**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА REST API ФІТНЕС-ЦЕНТРІВ З ВИКОРИСТАННЯМ LARAVEL»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.Є. Єрмішин
(ініціали та прізвище)

Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.пед.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єрмішину Дмитру Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка REST API фітнес-центрів з використанням Laravel

керівник роботи Панасенко Євген Валерійович, к.ф.-м.н, доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Практична реалізація коду.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. REST API фітнес-центрів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.02.2023 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 08.02.2023 | |
| | | | |
| 2. | Збір вихідних даних. | 21.02.2023 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 07.03.2023 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 11.04.2023 | |
| | | | |
| 5. | Розробка третього розділу. | 17.05.2023 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 01.06.2023 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 22.06.2023 | |

Студент _____ Д.Є. Єрмішин
(підпис) (ініціали та прізвище)

Керівник роботи _____ Є.В. Панасенко
(підпис) (ініціали та прізвище)

**Нормоконтроль пройдено**

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка REST API фітнес-центрів з використанням Laravel»: 48 с., 38 рис., 1 табл., 12 джерел, 1 додаток.

API, CRUD, GIT, GIT HOOKS, LARAVEL, PHP, PHP UNIT, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ.

Об’єкт дослідження – процес розробки REST API фітнес-центрів.

Мета роботи: розробка REST API фітнес-центрів.

Методи дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії.

У роботі наведено деякі з засобів веброзробки, зокрема мова програмування PHP та її фреймворк Laravel, описано декілька бібліотек, які були використані при розробці, розглянуто основні принципи API, зазначено що таке система керування версіями і для чого такі системи потрібні. Описано предметну область. Розроблено REST API фітнес-центрів. Наведено декілька прикладів роботи з API, зокрема отримання, додавання, редагування та видалення даних, описано процес продажу абонементів.

Результати роботи можуть бути використані для подальшого застосування, наприклад, у сферах надання фітнес або SPA-послуг.

# SUMMARY

Bachelor’s qualifying paper «Development of a REST API for Fitness Centers Using Laravel»: 48 pages, 38 figures, 1 table, 12 references, 1 supplement.

API, AUTOMATED TESTING, CRUD, GIT, GIT HOOKS, LARAVEL, PHP, PHP UNIT.

The object of the study is the process of developing a REST API for fitness centers.

The aim of the study is development of REST API for fitness centers.

The methods of research are object-oriented programming methods, software engineering methods.

The qualification paper presents the main theoretical approaches and means of software development. Web development tools are described, including the PHP programming language and its Laravel framework. Several libraries used during the development are mentioned. The basic principles of APIs are reviewed. It is defined what is a version control system and why such systems are needed. The subject area is described. The REST API for fitness centers is developed. Several examples of working with the API are given, including retrieving, adding, editing, and deleting data, and the process of selling subscriptions is described.

The results of the work can be used for further application, for example, in the areas of fitness or SPA services.

**ЗМІСТ**

# ВСТУП

У сучасному світі фітнес-центри стають все популярнішими місцями для здорового способу життя та підтримки фізичної форми. Зростаючий інтерес до фітнесу вимагає розробки ефективних інструментів для управління та координації діяльності мережі фітнес-центрів. Одним з ключових аспектів такої системи є розробка додатків програмного забезпечення, зокрема RESTful API для забезпечення взаємодії з системою.

Метою цієї кваліфікаційної роботи є розробка RESTful API системи мережі фітнес-центрів з використанням потужного та популярного фреймворку Laravel. Цей фреймворк забезпечує широкі можливості для розробки вебдодатків та API, швидку інтеграцію з базами даних та зручне управління маршрутизацією, що робить його ідеальним вибором для реалізації API системи мережі фітнес-центрів.

Цей кваліфікаційний проєкт має на меті розглянути основні етапи процесу розробки RESTful API засобами Laravel, включаючи проєктування архітектури, реалізацію та тестування.

Актуальність цього дослідження виправдана необхідністю розробки ефективних та масштабованих систем управління фітнес-центрами, які забезпечують зручний доступ до інформації та можливість взаємодії з системою через API. Цей проєкт може бути корисним як для розробників, які прагнуть збагатити свої навички у розробці API, так і для фахівців у галузі фітнесу, які мають намір розробити свою систему управління фітнес-центром.

# 1 ОГЛЯД МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ

## 1.1 Мова програмування PHP

PHP, що розшифровується як «PHP: препроцесор гіпертексту», – це широко поширена мова сценаріїв загального призначення з відкритим кодом, яка особливо підходить для веброзробки та може бути вбудована в HTML. Його синтаксис базується на C, Java та Perl, і його легко вивчити. Основна мета мови – дозволити веброзробникам швидко писати динамічно створювані вебсторінки, але ви можете зробити набагато більше за допомогою PHP [4]. Вона була створена у 1994 році і з того часу набула великої популярності та широкого застосування. Багато розробників вважають PHP легким для вивчення, що дозволяє швидко розробляти вебдодатки. PHP має велику та активну спільноту розробників, яка постійно надає допомогу та ділиться знаннями. Це означає, що є багато ресурсів, форумів, бібліотек та фреймворків, які сприяють розробці на PHP та роблять її більш ефективною. PHP підтримується більшістю вебсерверів, операційних систем та баз даних. Це дозволяє розробникам використовувати PHP на різних платформах і інтегрувати його з іншими технологіями.

## 1.2 Фреймворк Laravel та його бібліотека

Laravel – це високопродуктивний та елегантний фреймворк з відкритим вихідним кодом для розробки вебдодатків на мові програмування PHP [5]. Він був створений з метою спростити та прискорити процес розробки, забезпечуючи розробникам потужні інструменти та готові рішення для широкого спектру задач.

Фреймворк має чистий і елегантний синтаксис, який полегшує читання та розуміння коду. Він використовує сучасні концепції програмування, такі як

анонімні функції, простори імен та замикання. Laravel надає потужні інструменти для визначення маршрутів та управління запитами HTTP. Ви можете легко визначити маршрути і призначити їм контролери, що дозволяє створювати чітку структуру додатку.

Також, Laravel має вбудовану ORM (Object-Relational Mapping) під назвою Eloquent, яка дозволяє легко взаємодіяти з базою даних. Вона пропонує зручну модель для роботи з таблицями бази даних, а також включає підтримку міграцій і сідерів. Ще однією перевагою фреймворку є наявність вбудованих механізмів безпеки, які допомагають захистити ваш додаток від атак, таких як SQL-ін'єкції, перехресні сайтові скрипти (XSS) та інші. Він включає функції, такі як хешування паролів, захист від CSRF-атак, міжсайтові запити та інші. Ще одним плюсом Laravel можна вважати наявність дуже активної спільноти розробників. Спільнота включає як початківців у програмуванні, так і досвідчених розробників, які активно спілкуються, співпрацюють та допомагають одне одному. Існують офіційні форуми та дискусійні групи, такі як Laravel.io та Laracasts Forum, де розробники можуть обговорювати проблеми, задавати питання та ділитися своїм досвідом. Це важливий ресурс для отримання допомоги та знань від інших учасників спільноти. Laravel має докладну та добре написану офіційну документацію, яка стає важливим джерелом інформації для розробників. Крім того, спільнота розробників Laravel часто надає допомогу одне одному шляхом відповідей на питання в форумах, соціальних медіа та чат-каналах. Багато розробників Laravel створюють власні пакети та розширення, які вони безкоштовно публікують у Laravel Packagist або GitHub. Це дозволяє іншим розробникам використовувати та розширювати функціональність Laravel шляхом використання цих пакетів.

Наприклад, Laravel Pint – це незалежний інструмент для виправлення стилю PHP-коду для мінімалістів. Pint побудований на основі PHP-CS-Fixer і дозволяє легко забезпечити чистоту та узгодженість стилю вашого коду [6].

Досить частою задачею є керування ролями та повноваженнями всередені самої системи. Бібліотека spatie/laravel-permission є популярним розширенням

для фреймворку Laravel, яке надає потужні засоби для управління правами доступу і ролями в вашому додатку [7]. Бібліотека дозволяє вам визначати ролі та дозволи в вашому додатку. Ви можете легко визначити ролі (наприклад, адміністратор, модератор, користувач) і призначати їх користувачам. Крім того, ви можете визначати дозволи і призначати їх ролям або окремим користувачам. Бібліотека легко інтегрується з моделями користувачів Laravel, що дозволяє вам легко управляти ролями та дозволами для ваших користувачів. Вона додає методи і відношення до вашої моделі користувача, що спрощує роботу з ролями та дозволами.

Ще однією корисною бібліотекою є spatie/laravel-backup, яка надає зручні засоби для резервного копіювання вашої бази даних та файлів. Вона дозволяє автоматизувати процес резервного копіювання і забезпечує легкий доступ до збережених копій. Бібліотека дозволяє вам регулярно створювати резервні копії вашої бази даних. Вона підтримує різні драйвери баз даних, такі як MySQL, PostgreSQL, SQLite і багато інших. Ви можете встановлювати розклад резервного копіювання та зберігання, налаштовувати параметри копіювання та контролювати обсяг збережених копій [8].

Бібліотека knuckleswtf/scribe є потужним інструментом для автоматичної генерації документації для вашого API в Laravel. Вона дозволяє з легкістю створювати та підтримувати документацію для ваших API маршрутів, моделей, контролерів та інших компонентів [9].

### 1.3 API та його основні принципи

API (Application Programming Interface) – це набір правил та протоколів, які визначають, як програмні компоненти мають взаємодіяти між собою. Вона визначає набір функцій, методів і структур даних, які можуть використовуватися для розробки програмного забезпечення [2].

API дозволяє різним програмним системам та компонентам обмінюватися

інформацією та взаємодіяти один з одним. Вона виступає в ролі посередника, який дозволяє різним додаткам та сервісам обмінюватися даними та виконувати дії через стандартизований інтерфейс.

API може бути реалізована у різних форматах, таких як REST (Representational State Transfer), SOAP (Simple Object Access Protocol), GraphQL і багато інших. Кожен формат має свої особливості та протоколи, що використовуються для взаємодії з API.

REST API є одним з найпоширеніших стандартів для створення вебсервісів і взаємодії між клієнтами та серверами. Він базується на принципах архітектури Інтернету та використовує HTTP протокол для передачі даних [11].

Основні принципи REST API включають:

– REST API моделює різні об'єкти або ресурси, до яких можна отримати доступ за допомогою унікальних ідентифікаторів (URL), наприклад, вебсервіс для книжного магазину може мати ресурси, такі як /books (книги) або /authors (автори);
– REST API використовує різні методи HTTP для виконання операцій з ресурсами (найпоширеніші методи включають GET (отримання даних), POST (створення нового ресурсу), PUT (оновлення існуючого ресурсу) та DELETE (видалення ресурсу));
– REST API передає дані між клієнтом і сервером у вигляді репрезентацій ресурсів – це може бути у форматі JSON, XML або інших форматах (JSON є найпоширенішим форматом для передачі даних через REST API);
– REST API не зберігає стан клієнта між запитами: кожен запит до API містить усю необхідну інформацію для обробки запиту, і сервер не зберігає інформацію про попередні запити, це дозволяє більшу масштабованість і простоту взаємодії;
– REST API підтримує можливість кешування відповідей сервера: це дозволяє клієнтам кешувати отримані дані і покращує продуктивність та швидкість взаємодії.

REST API є широко використовуваним стандартом для розробки вебсервісів, особливо в контексті мобільних додатків та SPA (Single Page Applications). Він забезпечує простоту, масштабованість, незалежність від платформи та можливість інтеграції з різними системами.

### 1.4 Система контролю версій Git

Також при розробці доцільно використовувати систему контролю версій. Наприклад, Git – це система контролю версій, яка призначена для відстеження змін у файловій системі та спільної роботи над проєктами. Вона дозволяє зберігати всю історію змін файлів, відстежувати, хто вносить зміни і коли, а також відновлювати або порівнювати версії файлів [Pro Git].

Розглянемо сновні принципи та поняття Git.

*Репозиторій (Repository).* Репозиторій є основною одиницею зберігання проєкту в Git. Він містить всю історію змін, гілки (branches) та мітки (tags). Репозиторій може бути локальним (на вашому комп'ютері) або віддаленим (на сервері).

*Коміт (Commit).* Коміт представляє собою фіксацію змін у репозиторії. Він включає в себе змінені файли, повідомлення про коміт (яке пояснює, що було змінено) та автора коміту. Коміти дозволяють відстежувати, коли, хто і які зміни були внесені у проєкт.

*Гілки (Branches).* Гілки дозволяють вам створювати окремі лінії розвитку проєкту. Кожна гілка може містити свою власну версію файлів та комітів. Використання гілок дозволяє розробникам працювати над різними функціями або виправленнями помилок паралельно та злити їх пізніше.

*Злиття (Merge).* Злиття в Git дозволяє об'єднувати зміни з однієї гілки в іншу. Це часто використовується для об'єднання розроблених функцій або виправлень помилок в основну гілку проєкту.

*Віддалені репозиторії (Remote Repositories).* Віддалені репозиторії

знаходяться на сервері або хмарному сервісі і дозволяють спільно працювати над проєктом декільком розробникам. Вони забезпечують можливість синхронізації змін між різними копіями репозиторію.

Git є дуже потужною і гнучкою системою контролю версій, яка здатна впоратися з різними типами проєктів та командами розробників. Вона має широкий набір команд та можливостей, які дозволяють ефективно використовувати її для керування версіями проєкту та спільної роботи над ним.

Зокрема, Git підтримує так звані Git hooks. Git hooks – це скрипти, які можна виконувати автоматично на певній події в Git репозиторії. Вони дозволяють вам впроваджувати власні дії або виконувати перевірки перед або після певних операцій, таких як коміт, злиття гілок або отримання змін з віддаленого репозиторію [Pro Git].

Git hooks діляться на два типи: серверні та клієнтські.

*Серверні hooks*. Ці hooks виконуються на сервері, коли виконується певна операція, наприклад, при отриманні змін з віддаленого репозиторію або при злитті гілок. Деякі з популярних серверних hooks включають pre-receive, update і post-receive. Ви можете використовувати серверні hooks для виконання автоматичних перевірок, валідації коду або виконання спеціальних дій перед або після прийняття змін на сервері.

*Клієнтські hooks*. Ці hooks виконуються на стороні клієнта перед або після певних подій. Наприклад, pre-commit hook виконується перед комітом змін, і ви можете використовувати його для перевірки коду, форматування файлів або виконання автоматичних тестів. Інші популярні клієнтські hooks включають pre-push та post-checkout. Клієнтські hooks корисні для забезпечення якості коду, дотримання стандартів програмування та автоматизації процесу розробки.

Git hooks дозволяють вам налаштувати автоматичні перевірки, тестування та дії, які сприяють поліпшенню якості коду, забезпечують дотримання стандартів та допомагають уникнути проблем при спільній роботі над проєктом. Ви можете використовувати існуючі hooks або створювати власні, щоб виконувати специфічні завдання, які відповідають вашим потребам розробки.

При розробці RESTful API системи мережі фітнес-центрів було створено декілька таких скриптів. Для того, щоб тримати код у правильному стилі був розроблений скрипт, що представлено на рисунку 1.1.

```
#!/bin/sh
cmd="./vendor/bin/pint --test"
$cmd
status=$?
if [ $status -ne 0 ]
then
    ./vendor/bin/pint
    exit 1
fi
```

Рисунок 1.1 – Файл pre-commit

Цей скрипт виконується перед комітом (pre-commit). Таким чином перед тим, як зафіксувати певні зміни запускається pint скрипт для виправлення стилю коду, і якщо pint знайшов та вніс свої зміни для зберігання єдиного стилю, то коміт скасовується.

Після цього потрібно знову виконати коміт. Отже, цей Git hook скрипт підтримує стан коду в єдиному стилі.

Ще один Git hook скрипт (рис. 1.2) запускає автоматизовані тести перед тим, як відправляти зміни до віддаленого репозиторію (pre-push). Тобто якщо всі автоматизовані тести успішно проходять, то внесені зміни потрапляють до віддаленого репозиторію. А якщо хоча б один тест не проходить, то потрібно виправити всі помилки в коді, і тільки після того відправляти зміни до віддаленого репозиторію.

Таким чином, у віддаленому репозиторії завжди буде зберігатись працюючий код.

```
#!/bin/sh

cmd="php artisan test"
$cmd
status=$?

if [ $status -ne 0 ]
then
    echo "Tests failed"
    exit 1
fi
```

Рисунок 1.2 – Файл pre-push

**Висновки до розділу 1**

В цьому розділі було розглянуто засоби веброзробки, а саме мова програмування PHP та її фреймворк Laravel, використані бібліотеки цього фреймворку, зазначено що таке API, зокрема розглянуто його стандарт REST, наведено інформацію про систему контролю версій Git та її основні принципи. Також описано механізм Git hooks з відповідними прикладами, які плануються використовуватись при розробці API системи фітнес-центрів.

# 2 ПРОЄКТУВАННЯ СИСТЕМИ

## 2.1 Опис предметної області

Предметна область системи мережі фітнес-центрів охоплює всі аспекти, які пов’язані з керуванням та функціонуванням фітнес-центрів. Вона включає різні сутності, процеси та взаємодії, які відбуваються у цьому середовищі.

Філія – це одна з основних сутностей предметної області. Філії можуть мати різні локації – Зали, де реалізуються різноманітні послуги, такі як групові та індивідуальні заняття або SPA-послуги.

Ще однією з ключових сутностей є Абонемент. Абонементи визначають доступ клієнтів до послуг, які надає фітнес-центр.

Існують різні типи абонементів, такі як стандартний (діє під час певного періоду та має обмеження в кількості відвідувань), безлімітний (діє під час певного періоду, але не має обмежень в кількості відвідувань), а також free way (діє під час певного періоду, має обмеження в кількості відвідувань, але дозволяє відвідувати усі види послуг). Таким чином, абонементи можуть мати різні умови, обмеження та дати дії.

Клієнти представляють людей, користуються послугами фітнес-центрів. Вони можуть мати різні персональні дані, такі як ім’я, прізвище, номер телефону, дата народження.

Робітники – це фахівці, які надають надають індивідуальні консультації та проводять тренування клієнтам, або, наприклад, надають SPA-послуги.

Тренери можуть мати різні спеціалізації та сертифікації.

Розклади занять визначають час та місце проведення тренувань, групових занять та інших подій фітнес-центру. Події можуть бути регулярними чи ні та включати різні види активностей.

Також важливим аспектом є контроль платежів. Платежі включають оплату абонементів та інших витрат, пов’язаних з фітнес-центром. Слід ще

зазначити, що платежі можуть включати різні методи оплати, такі як готівка, кредитні картки, електронні платіжні системи.

Для власника такого бізнесу важливим є аналізувати статистику різних показників для прийняття рішень стосовно розвитку своєї справи. Збір та аналіз даних про продажі, відвідуваність, популярність послуг та інші показники є необхідними для ефективного управління філіями. Звіти та аналітика допомагають зрозуміти стан бізнесу, приймати рішення та планувати розвиток.

## 2.2 Концептуальне проєктування

На даному етапі слід зазначити, що в системі повинен бути Адміністратор системи, який буде отримувати повідомлення про помилки, зможе керувати користувачами системи, призначати їм певні ролі, а для цих ролей налаштовувати повноваження.

Повноваження вирішують задачу надання та обмеження доступу до певного функціоналу системи в залежності відповідно від наявності чи відсутності цих повноважень.

Далі Директор повинен мати змогу керувати філіями та залами в них, переглядати список платежів, контролювати типами подій на розкладі, мати доступ до перегляду різноманітних статистичних показників.

Менеджер повинен мати можливість продавати абонементи, керувати подіями на розкладі, створювати нових клієнтів в системі. А робітники повинні мати змогу дивитись власний розклад та відмічати відвідування клієнтів. Загальну структуру продемонстровано на рисунку 2.1.

Розглянемо більш детально процес продажу абонементів та процес відмітки відвідування. Після того, як клієнт звертається до менеджера для придбання абонементу, менеджер заповнює форму продажу, вказуючи клієнта, тип послуги, робітника, який буде надавати послугу, кількість відвідувань, система перевіряє на валідність отримані дані та відображає відповідь про це.

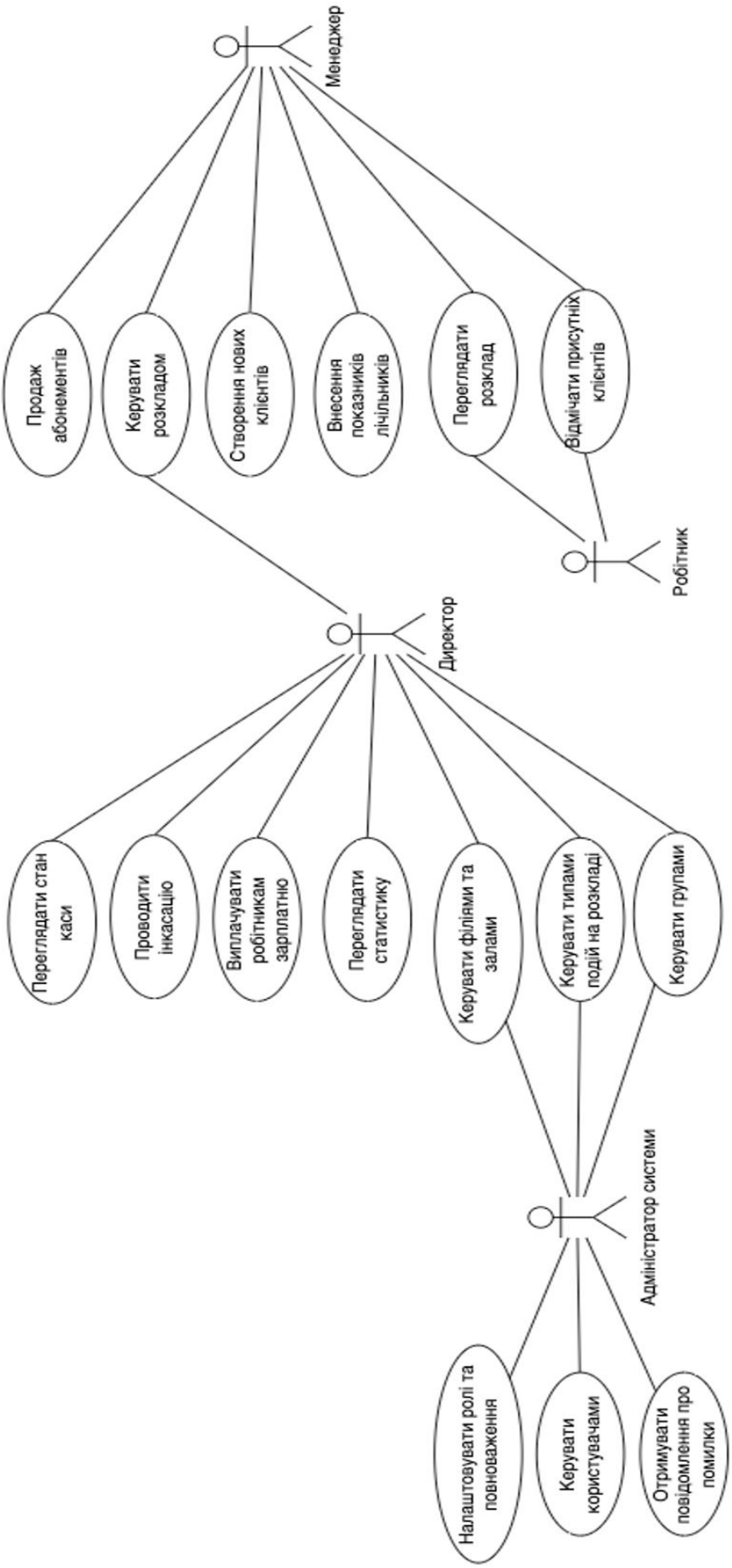


Рисунок 2.1 – Діаграма акторів та прецедентів

Після того, як дані були перевірені, менеджер підтверджує продаж, вказуючи суму та метод оплати клієнтом. Цей процес зображено на діаграмі послідовності на рисунку 2.2.

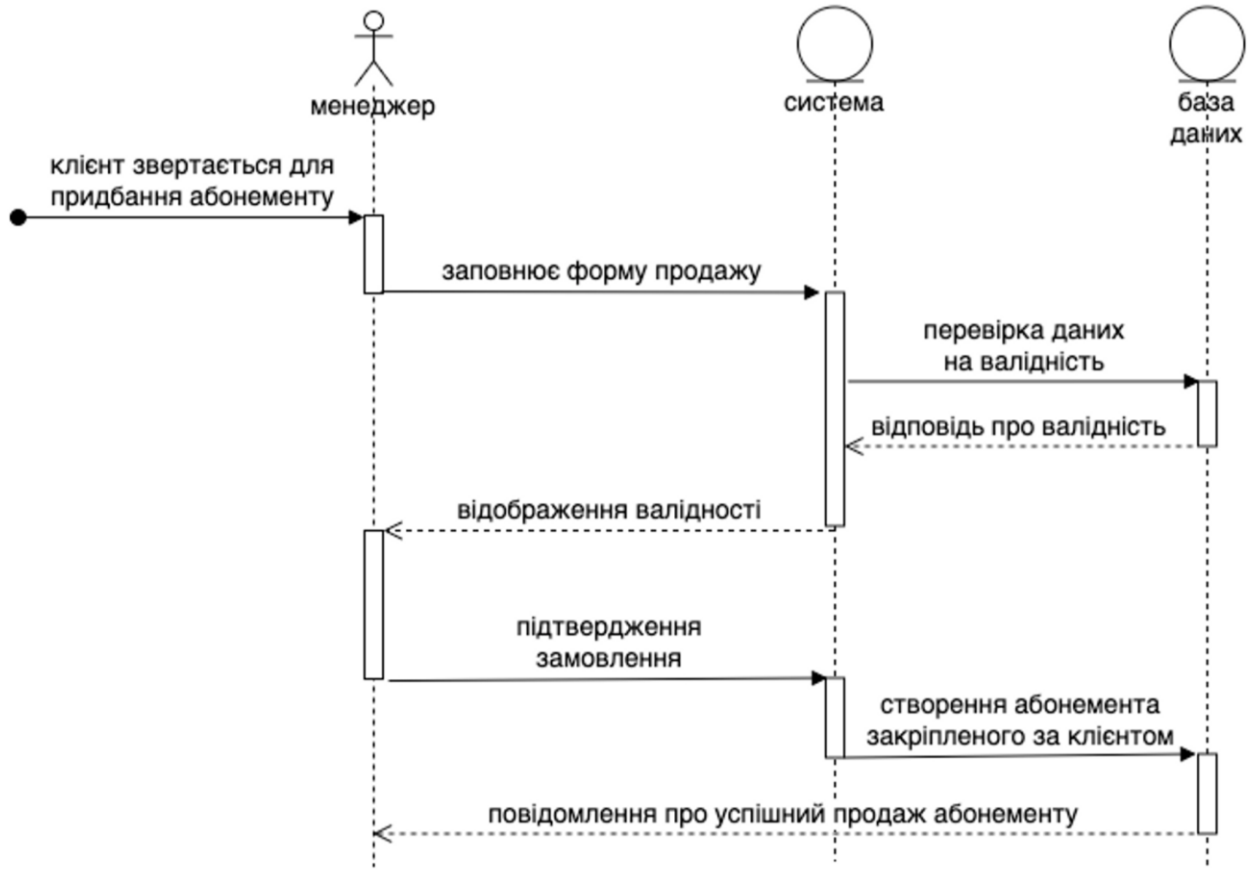


Рисунок 2.2 – Діаграма послідовності продажу абонементів

Однією з найчастіших операцій є відмітка відвідувань клієнтів, оскільки потрібно контролювати який клієнт скільки занять ще може відвідати. Ця операція продемонстрована на рис. 2.3.

З цієї схеми видно, що після успішного створення відмітки відвідування також створюється і нарахування в зарплатню робітника.

Таким чином, система повинна контролювати не тільки кількість доступних занять в абонементі, а ще й розраховувати зарплатню тренерам, яка залежить від кількості проведених ним тренувань та кількості клієнтів, з якими він працює.

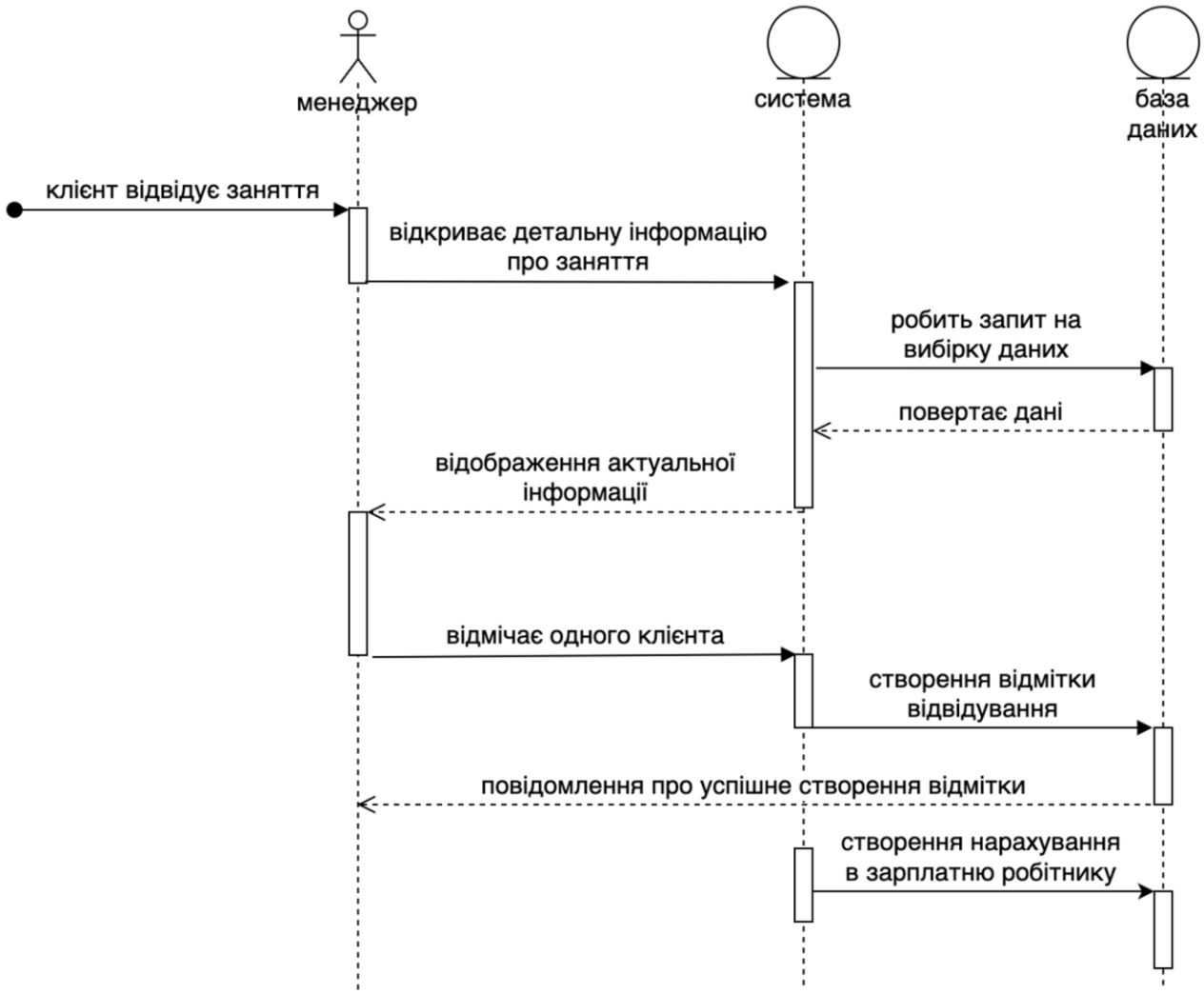


Рисунок 2.3 – Діаграма послідовності відмітки відвідувань

## 2.3 Логічне проєктування

На даному етапі слід розробити діаграму сутностей-відношень, де будуть перераховані всі основні сутності, вказані їх атрибути та зображені зв’язки між цими сутностями.

Виходячи з наведеної діаграми (рис. 2.4), маємо, що є Філія, в яку може входити декілька Залів. В кожному з Залів можуть проводитись Події різних Типів Послуг. Кожну Подію проводить певний робітник (Користувач), якому буде нараховуватись Зарплатня в залежності від кількості Відвідувань клієнтів.

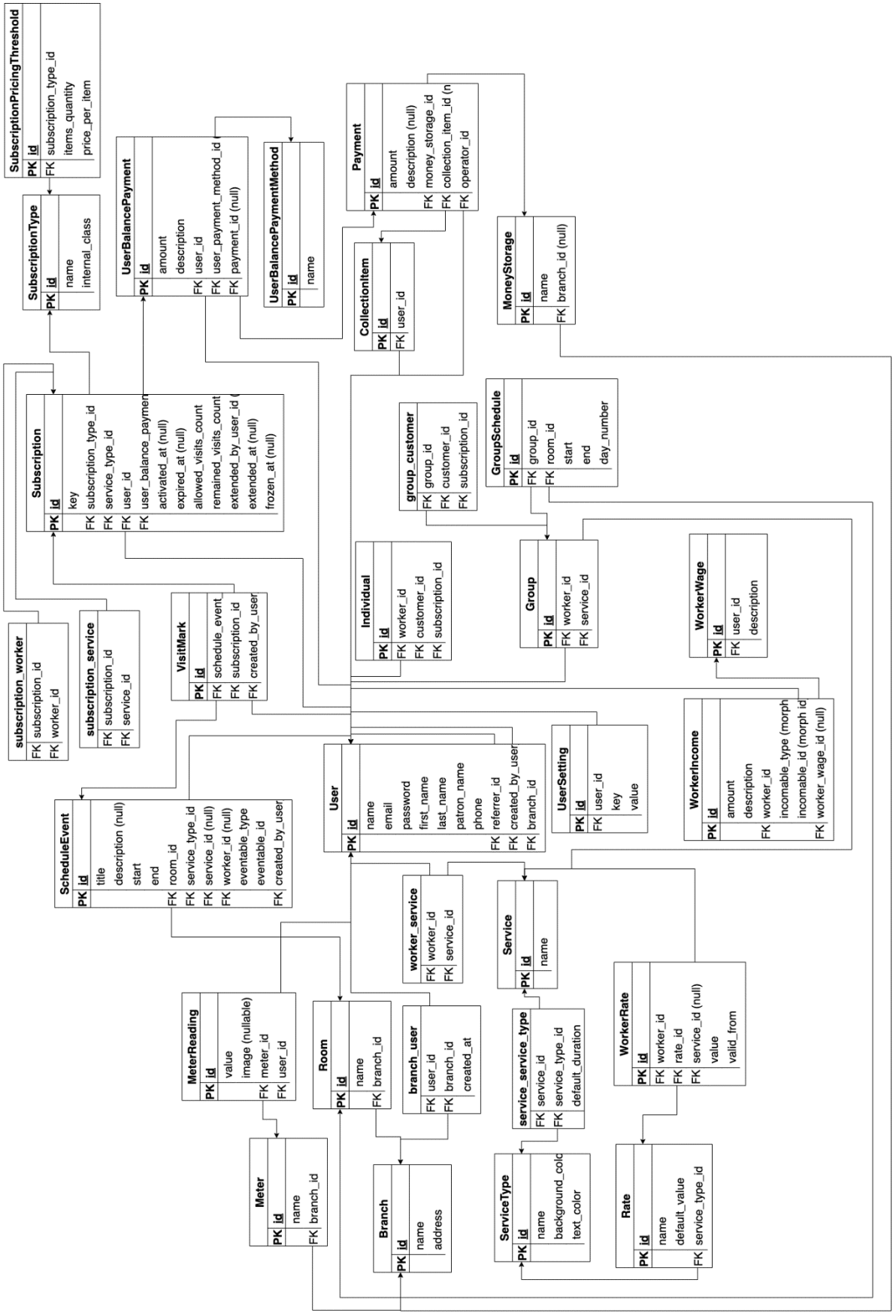


Рисунок 2.4 – Діаграма сутностей-відношень

Для того щоб відвідати певну Подію, клієнту потрібно придбати Абонемент, який характеризується своїм типом, строком дії, кількістю занять, Послугою, яку клієнт буде відвідувати і так далі. Також в базі даних будуть зберігатись усі Платежі, зроблені клієнтами за допомогою різних Методів Оплати. Кожен Платіж закріплений за своєю Касою, що дозволить отримувати інформацію про поточний баланс грошей на кожній Філії.

### 2.4. Фізичне проєктування

Для розуміння того, як буде працювати система загалом та як вона буде взаємодіяти з клієнтської програмою наведемо діаграму розгортання (рис. 2.5).

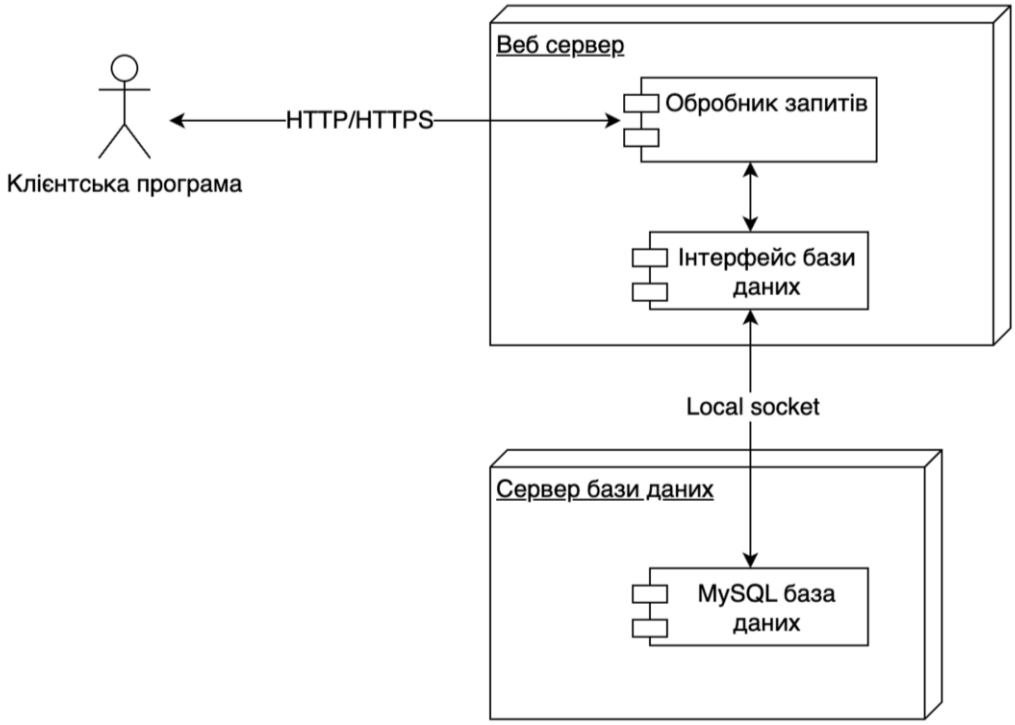


Рисунок 2.5 – Діаграма розгортання

Таким чином, клієнтська програма робить запит до серверу за протоколом HTTP, вебсервер обробляє цей запит, звертаючись до бази даних за допомогою інтерфейсу бази даних, та повертає результат обробки клієнту у форматі JSON.

## Висновки до розділу 2

Отже, в цьому розділі було описано предметну область, наведено діаграму акторів та прецедентів, діаграми послідовності для процесу продажу абонементів та процесу додавання відміток відвідування, описані сутності та зв’язки між ними за допомогою ER-діаграми та зображено взаємодію клієнтської програми з серверною частиною за допомогою діаграми розгортання.

# 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Для наочної перевірки роботи розробленого API доречно використовувати Postman. Postman – це популярний інструмент для розробки, тестування та документування API. Він надає зручну та потужну робочу зону для взаємодії з різними API та виконання різноманітних завдань [12].

Розглянемо, що включають основні функціональні можливості Postman.

*Відправка HTTP-запитів*. Postman дозволяє легко створювати та відправляти HTTP-запити, такі як GET, POST, PUT, DELETE і багато інших. Ви можете налаштувати різні параметри запиту, такі як заголовки, параметри, тіло запиту та автентифікацію.

*Тестування API*. Postman дозволяє вам створювати автоматичні тести для перевірки функціональності вашого API. Ви можете встановлювати перевірки на основі статусу відповіді, значень полів або регулярних виразів. Це допомагає переконатися, що ваше API працює належним чином.

*Середовища*. Postman дозволяє створювати різні середовища для ваших API, наприклад, розробка, тестування, виробництво тощо. Ви можете налаштувати змінні середовища, які дозволяють змінювати значення під час виконання запитів. Це зручно для роботи з різними конфігураціями вашого API.

*Створення документації*. Postman дозволяє генерувати документацію для вашого API на основі ваших запитів та колекцій. Ви можете створювати описові коментарі, додавати приклади використання та автоматично генерувати документацію у форматі HTML або Markdown.

*Спільна робота*. Postman надає можливість спільної роботи з командою. Ви можете зберігати та синхронізувати ваші колекції, середовища та змінні у хмарному сервісі Postman, що дозволяє вашій команді спільно використовувати та співпрацювати над проєктами.

Postman є потужним інструментом для розробки та тестування API, який дозволяє зробити процес розробки більш ефективним та продуктивним. Він

надає зручність, широкі можливості налаштування та спрощує взаємодію з API для розробників та тестувальників.

### 3.1 Реалізація CRUD операцій філій

CRUD – це базовий набір операцій, які можна виконувати з даними: C (create) – для створення об’єктів, R (read) – читання об’єктів, U (update) – редагування об’єктів, D (delete) – видалення об’єктів.

Розглянемо CRUD операції на прикладі моделі Філії. Оскільки фреймворк Laravel використовує шаблон MVC, то вся реалізація цих операцій знаходиться у файлі контролеру. Для демонстрації основних можливостей взаємодії з магазином розглянемо наступні методи контролеру:

– Index (отримання даних про всі філії);
– Store (створення філії);
– Show (отримання даних однієї конкретної філії);
– Update (редагування даних конкретної філії);
– Destroy (видалення філії).

Для того, щоб була можливість зробити HTTP-запит, який оброблюватиме контролер, спочатку потрібно додати маршрути в файлі routes/api.php (рис. 3.1).

```
use App\Http\Controllers\Api\BranchController;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:sanctum')->group(function () {
    // ...
    Route::apiResource('branches', BranchController::class);
    // ...
});
```

Рисунок 3.1 – Файл routes/api.php

Відповідно до офіційної документації Laravel, метод apiResource класу Route автоматично створить наступні маршрути (табл. 3.1).

Таблиця 3.1 – Маршрути для CRUD операцій

| Маршрут | HTTP метод | Метод контролеру |
|---|---|---|
| /api/branches | GET | Index |
| /api/branches | POST | Store |
| /api/branches/{id} | GET | Show |
| /api/branches/{id} | PUT/PATCH | Update |
| /api/branches/{id} | DELETE | Destroy |

Метод index контролеру повинен обробити запит та повернути список філій (рис. 3.2).

```
public function index(FilterRequest $filterRequest, BranchFilter $filters) {
        $this->authorize('Branch:access');
        $branchs = Branch::query()
           ->filter($filters)
           ->paginate($filterRequest->get('per_page'));
        return IndexCollection::make($branchs);
}
```

Рисунок 3.2 – Метод index контролеру

Оскільки філій потенційно може бути багато, тому слід завантажувати та повертати тільки потрібну їх кількість з можливістю довантажувати інші “сторінки” за необхідності. В той же час, потрібно ще фільтрувати ці філії, наприклад за їх назвами. У файлі app\Http\Requests\Branch\FilterRequest.php описані параметри, за якими можна фільтрувати та сортувати дані (рис. 3.3). В даному випадку доступна тільки назва філії.

```
// ...
public function rules() {
    return [
        'page' => ['nullable', 'integer'],
        'per_page' => ['nullable', 'integer'],
         'columnFilters.name' => ['nullable'],
        'sort' => ['nullable', 'array'],
        'sort.*.field' => ['nullable', 'string', Rule::in(['name'])],
        'sort.*.type' => ['nullable', 'string', Rule::in(['asc', 'desc'])],
    ];
}
// ...
```

Рисунок 3.3 – Опис параметрів фільтрації та сортувань

Безпосередньо логіка цієї фільтрації та сортування реалізована у файлі app\Filters\BranchFilter.php (рис. 3.4).

```
class BranchFilter extends BaseFilter {
    public function name($value) {
        $this->builder->where('name', 'like', '%'.$value.'%');
    }
    public function sort_by_name($direction) {
        $this->builder->orderBy('name', $direction);
    }
}
```

Рисунок 3.4 – Логіка фільтрації та сортування філій

Також слід зазначити, що перед вибіркою даних контролер перевіряє, чи є у користувача, який робить запит, повноваження, які дозволяють йому отримати список філій. Це реалізовано за допомогою виклику методу authorize, який за

замовчуванням присутній у Laravel контролерах.

Якщо у користувача немає потрібних повноважень, то йому повертається HTTP відповідь зі статусом 403 Forbidden. А якщо повноваження є, то робиться запит до бази даних використанням фільтрів та сортувань і передається у якості відповіді користувачу у відформатованому вигляді.

Форматування відповіді виконується за допомогою файлу app\Http\Resource\Branch\IndexCollection.php (рис. 3.5).

```
class IndexCollection extends ResourceCollection {
    protected $collection_counter;
    public function toArray($request) {
        $this->collection_counter = $request->per_page * ($request->page - 1);
        $branches = $this->collection->map(function ($branch) {
            $this->collection_counter++;
            return [
                'index' => $this->collection_counter,
                'id' => $branch->id,
                'name' => $branch->name,
            ];
        });
        return $branches;
    }
}
```

Рисунок 3.5 – Логіка форматування відповіді

Отже, у HTTP відповіді для кожної філії буде встановлений порядковий номер (index), ідентифікатор, який є первинним ключем у базі даних (id) та назва філії.

Приклад відповіді зображено на рис. 3.6.

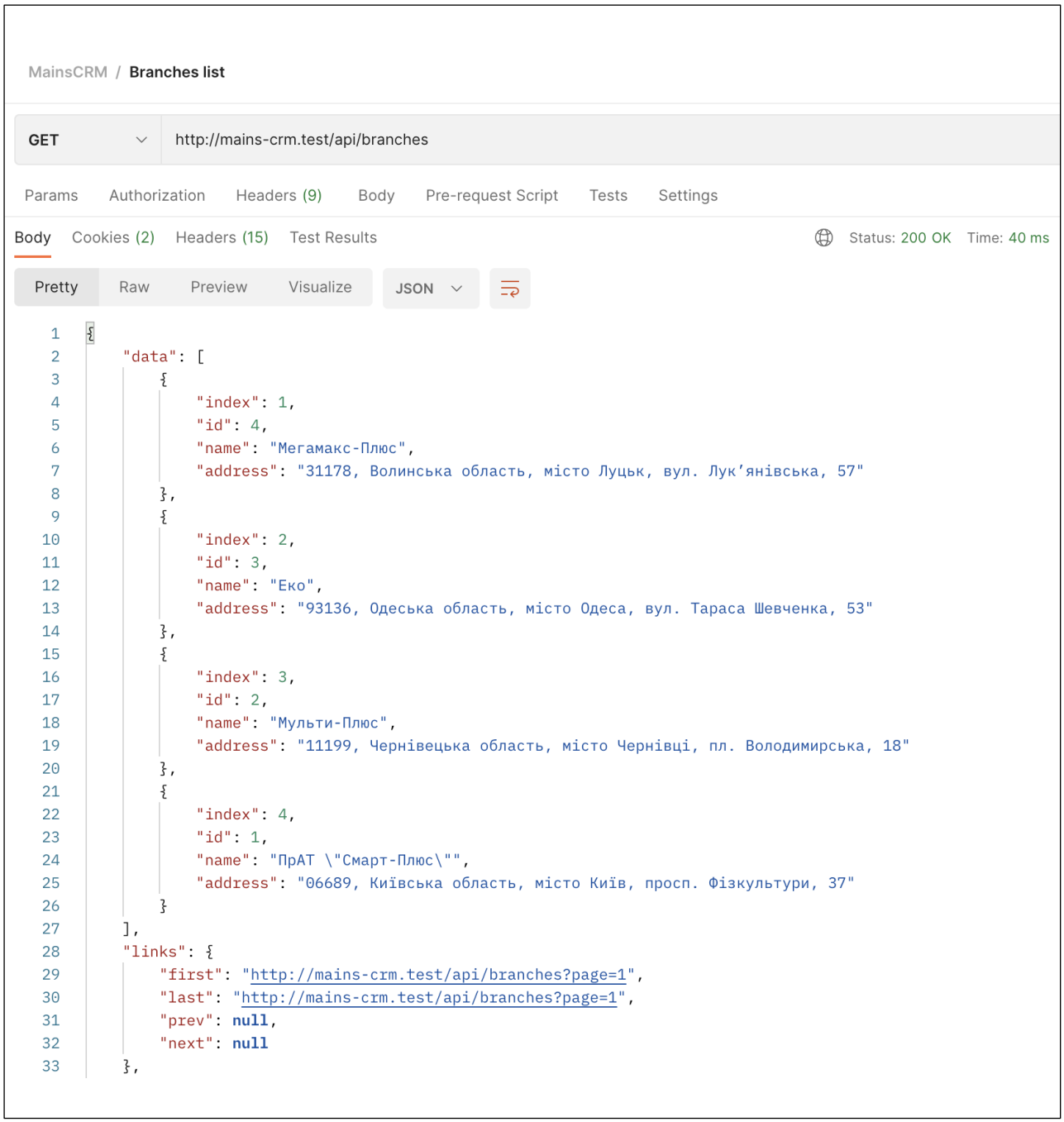


Рисунок 3.6 – Приклад відповіді у форматі JSON

Наступною операцією є створення філії (метод store контролеру).

Для цього клієнтська програма повинна надіслати POST запит із параметрами name та address.

Ці параметри та правила їх валідації знаходяться у файлі app\Http\Requests\Branch\StoreRequest.php (рис. 3.7).

```
class StoreRequest extends FormRequest {
    public function rules() {
        return [
            'name' => ['required', 'string'],
            'address' => ['nullable', 'string'],
        ];
    }
}
```

Рисунок 3.7 – Правила валідації створення філій

Аналогічним до попереднього розглянутого методу, метод store спочатку перевіряє чи є у користувача повноваження, щоб створювати філію. Після цього створюється філія, використовуючи передані дані, які пройшли перевірку. І насамперед клієнтській програмі повертається відповідь з інформацією про тільки що створену філію (рис. 3.8). Приклад відповіді, який отримає клієнтська програма після створення філії, продемонстровано на рис. 3.9.

```
public function store(StoreRequest $request) {
    $this->authorize('Branch:create');
    $branch = Branch::create($request->validated());
    return DefaultResource::make($branch);
}
```

Рисунок 3.8 – Метод store контролеру

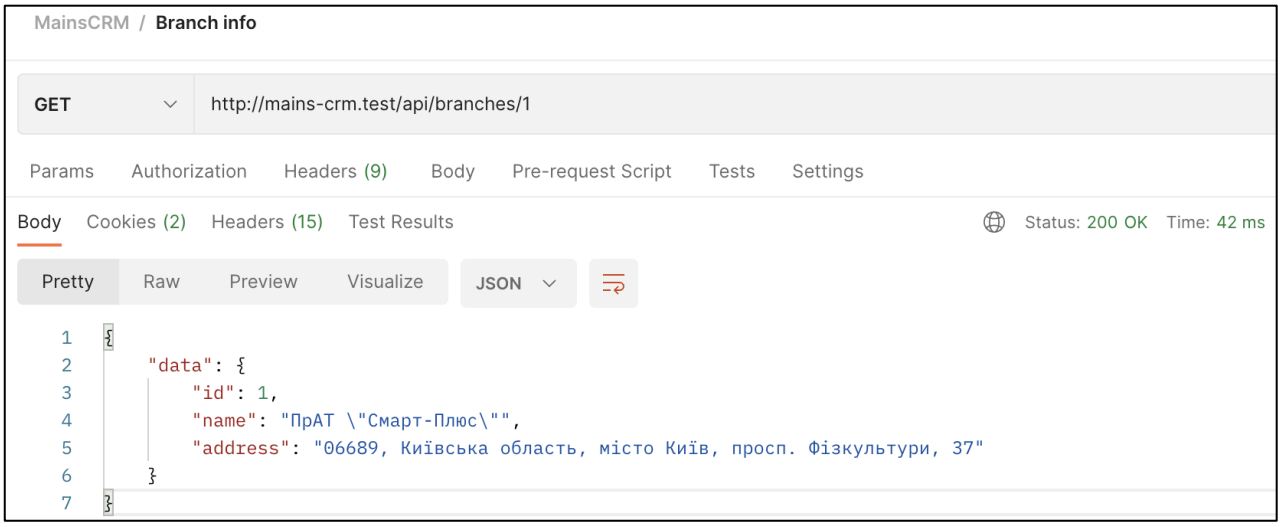


Рисунок 3.9 – Приклад відповіді про одну філію

Якщо необхідно просто отримати інформацію про існуючу філію, то потрібно відправити GET запит за маршрутом /api/branches/{id}. Таким чином, підставивши число 3 замість id, відповідь від системи буде така ж сама (рис. 3.9). Реалізація методу show контролеру наведена на рис. 3.10.

```
public function show(Branch $branch) {
    $this->authorize('Branch:view');
    return DefaultResource::make($branch);
}
```

Рисунок 3.10 – Метод show контролеру

Редагування філії відбувається схожим до процесу створення чином. Різниця полягає лише в маршруті (потрібно вказати яку саму філію треба відредагувати) та в HTTP методі. Якщо для створення використовується POST, то для редагування потрібно застосовувати PUT або PATCH. Правила валідації параметрів в даному випадку не відрізняються від тих, що були при створенні. А сама реалізація методу update контролеру зображена на рис. 3.11.

```
public function update(UpdateRequest $request, Branch $branch) {
    $this->authorize('Branch:edit');
    $branch->update($request->validated());
    return DefaultResource::make($branch);
}
```

Рисунок 3.11 – Метод update контролеру

Остання дія, яку буде розглянуто з CRUD операцій, – це видалення філії. За цю операцію відповідає метод destroy контролеру. Все, що він робить, – перевіряє повноваження на видалення філії у користувача та безпосередньо видаляє філію, якщо необхідні повноваження присутні. Код цього методу наведено на рис. 3.12.

```
    public function destroy(Branch $branch) {
        $this->authorize('Branch:delete');
        $branch->delete();
        return response()->noContent();
    }
```

Рисунок 3.12 – Метод delete контролеру

У відповідь клієнтська програма отримає “пусту відповідь” зі статусом 204 No content (рис. 3.13).

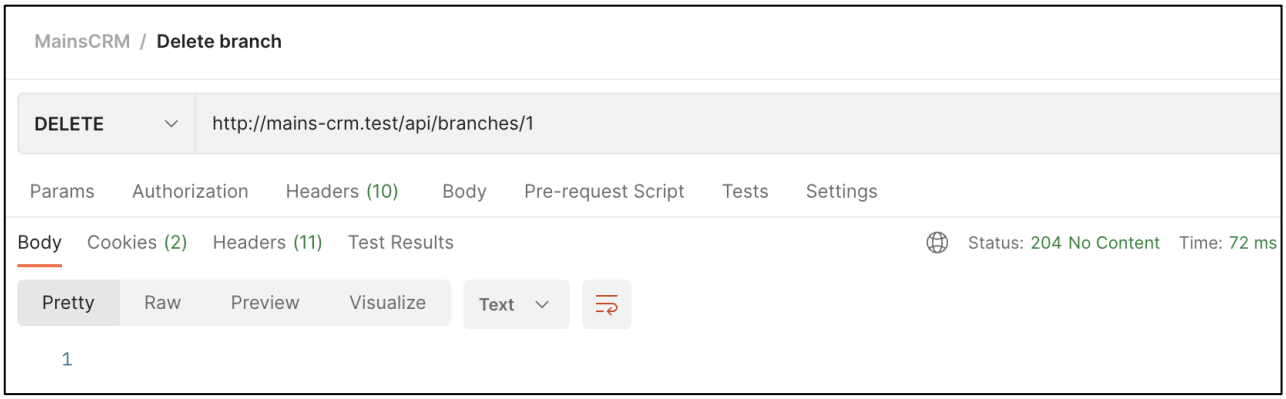


Рисунок 3.13 – Приклад відповіді про видалення філії

### 3.2 Реалізація процесу продажу абонементів

Перед безпосередньо реалізацією потрібно зрозуміти, як процес продажу абонементів повинен відбуватись. Виходячи з діаграми сутностей-відношень (рис. 2.4) маємо, що клієнт спочатку повинен поповнити свій баланс, а потім ці кошти списуються для придбання абонементу. Це дає змогу клієнту сплатити декількома методами за один і той самий абонемент, наприклад, половину готівкою, іншу половину – з карти. Також слід передбачити можливість продавати декілька абонементів одному клієнту одразу. Ще одним важливим моментом є те, що абонемент можна забронювати. Загалом, процес продажу абонементів представлено на рис. 3.14.

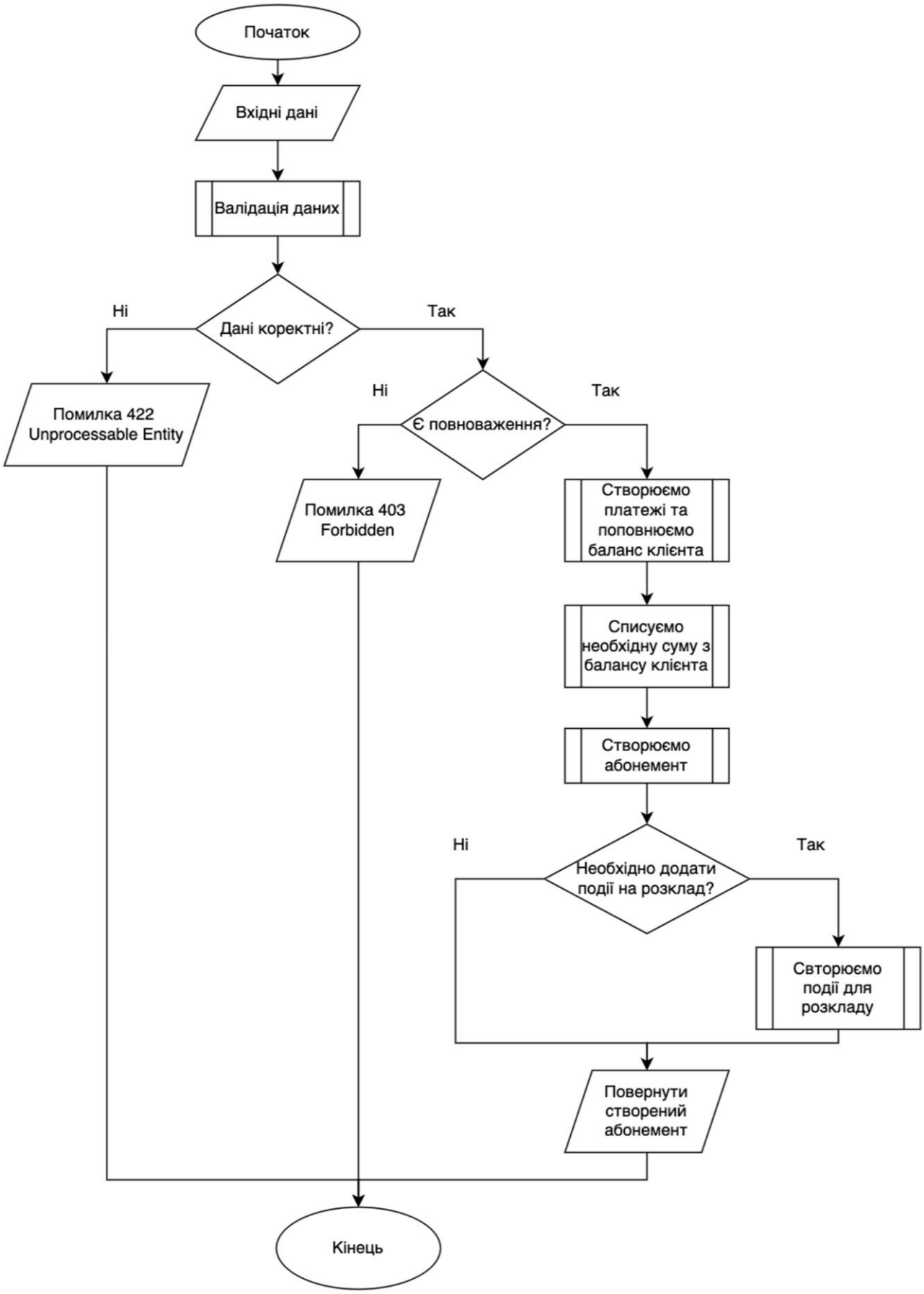


Рисунок 3.14 – Блок-схема продажу абонементів

При валідації даних критично важливим є перевірка усіх частин оплати, щоб їх сума збігалась з загальною ціною абонементу. А також, при частковій оплаті з балансу клієнта потрібно перевірити, щоб вказана сума була присутньою на балансі.

Якщо були передані некоректні дані, то клієнтській програмі буде повернута HTTP відповідь зі статусом 422 Unprocessable Entity з перерахуванням усіх полів, які не пройшли валідацію. В противному випадку, коли всі дані пройшли перевірку, то далі обробку виконує метод контролеру.

Спочатку виконується перевірка на наявність повноважень, а потім вже безпосередньо створюються платежі (якщо абонемент не бронюється, а продається), поповнюється баланс клієнта, створюється абонемент, списуються кошти з балансу клієнта, додаються події на розклад за необхідності, і нарешті повертається відповідь про створений абонемент.

Оскільки на даному етапі виконується досить велика кількість дій пов’язаних з базою даних, то в даному випадку доцільно використовувати транзакції бази даних.

Транзакція – це група послідовних операцій з базою даних, яка є логічною одиницею роботи з даними. Транзакція може бути виконана або цілком і успішно, або не виконана зовсім, і тоді вона не може справити ніякого ефекту.

Транзакції з базою даних використовуються для досягнення наступних цілей [3]:

1) забезпечення надійних робочих елементів, які дозволяють правильно відновити роботу у випадку збоїв та зберігати цілісність бази даних у випадку системних відмов, коли виконання операцій зупиняється (повністю або частково) і більшість операцій над базою даних залишаються незавершеними з нез’ясованим статусом;
2) для забезпечення роздільного доступу для процесів, що одночасно звертаються до бази даних (при відсутності ізоляції операцій результати, отримані процесами, можуть бути помилковими).

Розглянемо реалізацію методу контролеру, який відповідає за обробку

процесу продажу абонементів. Отже, спочатку відбувається перевірка наявності повноважень та отримуються перевірені дані (рис. 3.15).

```
$this->authorize('Subscription:create');
$data = $request->validated();
```

Рисунок 3.15 – Перевірка повноважень та отримання перевірених даних

Після цього створюються абонементи всередині транзакції і результат повертається у певному форматі (рис. 3.16).

```
$subscriptions = DB::transaction(function () use ($data) {
    // ...
});
return response()->json(['data' => DefaultResource::collection($subscriptions)], 201);
```

Рисунок 3.16 – Транзакція бази даних та повернення відповіді

Всередині транзакції спочатку створюються платежі та поповнюється баланс клієнт при умові, якщо абонемент продається, а не був забронюваний (див. рис. 3.17).

Далі в циклі створюються абонементи, виходячи з переданих даних, а потім ці абонементи повертаються в якості результату виконання транзакції (див. рис. 3.18).

```
        $amount_sum = 0;
        if (! $data['is_reservation']) {
            foreach ($data['payment_parts'] as $payment_part) {
                if ($payment_part['user_balance_payment_method_id'] === null) {
                    $amount_sum += $payment_part['amount'];
                    continue;
                }
                $payment = Payment::create([
                    'amount' => $payment_part['amount'],
                    'description' => 'Поповнення балансу клієнта для продажу абонементу',
                    'money_storage_id' => $payment_part['money_storage_id'],
                    'operator_id' => auth()->id(),
                ]);
                $user_balance_payment = UserBalancePayment::create([
                    'amount' => $payment_part['amount'],
                    'description' => 'Поповнення балансу для покупки абонементу',
                    'user_id' => $data['user_id'],
                    'user_balance_payment_method_id' => $payment_part['user_balance_payment_method_id'],
                    'payment_id' => $payment->id,
                ]);
                $amount_sum += $payment_part['amount'];
            }
        }
```

Рисунок 3.17 – Створення платежів та поповнення балансу

```
        $subscriptions = collect();
        foreach ($data['subscriptions'] as $subscription_data) {
          // ...
                $subscriptions->push($subscription);
        }
        return $subscriptions;
```

Рисунок 3.18 – Створення абонементів

В самому циклі спочатку генерується код абонемента (рис. 3.19).

```
$current_time = now();
$subscription_key = $current_time->timestamp;
```

Рисунок 3.19 – Генерація коду абонементу

Після цього з балансу клієнта списується необхідна сума, знову ж таки за умови, що це саме продаж, а не бронювання (рис. 3.20).

```
$user_balance_payment = null;
if (! $data['is_reservation']) {
    $user_balance_payment = UserBalancePayment::create([
        'amount' => -$amount_sum,
        'description' => 'Списання з балансу для покупки абонементу #'.$subscription_key,
        'user_id' => $data['user_id'],
    ]);
}
```

Рисунок 3.20 – Генерація коду абонементу

Далі безпосередньо створюється абонемент (рис. 3.21).

```
$subscription = Subscription::create([
    'key' => $subscription_key,
    'subscription_type_id' => $subscription_data['subscription_type_id'],
    'service_type_id' => $subscription_data['service_type_id'],
    'user_id' => $data['user_id'],
    'user_balance_payment_id' => $user_balance_payment?->id,
    'price' => $subscription_data['price'],
    'allowed_visits_count' => $subscription_data['allowed_visits_count'],
    'remained_visits_count' => $subscription_data['allowed_visits_count'],
]);
```

Рисунок 3.21 – Створення абонементу

Потім до абонемента додаються робітники, які можуть обслуговувати даного клієнта, а також послуги, які цей клієнт може відвідувати (рис. 3.22).

```
        if (array_key_exists('worker_ids', $subscription_data) &&
$subscription_data['worker_ids']) {
            $subscription->workers()->sync($subscription_data['worker_ids']);
        }
        if (array_key_exists('service_ids', $subscription_data) &&
$subscription_data['service_ids']) {
            $subscription->services()->sync($subscription_data['service_ids']);
        }
        $subscription->load(['workers', 'services']);
```

Рисунок 3.22 – Додавання обмежень до абонемента

Якщо був проданий абонемент, де передбачається персональне надання послуги (наприклад, індивідуальне тренування), то під час продажу можна одразу додати на розклад ці події (рис. 3.23).

```
        if (array_key_exists('events', $subscription_data) &&
$subscription_data['events']) {
            $worker = $subscription->workers->first();
            $service = $subscription->services->first();
            //…
            foreach ($subscription_data['events'] as $event) {
                $event['title'] = $worker?->full_name;
                ScheduleEvent::create($event);
            }
        }
        $subscription->load(['schedule_events']);
```

Рисунок 3.23 – Додавання подій на розклад

Приклад відповіді на запит продажу абонементу зображено на рисунку 3.24.

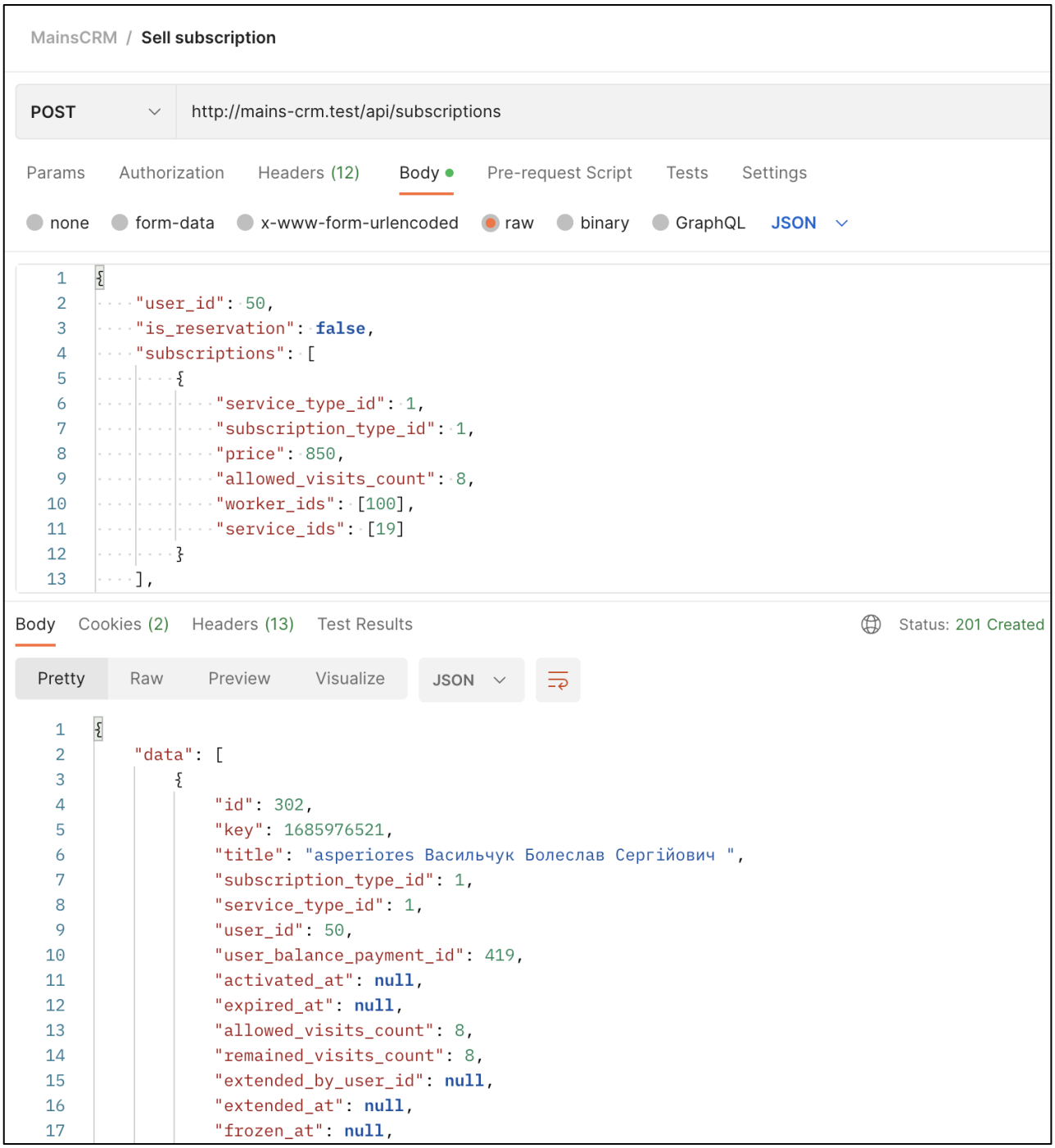


Рисунок 3.24 – Приклад відповіді після продажу абонементу

Таким чином було повністю розглянуто реалізацію процесу продажу абонементів.

### 3.3 Тестування системи

Тестування є важливою частиною розробки програмного забезпечення, включаючи проєкти, розроблені з використанням фреймворку Laravel. Laravel надає потужні інструменти для тестування, а основним інструментом для написання та виконання тестів є бібліотека PHPUnit.

PHPUnit – це популярний фреймворк для тестування в PHP, який забезпечує набір класів та методів для написання тестів. Laravel використовує PHPUnit як основний фреймворк для тестування своїх додатків.

Розглянемо, що включають основні можливості тестування в Laravel з використанням PHPUnit [10].

*Юніт-тести*. PHPUnit дозволяє писати юніт-тести для окремих класів, методів та функцій вашого додатку. Ви можете перевіряти правильність поведінки окремих компонентів та їх взаємодію з іншими частинами системи.

*Інтеграційні тести.* Laravel надає зручні засоби для написання інтеграційних тестів, які дозволяють перевірити взаємодію різних компонентів системи, таких як маршрути, контролери, моделі та база даних. Це дозволяє вам перевірити, чи працюють всі частини системи разом правильно.

*Функціональні тести*. За допомогою Laravel та PHPUnit ви можете створювати функціональні тести, які симулюють дії користувача та перевіряють правильність роботи системи на рівні вебінтерфейсу. Ви можете тестувати маршрути, форми, валідацію даних та інші аспекти вебдодатку.

*Мокування*. PHPUnit дозволяє створювати моки (mocks) та заглушки (stubs), які допомагають тестувати окремі частини системи, відділені від залежностей. Це дозволяє вам тестувати функціональність навіть у випадку, коли деякі залежності ще не реалізовані або недоступні.

*Покриття коду*. PHPUnit підтримує вимірювання покриття коду тестами. Ви можете перевірити, яка частина вашого коду була протестована, а яка залишилася непротестованою. Це допомагає забезпечити більшу надійність та якість вашого додатку.

Тестування у Laravel з використанням PHPUnit допомагає забезпечити правильну роботу вашого додатку, виявляти та усувати помилки, покращувати стабільність та зручність його використання. Воно є важливою складовою розробки на основі фреймворку Laravel та сприяє підтримці вашого додатку у хорошому стані протягом усього життєвого циклу.

Отже, були розроблені автоматизовані тести, які дозволяють проводити так зване smoke тестування. Основна ідея smoke тестування полягає в тому, щоб виконати швидку та легку перевірку основних функціональних можливостей програми, щоб переконатися, що система може працювати інтегровано та має базовий рівень функціональності. Таким чином, для кожного маршруту було розроблено по дві функції, які надсилають запити. Одна функція працює від особи, яка має необхідні повноваження, а інша – від особи, яка їх не має.

Розглянемо цей процес на прикладі тестування створення філії. Загалом, функція складається з трьох блоків:

– підготовка даних;

– виконання дії;

– перевірка результату дії.

Спільним для усіх функцій тестування, пов’язаних з роботою з філіями – є створення користувача, від особи якого буде виконуватись HTTP запит. Код, який створить користувачів з певними повноваженнями та без них зображено на рисунку 3.25.

```
    public function setUp(): void {
        parent::setUp();
        $this->user_with_permissions = User::factory()->create();
        $this->user_with_permissions->givePermissionTo('Branch:access',
'Branch:create', 'Branch:view', 'Branch:edit', 'Branch:delete');
        $this->user_without_permissions = User::factory()->create();
    }
```

Рисунок 3.25 – Створення користувача з повноваженнями та без

Для безпосереднього створення філії необхідно відправити POST запит за маршрутом /api/branches, передавши при цьому параметр name у якості назви філії (рис. 3.26).

```
        $response = $this->actingAs($this->user_with_permissions)-
>postJson('/api/branches', [
            'name' => 'Нова філія',
        ]);
```

Рисунок 3.26 – Виконання POST запиту

Нехай тест буде вважатись успішно пройденим, якщо у якості відповіді була отримана відповідь зі статусом 201 Created, якщо запит виконувався від особи користувача з повноваженнями, і 403 Forbidden – для користувача без необхідних повноважень.

Приклади перевірки цих результатів зображено на рисунку 3.27.

```
        $response->assertCreated();

        $response->assertForbidden();
```

Рисунок 3.27 – Перевірка статусів запиту

Тепер необхідно виконати команду “php artisan test”, яка запустить усі написані тести і відобразить результат (рис. 3.28).

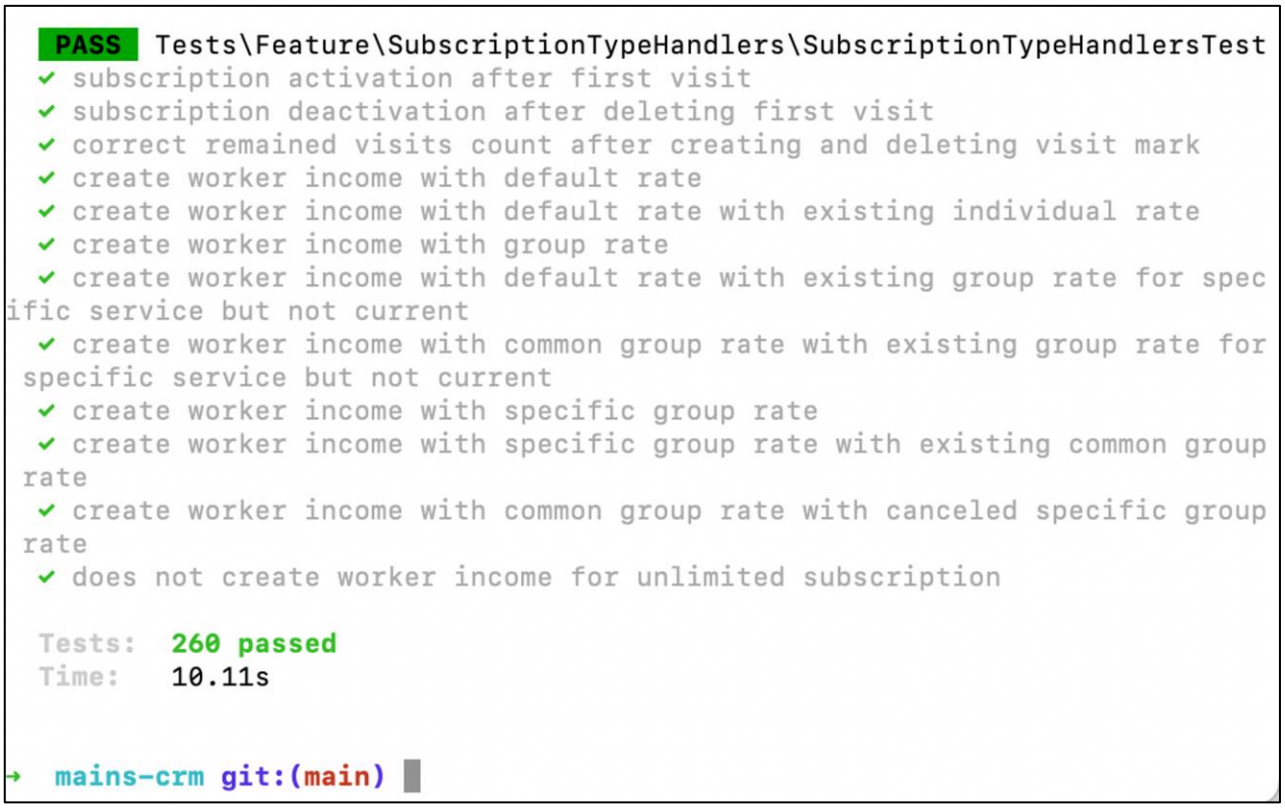


Рисунок 3.28 – Результат виконання тестів

**Висновки до розділу 3**

Таким чином, було розроблено реалізацію схеми сутностей-відношень з використанням фреймворку Laravel, описано концепцію CRUD операцій, розглянуто як можна створити обробку HTTP запитів для реалізації CRUD операцій відповідно до стандарту REST, детально розглянуто процес розробки функціоналу процесу продажу абонементів, описано механізм транзакцій бази даних, який дозволяє забезпечити надійну роботу та цілісність даних. Також були розроблені та виконані автоматизовані тести, які перевіряють працездатність всієї системи.

# ВИСНОВКИ

В роботі наведено короткий огляд сучасних засобів веброзробки, зокрема мова програмування PHP та її фреймворк Laravel, розглянуто корисні бібліотеки цього фреймворку, які дозволяють підтримувати код у єдиному стилі, автоматично генерувати документацію, робити резервні копії бази даних та дозволяють легко працювати з механізмом ролей та повноважень. Описано предметну область системи фітнес-центрів, включаючи всі сутності, їх атрибути та зв’язки між цими сутностями. Розроблено ER-діаграму та її програмну реалізацію з використанням фреймворку Laravel та системи управління базами даних MySQL. Також були розроблені тести для виконання автоматизованого тестування всієї системи, і було розглянуто процес розробки на прикладі функціоналу CRUD операцій з філіями та процесом продажу абонементів.

За допомогою моделей у Laravel можна досить легко підвищити ефективність запитів до бази даних за допомогою Eloquent ORM. Eloquent ORM надає можливість робити запити до бази даних без використання SQL, тому немає потреби витрачати багато часу на написання складного та оптимізованого коду в SQL, що значно пришвидшує розробку програмного забезпечення.

Отже, через можливість гнучкого налаштування моделей, через можливість використання функціоналу міграцій бази даних, через присутність вбудованого функціоналу роботи з HTTP запитами та відповідями, зокрема досить легка валідація даних, фреймворк Laravel ідеально підійшов для розробки REST API системи фітнес-цетрів.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Chacon S., Straub B. Pro Git (Second edition). Apress, 2023. 500 p.

2. Ball C. J. Hacking APIs: Breaking Web Application Programming Interfaces. No Starch Press, 2022. 368 p.

3. Taylor A. G. SQL All-In-One for Dummies. For Dummies, 2019. 768 p.

4. Документація по мові програмування PHP. URL: https://www.php.net/manual/en/index.php (дата звернення: 24.02.2023).

5. Офіційна документація Laravel. URL: https://laravel.com/docs (дата звернення: 25.02.2023).

6. Документація бібліотеки laravel/pint. URL: https://github.com/laravel/pint (дата звернення: 02.03.2023).

7. Документація бібліотеки laravel-permission. URL: https://spatie.be/docs/laravel-permission/v5/introduction (дата звернення: 05.03.2023).

8. Документація бібліотеки laravel-backup. URL: https://spatie.be/docs/laravel-backup/v8/introduction (дата звернення: 05.03.2023).

9. Документація бібліотеки Scribe. URL: https://scribe.knuckles.wtf/laravel (дата звернення: 10.03.2023).

10. PHPUnit Manual. URL: https://docs.phpunit.de/en/10.1 (дата звернення: 04.04.2023).

11. Що таке RESTful API? – Amazon Web Services. URL: https://aws.amazon.com/what-is/restful-api (дата звернення: 15.03.2023).

12. Що таке Postman? – Postman. URL. https://www.postman.com/product/what-is-postman (дата звернення: 15.04.2023).

# ДОДАТОК А

## Програмна реалізація REST API фітнес-центрів

Уся взаємодія клієнтської програми з серверною частиною відбувається наступним чином: клієнтська програма робить HTTP запит до API, система обробляє цей запит та повертає результат обробки клієнтській програмі.

У файлі routes/api.php (рис. А.1) знаходяться маршрути, які співвідносять HTTP запити з функціями контролерів, які будуть їх обробляти.

```
Route::middleware('auth:sanctum')->group(function() {
        Route::apiResource('rooms', RoomController::class);
        Route::get('payments/current_balance', [PaymentController::class,
'getCurrentBalance']);
        Route::apiResource('payments', PaymentController::class);
        // …
});
```

Рисунок А.1 – Маршрути у routes/api.php

Після файлу routes/api.php запит потрапляє безпосередньо до функції певного контролеру. На рисунку А.2 зображено функцію, яка обробляє запит на додавання клієнта до групи, щоб той мав змогу відвідувати групові заняття.

```
public function addCustomerToGroup(AddCustomerRequest $request) {
    $group_customer = GroupCustomer::create($request->validated());
    return response()->noContent();
}
```

Рисунок А.2 – Функція контролеру додавання клієнта до групи

У цій функції додається певний клієнт до певної групи, але перед цим

виконується перевірка даних на коректність. Цей процес виконується за допомогою класу AddCustomerRequest. Всередині цього класу перераховані всі правила валідації даних (рис. А.3), які були передані разом із запитом.

```
    public function rules() {
        return [
            'group_id' => ['required', 'integer'],
            'customer_id' => ['required', 'integer'],
            'subscription_id' => ['required', 'integer',
                function (string $attribute, mixed $value, Closure $fail) {
                    $subscription = Subscription::with(['subscription_type', 'services', 'workers'])->find($value);
                    $subscription_handler = new $subscription->subscription_type->internal_class($subscription);
                    $group = Group::with('customers')->find($this->group_id);
                    if(! $subscription_handler->customerCanBeAddedToGroup($group))
                        $fail('Клієнт не може бути доданий до групи.');
                },
            ],
        ];
    }
```

Рисунок А.3 – Правила валідації додавання клієнта в групу

Виходячи з цього, для того, щоб додати клієнта до групи, потрібно передати три параметри: ідентифікатор групи, ідентифікатор клієнта та ідентифікатор абонементу. Усі параметри є обов’язковими, при чому ще в залежності від типу абонементу буде виконуватись певна перевірка чи може клієнт бути доданий до цієї групи за даним абонементом, чи ні. Якщо всі параметри успішно пройшли перевірку, клієнт додається до групи, та

повертається результат у вигляді HTTP відповіді зі статусом 204 No content.

Таким чином, завдяки використанню Laravel можна розробити повноцінне REST API, яке забезпечує взаємодію з будь-якою клієнтською програмою. Шляхом прописування маршрутів у файлі routes/api.php та реалізації функцій у контролерах, можна забезпечити доступ до різноманітних функціональних можливостей системи фітнес-центрів через API. Це дозволяє використовувати систему зручним способом, будь то мобільний додаток, веб-сайт на Vue.js або навіть через бота у месенджері. Такий підхід робить систему гнучкою та доступною для різних типів клієнтів, що сприяє її широкому застосуванню та покращує користувацький досвід.