. Важливою характеристикою обрано простоту використання системи, що зменшує бар’єр переходу від звичних інструментів, таких як електронні таблиці та месенджери.

Отже, розробка платформи FitLead спрямована на поєднання базових можливостей сучасних систем онлайн-супроводу із адаптацією до умов локального ринку, що дозволяє отримати конкурентну перевагу за рахунок доступності, зрозумілості та орієнтації на потреби персональних тренерів.

1.2. Постановка задачі та план розробки

Після проведення аналізу основних конкурентів та особливостей процесу онлайн-супроводу клієнтів у сфері персонального тренінгу, було визначено основний функціонал та додаткові вимоги, що в сукупності створюють унікальну пропозицію та цінність для користувача. Відповідно задача полягає у розробці

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

програмного рішення для організації онлайн-супроводу клієнтів у сфері персонального фітнес-тренінгу у вигляді вебплатформи, що дозволить об'єднати ключові інструменти взаємодії між тренером і клієнтом в межах єдиного цифрового середовища, задовольнивши їх потреби та очікування.

Система повинна бути орієнтована на простоту використання, щоб мінімізувати бар'єр переходу з існуючих інструментів та забезпечити швидке впровадження в роботу тренерів. Також важливо відповідати очікуванням сучасних користувачів. Це включає не тільки зручність, а і швидкодію. Адаптація до мобільного інтерфейсу є важливою вимогою, адже велика частина користувачів буде користуватися сервісом з телефонів. Важливим аспектом є забезпечення безпеки даних користувачів, оскільки система передбачає зберігання персональної інформації. Необхідно реалізувати механізми автентифікації, авторизації та перевірку коректності вхідних даних.

Для реалізації зазначених сценаріїв потрібно спроектувати модель даних системи та визначити основні зв'язки між її сутностями. У системі мають бути представлені користувачі з ролями тренера і клієнта, зв'язок між тренером і клієнтом, запрошення, вправи, тренування, тренувальні програми, призначення програм, журнал виконання тренувань, чати, відеозвіти, медіафайли та сповіщення. Також необхідно передбачити правила доступу до цих даних, щоб тренер міг працювати лише зі своїми клієнтами, а клієнт мав доступ тільки до власних програм, повідомлень, звітів і показників прогресу.

Першим етапом є проектування архітектури програмного забезпечення та визначення технологічного стеку, оптимального для реалізації визначеного функціоналу. Це передбачає визначення структури системи та способів взаємодії її компонентів, вибір технологій та обґрунтування прийнятих рішень. Архітектура повинна забезпечувати можливість масштабування, підтримку та розширення функціоналу у майбутньому без надмірних витрат ресурсів.

Другим етапом є реалізація базової серверної та клієнтської частин застосунку відповідно до обраного стеку та архітектури. Після формування базової архітектури, розробка застосунку виконується поступово, через

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

реалізацію окремих функціональних частин. Кожна така частина охоплює завершений користувацький сценарій або його логічний фрагмент. Наприклад, окремо реалізуються авторизація, робота із запрошеннями, створення тренувальних програм, ведення журналу тренувань та інші функції. Такий підхід дає змогу зменшити складність розробки, перевіряти функціональність після кожного етапу та підтримувати стабільний стан системи протягом усього процесу створення програмного продукту.

Тестування програмного забезпечення здійснюється паралельно з процесом розробки функціоналу та включає перевірку коректності роботи окремих модулів, інтеграцій між компонентами системи та основних сценаріїв взаємодії користувача із застосунком. Використання тестування дозволяє своєчасно виявляти помилки та забезпечувати стабільність роботи програмного продукту.

Також потрібно налаштувати автоматизовану перевірку змін у коді. У межах CI/CD мають виконуватися збірка серверної та клієнтської частин, запуск інтеграційних тестів backend, перевірка frontend-проєкту та браузерні E2E smoke-тести основних сценаріїв. Після успішних перевірок у головній гілці повинні формуватися контейнерні образи, які надалі використовуються для запуску системи в середовищі розгортання.

Окремою частиною задачі є підготовка застосунку до запуску в середовищі, наближеному до реального. Для цього серверна та клієнтська частини повинні збиратися в окремі Docker-образи, а налаштування застосунку мають передаватися через змінні середовища без збереження секретів у відкритому коді. Середовище розгортання повинно забезпечувати запуск контейнерів, підключення до PostgreSQL, маршрутизацію зовнішніх HTTPS-запитів, SignalR-з'єднань, Uploadcare та Web Push.

Результатом виконання поставленої задачі є вебплатформа для онлайн-взаємодії між персональним тренером і клієнтом. Система повинна підтримувати основні сценарії супроводу клієнта, зокрема авторизацію, запрошення, створення тренувальних програм, ведення журналу тренувань, чат, відеозвіти,

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

сповіщення та перегляд прогресу. Розроблений застосунок має проходити автоматизовані перевірки та розгортатися на сервері у контейнеризованому середовищі. Після розгортання система повинна бути готовою до використання за основними сценаріями роботи тренера і клієнта.

1.3. Вибір архітектури системи

Для реалізації платформи FitLead обрано архітектуру модульного моноліту (рис. 1.4) [11]. Такий підхід дозволяє поєднати відносну простоту розгортання та розробки з достатнім рівнем структурованості системи. Основною вимогою до архітектури було забезпечення можливості поступового розширення функціоналу без суттєвого ускладнення інфраструктури та процесу розробки.

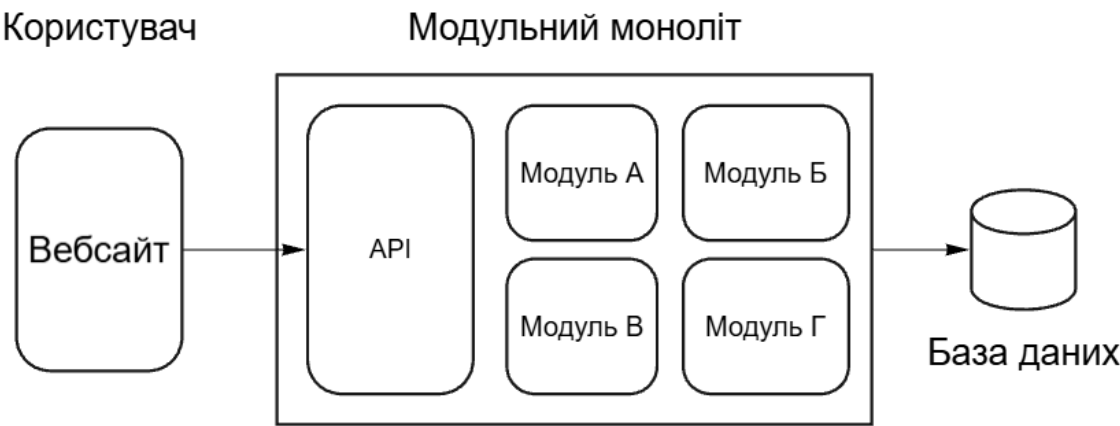


Рис. 1.4. Структура модульного моноліту

Система FitLead включає декілька логічно відокремлених частин предметної області, зокрема управління користувачами, формування тренувальних програм, відстеження прогресу клієнтів та комунікацію між тренером і клієнтом. Дані частини мають власну бізнес-логіку, проте тісно взаємодіють між собою. У такому випадку важливо забезпечити чітке розділення відповідальності між компонентами системи, зменшити зв’язаність коду та забезпечити можливість незалежного розвитку окремих модулів.

Альтернативним варіантом могла бути мікросервісна архітектура [12]. Вона передбачає поділ системи на незалежні сервіси, кожен з яких має свою бізнес-логіку, базу даних, а спілкування між ними організовується асинхронно,

за допомогою проміжного ПЗ – брокерів повідомлень. Це сприяє слабкому зв'язку, підтримуючи високу масштабованість (рис. 1.5).

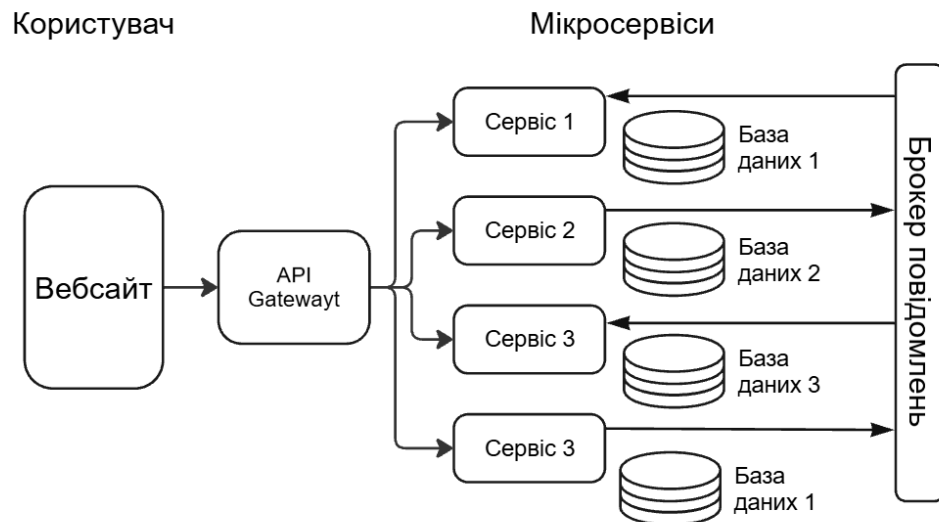


Рис. 1.5. Мікросервісна архітектура

Мікросервіси дозволяють масштабувати окремі частини системи незалежно та ізолювати зміни в межах окремого сервісу. Проте такий підхід суттєво ускладнює процес розробки і розгортання системи, оскільки потребує налаштування взаємодії між сервісами, механізмів обробки розподілених помилок, моніторингу та підтримки інфраструктури. Для проєкту середнього масштабу, що розробляється як кваліфікаційна робота, така складність є невиправданою і не створює суттєвих переваг на поточному етапі.

Класична монолітна архітектура, у якій усі компоненти системи реалізовані без чіткого поділу на модулі, не розглядалася через ризик зростання зв'язаності між компонентами. У довгостроковій перспективі це ускладнює внесення змін, тестування та підтримку системи, що не відповідає вимогам.

Отже, архітектура FitLead реалізована у вигляді модульного моноліту. Усі модулі працюють у межах одного серверного застосунку, однак їхня логіка розділена за межами предметних областей. Наприклад, модуль тренувань відповідає за вправи, тренування та програми, а модуль користувачів обробляє дані користувачів, ролі та зв'язки між тренером і клієнтом. Такий поділ не створює окремих сервісів, але дає змогу не змішувати логіку різних частин системи в одному шарі коду.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 1.6 показано загальну схему взаємодії компонентів у межах обраної архітектури. Клієнтська частина, реалізована на Next.js, надсилає HTTP-запити до ASP.NET API. API не містить основної бізнес-логіки, а приймає запит і передає відповідну команду або запит у MediatR. Далі MediatR визначає потрібний обробник у відповідному модулі та передає йому керування. Завдяки цьому API-рівень залишається тонким, а прикладна логіка зосереджується всередині модулів.

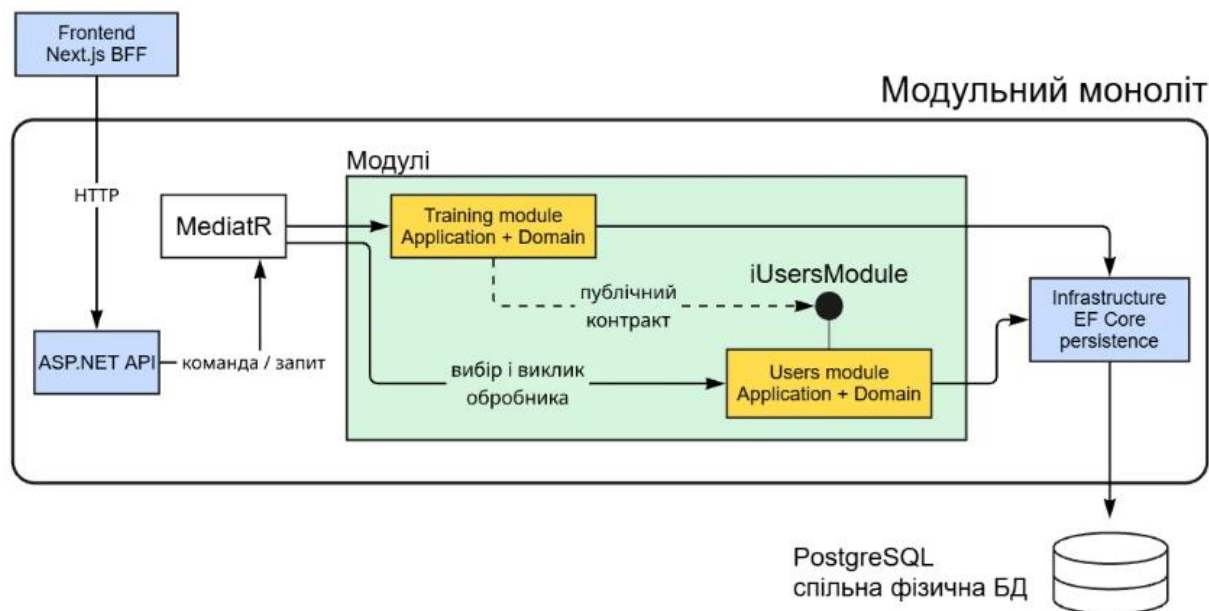


Рис. 1.6. Взаємодія компонентів у модульному моноліті FitLead

Взаємодія між модулями не виконується через прямий доступ до внутрішніх класів або таблиць іншого модуля. Якщо одному модулю потрібні дані або дія іншого модуля, він звертається до нього через публічний контракт. На схемі це показано на прикладі взаємодії Training module з Users module через інтерфейс IUsersModule. Такий підхід зберігає межі модулів і зменшує залежність між їхніми внутрішніми реалізаціями.

Робота з базою даних винесена в інфраструктурний шар. Для збереження даних використовується EF Core persistence та спільна фізична база PostgreSQL. При цьому модулі не повинні працювати з даними один одного напряму. Спільна база спрощує розгортання і підтримку системи, але логічне розділення модулів зберігається на рівні коду, контрактів і правил доступу до даних.

Такий варіант архітектури не забезпечує повної незалежності компонентів, як у мікросервісах, однак дозволяє уникнути надмірної архітектурної та інфраструктурної складності [13]. Система не потребує брокерів повідомлень і складної підтримки розподілених транзакцій. Водночас модулі залишаються відокремленими за відповідальністю, що спрощує розробку, тестування та подальше розширення функціоналу.

Для організації внутрішньої структури системи використано принципи Clean Architecture, які передбачають відокремлення бізнес-логіки від інфраструктурних компонентів. У межах даного підходу центральною частиною системи є модель предметної області та прикладна логіка, тоді як взаємодія з базою даних, зовнішніми сервісами та інтерфейсом користувача реалізується у зовнішніх шарах: Presentation та Infrastructure відповідно. Такий підхід дозволяє змінювати технічні аспекти реалізації без впливу на основну логіку системи, спрощує тестування та підвищує стабільність програмного продукту.

Для реалізації доменної логіки використано принципи Domain-Driven Design [14]. У межах цього підходу модель будується навколо бізнес-правил та інваріантів, які контролюють зміни та забезпечують цілісність логіки незалежно від реалізацій зовнішніх шарів. Вибір даного підходу дозволив ізолювати ключові процеси системи для їх незалежного тестування та гарантував стійкість ядра системи до змін у вимогах або інфраструктурних рішеннях. Для організації прикладної логіки використовується підхід Command Query Responsibility Segregation, який передбачає розділення операцій зміни стану системи та операцій отримання даних. Такий підхід спрощує структуру коду, полегшує тестування та дозволяє гнучко розширювати функціонал без ускладнення існуючих компонентів.

Отже, використання архітектури модульного моноліту у поєднанні з принципами чистої архітектури, методом розділення операцій та розробки з фокусом на предметній області дозволяє забезпечити структурованість системи, зручність підтримки та можливість подальшого розвитку платформи FitLead без надмірного ускладнення інфраструктури.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4. Обґрунтування вибору технологій та засобів розробки

Платформа .NET та мова програмування C# обрані для реалізації серверної частини системи. Стабільність платформи, довгострокова підтримка від Microsoft, висока продуктивність та розвинена екосистема інструментів для розробки серверних застосунків стали основними критеріями вибору. .NET має добре розвинені засоби тестування, логування та конфігурації застосунків, що спрощує підтримку програмного продукту у довгостроковій перспективі. За даними досліджень використання технологій розробниками C# та .NET зберігають стабільну популярність протягом тривалого часу з активним використанням у промисловій розробці [15]. Формування технологічного стеку програмної платформи FitLead враховує функціональні вимоги до системи, обрану архітектуру модульного моноліту, а також потребу підтримки, розширення та стабільності програмного продукту. Засоби розробки орієнтовані на зрілі технології з розвинутою екосистемою для реалізації складної бізнес-логіки, полегшення тестування системи та підтримки її розвитку у довгостроковій перспективі.

Платформа підтримує ефективну модель асинхронного програмування з обробкою великої кількості одночасних запитів без значного збільшення використання ресурсів сервера. Завдяки оптимізації середовища виконання та вебсерверу Kestrel ASP.NET Core демонструє високі показники продуктивності у незалежних тестах вебфреймворків [16]. Для реалізації API використовується ASP.NET Core [17] з підтримкою обробки HTTP-запитів та механізмів автентифікації і авторизації, що дозволяє реалізувати безпечну взаємодію між клієнтом і сервером. Конвеєр проміжного програмного забезпечення (middleware) у ASP.NET Core надає гнучкі засоби для додавання наскрізної функціональності, як-от обробки помилок, журналювання, кешування та валідації запитів. Інтегрована система впровадження залежностей спрощує організацію коду відповідно до принципів SOLID та полегшує модульне тестування.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Як систему управління базами даних обрано PostgreSQL – об'єктно-реляційну СУБД з відкритим вихідним кодом, яка широко використовується у вебзастосунках різного масштабу [18]. PostgreSQL підтримує розширений набір типів даних, зокрема JSON, JSONB, масиви та користувацькі типи, що дозволяє гнучко моделювати складні структури даних без надмірної денормалізації. Транзакційна модель з підтримкою ACID-властивостей забезпечує цілісність даних навіть у разі паралельного доступу великої кількості користувачів. Активна спільнота розробників та комерційні редакції з розширеною підтримкою дозволяють використовувати PostgreSQL у виробничих середовищах різного рівня критичності, від невеликих стартапів до високонавантажених корпоративних систем.

Взаємодія застосунку з базою даних реалізована через ORM Entity Framework Core – об'єктно-реляційний маппер, що інтегрується з платформою .NET та підтримує роботу з PostgreSQL через провайдер Npgsql. Використання ORM дозволяє описувати модель даних мовою C# та автоматично генерувати міграції схеми бази даних з відстеженням змін у часі. Підтримка LINQ-запитів забезпечує типобезпеку на рівні компіляції та зменшує ризик помилок під час написання SQL-запитів вручну. Поєднання Entity Framework Core та PostgreSQL формує надійну основу шару доступу до даних з можливістю переходу на іншу СУБД у разі зміни вимог проєкту.

Важливою практичною перевагою PostgreSQL є підтримка типів даних JSON та JSONB, що дозволяє зберігати напівструктуровані дані без необхідності зміни структури таблиць у разі розширення функціоналу. Наприклад, у модулі обміну повідомленнями текст повідомлення може зберігатися у звичайній колонці типу text, тоді як вкладення, реакції, посилання або службові параметри можуть зберігатися у колонці metadata типу JSONB. Такий підхід дозволяє поєднати переваги реляційної моделі (зв'язки, цілісність даних) із гнучкістю документ-орієнтованих структур, не переходячи на окрему NoSQL базу даних. PostgreSQL також підтримує спеціалізовані типи індексів, які дозволяють ефективно виконувати пошук у JSON-структурах або масивах значень [19]. Це

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

може бути корисним під час фільтрації даних або пошуку за параметрами, структура яких може змінюватися у процесі розвитку системи.

Альтернативною СУБД могла бути Microsoft SQL Server, яка добре інтегрується з платформою .NET. Проте PostgreSQL не поступається їй за базовими можливостями реляційної СУБД і водночас є повністю відкритим програмним забезпеченням без ліцензійних обмежень [20]. Крім того, PostgreSQL широко використовується у Linux-середовищах та добре інтегрується з Docker-контейнерами, що спрощує розгортання та тестування системи. Образ PostgreSQL має невеликий розмір та не потребує значних апаратних ресурсів для запуску, що є практичним для середовищ розробки та CI/CD.

Клієнтська частина системи FitLead реалізується з використанням фреймворку Next.js, побудованого на базі бібліотеки React. Вибір обумовлений підтримкою сучасної моделі React Server Components, яка дозволяє виконувати частину компонентів виключно на сервері [21]. Це зменшує обсяг клієнтського JavaScript-коду та пришвидшує час до першої інтерактивності сторінки.

Для системи FitLead це має практичну цінність, оскільки значна частина інтерфейсу (списки тренувальних програм, конструктори програм, історія активності) складається з елементів перегляду даних, які не потребують складної інтерактивності. Використання RSC дозволяє виконувати запити до основного API безпосередньо на сервері, передаючи клієнту вже сформований результат. Це знижує навантаження на головний потік браузера та покращує загальну продуктивність застосунку [22].

Next.js автоматично реалізує механізм розділення коду на рівні маршрутів, що гарантує завантаження лише того обсягу JavaScript, який необхідний для поточної сторінки [23]. Така оптимізація позитивно впливає на основні вебпоказники, зокрема LCP та TBT, які є ключовими метриками Google для оцінювання якості користувацького досвіду [24].

Вбудована файлова маршрутизація спрощує організацію структури проєкту та усуває потребу у сторонніх бібліотеках для навігації. Це дозволяє

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

зосередитися на реалізації бізнес-логіки системи, зберігаючи архітектурну цілісність клієнтського застосунку.

Для підготовки системи до розгортання обрано Docker. Серверна частина, клієнтська частина та база даних можуть запускатися як окремі контейнери з власними залежностями та конфігурацією. Це зменшує залежність від налаштувань конкретного сервера, оскільки застосунок запускається в однаковому середовищі під час локальної перевірки, тестування та розміщення на сервері. Для FitLead такий підхід є практичним, оскільки система складається з кількох компонентів, які повинні працювати разом: ASP.NET Core API, Next.js застосунок і PostgreSQL.

Для автоматизації перевірки та збирання компонентів використано GitHub Actions. Цей інструмент інтегрується з репозиторієм проєкту та дозволяє запускати перевірки під час pull request, push у головну гілку або вручну. У межах workflow виконуються збірка серверної та клієнтської частин, запуск тестів і формування Docker-образів. Для зберігання зібраних образів використовується GitHub Container Registry, а їх подальший запуск на сервері виконується через Dokploy, де налаштовуються змінні середовища, маршрутизація запитів і робота контейнерів.

Висновки до першого розділу

У першому розділі було проаналізовано предметну область онлайн-супроводу у сфері фітнесу та проведено огляд існуючих програмних рішень. На основі цього було визначено ключові функціональні вимоги до системи та актуальність розробки.

Аналіз платформ Trainerize, Trainero та Trainerfu показав, що сучасні системи онлайн-тренувань та супроводу забезпечують набагато ширший набір можливостей ніж потребує персональний тренер, і часто фокусуються на більший бізнес. Водночас, більшість існуючих рішень орієнтовані на глобальні ринки. Це створює потребу у програмному продукті, адаптованому до умов локального ринку та потреб індивідуальних тренерів.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савицький Р.С.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

На основі визначених вимог було обрано архітектуру модульного моноліту, яка дозволяє забезпечити структурованість системи без надмірного ускладнення інфраструктури. Даний підхід дозволяє логічно розділити систему на окремі функціональні модулі, зберігаючи простоту розгортання та підтримки. Використання принципів чистої архітектури, DDD та CQRS дозволить ізолювати бізнес-логіку від технічних деталей реалізації, покращити тестування компонентів та спростити подальший розвиток системи.

Обґрунтування вибору технологічного стеку показало доцільність використання платформи .NET та ASP.NET Core для реалізації серверної частини системи, оскільки дані технології забезпечують стабільність, високу продуктивність та наявність розвинених інструментів для створення вебзастосунків. Для зберігання даних обрано PostgreSQL, що поєднує можливості класичної реляційної бази даних із підтримкою напівструктурованих даних у форматі JSONB, що є практичним для реалізації модулів із гнучкою структурою даних, зокрема обміну повідомленнями або збереження додаткових параметрів тренувань.

Для реалізації клієнтської частини застосунку обрано Next.js, який підтримує сучасну модель React Server Components, що дозволяє зменшити обсяг клієнтського JavaScript-коду та підвищити швидкодію інтерфейсу. Це є доцільним для системи у якій велика кількість статичних сторінки для перегляду даних поєднується з інтерактивною взаємодією.

Таким чином, результати проведеного аналізу дозволили визначити функціональні вимоги, оптимальну архітектуру та технологічний стек для реалізації вебплатформи FitLead.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ СУПРОВОДУ КЛІЄНТІВ У СФЕРІ ФІТНЕСУ «FITLEAD»

2.1. Визначення варіантів використання та проектування об'єктно-орієнтованої структури системи

На етапі проектування системи було визначено ключових акторів та сценарії використання платформи під час організації онлайн-супроводу. Дані сценарії описують основні дії, що виконуються користувачами та системою у межах платформи. Було визначено основних акторів платформи та їхню взаємодію з функціональними можливостями FitLead. Опис функціоналу поділено на логічні модулі. Усі актори та стислий зміст доступного функціоналу наведено у табл. 2.1.

Таблиця 2.1

Виявлення акторів та доступних можливостей

Актор	Доступний функціонал
Неавторизований користувач	Головна сторінка сайту для ознайомлення з платформою. Можливість входу та авторизації.
Авторизований користувач – Тренер	Керування власною бібліотекою вправ; використання платформних вправ; створення тренувань; формування тренувальних програм; призначення програм клієнтам; відкликання призначень; створення і відкликання запрошень; перегляд клієнтів, їх профілів, програм, записів виконання тренувань, прогресу та відеозвітів; чат із клієнтами; надання фідбеку на відеозвіти; перегляд і позначення сповіщень.
Авторизований користувач – Клієнт	Прийняття запрошення тренера; перегляд свого тренера; ведення профілю; внесення антропометричних показників; додавання фото прогресу; перегляд призначених програм; фіксація виконання тренувань; чат із тренером; створення відеозвітів; перегляд фідбеку тренера; робота з медіафайлами; перегляд і позначення сповіщень.

Додатково, для кращого сприйняття було побудовано діаграму варіантів використання для акторів тренер та клієнт (рис. 2.1).

Варіанти використання модуля «Доступ, користувачі та запрошення»

Основний актор	Варіант використання	Функціонал
Неавторизований користувач	Створення облікового запису	Можливість зареєструватися в системі як тренер або клієнт за допомоги email та паролю.
Неавторизований користувач	Вхід до системи	Можливість увійти до власного облікового запису.
Авторизований користувач	Завершення роботи	Можливість вийти зі свого облікового запису.
Тренер	Створення запрошення	Можливість створити запрошення для підключення нового клієнта.
Тренер	Перегляд запрошень	Можливість переглядати раніше створені запрошення.
Тренер	Відкликання запрошення	Можливість скасувати створене запрошення.
Клієнт	Прийняття запрошення	Можливість приєднатися до тренера за отриманим запрошенням.
Неавторизований користувач	Перегляд запрошення	Можливість ознайомитися з інформацією про запрошення перед входом або реєстрацією.

Модуль «Тренувальний контент» відповідає за створення та організацію тренувальних матеріалів. Він надає тренеру можливість керувати бібліотекою вправ, формувати тренування з конкретними параметрами виконання, створювати тренувальні програми, розподіляти тренування за тижнями й днями та призначати готові програми підключеним клієнтам. Клієнт у межах цього модуля переглядає призначені програми та фіксує факт виконання тренувань. Актори та варіанти використання наведено у табл. 2.3.

Варіанти використання модуля «Тренувальний контент»

Основний актор	Варіант використання	Функціонал
Тренер	Керування вправами	Можливість створювати, переглядати, редагувати та видаляти вправи.
Тренер	Використання готових вправ	Можливість додавати готові вправи до власної бібліотеки.
Тренер	Керування тренуваннями	Можливість створювати, переглядати, перейменовувати та видаляти тренування.
Тренер	Наповнення тренування	Можливість додавати вправи до тренування та задавати параметри їх виконання.
Тренер	Керування програмами	Можливість створювати тренувальні програми та розміщувати тренування за тижнями й днями.
Тренер	Призначення програми	Можливість призначити тренувальну програму конкретному клієнту.
Тренер	Скасування призначення	Можливість відкликати призначену клієнту програму.
Клієнт	Перегляд програм	Можливість переглядати призначені тренувальні програми, тренування та описи вправ.
Клієнт	Фіксація виконання	Можливість відмічати виконане або пропущене тренування та залишати коротку нотатку.

Модуль «Профіль клієнта та прогрес» призначений для збереження персональної інформації клієнта та відстеження змін його фізичного стану. Клієнт може оновлювати анкету, вказувати цілі, рівень досвіду, обмеження та тренувальні вподобання, а також вести записи антропометричних показників і додавати фото прогресу. Тренер має доступ до цих даних для аналізу стану свого клієнта. Актори та варіанти використання наведено у табл. 2.4.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Таблиця 2.4

Варіанти використання модуля «Профіль клієнта та прогрес»

Основний актор	Варіант використання	Функціонал
Клієнт	Ведення профілю	Можливість вказати ціль, рівень підготовки, зріст, обмеження та тренувальні побажання.
Клієнт	Внесення показників тіла	Можливість додавати записи про вагу, відсоток жиру та об'єми тіла.
Клієнт	Редагування показників	Можливість змінювати або видаляти раніше внесені показники.
Клієнт	Додавання фото прогресу	Можливість завантажувати фото з датою, позначкою та нотаткою.
Клієнт	Перегляд власного прогресу	Можливість переглядати збережені показники та фото.
Тренер	Перегляд профілю клієнта	Можливість ознайомитися з анкетною підключеного клієнта.
Тренер	Перегляд прогресу клієнта	Можливість переглядати показники тіла та фото прогресу клієнта.

Модуль «Клієнтська база тренера» забезпечує тренеру централізований перегляд інформації про підключених клієнтів. У межах модуля реалізовано список клієнтів, короткий огляд стану конкретного клієнта, перегляд його активних програм, записів виконання тренувань, прогресу, профілю та останніх відеозвітів. Модуль використовується як робоча область тренера для контролю супроводу клієнтів. Актори та варіанти використання наведено у табл. 2.5.

Таблиця 2.5

Варіанти використання модуля «Клієнтська база тренера»

Основний актор	Варіант використання	Функціонал
Тренер	Перегляд клієнтів	Можливість бачити список підключених клієнтів.

Тренер	Перегляд короткого огляду клієнта	Можливість бачити основну інформацію про стан роботи з клієнтом.
Тренер	Перегляд програм клієнта	Можливість переглядати програми, призначені конкретному клієнту.
Тренер	Перегляд виконання тренувань	Можливість переглядати записи про виконані або пропущені тренування клієнта.
Тренер	Перегляд прогресу	Можливість перейти до показників тіла та фото прогресу клієнта.
Тренер	Перегляд відеозвітів	Можливість переглядати останні відеозвіти клієнта.
Тренер	Перегляд профілю	Можливість переглядати анкету клієнта.

Модуль «Комунікація та відеозвіти» забезпечує взаємодію між тренером і клієнтом через чат. У системі реалізовано створення або відкриття чату між пов'язаними користувачами, обмін текстовими повідомленнями, перегляд історії повідомлень, надсилання клієнтом відеозвіту з прикріпленими медіафайлами та надання тренером текстового фідбеку. Нові повідомлення доставляються в реальному часі. Актори та варіанти використання наведено у табл. 2.6.

Таблиця 2.6

Варіанти використання модуля «Комунікація та відеозвіти»

Основний актор	Варіант використання	Функціонал
Тренер	Початок спілкування з клієнтом	Можливість відкрити розмову з підключеним клієнтом.
Клієнт	Початок спілкування з тренером	Можливість відкрити розмову зі своїм тренером.
Тренер / Клієнт	Перегляд розмов	Можливість переглядати список розмов і окрему розмову.

Тренер / Клієнт	Надсилання повідомлень	Можливість обмінюватися текстовими повідомленнями.
Тренер / Клієнт	Перегляд історії повідомлень	Можливість переглядати попередні повідомлення в розмові.
Клієнт	Робота з відеозвітами	Можливість прикріпити відеоматеріали до звіту та надіслати їх тренеру; можливість переглянути відгук тренера щодо відеозвіту.
Тренер	Робота з відеозвітами	Можливість переглянути відеозвіт, надісланий клієнтом; можливість залишити текстовий відгук до відеозвіту клієнта.

Модуль «Сповіщення» забезпечує інформування користувачів про важливі події, пов'язані з тренувальним процесом і взаємодією між тренером та клієнтом. У межах модуля користувач може переглядати отримані сповіщення, бачити кількість непрочитаних сповіщень, позначати окремі або всі сповіщення як прочитані, а також керувати отриманням сповіщень на своєму пристрої. Сповіщення формуються для подій призначення тренувальної програми, надсилання клієнтом відеозвіту та надання тренером відповіді на відеозвіт. Актори та варіанти використання наведено у табл. 2.7.

Таблиця 2.7

Варіанти використання модуля «Сповіщення»

Основний актор	Варіант використання	Функціонал
Авторизований користувач	Перегляд сповіщень	Можливість бачити отримані сповіщення.
Авторизований користувач	Перегляд кількості непрочитаних сповіщень	Можливість бачити, скільки сповіщень ще не переглянуто.
Авторизований користувач	Позначення сповіщення прочитаним	Можливість позначити окреме сповіщення як прочитане, перейти до сторінки яка відноситься до повідомлення.

Базовим класом для доменних об'єктів є Entity який визначає наявність ідентифікатора сутності. Клас AggregateRoot наслідується від Entity і використовується для позначення коренів агрегатів. У межах діаграми агрегатами є Exercise, Workout, WorkoutLog, TrainingProgram та AssignedTrainingProgram. Вони виступають основними точками входу для виконання бізнес-операцій та відповідають за підтримку цілісності відповідних частин доменної моделі.

Сутність Exercise моделює вправу та містить описову інформацію про неї, зокрема назву, опис, групу м'язів, обладнання, джерело походження та посилання на медіа. Workout представляє шаблон тренування, який належить тренеру та містить набір вправ із параметрами виконання. Ці параметри винесені в сутність WorkoutExercise, що дозволяє зберігати кількість підходів, повторень, час відпочинку, навантаження, порядок виконання та примітку тренера.

TrainingProgram виступає агрегатом, що інкапсулює логіку формування тренувальної програми. Програма не містить тренування безпосередньо, а пов'язує їх через сутність TrainingProgramWorkout, у якій зберігаються номер тижня, день, порядок у межах дня та ідентифікатор тренування. Такий підхід дозволяє повторно використовувати шаблони тренувань у різних програмах без порушення меж агрегатів.

AssignedTrainingProgram описує факт призначення тренувальної програми конкретному клієнту тренером. Сутність містить дані про клієнта, тренера, програму, дату призначення, дату завершення, статус та можливе відкликання призначення. WorkoutLog використовується для фіксації результату виконання тренування клієнтом. У ньому зберігаються статус виконання, дата, оцінка складності, нотатка клієнта та зв'язок із призначеною програмою і конкретним тренуванням у програмі.

Окремо на діаграмі показано сутності TrainingProgramWorkout та WorkoutExercise, які наслідують Entity, але не є коренями агрегатів. Їхні методи мають рівень доступу internal, що означає використання лише в межах доменного шару. Це дозволяє агрегатам TrainingProgram і Workout керувати зміною

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

пов’язаних сутностей та не відкривати ці операції напряду для прикладного шару.

Такий підхід забезпечує чітке розмежування відповідальності між агрегатами, збереження інваріантів доменної моделі та можливість подальшого розширення функціональності без порушення основної структури модуля.

Для організації прикладної логіки використано підхід CQRS, який передбачає логічне розділення операцій зміни стану та операцій читання даних. На рисунку 2.3 показано взаємодію прикладного рівня з доменною моделлю. Команди та запити описуються окремими класами і обробляються відповідними обробниками, які викликаються за допомоги бібліотеки MediatR, що реалізує шаблон медіатор у межах одного процесу і зменшує зв’язування між компонентами системи. Командні обробники відповідають за зміну стану доменних об’єктів і взаємодіють з ними через інтерфейси репозиторіїв, тоді як запитні обробники зосереджені на читанні даних та формуванні DTO, не змінюючи стан доменних об’єктів.

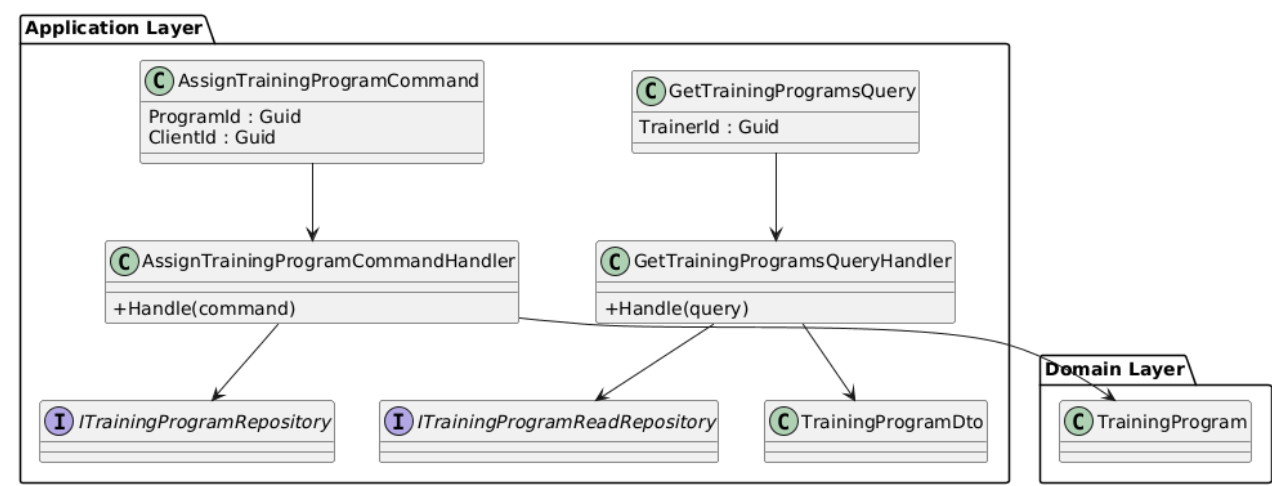


Рис. 2.3. Діаграма реалізації шаблону CQRS

На рисунку 2.4 наведено діаграму інфраструктурного рівня, яка демонструє реалізацію репозиторіїв на базі Entity Framework Core. Інфраструктурний рівень містить конкретні реалізації інтерфейсів репозиторіїв, визначених у прикладному рівні як абстракції доступу до даних, що дозволяє зберегти незалежність доменної моделі від деталей збереження інформації. Таким чином забезпечується дотримання правила залежностей Clean

Architecture, згідно з яким вихідні залежності спрямовані від зовнішніх шарів до внутрішніх, а внутрішні шари не залежать від зовнішніх.

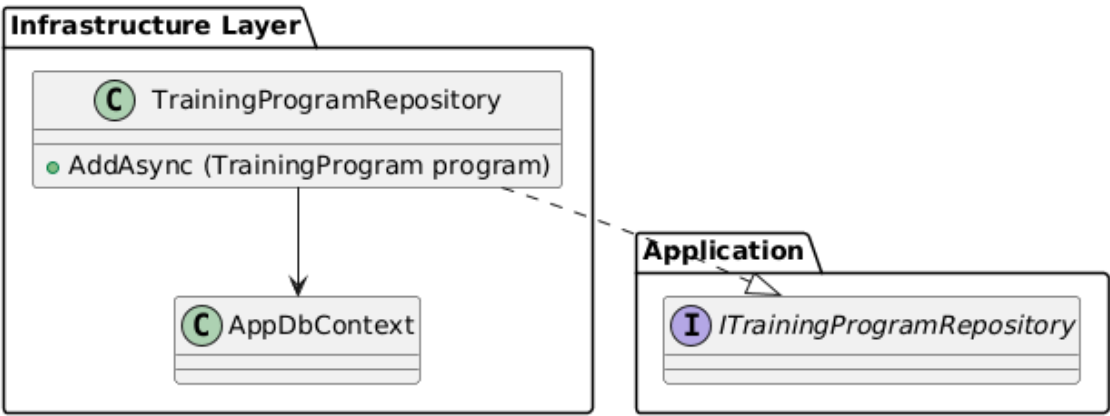


Рис. 2.4. Діаграма інфраструктурного рівня

У сукупності наведені діаграми демонструють цілісну об’єктно-орієнтовану модель системи, де доменна логіка ізольована від технічних аспектів, а архітектурні рішення сприяють масштабованості, легшому тестуванню та подальшому розвитку програмного продукту.

Для забезпечення цілісності модульного моноліту взаємодія між різними функціональними блоками та реалізована через механізм публічних контрактів. Кожен модуль визначає набір публічних інтерфейсів та DTO, які доступні для виклику іншими частинами системи. На рисунку 2.5 наведено діаграму взаємодії модулів.

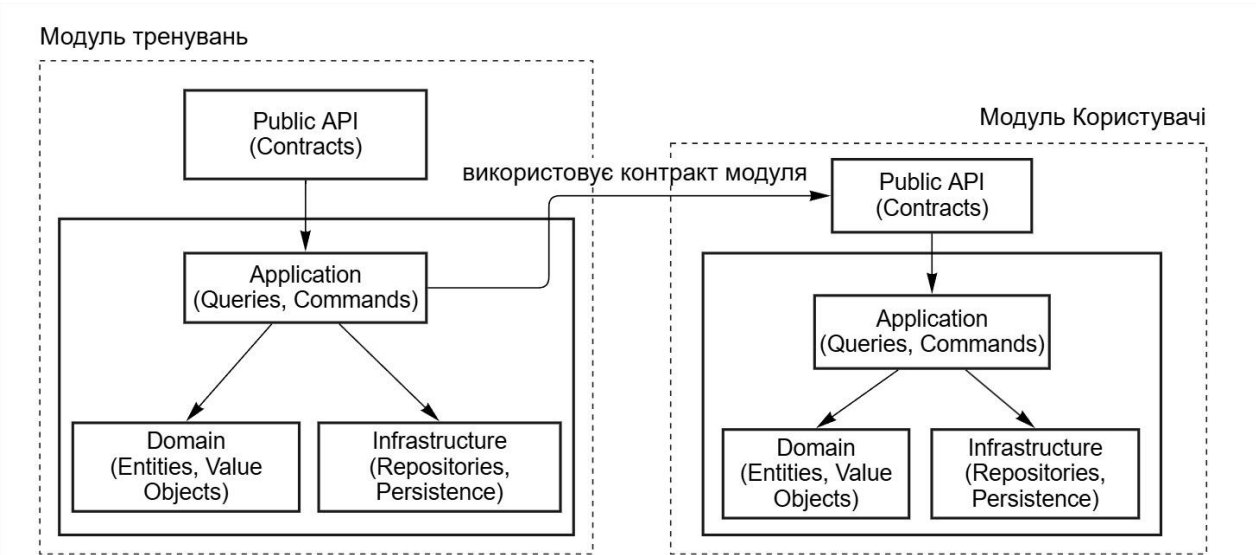


Рис. 2.5. Діаграма взаємодії модулів через Public API

Прямий доступ до внутрішніх сутностей іншого модуля заборонено на рівні архітектурних правил. Хоча модулі взаємодіють у межах одного процесу, такий підхід дозволяє логічно ізолювати внутрішні деталі модулів. Використання публічних контрактів гарантує, що зміни у внутрішній логіці або структурі даних одного модуля не призведуть до каскадних змін у інших.

2.2. Розробка бази даних системи

Для забезпечення зберігання, обробки та цілісності даних програмної платформи FitLead було спроектовано реляційну базу даних (рис. 2.6). Відповідно до обраної архітектури модульного моноліту, у системі використовується спільна база даних для всіх модулів. Це дозволяє забезпечити транзакційну цілісність та спростити інфраструктурну підтримку на початкових етапах розвитку продукту.

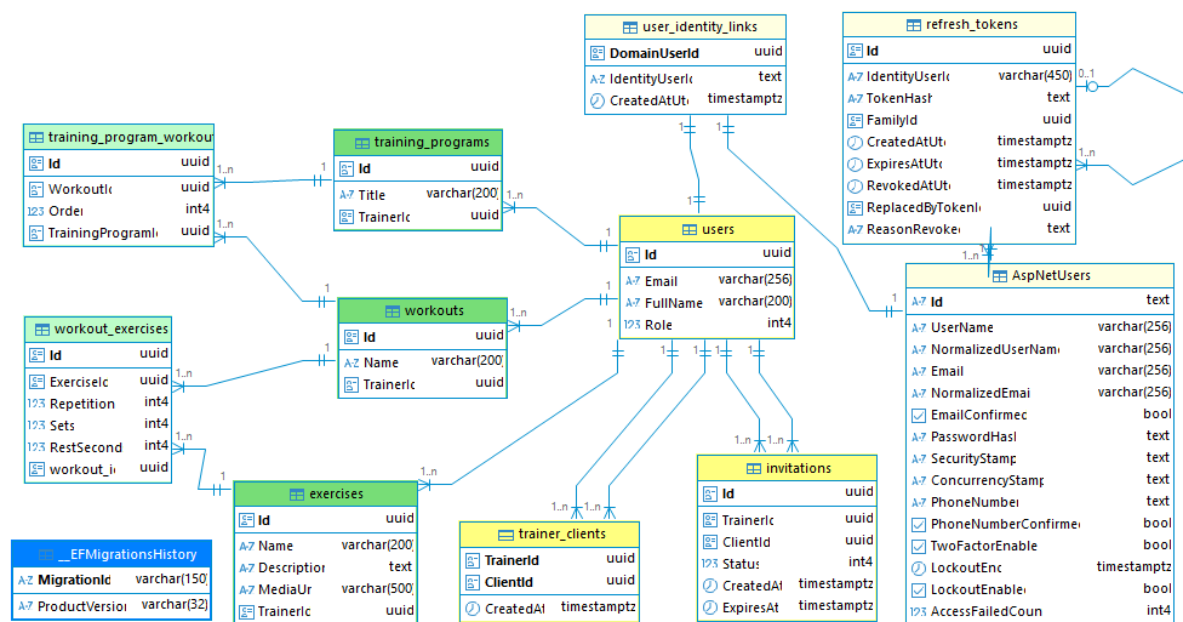


Рис. 2.6. Структура основних таблиць бази даних системи

Зв'язки між таблицями побудовані так, щоб відображати основні правила предметної області. Наприклад, тренувальна програма належить конкретному тренеру, призначення програми пов'язує тренера, клієнта і саму програму, а запис виконання тренування прив'язується до конкретного призначення. Для таких залежностей у схемі використовуються зовнішні ключі, які обмежують створення некоректних записів і підтримують узгодженість даних на рівні бази.

Проектування схеми даних базувалося на принципі, де доменні сутності не знають про деталі свого збереження. Перенесення об'єктно-орієнтованої моделі на реляційну структуру реалізовано за допомогою ORM Entity Framework Core, що дозволяє автоматизувати створення схеми через механізм міграцій.

Для ідентифікації записів у всіх таблицях використано тип даних UUID. Це забезпечує унікальність ідентифікаторів між різними модулями та спрощує потенційну міграцію даних у розподілену систему (мікросервіси) у майбутньому, оскільки виключає конфлікти під час злиття або розділення баз даних.

У таблицях також передбачено службові поля, які потрібні для контролю стану об'єктів і подальшої обробки даних. Для запрошень, призначень, відеозвітів, медіафайлів, сповіщень і Outbox-повідомлень використовуються статуси, що відображають поточний етап їх обробки. Дати створення, оновлення, прийняття, відкликання або обробки зберігаються у форматі UTC, щоб уникнути залежності від локального часу користувача або сервера.

Система безпеки та автентифікації інтегрована безпосередньо в схему БД через використання ASP.NET Core Identity. Це дозволяє централізовано керувати обліковими записами та зв'язками між системними користувачами та бізнес-профілями (тренерами/клієнтами).

У базі даних FitLead передбачено таблиці для збереження користувачів, зв'язків між тренерами і клієнтами, вправ, тренувань, тренувальних програм, чатів, відеозвітів, медіафайлів, сповіщень та службових подій. В основному тексті розглянуто таблиці, які відображають ключові частини предметної області та найважливіші сценарії роботи системи.

Однією з основних таблиць є `trainer_clients`, яка зберігає зв'язок між тренером і клієнтом (табл. 2.8). Саме цей зв'язок визначає, з якими клієнтами може працювати тренер, кому він може призначати тренувальні програми, відкривати чат і переглядати дані прогресу. Наявність окремої таблиці для такого зв'язку дозволяє не обмежуватися лише ролями користувачів, а фіксувати конкретні робочі відносини між учасниками платформи.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савінький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Опис таблиці trainer_clients

Поле	Тип	Призначення
TrainerId	uuid	Ідентифікатор тренера.
ClientId	uuid	Ідентифікатор клієнта.
CreatedAt	timestamp with time zone	Дата і час встановлення зв'язку.

Тренувальні програми зберігаються в таблиці training_programs (табл. 2.9). Вона містить основні дані програми, зокрема ідентифікатор тренера-власника, назву, кількість тижнів і кількість тренувальних днів на тиждень. Деталізація структури програми виконується через зв'язувальні (junction) таблиці, які визначають, які тренування входять до програми, на який тиждень і день вони призначені та в якому порядку мають виконуватися. Окремо в базі даних зберігаються призначення програм клієнтам і записи про виконання тренувань.

Таблиця 2.9

Опис таблиці training_programs

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор програми.
TrainerId	uuid	Ідентифікатор тренера-власника.
Title	varchar(200)	Назва тренувальної програми.
WeeksCount	integer	Кількість тижнів у програмі.
DaysPerWeek	integer	Кількість тренувальних днів на тиждень.

Для роботи з файлами використовується таблиця media_assets (табл. 2.10). Вона не зберігає сам вміст файлів, а містить службову інформацію про них: власника, зовнішнє сховище, ідентифікатор файлу у сховищі, адресу доступу, назву, тип вмісту, розмір, вид матеріалу та стан реєстрації. Такий підхід дозволяє використовувати один механізм для різних сценаріїв, зокрема медіа вправ, фото прогресу та матеріалів відеозвітів, не дублюючи логіку роботи з файлами в різних модулях системи.

Опис таблиці media_assets

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор файлу.
OwnerUserId	uuid	Ідентифікатор власника файлу.
StorageProvider	integer	Сховище, у якому розміщено файл.
StorageObjectId	varchar(500)	Ідентифікатор файлу у зовнішньому сховищі.
DeliveryUrl	varchar(2048)	Адреса для отримання файлу.
FileName	varchar(500), NULL	Початкова назва файлу.
ContentType	varchar(200)	Тип вмісту файлу.
SizeBytes	bigint	Розмір файлу в байтах.
Kind	integer	Вид матеріалу: зображення, відео або звук.
DurationSeconds	integer, NULL	Тривалість аудіо або відео в секундах.
Status	integer	Стан реєстрації файлу.
CreatedAtUtc	timestamp with time zone	Дата і час реєстрації файлу.

Повний перелік таблиць бази даних наведено в додатку А.

2.3. Проектування алгоритмів системи

Опис алгоритмів дозволяє візуалізувати ключові алгоритми роботи системи, які мають найбільше значення для розкриття предметної області та безпечного функціонування додатку. До таких процесів належать авторизація та захист запитів, оновлення і відкликання ключів користувацького сеансу, обробка сповіщень через таблицю вихідних подій, а також створення й опрацювання відеозвітів клієнтів.

На рисунку 2.7 наведено алгоритм входу користувача до системи. У цій схемі браузер не звертається безпосередньо до серверної частини ASP.NET API, а використовує серверний прошарок Next.js BFF. Такий підхід дозволяє централізувати обробку запитів від клієнтської частини та не відкривати внутрішню адресу API для браузера.

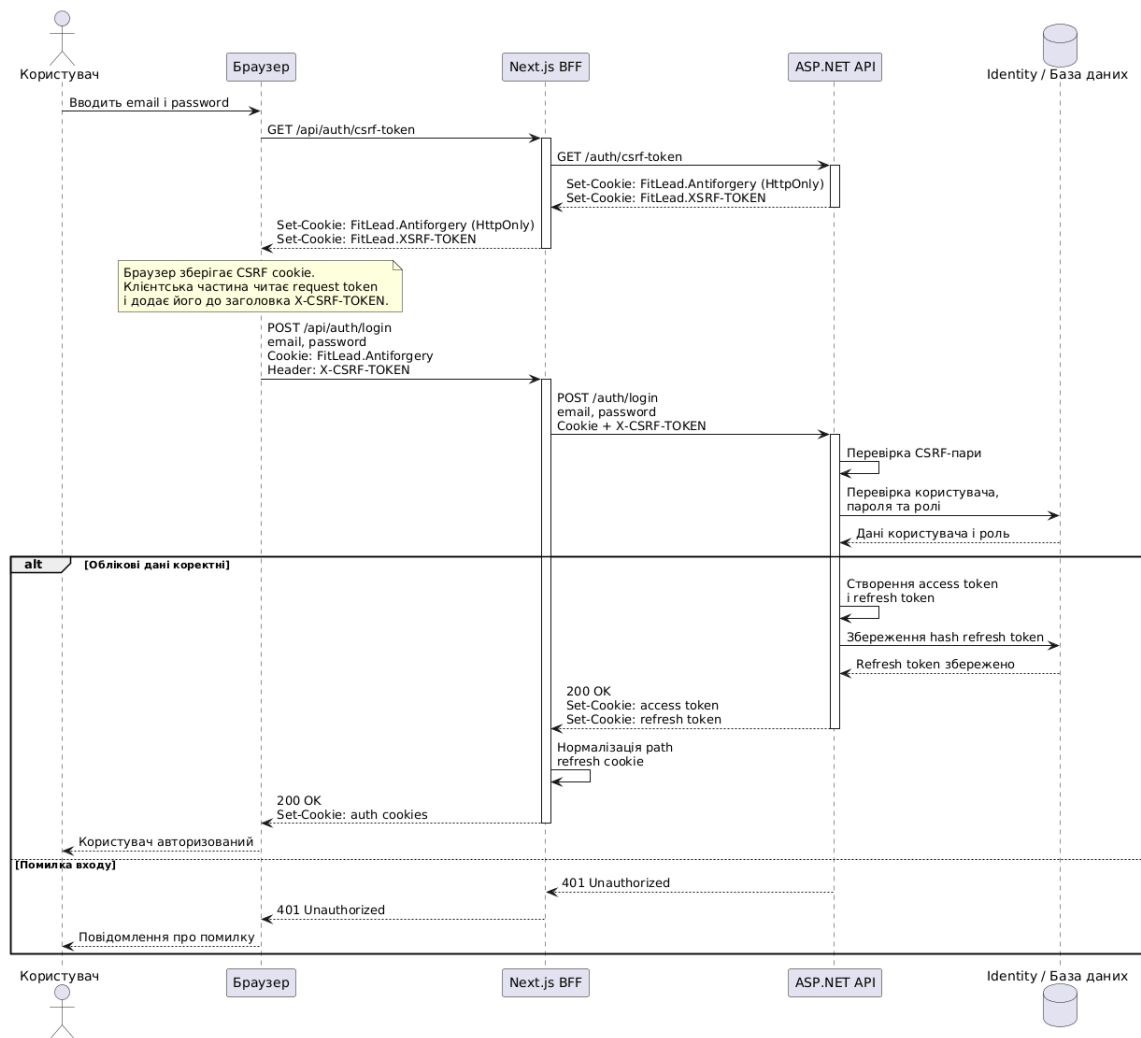


Рис. 2.7. Алгоритм входу користувача до системи

Перед виконанням входу браузер отримує CSRF-токен через маршрут Next.js BFF, який пересилає запит до ASP.NET API. У відповідь сервер створює два файли cookie: службовий FitLead.Antiforgery, недоступний для JavaScript, і FitLead.XSRF-TOKEN, значення якого клієнтська частина використовує для формування заголовка X-CSRF-TOKEN. Після цього форма входу надсилається разом із cookie та CSRF-заголовком, а Next.js BFF передає ці дані до ASP.NET API.

Серверна частина перевіряє CSRF-пару, облікові дані користувача, його роль і стан облікового запису. Якщо перевірки виконано успішно, система створює токен доступу та токен оновлення, причому токен оновлення зберігається в базі даних у вигляді хешу. Після цього API повертає захищені cookie filelead.access_token і передає ці дані до ASP.NET API.

CSRF-захист реалізовано через те, що система використовує файли cookie. Браузер автоматично додає cookie до запитів на відповідний домен, тому шкідлива сторінка теоретично могла б спробувати надіслати запит від імені вже авторизованого користувача. Щоб запобігти цьому, сервер вимагає не лише наявність cookie, а й окремий заголовок X-CSRF-TOKEN, значення якого клієнтська частина отримує з FitLead.XSRF-TOKEN. Сторонній сайт не може коректно сформулювати такий запит без доступу до токена, тому небезпечні HTTP-запити на зміну стану системи проходять додаткову перевірку.

Обраний механізм авторизації побудований так, щоб не зберігати токени у відкритому сховищі браузера та зменшити ризики їх використання стороннім кодом. Токен доступу має короткий термін дії, а токен оновлення може бути відкликаний або замінений під час оновлення сеансу. Запити з клієнтської частини проходять через Next.js BFF, тому внутрішня адреса ASP.NET API не відкривається браузеру, а передавання cookie, CSRF-токена та службових заголовків виконується централізовано. Для запитів, які змінюють стан системи, додатково перевіряється CSRF-токен.

Такий підхід складніший за схему, у якій токен просто передається в заголовок Authorization. У FitLead потрібно обробляти кілька cookie, оновлення сеансу, CSRF-захист і маршрутизацію запитів через BFF. Водночас ця реалізація не позиціонується як повна підтримка OAuth 2.0 або OpenID Connect, а використовується як прикладний механізм авторизації, достатній для поточних сценаріїв платформи.

На рисунку 2.8 наведено алгоритм виконання захищеного запиту після входу користувача до системи. Такий запит використовується для операцій, що потребують авторизації та можуть змінювати стан системи, наприклад створення тренувальної програми, надсилання повідомлення, призначення програми клієнту або оновлення даних профілю.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

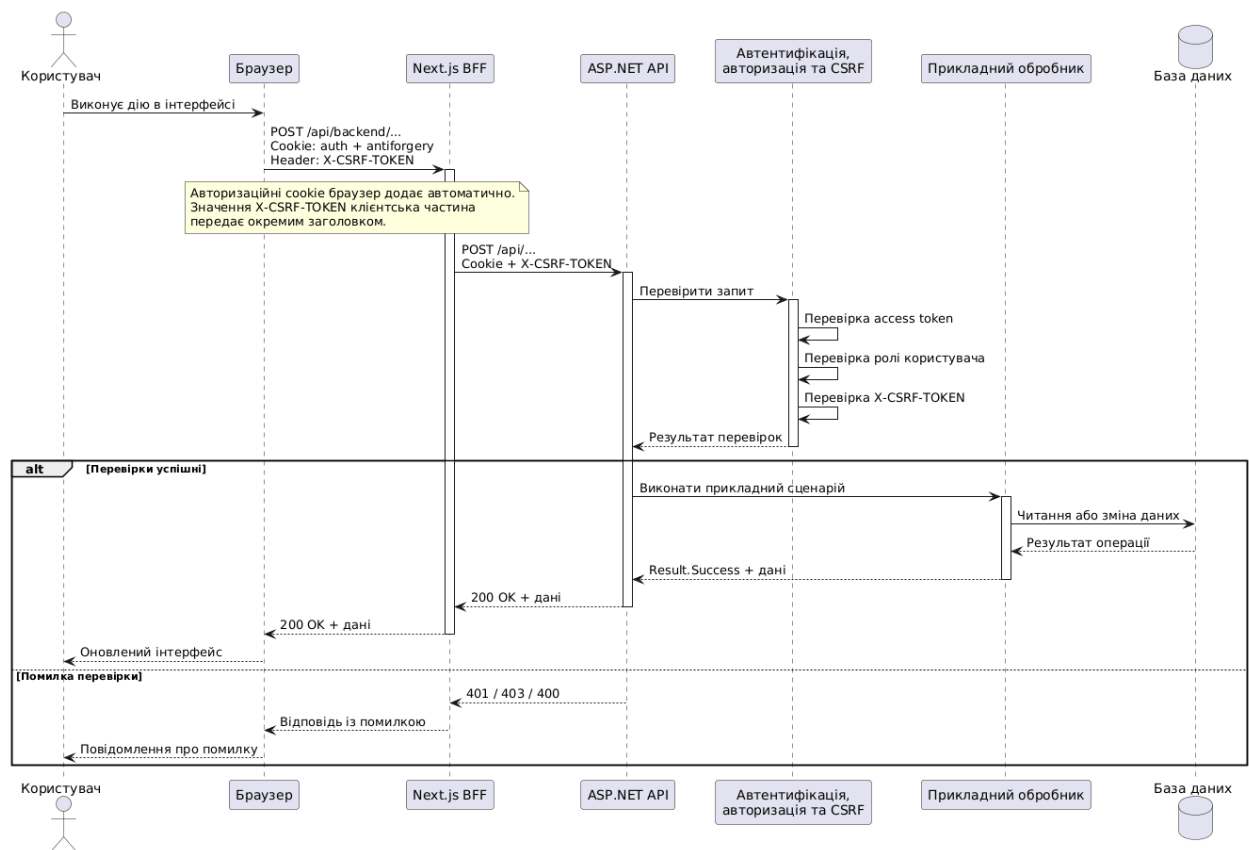


Рис. 2.8. Діаграма виконання захищеного запиту

Після дії користувача браузер надсилає запит до маршруту Next.js BFF. Оскільки дані для авторизації зберігаються у cookie, браузер автоматично додає їх до запиту на домен застосунку. Клієнтська частина додатково додає заголовок X-CSRF-TOKEN, значення якого береться з cookie FitLead.XSRF-TOKEN. Якщо всі перевірки успішні, запит передається до прикладного обробника, який виконує відповідний сценарій системи.

Такий алгоритм дозволяє не покладатися лише на факт наявності cookie для авторизації. Кожен захищений запит додатково проходить перевірку ролі, доступу до ресурсу та CSRF-захисту, що знижує ризик виконання небажаних дій від імені авторизованого користувача.

Окремим важливим процесом системи є оновлення користувацького сеансу за допомогою токена оновлення. Токен доступу має обмежений термін дії, тому система повинна мати механізм безпечного продовження сеансу без повторного введення пароля. У FitLead цей процес реалізовано через ротацію токенів оновлення. Під час кожного оновлення старий токен відкликається, а замість нього створюється новий токен у тій самій родині. Такий підхід дозволяє

підтримувати безперервну роботу користувача та водночас виявляти повторне або одночасне використання токена.

На рисунку 2.9 показано початковий етап оновлення користувацького сеансу. Перед відкриттям захищеної сторінки Next.js BFF перевіряє стан токена доступу. Оновлення запускається лише тоді, коли токен відсутній, пошкоджений або скоро завершується. Якщо оновлення не потрібне, користувач переходить до захищеної сторінки без додаткових дій.

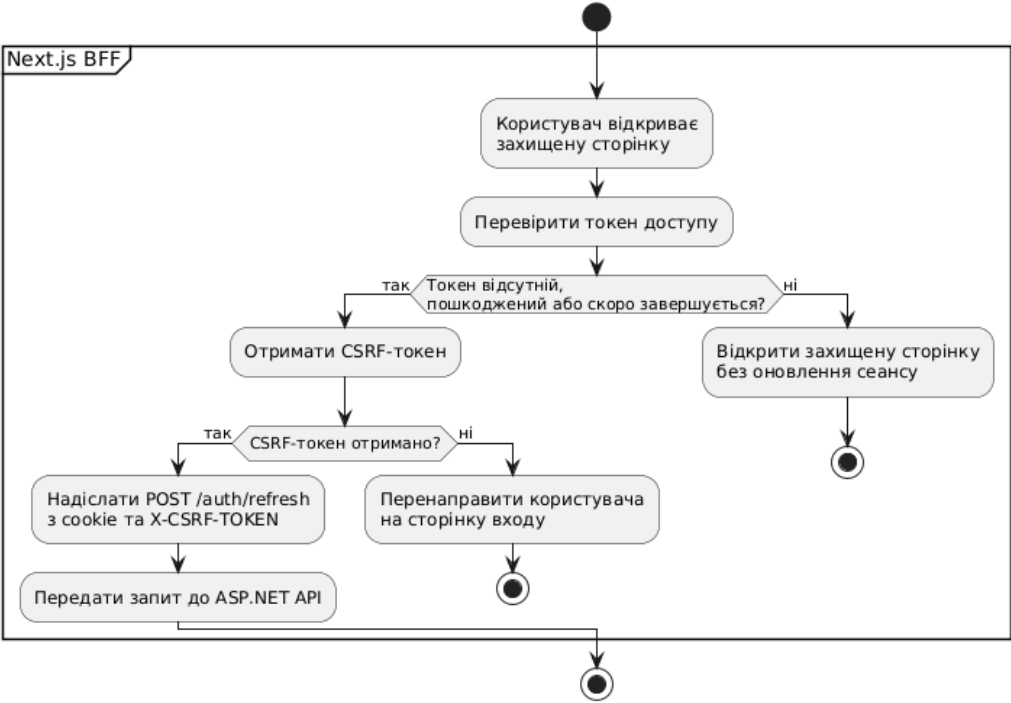


Рис. 2.9. Запуск оновлення сеансу на стороні Next.js BFF

Оскільки оновлення сеансу є операцією, що змінює стан системи, перед її виконанням Next.js BFF отримує CSRF-токен. Після цього до ASP.NET API надсилається запит оновлення з поточними cookie та заголовком X-CSRF-TOKEN. Якщо CSRF-токен отримати не вдалося, користувач перенаправляється на сторінку входу.

На рисунку 2.10 наведено обробку refresh token на стороні ASP.NET API. Сервер отримує токен оновлення з cookie, обчислює його хеш і шукає відповідний запис у базі даних. Відкрите значення токена в базі не зберігається, тому перевірка виконується саме за хешем. Подальша робота з токеном відбувається в межах транзакції, а запис блокується для запобігання одночасному використанню одного й того самого токена.

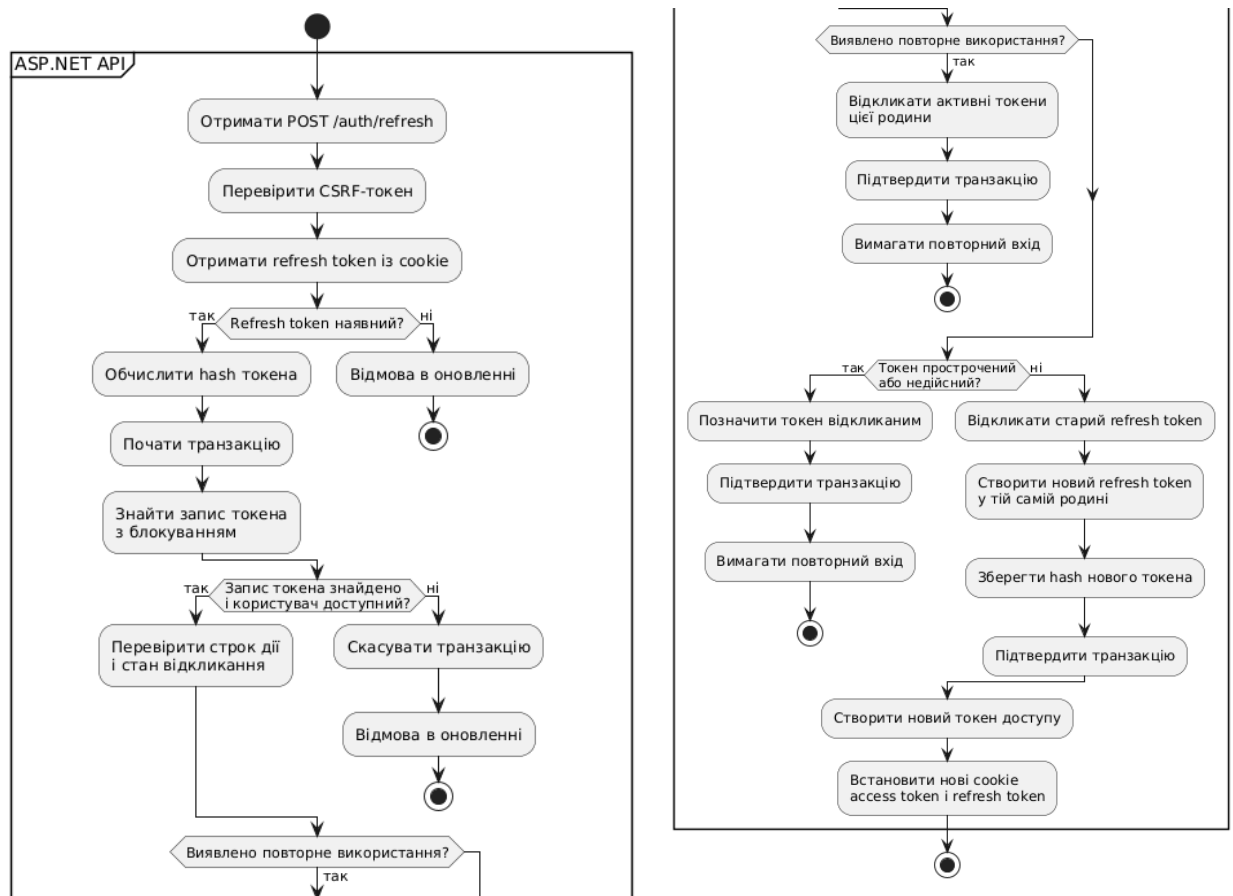


Рис. 2.10. Обробка refresh token на стороні ASP.NET API

Якщо токен коректний, не прострочений і не був відкликаний раніше, система відкликає старий refresh token та створює новий токен у тій самій родині. Після цього створюється новий токен доступу, а браузеру передаються нові захищені cookie. Якщо токен прострочений або недійсний, оновлення відхиляється. Якщо виявлено повторне або одночасне використання токена, система відкликає активні токени відповідної родини та вимагає від користувача повторного входу.

Таким чином, оновлення користувацького сеансу відбувається без повторного введення пароля, але з додатковими перевітками безпеки. Використання Next.js BFF у цьому процесі дозволяє централізовано перевіряти стан токена доступу, виконувати службові запити до API, передавати cookie та CSRF-заголовки, а також повертати користувача на потрібну захищену сторінку після успішного оновлення. У цій частині системи BFF виступає проміжним серверним шаром між браузером і API, що спрощує роботу із серверними

компонентами Next.js та зменшує кількість логіки, яку потрібно виконувати безпосередньо в клієнтській частині.

Для надійної обробки сповіщень у системі використано підхід Outbox [25]. Його суть полягає в тому, що прикладна операція не надсилає сповіщення безпосередньо під час виконання основного сценарію, а записує службове повідомлення до таблиці outbox_messages. Основні зміни в базі даних і Outbox-повідомлення зберігаються разом, тому система не втрачає подію навіть у разі помилки під час подальшої доставки. У FitLead цей процес складається з двох етапів.

На рисунку 2.11 наведено перший етап обробки сповіщення через Outbox.

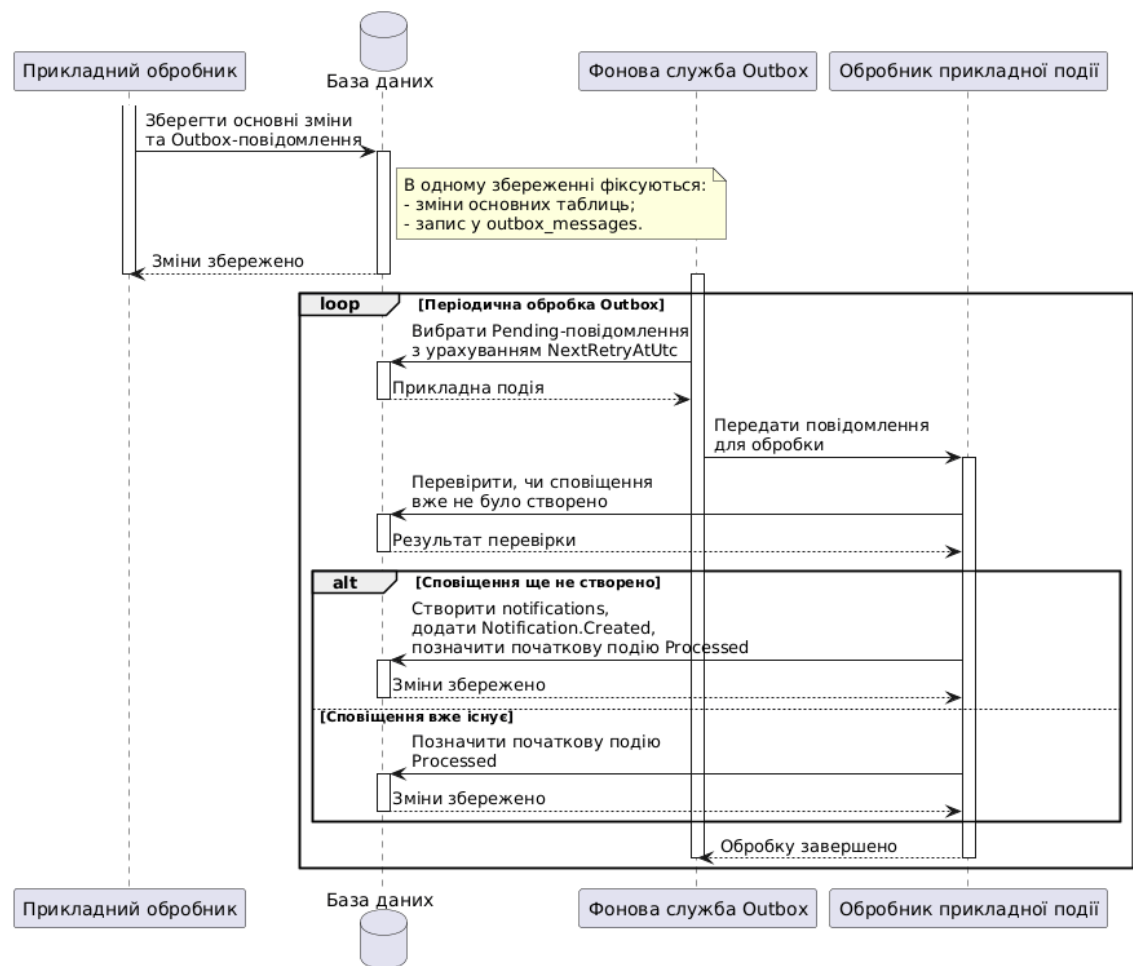


Рис. 2.11. Створення сповіщення на основі прикладної події

Під час виконання прикладної операції, наприклад призначення тренувальної програми або надсилання відеозвіту, обробник змінює основні дані системи та додає повідомлення до таблиці outbox_messages. Ці зміни зберігаються разом, тому прикладна подія не може бути втрачена окремо від

основної операції. Після цього фонові служба Outbox періодично вибирає повідомлення зі станом Pending, враховуючи час наступної спроби обробки. Вибране повідомлення передається обробнику прикладної події. Обробник перевіряє, чи відповідне сповіщення вже не було створено раніше, що захищає систему від дублювання під час повторної обробки.

Якщо сповіщення ще не існує, створюється запис у табл. notifications, додається нове Outbox-повідомлення типу Notification.Created, а початкова подія позначається як оброблена.

На діаграмі активація фонові служби показує не весь час її існування, а один цикл обробки повідомлення. У реальній системі ця служба працює постійно у фоновому режимі та періодично перевіряє наявність нових або повторно доступних Outbox-повідомлень.

На рисунку 2.12 наведено другий етап обробки сповіщення через Outbox. Якщо на попередньому етапі прикладна подія була перетворена на запис у табл. notifications, то на цьому етапі окрема подія Notification.Created використовується для фактичної доставки сповіщення користувачу. Фонові служба Outbox вибирає з таблиці outbox_messages повідомлення типу Notification.Created і передає його відповідному обробнику.

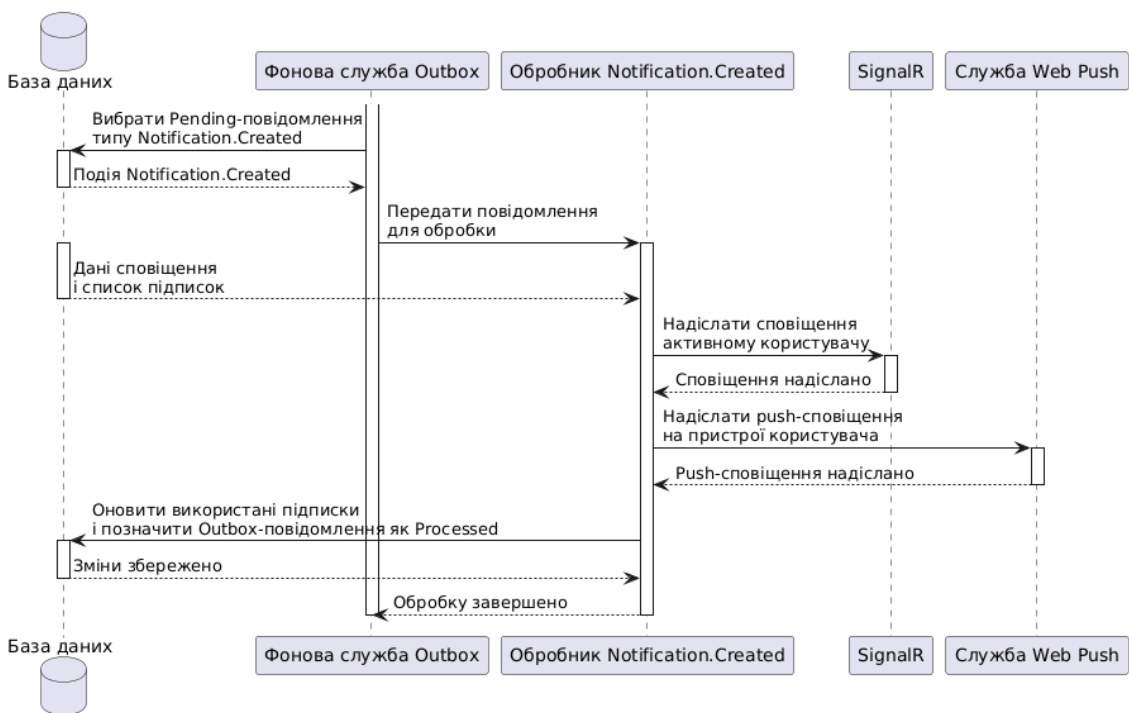


Рис. 2.12. Доставка сповіщення через Outbox після створення події

Обробник завантажує з бази даних створене сповіщення, дані отримувача та активні Web Push-підписки. Після цього сповіщення надсилається через SignalR, що дозволяє доставити його користувачу в реальному часі, якщо він перебуває в системі.

Додатково система використовує Web Push для доставки сповіщень на пристрої користувача. Це дозволяє повідомити користувача навіть тоді, коли він не перебуває безпосередньо на сторінці застосунку. Після успішного виконання обробник оновлює використані підписки та позначає Outbox-повідомлення як оброблене.

На діаграмі показано успішний сценарій доставки. У разі загальної помилки обробки Outbox-повідомлення не видаляється, а залишається доступним для повторної спроби. Якщо помилка виникає лише під час Web Push-доставки на окремий пристрій, вона не блокує обробку всієї події. Це потрібно для того, щоб одна недійсна підписка не заважала доставці сповіщення через інші канали або на інші пристрої.

Важливим процесом взаємодії між клієнтом і тренером у системі є створення та опрацювання відеозвіту. Цей сценарій є одним із ключових для супроводу клієнта, оскільки дозволяє тренеру не лише видавати тренувальні завдання, а й оцінювати якість їх виконання за надісланими матеріалами. Процес поєднує роботу з чатом, медіафайлами, перевіркою доступу, зміною стану звіту та надсиланням сповіщень. Для кращого пояснення алгоритм поділено на створення звіту клієнтом, його опрацювання тренером та окрему схему станів відеозвіту.

На рисунку 2.13 показано алгоритм створення відеозвіту клієнтом. Процес починається з відкриття або створення чату з тренером, після чого клієнт обирає зображення або відео для звіту. Завантаження файлів виконується у зовнішнє сховище, а FitLead реєструє відомості про ці файли у власній базі даних. Під час реєстрації система перевіряє наявність файлу у сховищі, визначає його тип і пов'язує медіаматеріал з поточним користувачем.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

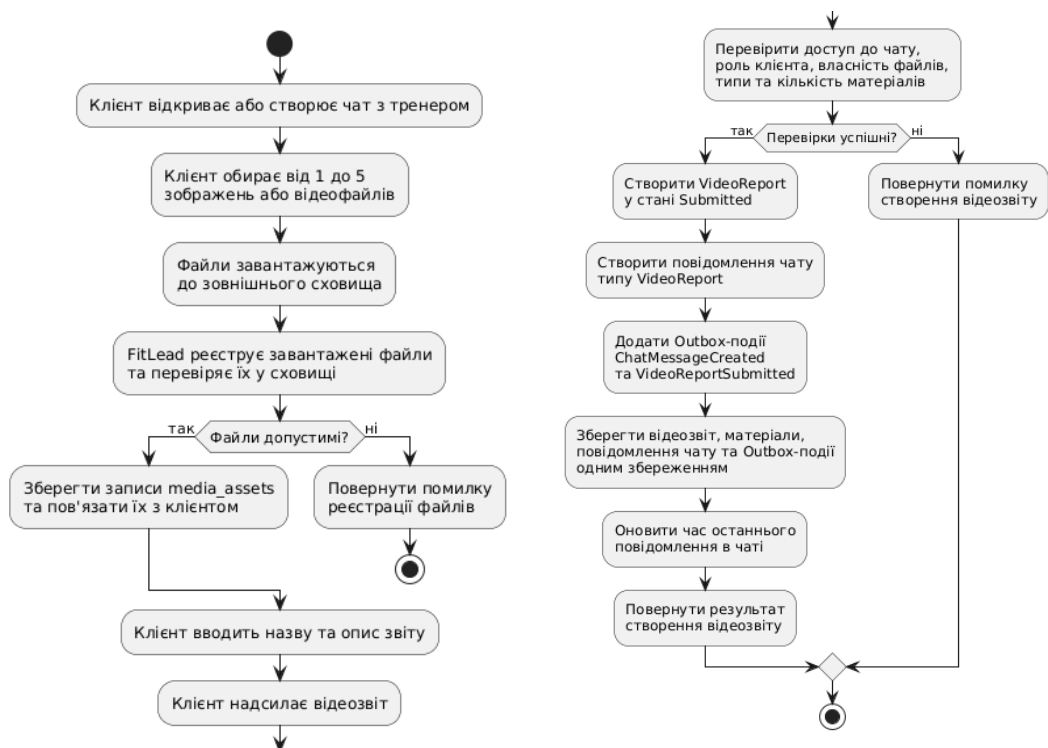


Рис. 2.13. Діаграма створення відеозвіту клієнтом

Після завантаження матеріалів клієнт вводить назву, опис і надсилає відеозвіт. Серверна частина перевіряє доступ користувача до чату, його роль, належність файлів поточному клієнту, допустимість типів файлів і кількість прикріплених матеріалів. До відеозвіту можуть додаватися лише зображення або відео, а сам звіт повинен містити від одного до п'яти файлів.

Якщо перевірки виконано успішно, система створює VideoReport у стані Submitted, додає прикріплені матеріали, формує повідомлення чату типу VideoReport та записує дві Outbox-події.

Подія ChatMessageCreated використовується для доставки нового повідомлення учасникам чату, а подія VideoReportSubmitted запускає створення сповіщення для тренера. Усі ці зміни зберігаються разом, що забезпечує цілісність процесу створення відеозвіту.

На рисунку 2.14 наведено алгоритм опрацювання відеозвіту тренером. Процес починається з відкриття чату з клієнтом та вибору конкретного відеозвіту для перегляду. Перед завантаженням матеріалів система перевіряє, чи має тренер доступ до відповідного чату та чи належить звіт цьому чату. Якщо перевірка не проходить, запит завершується помилкою доступу.



Рис. 2.14. Алгоритм опрацювання відеозвіту тренером

Простий перегляд матеріалів не змінює стан відеозвіту. Це важливо, оскільки факт відкриття звіту ще не означає, що тренер надав клієнту відповідь. Стан звіту змінюється лише тоді, коли тренер надсилає текстовий відгук. Перед збереженням система перевіряє роль тренера, належність звіту до чату, поточний стан Submitted, а також коректність тексту відгуку.

Після успішної перевірки текст відгуку зберігається безпосередньо у відеозвіті, встановлюється час перегляду, а стан змінюється на Reviewed. Окреме повідомлення чату для відгуку не створюється. Разом зі змінами додається Outbox-подія VideoReportReviewed, яка надалі використовується для створення та доставки сповіщення клієнту.

Отже, було розглянуто основні алгоритми роботи системи, які мають найбільше значення для її коректного та безпечного функціонування. Наведені діаграми демонструють ті процеси, які містять важливу прикладну логіку, перевірки доступу, зміну станів або асинхронну обробку подій.

Окрему увагу приділено алгоритмам авторизації та захисту запитів. Вони забезпечують захист операцій зміни даних за допомогою CSRF-токена та можливість виявлення повторного використання токена. Це дозволяє зменшити ризики несанкціонованого доступу, покращуючи безпеку додатку.

2.4. Реалізація вебплатформи супроводу клієнтів у сфері фітнесу

Усі модулі знаходяться в межах одного застосунку, однак код організований таким чином, щоб бізнес-логіка, прикладні сценарії, інфраструктурні залежності та API-рівень були розділені між окремими проєктами.

Проєкт FitLead.Domain містить доменну модель системи, тобто основні бізнес-сутності, правила та логіку предметної області. Саме цей шар описує ключові поняття системи, наприклад користувачів, тренування, запрошення та інші бізнес-об'єкти. Доменний шар не залежить від бази даних, вебфреймворку чи зовнішніх сервісів, що дозволяє зберігати бізнес-логіку незалежною від технічних деталей реалізації.

Проєкт FitLead.Application відповідає за прикладну логіку системи. У цьому шарі розміщуються сценарії використання, обробка команд і запитів, валідація вхідних даних, взаємодія з абстракціями репозиторіїв, сервісів і зовнішніх залежностей. Application-шар координує виконання бізнес-операцій, але не містить деталей доступу до бази даних чи конкретних механізмів інфраструктури. Завдяки цьому прикладні сценарії залишаються незалежними від способу їхнього виклику через HTTP API, фонову задачу або інший інтерфейс.

Проєкт FitLead.Infrastructure містить технічну реалізацію залежностей, необхідних для роботи системи. До цього шару належать механізми доступу до даних, міграції бази даних, реалізації сервісів ідентифікації, робота з часом, фонові задачі та інші інфраструктурні компоненти. На відміну від доменного та прикладного шарів, інфраструктурний шар залежить від конкретних технологій і надає реальні реалізації абстракцій, оголошених на рівні Application.

Проєкт FitLead.Api є вхідною точкою системи. Він відповідає за обробку HTTP-запитів, маршрутизацію, контролери або endpoints, авторизацію, перетворення вхідних даних у запити до Application-шару та повернення відповідей клієнту. API-шар не повинен містити складної бізнес-логіки, оскільки

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

його основна роль полягає в адаптації зовнішнього інтерфейсу до внутрішніх сценаріїв використання системи.

Важливою особливістю архітектури є те, що всередині шарів код групується переважно не за технічними категоріями, а за бізнес-логікою. Наприклад, у доменному та прикладному шарах виділяються окремі функціональні напрями на зразок Users, Trainings, Invitations, Identity. Це означає, що файли, пов'язані з певною бізнес-областю, знаходяться поруч, навіть якщо вони мають різні технічні ролі. Групування за бізнес-функціями полегшує навігацію по коду, оскільки розробник бачить повний контекст певного модуля в одному місці.

Загальна залежність між шарами відповідає принципу інверсії залежностей. Внутрішні шари не залежать від зовнішніх. Доменна логіка не знає про API чи базу даних, Application-шар працює з абстракціями, а Infrastructure-шар надає конкретні реалізації цих абстракцій. Завдяки цьому бізнес-правила залишаються стабільними навіть у разі зміни технічних деталей, наприклад ORM, способу автентифікації або зовнішніх сервісів [26].

Класи з FitLead.Domain описують бізнес-об'єкти системи, їхню поведінку та правила зміни стану. Наприклад, Invitation є агрегатом, який містить методи створення, прийняття, відкликання та перевірки стану запрошення. Внутрішній стан агрегату приховується через приватні сеттери, а зміна даних виконується лише через методи, у яких перевіряються бізнес-правила.

Далі наведена стисла реалізація методу створення запрошення:

```
public static Result<Invitation> Create(
    Guid trainerId,
    string tokenHash,
    DateTime createdAtUtc,
    DateTime expiresAtUtc)
{
    if (trainerId == Guid.Empty)...
    if (string.IsNullOrEmpty(tokenHash))...
    if (expiresAtUtc <= createdAtUtc)...
    return Result<Invitation>.Success...
}
```

Також агрегат Invitation містить інші методи для роботи зі станом запрошення:

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public Result Accept(Guid clientId, DateTime acceptedAtUtc)...
public Result Revoke(DateTime revokedAtUtc)...
public bool IsExpired(DateTime utcNow)...
private Result EnsurePending()...

```

Збереження даних виконується через Entity Framework Core. У цій реалізації EF Core використовується не як основа доменної моделі, а як інфраструктурний механізм роботи з базою даних. Доменні об'єкти залишаються частиною бізнес-логіки, а правила їх відображення у таблицях бази даних описуються окремо в інфраструктурному шарі.

Основна робота з базою даних зосереджена в проєкті FitLead.Infrastructure, у папці Persistence. Центральним класом для взаємодії з базою даних є FitLeadDbContext. Він наслідується від IdentityDbContext, тому один контекст використовується як для доменних таблиць застосунку, так і для таблиць ASP.NET Identity. Це дозволяє в межах одного EF контексту працювати з користувачами доменної моделі, identity користувачами, refresh токенами та зв'язками між ними.

Конфігурація таблиць знаходиться у окремих класах, які реалізують IEntityConfiguration. У методі OnModelCreating контекст автоматично підключає всі конфігурації з поточної збірки. Такий підхід дозволяє відокремити бізнес-модель від деталей збереження. Наприклад, доменний клас User не повинен знати, що він зберігається в таблиці users або що для нього створено унікальний індекс. Ці правила описуються в окремому класі UserConfiguration:

```

public sealed class UserConfiguration
: IEntityConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.ToTable("users");
        builder.HasKey(x => x.Id);
        builder.Property(x => x.Id)
            .ValueGeneratedNever();
        ...
        builder.HasIndex(x => x.Email)
            .IsUnique();
    }
}

```

Доступ до даних організований через репозиторії. В Application-шарі оголошені інтерфейси репозиторіїв, наприклад IUserRepository. Їхні реалізації

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

знаходяться в Infrastructure-шарі та використовують FitLeadDbContext. Завдяки цьому Application-шар не залежить напряду від Entity Framework Core і не знає деталей SQL-запитів, таблиць або провайдера бази даних.

У проєкті також використовується поділ репозиторіїв на операції запису та читання. Репозиторії для запису працюють переважно з доменними агрегатами, завантажують їх, додають у контекст або видаляють. Репозиторії для читання часто одразу відображають дані в DTO, які потрібні для відповіді користувачу. Завдяки цьому система не завантажує повні доменні агрегати там, де потрібне лише отримання даних.

На рівні Application-шару цей самий принцип реалізовано через окремі папки Commands і Queries всередині бізнес-модулів (рис. 2.15).

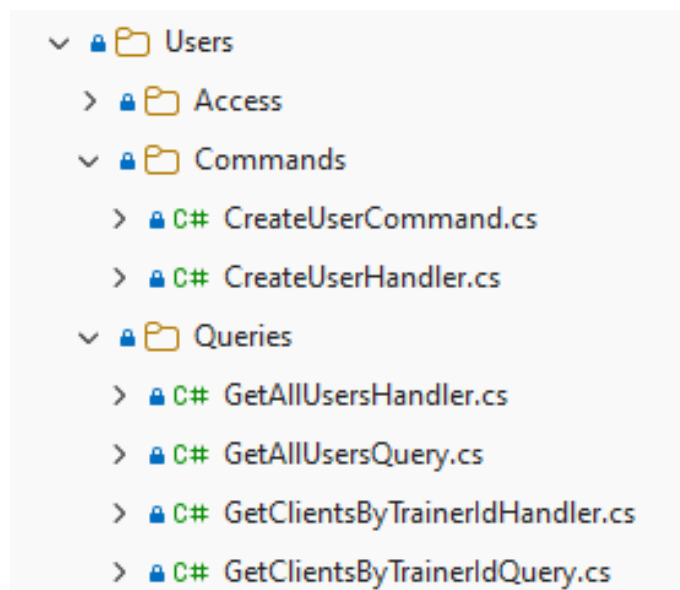


Рис. 2.15. Розділення команд та запитів всередині модуля Users

Команди відповідають за операції, які змінюють стан системи. Вони завантажують необхідні доменні об'єкти через репозиторії, перевіряють права доступу або бізнес-умови, викликають методи доменних сутностей і після цього зберігають зміни в базі даних. Запити не змінюють стан системи і використовуються для читання даних через відповідні репозиторії та повертають результат у вигляді DTO.

Для організації виконання команд і запитів у системі використовується бібліотека MediatR. Кожна команда або запит передається через mediator до відповідного обробника. Наприклад, команда CreateWorkoutCommand

обробляється класом `CreateWorkoutHandler`, а запит `GetWorkoutDetailsByIdQuery` обробляється класом `GetWorkoutDetailsByIdHandler`. Такий підхід дозволяє описувати кожен сценарій використання як окремий клас із чітко визначеною відповідальністю. MediatR виконує роль посередника між API рівнем і прикладними сценаріями. Контролери не звертаються напряму до репозиторіїв, доменних об'єктів або служб із бізнес-логікою. Натомість вони створюють команду або запит і передають їх через `_mediator.Send(...)`. Після цього виконання сценарію переходить до відповідного обробника. Завдяки цьому API шар залишається тонким і відповідає лише за HTTP маршрути, авторизацію, прийом вхідних даних і перетворення результату в HTTP відповідь.

Наприклад, контролер приймає HTTP-запит, перевіряє доступ за роллю тренера, формує команду `CreateWorkoutCommand` і передає її через MediatR:

```
var result = await _mediator.Send(
    new CreateWorkoutCommand(
        request.Name),
    cancellationToken);

return result.ToCreated(this);
```

Основна прикладна логіка виконується в `CreateWorkoutHandler`. Він отримує поточного користувача, додатково перевіряє його доменну роль, створює тренування, додає його до репозиторію та зберігає зміни. Таке розділення не прив'язує прикладний сценарій безпосередньо до контролера.

Усі команди та запити повертають результат через спільний тип `Result` або `Result<T>`. Його базова реалізація винесена в проєкт `FitLead.Common` і доступна різним шарам системи. `Result` використовується в доменній моделі, прикладних сценаріях, identity сервісах і на API рівні під час перетворення результатів у HTTP відповіді.

`Result` застосовується для явного повернення результату операції без використання винятків у звичайному бізнес-потоці. Наприклад, ситуації, коли тренування не знайдено, користувач не має доступу, назва не може бути порожньою або запрошення вже існує, є очікуваними варіантами виконання прикладного сценарію. Через `Result` такі помилки описуються явно та можуть бути оброблені без `try/catch`.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Отже, Application шар у FitLead побудований навколо окремих сценаріїв використання. Commands відповідають за зміну стану й працюють із доменними агрегатами, Queries відповідають за читання й повертають DTO, а MediatR забезпечує слабе зв'язування між контролерами та прикладною логікою [27].

Клієнтська частина системи FitLead реалізована як вебзастосунок на основі Next.js, React і TypeScript. Для побудови маршрутизації використовується App Router, який дозволяє організовувати сторінки, компонування, серверні компоненти та обробники маршрутів у межах єдиної файлової структури. Основний код клієнтської частини розміщено в каталозі src, де окремо виділено сторінки застосунку, функціональні модулі, типи даних, спільні компоненти та допоміжні бібліотеки (рис. 2.16).

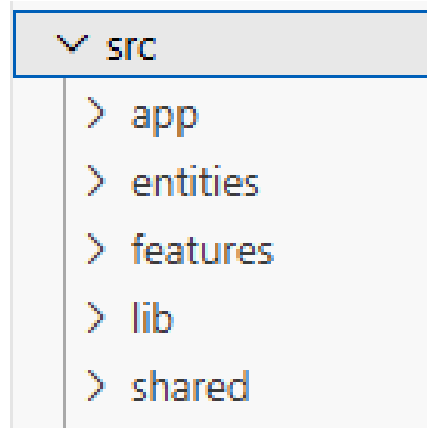


Рис. 2.16. Структура клієнтського додатку

Структура frontend-проекту побудована переважно за функціональним принципом. Папка src/app містить сторінки, компонування, групи маршрутів і route handlers, які також використовуються для реалізації BFF-шару. У папці src/features розміщено реалізацію окремих функціональних сценаріїв, зокрема авторизації, вправ, тренувань, тренувальних програм, запрошень, чатів, відеозвітів, медіафайлів і сповіщень. Компоненти інтерфейсу не зібрані в одну загальну папку, а розташовані всередині відповідних функціональних модулів. Це дозволяє зберігати код, пов'язаний з певною бізнес-можливістю, в одному місці.

У frontend частині окремо виділено типи основних даних, спільну логіку та повторно використовувані елементи інтерфейсу. У src/entities розміщено моделі

користувачів, вправ, тренувань, чатів і відеозвітів. Код для HTTP запитів, CSRF захисту, SignalR, Uploadcare та серверних API викликів зосереджено в src/lib. Спільні компоненти інтерфейсу, поля форм, повідомлення та допоміжні функції розміщено в src/shared.

Загальнодоступні сторінки розміщено в групі (marketing), сторінки входу та реєстрації в групі (auth), а основну частину особистого кабінету в групі (app). Назви груп не входять до URL-адреси, але дозволяють логічно структурувати різні частини застосунку. Наприклад, у системі реалізовано маршрути для панелі користувача, бібліотеки вправ, тренувань, тренувальних програм, профілю клієнта, запрошень, чатів і відеозвітів. Для сторінок із конкретними об'єктами використовуються динамічні сегменти маршруту, наприклад [clientId], [programId], [chatId] або [reportId].

Важливою особливістю реалізації є поділ сторінок на серверні та клієнтські компоненти. Файли page.tsx за замовчуванням є серверними компонентами. Вони використовуються для отримання поточного користувача, завантаження початкових даних, перевірки ролі та виконання перенаправлень. Наприклад, сторінка конкретного чату завантажує дані чату та початковий список повідомлень на сервері, а сторінка тренувальної програми отримує дані програми, тренування, клієнтів і призначення перед передачею їх у компоненти інтерфейсу.

Клієнтські компоненти позначаються директивою "use client" і відповідають за інтерактивну поведінку. До них належать форми входу, створення вправ, створення тренувань, надсилання відеозвітів, робота з чатом, завантаження файлів і відображення сповіщень. Саме в таких компонентах використовується локальний стан, обробка подій браузера, надсилання форм, робота з SignalR та виклик router.refresh() після зміни даних. Це дозволяє поєднати серверне завантаження початкової інформації з інтерактивністю, необхідною для роботи вебзастосунку.

Взаємодія frontend частини з сервером організована через окремі API клієнти та допоміжні функції. Серверні компоненти можуть звертатися до

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

ASP.NET API із серверного середовища Next.js, передаючи необхідні cookie з поточного запиту. Завдяки цьому вони можуть завантажувати захищені дані від імені поточного користувача. Клієнтські компоненти не звертаються безпосередньо до внутрішньої адреси backend API, а використовують маршрути Next.js, які надалі передають запити до серверної частини.

Функція buildApiUrl визначає відповідний маршрут Next.js для виконання запиту. Запити авторизації спрямовуються через /api/auth, а запити до прикладних можливостей системи через /api/backend. Виконання клієнтських HTTP запитів централізовано у спільній функції, яка формує адресу, передає метод, заголовки, тіло запиту, cookie та параметри кешування. Для запитів, що змінюють стан системи, функція додатково отримує CSRF токен і передає його в заголовок X-CSRF-TOKEN.

Окремим елементом взаємодії серверної частини із сервером є BFF шар, реалізований за допомогою Next.js Route Handlers. У FitLead для цього використано два основні напрями маршрутизації. Окремий маршрут відповідає за операції авторизації, а універсальний маршрут використовується для прикладних API запитів. Відповідні обробники розміщено у файлах src/app/api/auth/[...path]/route.ts та src/app/api/backend/[...path]/route.ts. Загальна схема взаємодії frontend частини з ASP.NET API наведена на рисунку 2.17.

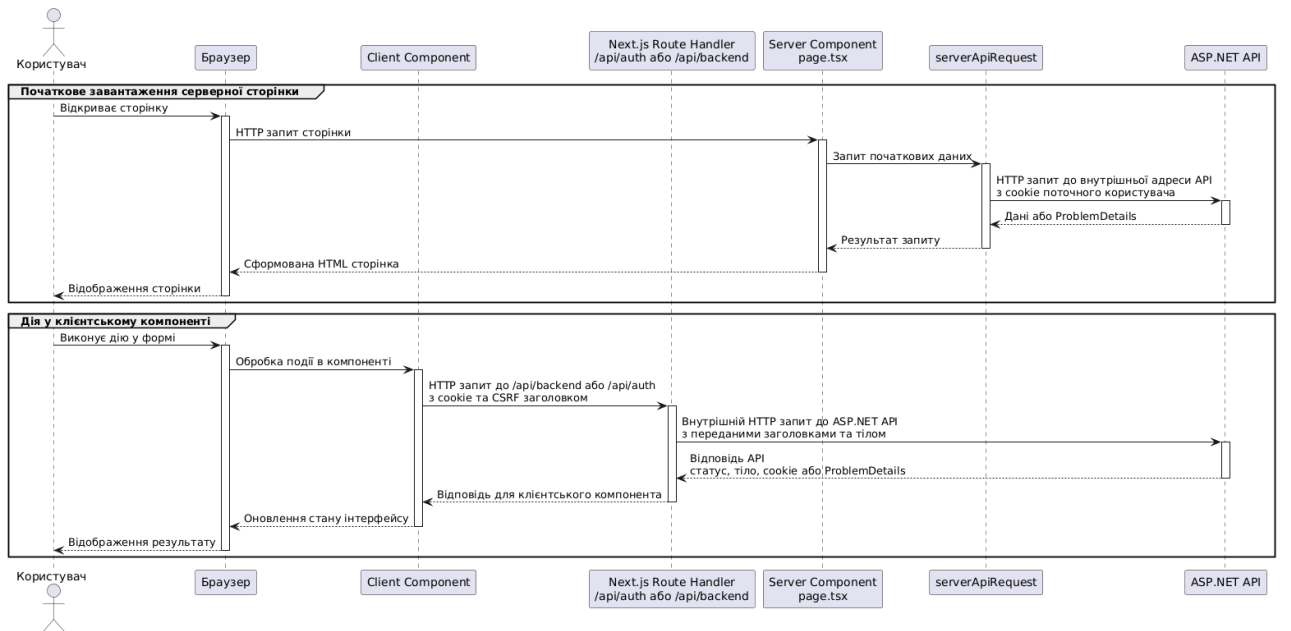


Рис. 2.17. Потоки взаємодії frontend частини з ASP.NET API у FitLead

У системі використовуються два різні потоки звернення до серверної частини. Запити з клієнтських компонентів проходять через Route Handlers, оскільки вони виконуються з браузера. Натомість серверні компоненти Next.js можуть звертатися до ASP.NET API без проміжного обробника маршруту, а за допомогою спільної функції `serverApiRequest`.

Для операцій авторизації використовується обробник маршруту `src/app/api/auth/[...path]/route.ts`. Він приймає лише визначений набір `auth` операцій, зокрема вхід, реєстрацію, вихід, оновлення сесії, отримання CSRF токена та поточного користувача.

Прикладні запити обробляються маршрутом `src/app/api/backend/[...path]/route.ts`. Він приймає запити з клієнтських компонентів і передає їх до відповідних маршрутів ASP.NET API. Наприклад, запит до `/api/backend/exercises` перетворюється на запит до `/api/exercises`, а `/api/backend/training-programs/{id}` передається на `backend` маршрут тренувальних програм.

Основна роль BFF шару полягає в тому, щоб приховати внутрішню адресу ASP.NET API від браузера та централізовано контролювати передавання `cookie`, CSRF заголовків і відповідей серверної частини. Завдяки цьому клієнтські компоненти працюють із маршрутами Next.js, а бізнес логіка, авторизація та основні перевірки залишаються на рівні ASP.NET API.

Для роботи з медіафайлами у FitLead використовується зовнішнє сховище Uploadcare. Такий підхід дозволяє не передавати великі файли через ASP.NET API та не зберігати їхній вміст у базі даних. У PostgreSQL зберігаються лише структуровані записи про медіафайли, зокрема ідентифікатор об'єкта у сховищі, адресу доступу, тип файлу, розмір, власника та стан запису. Послідовність роботи з медіафайлом наведена на рисунку 2.18.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

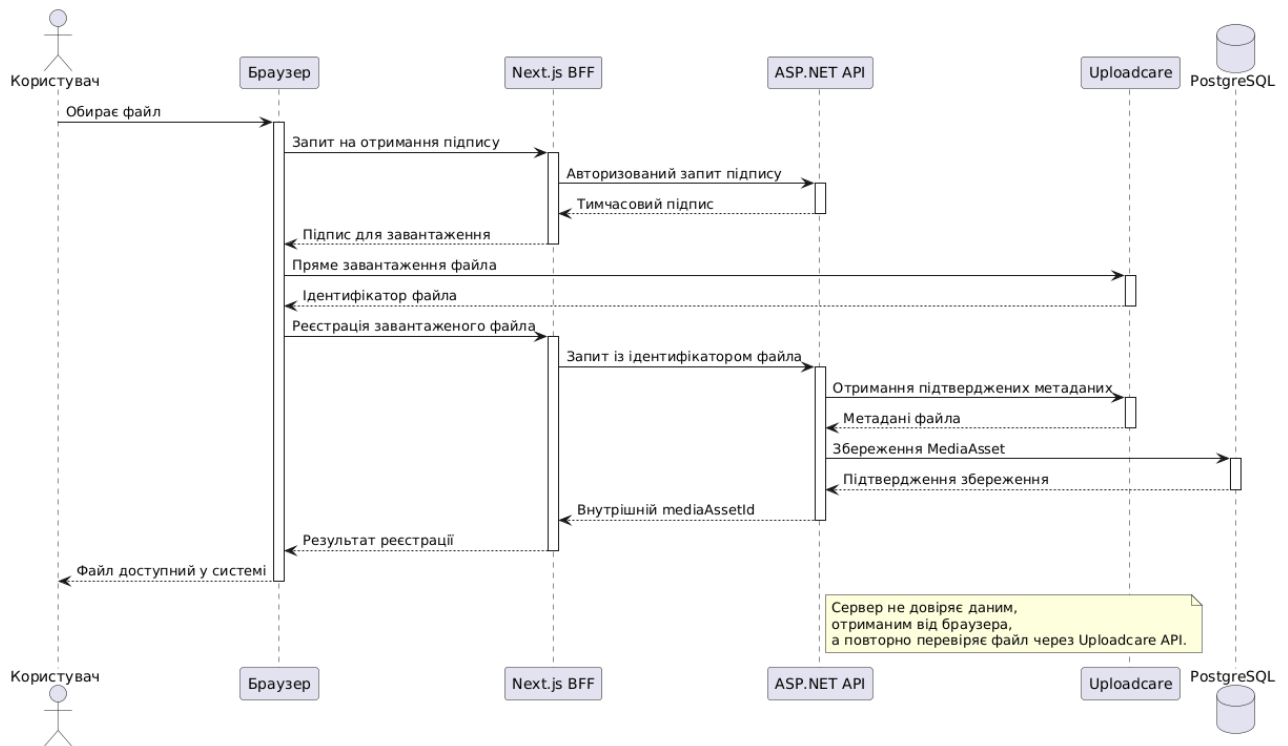


Рис. 2.18. Послідовність роботи з медіафайлом

Процес роботи з медіафайлом складається з двох основних етапів. Спочатку frontend отримує тимчасовий підпис для захищеного завантаження та передає файл безпосередньо до Uploadcare. На цьому етапі файл не проходить через сервер FitLead, що зменшує навантаження на ASP.NET API.

Після успішного завантаження frontend надсилає до FitLead запит на реєстрацію файлу. Цей запит проходить через Next.js BFF до ASP.NET API. Серверна частина не використовує дані з браузера як остаточне джерело інформації, а повторно звертається до Uploadcare API за ідентифікатором об'єкта та отримує підтверджені метадані. Це дозволяє перевірити, що файл справді існує у сховищі, а його характеристики не були підмінені на стороні клієнта.

На основі перевірених даних створюється доменний об'єкт MediaAsset. Власником медіафайла автоматично призначається поточний користувач, тому ідентифікатор власника не приймається з frontend частини. Після створення доменного об'єкта запис додається до репозиторію та зберігається в базі даних.

Зареєстрований медіафайл надалі використовується в інших функціональних модулях системи через внутрішній ідентифікатор mediaAssetId. Наприклад, він може бути пов'язаний із вправою, фотографією прогресу клієнта

або відеозвітом. У кожному з таких сценаріїв сервер додатково перевіряє, що медіафайл належить поточному користувачу, має активний стан і відповідає допустимому типу для конкретної операції.

Використання Uploadcare дозволяє відокремити зберігання файлів від основної бази даних і серверної логіки застосунку. База даних містить лише записи про медіаактиви, а самі файли зберігаються в зовнішньому сховищі. Це зменшує розмір бази даних, спрощує резервне копіювання, знижує навантаження на сервер застосунку та дає змогу використовувати зовнішню інфраструктуру для доставки медіафайлів користувачам.

Одним із ключових засобів взаємодії тренера і клієнта у FitLead є чат. У системі чат створюється для одного тренера та одного клієнта. Групові чати в поточній реалізації не передбачені, оскільки чат використовується як приватний канал супроводу клієнта тренером.

У доменній моделі чат представлений сутністю Chat. Вона описує приватний канал між тренером і клієнтом, зберігає ідентифікатори учасників, дату створення та час останнього повідомлення. Створення чату відбувається через доменний метод Chat.Create, а оновлення останньої активності виконується через метод MarkMessageCreated. Прикладні обробники не змінюють ці дані напрямую, а лише координують виконання сценарію.

Окремі повідомлення зберігаються в сутності ChatMessage. Вона містить ідентифікатор чату, автора, тип повідомлення, текст або посилання на відеозвіт, а також час створення. У поточній реалізації використовуються текстові повідомлення та повідомлення, пов'язані з відеозвітом. Доменна сутність ChatMessage також містить правила створення повідомлення, зокрема перевірку учасника чату, наявності тексту та допустимої довжини повідомлення.

Під час надсилання текстового повідомлення сервер спочатку перевіряє доступ поточного користувача до чату. Після цього створюється доменний об'єкт повідомлення, оновлюється час останньої активності чату та додається Outbox подія Messenger.ChatMessageCreated. Повідомлення, оновлення чату та подія доставки зберігаються одним викликом SaveChangesAsync, що забезпечує

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

узгодженість між основними даними та подією для подальшої доставки. Послідовність створення повідомлення наведена на рисунку 2.19.

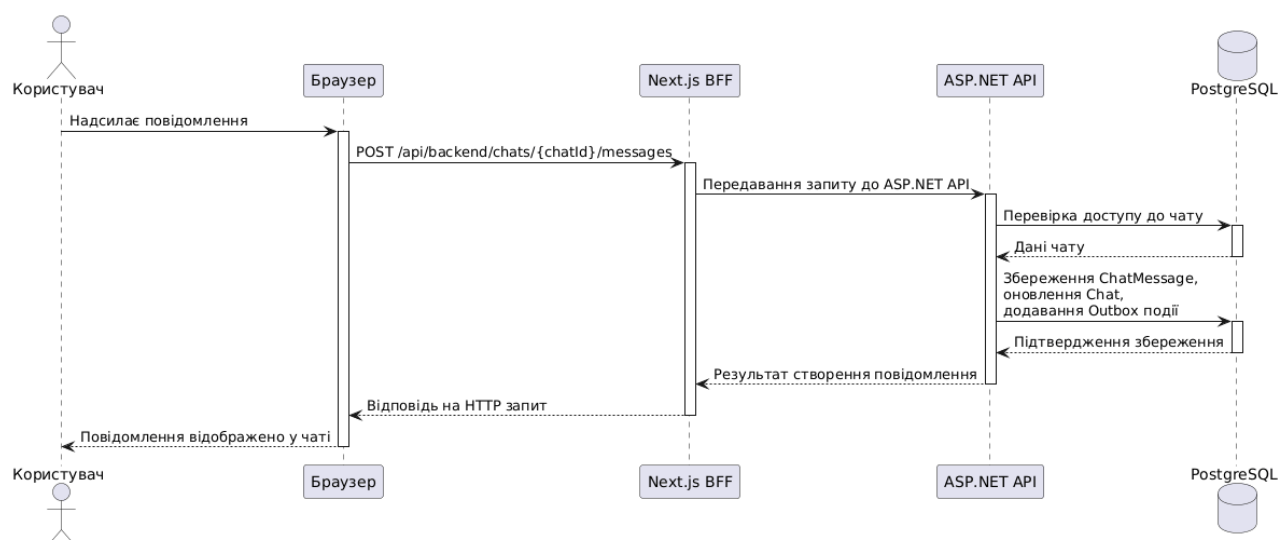


Рис. 2.19. Створення повідомлення в чаті FitLead

Після створення повідомлення користувач отримує відповідь на HTTP запит, а повідомлення може бути відображене в інтерфейсі відправника. Однак доставка іншим активним учасникам чату виконується окремо через механізм Outbox і SignalR. Це дозволяє не прив'язувати основний запит створення повідомлення до доставки в реальному часі. Асинхронна доставка повідомлення наведена на рисунку 2.20.

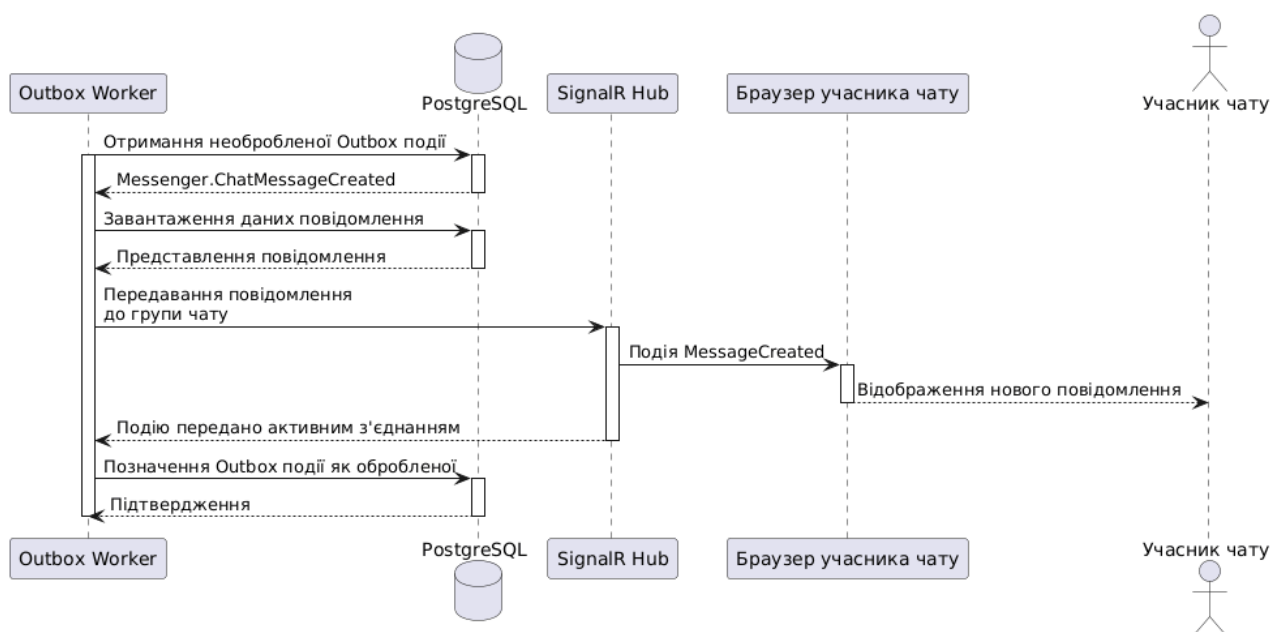


Рис. 2.20. Доставка повідомлення через Outbox і SignalR

Outbox Worker отримує подію Messenger.ChatMessageCreated, завантажує з бази даних представлення створеного повідомлення та передає його до SignalR групи відповідного чату. Повідомлення надсилається лише активним з'єднанням користувачів, які приєднані до цієї групи.

Приєднання до SignalR групи також захищене. SignalR вузол вимагає авторизації, а метод JoinChat додатково перевіряє, чи має користувач доступ до конкретного чату. Тому користувач не може підключитися до довільної групи лише за ідентифікатором чату.

Отже, реалізація чату поєднує звичайні HTTP запити та обмін даними в реальному часі. HTTP використовується для перевірки доступу, створення повідомлення і збереження його в базі даних, Outbox забезпечує асинхронну обробку події, а SignalR доставляє створене повідомлення активним учасникам відповідного чату.

Для асинхронної обробки подій у FitLead використовується шаблон програмування Outbox. До таблиці outbox_messages записуються службові повідомлення, які мають бути оброблені після виконання основної прикладної операції. Наприклад, під час створення повідомлення чату, надсилання відеозвіту або призначення тренувальної програми система додає відповідну Outbox-подію. Завдяки цьому основні зміни в базі даних і подія для подальшої обробки зберігаються разом.

Outbox-повідомлення містить тип події, дані події, час створення, стан обробки та службову інформацію для повторних спроб. У системі використовуються стани Pending, Processed і Failed. Повідомлення зі станом Pending очікує на обробку, Processed означає успішне завершення, а Failed використовується для подій, які не вдалося обробити після вичерпання кількості спроб.

Фонова служба Outbox періодично перевіряє таблицю outbox_messages, вибирає повідомлення, готові до обробки, і передає їх відповідному обробнику за типом події. Якщо обробка завершується успішно, повідомлення переходить у стан Processed. Якщо виникає помилка, система або планує повторну спробу з

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

урахуванням затримки, або переводить повідомлення у стан Failed. Такий підхід дозволяє не втрачати прикладні події у випадку тимчасових збоїв.

Outbox використовується не лише для чату, а й для механізму сповіщень. Після виконання прикладної дії, наприклад призначення тренувальної програми або надсилання відеозвіту, відповідний обробник Outbox події створює запис у таблиці notifications для конкретного одержувача. Перед створенням сповіщення система перевіряє, чи не існує вже запису для тієї самої початкової події. Це дозволяє уникнути дублювання під час повторної обробки Outbox-повідомлення.

Формування сповіщення і його доставка розділені між різними подіями. Початкова подія описує бізнесовий факт, наприклад призначення програми клієнту, а подія Notification.Created використовується для доставки вже створеного сповіщення через доступні канали. Для активного інтерфейсу користувача використовується SignalR, завдяки чому нове сповіщення може з'явитися без повторного завантаження сторінки.

Крім доставки через SignalR, система підтримує Web Push для зареєстрованих пристроїв користувача. Якщо Web Push увімкнений, сповіщення надсилається на активні підписки користувача. Помилка доставки на один пристрій не зупиняє обробку інших підписок, тому тимчасова проблема з окремим браузером або пристроєм не блокує доставку через інші доступні канали.

Таким чином, Outbox у FitLead відповідає за фонову та повторювану обробку подій, таблиця notifications зберігає самі сповіщення користувачів, SignalR доставляє їх активним користувачам, а Web Push використовується для зареєстрованих пристроїв. Така структура дає змогу відокремити створення сповіщення від його доставки та підвищує надійність обробки подій.

Висновки до другого розділу

У другому розділі розглянуто проектування та реалізацію основних компонентів вебплатформи FitLead. Спочатку було визначено, як із системою взаємодіють тренер і клієнт, які сценарії для них є основними та які функції повинна підтримувати платформа. На основі цих сценаріїв сформовано

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

структуру предметної області, виділено ключові сутності системи та зв'язки між ними.

Подальше проєктування було зосереджене на побудові моделі даних і логіки роботи системи. Для збереження інформації визначено таблиці, зв'язки, обмеження, необхідні для роботи з користувачами, тренувальними програмами, чатами, відеозвітами, медіафайлами, сповіщеннями та службовими подіями. Окремо описано алгоритми для процесів, у яких важливими є контроль доступу, узгодженість даних і безпечна обробка запитів.

Практична реалізація системи виконана з поділом на серверну та клієнтську частини. Серверна частина побудована як модульний моноліт із розділенням доменної, прикладної, інфраструктурної та API-логіки. Клієнтська частина реалізована на основі Next.js, React і TypeScript, а взаємодія з серверною частиною організована через BFF шар. Така структура дала змогу відокремити бізнес-логіку від деталей інтерфейсу, роботи з базою даних і зовнішніми сервісами.

У межах реалізації також розроблено механізми, необхідні для повноцінної роботи платформи. До них належать авторизація користувачів, CSRF захист, робота з медіафайлами через зовнішнє сховище, приватний чат між тренером і клієнтом, відеозвіти, сповіщення, оновлення в реальному часі та фонові обробка подій через Outbox. Ці компоненти пов'язані між собою та забезпечують виконання основних функціональних вимог платформи.

Отже, результатом другого розділу стала спроектована й реалізована основа вебплатформи FitLead. Архітектурні рішення, модель даних, алгоритми та програмні компоненти формують цілісну структуру системи.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ

3.1. Розгортання вебплатформи та налаштування неперервної інтеграції

Після реалізації основних функціональних можливостей системи необхідно забезпечити її запуск у стабільному середовищі та автоматизувати перевірку змін у коді. Для цього у FitLead використовується контейнеризований підхід до розгортання та засоби неперервної інтеграції. Контейнеризація дає змогу запускати серверну частину, клієнтську частину та базу даних в ізольованих середовищах із визначеними залежностями. Використання CI/CD є доцільним і для невеликих програмних проєктів, оскільки дає змогу автоматизувати перевірку змін, зменшити кількість ручних дій і зробити процес підготовки програмного продукту до запуску більш контрольованим [28].

Загальну схему розгортання вебплатформи наведено на рисунку 3.1.

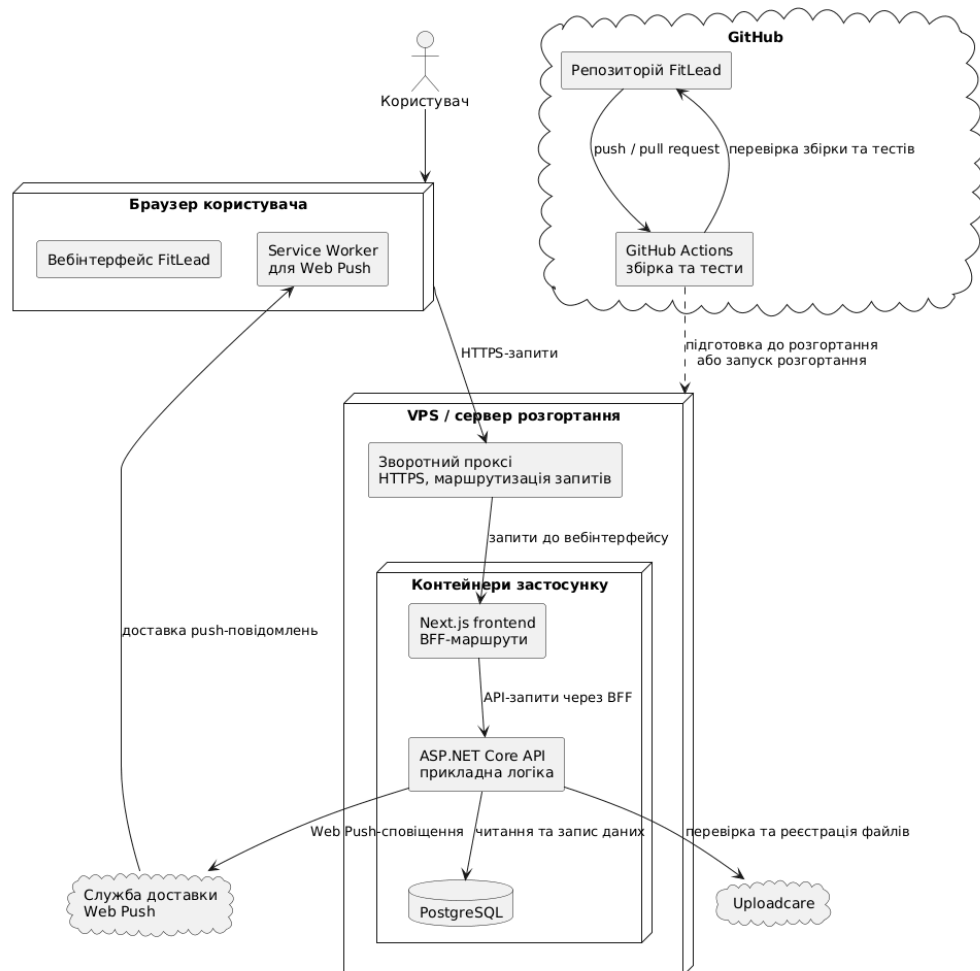


Рис. 3.1. Загальна схема розгортання вебплатформи FitLead

Схема розгортання системи передбачає роботу користувача через браузер, який взаємодіє з клієнтською частиною вебплатформи. Клієнтська частина звертається до серверної частини через маршрути Next.js і BFF-шар, а серверна частина виконує прикладну логіку, працює з базою даних PostgreSQL та взаємодіє із зовнішніми сервісами, зокрема Uploadcare і службами Web Push. Вхідні HTTP-запити до застосунку обробляються через зворотний проксі, який спрямовує їх до відповідних контейнерів.

Маршрутизація запитів у середовищі розгортання налаштовується через Dokploy. Клієнтська частина fitlead-web доступна за основним шляхом / і спрямовується на контейнерний порт 3000. Серверна частина fitlead-api використовується для обробки SignalR-з'єднань і доступна за шляхом /hubs, який спрямовується на порт 8080. Завдяки цьому основний вебінтерфейс відкривається через Next.js, а з'єднання в реальному часі передаються до ASP.NET Core API. Налаштування маршрутизації контейнерів наведено на рисунках 3.2 і 3.3.

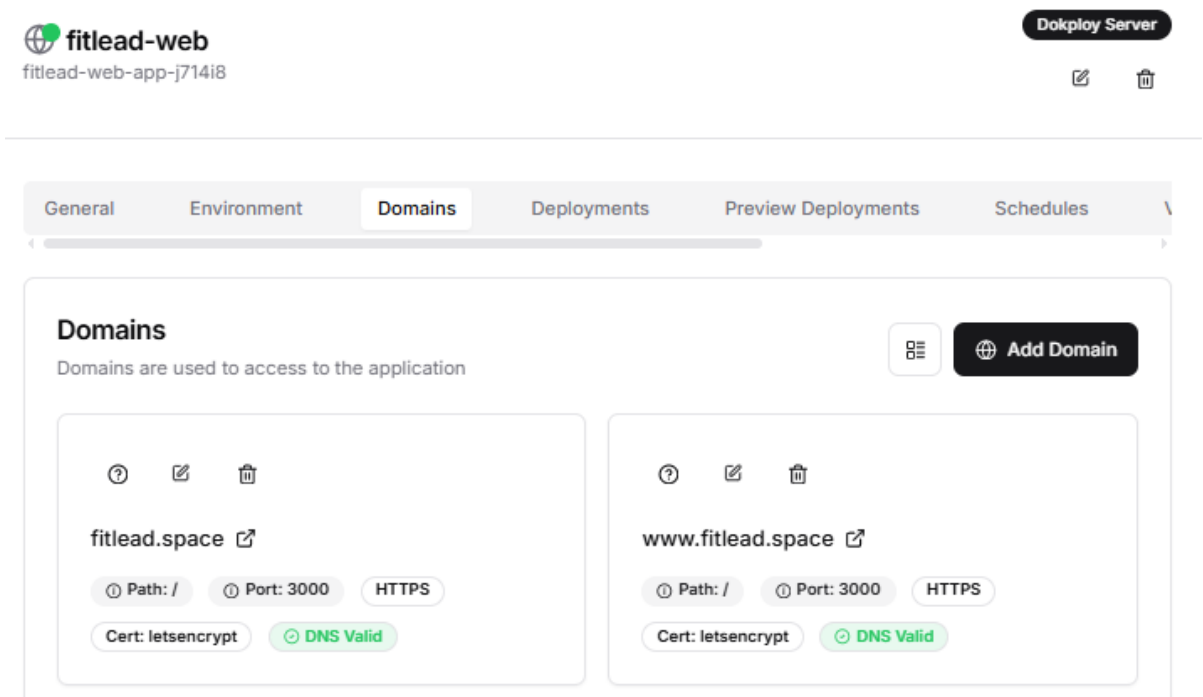


Рис. 3.2. Маршрутизація запитів до frontend контейнера у Dokploy

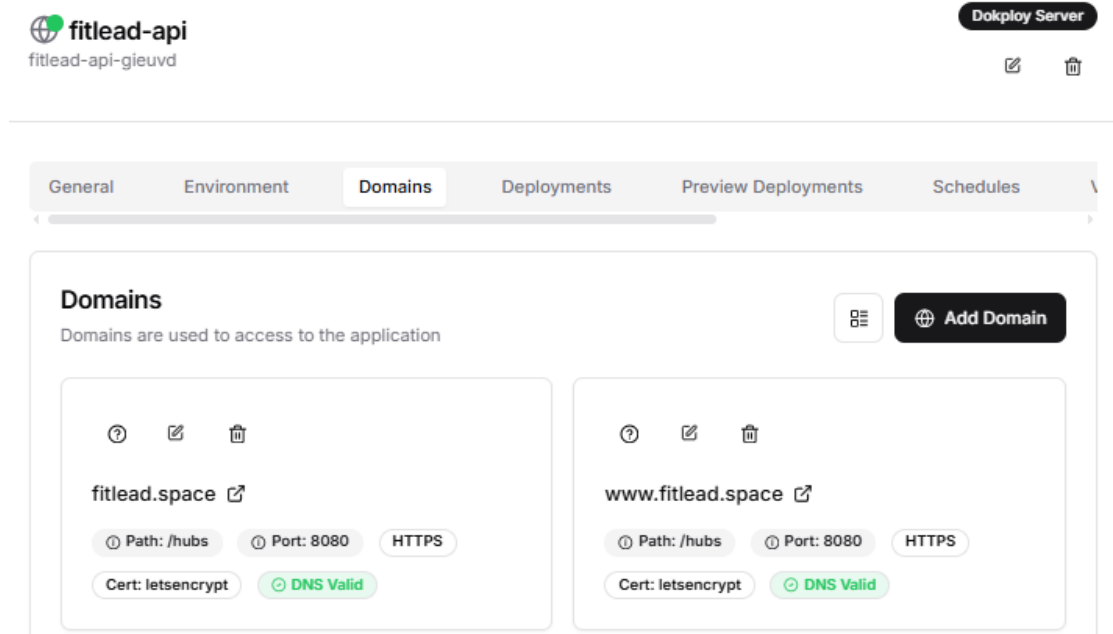


Рис. 3.3. Маршрутизація запитів до API-контейнера у Dokploy

Конфігурація компонентів FitLead винесена за межі програмного коду та задається через змінні середовища у середовищі розгортання. Для серверної частини до таких параметрів належать рядок підключення до PostgreSQL, налаштування авторизації, параметри cookie, ключі Uploadcare, VAPID-ключі для Web Push та адреси клієнтської частини. Значення цих параметрів не зберігаються у відкритому репозиторії, оскільки частина з них є секретами. Клієнтська частина також використовує змінні середовища, однак значення з префіксом NEXT_PUBLIC_ потрапляють до браузерної збірки, тому в них передаються лише публічні параметри, наприклад публічний ключ Uploadcare.

Для спрощення розгортання компонентів FitLead використовується контейнеризація за допомогою Docker. Серверна та клієнтська частини мають окремі Dockerfile, що дає змогу збирати й запускати їх як самостійні контейнери. Такий підхід зменшує залежність від налаштувань конкретного сервера та дозволяє запускати застосунок у відтворюваному середовищі. Загальний процес автоматичної перевірки, збирання та розгортання компонентів наведено на рисунку 3.4.

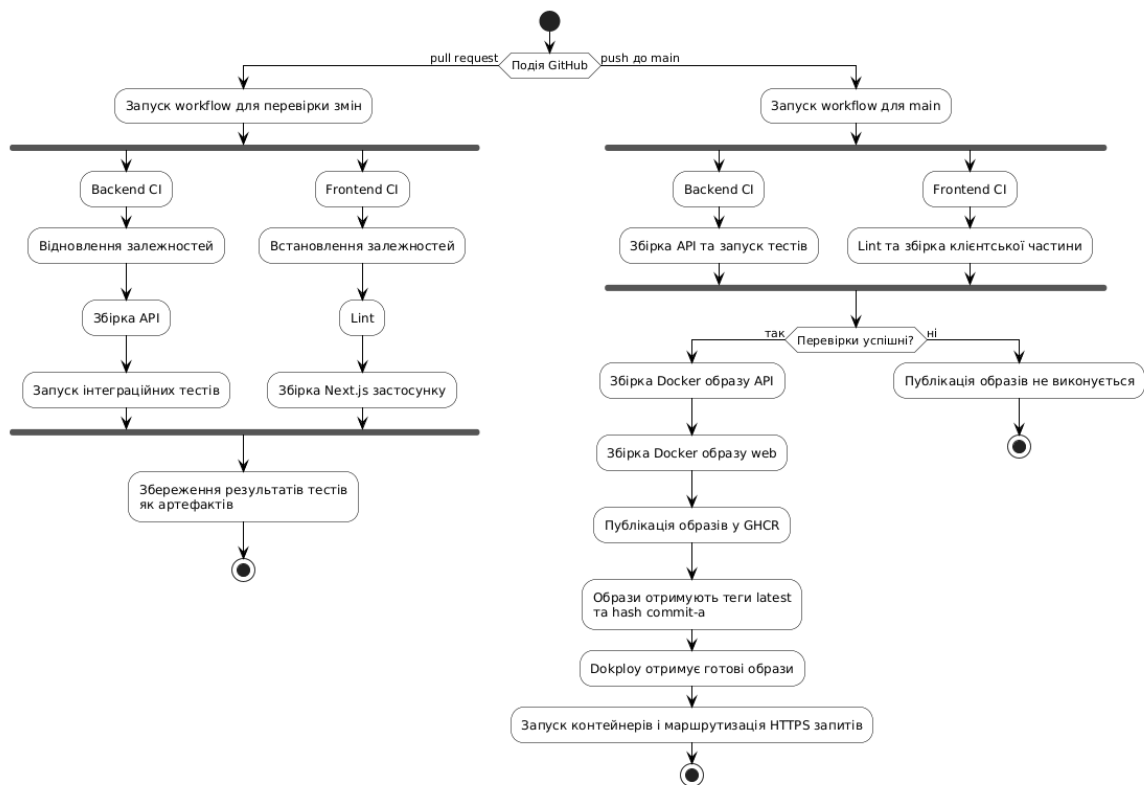


Рис. 3.4. CI/CD та розгортання компонентів FitLead

Серверна частина контейнеризується за допомогою багатоетапного збирання. На першому етапі використовується образ .NET SDK, у якому відновлюються залежності та виконується публікація застосунку в конфігурації Release. На другому етапі результат публікації копіюється до меншого образу ASP.NET Runtime, який містить лише компоненти, необхідні для запуску API. Клієнтська частина також збирається у декілька етапів. Спочатку встановлюються залежності, далі виконується production-збірка Next.js застосунку, після чого формується кінцевий runtime-образ. Повні лістинги Dockerfile для серверної та клієнтської частин наведено в додатку Б.

Для автоматизації перевірки змін у проєкті використовується GitHub Actions. Workflow описано у файлі `.github/workflows/ci.yml`. Він запускається під час створення або оновлення pull request до гілок main і dev, під час push до гілки main, а також вручну через workflow_dispatch. Це дозволяє перевіряти зміни до їхнього об'єднання з основними гілками та окремо запускати перевірку за потреби. Повний лістинг workflow файлу наведено в додатку Б.

Перевірка серверної частини виконується в завданні backend-ci. У ньому відновлюються залежності, збирається API проєкт, збирається проєкт інтеграційних тестів і запускаються тести з формуванням результатів у форматі .trx та даних покриття коду. Після виконання тестів результати зберігаються як артефакт workflow.

Перевірка клієнтської частини виконується в завданні frontend-ci. У ньому GitHub Actions готує середовище Node.js, встановлює залежності відповідно до package-lock.json, перевіряє frontend код і виконує збірку Next.js застосунку.

Публікація Docker образів винесена в окреме завдання publish-images. Воно виконується лише після успішного завершення backend-ci та frontend-ci і запускається тільки для push у гілку main. Завдяки цьому pull request використовується для перевірки змін, але не публікує контейнерні артефакти. У межах цього завдання збираються Docker образи серверної та клієнтської частин і публікуються до GitHub Container Registry з тегом latest та тегом, що відповідає hash поточного коміту.

Процес розгортання відокремлено від процесу збирання. GitHub Actions відповідає за перевірку коду, запуск тестів, збирання клієнтської частини та публікацію Docker образів. Сервер розгортання через Dokploy отримує вже готові образи, запускає контейнери, передає змінні середовища та налаштовує маршрутизацію зовнішніх HTTPS запитів. Клієнтський контейнер обробляє основний вебінтерфейс, а серверний контейнер використовується для backend запитів, зокрема для з'єднань у реальному часі через SignalR.

Отже, налаштований процес CI/CD забезпечує перевірку змін перед об'єднанням із основними гілками, автоматичне збирання контейнерних образів після успішних перевірок у main та підготовку артефактів для подальшого розгортання. Це дозволяє запускати на сервері не вихідний код, а вже зібрані й перевірені Docker образи.

3.2. Опис інтерфейсу вебплатформи

Інтерфейс вебплатформи FitLead адаптовано під різний розміру екрану, зокрема і мобільні пристрої. Тренери та клієнти мають свої кабінети, де

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

відбувається їхня робота на платформі. Неавторизований користувач бачить сторінки що дають змогу ознайомитися з платформою. Головна сторінка зображена на рисунку 3.5.

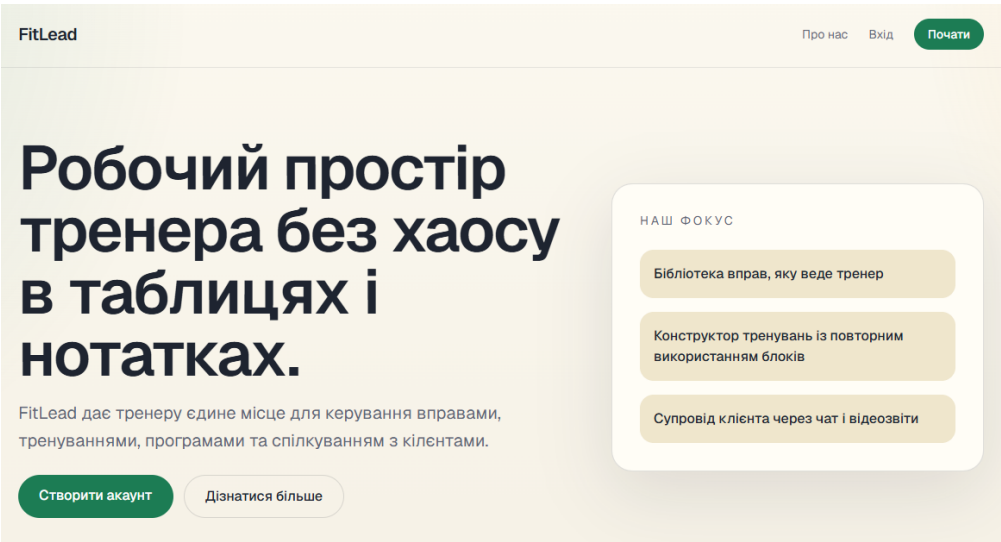


Рис. 3.5. Головна сторінка доступна неавторизованим користувачам

На цій сторінці користувач може зрозуміти, на чому зосереджена платформа. У розділі «Про нас» коротко описано можливості для тренера і клієнта, а також показано основний інструмент супроводу в системі (рис. 3.6).

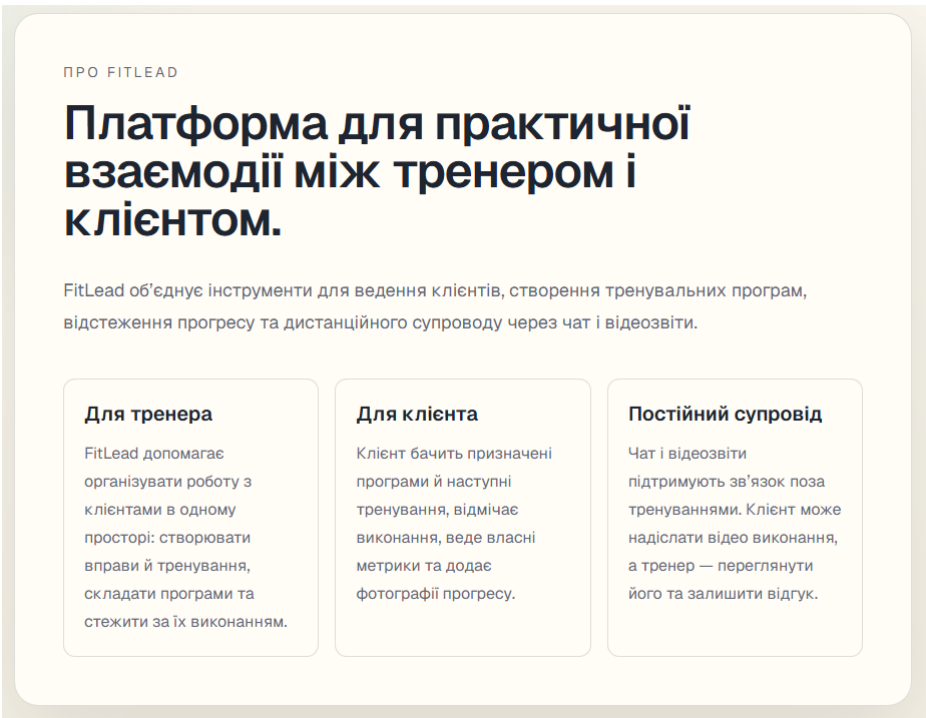


Рис. 3.6. Додаткова інформація про платформу

Щоб почати чи продовжити роботу з основним функціоналом платформи користувач має зареєструватись чи увійти відповідно.

Далі показано інтерфейс функціоналу тренера, так як він містить більшість функцій, керує програмами запрошенням клієнта до співпраці.

У кабінеті тренера відображається навігація між основними розділами платформи (рис. 3.7). У верхній частині показано дані поточного користувача, його роль і кнопку сповіщень. Через панель навігації тренер може перейти до клієнтів, запрошень, чатів, відеозвітів, вправ, тренувань і програм, а також вийти з облікового запису.

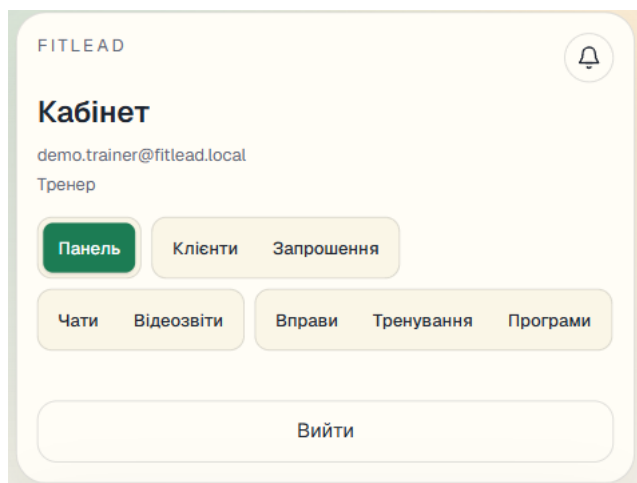


Рис. 3.7. Навігаційна панель для тренера

Початковою сторінкою для тренера є панель, на якій зібрано короткий стан його роботи в системі. У верхній частині показано кількість підключених клієнтів, активних призначених програм, нових сповіщень і відеозвітів, які очікують на відгук. Так тренер одразу бачить, які розділи потребують уваги, без переходу на кожную сторінку окремо (рис. 3.8).

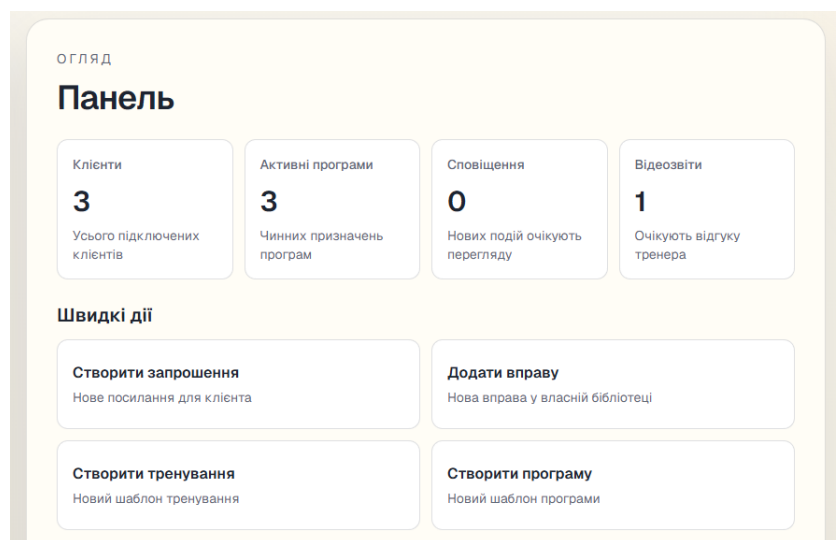


Рис. 3.8. Сторінка «Панель» для тренера

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Нижче розміщено блок швидких дій. Через нього тренер може створити запрошення для нового клієнта, додати вправу до власної бібліотеки, створити шаблон тренування або нову тренувальну програму. Ці дії винесено на панель, оскільки вони часто використовуються під час підготовки роботи з клієнтами.

Окремі інформаційні блоки на панелі є інтерактивними. Наприклад, блок із відеозвітами реагує на наведення курсора та відкриває відповідну сторінку зі звітами, які очікують перевірки. Це важливо для роботи тренера, оскільки відеозвіти використовуються для оцінки техніки виконання вправ і надання клієнту зворотного зв'язку (рис. 3.9).

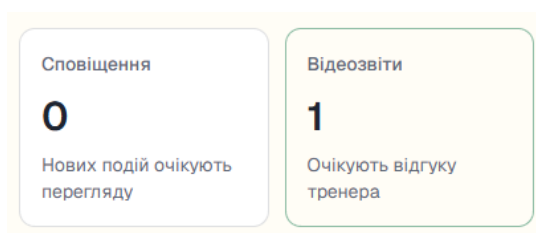


Рис. 3.9. Ефекти під час наведення на область про відеозвіти

Це є особливо корисним у контексті відеозвітів, адже це є основним засобом супроводу клієнта тренером. Тренер має переглядати відеозвіти клієнтів, які надсилають результати тренувань та очікують відгуку.

Сторінка «Запрошення» необхідна щоб додати клієнта (рис. 3.10). Сторінка містить область для створення запрошення та перегляду стану минулих запрошень.

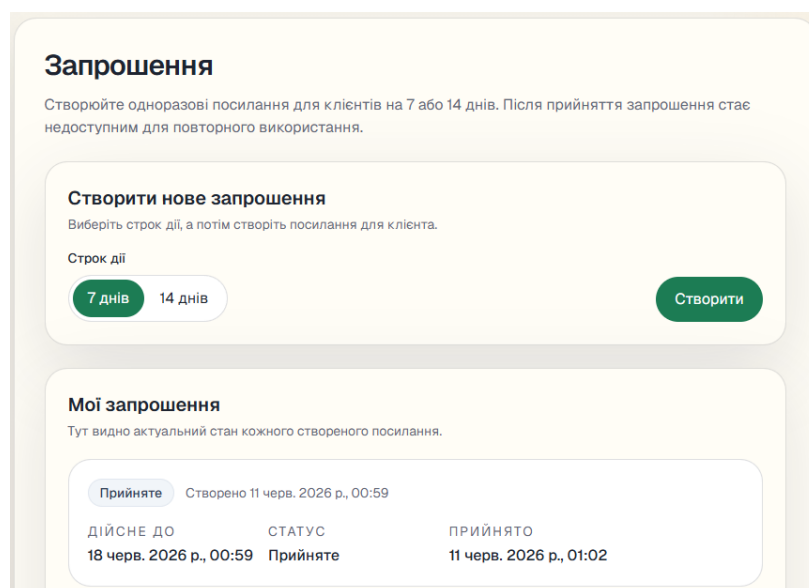


Рис. 3.10. Сторінка «Запрошення»

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

Після натискання на кнопку «Створити» тренер отримує посилання, яке має надіслати клієнту через довільний канал комунікації (рис. 3.11). Платформа не є місцем для пошуку тренерів і не пропонує маркетингові можливості для тренерів.

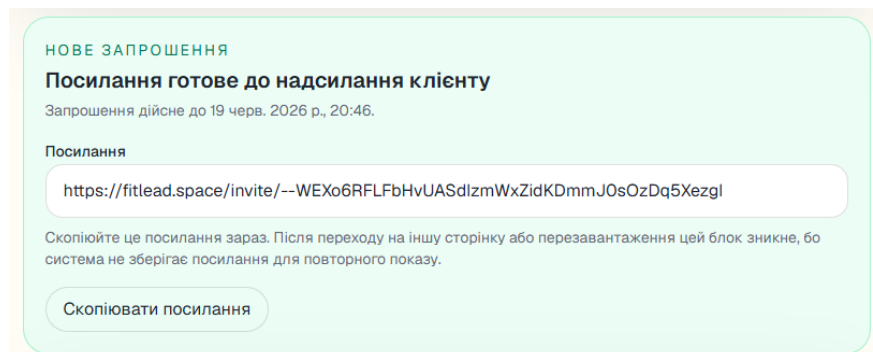


Рис. 3.11. Сформоване посилання для запрошення клієнта

Клієнт може побачити інформацію без входу до системи, проте щоб прийняти запрошення, має спершу увійти. Сторінка із запрошення для клієнта зображена на рисунку 3.12.

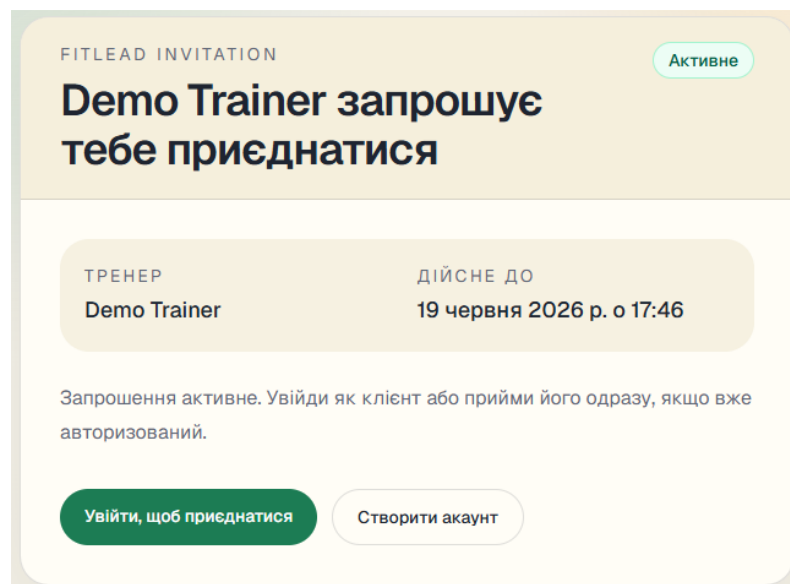


Рис. 3.12. Вигляд запрошення для неавторизованого клієнта

Наступна сторінка тренера це «Клієнти». Вона містить список клієнтів та можливість їх пошуку. Додатково з цієї сторінки можна побачити інформацію про тренувальну програму клієнта, відкрити чат з ним та відкрити сторінку клієнта. Сторінка клієнта це область, де тренер може бачити всі необхідні дані клієнта (рис. 3.13).

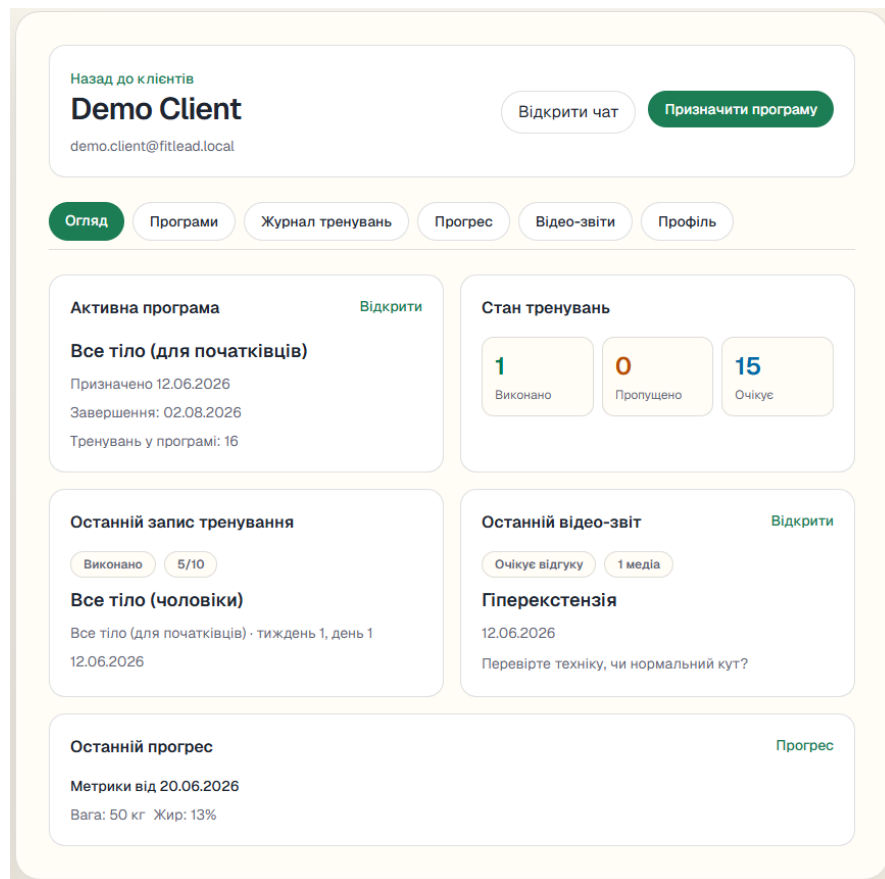


Рис. 3.13. Сторінка для роботи з конкретним клієнтом

Вкладка «Огляд» містить основну інформацію про клієнта, яка необхідна тренеру. Тут видно активну програму, стан тренувань по програмі, останній запис тренування, відеозвіт та прогрес клієнта. Також тренер може відкрити чат та призначити програму цьому клієнту, якщо така ще не призначена. Сторінка дозволяє тренеру зручно працювати з клієнтами та отримувати швидкий доступ до основної інформації та містить вкладки, де знаходяться усі необхідні дані про клієнта.

Вебплатформа має вбудований чат для спілкування тренера з конкретним клієнтом (рис. 3.14). У чаті можна обмінюватися текстовими повідомленнями, а також переглядати відеозвіти, які клієнт надсилає на перевірку. Біля назви чату відображається індикатор підключення, що показує стан з'єднання в реальному часі. Відеозвіт у чаті подається окремим повідомленням зі статусом розгляду та кнопкою для відкриття деталей.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

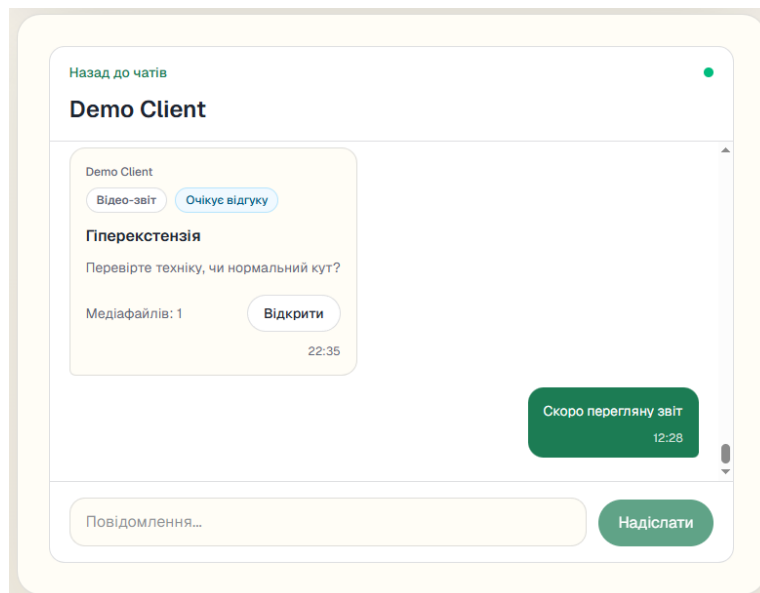


Рис. 3.14. Чат між тренером і клієнтом із надісланим відеозвітом

Відеозвіт містить назву, короткий опис від клієнта та прикріплені медіафайли (рис. 3.15). Клієнт може додати до звіту до п'яти фото або відео, які допомагають тренеру оцінити техніку виконання вправи. Після перегляду матеріалів тренер залишає текстовий відгук і надсилає його клієнту. Після цього звіт переходить у стан перевіреного, а клієнт отримує відповідне сповіщення.

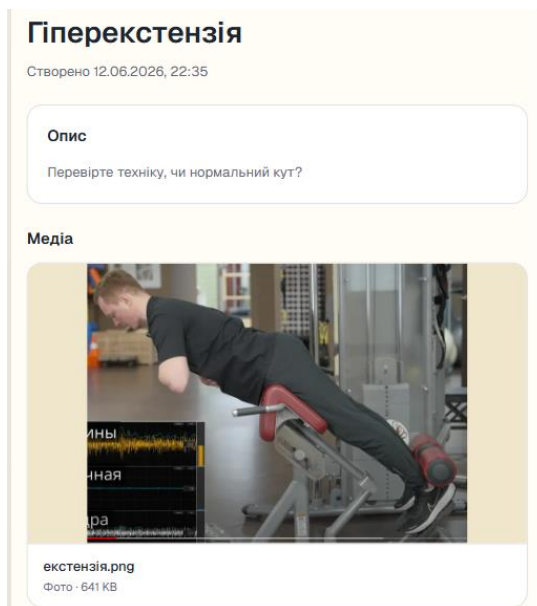


Рис. 3.15. Структура відеозвіту

На сторінці «Відеозвіти» тренер бачить звіти клієнтів, які ще очікують на перевірку (рис. 3.16). Такий розділ винесено окремо, щоб тренеру не потрібно було шукати непереглянуті матеріали в різних чатах. Після відкриття звіту він може переглянути прикріплені фото або відео та залишити клієнту відгук.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савицький Р.С.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

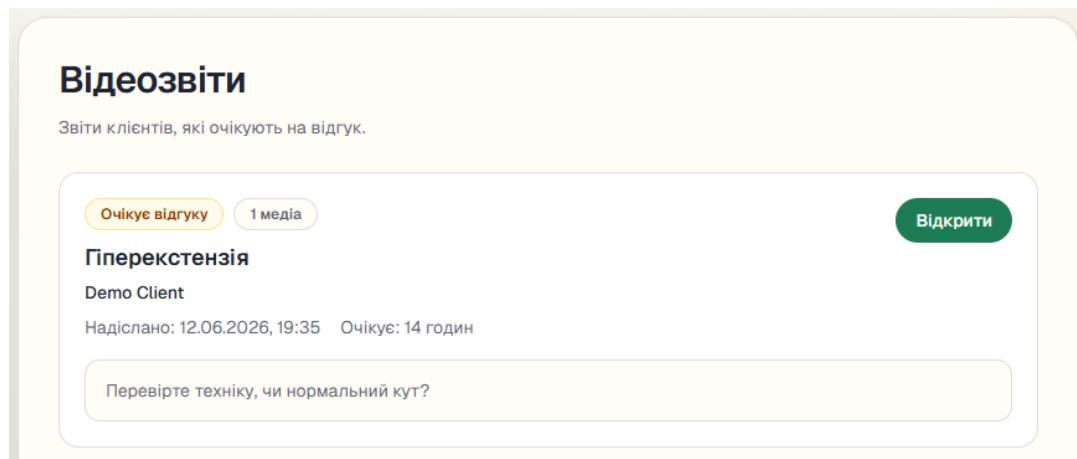


Рис. 3.16. Сторінка «Відеозвіти»

Наступні сторінки безпосередньо відповідають за формування тренувальної програми. Сторінка «Вправи» містить власну бібліотеку тренера та готову бібліотеку від платформи. Тренер може створити свою вправу, а може використовувати заготовлені вправи, щоб скопіювати та змінити під себе (рис. 3.17). Це допомагає швидше почати створювати тренування, програми, та ознайомитися з іншим функціоналом платформи.

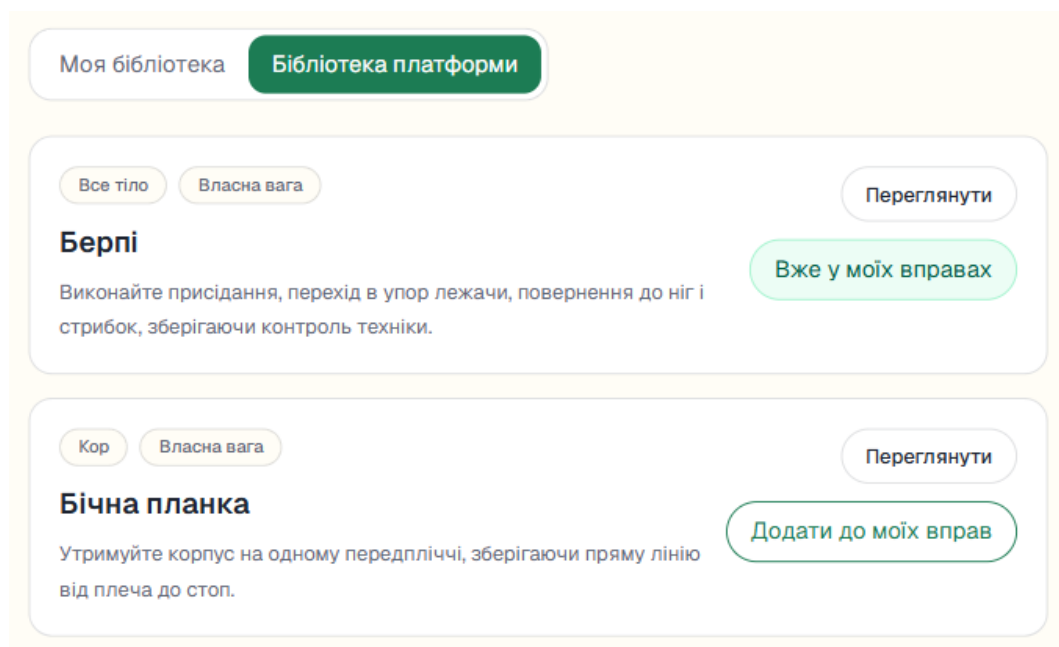


Рис. 3.17. Вправи платформи та опція копіювання

Під час редагування чи створення вправи тренер може змінити параметри та додати свій медіа файл, тобто відео чи фото для пояснення.

На сторінці «Тренування» тренер створює шаблони занять і наповнює їх вправами. Під час додавання вправи відкривається форма вибору з бібліотеки, де доступні вправи тренера та вправи платформи. Для кожної вправи задаються

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

параметри виконання: кількість підходів, повторень, вага, час відпочинку та примітка тренера (рис. 3.18). Після додавання вправа стає частиною шаблону тренування.

Вибір вправи

Оберіть вправу з бібліотеки та задайте параметри виконання.

УсіМоїПлатформа

Пошук вправи

Берпі

Бібліотека

Все тіло

Власна вага

Виконайте присідання, перехід в упор лежачи, повернення до ніг і стрибок, зберігаючи контроль техніки.

Бічна планка

Бібліотека

Кор

Власна вага

Утримуйте корпус на одному передпліччі, зберігаючи пряму лінію від плеча до стоп.

Випади вперед

Бібліотека

Ноги

Власна вага

Зробіть крок вперед, опустіть заднє коліно донизу та поверніться у вихідне положення, тримаючи корпус стабільним.

Підходи

Повторення

Вага, кг

Відпочинок, сек

3

10

60

Нотатка тренера

Додати до тренуванняСкасувати

Рис. 3.18. Додавання вправ з параметрами виконання до тренування

На сторінці «Програми» тренер так само бачить список тренувальних програм та може створювати нові. Сторінка з детальним переглядом тренувальної програми зображена на рисунку 3.19.

Назад

Все тіло (для початківців)

4 тиж. · 7 дн./тиждень

Призначити клієнту

Призначені клієнти

Активні клієнти показані одразу, неактивні приховані.

1

Всього

1

Активні

0

Завершили

Demo Client

Активний

Manual

Відкликати

Призначено: 12.06.2026 · Доступ до: 02.08.2026

Тиждень: 1234

День 1

1 трен.

1

Тренування

Відкрити

Все тіло (чоловіки)

Прибрати

Додати тренування

День 2

Порожньо

Тренування не додано

Додати тренування

Рис. 3.19. Інформація та керування тренувальною програмою

На цій сторінці тренер бачить клієнтів які виконують програму, може також призначати її клієнтам. Також тренер бачить кількість днів та тижнів які він вказував під час створенні програми. Тренувальна програма не прив'язана до календаря, а містить структуру тижнів та днів. На кожен день та тиждень тренер може додавати створені раніше тренування, таким чином формуючи тренувальну програму.

Клієнт має головну сторінку свого кабінету «Панель» (рис. 3.20). На ній показано тренера, з яким працює клієнт, активну програму та найближче тренування за цією програмою. З цієї сторінки клієнт може перейти до чату з тренером, відкрити призначену програму або одразу переглянути наступне тренування. У блоці активної програми також відображається поточний стан виконання.

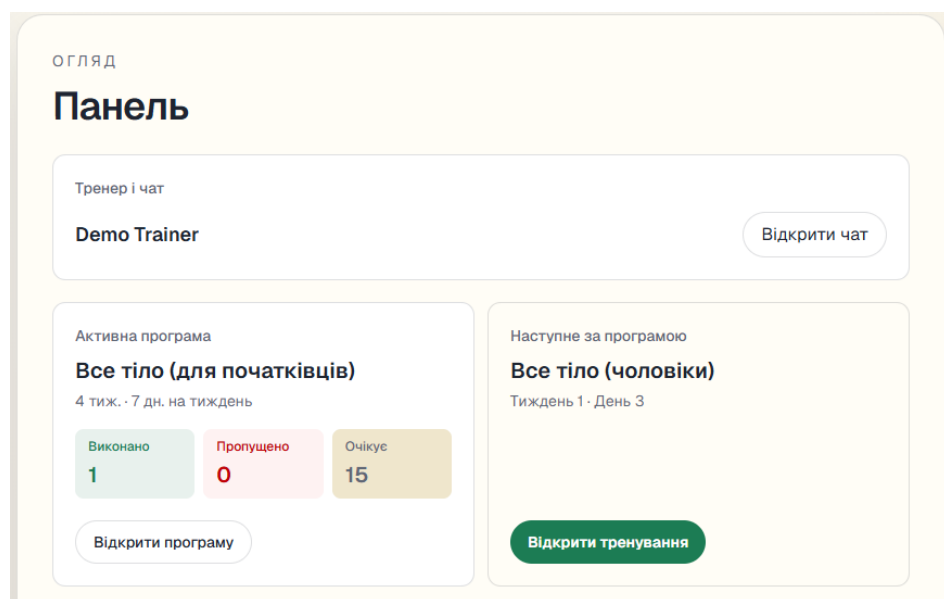


Рис. 3.20. Сторінка «Панель» для клієнта

З цієї сторінки клієнт може відкрити чат з тренером, перейти до програми та одразу до наступного тренування за програмою.

Програма виглядає як список днів, де також можна переключатися між тижнями програми. Далі клієнт може переглянути конкретне тренування та деталі виконання вправ (рис. 3.21). Натиснувши на саму вправу, можна перейти на сторінку з її опис, FitLead дає можливість краще ознайомитися з інструкціями тренера.

Назад до програми

Тиждень 1 День 1 #1

Все тіло (чоловіки)

Вправи

1 Груді Гантелі

Жим гантелей лежачи

Лежачи на лаві, вичавлюйте гантелі вгору над грудьми та повільно опускайте їх до комфортної амплітуди.

Підходи 3 Повторення 15 Вага 5 кг Відпочинок 120 сек

2 Ноги Тренажер

Жим ногами

Виштовхуйте платформу ногами, не блокуючи коліна повністю та не відриваючи таз від сидіння.

Підходи 3 Повторення 15 Вага 20 кг Відпочинок 60 сек

Рис. 3.21. Деталі тренування у програмі

Клієнт також має сторінку особистого профілю (рис. 3.22). У ній він вказує ціль тренувань, рівень підготовки, зріст, обмеження та побажання до занять. Окремі вкладки призначені для метрик тіла і фото прогресу. Ці дані використовуються тренером під час перегляду профілю клієнта та підготовки подальшого супроводу.

Профіль клієнта

Профіль Метрики Фото прогресу

Ціль

Набір м'язів.

Рівень підготовки Початковий Зріст, см 182

Обмеження / застереження

-

Побажання до тренувань

Силові тренування та кардіо.

Рис. 3.22. Профіль клієнта

Користувачі отримують сповіщення про основні події платформи, зокрема призначення програми, надсилання відеозвіту або відповідь на нього, а за

потреби можуть увімкнути браузерні сповіщення, щоб бачити такі події без відкриття сайту.

3.3. Тестування роботи програмного продукту

У межах проєкту FitLead тестування організовано як частину процесу розробки, а не як окремий завершальний етап. Основний акцент зроблено на серверній частині, оскільки саме вона містить бізнес-правила, роботу з базою даних і обробку змін стану системи. Такий підхід дає змогу перевіряти не лише окремі методи, а й поведінку функціональних сценаріїв у взаємодії кількох шарів застосунку.

Для серверної частини FitLead використано інтеграційні тести, які перевіряють роботу API разом із прикладною логікою, Entity Framework Core та базою даних. Тестовий сервер ASP.NET Core створюється за допомогою WebApplicationFactory, а PostgreSQL запускається в контейнері. Це дозволяє перевіряти сценарії з реальною реляційною базою даних, її обмеженнями, транзакціями та поведінкою запитів, а не зі спрощеною заміною в оперативній пам'яті. WebApplicationFactory використовується для створення тестового сервера в інтеграційних тестах ASP.NET Core, а Testcontainers for .NET дає змогу запускати PostgreSQL контейнер для тестового середовища [29].

Інтеграційні тести виконуються в ізольованому середовищі з окремою тестовою базою даних. Під час перевірок створюються користувачі, тренування, програми та інші записи, тому стан бази очищується між тестами. Це зменшує вплив попередніх запусків на наступні перевірки та дозволяє повторювано тестувати основні сценарії роботи системи.

Тести додавалися поступово разом зі стабілізацією функціональності. Під час додавання нового функціоналу відповідні сценарії покривалися тестами, щоб зменшити ризик регресії в основних частинах системи. Окремо перевірялися не лише успішні сценарії, а й ситуації з помилками доступу або некоректними діями користувача. До таких перевірок належать спроби виконати дію без авторизації, доступ до чужих даних, робота з неіснуючими об'єктами та повторне виконання операцій, які мають обмеження за станом. Це важливо для FitLead, оскільки

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

система працює з даними тренера і клієнта, а помилка в правилах доступу може призвести до відображення або зміни чужої інформації.

Для оцінювання покриття серверного коду використовується coverlet. Він формує дані про покриття рядків, гілок та інших елементів .NET-проєкту [30]. У поточному стані проєкту покриття рядків серверного коду становить 79%, а покриття гілок 59% (рис. 3.23).

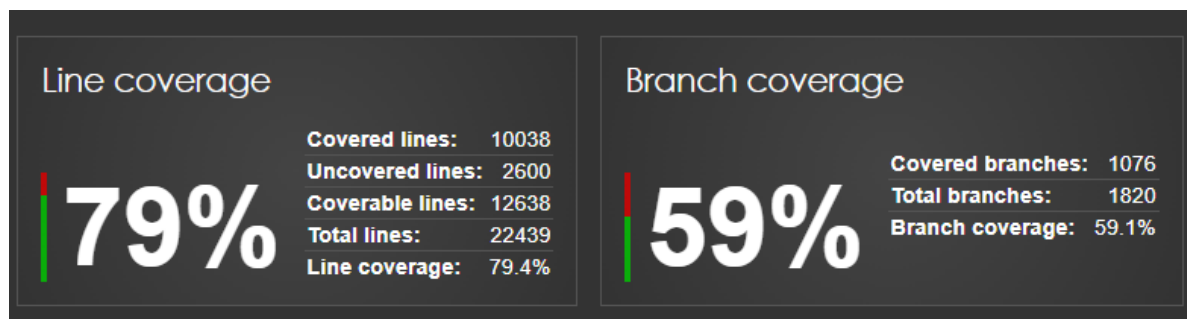


Рис. 3.23. Результати покриття коду тестами

Звіт про покриття використовується для аналізу стану тестів після запуску. Він показує, які модулі, класи та методи вже перевіряються автоматизованими тестами, а які залишаються поза перевіркою. Також у звіті можна зіставити покриття з показниками складності окремих методів, що допомагає визначити ділянки коду, які варто перевірити першочергово під час подальшої розробки. Водночас прості DTO, допоміжні класи або нестабілізовані сценарії не розглядаються як критичні лише через відсутність покриття.

Окремо під час розробки виконувалася ручна перевірка інтерфейсу в браузері. Вона стосувалася сценаріїв, де важливими є не тільки відповіді API, а й стан сторінки після дії користувача: перенаправлення після входу, оновлення даних після створення вправи або програми, відображення повідомлень про помилки, робота форм, відкриття модальних вікон, завантаження медіафайлів і зміна станів відеозвітів. Такі перевірки виконувалися паралельно з реалізацією сторінок, а найбільш критичні браузерні сценарії надалі були винесені в smoke-тести.

Для клієнтської частини FitLead налаштовано smoke-тестування через Playwright. Ці тести запускають не окремі компоненти інтерфейсу, а вебплатформу в браузері та проходять основні сценарії роботи користувача.

Перевірка виконується на production-збірці Next.js, яка працює через BFF-шар із реальним ASP.NET API та окремою PostgreSQL базою даних. API у цих тестах не замінюється mock-об'єктами, тому перевіряється фактична взаємодія між frontend частиною, backend частиною, cookie, серверними компонентами та базою даних.

Такий формат тестування використовується для перевірки взаємодії між частинами системи. Тести перевіряють не тільки наявність сторінок, а й те, чи проходить користувач повний шлях від відкриття сторінки до отримання даних із backend частини.

Було реалізовано 13 frontend smoke-тестів. Вони охоплюють базову авторизацію, оновлення сесії через refresh token, захист приватних маршрутів, основні сценарії тренера і клієнта, а також відкриття чату з перевіркою real-time з'єднання. Окремо перевіряється, що неавторизований користувач потрапляє на сторінку входу, авторизований користувач відкриває свій кабінет, а після виходу з системи авторизаційні cookie більше не використовуються.

Для продуктових сценаріїв використовується окреме тестове наповнення даними. Backend у спеціальному режимі створює тренера, клієнта та тестові дані. Тести можуть перевірити реальні сторінки платформи без ручної підготовки даних перед кожним запуском. Цей режим вмикається тільки через конфігурацію тестового середовища і не використовується за замовчуванням.

Playwright налаштовано так, щоб результати тестів були стабільними та простими для аналізу. Тести виконуються одним worker-ом, кожна перевірка отримує окремий browser context, а дані браузера не переходять між тестами. Під час помилок зберігаються screenshot, video та trace, тому після невдалого запуску можна подивитися, на якому етапі сценарій зламався.

Для локального запуску створено окремий скрипт. Він готує тестове середовище, запускає PostgreSQL у контейнері, застосовує міграції, вмикає smoke-дані, запускає API, збирає frontend частину та після цього виконує Playwright-тести. Після завершення перевірки тестове середовище очищується.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

Це дає змогу запускати frontend smoke-тести локально без ручного налаштування окремої бази даних або backend сервера.

У GitHub Actions ці перевірки винесено в окреме завдання frontend-smoke (див. додаток Б). Воно запускається для pull request до основних гілок, після push у main та вручну. У CI піднімається PostgreSQL service, збирається API, застосовуються міграції, запускається backend у smoke-середовищі, після чого збирається production-версія Next.js і виконуються Playwright-тести. Якщо перевірка завершується з помилкою, GitHub Actions зберігає HTML-звіт Playwright, screenshots, videos, traces та лог API.

Результати автоматизованих перевірок використовуються під час інтеграції змін у проєкт. Якщо один із тестових етапів завершується з помилкою, workflow зупиняється, а зміни не проходять повний цикл перевірки. Для аналізу невдалого запуску зберігаються результати тестів, звіти покриття, Playwright-звіти та службові файли, які показують, на якому етапі виникла помилка.

Frontend smoke-тести також пов'язані з процесом публікації Docker-образів. Завдання publish-images запускається лише після проходження backend, frontend і smoke-перевірок. Тому контейнерні образи не публікуються, якщо основні браузерні сценарії платформи не пройшли тестування.

Висновки до третього розділу

У третьому розділі описано розгортання, інтерфейс і перевірку роботи вебплатформи FitLead. Для запуску системи використано Docker-контейнери, у яких окремо працюють серверна та клієнтська частини. Маршрутизація зовнішніх запитів налаштована через Dokploy, а перевірка змін і підготовка контейнерних образів виконуються за допомогою GitHub Actions.

Також розглянуто інтерфейс платформи для тренера і клієнта. Наведені сторінки відображають основні робочі сценарії. Інтерфейс побудований навколо повсякденної взаємодії тренера з клієнтом, а не лише навколо окремих функцій системи.

Перевірка роботи FitLead виконувалася через інтеграційні тести серверної частини та smoke-тести клієнтської частини. Backend-тести перевіряють API,

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

бізнес-правила, авторизацію, роботу з базою даних і зміну стану системи. Frontend smoke-тести запускають платформу в браузері та проходять основні сценарії через production-збірку Next.js, BFF-шар, ASP.NET API і тестову PostgreSQL базу даних.

У результаті, в третьому розділі описано розгортання вебплатформи, основні сторінки інтерфейсу та перевірку роботи системи. Система збирається в контейнерні образи, запускається в підготовленому середовищі та проходить автоматизовані перевірки перед публікацією.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіицький Р.С.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи спроектовано та реалізовано вебплатформу FitLead для організації онлайн-супроводу у сфері фітнесу. Розроблена система об'єднує інструменти для створення тренувальних програм, ведення клієнтів, обміну повідомленнями, роботи з відеозвітами, медіафайлами та сповіщеннями.

У першому розділі було досліджено предметну область, визначено основні проблеми, які виникають під час використання розрізнених інструментів для комунікації, планування тренувань і відстеження прогресу. Також було проаналізовано наявні рішення, сформовано функціональні вимоги до системи та обґрунтовано вибір технологій для реалізації серверної, клієнтської частини й інфраструктури проєкту.

У другому розділі виконано проєктування та програмну реалізацію основних компонентів FitLead. Було визначено сценарії взаємодії тренера і клієнта, побудовано модель предметної області, спроектовано структуру бази даних та описано основні алгоритми роботи системи. Серверну частину реалізовано як модульний моноліт із поділом на доменний, прикладний, інфраструктурний та API-рівні, а клієнтську частину створено на основі Next.js, React і TypeScript. Окремо реалізовано механізми авторизації, CSRF захисту, роботи з медіафайлами, приватного чату, відеозвітів, сповіщень, оновлень у реальному часі та фонові обробки подій через Outbox.

У третьому розділі описано запуск і перевірку розробленого програмного продукту. Для розгортання системи використано контейнеризацію за допомогою Docker, налаштовано маршрутизацію компонентів у середовищі Dokploy та автоматизовано перевірку змін через GitHub Actions. Також було розглянуто інтерфейс вебплатформи для тренера і клієнта та описано підхід до тестування серверної частини за допомогою інтеграційних тестів.

Практичне значення отриманих результатів полягає в тому, що розроблена вебплатформа може використовуватися персональними тренерами для організації онлайн-роботи з клієнтами в єдиному середовищі. FitLead дозволяє

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

зменшити потребу у використанні сторонніх інструментів, зробити взаємодію між тренером і клієнтом більш структурованою та спростити супровід тренувального процесу. Розроблений програмний продукт може бути запущений як пробна версія для обмеженої кількості користувачів з метою збору відгуків, перевірки зручності основних сценаріїв і визначення подальших напрямів розвитку платформи.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Online/Virtual Fitness Market (2022–2032): Size, Share, Competitive Landscape and Trend Analysis Report [Електронний ресурс]. URL: <https://www.alliedmarketresearch.com/virtual-online-fitness-market> (дата звернення: 13.04.2026).
2. Фітнес тренер онлайн – персональні тренування онлайн з тренером [Електронний ресурс]. URL: <https://buki.com.ua/tutors-online/fitnes-trener/> (дата звернення: 13.04.2026).
3. Ukraine Crisis Response Plan 2026 [Електронний ресурс]. URL: <https://crisisresponse.iom.int/response/ukraine-crisis-response-plan-2026> (дата звернення: 13.04.2026).
4. Kemp S. Digital 2024: Ukraine [Електронний ресурс]. URL: <https://datareportal.com/reports/digital-2024-ukraine> (дата звернення: 13.04.2026).
5. Kemp S. Digital 2026: Ukraine [Електронний ресурс]. URL: <https://datareportal.com/reports/digital-2026-ukraine> (дата звернення: 13.04.2026).
6. Гольцев К. О., Савіцький Р. С. Використання Clean Architecture та патерну CQRS з MediatR у розробці фітнес-платформи. Комп'ютерні інтелектуальні системи та мережі: матеріали XIX Всеукраїнської науково-практичної WEB конференції аспірантів, студентів та молодих вчених, м. Кривий Ріг, 25–27 березня 2026 р. Кривий Ріг: Криворізький національний університет, 2026. С. 78–79.
7. Гольцев К. О., Савіцький Р. С. Розробка фітнес-платформ на основі Next.js. Комп'ютерні технології: інновації, проблеми, рішення: тези доповідей VIII Всеукраїнської науково-технічної конференції, м. Житомир, 02–03 грудня 2025 р. Житомир: Житомирська політехніка, 2025. С. 199–200.
8. ABC Trainerize [Електронний ресурс]. URL: <https://www.trainerize.com/> (дата звернення: 28.04.2026).

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

9. Trainero.com [Електронний ресурс]. URL: <https://trainero.com/uk> (дата звернення: 28.04.2026).
10. Trainerfu [Електронний ресурс]. URL: <https://www.trainerfu.com/index.html> (дата звернення: 28.04.2026).
11. What Is a Modular Monolith? [Електронний ресурс]. URL: <https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith> (дата звернення: 05.05.2026).
12. Microservices architecture style [Електронний ресурс]. URL: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices> (дата звернення: 05.05.2026).
13. Modular Monolith Communication Patterns [Електронний ресурс]. URL: <https://www.milanjovanovic.tech/blog/modular-monolith-communication-patterns> (дата звернення: 05.05.2026).
14. Еванс Е. Domain-Driven Design: Tackling Complexity in the Heart of Software. Upper Saddle River: Addison-Wesley, 2003. 560 с.
15. Mondal S., Mondal D., Roy C. K. Investigating Technology Usage Span by Analyzing Users' Q&A Traces in Stack Overflow [Електронний ресурс]. 30th Asia-Pacific Software Engineering Conference (APSEC 2023). 2023. URL: <https://arxiv.org/abs/2312.03182> (дата звернення: 05.05.2026).
16. Why Develop Websites Using ASP.NET? [Електронний ресурс]. URL: <https://www.cisin.com/coffee-break/why-develop-websites-using-asp-net.html> (дата звернення: 06.05.2026).
17. Overview of ASP.NET Core [Електронний ресурс]. URL: <https://learn.microsoft.com/hr-hr/aspnet/core/overview?view=aspnetcore-9.0> (дата звернення: 06.05.2026).
18. PostgreSQL Global Development Group. About PostgreSQL [Електронний ресурс]. URL: <https://www.postgresql.org/about/> (дата звернення: 06.05.2026).

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савицький Р.С.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

19. PostgreSQL Global Development Group. *JSON Types* [Електронний ресурс]. URL: <https://www.postgresql.org/docs/current/datatype-json.html> (дата звернення: 06.05.2026).
20. Popov R., Gerasimov V. Strategic considerations for database systems: PostgreSQL vs. Microsoft SQL Server. Automation of Technological and Business Processes. 2025. Vol. 17, № 3. P. 54–59. DOI: <https://doi.org/10.15673/atbp.v17i3.3156>.
21. React Server Components RFC. *GitHub: reactjs/rfcs*. [Електронний ресурс]. URL: <https://github.com/reactjs/rfcs/blob/main/text/0188-server-components.md> (дата звернення: 07.05.2026).
22. The Web Almanac 2024: JavaScript Performance. *HTTP Archive*. [Електронний ресурс]. URL: <https://almanac.httparchive.org/en/2024/javascript> (дата звернення: 08.05.2026).
23. Next.js Documentation: Optimizing JavaScript. *Vercel Inc.* [Електронний ресурс]. URL: <https://nextjs.org/docs/app/building-your-application/optimizing/scripts> (дата звернення: 08.05.2026).
24. Web Vitals – Patterns for a Faster Web. *web.dev by Google Chrome Team*. [Електронний ресурс]. URL: <https://web.dev/vitals/> (дата звернення: 10.05.2026).
25. Laigner R., Almeida A. C., Assunção W. K. G., Zhou Y. An Empirical Study on Challenges of Event Management in Microservice Architectures [Електронний ресурс]. URL: <https://arxiv.org/abs/2408.00440> (дата звернення: 10.05.2026).
26. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
27. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.
28. Ccallo-Luque M., Quispe-Quispe A. Adoption and Adaptation of CI/CD Practices in Very Small Software Development Entities: A Systematic Literature

Review [Електронний ресурс]. URL: <https://arxiv.org/abs/2410.00623> (дата звернення: 15.05.2026).

29.Til J. van der, Costello M., Calvarro Nelson J. Integration tests in ASP.NET Core [Електронний ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-10.0&pivots=xunit> (дата звернення: 14.05.2026).

30.Coverlet: cross platform code coverage for .NET [Електронний ресурс]. URL: <https://github.com/coverlet-coverage/coverlet> (дата звернення: 15.05.2026).

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савінький Р.С.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

Опис таблиці users

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор користувача.
Email	varchar(256)	Адреса електронної пошти користувача.
FullName	varchar(200)	Повне ім'я користувача.
Role	integer	Роль користувача: тренер або клієнт.

Таблиця 2

Опис таблиці invitations

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запрошення.
TrainerId	uuid	Ідентифікатор тренера, який створив запрошення.
TokenHash	text	Захищене значення ключа запрошення.
Status	integer	Поточний стан запрошення.
CreatedAtUtc	timestamp with time zone	Дата і час створення запрошення.
ExpiresAtUtc	timestamp with time zone	Дата і час завершення дії запрошення.
AcceptedByClientId	uuid, NULL	Ідентифікатор клієнта, який прийняв запрошення.
AcceptedAtUtc	timestamp with time zone, NULL	Дата і час прийняття запрошення.

Таблиця 3

Опис таблиці client_profiles

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор профілю.
ClientId	uuid	Ідентифікатор клієнта.
Goal	varchar(500), NULL	Тренувальна ціль клієнта.
ExperienceLevel	integer, NULL	Рівень тренувального досвіду.
HeightCm	integer, NULL	Зріст клієнта в сантиметрах.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

продовж. табл. 3

Limitations	varchar(1000), NULL	Фізичні обмеження або протипоказання.
TrainingPreferences	varchar(1000), NULL	Тренувальні вподобання клієнта.
AdditionalInfo	varchar(1000), NULL	Додаткова інформація про клієнта.
CreatedAtUtc	timestamp with time zone	Дата і час створення профілю.
UpdatedAtUtc	timestamp with time zone, NULL	Дата і час останнього оновлення профілю.

Таблиця 4

Опис таблиці client_body_metric_entries

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запису.
ClientId	uuid	Ідентифікатор клієнта.
RecordedAt	date	Дата вимірювання.
WeightKg	numeric(6,2), NULL	Вага клієнта в кілограмах.
BodyFatPercent	numeric(5,2), NULL	Відсоток жирової тканини.
ChestCm	numeric(6,2), NULL	Обхват грудей у сантиметрах.
WaistCm	numeric(6,2), NULL	Обхват талії у сантиметрах.
HipsCm	numeric(6,2), NULL	Обхват стегон у сантиметрах.
ArmCm	numeric(6,2), NULL	Обхват руки у сантиметрах.
ThighCm	numeric(6,2), NULL	Обхват стегна у сантиметрах.
Note	varchar(1000), NULL	Примітка до вимірювання.
CreatedAtUtc	timestamp with time zone	Дата і час створення запису.
UpdatedAtUtc	timestamp with time zone, NULL	Дата і час останнього оновлення запису.

Таблиця 5

Опис таблиці client_progress_photos

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор фотографії.
ClientId	uuid	Ідентифікатор клієнта.
MediaAssetId	uuid	Ідентифікатор завантаженого файлу.
TakenAt	date	Дата створення фотографії.
Label	integer	Позначення ракурсу фотографії.
Note	varchar(1000), NULL	Примітка до фотографії.
CreatedAtUtc	timestamp with time zone	Дата і час додавання фотографії.

Таблиця 6

Опис таблиці exercises

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор вправи.
OwnerTrainerId	uuid, NULL	Ідентифікатор тренера-власника; відсутній для платформних вправ.
Source	integer	Джерело вправи: платформа або тренер.
CopiedFromExerciseId	uuid, NULL	Ідентифікатор платформної вправи, з якої створено копію.
Name	varchar(200)	Назва вправи.
Description	text	Опис техніки виконання вправи.
MediaAssetId	uuid, NULL	Ідентифікатор прикріпленого медіаматеріалу.
MuscleGroup	integer, NULL	Основна група м'язів.
Equipment	integer, NULL	Необхідне обладнання.

Таблиця 7

Опис таблиці workouts

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор тренування.
Name	varchar(200)	Назва тренування.
TrainerId	uuid	Ідентифікатор тренера-власника.

Таблиця 8

Опис таблиці workout_exercises

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запису.
workout_id	uuid	Ідентифікатор тренування.
ExerciseId	uuid	Ідентифікатор вправи.
Order	integer	Порядок вправи у тренуванні.
Repetitions	integer	Кількість повторень.
Sets	integer	Кількість підходів.
LoadKg	numeric, NULL	Заплановане навантаження в кілограмах.
RestSeconds	integer	Тривалість відпочинку в секундах.
TrainerNote	varchar(1000), NULL	Примітка тренера щодо виконання.

Таблиця 9

Опис таблиці training_program_workouts

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запису.
TrainingProgramId	uuid	Ідентифікатор тренувальної програми.
WorkoutId	uuid	Ідентифікатор тренування.
WeekNumber	integer	Номер тижня програми.
DayNumber	integer	Номер дня тижня.
OrderInDay	integer	Порядок тренування протягом дня.

Таблиця 10

Опис таблиці assigned_training_programs

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор призначення.
TrainerId	uuid	Ідентифікатор тренера.
ClientId	uuid	Ідентифікатор клієнта.
TrainingProgramId	uuid	Ідентифікатор призначеної програми.
Status	integer	Поточний стан призначення.
AccessSource	integer	Спосіб надання доступу до програми.

Таблиця 11

Опис таблиці workout_logs

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запису.
AssignedTrainingProgramId	uuid	Ідентифікатор призначеної програми.
TrainingProgramWorkoutId	uuid	Ідентифікатор тренування у програмі.
ClientId	uuid	Ідентифікатор клієнта.
TrainerId	uuid	Ідентифікатор тренера.
Status	integer	Стан виконання тренування.
PerformedAtUtc	timestamp with time zone, NULL	Дата і час фактичного виконання.
ClientNote	varchar(1000), NULL	Примітка клієнта.
DifficultyRating	integer, NULL	Оцінка складності тренування.
CreatedAtUtc	timestamp with time zone	Дата і час створення запису.
UpdatedAtUtc	timestamp with time zone, NULL	Дата і час останнього оновлення запису.

Таблиця 12

Опис таблиці chats

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор розмови.
TrainerId	uuid	Ідентифікатор тренера.
ClientId	uuid	Ідентифікатор клієнта.
CreatedAtUtc	timestamp with time zone	Дата і час створення розмови.
LastMessageAtUtc	timestamp with time zone, NULL	Дата і час останнього повідомлення.

Таблиця 13

Опис таблиці chat_messages

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор повідомлення.

продовж. табл. 13

ChatId	uuid	Ідентифікатор розмови.
SenderId	uuid	Ідентифікатор відправника.
Type	integer	Тип повідомлення.
Text	varchar(4000), NULL	Текст повідомлення.
VideoReportId	uuid, NULL	Ідентифікатор прикріпленого відеозвіту.
CreatedAtUtc	timestamp with time zone	Дата і час надсилання повідомлення.

Таблиця 14

Опис таблиці video_reports

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор відеозвіту.
ChatId	uuid	Ідентифікатор розмови.
ClientId	uuid	Ідентифікатор клієнта.
TrainerId	uuid	Ідентифікатор тренера.
Title	varchar(200)	Назва відеозвіту.
Description	varchar(2000), NULL	Опис відеозвіту.
Status	integer	Стан розгляду відеозвіту.
CreatedAtUtc	timestamp with time zone	Дата і час створення відеозвіту.
ReviewedAtUtc	timestamp with time zone, NULL	Дата і час перегляду тренером.
TrainerFeedback Text	varchar(4000), NULL	Відповідь тренера на відеозвіт.

Таблиця 15

Опис таблиці notifications

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор сповіщення.
RecipientUserId	uuid	Ідентифікатор отримувача.
Type	integer	Тип події, про яку створено сповіщення.
Title	varchar(200)	Заголовок сповіщення.

Опис таблиці push_subscriptions

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор підписки.
UserId	uuid	Ідентифікатор користувача.
Endpoint	varchar(2048)	Адреса служби доставки сповіщень.
P256dh	varchar(512)	Відкритий ключ підписки.
Auth	varchar(512)	Ключ перевірки підписки.
UserAgent	varchar(500), NULL	Відомості про браузер або пристрій.
CreatedAtUtc	timestamp with time zone	Дата і час створення підписки.
LastUsedAtUtc	timestamp with time zone, NULL	Дата і час останнього використання.
RevokedAtUtc	timestamp with time zone, NULL	Дата і час відкликання підписки.

Опис таблиці outbox_messages

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор події.
Type	varchar(200)	Тип події.
Payload	jsonb	Дані події у структурованому вигляді.
OccurredAtUtc	timestamp with time zone	Дата і час виникнення події.
Status	integer	Стан обробки події.
ProcessedAtUtc	timestamp with time zone, NULL	Дата і час успішної обробки.
RetryCount	integer	Кількість повторних спроб обробки.
NextRetryAtUtc	timestamp with time zone, NULL	Час наступної спроби обробки.
Error	varchar(4000), NULL	Опис помилки обробки.

Таблиця 18

Опис таблиці user_identity_links

Поле	Тип	Призначення
DomainUserId	uuid	Ідентифікатор користувача у табл. users.
IdentityUserId	text	Ідентифікатор облікового запису авторизації.
CreatedAtUtc	timestamp with time zone	Дата і час створення зв'язку.

Таблиця 19

Опис таблиці refresh_tokens

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор ключа.
IdentityUserId	varchar(450)	Ідентифікатор облікового запису авторизації.
TokenHash	text	Захищене значення ключа.
FamilyId	uuid	Ідентифікатор групи пов'язаних ключів.
CreatedAtUtc	timestamp with time zone	Дата і час створення ключа.
ExpiresAtUtc	timestamp with time zone	Дата і час завершення дії ключа.
RevokedAtUtc	timestamp with time zone, NULL	Дата і час відкликання ключа.
ReasonRevoked	text, NULL	Причина відкликання ключа.
ReplacedByTokenId	uuid, NULL	Ідентифікатор ключа, яким його замінено.

Таблиця 20

Опис таблиці AspNetUsers

Поле	Тип	Призначення
Id	text	Унікальний ідентифікатор облікового запису.
UserName	varchar(256), NULL	Ім'я користувача.
NormalizedUserName	varchar(256), NULL	Нормалізоване ім'я для пошуку.
Email	varchar(256), NULL	Адреса електронної пошти.
NormalizedEmail	varchar(256), NULL	Нормалізована адреса для пошуку.
EmailConfirmed	boolean	Ознака підтвердження електронної пошти.

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савицький Р.С.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

PasswordHash	text, NULL	Захищене значення пароля.
SecurityStamp	text, NULL	Значення для перевірки змін безпеки.
ConcurrencyStamp	text, NULL	Значення для контролю одночасних змін.
PhoneNumber	text, NULL	Номер телефону.
PhoneNumberConfirmed	boolean	Ознака підтвердження номера телефону.
TwoFactorEnabled	boolean	Ознака двоетапної перевірки.
LockoutEnd	timestamp with time zone, NULL	Час завершення блокування.
LockoutEnabled	boolean	Ознака можливості блокування.
AccessFailedCount	integer	Кількість невдалих спроб входу.

Таблиця 21

Опис таблиціAspNetRoles

Поле	Тип	Призначення
Id	text	Унікальний ідентифікатор ролі.
Name	varchar(256), NULL	Назва ролі.
NormalizedName	varchar(256), NULL	Нормалізована назва для пошуку.
ConcurrencyStamp	text, NULL	Значення для контролю одночасних змін.

Таблиця 22

Опис таблиціAspNetUserRoles

Поле	Тип	Призначення
UserId	text	Ідентифікатор облікового запису.
RoleId	text	Ідентифікатор призначеної ролі.

Таблиця 23

Опис таблиці AspNetUserClaims

Поле	Тип	Призначення
Id	integer	Унікальний ідентифікатор твердження.
UserId	text	Ідентифікатор облікового запису.
ClaimType	text, NULL	Тип твердження.
ClaimValue	text, NULL	Значення твердження.

Таблиця 24

Опис таблиці AspNetRoleClaims

Поле	Тип	Призначення
Id	integer	Унікальний ідентифікатор твердження.
RoleId	text	Ідентифікатор ролі.
ClaimType	text, NULL	Тип твердження.
ClaimValue	text, NULL	Значення твердження.

Таблиця 25

Опис таблиці AspNetUserLogins

Поле	Тип	Призначення
LoginProvider	text	Назва постачальника входу.
ProviderKey	text	Ідентифікатор користувача у постачальника.
ProviderDisplay Name	text, NULL	Відображувана назва постачальника.
UserId	text	Ідентифікатор облікового запису.

Таблиця 26

Опис таблиці AspNetUserTokens

Поле	Тип	Призначення
UserId	text	Ідентифікатор облікового запису.
LoginProvider	text	Постачальник, якому належить ключ.
Name	text	Назва ключа.
Value	text, NULL	Значення ключа.

Опис таблиці video_report_media

Поле	Тип	Призначення
Id	uuid	Унікальний ідентифікатор запису.
VideoReportId	uuid	Ідентифікатор відеозвіту.
MediaAssetId	uuid	Ідентифікатор прикріпленого файлу.
OrderInReport	integer	Порядок матеріалу у відеозвіті.

1) Dockerfile для контейнеризації серверної частини FitLead

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
```

```
COPY FitLead/FitLead.Common/FitLead.Common.csproj FitLead/FitLead.Common/
COPY FitLead/FitLead.Domain/FitLead.Domain.csproj FitLead/FitLead.Domain/
COPY FitLead/FitLead.Application/FitLead.Application.csproj FitLead/FitLead.Application/
COPY FitLead/FitLead.Infrastructure/FitLead.Infrastructure.csproj FitLead/FitLead.Infrastructure/
COPY FitLead/FitLead.Api/FitLead.Api.csproj FitLead/FitLead.Api/
```

```
RUN dotnet restore FitLead/FitLead.Api/FitLead.Api.csproj
```

```
COPY FitLead/ FitLead/
```

```
RUN dotnet publish FitLead/FitLead.Api/FitLead.Api.csproj \
  --configuration Release \
  --output /app/publish \
  --no-restore
```

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
WORKDIR /app
```

```
ENV ASPNETCORE_URLS=http://+:8080
EXPOSE 8080
```

```
COPY --from=build /app/publish .
```

```
ENTRYPOINT ["dotnet", "FitLead.Api.dll"]
```

2) Dockerfile для контейнеризації клієнтської частини FitLead

```
FROM node:22-alpine AS deps
WORKDIR /app
```

```
COPY package.json package-lock.json ./
RUN npm ci
```

```
FROM node:22-alpine AS build
WORKDIR /app
```

```
ARG NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY
```

```
ENV NEXT_TELEMETRY_DISABLED=1
ENV
```

```
NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY=$NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY
```

```
COPY --from=deps /app/node_modules ./node_modules
COPY . .
```

```
RUN npm run build
```

		Гольцев К.О.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савицький Р.С.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```
FROM node:22-alpine AS runtime
WORKDIR /app

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV HOSTNAME=0.0.0.0
ENV PORT=3000

EXPOSE 3000

COPY --from=build /app/public ./public
COPY --from=build /app/.next/standalone ./
COPY --from=build /app/.next/static ./next/static

CMD ["node", "server.js"]
```

3) Файл CI/CD

```
name: FitLead CI

on:
  pull_request:
    branches:
      - main
      - dev
  push:
    branches:
      - main
  workflow_dispatch:

permissions:
  contents: read
  packages: write

env:
  DOTNET_VERSION: 8.0.x
  NODE_VERSION: 22
  CONFIGURATION: Release
  API_PROJECT: FitLead/FitLead.Api/FitLead.Api.csproj
  TEST_PROJECT: tests/FitLead.IntegrationTests/FitLead.IntegrationTests.csproj
  TEST_RESULTS_DIR: tests/TestResults

concurrency:
  group: ci-${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

jobs:
  backend-ci:
    name: Backend CI
    runs-on: ubuntu-latest

    steps:
```

```

- name: Checkout
  uses: actions/checkout@v4

- name: Setup .NET
  uses: actions/setup-dotnet@v4
  with:
    dotnet-version: ${{ env.DOTNET_VERSION }}

- name: Cache NuGet packages
  uses: actions/cache@v4
  with:
    path: ~/.nuget/packages
    key: nuget-${{ runner.os }}-${{ hashFiles('**/*.csproj') }}
    restore-keys: |
      nuget-${{ runner.os }}-

- name: Restore dependencies
  run: |
    dotnet restore "$API_PROJECT"
    dotnet restore "$TEST_PROJECT"

- name: Build API
  run: dotnet build "$API_PROJECT" --configuration "$CONFIGURATION" --no-restore

- name: Build integration tests
  run: dotnet build "$TEST_PROJECT" --configuration "$CONFIGURATION" --no-restore

- name: Run tests with coverage
  run: |
    dotnet test "$TEST_PROJECT" \
      --configuration "$CONFIGURATION" \
      --no-build \
      --logger "trx;LogFileName=test-results.trx" \
      --collect:"XPlat Code Coverage" \
      --settings tests/coverage.runsettings \
      --results-directory "$TEST_RESULTS_DIR"

- name: Upload test results
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: test-results
    path: ${{ env.TEST_RESULTS_DIR }}
    if-no-files-found: warn

```

frontend-ci:

name: Frontend CI

runs-on: ubuntu-latest

env:

NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY: test-public-key

defaults:

run:

		Гольцев К.О.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савицький Р.С.				109
Змн.	Арк.	№ докум.	Підпис	Дата		

working-directory: client

steps:

- name: Checkout
uses: actions/checkout@v4
- name: Setup Node.js
uses: actions/setup-node@v4
with:
node-version: \${{ env.NODE_VERSION }}
cache: npm
cache-dependency-path: client/package-lock.json
- name: Install dependencies
run: npm ci
- name: Lint frontend
run: npm run lint
- name: Build frontend
run: npm run build

frontend-smoke:

name: Frontend Smoke

runs-on: ubuntu-latest

env:

ASPNETCORE_ENVIRONMENT: Smoke

ASPNETCORE_URLS: http://127.0.0.1:5178

ConnectionStrings__DefaultConnection:

Host=127.0.0.1;Port=5432;Database=fitlead_smoke;Username=postgres;Password=postgres

DemoSeed__Enabled: "false"

SmokeSeed__Enabled: "true"

ClientApp__BaseUrl: http://localhost:3000

Cors__AllowedOrigins__0: http://localhost:3000

Uploadcare__PublicKey: test-public-key

Uploadcare__SecretKey: test-secret-key

OutboxProcessor__Enabled: "false"

Push__Enabled: "false"

API_BASE_URL: http://127.0.0.1:5178

FITLEAD_SMOKE_TEST: "true"

NEXT_PUBLIC_SIGNALR_BASE_URL: http://localhost:5178

NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY: test-public-key

PLAYWRIGHT_BASE_URL: http://localhost:3000

services:

postgres:

image: postgres:16-alpine

env:

POSTGRES_DB: fitlead_smoke

POSTGRES_USER: postgres

POSTGRES_PASSWORD: postgres

ports:

- 5432:5432

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савицький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		110

```
options: >-
--health-cmd "pg_isready -U postgres -d filelead_smoke"
--health-interval 5s
--health-timeout 5s
--health-retries 10
```

steps:

- name: Checkout
uses: actions/checkout@v4
- name: Setup .NET
uses: actions/setup-dotnet@v4
with:
dotnet-version: \${{ env.DOTNET_VERSION }}
- name: Setup Node.js
uses: actions/setup-node@v4
with:
node-version: \${{ env.NODE_VERSION }}
cache: npm
cache-dependency-path: client/package-lock.json
- name: Cache NuGet packages
uses: actions/cache@v4
with:
path: ~/.nuget/packages
key: nuget-\${{ runner.os }}-\${{ hashFiles('**/packages.lock.json', '**/*.csproj') }}
restore-keys: |
nuget-\${{ runner.os }}-
- name: Restore and build API
run: |
dotnet restore "\$API_PROJECT" --locked-mode
dotnet build "\$API_PROJECT" \
--configuration "\$CONFIGURATION" \
--no-restore
- name: Restore local .NET tools
run: dotnet tool restore
- name: Generate smoke JWT signing keys
shell: bash
run: |
openssl genpkey \
-algorithm RSA \
-pkeyopt rsa_keygen_bits:2048 \
-out "\$RUNNER_TEMP/smoke-private.pem"
openssl rsa \
-pubout \
-in "\$RUNNER_TEMP/smoke-private.pem" \
-out "\$RUNNER_TEMP/smoke-public.pem"

{

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савицький Р.С.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

echo "Jwt__RsaPrivateKeyPem<<EOF"
cat "$RUNNER_TEMP/smoke-private.pem"
echo "EOF"
echo "Jwt__RsaPublicKeyPem<<EOF"
cat "$RUNNER_TEMP/smoke-public.pem"
echo "EOF"
} >> "$GITHUB_ENV"

```

- name: Apply EF Core migrations

```

run: |
  dotnet tool run dotnet-ef database update \
    --project FitLead/FitLead.Infrastructure/FitLead.Infrastructure.csproj \
    --startup-project "$API_PROJECT" \
    --connection "$ConnectionStrings__DefaultConnection" \
    --configuration "$CONFIGURATION" \
    --no-build

```

- name: Start API

```

shell: bash
run: |
  nohup dotnet run \
    --project "$API_PROJECT" \
    --configuration "$CONFIGURATION" \
    --no-build \
    --no-restore \
    > "$RUNNER_TEMP/fitlead-api.log" 2>&1 &

```

```

for attempt in {1..60}; do
  if curl --fail --silent http://127.0.0.1:5178/health > /dev/null; then
    exit 0
  fi

```

```

  sleep 1
done

```

```

cat "$RUNNER_TEMP/fitlead-api.log"
exit 1

```

- name: Install frontend dependencies

```

working-directory: client
run: npm ci

```

- name: Install Playwright Chromium

```

working-directory: client
run: npx playwright install --with-deps chromium

```

- name: Build frontend for smoke tests

```

working-directory: client
run: npm run build

```

- name: Run frontend smoke tests

```

working-directory: client
run: npx playwright test --workers=1

```

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савицький Р.С.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```
- name: Upload smoke diagnostics
  if: failure()
  uses: actions/upload-artifact@v4
  with:
    name: frontend-smoke-diagnostics
    path: |
      client/playwright-report
      client/test-results
      ${runner.temp}}/fitlead-api.log
    if-no-files-found: warn
    retention-days: 7
```

```
publish-images:
  name: Publish Docker images
  runs-on: ubuntu-latest
  needs:
    - backend-ci
    - frontend-ci
  if: github.event_name != 'pull_request' && github.ref == 'refs/heads/main'
  env:
    NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY: ${{
vars.NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY }}
```

```
steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Setup Docker Buildx
    uses: docker/setup-buildx-action@v3

  - name: Validate web build variables
    shell: bash
    run: |
      if [ -z "$NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY" ]; then
        echo "::error::Repository variable NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY is
required to build the web image."
        exit 1
      fi

  - name: Prepare GHCR image names
    id: images
    shell: bash
    run: |
      owner_lowercase="${GITHUB_REPOSITORY_OWNER,,}"
      api_image="ghcr.io/${owner_lowercase}/fitlead-api"
      web_image="ghcr.io/${owner_lowercase}/fitlead-web"

      echo "api=${api_image}" >> "$GITHUB_OUTPUT"
      echo "web=${web_image}" >> "$GITHUB_OUTPUT"

  {
```

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савінький Р.С.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

```

echo "### Docker images"
echo "- ${api_image}:latest"
echo "- ${api_image}:${GITHUB_SHA}"
echo "- ${web_image}:latest"
echo "- ${web_image}:${GITHUB_SHA}"
} >> "$GITHUB_STEP_SUMMARY"

- name: Login to GHCR
  uses: docker/login-action@v3
  with:
    registry: ghcr.io
    username: ${github.actor}
    password: ${secrets.GITHUB_TOKEN}

- name: Build and push API image
  uses: docker/build-push-action@v6
  with:
    context: .
    file: FitLead/FitLead.Api/Dockerfile
    push: true
    cache-from: type=gha,scope=fitlead-api
    cache-to: type=gha,mode=max,scope=fitlead-api
    tags: |
      ${steps.images.outputs.api}:latest
      ${steps.images.outputs.api}:${github.sha}

- name: Build and push web image
  uses: docker/build-push-action@v6
  with:
    context: client
    file: client/Dockerfile
    push: true
    cache-from: type=gha,scope=fitlead-web
    cache-to: type=gha,mode=max,scope=fitlead-web
    build-args: |
      NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY=${
env.NEXT_PUBLIC_UPLOADCARE_PUBLIC_KEY}
    tags: |
      ${steps.images.outputs.web}:latest
      ${steps.images.outputs.web}:${github.sha}

```

		Гольцев К.О.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савінький Р.С.				114
Змн.	Арк.	№ докум.	Підпис	Дата		