ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«СИСТЕМА УПРАВЛІННЯ ВЗАЄМОДІЇ З КЛІЄНТАМИ ІЗ ЗАСТОСУВАННЯМ АРХІТЕКТУРИ MVC ТА RESTFUL API»

на здобуття освітнього ступеня бакалавр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Юрій МОРОЗ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСД-41 Юрій МОРОЗ
(ім'я, ПРІЗВИЩЕ)

Керівник Дмитро КОЗЛОВ
к.т.н. *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” ___2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Морозу Юрію Євгеновичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Система управління взаємодії з клієнтами із застосуванням архітектури MVC та RESTful API
керівник кваліфікаційної роботи: Дмитро КОЗЛОВ
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “24” лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
Системи управління взаємодією з клієнтами, що потребують вдосконалення архітектурного підходу для підвищення ефективності бізнес-процесів ті покращення обслуговування клієнтів у фітнес-індустрії

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
2. Аналіз предметної області та постановка задачі.
3. Огляд існуючих рішень та створення вимог до системи.
4. Тестування та оцінка ефективності системи.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз предметної області. Вивчення потреб фітнес-клубів у сфері управління взаємодією з клієнтами.	03.04.2025 р.	
2	Визначення вимог до системи.	10.04.2025 р.	
3	Розробка архітектури на основі MVC та RESTful API. Створення діаграм класів та взаємодії.	21.04.2025 р.	
4	Вибір технологій (Java, Spring Boot). Розробка основних компонентів: контролерів, сервісів і DAO-шару.	29.04.2025 р.	
5	Створення та документування ендпоінтів для управління тренуваннями та профілями користувачів. Впровадження механізмів обробки запитів (GET, POST, PUT, DELETE).	03.05.2025 р.	
6	Проведення модульного тестування ендпоінтів.	8.05.2025 р.	
7	Написання технічної документації для розробленої системи та оформлення користувацької документації для кінцевих користувачів.	15.05.2025 р.	

Здобувач вищої освіти _____ Юрій МОРОЗ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Дмитро КОЗЛОВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 стор., 68 рис., 34 джерела.

Мета роботи: Розробити та реалізувати CRM-систему для відслідковування та бронювання тренувань у фітнес-клубі, що забезпечує зручний вибір типу тренування, тренера та вільного часу користувача через веб-API.

Об'єкт дослідження - процеси організації, управління та автоматизації бронювання тренувань у фітнес-клубі за допомогою інформаційних систем.

Предмет дослідження - методи, засоби та технології проєктування, розробки та впровадження CRM-системи для бронювання тренувань на базі Java, Spring Boot, Spring Security, Spring MVC та СУБД MySQL.

Короткий зміст роботи. У дипломному проекті розглянуто розробку CRM системи для відслідковування та бронювання тренувань для користувачів фітнес клубу. Актуальність теми зумовлена потребою легкої можливості підбору тренування з урахуванням вибору типу тренування, вибору тренера та вибору вільного часу через зручний додаток.

В рамках проекту було проаналізовано предметну область, досліджено існуючі рішення та визначено основні вимоги до системи. Реалізацію системи виконано засобами мови Java із використання фреймворку Spring Boot, Spring Security та Spring MVC, а також база даних MySQL.

Результатом є стабільно працююча Gym CRM API, яка може бути задіяна для покращення роботи з клієнтами для існуючого бізнеса фітнес клубів. Розроблене рішення є дуже ефективним, гнучким та масштабованим, із можливостями для подальшого розвитку та покращення.

Ключові слова:

SPRING, RESTFUL, APPLICATIOIN PROGRAMMING INTERFACE(API), POSTMAN, CRM, SWAGGER, СИСТЕМА, АВТОМАТИЗАЦІЯ, ТРЕНУВАННЯ, КОРИСТУВАЧ, АРХІТЕКТУРА.

ЗМІСТ

ВСТУП

Актуальність дослідження обумовлена постійно зростаючими вимогами до управління бізнес-процесами у сфері фітнес-індустрії. Сучасні фітнес-клуби стикаються з необхідністю ефективного управління взаємодією з клієнтами, що включає не лише бронювання тренувань, а й забезпечення зручних умов для вибору типу тренування, тренера та вільного часу. У зв'язку з цим, розробка системи, яка б відповідала цим вимогам та забезпечувала захищений доступ користувачів, стає надзвичайно важливою.

Мета даної дипломної роботи полягає у розробці та імплементації CRM-системи управління взаємодією з клієнтами фітнес-клубу, яка б відповідала сучасним вимогам ринку.

Предметом дослідження є процеси управління взаємодією з клієнтами у фітнес-клубах, а також технології, що забезпечують ці процеси.

Об'єктом роботи виступає CRM-система, розроблена для автоматизації бізнес-процесів фітнес-клубів, зокрема функцій бронювання, управління профілями користувачів та аналітики.

У межах роботи проведено аналіз предметної області й існуючих рішень, що дозволив виокремити ключові вимоги до системи, такі як зручність користування, масштабованість та надійність. Реалізація бек-енду виконана мовою Java на базі Spring Boot, що забезпечує високу продуктивність, а також модульні компоненти Spring MVC для організації контролерів, сервісів і DAO-шару. Використання Spring Security гарантує аутентифікацію та авторизацію користувачів, що є критично важливим для захисту даних. RESTful API забезпечує гнучку інтеграцію з фронтенд-додатком або мобільними клієнтами, спрощуючи подальше розширення функціоналу системи.

Отримана Gym CRM API демонструє стабільну роботу в базових сценаріях, таких як реєстрація клієнтів, перегляд розкладу, бронювання та скасування тренувань, а також управління профілем і ролями. Завдяки чіткому розділенню

шарів та контрактам REST-інтерфейсів, система легко адаптується під зміни бізнес-процесів.

Перспективи подальшого розвитку включають додавання аналітичних модулів для генерації звітів про відвідуваність і фінансові показники, інтеграцію з платіжними сервісами та впровадження push-повідомлень. Таким чином, створений прототип може бути успішно використаний для автоматизації роботи фітнес-клубів і слугувати основою для масштабних корпоративних рішень. Результати даної роботи можуть суттєво підвищити ефективність бізнес-процесів у сфері фітнесу, що робить її надзвичайно актуальною в умовах сучасного ринку.

1 ОСНОВИ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ ВЗАЄМОДІЄЮ З КЛІЄНТАМИ

1.1 Поняття CRM-систем та їх роль у сучасному бізнесі

Узагальнено CRM (англ. Customer Relationship Management) – це стратегія управління взаєминами з клієнтами, що передбачає збір і аналіз даних на всіх етапах взаємодії з ними. CRM-система – програмний інструмент, який централізовано зберігає інформацію про клієнтів (контактні дані, історію спілкування тощо) та автоматизує процеси продажів, маркетингу та обслуговування. Завдяки цьому компанія може більш глибоко розуміти потреби клієнтів і підтримувати довготривалі відносини з ними. Таким чином, основне призначення CRM – підвищити ефективність взаємодії з клієнтами та збільшити прибутковість бізнесу.

Історія CRM сягає далеко за 1980-ті роки. Процес управління інформацією про клієнтів існував ще в паперову епоху, коли контактні дані фіксувались у фізичних довідниках (наприклад, Rolodex). Поява комп'ютерів і Інтернету докорінно змінила ситуацію: до 1980-х рр. з'явився термін CRM, але фактично перші електронні системи для обліку клієнтів існували з 1950–60-х рр. Спочатку CRM-системи були переважно засобом обліку транзакцій і контактів. Проте з розвитком технологій функціонал розширювався: у 1990-х впроваджувалась автоматизація відділів продажів (Sales Force Automation), а до початку 2000-х з'явились хмарні та відкриті CRM-рішення. Сьогодні CRM поєднують продажі, маркетинг і підтримку клієнтів в єдиній екосистемі, забезпечуючи персоналізовану взаємодію з клієнтом і багатоканальний підхід.

Класифікація CRM-систем [4].

CRM-системи зазвичай розрізняють за основним призначенням. Традиційно виділяють три типи: операційні, аналітичні та колаборативні CRM.

Операційні CRM орієнтовані на автоматизацію повсякденної роботи: вони підтримують процеси лідогенерації, автоматизації маркетингу, продажів та

обслуговування клієнтів. Завдяки цьому прискорюється обробка заявок та підвищується продуктивність відділів продажів і маркетингу.

Аналітичні CRM зосереджені на зборі та аналізі даних про клієнтів і операції компанії. Вони накопичують інформацію з усіх джерел (продажі, маркетинг, підтримка) і надають інструменти для аналітики, звітності та прогнозування. Це дозволяє виявити слабкі місця в бізнесі, оцінити ефективність кампаній та покращити обслуговування клієнтів.

Колаборативні (спільні) CRM сприяють обміну інформацією між відділами та каналами комунікації. Вони забезпечують єдиний «погляд» на клієнта, поєднуючи дані з продажів, маркетингу та служби підтримки. Завдання колаборативної CRM – уніфікувати робочі процеси, щоб різні команди могли швидко передавати ліди та повну історію взаємодії. Нові класифікації також включають CRM для управління кампаніями та стратегічні CRM, але всі сучасні системи часто поєднують елементи всіх типів.

Функції та можливості сучасних CRM-систем

Сучасні CRM-системи містять низку модулів і функцій, що охоплюють ключові бізнес-процеси. До основних можливостей належать:

Управління контактами: централізоване збереження даних про клієнтів (контактна інформація, історія комунікацій) у єдиній базі. Це дозволяє легко знаходити інформацію про кожного клієнта та швидко отримувати повну картину його взаємин з компанією.

Автоматизація продажів: відстеження лідів і угод, планування завдань менеджерів, прогнозування продажів і робота з воронкою збуту. CRM допомагає контролювати всі етапи угоди – від першого контакту до закриття – і генерувати нагадування про подальші дії для підвищення конверсії.

Управління маркетингом: інструменти планування та проведення маркетингових кампаній, сегментації аудиторії та оцінки ефективності рекламних активностей. CRM-системи дозволяють запускати email-розсилки, таргетувати рекламу на певні групи клієнтів і вимірювати результати в реальному часі.

Підтримка клієнтів: обробка звернень через інтегровані канали (чат-боти, тікет-системи, кол-центри) і відстеження історії звернень. Модуль сервісу збирає всі запити клієнтів, спрямовуючи їх фахівцям служби підтримки, що підвищує якість обслуговування і швидкість реакції.

Аналітика та звітність: CRM генерує статистичні звіти про продажі, маркетингові кампанії, продуктивність відділів та задоволеність клієнтів. Звіти формуються на підставі зібраних даних і допомагають керівництву оцінити ефективність роботи, виявити тенденції й оптимізувати процеси. Завдяки аналітиці бізнес може ухвалювати обґрунтовані рішення та покращувати показники своєї діяльності [18].

Переваги впровадження CRM у компаніях різного масштабу

Впровадження CRM-систем приносить суттєві переваги компаніям будь-якого розміру. **Централізація даних** про клієнтів дозволяє комплексно аналізувати інформацію та краще розуміти потреби цільової аудиторії. Автоматизація бізнес-процесів скорочує час на рутинні операції: менеджери можуть швидше обробляти замовлення й обдзвонювати лідів, що підвищує продуктивність відділів продажів. Інтеграція маркетингових кампаній із CRM забезпечує точніше таргетування — правильна сегментація аудиторії зменшує витрати на рекламу і збільшує конверсію. Крім того, підтримка клієнтів стає ефективнішою: централізоване зберігання історії звернень дає змогу надавати швидку й персоналізовану допомогу. В результаті компанії отримують зростання продажів і прибутку: наприклад, дослідження вказують на +29% до загального обсягу продажів у компаній, що використовують CRM. Аналіз ROI показує, що в середньому на кожен вкладений долар у CRM повертається приблизно $8.71. CRM-системи також підвищують утримання клієнтів — за даними TrackVia, застосування CRM може збільшити коефіцієнт лояльності на 27%.

При цьому масштабність бізнесу впливає на вибір конкретного рішення. Для малого та середнього бізнесу популярні **хмарні CRM-рішення** (SaaS), що дозволяють швидко запустити систему без великих капіталовкладень і з мінімальним IT-підтримкою. Такі рішення зазвичай надають основний набір

функцій «з коробки» і оплати у форматі підписки. Великі корпорації зазвичай обирають більш гнучкі та кастомізовані CRM: вони інтегрують їх із існуючою ERP/BI-інфраструктурою та налаштовують під специфічні внутрішні процеси. Згідно зі статистикою, CRM-системи вже майже повсюдно використовують великі компанії: приблизно 91% організацій із понад десятьма працівниками застосовують CRM для управління відносинами з клієнтами. Для цих фірм CRM стає ключовим інструментом координації маркетингу, продажів і підтримки на глобальному рівні [1].

Впровадження CRM-системи сприяє загальному підвищенню ефективності бізнес-процесів. Автоматизація обробки заявок, своєчасне формування звітів та інтерактивні панелі керування (дашборди) дозволяють керівництву швидко реагувати на зміни ринку і коригувати стратегії. Так, за даними Salesforce, впровадження CRM підвищує точність прогнозування продажів у середньому на 42%, що допомагає компаніям краще планувати ресурси та досягати встановлених цілей. Швидка та персоніфікована реакція на запити клієнтів підвищує рівень їхньої задоволеності й лояльності, а це критично важливо в умовах конкуренції. Загалом, CRM надає компаніям конкурентні переваги: вона дозволяє швидше відстежувати важливі метрики (продажі, прибутковість, ефективність маркетингу), приймати обґрунтовані рішення і будувати цінність клієнтського досвіду. Дослідження підтверджують: користувачі CRM відзначають зростання продажів і маркетингової віддачі, а в багатьох випадках — збільшення рентабельності та утримання клієнтів. Завдяки цьому компанія починає діяти більш проактивно й гнучко, що робить її більш конкурентоспроможною на ринку [1].

Сьогодні CRM-системи швидко розвиваються з урахуванням новітніх технологій. **Хмарні CRM-рішення** продовжують домінувати на ринку (близько 87% усіх впроваджень припадає на cloud-сервіси), оскільки вони забезпечують легке масштабування та доступ до інформації з будь-яких пристроїв. **Мобільність** стає стандартом: CRM-системи пропонують мобільні додатки для iOS та Android,

що дозволяють менеджерам бути на зв’язку у дорозі. Вони підтримують аналітику «на ходу» і навіть голосове введення даних. **Штучний інтелект** (AI) інтегрується в CRM для підвищення автоматизації й персоналізації. Застосування AI у CRM включає аналіз великих масивів даних, оцінку поведінки клієнтів, автоматичне генерування рекомендацій і звітів. За результатами опитувань, більшість підприємців відзначають, що AI дає бізнесу можливості масштабування, які раніше були недоступні. Паралельно зростає увага до **омніканальності**: CRM-інструменти об’єднують комунікацію через електронну пошту, соцмережі, месенджери, чат-боти та інші канали, створюючи єдиний профіль клієнта. Також формуються тренди на використання передбачуваної аналітики, автоматизованих чат-ботів та емоційного аналізу взаємодій, що робить CRM більш інтелектуальним і орієнтованим на клієнта. З огляду на ці тенденції, CRM-системи перетворюються на стратегічний інструмент, який допомагає бізнесу швидко адаптуватися до ринкових змін і посилювати конкурентну позицію [19].

1.2 Архітектурні підходи до розробки CRM-систем

Побудова ефективної CRM-системи неможлива без виваженого вибору архітектурного підходу. Архітектура визначає, як компоненти програми взаємодіятимуть один з одним, як масштабуватиметься рішення та наскільки легко в нього можна буде додавати новий функціонал. Огляд основних підходів допомагає обрати ту модель, яка найкраще відповідатиме бізнес-вимогам, обсягам даних та технічним можливостям організації.

Традиційним рішенням для бізнес-додатків є шарова (або багатошарова) архітектура. Зазвичай вона включає три базові рівні:

1. **Presentation Layer (UI-шар)** — відповідає за взаємодію з кінцевим користувачем (веб-інтерфейс, мобільний додаток).

2. **Business Logic Layer (сервісний шар)** — містить бізнес-правила, обробку даних, валідацію й координацію операцій.

3. **Data Access Layer (шар доступу до даних)** — реалізує взаємодію з базою даних, ORM-механізми, персистенцію об’єктів.

Переваги шарової архітектури — чіткий поділ обов’язків, легкість тестування окремих модулів та відокремлення від впроваджених технологій. Недолік — у великих додатках усі шари можуть зростати монолітно, складність підтримки з часом зростає [2].

1.2.1 Архітектура Model–View–Controller (MVC)

MVC є спеціалізованим варіантом шарової моделі, який виділяє три ключові компоненти:

- **Model** — модель предметної області, яка інкапсулює дані та логіку їх обробки.
- **View** — відображення даних (HTML, JSON, XML).
- **Controller** — приймає HTTP-запити, викликає методи сервісного шару та віддає користувачеві готовий вигляд даних.

У контексті CRM MVC дозволяє розділити інтерфейс користувача від бізнес-логіки, що спрощує розробку комплексних форм управління контактами, угодами та звітністю. Багато фреймворків (наприклад, Spring MVC, ASP .NET MVC, Ruby on Rails) надають готові механізми маршрутизації, ін’єкції залежностей і прив’язки даних, що пришвидшує старт проєкту.

Зі зростанням розподілених систем популярність набуває підхід із розбиттям CRM на незалежні сервіси (мікросервіси), які комунікують через RESTful API. Кожен сервіс виконує окрему роль: управління користувачами, обробка лідів, аналітика, розсилка повідомлень тощо.

REST (Representational State Transfer) задає набір обмежень: ресурси доступні за унікальними URL, взаємодія відбувається через стандартні HTTP-методи (GET, POST, PUT, DELETE), а повідомлення несуть гіпермедіа-посилання [33].

Мікросервіси забезпечують гнучке масштабування: можна окремо збільшувати кількість екземплярів «сервісу маркетингу» без впливу на «сервіс обробки платежів».

Переваги REST-мікросервісної архітектури:

- **Незалежний розгортання** — кожен сервіс можна оновлювати, не зупиняючи всю платформу.
- **Гетерогенність технологій** — сервіси можуть бути написані на різних мовах і використовувати різні СУБД.
- **Стійкість до відмов** — збій одного сервісу не обов’язково виведе з ладу всю систему.

Водночас недоліки: підвищена складність оркестрації, необхідність налаштування API-Gateway та обробки транзакцій на межах сервісів.

1.2.2 Поділена за доменами архітектура (Domain-Driven Design)

Підхід Domain-Driven Design (DDD) фокусується на виділенні окремих предметних областей (доменів) і побудові «контекстів» (Bounded Context), у межах яких застосовуються власні моделі даних і термінологія. Для CRM це може означати розділення на такі домени, як «Управління клієнтами», «Продажі», «Маркетинг» та «Підтримка клієнтів». Кожен домен містить агрегати, сутності та сервіси, що спрощує підтримку обширних бізнес-правил і дозволяє реалізовувати складні бізнес-процеси в єдиній системі з чіткими кордонами відповідальності [32].

1.2.3 Поділена за подіями архітектура (Event-Driven Architecture)

У CRM-платформах дедалі ширше застосовується event-driven підхід: зміну стану сутності (наприклад, створення нового ліда, бронювання тренування або успішне завершення продажу) надсилають у вигляді повідомлення (події) до шини подій або черги (Kafka, RabbitMQ). Інші компоненти підписуються на

потрібні їм типи подій і реагують відповідно: оновлюють аналітичні звіти, надсилають email, запускають робочі процеси [34].

Переваги EDA:

- Асинхронна обробка завдань (зменшується затримка в основних потоках).
- Висока розширюваність завдяки додаванню нових підписників на ті самі події.

1.2.4 Вибір архітектури під потреби фітнес-CRM

Для системи управління бронюванням тренувань фітнес-клубу доцільно комбінувати MVC на рівні фронтенда (наприклад, для адмін-панелі та користувацького порталу), шарову серверну структуру та RESTful API для мобільних і веб-клієнтів. Усередині API можна окремо виділити сервіси для реєстрації, керування розкладом, моніторингу завантаженості тренерів та аналітики. Використання DDD допоможе чітко розмежувати функціональні зони (бронювання, профілі користувачів, оплати), а event-driven паттерн полегшить імплементацію сповіщень та інтеграцію з push-сервісами або зовнішніми платіжними шлюзами.

Таким чином, оптимальною стане **гібридна архітектура**, що поєднує переваги шарового поділу, MVC-контролерів, REST-мікросервісів та event-driven взаємодії. Такий підхід гарантує високу гнучкість, масштабованість і стійкість до навантажень, відповідаючи потребам сучасного фітнес-бізнесу.

1.3 Технологічні засоби для реалізації CRM (Java, Spring Boot, MySQL)

У сучасних корпоративних рішеннях для побудови масштабованих, надійних і гнучких CRM-систем найчастіше обирають стек технологій на базі мови програмування Java, фреймворку Spring Boot і реляційної СУБД MySQL. Такий набір забезпечує високий рівень платформної незалежності, велику спільноту розробників та багату екосистему готових модулів [10].

Java

Java продовжує залишатися однією з провідних мов для розробки серверних бізнес-додатків завдяки своїй портативності (JVM), стабільності та продуктивності під багатонитковими навантаженнями. Завдяки принципам об’єктно-орієнтованого програмування вона дозволяє чітко моделювати предметну область CRM: клієнтів, угоди, тренування, ролі користувачів. Наявність стандартних специфікацій (JDBC, JPA, Servlets) і вбудована Garbage Collection сприяють підтримці якості коду та зниженню ризиків витоків пам’яті.

Spring Boot - є основою швидкого старту проєктів завдяки авто-налаштуванню залежностей і мінімальній конфігурації. Він дозволяє структуровано впроваджувати архітектуру MVC за допомогою модуля Spring MVC, керувати бізнес-логікою в сервісному шарі, а також ізолювати доступ до даних через Spring Data. Додаткові модулі екосистеми Spring значно розширюють функціонал:

– **Spring Data JPA** спрощує роботу з персистентністю, генерує SQL-запити на основі імен методів репозиторіїв та підтримує лениве або жорстке завантаження сутностей.

– **Spring Security** забезпечує гнучку систему автентифікації та авторизації, дозволяючи легко інтегрувати реєстрацію, роботу з ролями й правами доступу.

– **Spring Actuator** надає готові ендпоінти для моніторингу стану програми, що особливо корисно в умовах продакшен-серверів і при налаштуванні CI/CD.

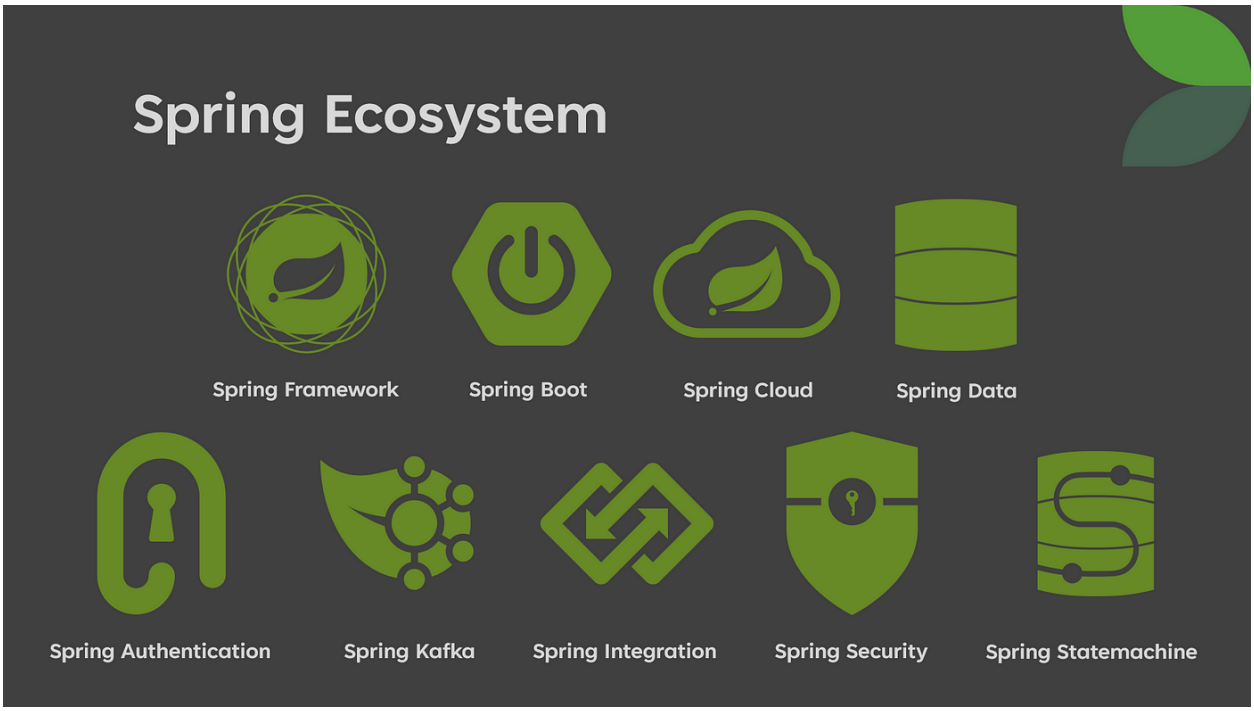


Рисунок 1.1 – Екосистема фреймворку Spring

Використання **Maven** або **Gradle** як системи управління збіркою дозволяє автоматизувати процес отримання залежностей, компіляції, запуску юніт-тестів та формування артефактів (JAR/WAR), що пришвидшує цикл розробки та забезпечує відтворюваність середовища.

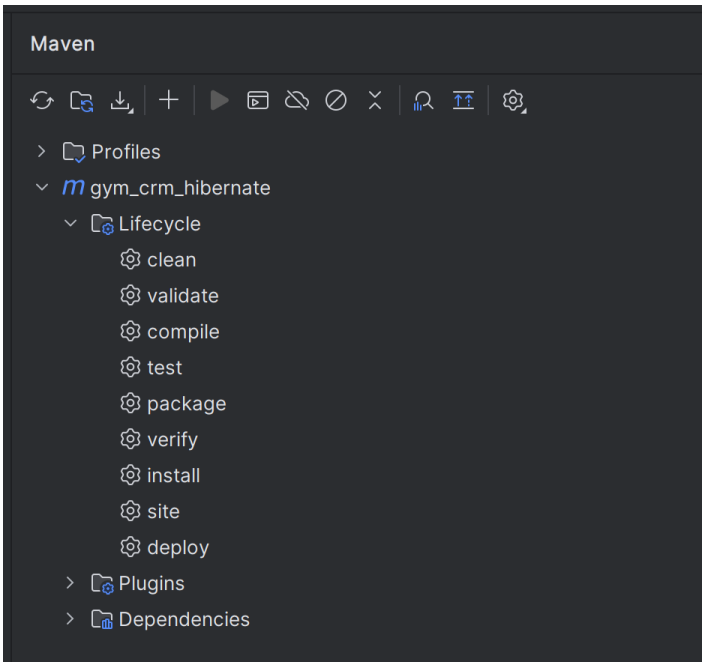


Рисунок 1.2 – Maven lifecycle

У ролі реляційної СУБД MySQL в CRM-системах затребувана завдяки високій продуктивності при обробці транзакцій, широким можливостям

налаштування індексів та відмовостійким механізмам (InnoDB, реплікація, групові репліки). Простота адміністрування, велика кількість інструментів для бекапу й моніторингу (MySQL Workbench) робить її зручною для бізнесів різного масштабу. Стандартний тонкий драйвер JDBC або реалізація через Spring Data JPA забезпечує безшовне з'єднання з Java-додатком.

Рисунок 1.3 – MySQL logo

Для підвищення ефективності командної роботи та якості коду застосовують IDE (IntelliJ IDEA, Eclipse), системи контролю версій (Git), інструменти CI/CD (Jenkins, GitLab CI) і контейнери (Docker) для уніфікації середовища розробки та розгортання. Використання Swagger/OpenAPI дає змогу автоматично документувати REST-ендпоінти та перевіряти коректність запитів у процесі тестування.

Усі перераховані компоненти утворюють інтегрований стек, який забезпечує швидке прототипування, гнучкість розширення функціоналу та простоту обслуговування. Завдяки модульній структурі та великій спільноті підтримки, рішення на базі Java, Spring Boot і MySQL легко адаптуються під специфіку фітнес-клубу та вимоги CRM-системи з бронювання тренувань.

1.4 Приклади застосування CRM у фітнес-бізнесі

У фітнес-індустрії CRM-системи перетворилися з додаткового інструмента на незамінний елемент клієнт-орієнтованого сервісу. Провідні мережі, як-от Total Fitness та Sport Life, успішно впровадили CRM для побудови довгострокових відносин із абонентами, оптимізації внутрішніх процесів і підвищення лояльності.

Total Fitness

Сегментація та персоналізація пропозицій. Завдяки зібраним даним про частоту відвідувань, улюблені напрямки тренувань та історію покупок обладнання, Total Fitness автоматизувала відправку індивідуальних акцій та бонусів. Наприклад, клієнтам, які відвідують групові заняття з йоги більше двох разів на тиждень, надходять спеціальні пропозиції на майстер-класи з розтяжки чи медитації [7].

Лояльні програми і накопичення балів. В CRM реалізовано механізми нарахування балів за кожну покупку абонемента чи персонального тренування. Накопичені бали можна використати на знижки в магазині спортивного харчування або на участь у закритих подіях клубу. Система автоматично сповіщає користувача про досягнення нових рівнів лояльності та надає доступ до привілейованих послуг.

Аналітика та прогнозування потоків. Інтеграція CRM із BI-модулем дозволяє Total Fitness прогнозувати навантаження у залах і підлаштовувати розклад групових занять. Це сприяє оптимальному розподілу тренерів та обладнання, зменшуючи черги й підвищуючи рівень задоволеності клієнтів.

Sport Life

Мультиканальна комунікація. CRM-платформа Sport Life об’єднує SMS-розсилки, email-маркетинг, пуш-повідомлення в мобільному додатку та соціальні мережі. Завдяки цьому учасники отримують нагадування про тренування, зміни в розкладі та ексклюзивні пропозиції через зручний для них канал [6].

Онлайн-бронювання та самообслуговування. Через інтеграцію CRM із веб-інтерфейсом та мобільним додатком клієнти Sport Life самостійно обирають дати й час тренувань, бронюють вільні місця на групові заняття або записуються на консультації з тренером. Вся інформація про бронювання синхронізується в реальному часі, що мінімізує ймовірність дублювання або помилок.

Моніторинг задоволеності клієнтів. Після кожного із тренувань CRM автоматично надсилає клієнту коротке опитування про якість занять і рівень обслуговування.

Практика Total Fitness і Sport Life демонструє, як CRM-системи в фітнес-сегменті забезпечують:

1. **Підвищення утримання клієнтів** завдяки персоналізованим комунікаціям і програмам лояльності.

2. **Оптимізацію операційних процесів** через автоматизацію бронювання та аналітику завантаженості.

3. **Зростання доходів** за рахунок крос-продажів супутніх послуг (магазин, майстер-класи) і більш точного таргетування акцій.

Ці приклади свідчать, що правильно налаштована CRM-система стає фундаментом для ефективного управління бізнесом фітнес-клубу, дозволяючи компаніям оперативно адаптуватися до потреб ринку та підвищувати конкурентоспроможність [10].

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Характеристика предметної області (фітнес-клуб, бронювання тренувань)

Предметна область дослідження охоплює діяльність фітнес-клубів, спрямовану на організацію, планування та контроль спортивно-оздоровчих послуг із функцією онлайн-бронювання тренувань. Фітнес-клуб уявляється як комплексна екосистема, до якої входять: адміністрація закладу, тренери з різних напрямів (кардіо, силові, групові заняття), інфраструктура (тренажерні зали, студії для групових практик, зона відпочинку), а також сама клієнтська база (як індивідуальні відвідувачі, так і корпоративні абоненти).

Ключовими елементами процесу бронювання в цій галузі є:

Ресурсний простір: кожне тренування проводиться у певному залі або студії, де можуть одночасно займатися обмежена кількість людей; обладнання тренажерного залу також має бути доступним у зазначений час.

Розклад і час: тренери працюють за заздалегідь сформованим графіком, що включає регулярні групові заняття та індивідуальні сесії за попереднім записом; система бронювання повинна враховувати обидва типи з метою оптимального завантаження співробітників і приміщень.

Ролі користувачів:

- **Адміністратор** відповідає за створення розкладу, управління списками тренерів та призначенням абонементів;
- **Тренер** має можливість переглядати свої заняття, підтверджувати або відхиляти індивідуальні бронювання та залишати відгуки про відвідувачів;
- **Клієнт** реєструється в системі, переглядає доступні тренування (за фільтрами “тип заняття”, “час”, “тренер”), бронює місце та може скасовувати або переносити бронювання відповідно до умов абонементу.

Система повинна забезпечувати облік різних типів абонементів (одноразовий квиток, місячний абонемент, пакет індивідуальних занять) і відповідно валідувати

право бронювання. При цьому важливим є механізм блокування “перетягування” клієнтів, коли більше, ніж дозволено, осіб намагаються зарезервувати місце на одній сесії, а також гнучка політика скасування (наприклад, не пізніше ніж за 2 години до початку).

У загальному вигляді діяльність фітнес-клубу з бронювання тренувань реалізується через такі бізнес-процеси:

– Управління контингентом тренерів (реєстрація, редагування кваліфікації, графіків).

– Планування розкладу (створення та коригування групових занять).

– Реєстрація та авторизація клієнтів (прив’язка абонементу до облікового запису).

– Онлайн-бронювання (відображення вільних слотів, підтвердження).

– Механізм сповіщень (email/SMS/Push про підтвердження, скасування або нагадування).

– Аналітика завантаженості (звітність адміністратора про відвідуваність, завантаження залів і ефективність тренерів).

Чітке розуміння предметної області дозволяє сформулювати вимоги до CRM-системи, спрямованої на автоматизацію бронювання тренувань, та побудувати архітектуру, що забезпечить баланс між гнучкістю, надійністю і зручністю для всіх учасників процесу [3].

2.2 Аналіз існуючих рішень та сервісів

Для побудови конкурентоспроможної CRM-системи з функціями бронювання тренувань важливо дослідити та порівняти наявні на ринку рішення. Нижче розглянуто кілька популярних платформ, їх функціональні особливості та обмеження.

Mindbody

Опис і ключові можливості. Mindbody — один із лідерів світового ринку фітнес-CRM. Забезпечує управління розкладами, онлайн-оплатою,

маркетинговими розсилками та аналітикою. Система надає мобільний додаток для клієнтів із можливістю бронювання, а також мобільний застосунок для тренерів із календарем та списком клієнтів.

Переваги. Інтуїтивно зрозумілий інтерфейс, висока ступінь автоматизації маркетингу (email- і SMS-кампанії), вбудована підтримка багатофілійності.

Недоліки. Досить висока вартість підписки для малих і середніх клубів; складна налаштовуваність бізнес-логіки поза межами стандартного функціоналу; відчутні комісії за платіжні транзакції.

Glofox

Опис і ключові можливості. Glofox фокусується на студіях і невеликих фітнес-клубах. Забезпечує функції бронювання групових і приватних занять, облік абонементів, інтегровану систему лояльності та просту панель адміністратора. Має власні мобільні додатки як для кінцевих користувачів, так і для персоналу.

Переваги. Гнучкі шаблони абонементів і тарифних планів, швидкий впроваджувальний процес, прозора структура оплати.

Недоліки. Обмежені можливості кастомізації звітів та інтерфейсу; відсутність глибокої аналітики продажів порівняно з «важковаговиками» ринку.

Zen Planner

Опис і ключові можливості. Платформа, що поєднує CRM, облік відвідуваності та функції фінансової звітності. Окремий акцент — на управлінні членством і автоматизації збору платежів. Має мультифункціональні інформаційні панелі та можна підключити сторонні додатки через REST API.

Переваги. Вбудований процесинг платежів, можливість тонкого налаштування ролей користувачів, потужна система інтеґрацій.

Недоліки. Потребує кваліфікованої IT-підтримки на старті; інтерфейс адміністратора здається перевантаженим для малої студії.

FitSW

Опис і ключові можливості. Орієнтована насамперед на персональні тренери й невеликі фітнес-студії. Пропонує базовий набір інструментів: графіки тренувань, відстеження прогресу клієнтів, повідомлення, управління платежами.

Переваги. Низький поріг входу: безкоштовний тариф із базовим функціоналом; легкий перехід на платні версії.

Недоліки. Мінімальна набірність бізнес-модулів: немає корпоративного обліку, складні інтеграції з іншими системами, обмежена звітність.

Локальні сервіси (Україна):

SportMaster CRM — рішення для регіональних мереж із українським інтерфейсом, підтримкою локальних платіжних шлюзів і SMS-операторами. Пропонує інструменти для організації групових занять, управління тренерським складом, автоматизовані сповіщення клієнтів.

FitBooking.UA — стартап, що спеціалізується на агрегуванні пропозицій різних клубів та студій. Клієнти можуть одночасно бронювати сесії в кількох клубах, а керівники закладів отримують централізовану CRM-панель.

Загальні висновки аналізу:

Галузева спеціалізація. Більшість західних рішень мають універсальний характер, але надто потужний функціонал може виявитися надмірним для клубів малого та середнього масштабу.

Вартість та підписка. Для малих студій критично важлива низька вартість впровадження й простота налаштувань, оскільки складні CRM-проєкти потребують залучення IT-фахівців і значних початкових інвестицій.

Інтеграційні можливості. Наявність відкритого API є ключовим критерієм для подальшого масштабування: це дозволяє під’єднувати мобільні додатки, платіжні шлюзи, BI-системи та інші сервіси без кардинальної переробки базового рішення.

Локальні особливості. Врахування національних стандартів комунікацій (SMS-оповіщення українською мовою, локальні платіжні методи) забезпечує кращу адаптацію на регіональному ринку.

Рівень кастомізації. Ідеальне рішення поєднує готові шаблони бізнес-процесів і можливість тонких налаштувань під специфічні вимоги закладу.

Отже, на підставі аналізу існуючих продуктів, для створення власної фітнес-орієнтованої CRM-системи доцільно передбачити:

1. Модульну архітектуру з відкритим REST API.
2. Підтримку мультифілійності та гнучке налаштування абонементних тарифів.
3. Інструменти автоматизованих маркетингових розсилок і SMS-повідомлень.
4. Адміністративну панель із простою візуалізацією ключових показників (завантаженість, продажі, лояльність).
5. Локалізацію інтерфейсу й інтеграцію з українськими платіжними та телеком-операторами.

Такий підхід забезпечить баланс між готовим функціоналом і можливостями подальшого розширення під потреби фітнес-клубу.

2.3 Постановка задачі розробки CRM-системи та визначення функціональних і нефункціональних вимог

На основі аналізу предметної області та існуючих рішень формулюються завдання, які має вирішувати розроблювана CRM-система для фітнес-клубу. Головною метою проєкту є створення програмного продукту, що автоматизує процеси взаємодії з клієнтами, бронювання тренувань та управління абонементами, сприяючи підвищенню ефективності роботи клубу та задоволеності відвідувачів.

Завдання функціонального рівня:

– **Управління акаунтами користувачів.** Забезпечити реєстрацію, верифікацію та керування ролями (адміністратор, тренер, клієнт) із можливістю відновлення паролю та налаштування персонального профілю.

– **Організація розкладу занять.** Розробити інтерфейси для створення та редагування графіків групових і індивідуальних тренувань із прив’язкою до конкретних залів, тренерів та типів занять.

– **Механізм бронювання.** Реалізувати процедуру резервування місць: від вибору слоту в календарі до підтвердження через систему сповіщень; передбачити умови скасування та перенесення бронювання.

– **Облік абонементів і оплат.** Ввести логіку перевірки дійсності абонементу, автоматичний лічильник залишкових занять та інтеграцію з платіжним модулем для фіксації транзакцій.

– **Сповіщення та нагадування.** Налаштувати автоматичну розсилку email/SMS/Push-повідомлень про нові бронювання, нагадування за декілька годин до заняття та інформування про зміни в розкладі.

Завдання нефункціонального рівня:

1. **Навантажувальна здатність.** Гарантувати коректну роботу системи при одночасному навантаженні до 500 активних користувачів, із мінімальним часом відповіді (<300 мс) на API-запити.

2. **Безпека та відмовостійкість.** Забезпечити надійний захист персональних даних (шифрування паролів, HTTPS, рольова модель доступу), а також механізми резервного копіювання бази даних.

3. **Масштабованість.** Спроектувати систему так, щоб додавання нових модулів (аналітика, маркетингові кампанії) та інтеграція з зовнішніми сервісами виконувались без значних переробок існуючого коду.

4. **Юзабіліті.** Створити інтуїтивний інтерфейс адміністрування та клієнтський портал із адаптивним дизайном, що полегшить роботу співробітників та клієнтів на різних пристроях.

Обмеження та умови реалізації:

1. Використання технологічного стека Java 11+, Spring Boot, Spring Security, Spring Data JPA і MySQL.
2. Побудова RESTful API з підтримкою JSON і версіонуванням ендпоінтів.

3. Архітектурний підхід — шарова структура із застосуванням MVC та можливістю розгортання в контейнерах Docker.

4. Стандарт розробки — код має бути покритий юніт- та інтеграційними тестами із загальним рівнем покриття не менше 70 %.

Таким чином, конкретизація функціональних і нефункціональних завдань разом із технічними обмеженнями формує чітке технічне завдання, що стане орієнтиром на всіх етапах життєвого циклу проєкту — від проєктування до впровадження та супроводу.

3 РОЗРОБКА CRM-СИСТЕМИ З ВИКОРИСТАННЯМ MVC ТА RESTful API

3.1 Архітектура розробленої системи

Розроблена CRM-система побудована за модульно-шаровим підходом із використанням архітектури MVC, доповненим виділенням окремих доменних модулів. Це дозволяє чітко розділити відповідальність, спростити тестування й подальшу еволюцію рішення.

Загальна схема компонентів:

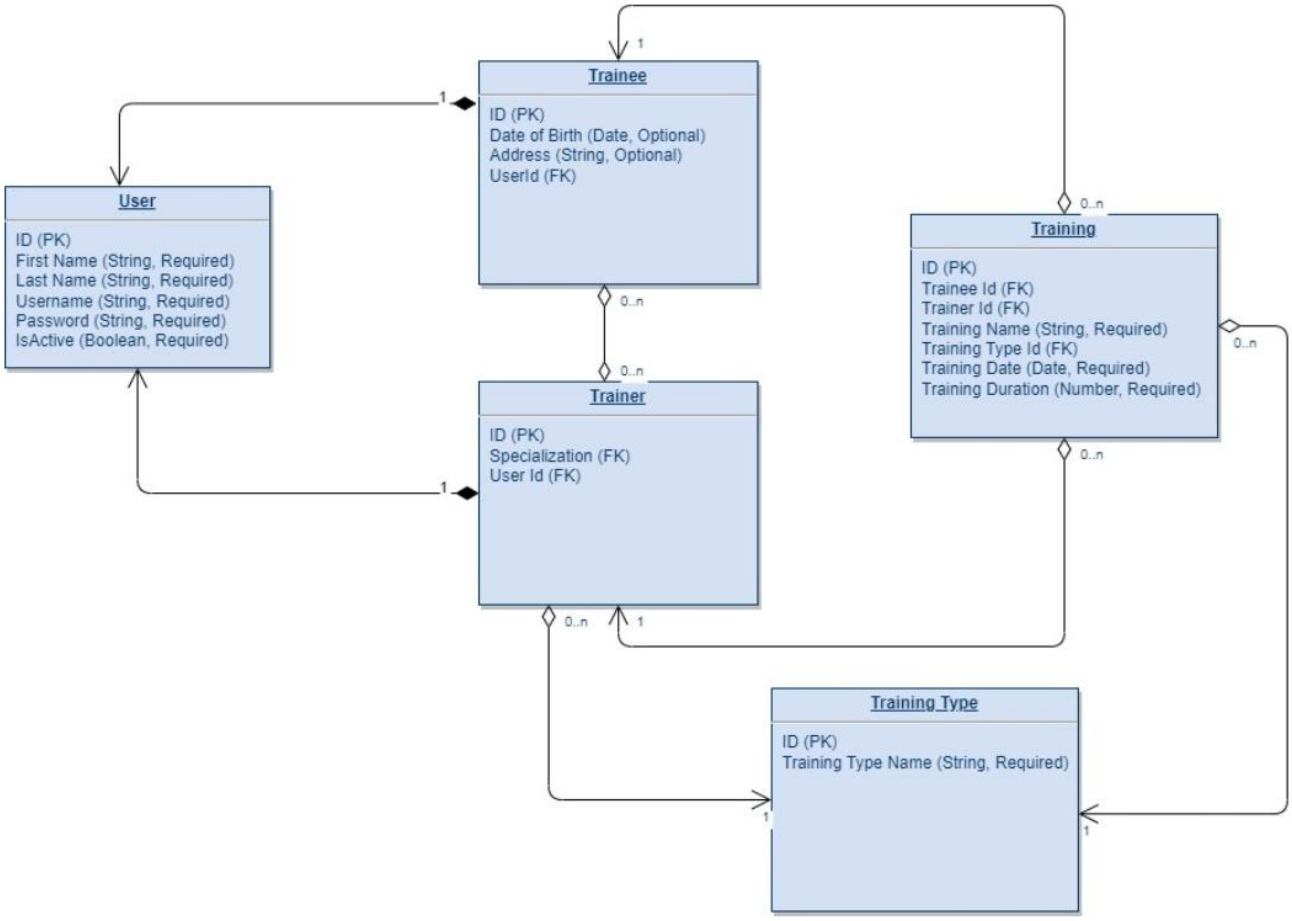


Рисунок 3.1 – діаграма класів-моделей

- Контролерний шар (Controller Layer)
  - REST-контролери (@RestController) обробляють HTTP-запити (див. рисунок 3.2)

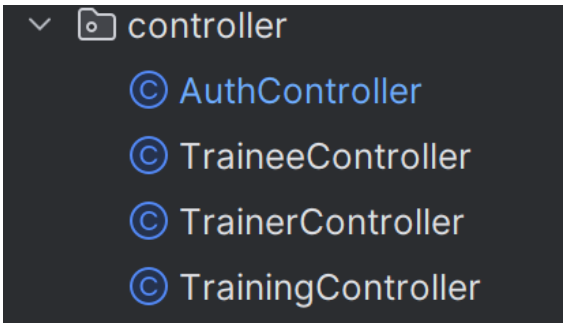


Рисунок 3.2 – пакет контролерів

- Маршрутизація запитів ґрунтується на версіонуванні URL (/api/v1/...)
- Перетворення вхідних JSON у DTO-об’єкти (див. рисунок 3.3)

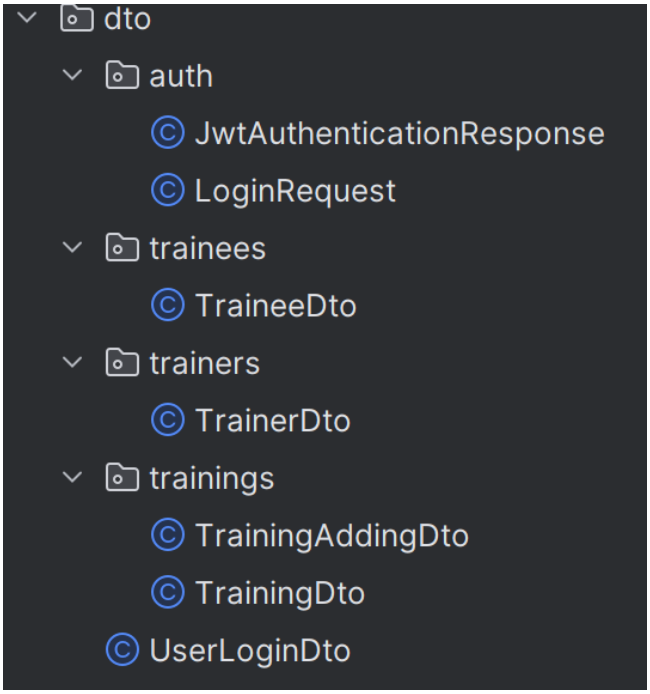


Рисунок 3.3 – структура DTO пакета

– Сервісний шар (Service Layer)

– Доменні сервіси (@Service): реалізують бізнес-логіку кожного модуля(див. рисунок 3.4)

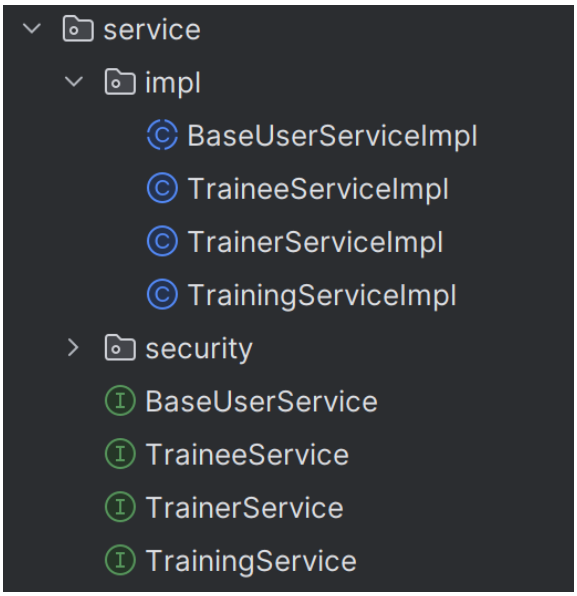


Рисунок 3.4 – пакет сервісів

– Простір назв коду поділено за областями відповідальності на пакети (див. рисунок 3.5)

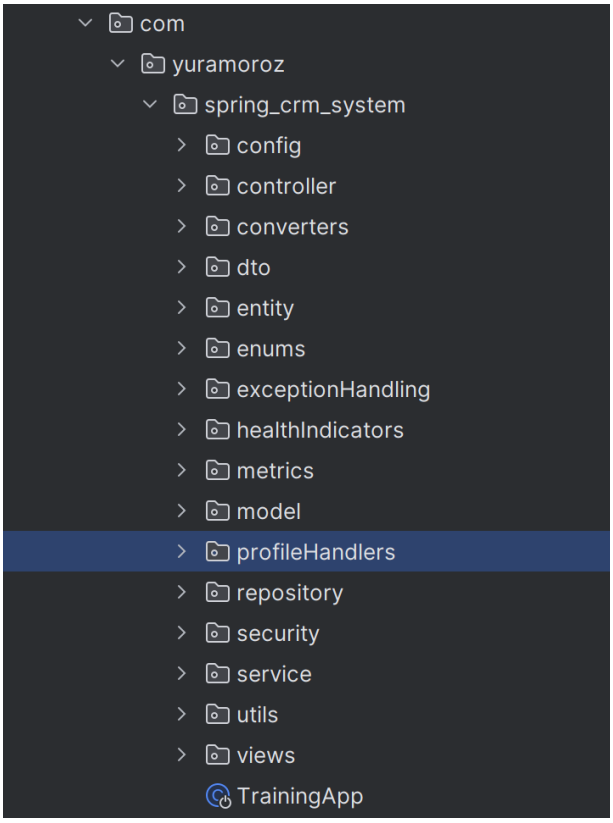


Рисунок 3.5 – структура пакетів додатку

– Шар доступу до даних (Data Access Layer)

– Repository-інтерфейси (Spring Data JPA) для кожної сутності: TraineeRepository, TrainerRepository, TrainingRepository, (див. рисунок 3.6):

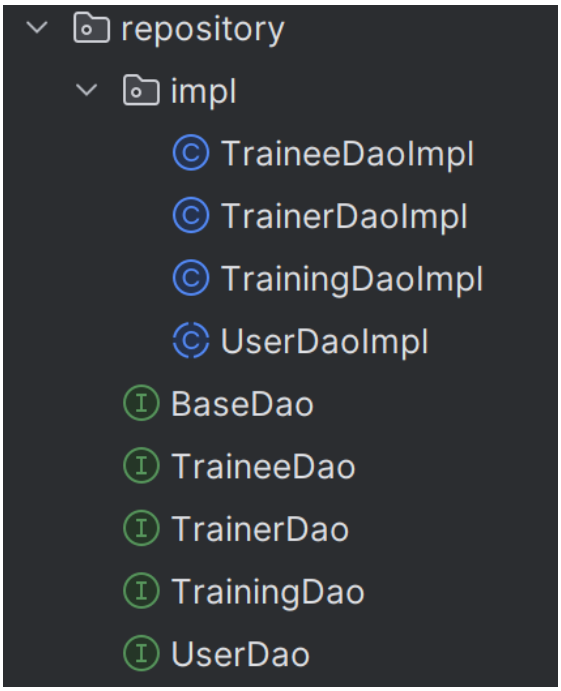


Рисунок 3.6 – структура пакетів репозиторія

– Використання JPA-сущностей із анотаціями @Entity, @Table, валідація полів: @NotBlank, @JsonFormat, а також утилітні методи для перевірки полів (див. рисунок 3.7):

```
2 inheritors  Yura-Moroz
@Data
@SuperBuilder
@NoArgsConstructor
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "users")
public abstract class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    @JsonView({TrainingViews.UpdateTraineeTrainings.class, TrainingViews.GetResp.class, TraineeViews.GetResp.class})
    private Long id;

    @NotNull
    @Column(name = "first_name", nullable = false)
    private String firstName;

    @NotNull
    @Column(name = "last_name", nullable = false)
    private String lastName;

    @NotNull
    @Column(name = "user_name", nullable = false)
    private String userName;

    @NotNull
    @Column(name = "password", nullable = false)
    private String password;

    @NotNull
    @Column(name = "is_active", nullable = false)
    private boolean active;

}
```

Рисунок 3.7 – модель юзера (базового класу для моделей Trainee та Trainer)

- База даних (Persistence Layer)
  - MySQL з MySQL Workbench
  - Схема БД містить таблиці users, trainees, trainers, trainings (див. рисунок 3.8)

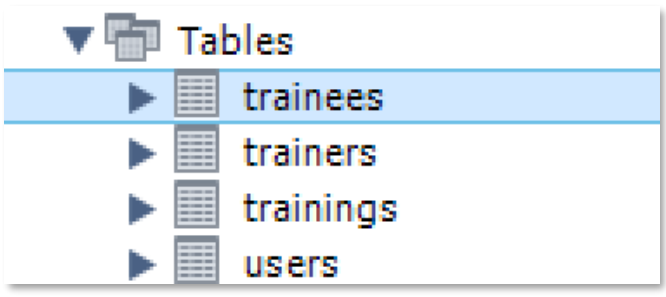


Рисунок 3.8 – схема БД

3.2 Реалізація основних функцій системи

У рамках реалізації дипломного проекту було розроблено RESTful API для керування користувачами, тренерами та тренуваннями. Уся логіка розподілена на чотири основні контролери:

1. Training Controller

Відповідає за операції зі створення та отримання інформації про тренування.

– **POST /gym-api/trainings**

Створити нове тренування. Приймає в тілі запиту DTO з даними: ім’я клієнта (traineeUsername), ім’я тренера (trainerUsername), дата та час проведення тренування, опис вправ. Повертає HTTP 201 Created з деталями створеного об’єкта.

– **GET /gym-api/trainings/types**

Отримати список типів існуючих тренувань. Повертає масив рядків із полями id та порядкового номеру типу тренування.

– **GET /gym-api/trainings/trainer-trainings-date-range**

Отримати перелік тренувань певного тренера за вказаний інтервал дат. Приймає параметри запиту: trainerUsername, startDate, endDate. Повертає масив DTO тренувань.

– **GET /gym-api/trainings/trainee-trainings-date-range**

Отримати перелік тренувань певного клієнта (trainee) за діапазон дат. Параметри: traineeUsername, startDate, endDate. Повертає масив DTO тренувань.

2. Trainer Controller

Забезпечує керування профілями тренерів.

– **PUT /gym-api/trainers/{id}**

Оновити існуючий профіль тренера за його id. У тілі запиту передаються нові значення полів (ім’я, прізвище, спеціалізація тощо). Повертає HTTP 200 OK і DTO оновленого тренера.

– **PUT /gym-api/trainers/password**

Змінити пароль тренера. Приймає об’єкт із полями username, oldPassword, newPassword. Перевіряє правильність старого пароля та шаблон нового, повертає HTTP 204 No Content.

– **POST /gym-api/trainers**

Створити нового тренера. Тіло запиту містить необхідні дані для реєстрації (username, email, пароль, профільна інформація). HTTP 201 Created із DTO тренера.

– **PATCH /gym-api/trainers/status**

Переключити активний статус тренера (active/inactive). Приймає username та новий статус, повертає HTTP 200 OK і оновлений статус.

– **GET /gym-api/trainers/{username}**

Отримати профіль тренера за username. Повертає DTO із персональною інформацією та статусом.

– **GET /gym-api/trainers/{username}/unassigned**

Отримати список тренерів, які ще не були призначені даному клієнту (traineeUsername передається в параметрах). Використовується для вибору доступного тренера.

3. Trainee Controller

Керує профілями клієнтів (trainees) і їх тренуваннями.

– **PUT /gym-api/trainees/{username}/update-trainings**

Оновити список тренувань для конкретного клієнта. Приймає список DTO тренувань, повертає HTTP 200 OK із актуалізованим переліком тренувань.

– **PUT /gym-api/trainees/{id}**

Оновити профіль trainee за id. Аналогічно до тренерського PUT.

– **PUT /gym-api/trainees/password**

Змінити пароль клієнта. Приймає username, oldPassword, newPassword. HTTP 200 OK.

– **POST /gym-api/trainees**

Зареєструвати нового клієнта. Повертає HTTP 201 Created із username та password.

– **PATCH /gym-api/trainees/status**

Переключити активний статус клієнта. Повертає HTTP 200 OK із новим станом полягнення.

– **GET /gym-api/trainees/{username}**

Отримати профіль клієнта за username.

– **DELETE /gym-api/trainees/{username}**

Видалити профіль клієнта за username. Повертає HTTP 200 OK.

4. Auth Controller

Забезпечує механізми аутентифікації та виходу з системи.

– **POST /auth/login**

– Виконати вхід користувача. Приймає username та password, перевіряє облікові дані, повертає токен доступу (JWT) та статус HTTP 202 Accepted.

– **POST /auth/logout**

– Виконати вихід із системи. Інвалідує токен та повертає HTTP 200 OK.

3.3 Інтеграція бази даних та забезпечення безпеки

Інтеграція із БД забезпечується DAO шаром, як і зазначалося раніше, але до цього ще додається конфігурація, яка прописується у файлі application.properties який Spring за замовчуванням шукає для відтворення конфігураційних налаштувань [24,25]:

application.properties

```
# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/my_db
spring.datasource.username=****
spring.datasource.password=****
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA and Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=create-drop
```

Рисунок 3.9 – Application.properties файл

Кожен користувач повинен бути автентифікованим у системі, щоб виконувати будь-які дії з бронювання тренувань, редагування профілю, керування списком тренувань тощо. Для неавтентифікованих користувачів доступні лише два ендпоінти в контролері Auth: **sign-up** та **sign-in**.

Для реалізації цього функціоналу використано модуль безпеки Spring Security. Аутентифікація здійснюється за допомогою JWT-токена, який генерується на основі поля username, дати і часу створення токена та терміну його дії. Кожного разу, коли користувач звертається до захищеного ресурсу, клієнт надсилає на сервер свій JWT. Сервер із отриманих даних відтворює (регенерує) токен із перевіркою цифрового підпису, відомого лише серверу. Якщо підпис або вхідні дані не збігаються, токен визнається недійсним, і доступ до ресурсу забороняється. Таким чином забезпечується надійний захист від несанкціонованого доступу та модифікації даних [13,14,15, 16,17].

3.4 Тестування системи та перевірка коректності роботи API

Тестування якості коду досягнуто покриттям тестами 80% коду. Усі тести гарантують, що код працює належним чином та витримує перевірки нестандартної поведінки та навантажень.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 93, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  38.340 s
[INFO] Finished at: 2025-05-17T02:09:24+03:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Рисунок 3.11 – результат виконання усіх наявних unit-тестів

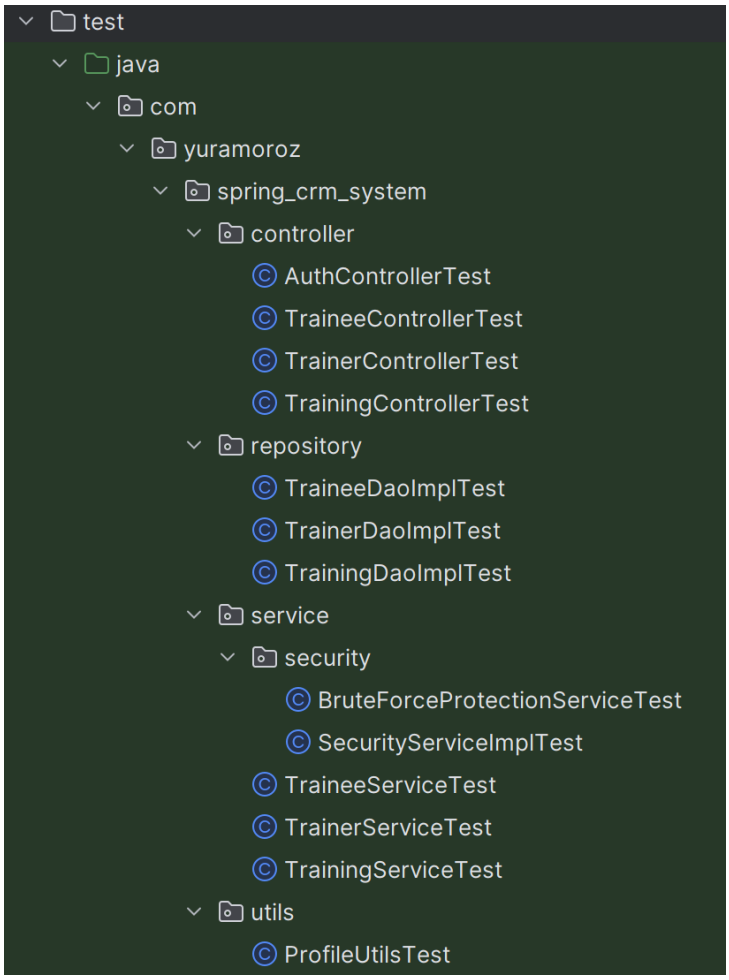


Рисунок 3.10 – структура пакету з unit-тестами

У рисунку 3.10 представлена структура пакету з юніт-тестами для проекту, що має назву spring_crm_system. Він організований за певною ієрархією, що дозволяє зручно розділяти тести за різними компонентами системи:

– **controller**: містить тести для контролерів, які відповідають за обробку HTTP-запитів. Тут представлені такі класи, як AuthControllerTest, TraineeControllerTest тощо.

– **repository**: охоплює тести для класів доступу до даних, що виконують операції з базою даних. Наприклад, TraineeDaoImplTest та TrainerDaoImplTest.

– **service**: включає тести для бізнес-логіки програми. Тут розділ безпеки має свої тести, такі як BruteForceProtectionServiceTest.

– **utils**: містить тести для утилітних класів, наприклад, ProfileUtilsTest.

Для гарантування справності роботи API усі запити, для тестування різної поведінки роботи, були виконані у Postman. Також для документування та кращої візуалізації API запитів було під’єднано Swagger (див. рисунок 3.12, 3.13):

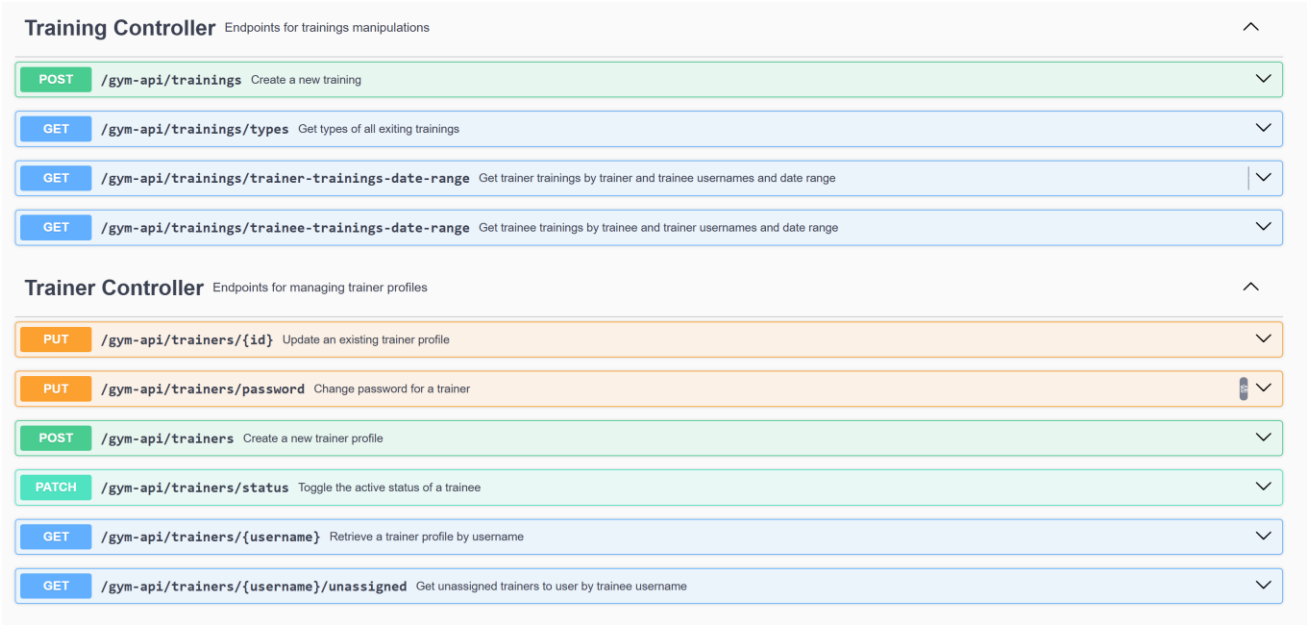


Рисунок 3.12 – Swagger UI із ендпоінтами для керування тренуваннями та тренерських профілів

Рисунок 3.13 – Swagger UI із ендпоінтами для керування профілів користувачів, а також ендпоінти для логіну та логауту

Створення користувача у Postman (див. рисунок 3.14):

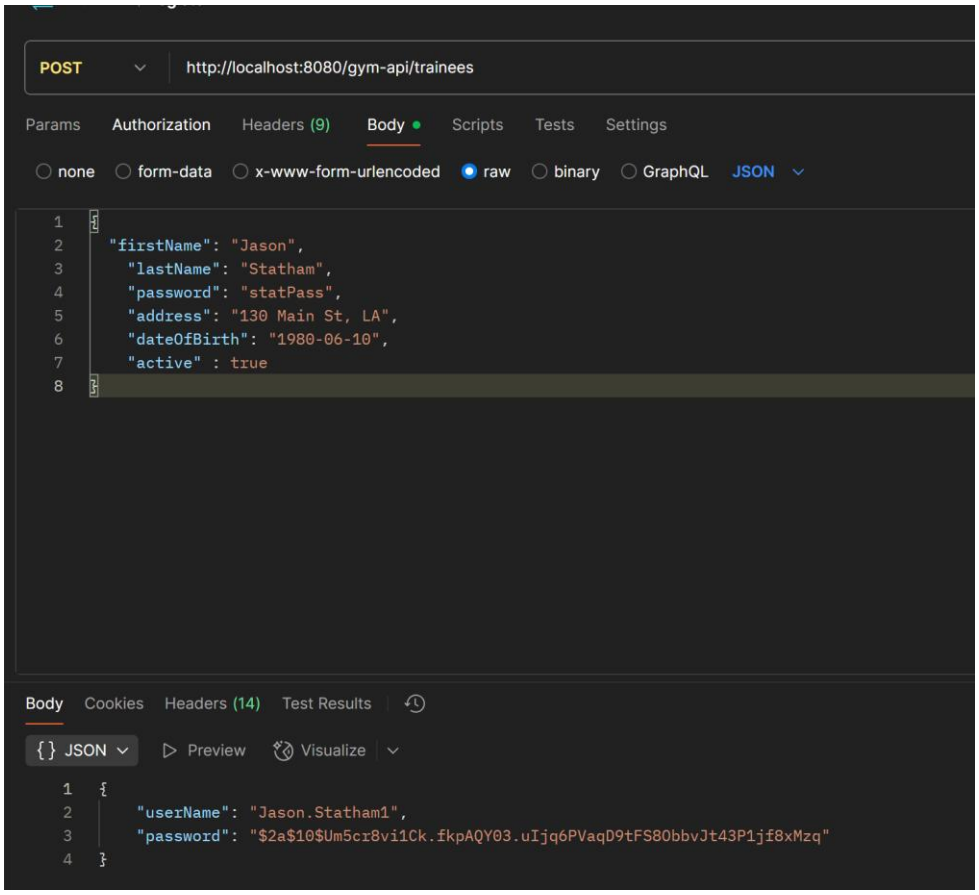


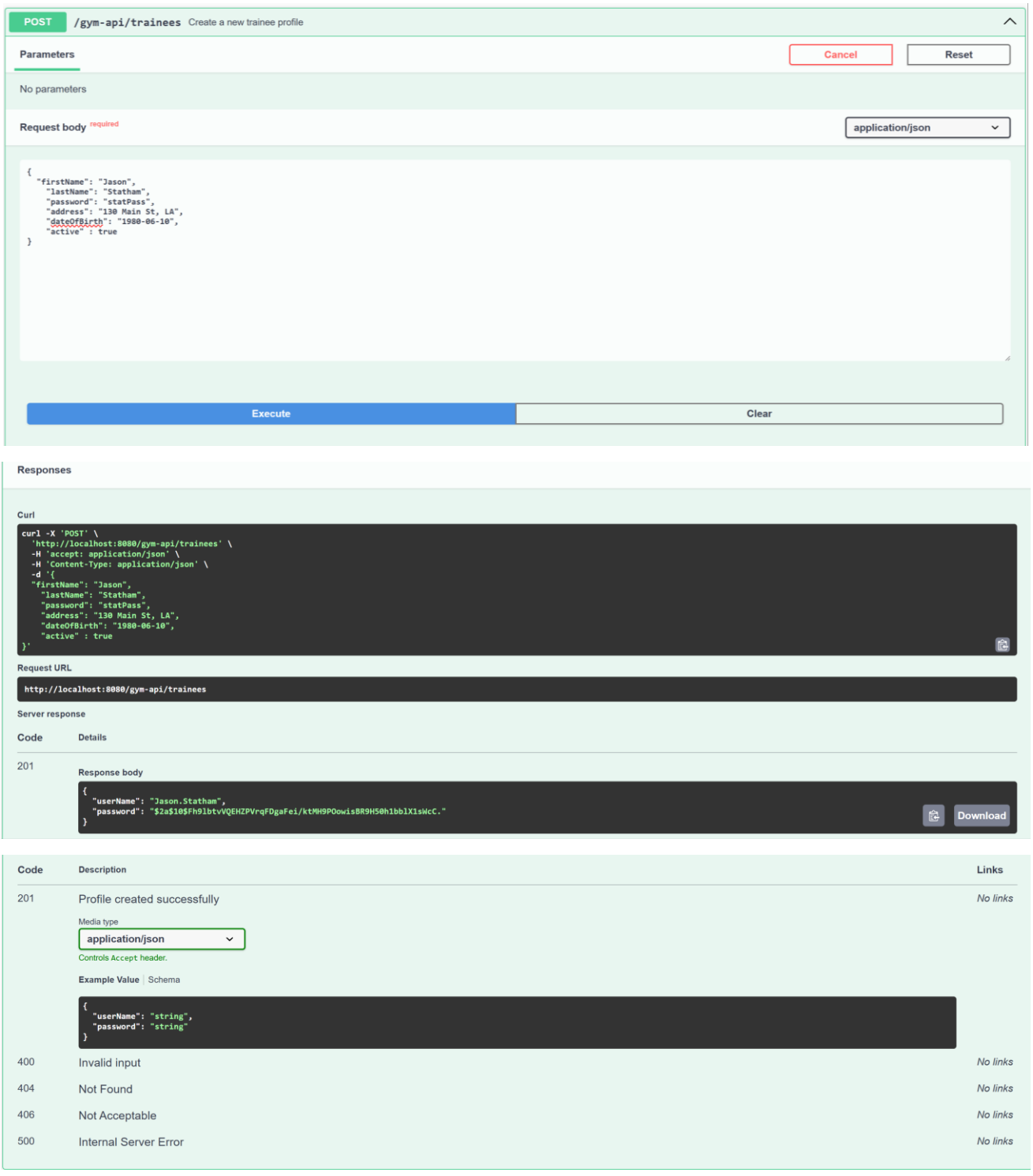
Рисунок 3.14 – реєстрація користувача за допомогою засобів Postman

1. **HTTP Метод**: Використовується метод POST, що означає, що дані будуть надіслані на сервер для створення нового ресурсу (в даному випадку, нового тренера або учня).
1. **URL**: Вказує адресу, куди надсилається запит. У цьому випадку це точка доступу для роботи з "trainees".

– **Body**: У розділі тіла запиту представлено дані в форматі JSON:
  – **firstName**: "Jason" – ім'я користувача.
  – **lastName**: "Statham" – прізвище користувача.
  – **password**: "statPass" – пароль.
  – **address**: "130 Main St, LA" – адреса проживання.
  – **dateOfBirth**: "1980-06-10" – дата народження.
  – **active**: true – статус активності користувача.
– **Response**: У нижній частині зображення показано, як API відповідає на запит:

– **userName**: "Jason.Statham1" – згенероване ім'я користувача.

– **password**: закодований пароль, що свідчить про успішну реєстрацію.



Рисунки 3.15 – реєстрація користувача за допомогою засобів Swagger

Після реєстрації користувача ми можемо побачити username, який генерується сервером автоматично базуючись на переданих даних користувача, а також у

відповіді ми можемо побачити пароль у захешованному вигляді, як він зберігається у БД (див. рисунок 3.16):

	is_active	id	first_name	last_name	password	user_name
▶	1	1	Jason	Statham	$2a$10$Fh9lbtvVQEHZPVrqFDgaFei/ktMH9POo...	Jason.Statham

Рисунок 3.16 – представлення створеного користувача у таблиці users

Далі спробуємо отримати користувача за його username:

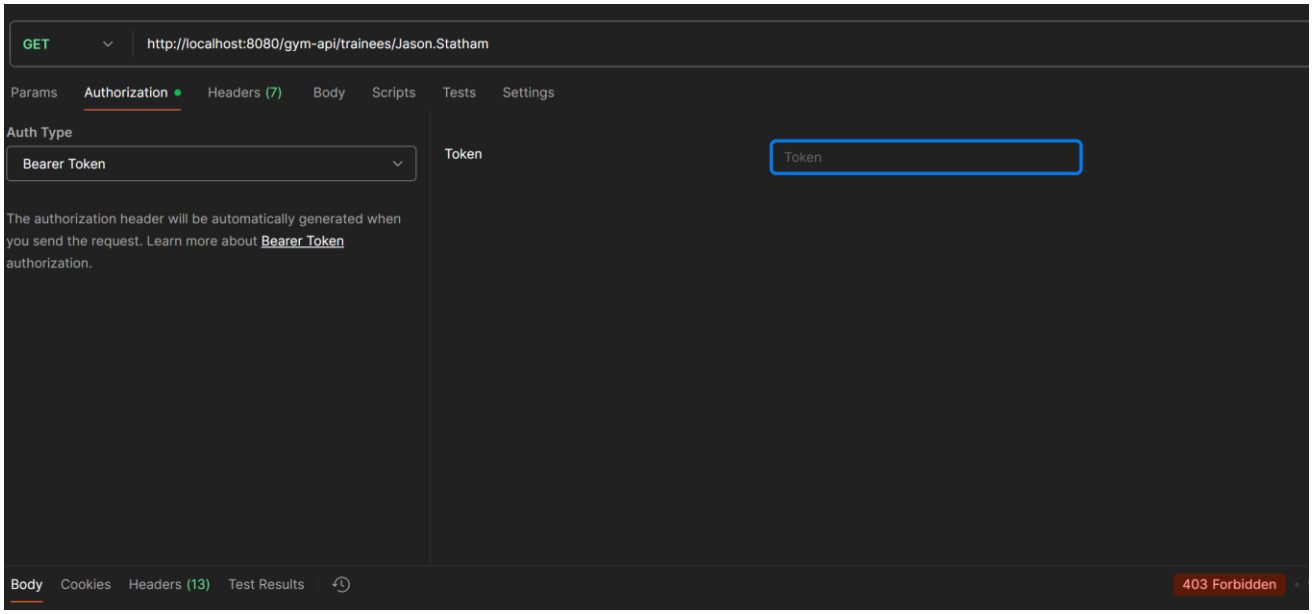


Рисунок 3.17 – спроба отримати користувача без access токену

Як ми можемо побачити, при відсутності або некоректності токену, ми отримуємо 403 помилку, що свідчить про те, що нам відмовлено у доступі.

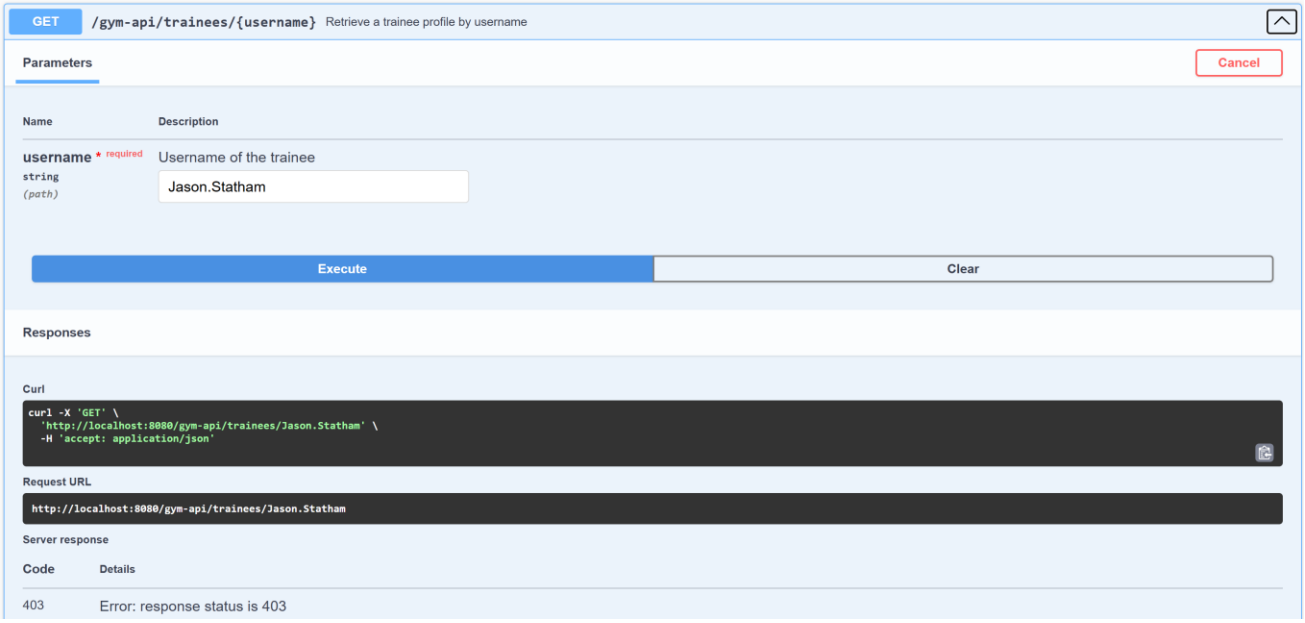


Рисунок 3.18 – спроба отримати користувача без access токену у Swagger

Щоб виконати запити для яких потрібна автентифікація, ми повинні увійти в систему та передати access токен у заголовках.

Тому виконаємо login запит, щоб отримати токен доступу для виконання запиту по отриманню користувача:

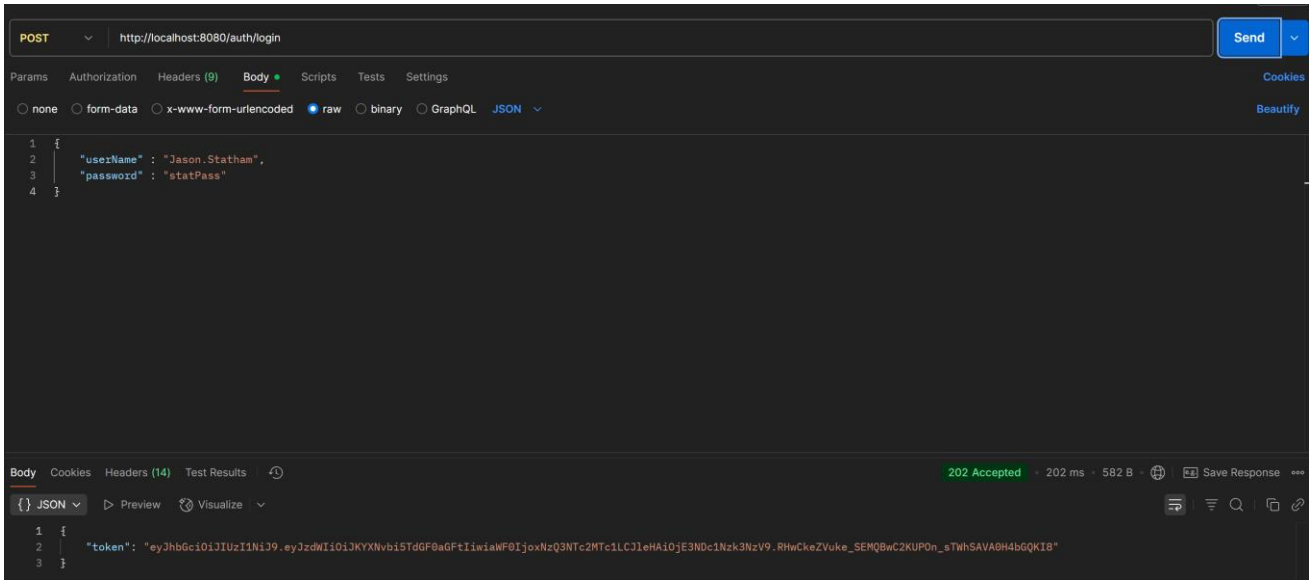


Рисунок 3.19 – логін запит для користувача у Postman

Виконаємо те саме у Swagger:

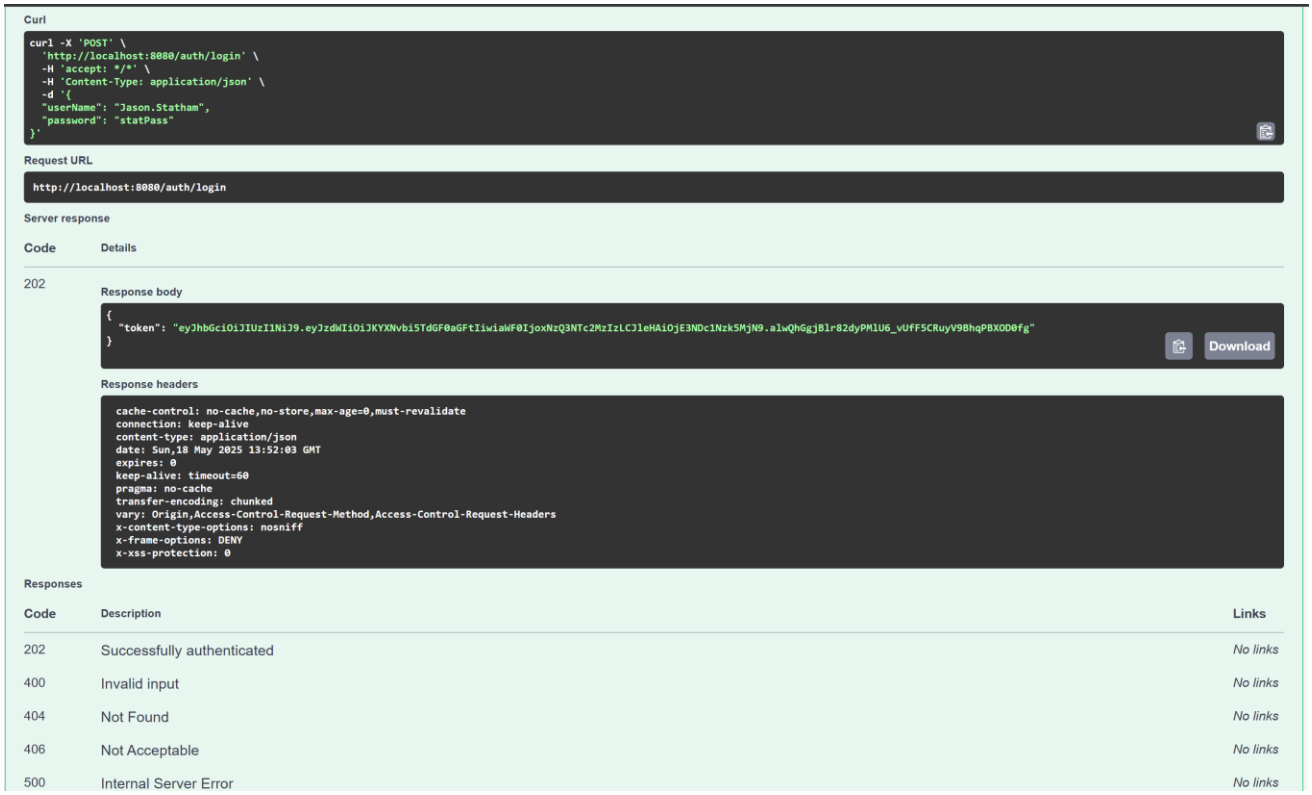


Рисунок 3.20 – логін запит для користувача у Swagger

Тепер маючи токен доступу виконаємо запит по отриманню користувача (див. рисунок 3.21):

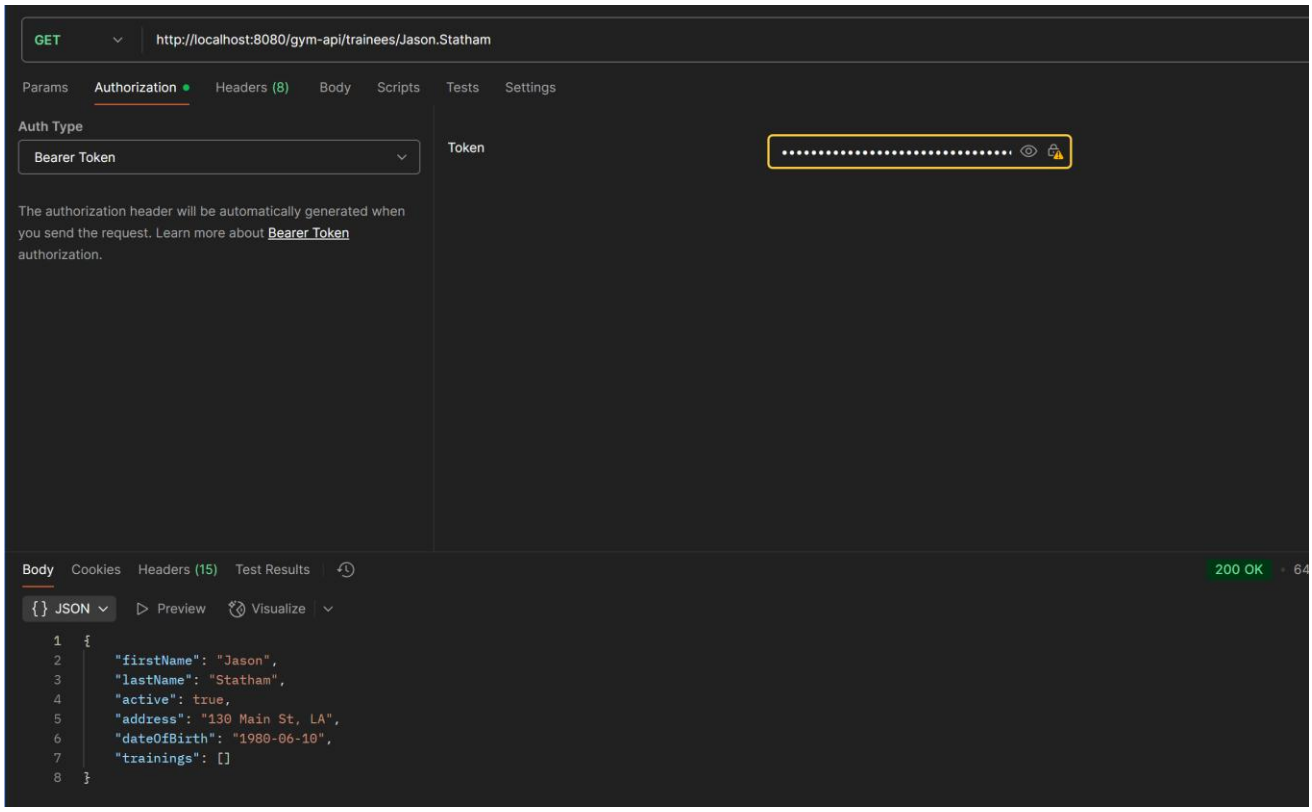


Рисунок 3.21 – запит для отримання користувача по username у Postman

На представленому рисунку 3.21 зображено запит у Postman для отримання інформації про користувача за його ім'ям користувача (username) з використанням токена доступу.

Цей процес демонструє, як можна отримати детальну інформацію про користувача, маючи валідний токен доступу. Використання токенів є поширеною практикою в сучасних веб-додатках для забезпечення безпеки та контролю доступу. Завдяки такій авторизації, лише авторизовані користувачі можуть отримувати або змінювати дані, що підвищує рівень захисту інформації.

Далі спробуємо змінити статус користувача. Зараз статус для нашого юзера – active. Спробуємо виконати запит по зміні статусу для профіля юзера, щоб профіль був неактивний (див. рисунок 3.22):

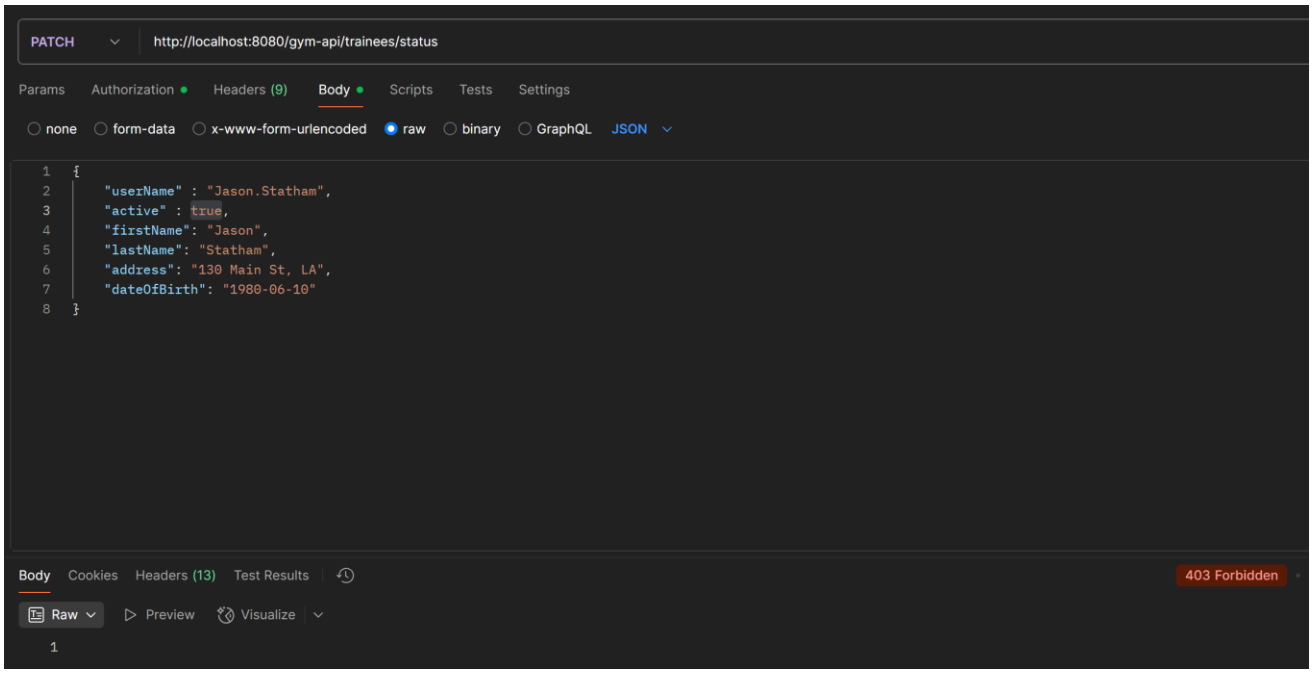


Рисунок 3.22 – запит для зміни статусу профіля користувача у Postman

Як ми можемо бачити, ми отримали відмову із кодом 403, бо не передали токен. Так як раніше ми вже увійшли в систему, використовуємо токен, що повернувся при запиті /login (див. рисунок 3.23):

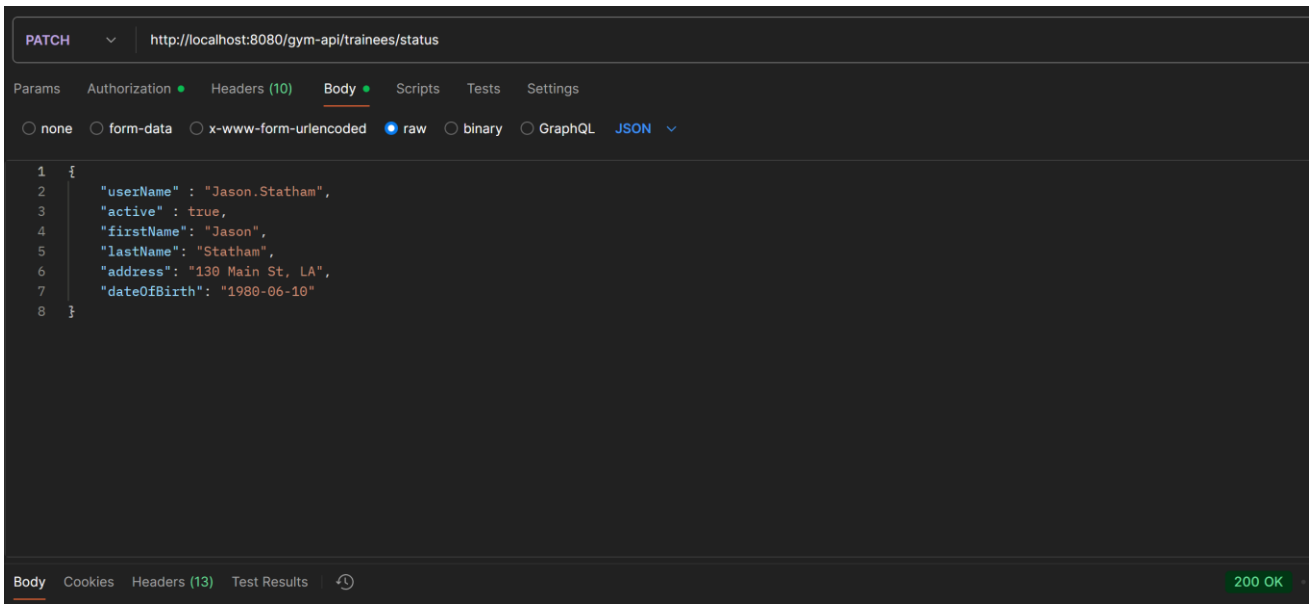


Рисунок 3.23 – запит для зміни статусу профіля користувача у Postman (вдалий)

Тепер перевіримо чи дійсно статус змінено. Для цього отримаємо користувача по username (див. рисунок 3.24):

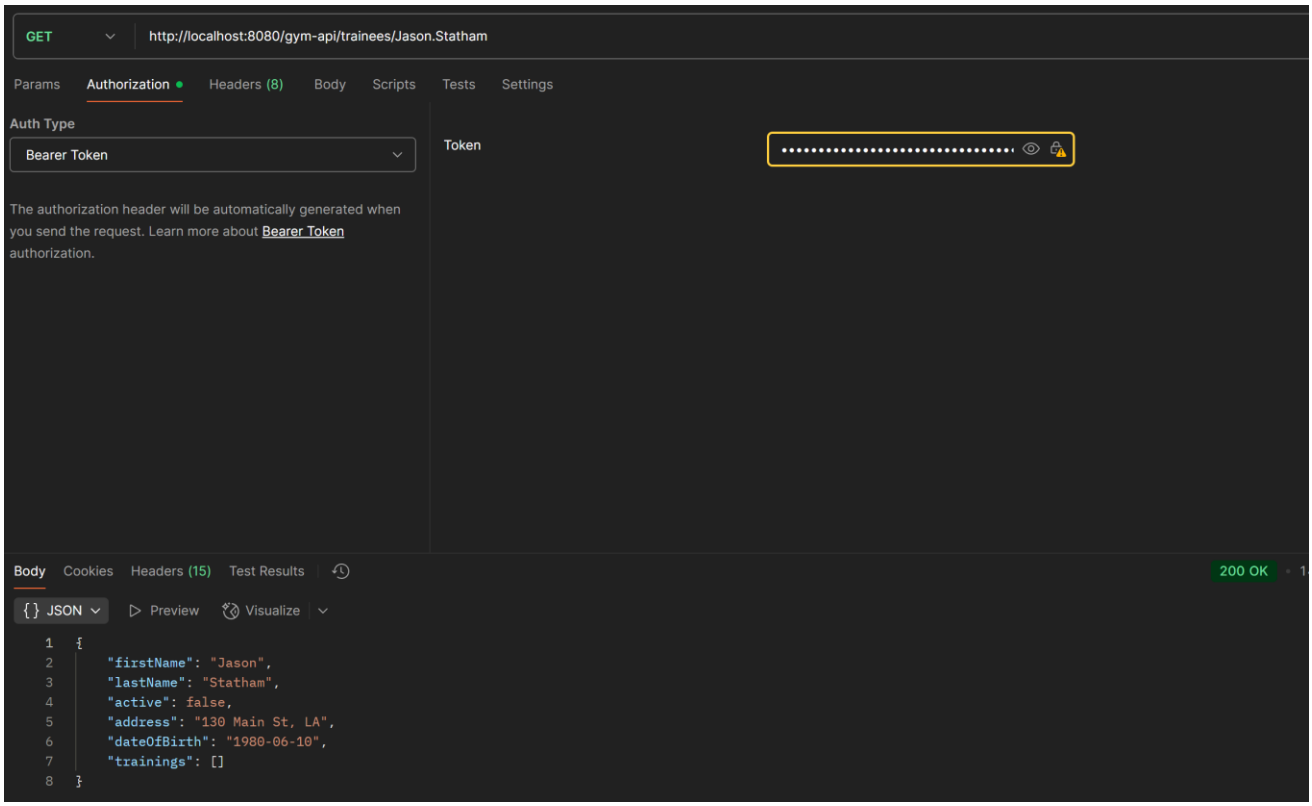


Рисунок 3.24 – запит для отримання користувача із БД

Як ми можемо побачити, показник статусу для цього профіля користувача змінився на **false**.

Якщо ми подивимось у БД, то зможемо ще раз переконатись, що статус активності профіля дорівнює нулю, що означає false (див. рисунок 3.25):

true – 1,

false – 0

	is_active	id	first_name	last_name	password	user_name
▶	0	1	Jason	Statham	$2a$10$Fh9lbtvVQEHZPVrqFDgaFei/ktMH9POo...	Jason.Statham

Рисунок 3.25 – представлення юзера у БД в таблиці users

Наступним кроком перевіримо функціональність по зміні паролю для користувача. На цьому етапі ми також повинні передати токен доступу, як і для усіх інших запитів, тому відмову сервера із response status 403 я більше демонструвати не буду. Спробуємо передати невалідний старий пароль для цього запиту й подивимось на результат (див. рисунок 3.26):

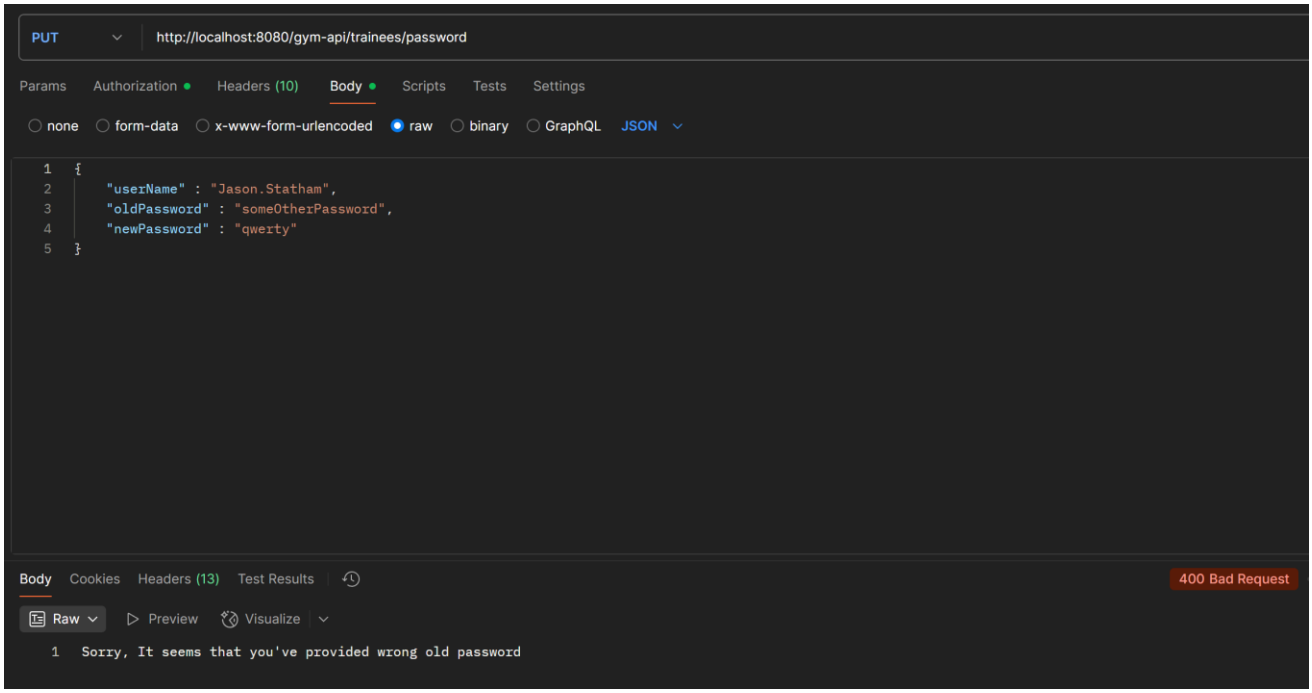


Рисунок 3.26 – результат виконання запиту по зміні паролю для юзера, коли старий пароль невалідний

Як ми бачимо, система сповіщає, що старий пароль недійсний. Щоб змінити пароль, користувач повинен передати:

- ім’я користувача,
- правильний старий пароль,
- новий пароль, який відповідає шаблону (у нашому випадку — довжина від 4 до 10 символів).

```
Yura-Moroz
public static String hashPassword(String password) {
    if(!verify(password)) throw new IllegalArgumentException("Password should be 4-10 chars in length");
    return passwordEncoder.encode(password);
}
```

Рисунок 3.27 – демонстрація сповіщення користувача при неправильному шаблоні для створення паролю (у коді)

Наступним кроком перевіримо чи дійсно юзер бачить попередження про невідповідність нового паролю до заданого шаблону (див. рисунок 3.28):

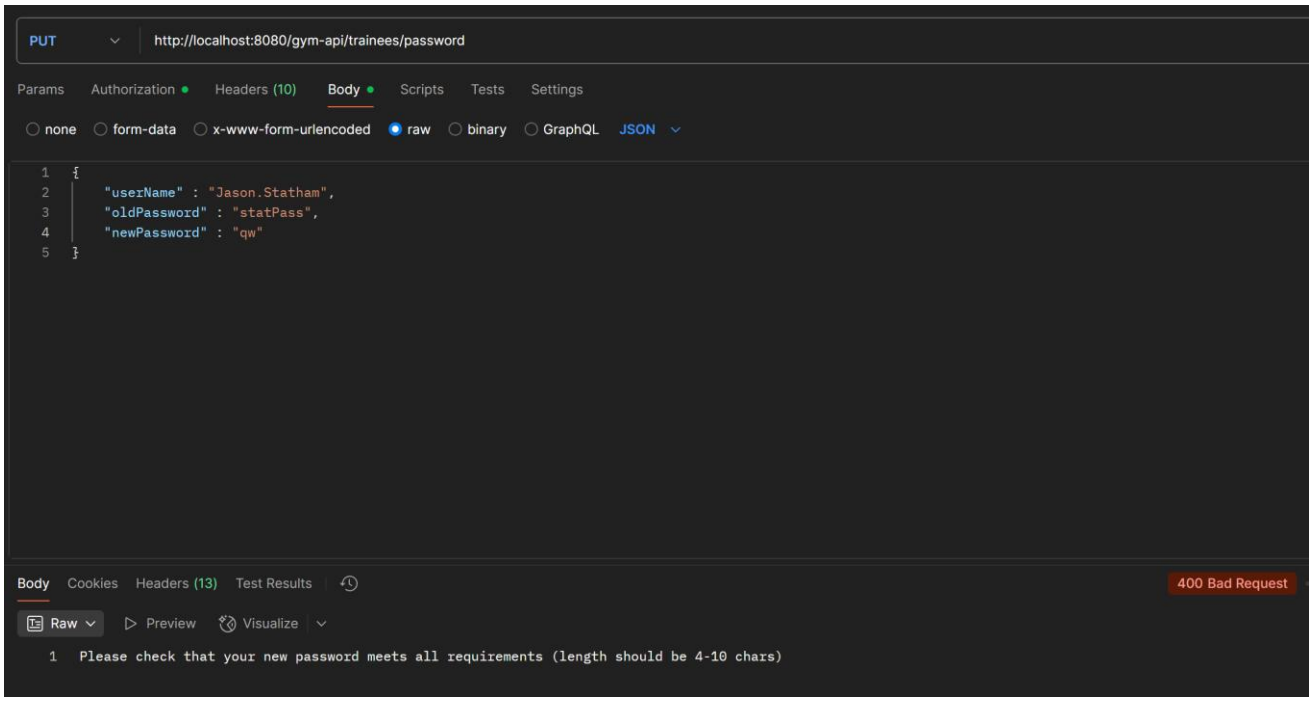


Рисунок 3.28 – демонстрація сповіщення користувача при неправильному шаблоні для створення паролю

Response: Внизу екрана видно повідомлення про помилку:

- Код статусу 400 Bad Request вказує на те, що запит не може бути оброблений через невірний формат або невідповідність даних.
- Повідомлення: "Please check that your new password meets all requirements (length should be 4-10 chars)" – це попередження системи, що новий пароль не відповідає встановленим вимогам. У даному випадку, новий пароль занадто короткий: він має містити від 4 до 10 символів.

Цей процес демонструє важливість валідації даних на етапі введення. Система забезпечує зручність для користувачів, надаючи їм зрозумілі повідомлення про помилки. Це не лише покращує досвід користувача, але й підвищує безпеку, запобігаючи встановленню слабких паролів. Запит на зміну пароля і відповідь системи ілюструють важливість дотримання вимог до паролів для забезпечення надійності облікових записів.

Тепер спробуємо змінити пароль для профіля користувача враховуючи всі вимоги (див. рисунок 3.29):

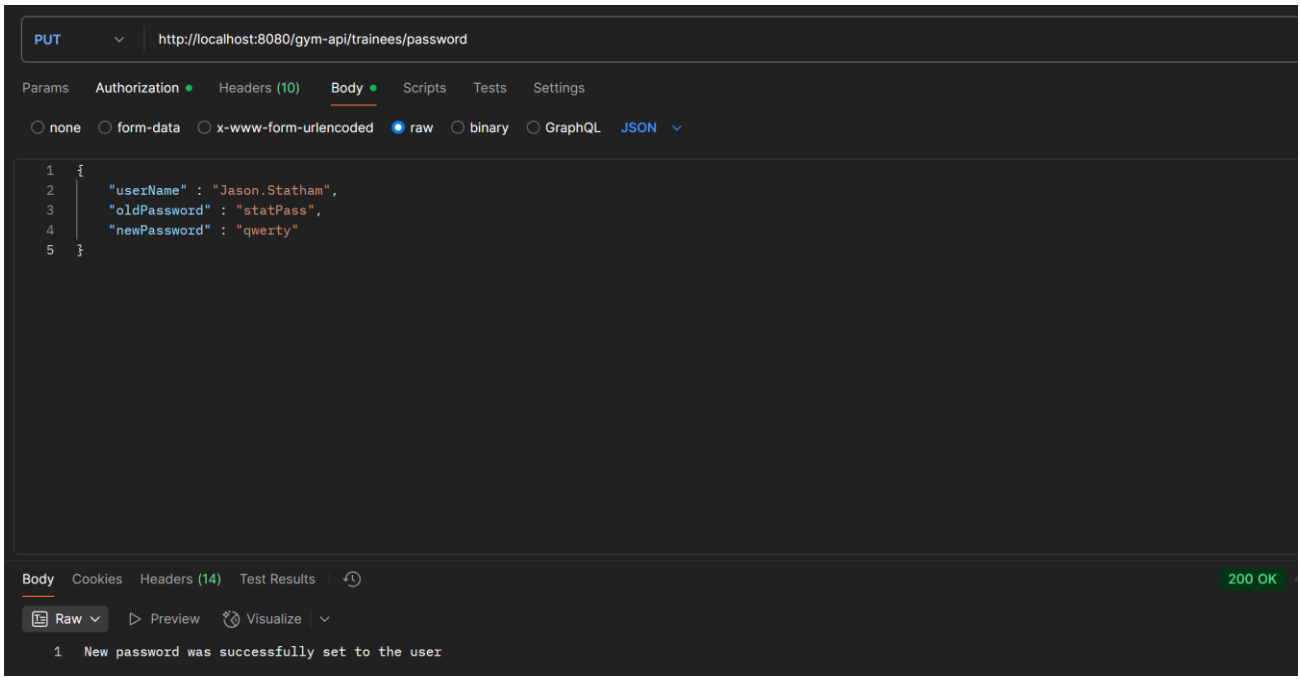


Рисунок 3.29 – демонстрація сповіщення користувача при неправильному шаблоні для створення паролю

Ми є свідками того, що пароль був успішно змінений для цього юзера. Щоб переконатись в цьому, виконаємо логін ще раз, але із старим паролем (див. рисунок 3.30):

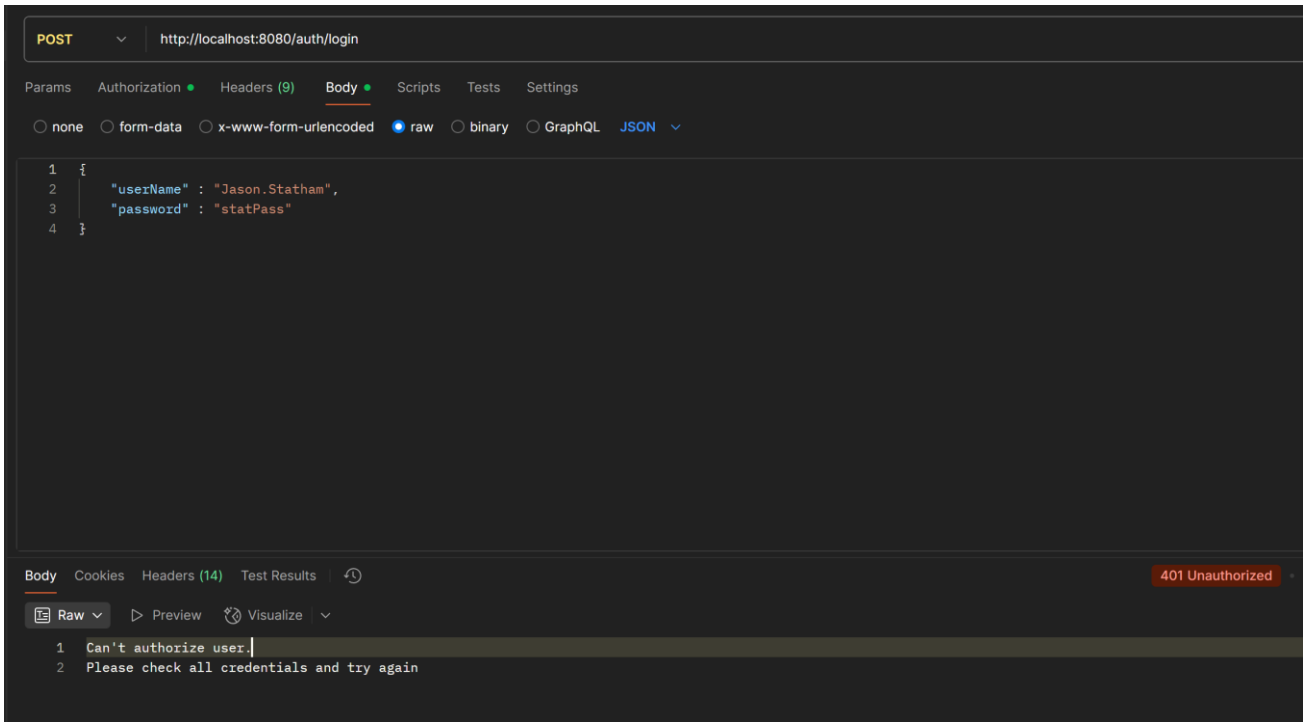


Рисунок 3.30 – демонстрація неправильних вхідних даних при /login запиті

Виконаємо той самий запит із новим паролем (див. рисунок 3.31):

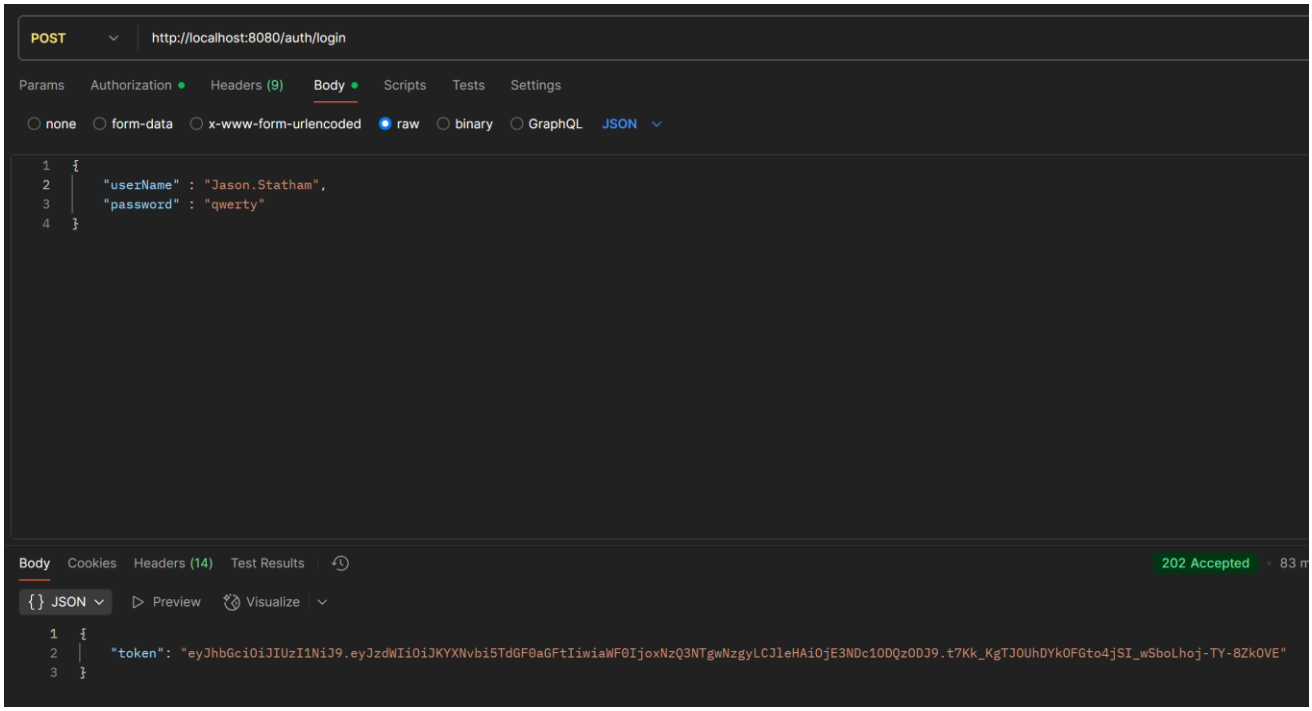


Рисунок 3.31 – демонстрація валідних вхідних даних при /login запиті

Response: Внизу екрана видно, як API відповідає на запит:

- Код статусу 202 Accepted вказує на те, що запит прийнято, але ще не оброблено. Це звичайно означає, що сервер успішно отримав запит.
- У відповіді в форматі JSON міститься поле **token**, що є токеном доступу. Цей токен використовується для авторизації подальших запитів до API.

Токен має вигляд:

"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKason.StathamIiwibmFtZSI6Ikphc29uIiwiaWF0IjoxNTE2MjM5MjIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"

Він містить зашифровану інформацію, яка підтверджує особу користувача.

Цей процес демонструє, як користувач може авторизуватися в системі, надсилаючи свої дані, а також отримати токен доступу для подальшої взаємодії з API [17].

Наступним етапом виконаємо запит по зміні профілю для користувача по його ідентифікатору. Змінюватись можуть усі поля окрім username. Для демонстрації його виконаємо апдейт запит, де спробуємо змінити username у тілі запиту (див. рисунок 3.32):

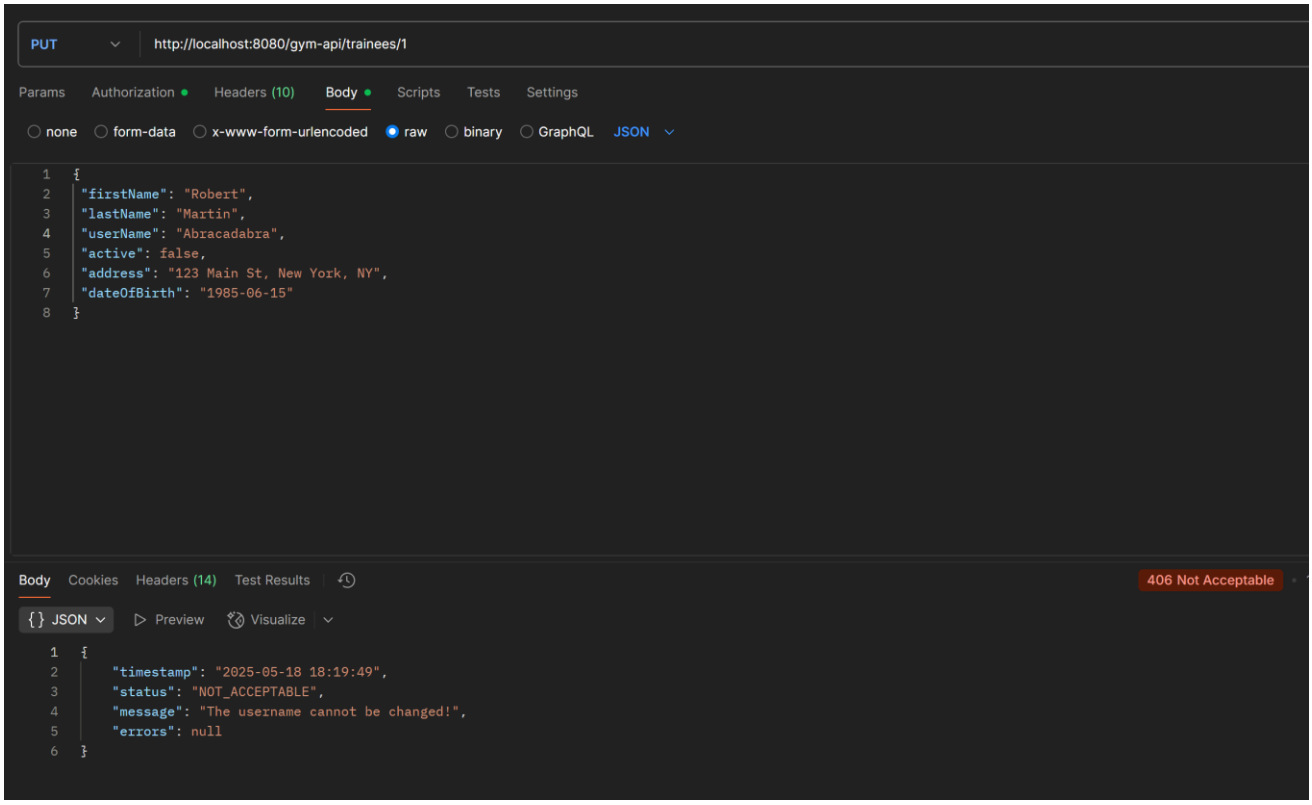


Рисунок 3.32 – відмова у зміні поля username для профілю користувача

Response: У нижній частині екрана видно відповідь сервера:

1. Код статусу 406 Not Acceptable свідчить про те, що запит не може бути оброблений, оскільки користувач не має права змінювати своє ім'я користувача.
2. У відповіді в форматі JSON вказано:
   – **timestamp**: "2025-05-18 18:19:49" – дата та час виникнення помилки.
   – **status**: "NOT_ACCEPTABLE" – статус помилки.
   – **message**: "The username cannot be changed!" – повідомлення, яке пояснює причину відмови в обробці запиту.
   – **errors**: null – жодних додаткових помилок не виявлено.

Результатом є 406 статус код та повідомлення про те, що нам не дозволено змінювати username для профіля користувача.

Продемонструємо успішний сценарій, де успішно виконується зміна прописаних полів (див. рисунок 3.33):

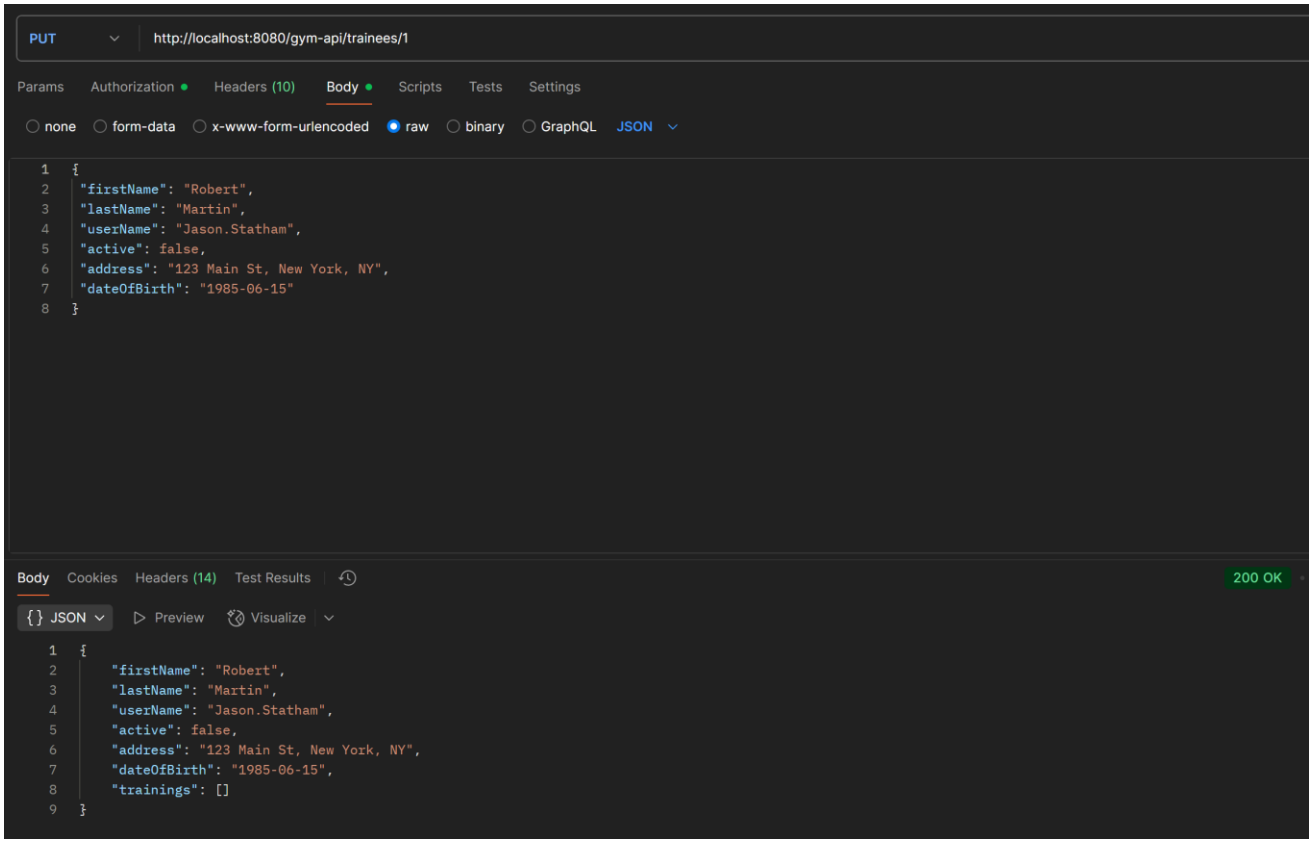


Рисунок 3.33 – успішна зміна необхідних полів для профілю користувача

Переконаємось, що дійсно нові поля при запиті юзера відображаються (фото праворуч), (див. рисунок 3.34), та зможемо порівняти із старими полями для цього ж юзера (фото ліворуч), (див. рисунок 3.34):

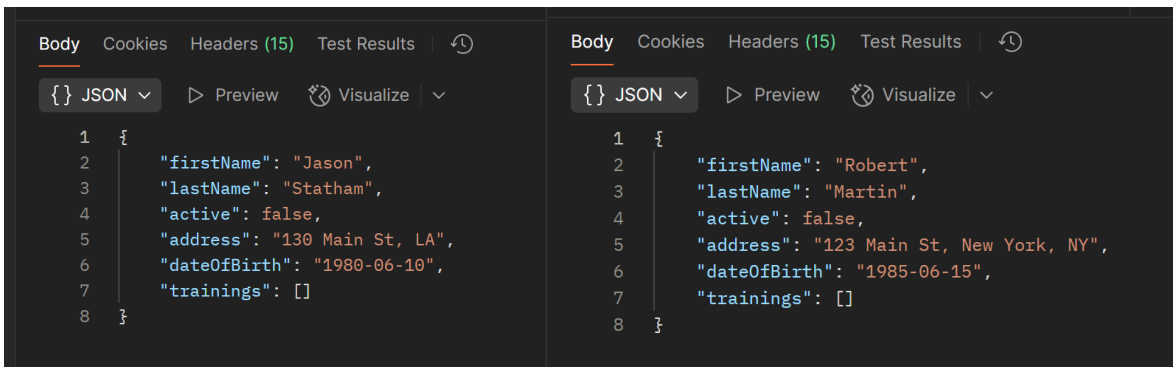


Рисунок 3.34 – порівняння старих і нових даних для профілю користувача

Переходимо до профілю тренера та спробуємо зареєструвати нового тренера у системі (див. рисунок 3.35):

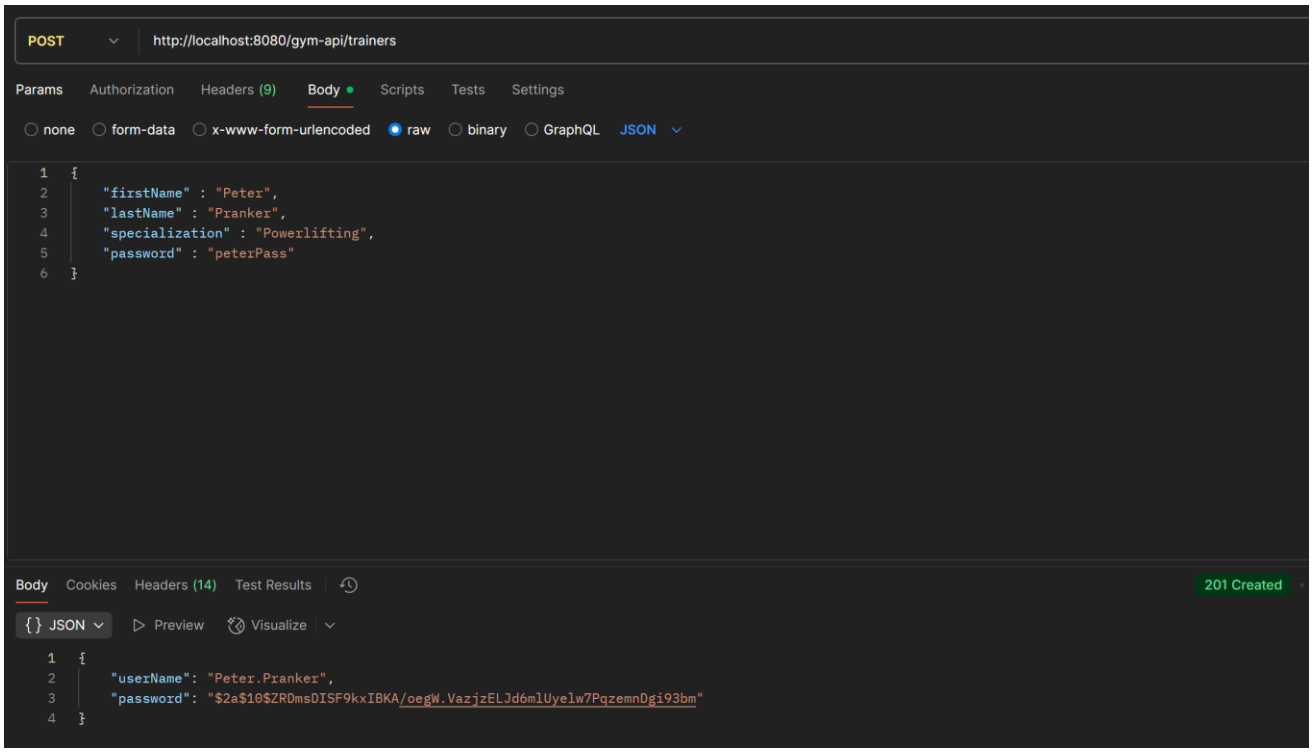


Рисунок 3.35 – створення нового профілю для тренера

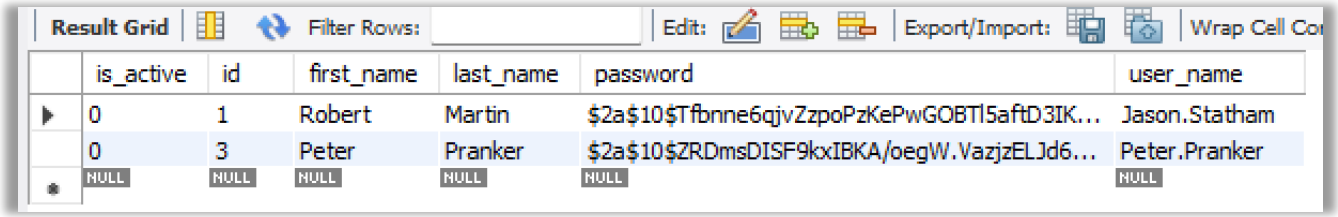


Рисунок 3.36 – результат створення нового тренера у таблиці users

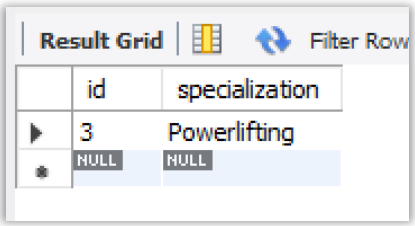


Рисунок 3.37 – результат створення нового тренера у таблиці trainers

В результаті маємо створеного тренера у БД. Результатом запиту маємо сгенерований юзернейм та захешований пароль, як і у випадку із користувачем(Trainee).

Виконаємо отримання тренера по юзернейму (див. рисунок 3.38):

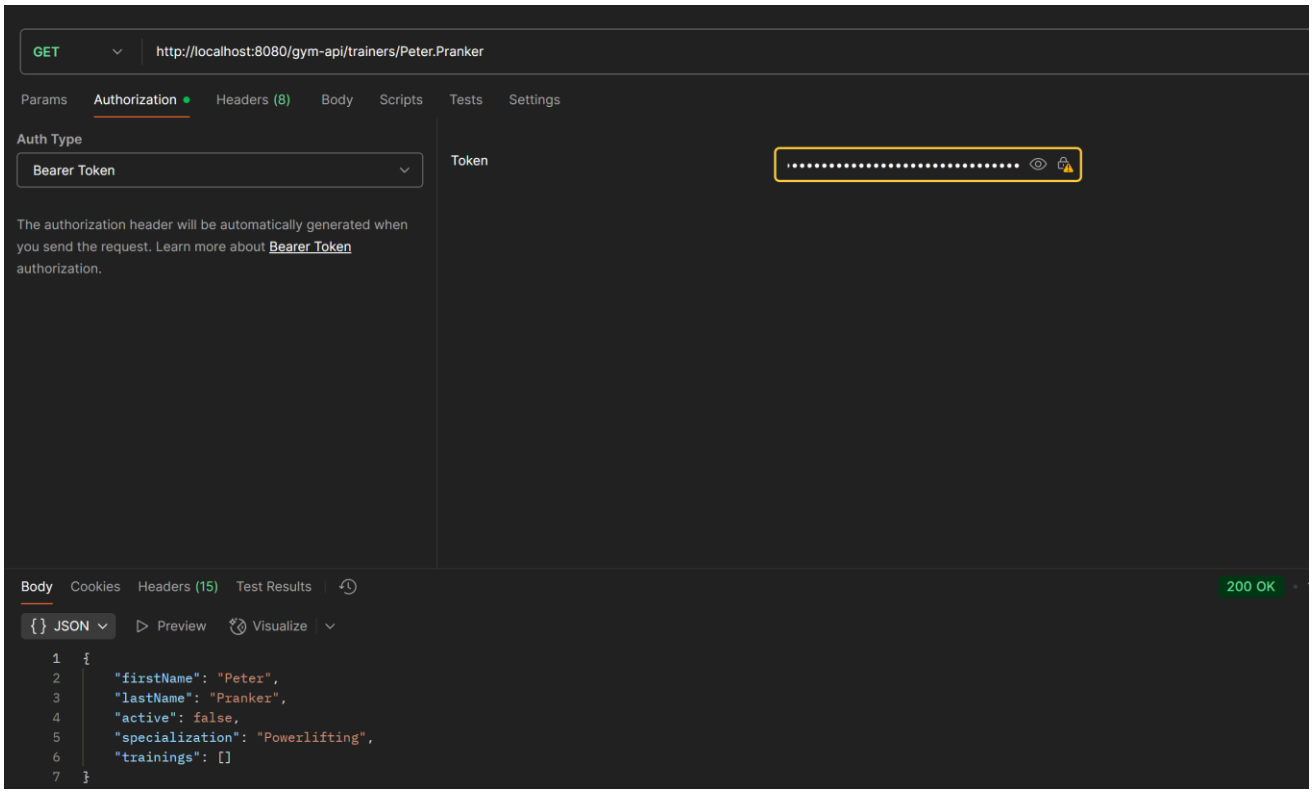


Рисунок 3.38 – результат отримання тренера по username

Далі змінемо пароль для профіля тренера використовуючи одразу валідні вхідні параметри (див. рисунок 3.39):

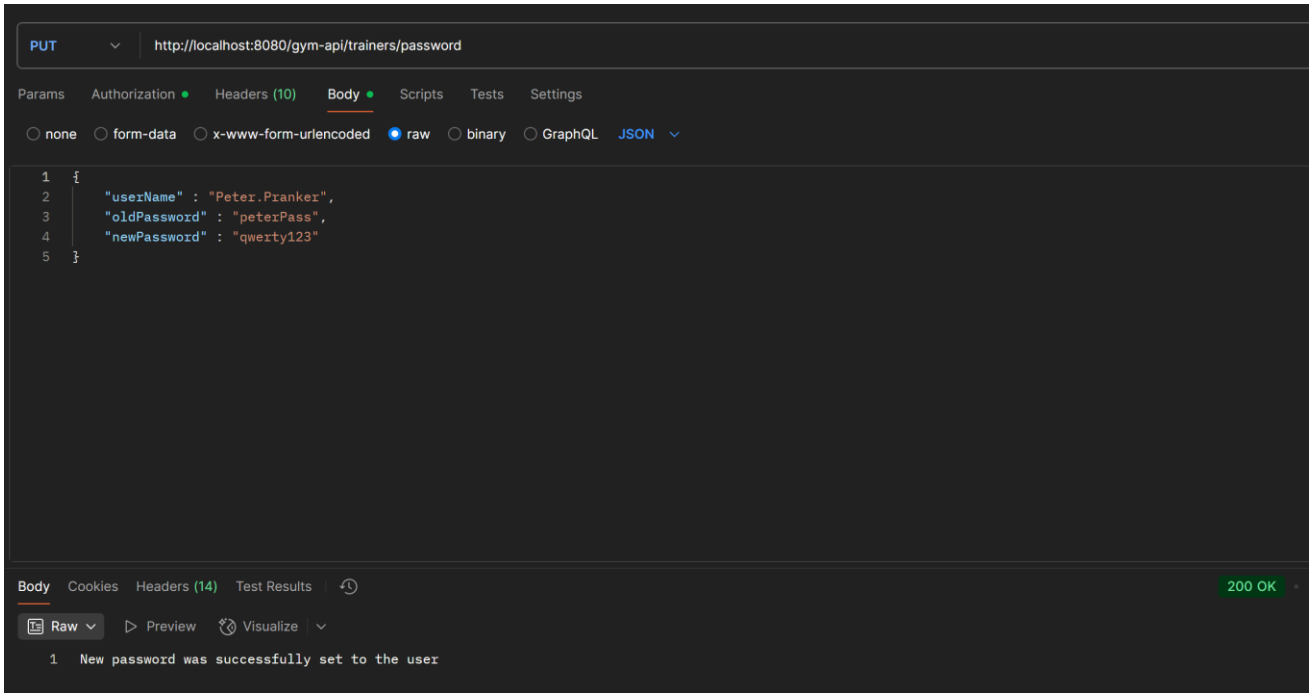


Рисунок 3.39 – результат зміни паролю для тренера

Усі запит для тренера, окрім реєстрації, повинні бути доступні тільки впроваджуючи валідний токен доступу. Усі вищенаведені запити були зроблені

після логіну тренера у систему. Виконаємо наступний запит для зміни статусу профіля тренера спочатку із недійсним токеном (див. рисунок 3.40):

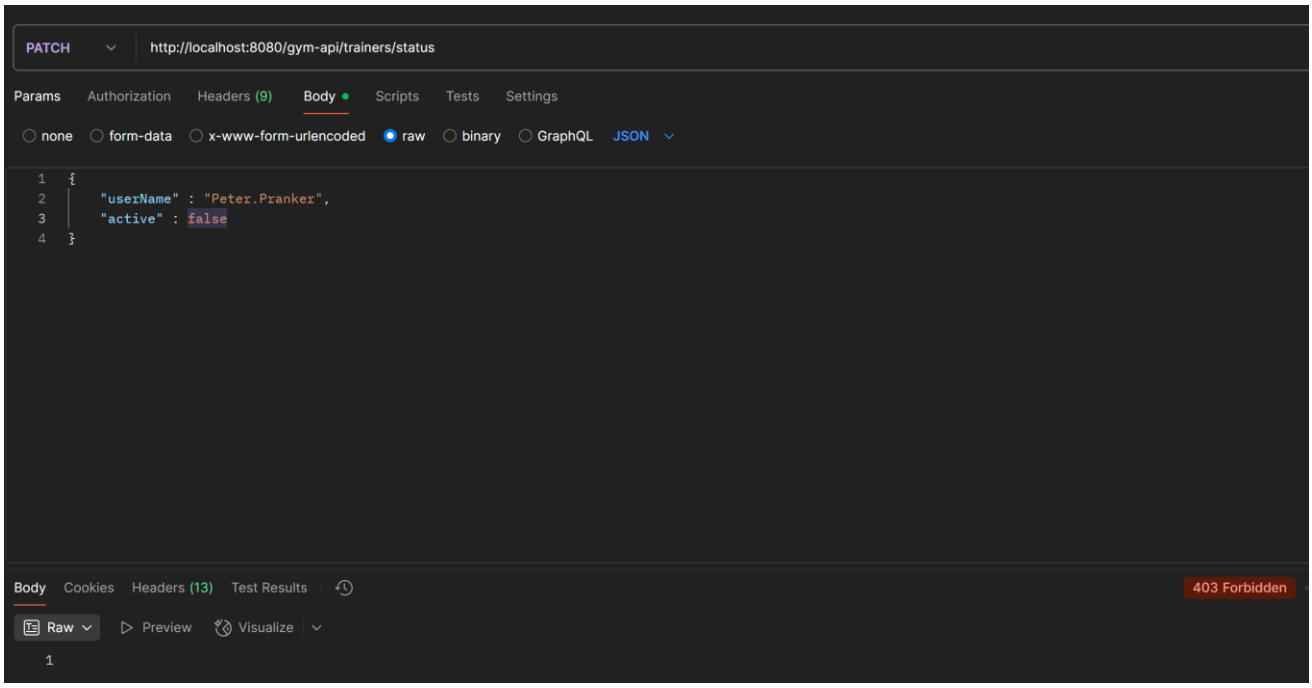


Рисунок 3.40 – результат відмови сервера без дійсного токену

Тепер той самий запит із валідним токеном доступу (див. рисунок 3.41):

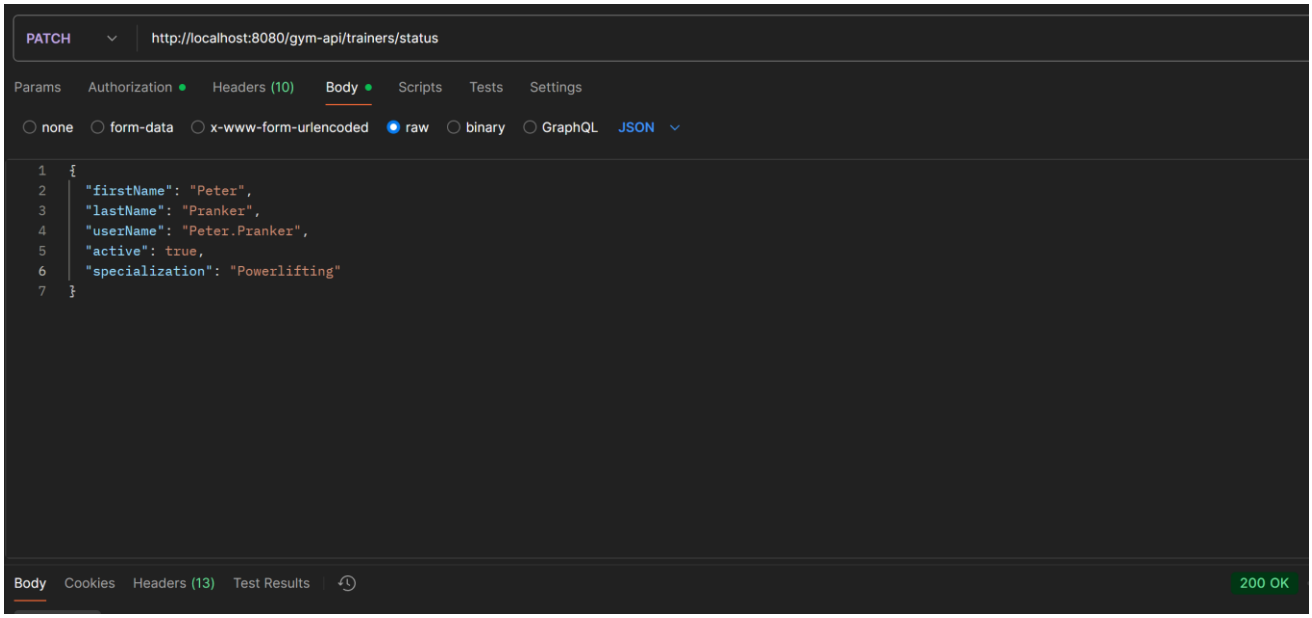


Рисунок 3.41 – результат успішної зміни статусу для тренера

Переконаємось в змінах. Для цього виконаємо отримання профілю тренера за username, щоб запевнитись, що статус дійсно змінено на протилежний та усі інші поля залишились без змін (див. рисунок 3.42):

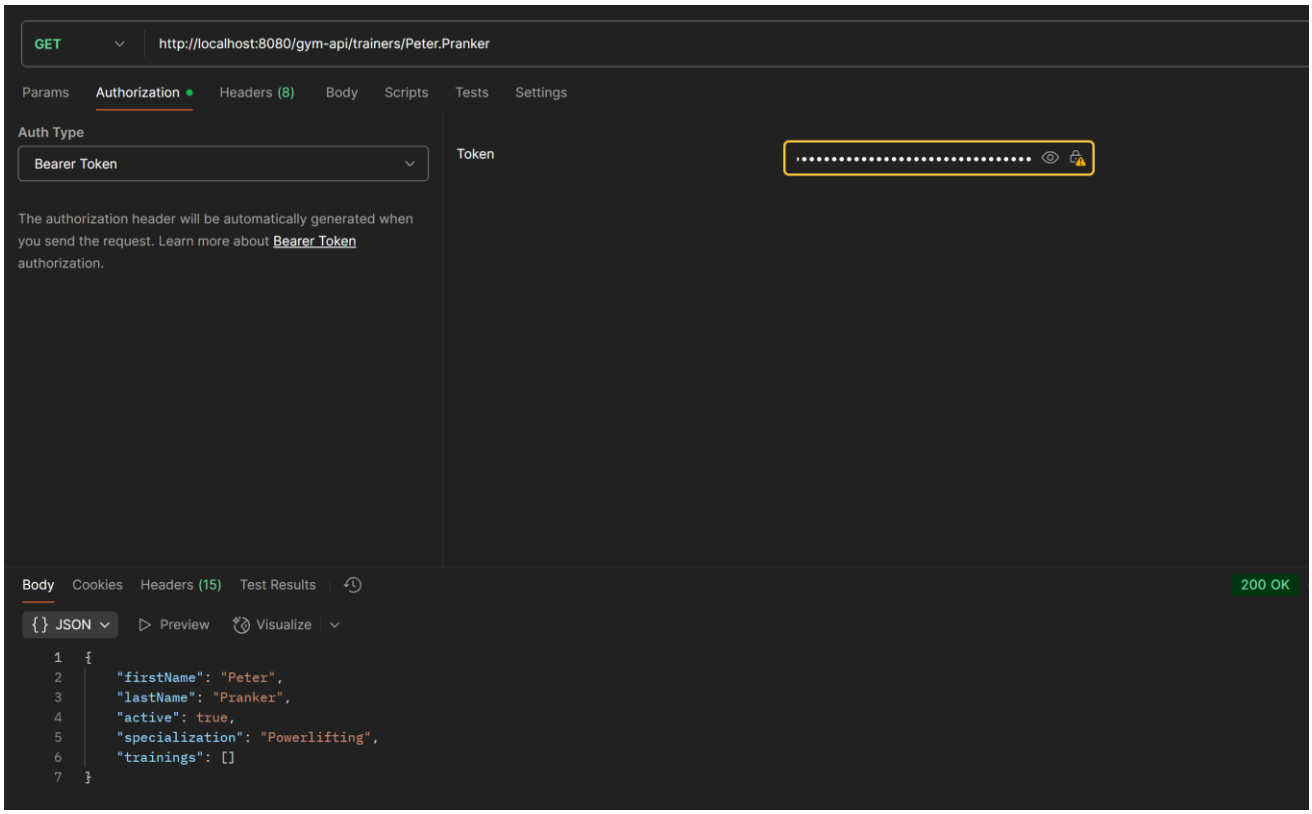


Рисунок 3.42 – результат отримання тренера після зміни статусу

До цього статус тренера був неактивний (false), тепер бачимо true. Перевіримо БД для запевнення, що дані дійсно приходять правильно та відповідають усім значенням у базі даних (див. рисунок 3.43):

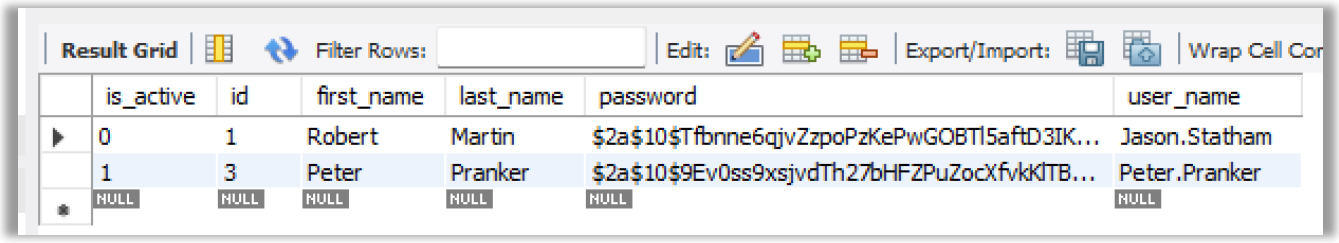


Рисунок 3.43 – результат змін у БД

Усі зміни також відповідають картині у базі даних. До цього статус був 0, що є цифровим представленням для булевого false значення.

Наступним кроком спробуємо оновити профіль тренера, але із зміною username (див. рисунок 3.44):

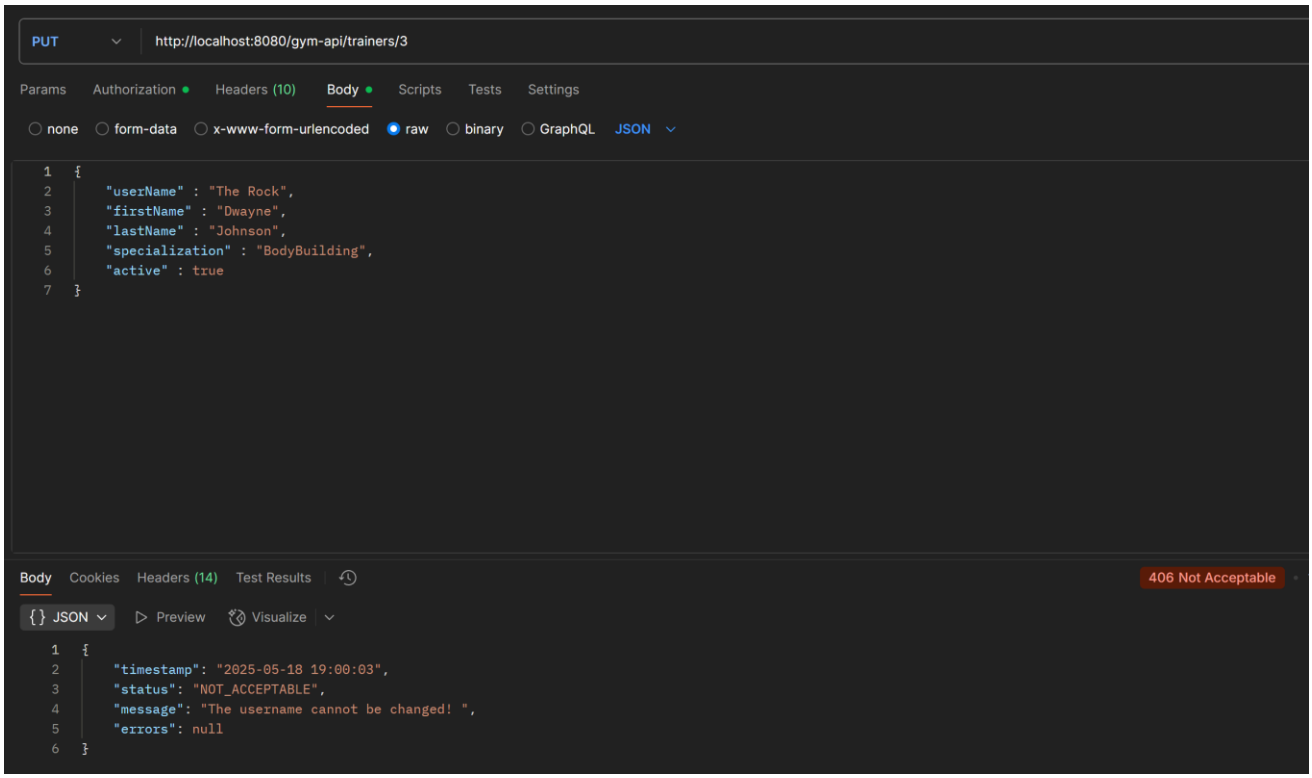


Рисунок 3.44– результат відмови сервера та надання повідомлення про проблему

Response: У нижній частині екрана видно відповідь сервера:

– Код статусу 406 Not Acceptable свідчить про те, що запит не може бути оброблений через те, що зміна імені користувача заборонена.

– У відповіді в форматі JSON вказано:

  – **timestamp**: "2025-05-18 19:00:03" – дата та час виникнення помилки.

  – **status**: "NOT_ACCEPTABLE" – статус помилки.

  – **message**: "The username cannot be changed!" – повідомлення, що пояснює причину відмови в обробці запиту.

  – **errors**: null – жодних додаткових помилок не виявлено.

Результатом є вже знайомий патерн поведінки, що попереджає нас про те, що юзернейм повинен залишатись без змін.

Тепер черга правильного виконання оновлення для профілю тренера за його ідентифікатором (див. рисунок 3.45):

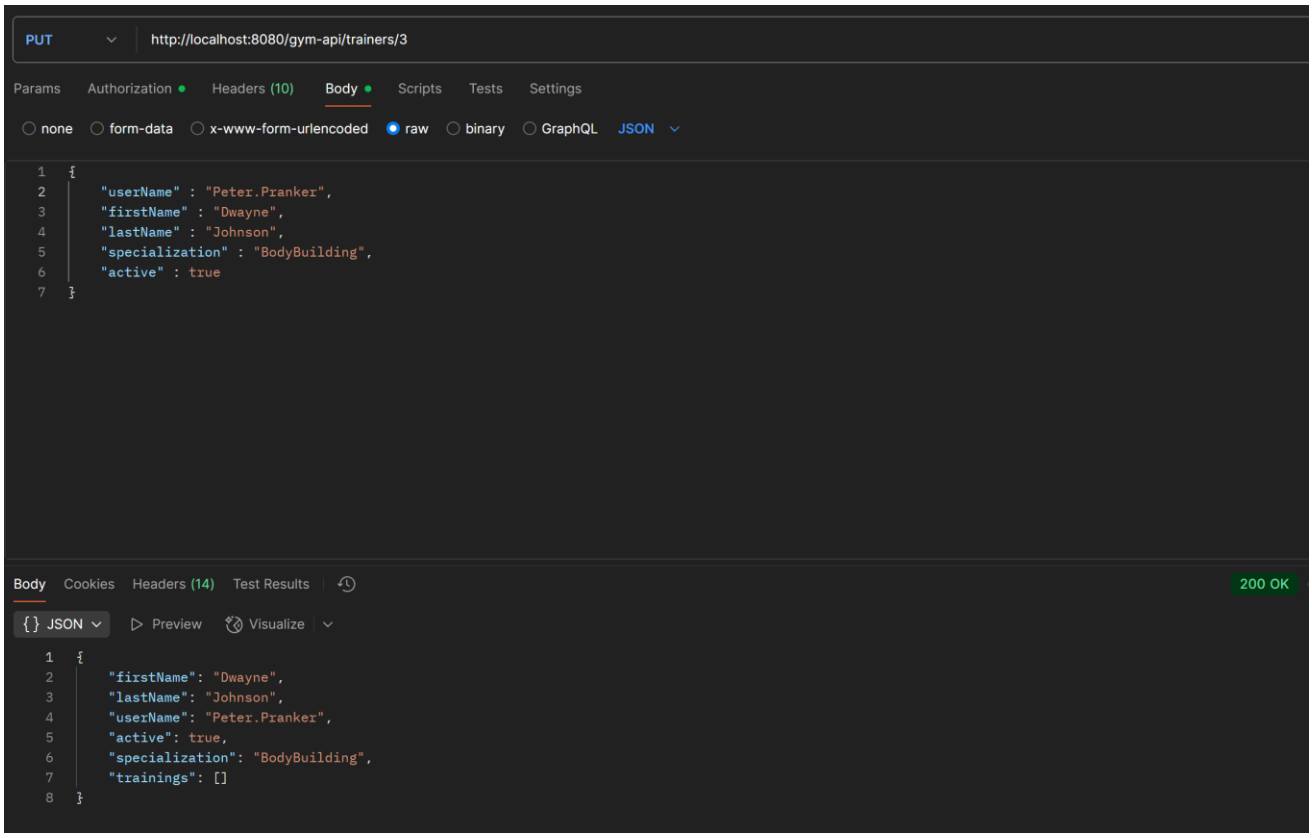


Рисунок 3.45 – результат успішного оновлення профіля тренера

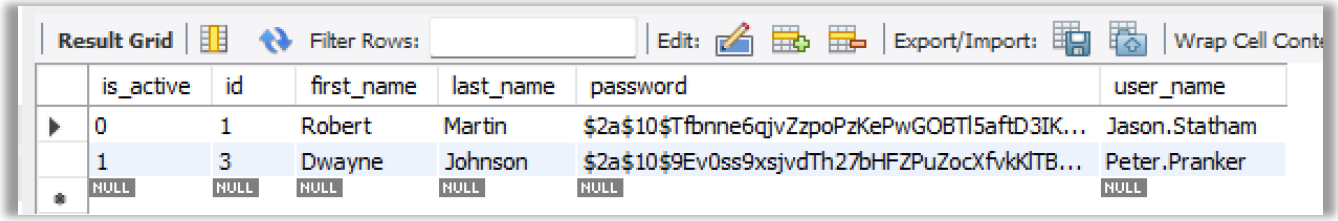


Рисунок 3.46 – результат змін у БД в таблиці users

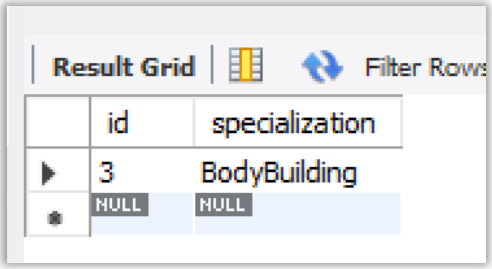


Рисунок 3.47 – результат змін у БД в таблиці trainers

Враховуючи, що ми вже маємо профілі тренера та відвідувача у БД, можемо виконати запит із отримки тренерів, які не мають відношення до певного відвідувача (див. рисунок 3.48):

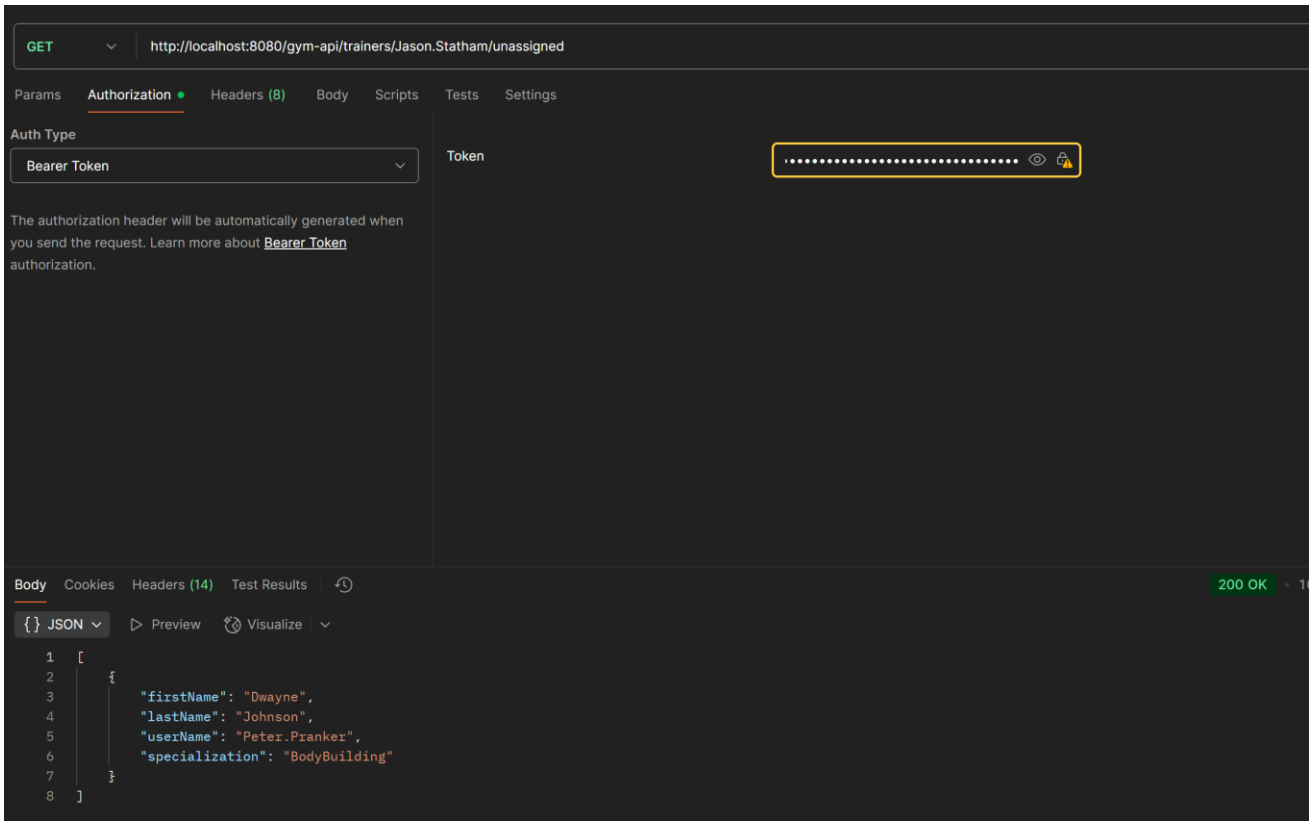


Рисунок 3.48– отримання тренерів, із якими у користувача не має тренеувань

Response: У нижній частині екрана видно, як API відповідає на запит:

- Код статусу 200 OK свідчить про успішне виконання запиту.
- У відповіді в форматі JSON представлено інформацію про тренера, яка включає:
  - **firstName**: "Dwayne" – ім'я тренера.
  - **lastName**: "Johnson" – прізвище тренера.
  - **userName**: "Pete.Pranker" – ім'я користувача тренера.
  - **specialization**: "BodyBuilding" – спеціалізація тренера.

Даний запит і відповідь демонструють, що система здатна повертати інформацію про тренера, навіть якщо у нього немає активних сесій або призначень. У цьому випадку, оскільки в базі даних зареєстровано лише одного тренера, API повертає саме його дані.

Настав час створити тренування та закріпити його за нашим тренером та відвідувачем. Спробуємо виконати запит без токену (див. рисунок 3.49):

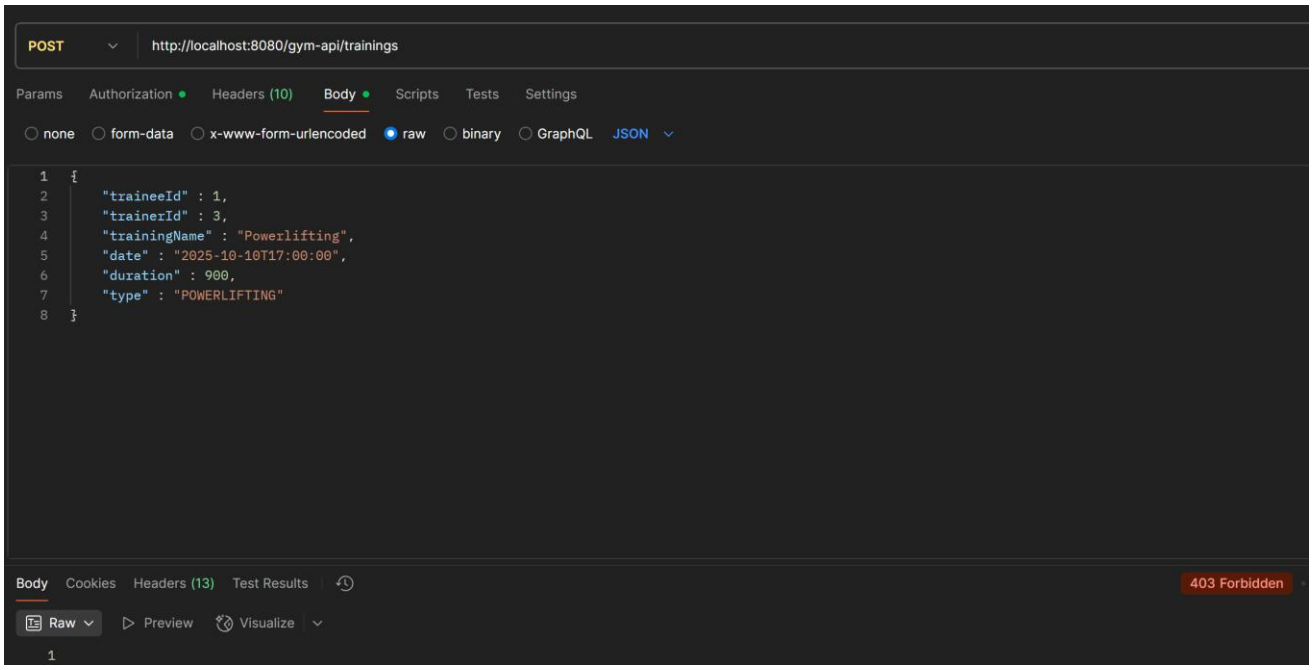


Рисунок 3.49 – відмова у доступі при спробі створенні тренування

Як і усі інші запити, які не відносяться до реєстрації або ж логіну, ми не можемо виконати цей запит без надання дійсного токену доступу. Виконаймо цей запит із автентифікацією (див. рисунок 3.50):

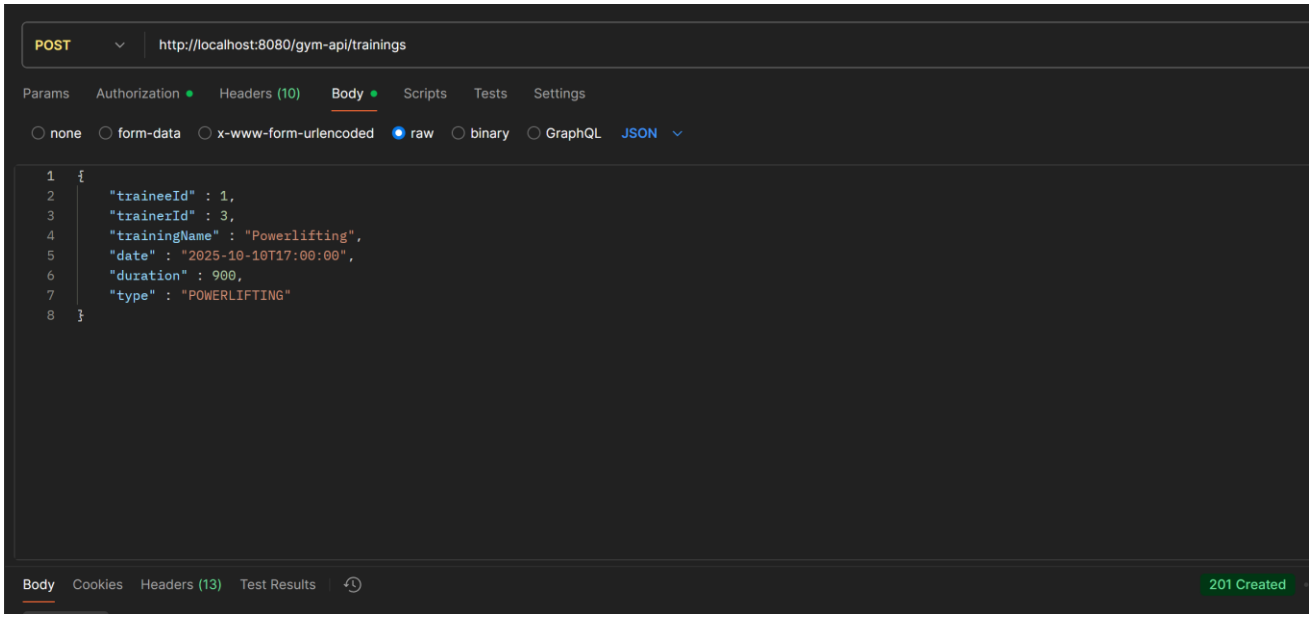


Рисунок 3.50 – результат успішного створення тренування

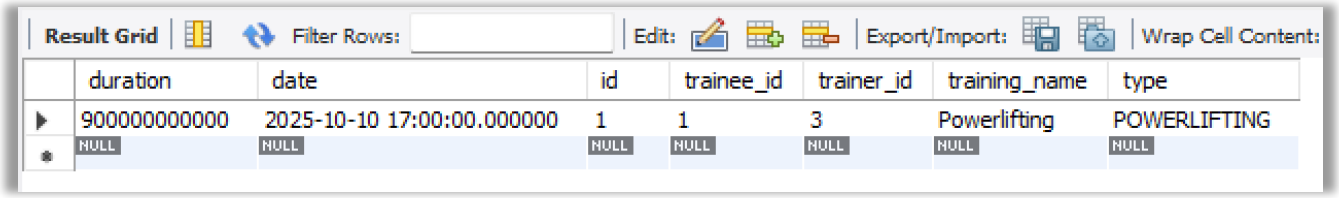


Рисунок 3.51 – результат успішного створення тренування у БД в таблиці trainings

Реалізований функціонал надає змогу побачити усі типи тренувань та їх порядковий номер, які присутні у БД на даний момент (див. рисунок 3.52):

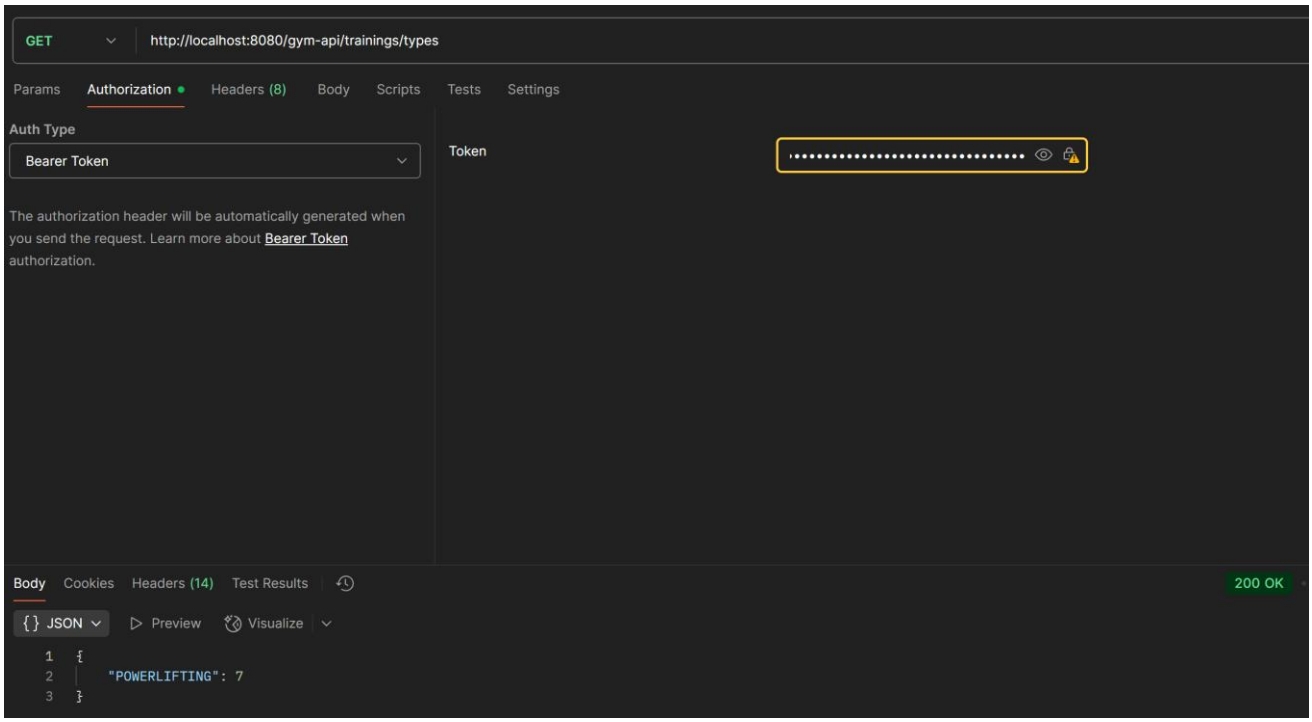
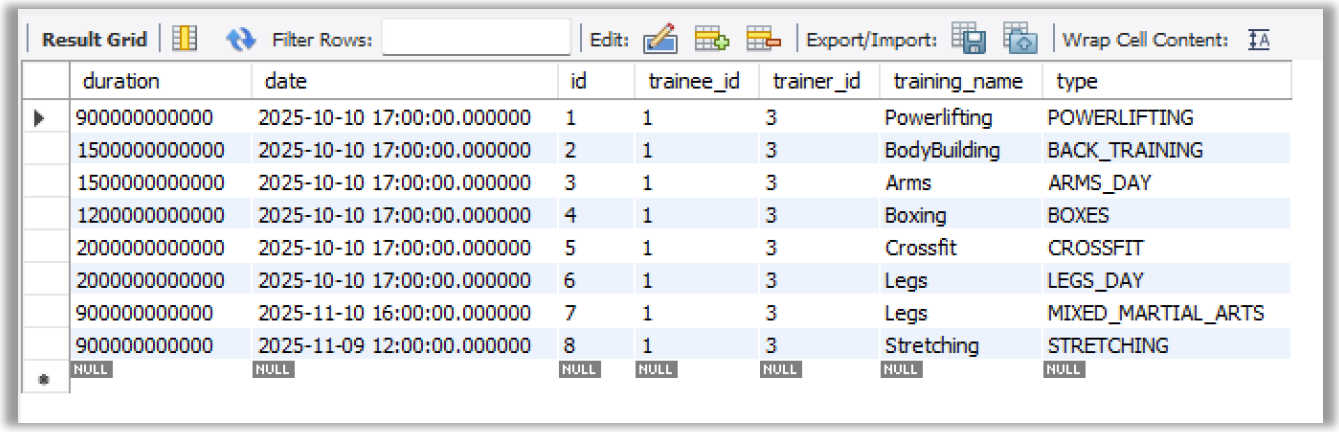


Рисунок 3.52 – результат отримання наявних типів тренувань

Для демонстрації наступних запитів створимо ще декілька тренувань для тих самих користувачів, що наразі є в нашій системі.

Результат виконання запитів на створення можна побачити у БД. Виконаємо запит до бази даних, щоб переконатись в створенні нових тренувань для існуючих юзерів (див. рисунок 3.53):



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

duration	date	id	trainee_id	trainer_id	training_name	type
900000000000	2025-10-10 17:00:00.000000	1	1	3	Powerlifting	POWERLIFTING
1500000000000	2025-10-10 17:00:00.000000	2	1	3	BodyBuilding	BACK_TRAINING
1500000000000	2025-10-10 17:00:00.000000	3	1	3	Arms	ARMS_DAY
1200000000000	2025-10-10 17:00:00.000000	4	1	3	Boxing	BOXES
2000000000000	2025-10-10 17:00:00.000000	5	1	3	Crossfit	CROSSFIT
2000000000000	2025-10-10 17:00:00.000000	6	1	3	Legs	LEGS_DAY
900000000000	2025-11-10 16:00:00.000000	7	1	3	Legs	MIXED_MARTIAL_ARTS
900000000000	2025-11-09 12:00:00.000000	8	1	3	Stretching	STRETCHING
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 3.53 – тренування у таблиці trainings

Подивимось на запит із отримання усіх існуючих типів тренувань у БД після створення нових тренувань. Для цього виконаємо запит й подивимось список існуючих тренувань ще раз (див. рисунок 3.54):

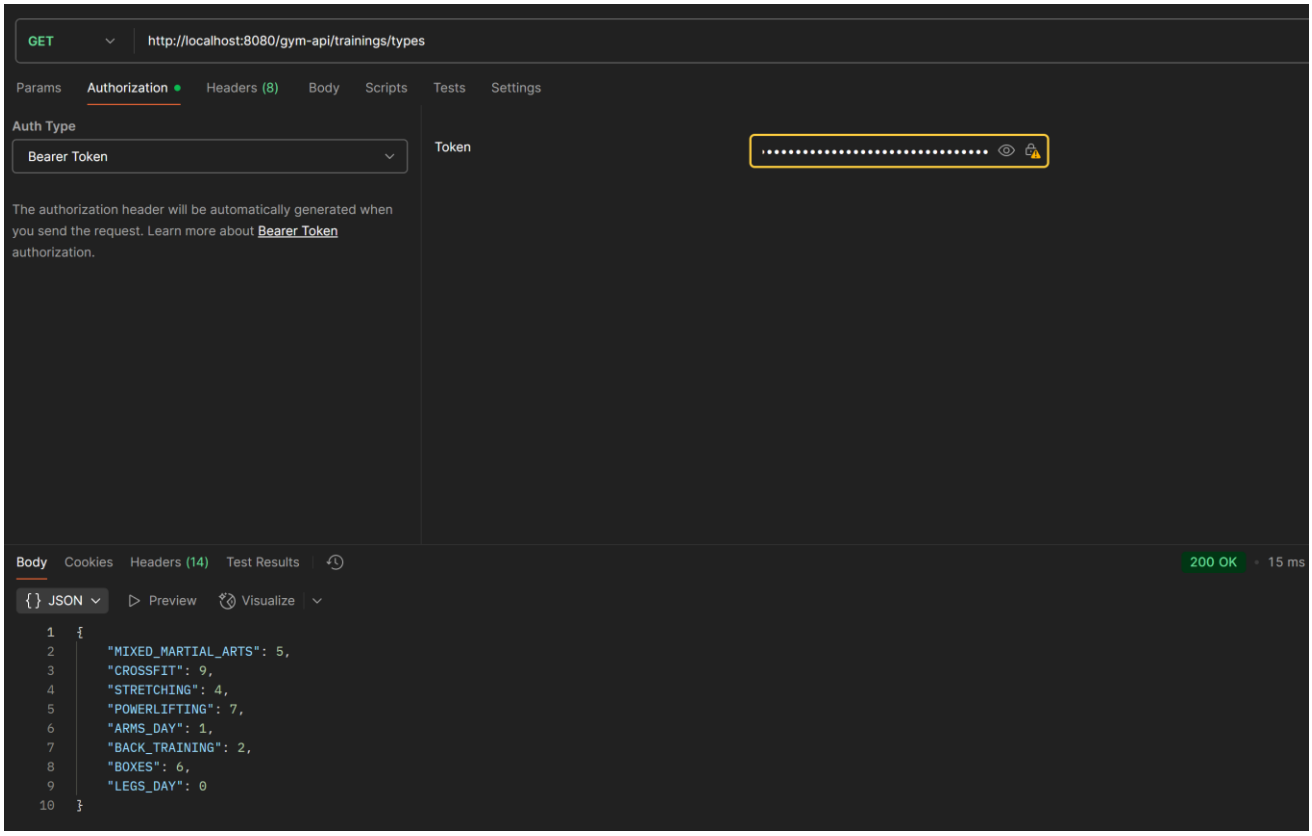


Рисунок 3.54 – результат отримання наявних типів тренувань

Результат отримання типів усіх створених тренувань, які є наявні у БД та їх порядковий номер.

Також підтримується отримання тренувань за фільтрацією по юзеру, тренеру, типу тренування та діапазоном дат і часу (див. рисунок 3.55):

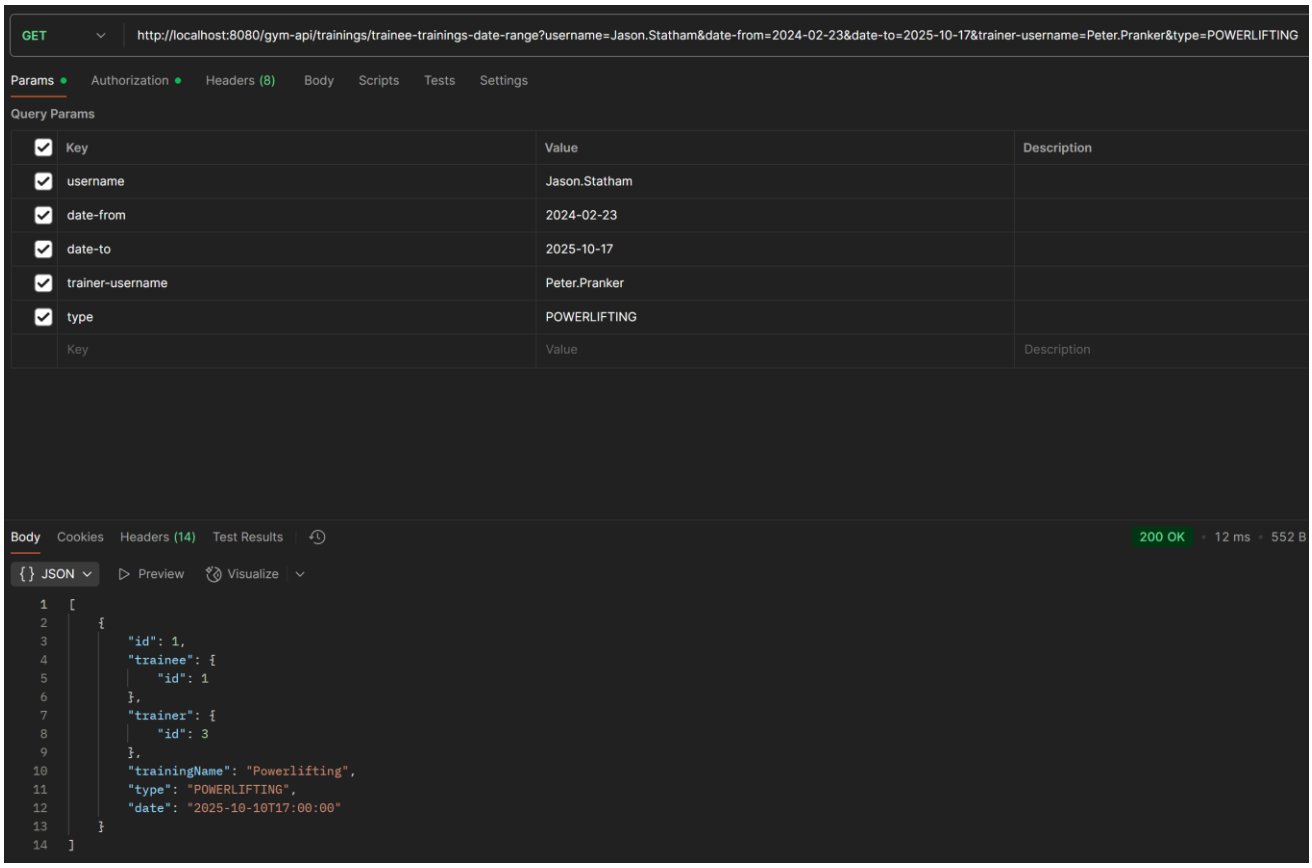
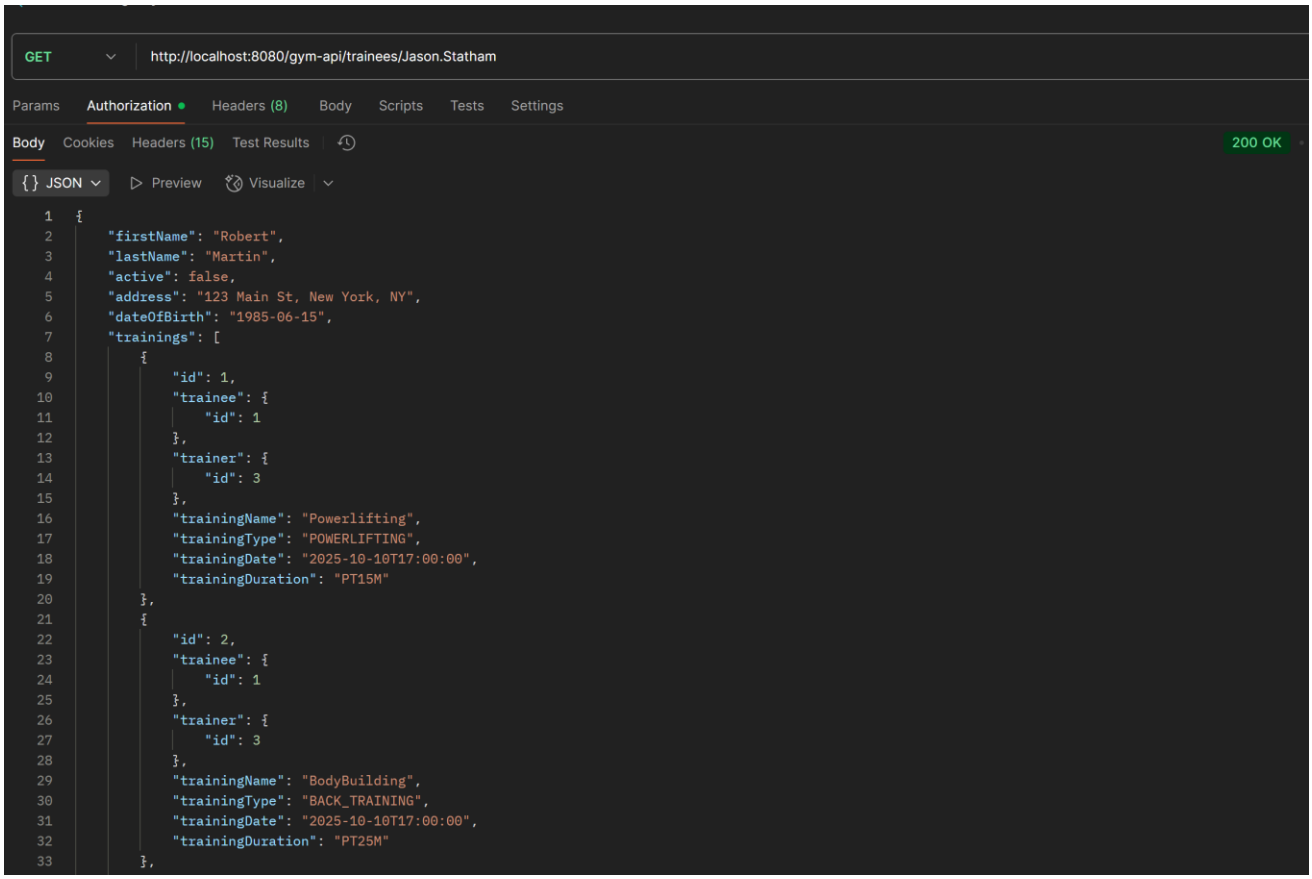
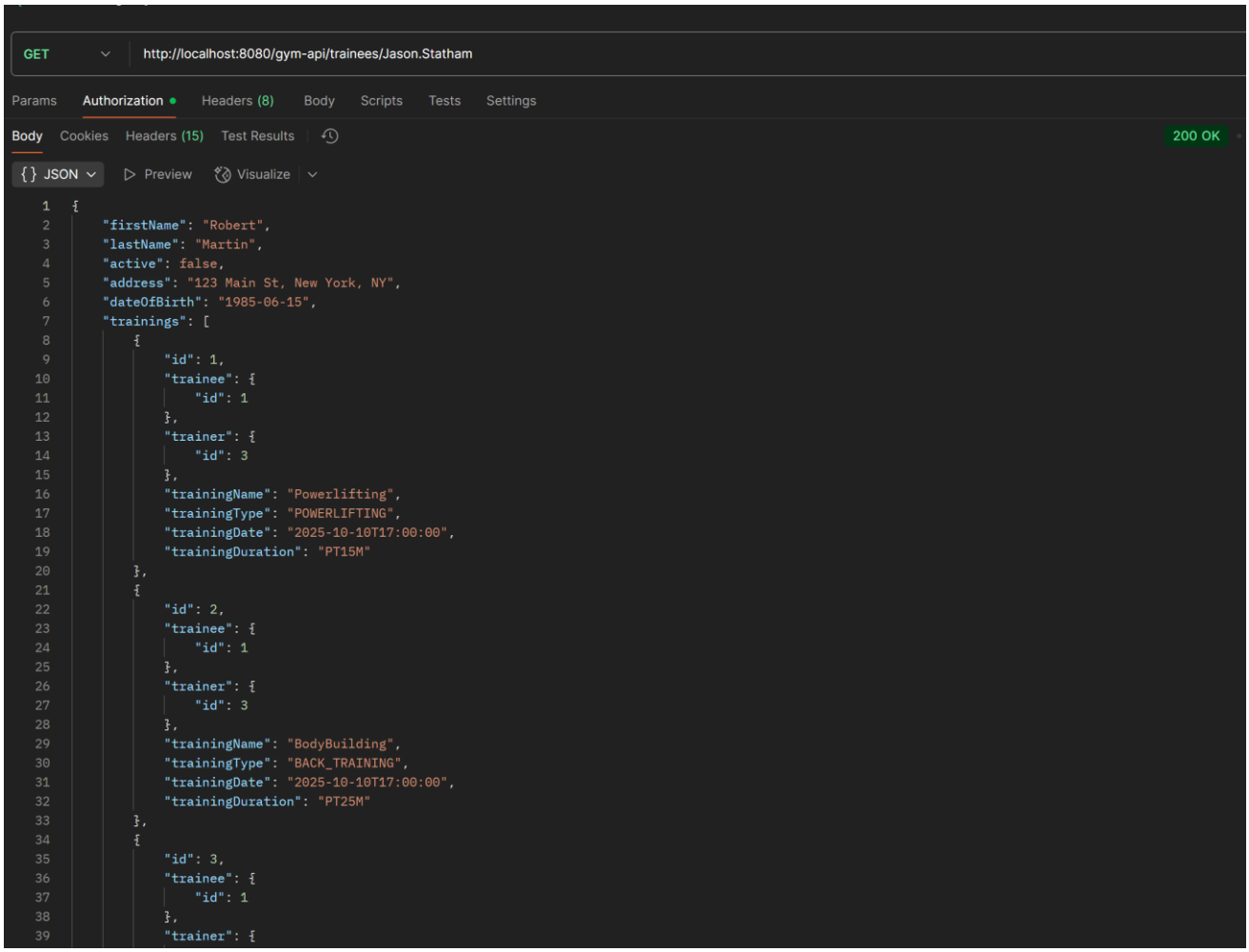


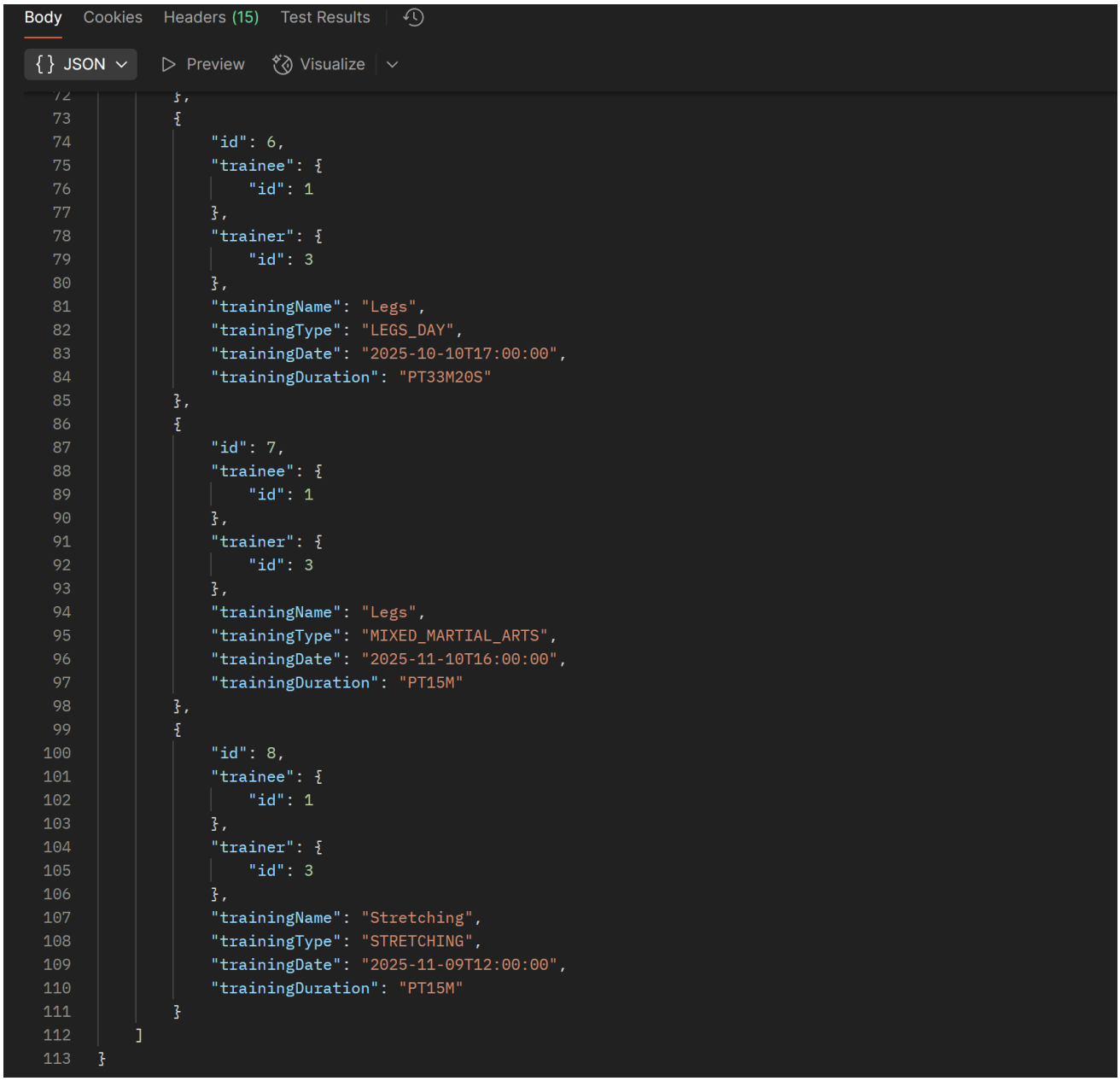
Рисунок 3.55– отримання тренувань за фільтрами

Даний запит і відповідь демонструють можливість фільтрації даних про тренування клієнтів за конкретними критеріями, такими як ім'я користувача, дати проведення тренувань, ім'я тренера та тип тренування. Це забезпечує зручність для користувачів, які прагнуть отримати детальну інформацію про свої тренування в певний період часу. Наявність авторизації через токен підтверджує безпеку доступу до даних, що є важливим аспектом у системах управління тренуваннями. У даному прикладі тільки одне тренування задовольняє певній фільтрації.

На останок спробуємо змінити список тренувань для нашого відвідувача. Зараз його список тренувань виглядає наступним чином (див. рисунки 3.56):


GET
http://localhost:8080/gym-api/trainees/Jason.Statham
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Body
Cookies
Headers (15)
Test Results
200 OK
JSON
Preview
Visualize
{
"firstName": "Robert",
"lastName": "Martin",
"active": false,
"address": "123 Main St, New York, NY",
"dateOfBirth": "1985-06-15",
"trainings": [
{
"id": 1,
"trainee": {
"id": 1
},
"trainer": {
"id": 3
},
"trainingName": "Powerlifting",
"trainingType": "POWERLIFTING",
"trainingDate": "2025-10-10T17:00:00",
"trainingDuration": "PT15M"
},
{
"id": 2,
"trainee": {
"id": 1
},
"trainer": {
"id": 3
},
"trainingName": "BodyBuilding",
"trainingType": "BACK_TRAINING",
"trainingDate": "2025-10-10T17:00:00",
"trainingDuration": "PT25M"
},


GET
http://localhost:8080/gym-api/trainees/Jason.Statham
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Body
Cookies
Headers (15)
Test Results
200 OK
JSON
Preview
Visualize
{
"firstName": "Robert",
"lastName": "Martin",
"active": false,
"address": "123 Main St, New York, NY",
"dateOfBirth": "1985-06-15",
"trainings": [
{
"id": 1,
"trainee": {
"id": 1
},
"trainer": {
"id": 3
},
"trainingName": "Powerlifting",
"trainingType": "POWERLIFTING",
"trainingDate": "2025-10-10T17:00:00",
"trainingDuration": "PT15M"
},
{
"id": 2,
"trainee": {
"id": 1
},
"trainer": {
"id": 3
},
"trainingName": "BodyBuilding",
"trainingType": "BACK_TRAINING",
"trainingDate": "2025-10-10T17:00:00",
"trainingDuration": "PT25M"
},
{
"id": 3,
"trainee": {
"id": 1
},
"trainer": {



Рисунки 3.56 – список тренувань для користувача за username

На рисунках 3.56 ми бачимо список усіх тренувань наявних для певного користувача, які ми створили раніше.

Тепер виконаємо запит, який призначить новий список тренувань для користувача. При цьому ідентифікатори користувача та тренера, повинні бути валідними. Після виконання даної операції, усі старі тренування, які не були включені у список, будуть видалені із бази даних.

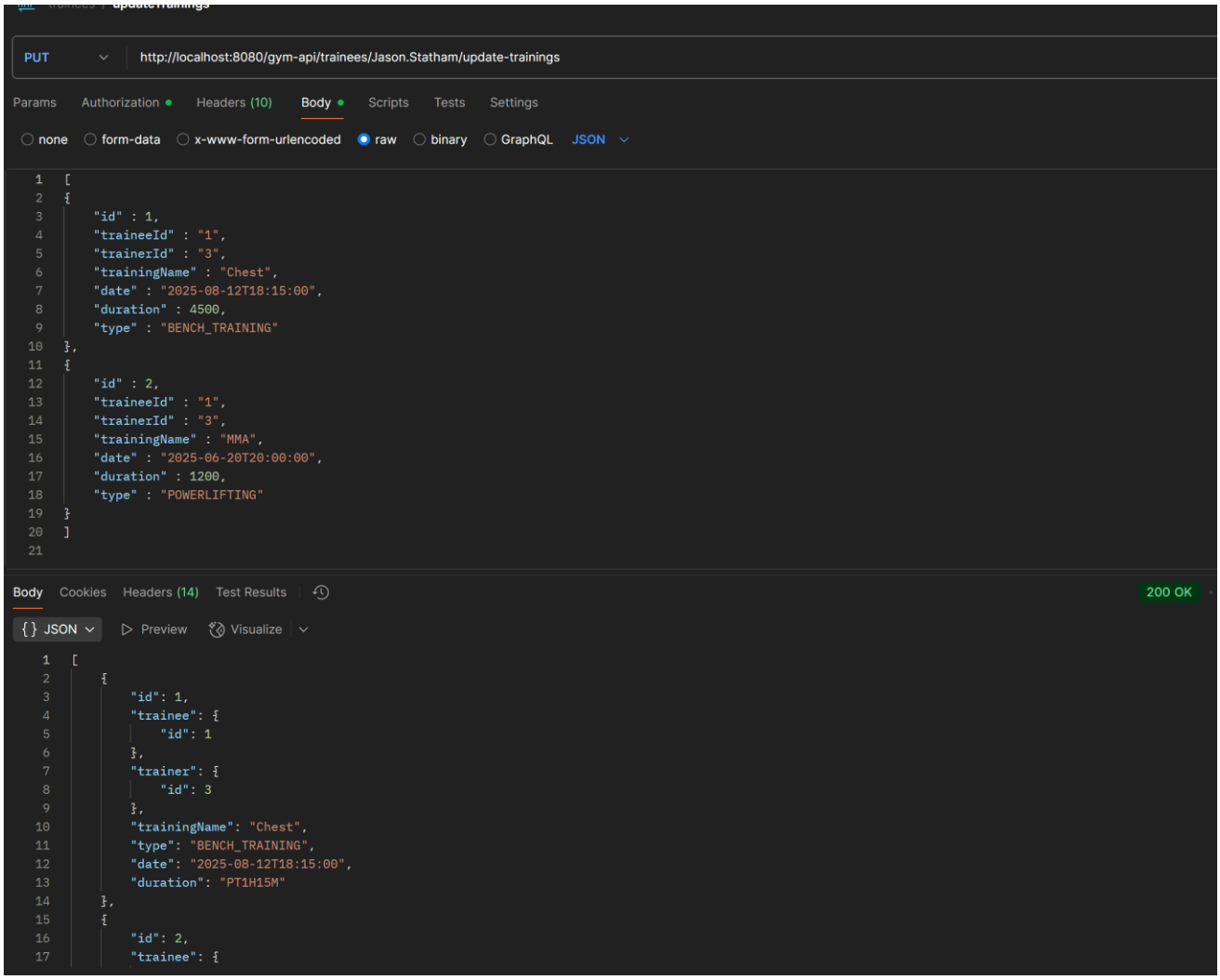


Рисунок 3.57 – оновлення списку тренувань для користувача

В результаті маємо оновлений список тренувань для юзера. Та якщо ми подивимось у БД, старих тренувань там більше не існує (див. рисунок 3.58):

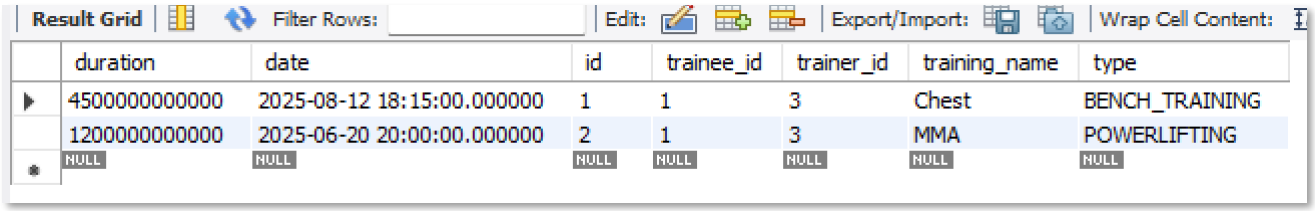


Рисунок 3.58 – Усі тренування у БД в таблиці trainings

Оскільки в системі зареєстровані лише один користувач і один тренер, тренування були створено саме для них. Після оновлення в відповіді відображається актуалізований перелік тренувань для цього єдиного клієнта.

Після демонстрації основного функціоналу, створений користувач нам вже не потрібен. Ми можемо жорстоко видалити його профіль за допомогою наступного запиту (див. рисунок 3.59):

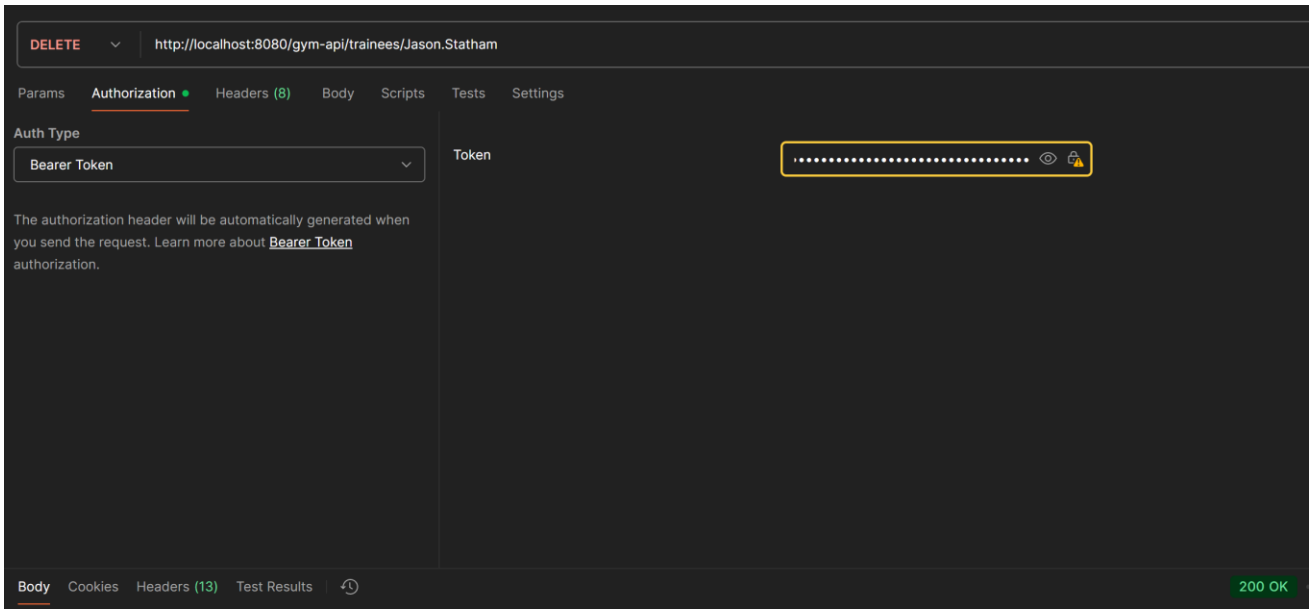


Рисунок 3.59 – Результат видалення юзера

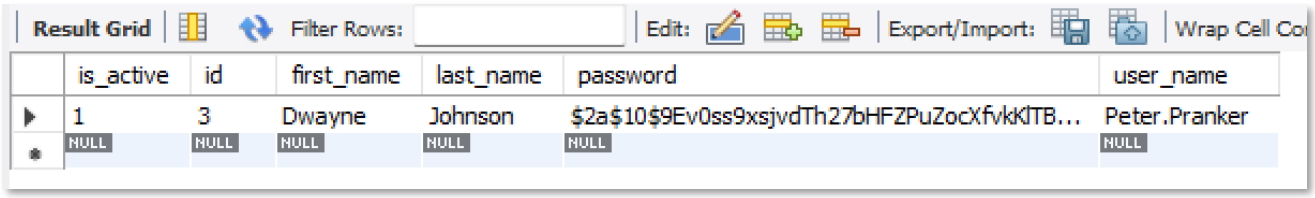


Рисунок 3.60 – Результат видалення юзера із БД в таблиці users

Оскільки нашого юзера вже не існує, тренування, що були закріплені за цим користувачем, автоматично видаляються із БД (див. рисунок 3.61):

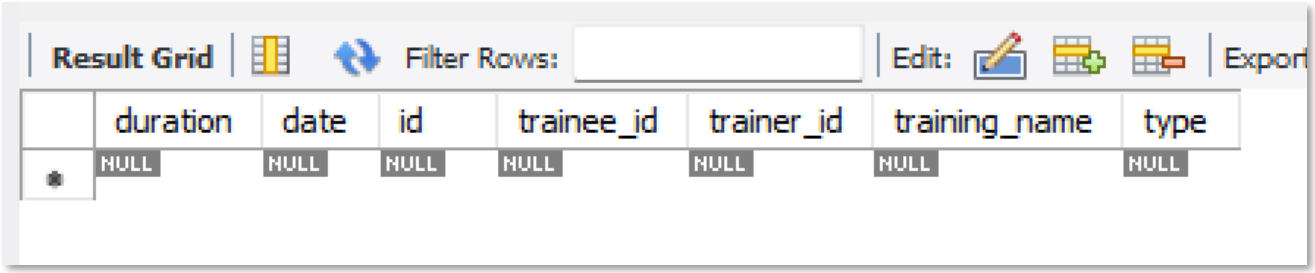


Рисунок 3.61 – Усі тренування у БД в таблиці trainings

Як ми можемо бачити, таблиця тренувань порожня, оскільки нашого єдиного юзера більше не існує, відповідно усі його тренування видалилися.

ВИСНОВКИ

У ході виконання дипломного проекту була розроблена CRM-система для управління тренуваннями, користувачами та тренерами, яка відповідає сучасним вимогам фітнес-індустрії. Система забезпечує високий рівень надійності, безпеки та продуктивності. Створений API-інтерфейс на основі Spring Boot дозволяє оперативно реєструвати та автентифікувати користувачів, управляти їхніми даними та історією тренувань, а також здійснювати розподіл і облік занять між тренерами.

Серед досягнутого варто виділити:

Повноцінна авторизація та аутентифікація: Реалізовано механізм реєстрації нових користувачів із верифікацією даних і захистом паролів відповідно до сучасних вимог безпеки. Забезпечено видачу токенів доступу з підтримкою коду відповіді HTTP 202, що підтверджує успішний процес входу.

Управління даними користувачів і тренерів: Створено структуру сутностей User та Trainer з необхідними полями для зберігання персональних даних та профілів. Розроблено сервіси для додавання, оновлення та видалення записів у базі даних із використанням Hibernate та Spring Data JPA.

CRUD-операції для тренувань: Запроваджено можливість створення, перегляду та оновлення тренувань з урахуванням доступності тренера та користувача в базі даних. Вбудовано механізм фільтрації тренувань за датами, тренером та типом, що дозволяє отримувати актуальні списки для будь-якого користувача.

Безпека й надійність: Забезпечено захист даних від несанкціонованого доступу за допомогою механізмів аутентифікації та ролей. Використано валідацію запитів і шаблонів введення (password pattern), що знижує ризик введення некоректних або небезпечних даних.

Готовність до масштабування: Архітектура API передбачає можливість міграції на хмарні сервіси AWS, включаючи Lambda для обробки запитів, DynamoDB для зберігання даних, S3 для зберігання статичних ресурсів і Cognito

для управління користувачами. Модульна структура проекту спрощує внесення нових функціональних блоків і розподіл навантаження.

Науково-практична новизна та значущість

Запропонована CRM-система відрізняється інтегрованим підходом до управління тренуваннями й користувачами, що суттєво спрощує бізнес-процеси фітнес-клубів. Використання сучасних фреймворків Java та орієнтація на мікросервісну структуру забезпечують гнучкість та адаптивність рішення до змінних вимог ринку. Результати роботи можуть стати основою комерційних продуктів або ядром уніфікованої платформи для мережевих фітнес-закладів.

Обмеження та подальші перспективи

Хоча реалізоване API забезпечує міцну функціональну базу, проект наразі не має користувацького інтерфейсу (UI), що обмежує його безпосереднє застосування кінцевими користувачами. У подальших роботах доцільно:

- Розробити фронтенд-застосунок (веб або мобільний) із використанням сучасних бібліотек (React, Vue.js або Angular).
- Додати функціонал звітності та аналітики, інтегрувавши системи збору статистики та візуалізації ключових показників.
- Розширити набір ролей і прав доступу, включивши адміністративні та маркетингові модулі.

У цілому, розроблена CRM-система відповідає поставленим цілям дипломного проекту. Вона забезпечує безпечне та ефективне керування даними користувачів, тренерів і тренувань, надає гнучкі можливості для масштабування та адаптації до хмарної інфраструктури AWS. Отримане рішення є міцним технічним фундаментом для подальшого розвитку, інтеграції з іншими сервісами та створення повноцінного продукту, що сприятиме підвищенню ефективності бізнес-процесів фітнес-індустрії та збільшенню прибутковості компаній.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джос М. С. Вдосконалення механізму управління закладів фітнес-індустрії: магістерська робота. 2021. URL: https://dspace.znu.edu.ua/jspui/handle/12345/5117 (дата звернення: 02.02.2025).

2. Пригода А. Мікросервісна архітектура, як основа для створення CRM-системи. Наука і техніка сьогодні. 2024. URL: https://doi.org/10.52058/2786-6025-2024-8(36)-1140-1151 (дата звернення: 08.02.2025).

3. Ралік І. Адаптивна CRM-система на основі когнітивного аналізу клієнтської поведінки. Наука і техніка сьогодні. 2025. URL: https://doi.org/10.52058/2786-6025-2025-3(44)-1436-1447 (дата звернення: 10.02.2025).

4. Що таке CRM-система та як вона працює? Creatio. URL: https://www.creatio.com/ua/crm/what-is-crm (дата звернення: 14.02.2025).

5. CRM для інтернет-магазину, продажу товарів - SalesDrive. URL: https://salesdrive.ua/?utm_source=google&utm_medium=cpc&utm_campaign=crm&gad_source=1&gad_campaignid=1042140871&gclid=EAIaIQobChMIurLNiaK8jQMVyw-iAx34QBCqEAAYAyAAEgKZm_D_BwE (дата звернення: 14.02.2025).

6. Sport Life – мережа фітнес-клубів №1 в Україні. URL: https://sportlife.ua/uk/ (дата звернення: 17.03.2025).

7. Total Fitness – Мережа фітнес клубів. URL: https://totalfitness.com.ua (дата звернення: 17.03.2025).

8. Zmerzlyi I. Мікросервісна архітектура. Medium. URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d (дата звернення: 22.03.2025).

9. Фреймворк Spring та його особливості. Highload.tech - медіа для розробників. URL: https://highload.tech/uk/frejmvork-spring-ta-jogo-osoblivosti/ (дата звернення: 23.03.2025).

10. Шелудченко М. Д., Астістова Т. І. Дослідження та розробка математичного забезпечення для мережевої системи управління контентом з

використанням мови програмування Java на основі технологій Spring та Hibernate. URL: https://er.knutd.edu.ua/handle/123456789/5167 (дата звернення: 19.03.2025).

11. itProger – що таке Hibernate? URL: https://itproger.com/ua/spravka/java/hibernate (дата звернення: 29.03.2025).

12. Baeldung – Spring MVC. URL: https://www.baeldung.com/spring-mvc-model-model-map-model-view (дата звернення: 20.04.2025).

13. JavaRush – Spring Security. URL: https://javarush.com/quests/lectures/questspringsecurity.level01.lecture00 (дата звернення: 02.04.2025).

14. Habr. Регистрация и авторизация с помощью Spring Security на примере простого приложения. URL: https://habr.com/ru/articles/482552/ (дата звернення: 15.04.2025).

15. Habr. JWT-аутентификация при помощи Spring Boot 3 и Spring Security 6. URL: https://habr.com/ru/articles/784508/ (дата звернення: 15.04.2025).

16. Medium. Spring Security Guide. Part 1: Introduction. URL: https://medium.com/@ihor.polataiko/spring-security-guide-part-1-introduction-c2709ff1bd98 (дата звернення: 10.04.2025).

17. Medium. Implement JWT authentication in a Spring Boot 3 application. URL: https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac (дата звернення: 26.04.2025).

18. Бачинський Д. Що таке CRM та як обрати її для вашого бізнесу. *Блог NetHunt*. URL: https://nethunt.ua/blog/shcho-takie-crm-sistiema-povnii-ghid-po-viboru-crm-dlia-pochatkivtsiv/ (дата звернення: 19.03.2025).

19. HostIQ. Найкращі CRM для бізнесу: ТОП-16. URL: https://hostiq.ua/blog/ukr/best-crm-systems/ (дата звернення: 19.03.2025).

20. Baeldung. The Spring @Controller and @RestController Annotations. URL: https://www.baeldung.com/spring-controller-vs-restcontroller (дата звернення: 02.04.2025).

21. Spring. Securing a Web Application. URL:https://spring.io/guides/gs/securing-web (дата звернення: 03.04.2025).

22. Baeldung. Hibernate Inheritance Mapping. URL:https://www.baeldung.com/hibernate-inheritance (дата звернення: 25.03.2025).

23. JavaRush – Never cache data in Hibernate. URL:https://javarush.com/quests/lectures/questhibernate.level14.lecture04 (дата звернення: 25.03.2025).

24. JavaRush – Розбираємо бази даних та мову SQL. (Частина 2) - "Java-проект від А до Я". URL: https://javarush.com/ua/groups/posts/uk.2957.rozbiramo-bazi-danikh-ta-movu-sql-chastina-2---java-proekt-vd-a-do-ja (дата звернення: 27.03.2025).

25. StackOverflow – What are the possible values of the Hibernate hbm2ddl.auto configuration and what do they do. URL: https://stackoverflow.com/questions/438146/what-are-the-possible-values-of-the-hibernate-hbm2ddl-auto-configuration-and-wha (дата звернення: 28.03.2025).

26. StackOverflow – How to print a query string with parameter values when using Hibernate? URL: https://stackoverflow.com/questions/1710476/how-to-print-a-query-string-with-parameter-values-when-using-hibernate (дата звернення: 28.03.2025).

27. StackOverflow – JPA JoinColumn vs mappedBy. URL:https://stackoverflow.com/questions/11938253/jpa-joincolumn-vs-mappedby (дата звернення: 29.03.2025).

28. StackOverflow – PersistentObjectException: detached entity passed to persist thrown by JPA and Hibernate. URL: https://stackoverflow.com/questions/13370221/persistentobjectexception-detached-entity-passed-to-persist-thrown-by-jpa-and-h (дата звернення: 05.04.2025).

29. StackOverflow – Difference between Role and GrantedAuthority in Spring Security. URL: https://stackoverflow.com/questions/19525380/difference-between-role-and-grantedauthority-in-spring-security (дата звернення: 06.04.2025).

30. StackOverflow – Unit testing with Spring Security. URL: https://stackoverflow.com/questions/360520/unit-testing-with-spring-security

(дата звернення: 07.04.2025).

31. StackOverflow – How Spring Security Filter Chain works? URL: https://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works (дата звернення: 07.04.2025)

32. DOU. Що таке Domain-Driven Design та на якому етапі варто його впроваджувати в продукт? URL: https://dou.ua/forums/topic/39874/ (дата звернення: 04.03.2025)

33. JavaRush – Підхід MVC. URL:https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture02 (дата звернення: 05.03.2025)

34. AWS. Event-Driven Architecture. URL: https://aws.amazon.com/ru/event-driven-architecture/#:~:text=An%20event-driven%20architecture%20uses,on%20an%20e-commerce%20website (дата звернення: 06.03.2025)

ДОДАТОК А

Частина контролерів

Додаток повинен дозволяти користувачам і тренерам зареєструвати свій профіль, надаючи тим, хто тренується, можливість вибрати одного або кількох тренерів:

Користувач:

```
Yura-Moroz
@Tag(name = "Trainee Controller", description = "Endpoints for managing trainee profiles")
@RequestMapping("/gym-api/trainees")
@RestController
@Validated
@RequiredArgsConstructor
public class TraineeController {

    private final TraineeService traineeService;

    private final ConversionService conversionService;

    private final MeterRegistry meterRegistry;


    Yura-Moroz
    @Operation(summary = "Create a new trainee profile")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "Profile created successfully",
                    content = {@Content(mediaType = "application/json",
                            schema = @Schema(implementation = TraineeDto.class))}),
            @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
            @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
    })
    @PostMapping
    @JsonView(TraineeViews.Login.class)
    public ResponseEntity<TraineeDto> createProfile(
            @Parameter(description = "Trainee details for creation", required = true)
            @RequestBody
            @Valid
            @JsonView(TraineeViews.Input.class) TraineeDto traineeDto) {

        meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "POST /gym-api/trainees").increment();

        return ResponseEntity.status(HttpStatus.CREATED)
                .body(conversionService.convert(traineeService.save(traineeDto), TraineeDto.class));
    }
```

Контролер Trainee та метод createProfile() для створення профіля користувача

Тренер:

```
Yura-Moroz
@Tag(name = "Trainer Controller", description = "Endpoints for managing trainer profiles")
@RequestMapping("/gym-api/trainers")
@RestController
@Validated
@RequiredArgsConstructor
public class TrainerController {

    private final TrainerService trainerService;

    private final ConversionService conversionService;

    private final MeterRegistry meterRegistry;

    Yura-Moroz
    @Operation(summary = "Create a new trainer profile")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "Profile created successfully",
                    content = {@Content(mediaType = "application/json",
                            schema = @Schema(implementation = TrainerDto.class))}),
            @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
            @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
    })
    @PostMapping
    @JsonView(TrainerViews.Login.class)
    public ResponseEntity<TrainerDto> createProfile(
            @Parameter(description = "Trainer details for creation", required = true)
            @RequestBody @Valid
            @JsonView(TrainerViews.Input.class) TrainerDto trainerDto) {

        meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "POST /gym-api/trainers").increment();

        return ResponseEntity.status(HttpStatus.CREATED).
                body(conversionService.convert(trainerService.save(trainerDto), TrainerDto.class));
    }
```

Контролер Trainer та метод createProfile() для створення профіля тренера

Додаток також має забезпечувати функціональність для зміни інформації профілю та активації або деактивації профілів:

Для профілю тренера:

```
Yura-Moroz
@Operation(summary = "Update an existing trainer profile")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Profile updated successfully",
                content = {@Content(mediaType = "application/json",
                        schema = @Schema(implementation = TrainerDto.class))}),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PutMapping("/{id}")
@JsonView(TrainerViews.UpdatingResp.class)
public ResponseEntity<TrainerDto> updateProfile(
        @Parameter(description = "ID of the trainer to update", required = true)
        @PathVariable long id,
        @Parameter(description = "Updated trainee details", required = true)
        @RequestBody
        @JsonView(TrainerViews.UpdatingReq.class)
        @Valid TrainerDto trainerUpdatingDto) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "PUT /gym-api/trainers/{id}").increment();

    return ResponseEntity.status(HttpStatus.OK)
            .body(conversionService.convert(trainerService.update(id, trainerUpdatingDto), TrainerDto.class));
}
```

метод updateProfile() для оновлення профіля тренера

```
Yura-Moroz
@Operation(summary = "Toggle the active status of a trainee")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Status toggled successfully", content = @Content),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PatchMapping("/status")
public ResponseEntity<Void> changeStatus(
        @Parameter(description = "Trainer details with username for status update", required = true)
        @RequestBody @Valid
        @JsonView(TrainerViews.Status.class)
        TrainerDto trainerDto) {

    meterRegistry.counter( name: "endpoint.trainer.calls",  ...tags: "endpoint", "PATCH /gym-api/trainers/status").increment();

    Trainer trainer = trainerService.getByUsername(trainerDto.getUserName()).get();
    if (trainer.isActive()) trainerService.deactivate(trainer);
    else trainerService.activate(trainer);

    return ResponseEntity.status(HttpStatus.OK).build();
}
```

метод changeStatus() для зміни статусу профіля тренера

```
Yura-Moroz
@Operation(summary = "Change password for a trainer")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Password changed successfully", content = @Content),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PutMapping("/password")
public ResponseEntity<String> changePassword(
        @Parameter(description = "User login details containing old and new passwords ", required = true)
        @RequestBody @Valid UserLoginDto userLoginDto) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "PUT /gym-api/trainers/password").increment();

    Trainer trainer = trainerService.getByUsername(userLoginDto.getUserName()).get();
    PasswordChangingResult result = trainerService.changePassword(trainer, userLoginDto.getOldPassword(), userLoginDto.getNewPassword());
    return result.isSucceed() ?
            new ResponseEntity<>(result.getMessage(), HttpStatus.OK) :
            new ResponseEntity<>(result.getMessage(), HttpStatus.BAD_REQUEST);
}
```

changePassword() для оновлення паролю для профіля тренера

Для профілю користувача:

```
Yura-Moroz
@Operation(summary = "Change password for a trainee")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Password changed successfully", content = @Content),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PutMapping("/password")
public ResponseEntity<String> changePassword(
        @Parameter(description = "User login details containing old and new passwords", required = true)
        @RequestBody @Valid UserLoginDto userLoginDto) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "PUT /gym-api/trainees/password").increment();

    Trainee trainee = traineeService.getByUsername(userLoginDto.getUserName()).get();

    PasswordChangingResult result = traineeService.changePassword(trainee, userLoginDto.getOldPassword(), userLoginDto.getNewPassword());
    return result.isSucceed() ?
            new ResponseEntity<>(result.getMessage(), HttpStatus.OK) :
            new ResponseEntity<>(result.getMessage(), HttpStatus.BAD_REQUEST);
}
```

changePassword() для оновлення паролю для профіля користувача

```
Yura-Moroz
@Operation(summary = "Update an existing trainee profile")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Profile updated successfully",
                content = {@Content(mediaType = "application/json",
                        schema = @Schema(implementation = TraineeDto.class))}),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PutMapping("/{id}")
@JsonView(TraineeViews.UpdateResp.class)
public ResponseEntity<TraineeDto> updateProfile(
        @Parameter(description = "ID of the trainee to update", required = true)
        @PathVariable long id,
        @Parameter(description = "Updated trainee details", required = true)
        @RequestBody @Valid
        @JsonView(TraineeViews.UpdateReq.class) TraineeDto traineeUpdatingDto) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "PUT /gym-api/trainees/{id}").increment();

    return ResponseEntity.status(HttpStatus.OK)
            .body(conversionService.convert(traineeService.update(id, traineeUpdatingDto), TraineeDto.class));
}
```

метод updateProfile() для оновлення профіля користувача

```
Yura-Moroz
@Operation(summary = "Toggle the active status of a trainee")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Status toggled successfully", content = @Content),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
})
@PatchMapping("/status")
public ResponseEntity<Void> changeStatus(
        @Parameter(description = "Trainee details with username for status update", required = true)
        @RequestBody @Valid
        @JsonView(TraineeViews.Status.class) TraineeDto traineeDto) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "PATCH /gym-api/trainees/status").increment();

    Trainee trainee = traineeService.getByUsername(traineeDto.getUserName()).get();
    if (trainee.isActive()) traineeService.deactivate(trainee);
    else traineeService.activate(trainee);

    return ResponseEntity.status(HttpStatus.OK).build();
}
```

метод changeStatus() для зміни статусу профіля користувача

Також для профіля юзера повинен бути реалізований функціонал для можливості видалення профіля:

```
Yura-Moroz
@Operation(summary = "Delete a trainee profile by username")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Profile deleted successfully", content = @Content),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
        @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
        @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
        @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
        @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)

})
@DeleteMapping("/{username}")
public ResponseEntity<Void> deleteProfile(
        @Parameter(description = "Username of the trainee to delete", required = true)
        @PathVariable String username) {

    meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "DELETE /gym-api/trainees/{username}")
            .increment();

    traineeService.deleteByUsername(username);
    return ResponseEntity.status(HttpStatus.OK).build();
}
```

метод deleteProfile() для видалення профіля користувача

Для забезпечення основної функціональності API, додаток повинен забезпечувати створення тренувань:

```
Yura-Moroz
@Tag(name = "Training Controller", description = "Endpoints for trainings manipulations")
@RequestMapping("/gym-api/trainings")
@RestController
@Validated
@RequiredArgsConstructor
public class TrainingController {

    private final TrainingService trainingService;

    private final ConversionService conversionService;

    private final MeterRegistry meterRegistry;

    Yura-Moroz
    @Operation(summary = "Create a new training")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "New training created successfully", content = @Content),
            @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
            @ApiResponse(responseCode = "403", description = "Forbidden", content = @Content),
            @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
    })
    @PostMapping
    public ResponseEntity<Void> addTraining(@RequestBody
                                            @Valid
                                            TrainingAddingDto trainingDto) {

        meterRegistry.counter( name: "endpoint.calls",  ...tags: "endpoint", "POST /gym-api/trainings").increment();

        trainingService.save(conversionService.convert(trainingDto, Training.class));
        return ResponseEntity.status(HttpStatus.CREATED).body(null);
    }
```

Контролер Training та метод addTraining() для створення тренування

ДОДАТОК Б

Частина Сервісів

```java
@Slf4j
@AllArgsConstructor
public abstract class BaseUserServiceImpl<T extends User, R extends UserDao<T>> implements BaseUserService<T> {

    protected final R repository;

    Yura-Moroz
    @Override
    @Transactional
    public T save(T user) {
        log.info("Trying to save user...");

        if (user == null) throw new IllegalArgumentException("Expected User but no proper data was provided");

        user.setUserName(ProfileUtils.generateUsername(user, repository::ifExistByUsername));
        user.setPassword(PasswordHandler.hashPassword(user.getPassword()));

        return repository.save(user);
    }
```

Базовий сервіс та метод для створення юзера у БД

```java
    Yura-Moroz
    @Override
    @Transactional
    public PasswordChangingResult changePassword(T user, String oldPassword, String newPassword) {

        boolean succeed = false;
        String resultMessage;

        if (user == null) {
            resultMessage = "Can't change password when the user is null";
        } else if (!PasswordHandler.ifPasswordMatches(oldPassword, user.getPassword())) {
            resultMessage = "Sorry, It seems that you've provided wrong old password";
        } else if (!PasswordHandler.verify(newPassword)) {
            resultMessage = "Please check that your new password meets all requirements (length should be 4-10 chars)";
        } else if (!repository.ifExistById(user.getId())) {
            resultMessage = "Sorry, can't change password because provided user doesn't exist...";
        } else {
            user.setPassword(PasswordHandler.hashPassword(newPassword));
            repository.update(user);
            resultMessage = "New password was successfully set to the user";
            succeed = true;
        }

        return PasswordChangingResult.builder()
                .succeed(succeed)
                .message(resultMessage)
                .build();
    }
```

Базовий сервіс та метод для зміни паролю для профіля

```
Yura-Moroz
@Override
public boolean activate(T user) {
    log.info("Activating user profile");
    if (user == null) return false;

    user.setActive(true);
    return repository.update(user).isActive();
}

Yura-Moroz
@Override
public boolean deactivate(T user) {
    log.info("Deactivating user profile");
    if (user == null) return false;

    user.setActive(false);
    return !repository.update(user).isActive();
}
```

Базовий сервіс та методи для зміни статусу профіля

```
Yura-Moroz
@Override
public void delete(T user) {
    log.info("Deleting user...");
    repository.delete(user);
}

Yura-Moroz
@Override
public long count(){
    log.info("Getting users count");
    return repository.count();
}
```

Базовий сервіс та метод для видалення, а також службовий метод для підрахування кількості існуючих профілів у БД

ДОДАТОК В

Базовий ДАО клас для усіх юзерів

```java
@Slf4j
public abstract class UserDaoImpl<T extends User> implements UserDao<T> {

    @PersistenceContext
    protected EntityManager entityManager;

    private final Class<T> clazz;

    Yura-Moroz
    public UserDaoImpl(Class<T> clazz) { this.clazz = clazz; }

    Yura-Moroz
    @Override
    public Optional<T> getById(long id) {
        log.info("Getting user by id");

        T entity = entityManager.find(clazz, id);
        return Optional.ofNullable(entity);
    }

    Yura-Moroz
    @Override
    public List<T> getAll() {
        log.info("Getting a list of all users present in the DB");

        Query query = entityManager.createQuery( s: "SELECT user FROM " + clazz.getName() + " user");
        return query.getResultList();
    }

    Yura-Moroz
    @Override
    public boolean ifExistById(long id) {
        log.info("Checking if user exist by id");
        return entityManager.find(clazz, id) != null;
    }
```

Базовий DAO клас для усіх юзерів

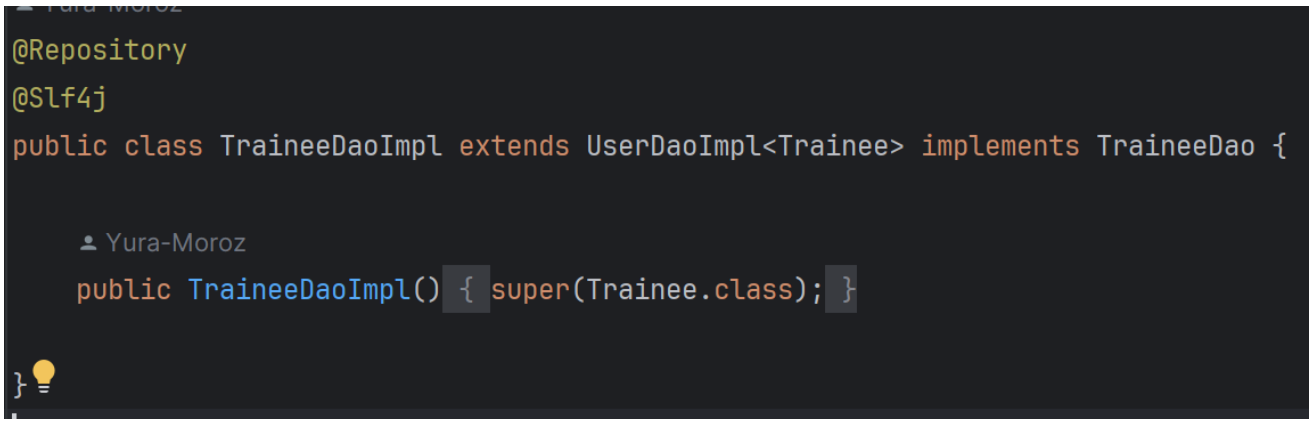
```
Yura-Moroz
@Override
@Transactional
public T save(T entity) {
    log.info("Trying to save an entity to the DB");
    //entityManager.persist() returned type is void
    entityManager.persist(entity);
    return entity;
}

Yura-Moroz
@Override
@Transactional
public T update(T entity) {
    log.info("Trying to update an entity in the DB");
    return entityManager.merge(entity);
}

Yura-Moroz
@Override
@Transactional
public void delete(T entity) {
    log.info("Trying to delete a user from the DB");
    entityManager.remove(entity);
}

Yura-Moroz
@Override
public Optional<T> getByUsername(String username) {
    log.info("Trying to get user by '" + username + "' login");

    String jpqlQuery = "SELECT user FROM " + clazz.getName() + " user WHERE user.userName = :login";
    Query query = entityManager.createQuery(jpqlQuery);
    query.setParameter( s: "login", username);

    try {
        T returnedUser = (T) query.getSingleResult();
        return Optional.ofNullable(returnedUser);
    } catch (NoResultException e) {
        return Optional.empty();
    }
}
```

інші методи базового DAO класу

```
    Yura-Moroz
    @Override
    public boolean ifExistByUsername(String username) {
        log.info("Checking if user exists with '" + username + "' login");

        String jpqlQuery = "SELECT COUNT(user) FROM " + clazz.getName() + " user WHERE user.userName = :login";
        Query query = entityManager.createQuery(jpqlQuery);
        query.setParameter( s: "login", username);
        Long counter = (Long) query.getSingleResult();

        return counter > 0;
    }

    Yura-Moroz
    @Override
    public long count(){
        log.info("Getting " + clazz.getName() + " entities count from the DB");

        String jpql = "SELECT COUNT(user) FROM " + clazz.getName() + " user";
        return (Long) entityManager.createQuery(jpql).getSingleResult();
    }

}
```

додаткові методи базового DAO класу

Клас ДАО клієнта фітнес-клубу

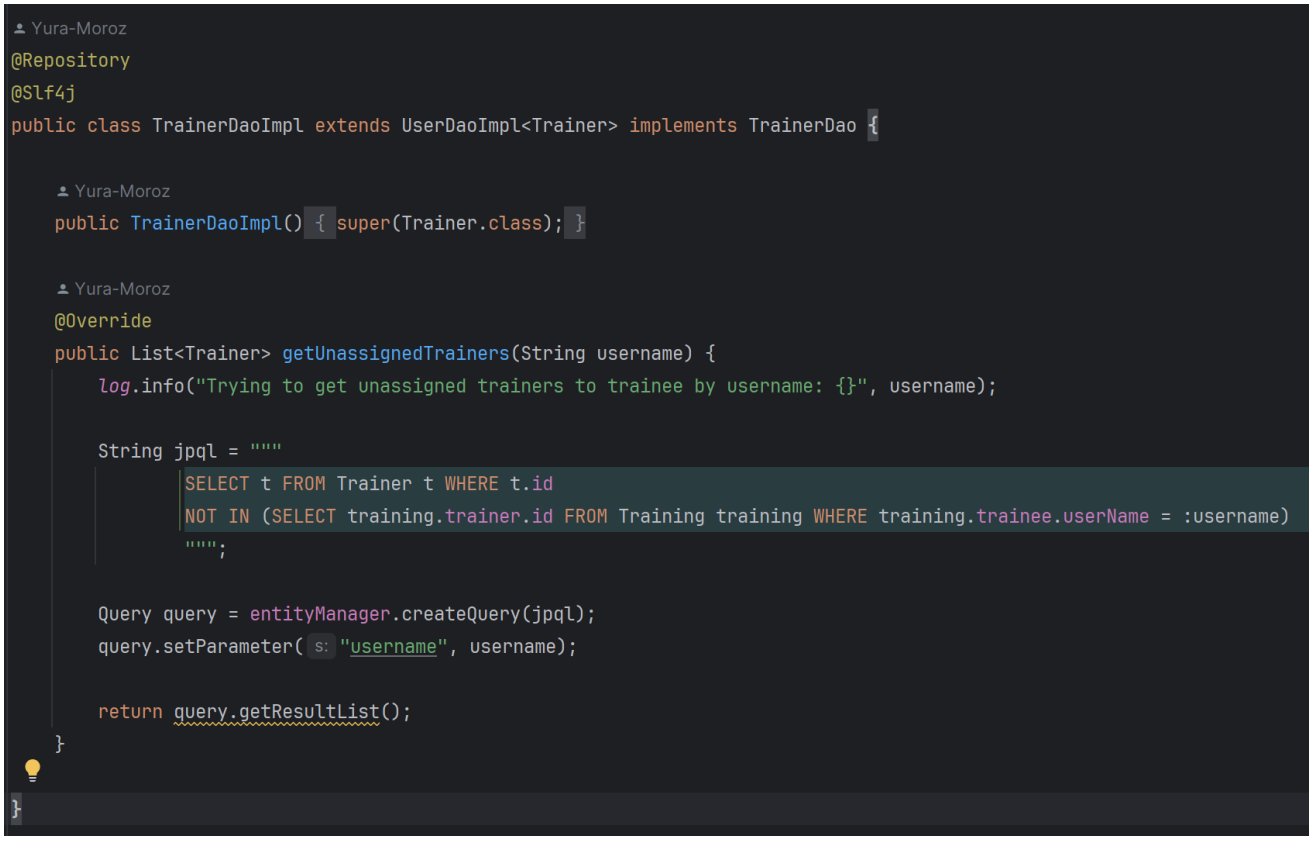


```
@Repository
@Slf4j
public class TraineeDaoImpl extends UserDaoImpl<Trainee> implements TraineeDao {

    Yura-Moroz
    public TraineeDaoImpl() { super(Trainee.class); }

}
```

Trainee DAO клас

Клас TraineeDaoImpl повністю наслідує поведінку базового UserDaoImpl та не перевизначає його методи.

Клас ДАО тренера фітнес-клубу



```
Yura-Moroz
@Repository
@Slf4j
public class TrainerDaoImpl extends UserDaoImpl<Trainer> implements TrainerDao {

    Yura-Moroz
    public TrainerDaoImpl() { super(Trainer.class); }

    Yura-Moroz
    @Override
    public List<Trainer> getUnassignedTrainers(String username) {
        log.info("Trying to get unassigned trainers to trainee by username: {}", username);

        String jpql = """
                SELECT t FROM Trainer t WHERE t.id
                NOT IN (SELECT training.trainer.id FROM Training training WHERE training.trainee.userName = :username)
                """;

        Query query = entityManager.createQuery(jpql);
        query.setParameter( s: "username", username);

        return query.getResultList();
    }
}
```

Trainer DAO клас

Клас ДАО для тренувань

```java
@Repository
@Slf4j
public class TrainingDaoImpl implements TrainingDao {

    @PersistenceContext
    private EntityManager entityManager;

    Yura-Moroz
    @Override
    public Optional<Training> getById(long id) {
        log.info("Getting a training by id");

        Training training = entityManager.find(Training.class, id);
        return Optional.ofNullable(training);
    }

    Yura-Moroz
    @Override
    public List<Training> getAll() {
        log.info("Getting a list of all trainings in the DB");

        Query query = entityManager.createQuery( s: "SELECT t FROM Training t");
        return query.getResultList();
    }

    Yura-Moroz
    @Override
    @Transactional
    public Training save(Training training) {
        log.info("Trying to save a training to the DB");
        // entityManager.persist() returned type is void
        entityManager.persist(training);
        return training;
    }
```

Training DAO клас

```java
Yura-Moroz
@Override
public boolean ifExistById(long id) {
    log.info("Checking if user exist by id");
    return entityManager.find(Training.class, id) != null;
}

Yura-Moroz *
@Override
public List<Training> getTrainingsByTraineeUsernameAndDateRange(String traineeLogin, LocalDate dateFrom, LocalDate dateTo, String trainerLogin,
                                                                TrainingType trainingType) {
    log.info("Trying to get trainings by criteria from DB");

    String jpql = """
            SELECT t FROM Training t WHERE t.trainee.userName = :traineeLogin
            AND t.trainingDate >= :dateFrom AND t.trainingDate <= :dateTo
            AND (t.trainer.userName = :trainerLogin)
            AND t.trainingType = :trainingType
            """;

    Query query = entityManager.createQuery(jpql);
    query.setParameter( s: "traineeLogin", traineeLogin);
    query.setParameter( s: "dateFrom", dateFrom == null ? LocalDate.now().atTime( hour: 0,  minute: 0,  second: 0) : dateFrom.atTime( hour: 0,  minute: 0,  second: 0));
    query.setParameter( s: "dateTo", dateTo == null ? LocalDate.now().atTime( hour: 23,  minute: 59,  second: 59) : dateTo.atTime( hour: 23,  minute: 59,  second: 59));
    query.setParameter( s: "trainerLogin", trainerLogin);
    query.setParameter( s: "trainingType", trainingType);

    return query.getResultList();
}
```

Training DAO клас та його методи

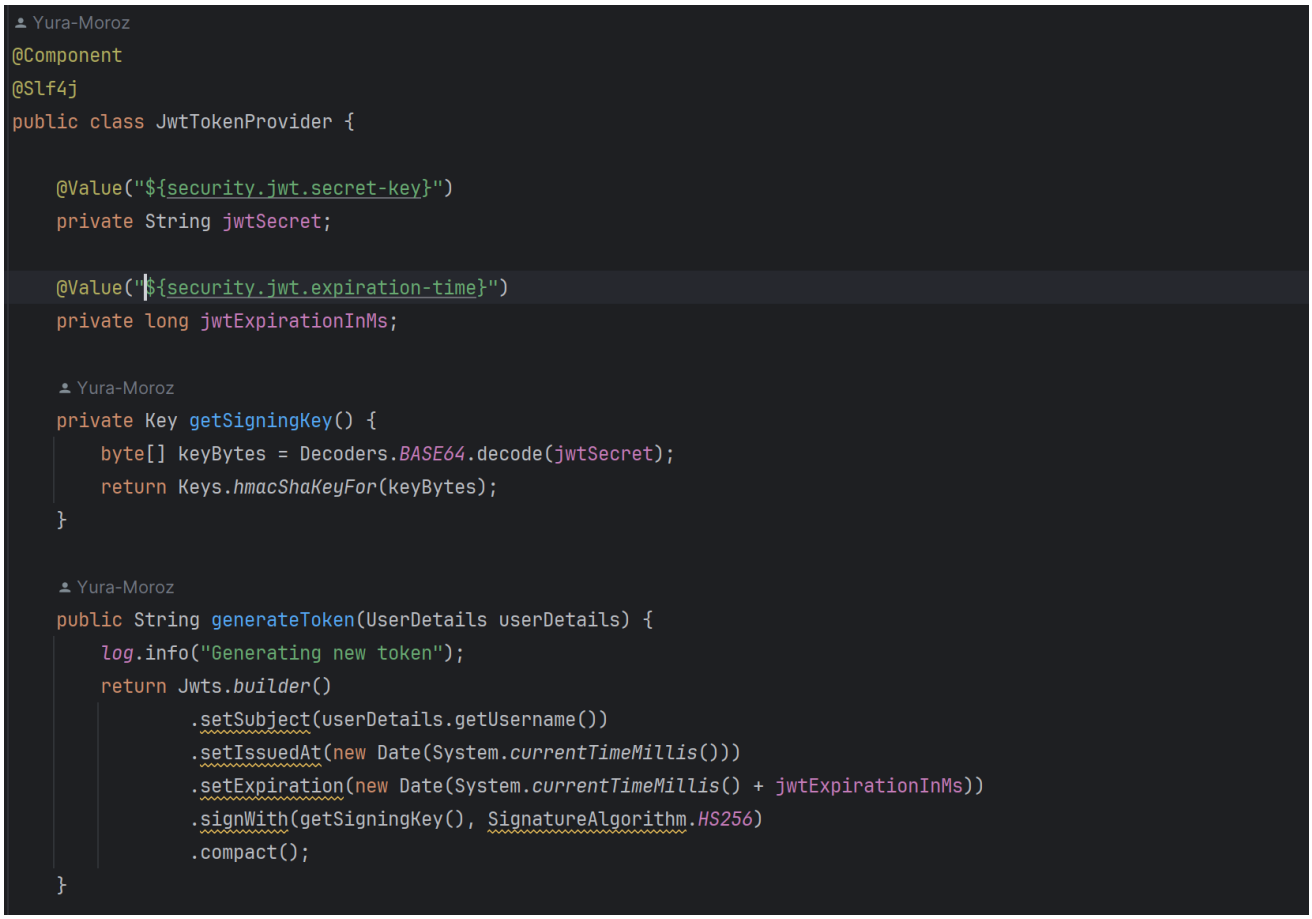
```java
Yura-Moroz *
@Override
public List<Training> getTrainingsByTrainerUsernameAndDateRange(String trainerLogin, LocalDate dateFrom, LocalDate dateTo, String traineeLogin,
                                                                TrainingType trainingType) {
    log.info("Trying to get trainings by criteria from DB");

    String jpql = """
            SELECT t FROM Training t where t.trainer.userName = :trainerLogin
            AND t.trainingDate >= :dateFrom AND t.trainingDate <= :dateTo
            AND t.trainee.userName = :traineeLogin
            AND t.trainingType = :trainingType
            """;

    Query query = entityManager.createQuery(jpql);
    query.setParameter( s: "trainerLogin", trainerLogin);
    query.setParameter( s: "dateFrom", dateFrom == null ? LocalDate.now().atTime( hour: 0,  minute: 0,  second: 0) : dateFrom.atTime( hour: 0,  minute: 0,  second: 0));
    query.setParameter( s: "dateTo", dateTo == null ? LocalDate.now().atTime( hour: 23,  minute: 59,  second: 59) : dateTo.atTime( hour: 23,  minute: 59,  second: 59));
    query.setParameter( s: "traineeLogin", traineeLogin);
    query.setParameter( s: "trainingType", trainingType);

    return query.getResultList();
}

Yura-Moroz
@Override
public long count(){
    log.info("Getting Training entities count from the DB");

    String jpql = "SELECT COUNT(training) FROM Training training";
    return (Long) entityManager.createQuery(jpql).getSingleResult();
}

Yura-Moroz
public Training update(Training training) {
    log.info("Updating Training with id: " + training.getId());
    return entityManager.merge(training);
}
```

Training DAO клас та його методи

ДОДАТОК Г

```
Yura-Moroz
@RestController
@RequestMapping("/auth")
@RequiredArgsConstructor
@Validated
public class AuthController {

    private final SecurityService securityService;

    Yura-Moroz
    @Operation(summary = "To log in user to the system")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "202", description = "Successfully authenticated", content = @Content),
            @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
            @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
    })
    @PostMapping("/login")
    public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {
        try {
            return ResponseEntity.status(HttpStatus.ACCEPTED).body(securityService.login(loginRequest));
        } catch (RuntimeException ex) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Can't authorize user." +
                    "\nPlease check all credentials and try again");
        }
    }

    Yura-Moroz
    @Operation(summary = "To log out user from the system")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully logged out", content = @Content),
            @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content),
            @ApiResponse(responseCode = "404", description = "Not Found", content = @Content),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal Server Error", content = @Content)
    })
    @PostMapping("/logout")
    public ResponseEntity<String> logoutUser(@RequestHeader(value = "Authorization", required = false) String authHeader) {
        return ResponseEntity.ok(securityService.logout(authHeader));
    }
}
```

клас контролер AuthController та методи для реєстрації та аутентифікації



```
Yura-Moroz
@Component
@Slf4j
public class JwtTokenProvider {

    @Value("${security.jwt.secret-key}")
    private String jwtSecret;

    @Value("${security.jwt.expiration-time}")
    private long jwtExpirationInMs;

    Yura-Moroz
    private Key getSigningKey() {
        byte[] keyBytes = Decoders.BASE64.decode(jwtSecret);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    Yura-Moroz
    public String generateToken(UserDetails userDetails) {
        log.info("Generating new token");
        return Jwts.builder()
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationInMs))
                .signWith(getSigningKey(), SignatureAlgorithm.HS256)
                .compact();
    }
```

клас JwtTokenProvider та методи для отримання ключа та генерації токену

```
    /**
     * Parses the token (assuming it is a JWS, i.e., a signed JWT) and validates the signature and claims (such as expiration).
     * If the token is invalid (bad signature, expired, malformed, etc.), one of the exceptions in the catch block will be thrown.
     */
    Yura-Moroz
    public boolean validateToken(String authToken) {
        log.info("Validating provided token: " + authToken);
        try {
            Jwts.parser() JwtParserBuilder
                    .setSigningKey(getSigningKey())
                    .build() JwtParser
                    .parseClaimsJws(authToken);
            return true;
        } catch (JwtException e) {
            log.error(e.getMessage());
        }
        return false;
    }

    Yura-Moroz
    private <T> T extractClaim(String token, Function<Claims, T> claimsResolvers) {
        final Claims claims = extractAllClaims(token);
        return claimsResolvers.apply(claims);
    }

    Yura-Moroz
    public String extractUsernameFromJwt(String token) { return extractClaim(token, Claims::getSubject); }

    Yura-Moroz
    private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

    Yura-Moroz
    private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

    Yura-Moroz
    private Claims extractAllClaims(String token) {
        return Jwts.parser() JwtParserBuilder
                .setSigningKey(getSigningKey())
                .build() JwtParser
                .parseClaimsJws(token) Jws<Claims>
                .getBody();
    }
}
```

клас JwtTokenProvider допоміжні методи для роботи із токеном

```java
Yura-Moroz
@Service
@Slf4j
public class BruteForceProtectionService {

    private final int MAX_ATTEMPT = 3;
    private final long BLOCK_TIME = 5 * 60 * 1000; // 5 minutes in milliseconds

    private final Map<String, FailedLogin> attemptsCache = new ConcurrentHashMap<>();
    private final Set<String> blacklist = ConcurrentHashMap.newKeySet();

    Yura-Moroz
    public void loginSucceeded(String key) {
        log.info("Removing user's username from attempts");
        attemptsCache.remove(key);
    }

    Yura-Moroz
    public void loginFailed(String key) {
        log.info("Checking the number of attempts left before the user is blocked");
        FailedLogin failedLogin = attemptsCache.get(key);
        if (failedLogin == null) {
            failedLogin = new FailedLogin( attempts: 1, System.currentTimeMillis());
        } else {
            failedLogin.increment();
        }
        attemptsCache.put(key, failedLogin);
    }
```

клас BruteForceProtectionService та допоміжні методи для роботи із автентифікацією

```
public boolean isBlocked(String key) {
    log.info("Setting user status to 'blocked' for " + (BLOCK_TIME/60/1000) + " minutes");
    FailedLogin failedLogin = attemptsCache.get(key);
    if (failedLogin == null) {
        return false;
    }
    if (failedLogin.getAttempts() >= MAX_ATTEMPT) {
        long timeSinceLastAttempt = System.currentTimeMillis() - failedLogin.getLastAttempt();
        if (timeSinceLastAttempt < BLOCK_TIME) {
            return true;
        } else {
            // Block time expired—reset counter.
            attemptsCache.remove(key);
            return false;
        }
    }
    return false;
}

 Yura-Moroz
public void blacklistToken(String token) {
    log.info("Putting token to blacklist");
    blacklist.add(token);
}

 Yura-Moroz
public boolean isTokenBlacklisted(String token) {
    log.info("Checking if token is blacklisted");
    return blacklist.contains(token);
}

 Yura-Moroz
@Getter
@AllArgsConstructor
private static class FailedLogin {
    private int attempts;
    private long lastAttempt;

     Yura-Moroz
    public void increment() {
        log.info("Setting new number for attempts and a time for the last one");
        attempts++;
        lastAttempt = System.currentTimeMillis();
    }
```

клас BruteForceProtectionService та допоміжні методи для роботи із автентифікацією

```java
Yura-Moroz
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final TrainerService trainerService;
    private final TraineeService traineeService;

    Yura-Moroz
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<? extends User> optionalUser = trainerService.getByUsername(username);
        if (optionalUser.isEmpty()) {
            optionalUser = traineeService.getByUsername(username);
        }
        if (optionalUser.isEmpty()) {
            throw new UsernameNotFoundException("User not found with username: " + username);
        }
        User user = optionalUser.get();

        return new org.springframework.security.core.userdetails.User(
                user.getUserName(),
                user.getPassword(),
                Collections.emptyList()
        );
    }
}
```

клас CustomUserDetailsService та метод для отримання вхідних даних юзера для автентифікації

```java
Yura-Moroz
@Service
@RequiredArgsConstructor
@Slf4j
public class SecurityServiceImpl implements SecurityService {

    private final AuthenticationManager authenticationManager;
    private final JwtTokenProvider tokenProvider;
    private final BruteForceProtectionService bruteForceProtectionService;

    Yura-Moroz
    @Override
    public JwtAuthenticationResponse login(LoginRequest loginRequest) {
        String username = loginRequest.getUserName();

        if (bruteForceProtectionService.isBlocked(username)) {
            throw new ResponseStatusException(HttpStatus.TOO_MANY_REQUESTS,
                    "User is temporarily blocked due to multiple failed login attempts. Please, try again later.");
        }

        try {
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(username, loginRequest.getPassword()));
            /** On successful authentication, reset failed attempts */
            bruteForceProtectionService.loginSucceeded(username);
            SecurityContextHolder.getContext().setAuthentication(authentication);
            UserDetails userDetails = (UserDetails) authentication.getPrincipal();
            String jwt = tokenProvider.generateToken(userDetails);
            return new JwtAuthenticationResponse(jwt);

        } catch (BadCredentialsException ex) {
            /** Record failed attempt */
            bruteForceProtectionService.loginFailed(username);
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Invalid username or password");
        }
    }
```

```java
    Yura-Moroz
    @Override
    public String logout(String authHeader) {
        return Optional.ofNullable(authHeader)
                .filter(header -> header.startsWith("Bearer "))
                .map(header -> header.substring("Bearer ".length()).trim())
                .map(token -> {
                    bruteForceProtectionService.blacklistToken(token);
                    return "Logout succeed";
                })
                .orElseThrow(() -> new ResponseStatusException(HttpStatus.BAD_REQUEST, "Missing or invalid Authorization header"));
    }
}
```

клас Security service

1. Аутентифікація через JWT

- **Метод login() у SecurityServiceImpl**
  - Отримує DTO LoginRequest з полями userName та password.
  - Перевіряє у BruteForceProtectionService, чи користувач не заблокований після трьох невдалих спроб.
  - Викликає authenticationManager.authenticate(...), який делегує аутентифікацію вбудованим провайдерам Spring Security.
  - У разі успіху виконується:
    - Скидання лічильника невдалих спроб в bruteForceProtectionService.
    - Занесення об’єкта Authentication у SecurityContextHolder.
    - Отримання UserDetails з CustomUserDetailsService (див. нижче).
    - Генерація JWT-токена через JwtTokenProvider на основі username, часу створення та секретного ключа.
    - Повернення клієнту JwtAuthenticationResponse з готовим токеном.
- **Валідація кожного запиту**
  - Клієнт надсилає токен у заголовку Authorization: Bearer <token>.
  - Вбудований фільтр Spring Security (наприклад, JwtAuthenticationFilter) перехоплює запит, витягує й перевіряє підпис токена через JwtTokenProvider.
  - Якщо токен валідний і не знайдений у «чорному списку» (див. блок лістинга), то в SecurityContextHolder встановлюється відповідний Authentication, і запит потрапляє до контролера.

2. Завантаження деталей користувача

Клас CustomUserDetailsService реалізує інтерфейс UserDetailsService і відповідає за пошук користувачів у двох бізнес-сервісах:

- **trainerService.getByUsername(username)** — шукає тренера з таким логіном.

- Якщо перший пошук не дав результату, викликається **traineeService.getByUsername(username)** — пошук серед клієнтів (слухачів).
- Якщо користувача не знайдено, викидається UsernameNotFoundException, що перериває процес аутентифікації.
- Інакше повертається Spring-об’єкт UserDetails, зберігаючи в ньому логін і зашифрований пароль. Рольова модель у проєкті поки не використовується (порожній список авторитетів).

3. Захист від brute-force атак

Клас BruteForceProtectionService відповідає за облік невдалих спроб входу та тимчасову блокування:

- **Лічильник спроб**
  - Перший невдалий логін створює запис у attemptsCache із полем attempts = 1.
  - Кожна наступна невдала спроба збільшує лічильник методом increment(), оновлюючи lastAttempt = System.currentTimeMillis().
- **Блокування користувача**
  - Якщо лічильник спроб досягає MAX_ATTEMPT (3), метод isBlocked() перевіряє, чи не минуло BLOCK_TIME (5 хвилин) з моменту останньої невдалої спроби.
  - Якщо час блокування ще триває, користувач не зможе пройти аутентифікацію до його закінчення. Якщо час вичерпано – лічильник обнуляється.
- **Скидання лічильника**
  - Після успішного входу метод loginSucceeded() видаляє записи про невдалі спроби.

4. Вихід із системи та чорно-білий список

- **Метод logout(String authHeader)**
  - Вирізає токен зі значення заголовка Authorization.
  - Додає його до внутрішнього blacklist у BruteForceProtectionService методом blacklistToken(token).
  - Повертає повідомлення про успішний вихід.
- **Перевірка черного списку**
  - При обробці кожного запиту фільтр має викликати у BruteForceProtectionService.isTokenBlacklisted(token).
  - Якщо токен «заблоковано», доступ за цим токеном забороняється, навіть якщо сам JWT є валідним за підписом.

5. Переваги та гарантії безпеки

- **Без стану (stateless)** — система не зберігає сесії на сервері, а покладається на самодостатні JWT.
- **Захист від підбору пароля** завдяки обмеженню кількості спроб і тимчасовому блокуванню.
- **Гнучке розширення** — у майбутньому можна додати підтримку ролей і прав доступу, реалізувати оновлення токенів (refresh tokens) або багатофакторну аутентифікацію.
- **Контроль виходу** — чорно-білий список дозволяє негайно анулювати токени при logout або при виявленні компрометації.

Таким чином, поєднання Spring Security, JWT та власного механізму захисту від brute-force атак забезпечує надійний і масштабований шар безпеки для CRM-системи фітнес-клубу.

“СИСТЕМА УПРАВЛІННЯ ВЗАЄМОДІЇ З КЛІЄНТАМИ ІЗ ЗАСТОСУВАННЯМ АРХІТЕКТУРИ MVC ТА RESTFUL API”

ВИКОНАВ: СТУДЕНТ ГРУПИ ІСД-41
МОРОЗ Ю. Є.

ВСТУП

Актуальність дослідження зумовлена зростаючими вимогами до управління бізнес-процесами у фітнес-індустрії. Сучасні фітнес-клуби потребують ефективного управління взаємодією з клієнтами, що включає бронювання тренувань та зручний вибір типу заняття.

Мета дипломної роботи — розробка та імплементація CRM-системи для управління взаємодією з клієнтами фітнес-клубу, яка відповідає сучасним вимогам.

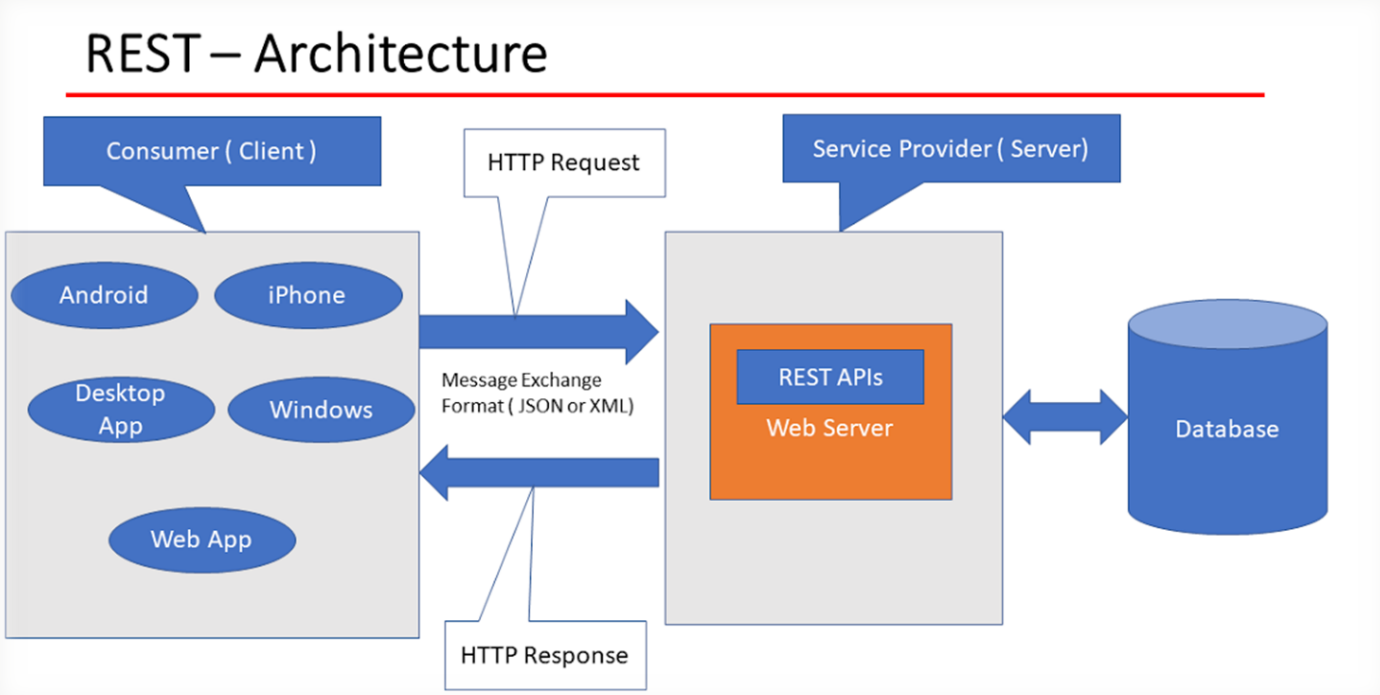
Предметом дослідження є процеси управління взаємодією з клієнтами, а **об'єктом** — CRM-система для автоматизації бізнес-процесів фітнес-клубів.

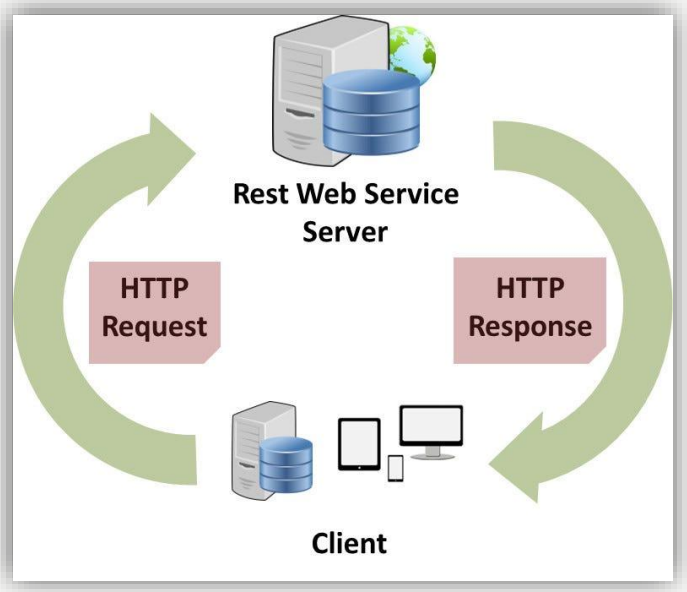
У роботі проведено аналіз предметної області, що дозволив виокремити ключові вимоги до системи: зручність користування, масштабованість та надійність. Бек-енд реалізовано на Java з використанням Spring Boot, що забезпечує високу продуктивність і безпеку.

Отримана Gym CRM API демонструє стабільну роботу в основних сценаріях, таких як реєстрація клієнтів та управління профілем. Перспективи розвитку включають аналітичні модулі, інтеграцію з платіжними сервісами та впровадження push-повідомлень. Результати роботи можуть суттєво підвищити ефективність бізнес-процесів у фітнес-індустрії.

АРХІТЕКТУРА

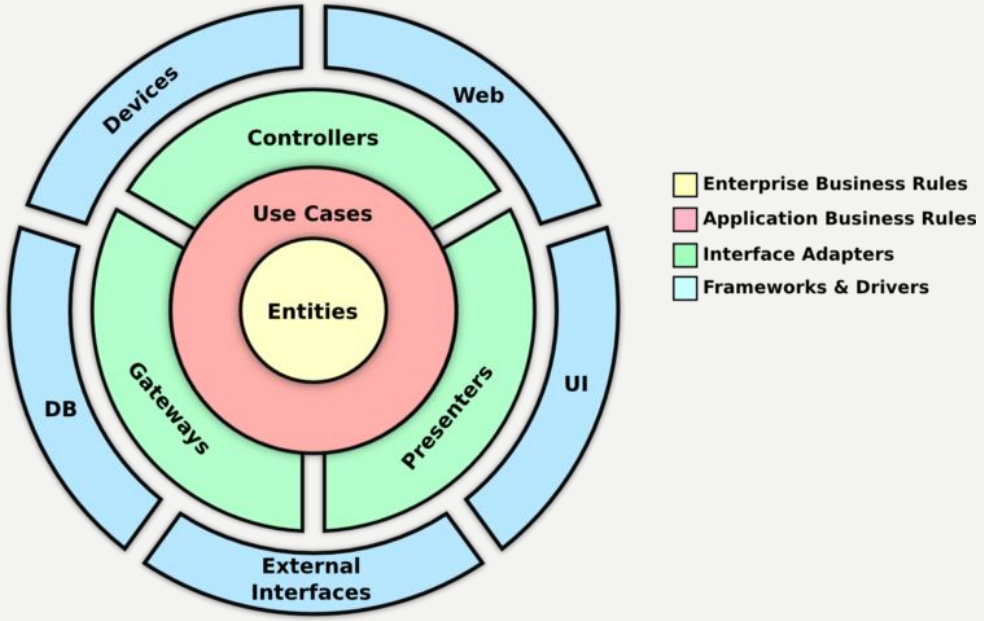
Для системи управління бронюванням тренувань фітнес-клубу доцільно комбінувати MVC на рівні фронтенда (наприклад, для адмін-панелі та користувацького порталу), шарову серверну структуру та RESTful API для мобільних і веб-клієнтів. Усередині API можна окремо виділити сервіси для реєстрації, керування розкладом, моніторингу завантаженості тренерів та аналітики.

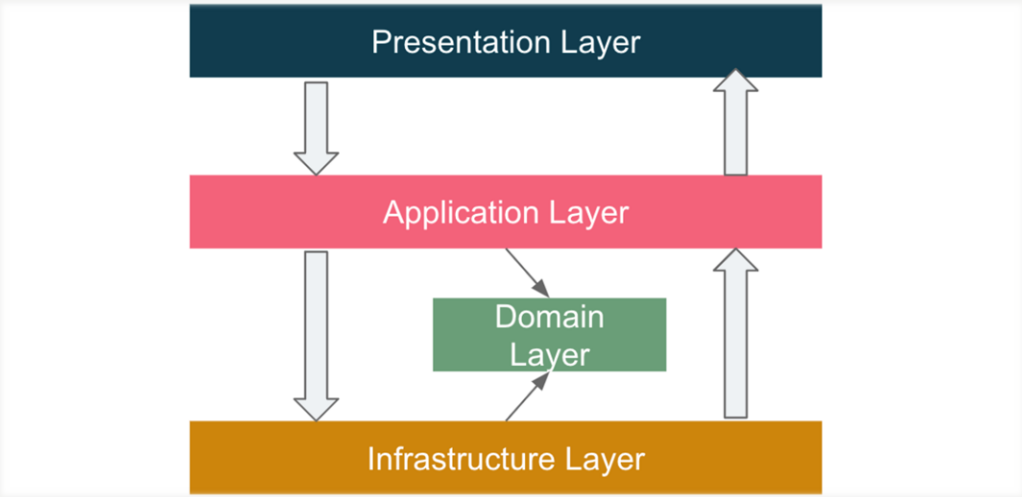




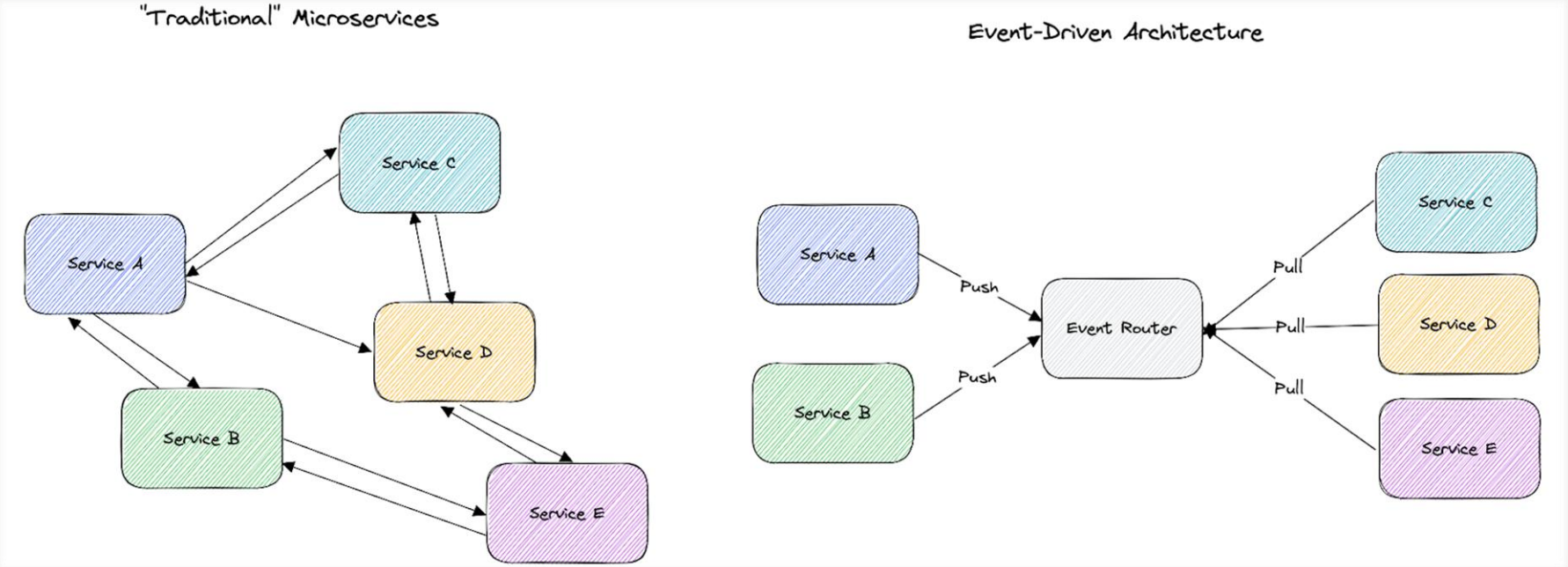
АРХІТЕКТУРА

Підхід Domain-Driven Design (DDD) фокусується на виділенні окремих предметних областей (доменів) і побудові «контекстів» (Bounded Context), у межах яких застосовуються власні моделі даних і термінологія. Для CRM це може означати розділення на такі домени, як «Управління клієнтами», «Продажі», «Маркетинг» та «Підтримка клієнтів». Кожен домен містить агрегати, сутності та сервіси, що спрощує підтримку обширних бізнес-правил і дозволяє реалізовувати складні бізнес-процеси в єдиній системі з чіткими кордонами відповідальності. Використання DDD допоможе чітко розмежувати функціональні зони (бронювання, профілі користувачів, оплати)





АРХІТЕКТУРА

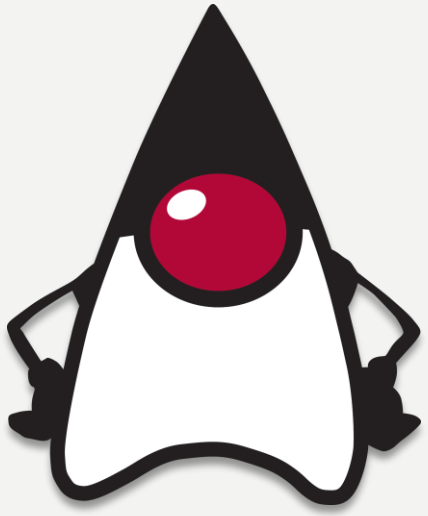


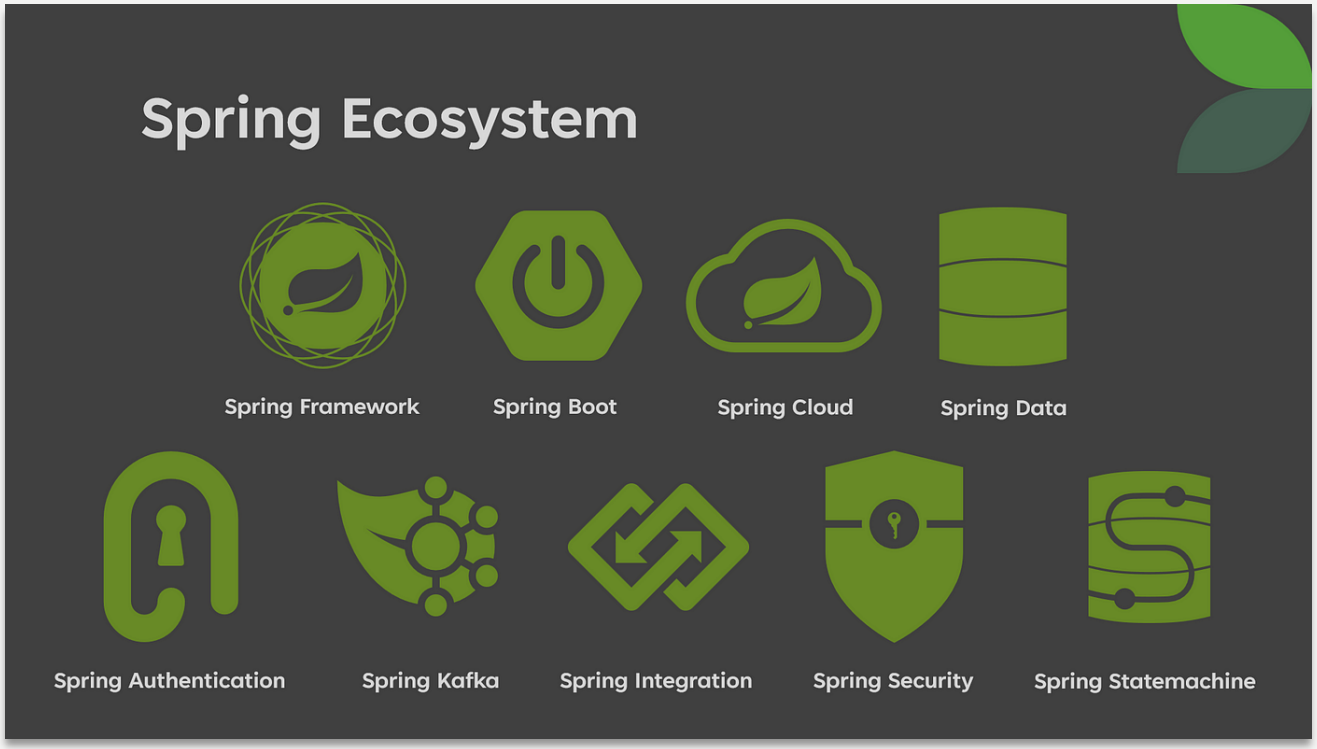
У CRM-платформах дедалі ширше застосовується event-driven підхід: зміну стану сутності (наприклад, створення нового ліда, бронювання тренування або успішне завершення продажу) надсилають у вигляді повідомлення (події) до шини подій або черги (Kafka, RabbitMQ). Інші компоненти підписуються на потрібні їм типи подій і реагують відповідно: оновлюють аналітичні звіти, надсилають email, запускають робочі процеси. Event-driven патерн полегшить імплементацію сповіщень та інтеграцію з push-сервісами або зовнішніми платіжними шлюзами.

Таким чином, оптимальною стане гібридна архітектура, що поєднує переваги шарового поділу, MVC-контролерів, REST-мікросервісів та event-driven взаємодії. Такий підхід гарантує високу гнучкість, масштабованість і стійкість до навантажень, відповідаючи потребам сучасного фітнес-бізнесу.

ТЕХНОЛОГІЧНІ ЗАСОБИ ДЛЯ РЕАЛІЗАЦІЇ CRM (JAVA, SPRING BOOT, MYSQL)

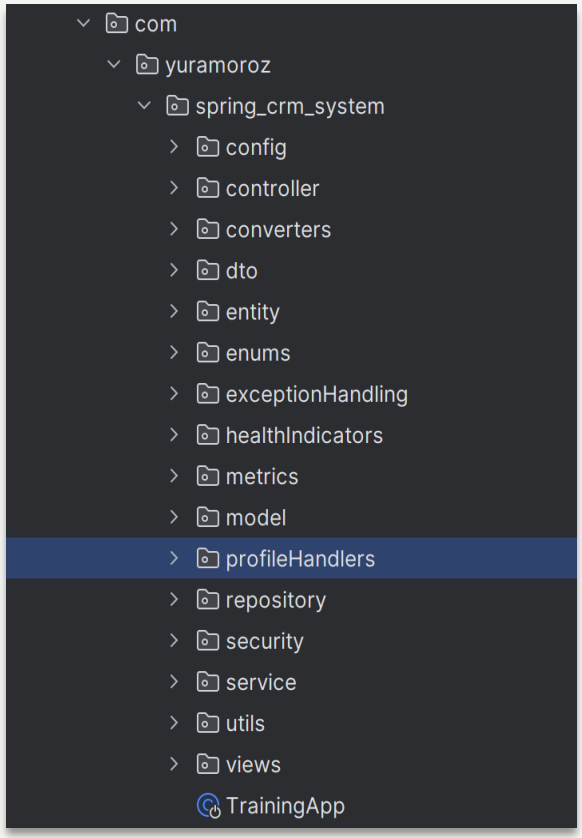
У сучасних корпоративних рішеннях для побудови масштабованих, надійних і гнучких CRM-систем найчастіше обирають стек технологій на базі мови програмування Java, фреймворку Spring Boot і реляційної СУБД MySQL. Такий набір забезпечує високий рівень платформної незалежності, велику спільноту розробників та багату екосистему готових модулів.

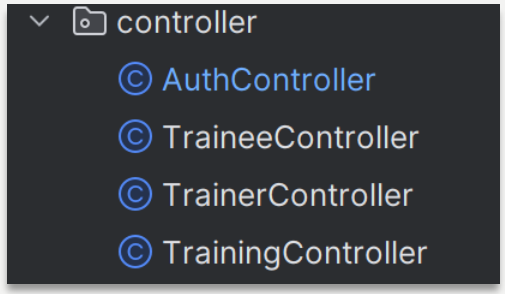


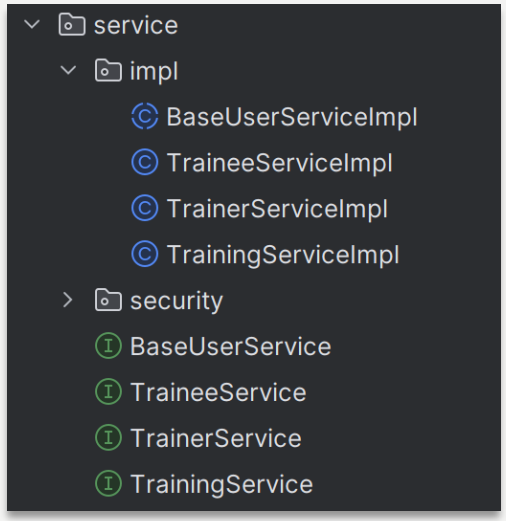


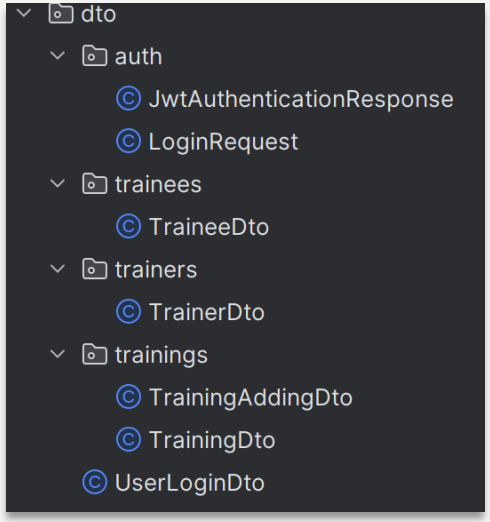
СТРУКТУРА

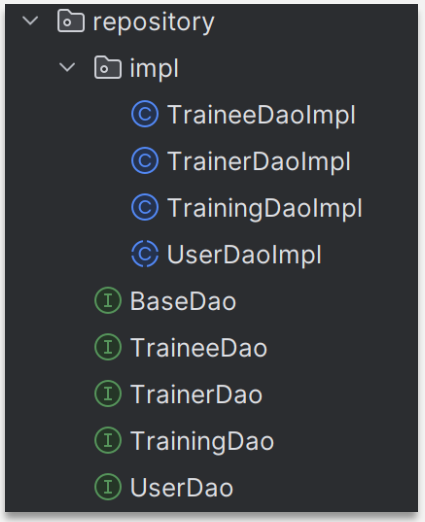
Простір коду поділено за областями відповідальності на пакети, де кожна область відповідає лише за свій функціонал:











ДЕМО

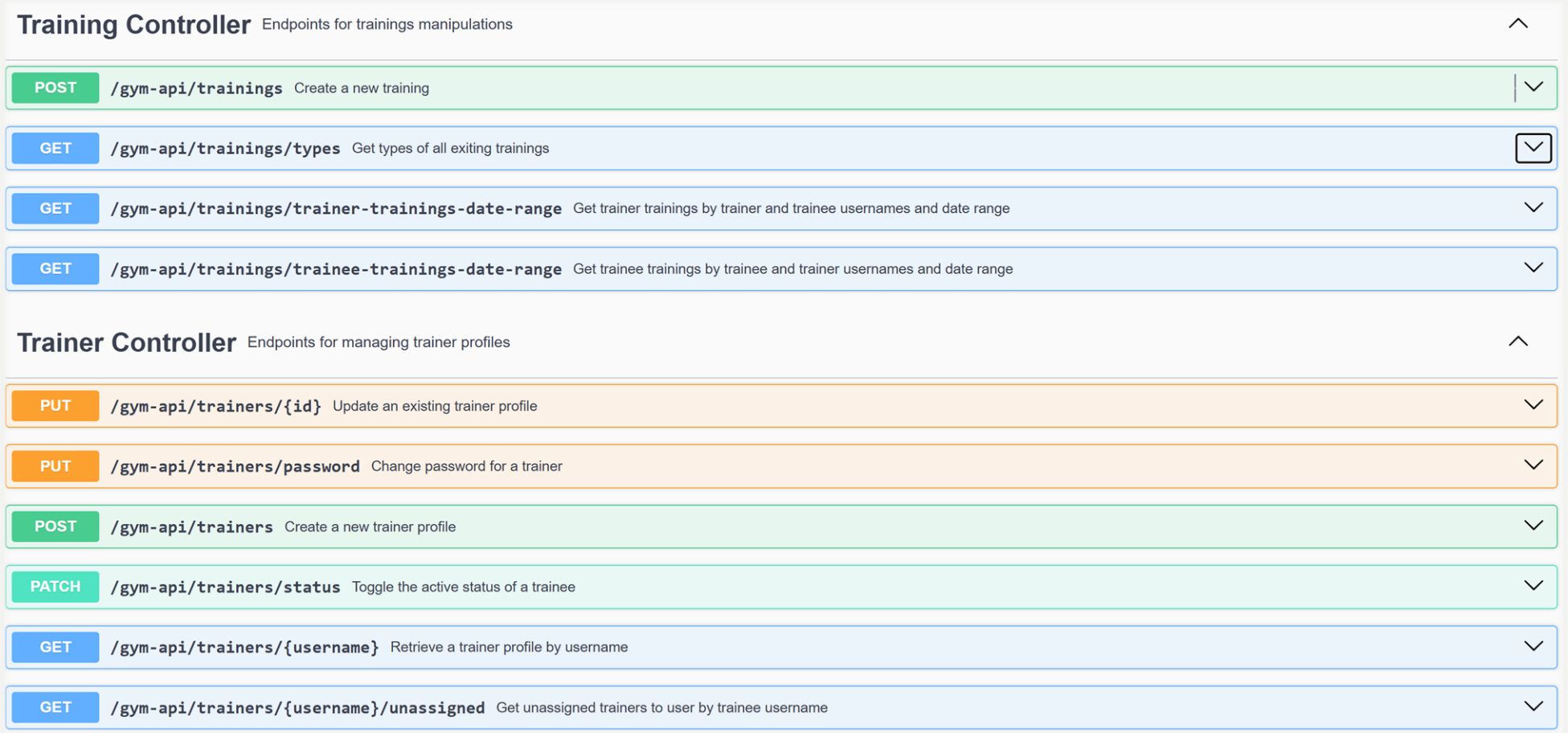


Рисунок ілюструє структуру API для управління тренуваннями та профілями тренерів у CRM-системі, включаючи різні методи для створення, оновлення та отримання даних про тренування та тренерів.

ДЕМО

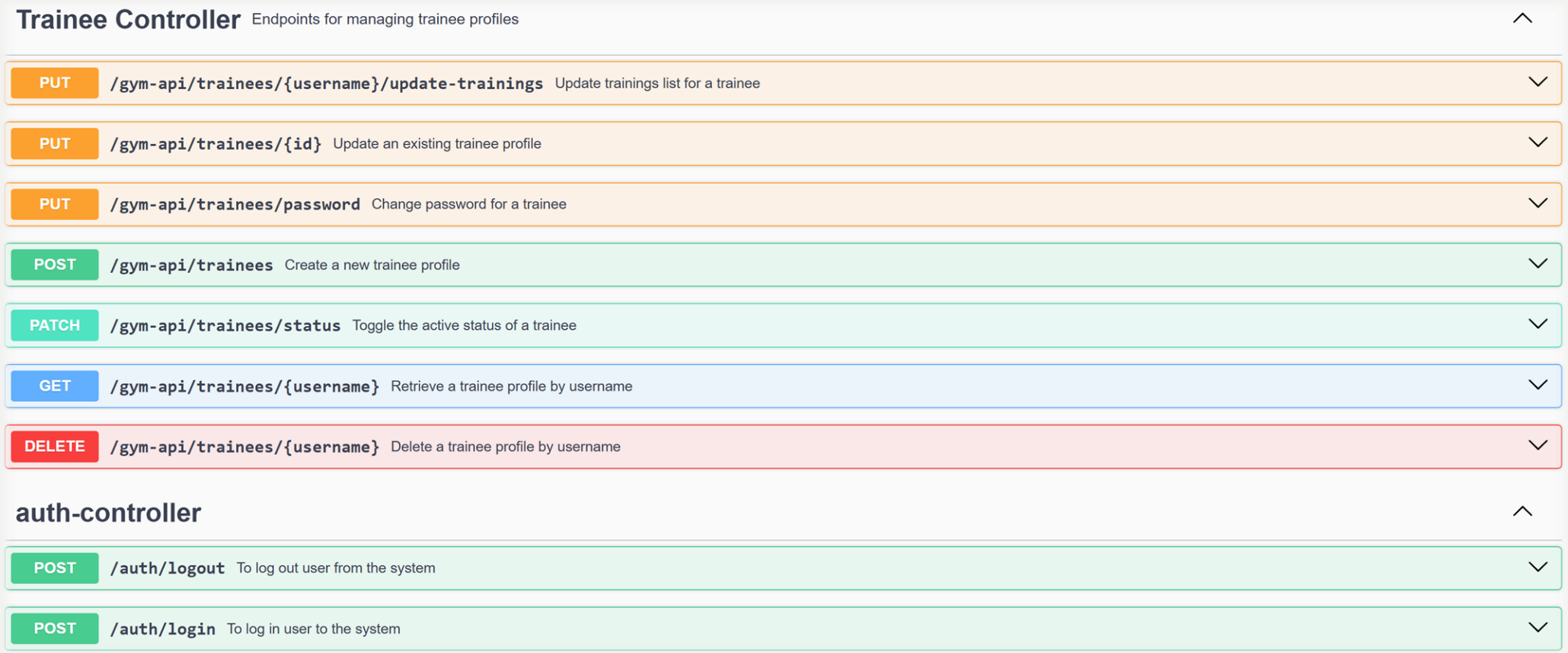


Рисунок демонструє структуру API для управління профілями клієнтів у CRM-системі, включаючи методи для оновлення, створення, видалення та автентифікації користувачів.

ВИСНОВОК

У ході виконання дипломного проекту була розроблена CRM-система для управління тренуваннями, користувачами та тренерами, яка відповідає сучасним вимогам фітнес-індустрії. Система забезпечує високий рівень надійності, безпеки та продуктивності. Створений API-інтерфейс на основі Spring Boot дозволяє оперативно реєструвати та автентифікувати користувачів, управляти їхніми даними та історією тренувань, а також здійснювати розподіл і облік занять між тренерами.

У цілому, розроблена CRM-система відповідає поставленим цілям дипломного проекту. Вона забезпечує безпечне та ефективне керування даними користувачів, тренерів і тренувань, надає гнучкі можливості для масштабування та адаптації до хмарної інфраструктури AWS. Отримане рішення є міцним технічним фундаментом для подальшого розвитку, інтеграції з іншими сервісами та створення повноцінного продукту, що сприятиме підвищенню ефективності бізнес-процесів фітнес-індустрії та збільшенню прибутковості компаній.

ДЯКУЮ ЗА УВАГУ!