МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проектування та розробка інформаційної системи для керування спорт-студією»**

Виконала: студентка 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Кирилюк Оксана Іванівна

Керівник: *доктор філософії з прикладної математики, старший викладач кафедри інформаційних технологій та аналітики даних*
Красюк Богдан Віталійович

Рецензент: *Розробник програмного забезпечення в ТОВ ОБНОВА-ЄВРОШОП, викладач кафедри інформаційних технологій та аналітики даних НаУОА*
Шведюк Володимир Володимирович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: **Проектування та розробка інформаційної системи для керування спорт-студією**

Автор: Кирилюк Оксана Іванівна

Науковий керівник: ***доктор філософії з прикладної математики, викладач, фахівець-практик, Красюк Богдан Віталійович***

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: ***66*** *с., 3 рис., 6 табл., 0 додатків, 39 джерел.*

Ключові слова: ***і****нформаційна система, спортивна студія, серверна частина, клієнтська частина, база даних, GraphQL, авторизація, безпека, JWT, маршрутизація, SPA*

Короткий зміст праці:
У роботі розглянуто розробку інформаційної системи для управління спортивною студією, що складається з серверної та клієнтської частин. Серверна частина базується на грамотно спроєктованій базі даних та GraphQL API, що забезпечує ефективний обмін даними і гнучку взаємодію з клієнтом. Реалізовано механізми авторизації з використанням JWT і розмежуванням доступу за ролями (адміністратор, тренер, клієнт). Клієнтська частина побудована на React із застосуванням Chakra UI для швидкого створення адаптивного інтерфейсу, Redux Toolkit для управління станом і React Router для маршрутизації. У роботі також описано організацію бізнес-логіки, тестування і налаштування інфраструктурного рівня. Запропонована архітектура забезпечує безпеку, масштабованість і зручність використання системи.

_____

ANNOTATION
of the qualification work
for obtaining the educational degree of Bachelor

Topic: Design and Development of an Information System for Managing a Sports Studio

Author: Oksana Ivanivna Kyrylyuk

Academic Advisor: Doctor of Philosophy in Applied Mathematics, Lecturer, Practitioner Specialist, Bohdan Vitaliyovych Krasiuk

Defended on «……»…… 20__.

Explanatory note to the qualification work: *66 pages, 3 figures, 6 tables, 0 appendices, 39 sources.*

Keywords: *information system, sports studio, server side, client side, database, GraphQL, authorization, security, JWT, routing, SPA*

Brief summary of the work:
The work considers the development of an information system for managing a sports studio, consisting of server and client parts. The server side is based on a well-designed database and a GraphQL API, providing efficient data exchange and flexible client interaction. Authorization mechanisms are implemented using JWT with role-based access control (administrator, coach, client). The client side is built with React using Chakra UI for rapid creation of a responsive interface, Redux Toolkit for state management, and React Router for routing. The work also describes the organization of business logic, testing, and infrastructure-level setup. The proposed architecture ensures system security, scalability, and ease of use.

_____

ЗМІСТ

ВСТУП **6**
РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ **8**
1.1. Опис предметного середовища (функціональної моделі, процесу діяльності) 8
1.2. Огляд наявних аналогів 12
1.2.1 Універсальні CRM-системи 13
1.2.2 Спеціалізовані платформи для фітнес-центрів 13
1.3. Постановка задачі 15
1.3.1 Визначення проблем, які має вирішити розроблювана інформаційна система 15
1.3.2. Основні вимоги до інформаційної системи 16
ВИСНОВКИ ДО РОЗДІЛУ 1 18
РОЗДІЛ 2. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ **19**
2.1 База даних 19
2.1.1 Опис структури бази даних та її таблиць 20
2.1.2 Призначення та обґрунтування використання зв’язків 25
2.2 Розробка програмного інтерфейсу прикладного рівня та механізмів авторизації 27
2.2.1 Концепція авторизації та безпеки 27
2.2.2 Реалізація механізмів Guard та налаштування кастомного Guard для ролей 29
2.4 Впровадження інфраструктурного рівня та бізнес логіки 33
2.4.1 Базове налаштування проекту 34
2.4.2 Налаштування модуля тестування 42
2.4.3 Реалізація базових компонентів для перевикористання 45
ВИСНОВКИ ДО РОЗДІЛУ 2 48
РОЗДІЛ 3. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ **49**
3.1 Загальні принципи побудови клієнтської частини 49
3.2 Модуль авторизації 49
3.3 Ролі та доступи 52
3.2.1 Адміністратор: 53
1. Управління користувачами 53
2. Управління розкладом тренувань 54
3. Управління абонементами 54
4. Обробка заявок на купівлю абонементів 55
5. Перегляд аналітики 55
3.2.2 Тренер: Реалізація з точки зору фронтенду 56

1. Перегляд розкладу тренувань 56
2. Відмітка присутності 56
3. Перегляд історії тренувань 57
3.2.3 Клієнт: Реалізація з точки зору фронтенду 57
1. Перегляд доступних тренувань 57
2. Запис на тренування 58
3. Перегляд особистого кабінету 58
4. Управління абонементами 59
5. Повідомлення та сповіщення 59
6. Аналіз активності 60
ВИСНОВКИ ДО РОЗДІЛУ 3 62
ВИСНОВКИ **63**
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **66**

ВСТУП

Сучасний розвиток інформаційних технологій значно впливає на різні сфери діяльності, зокрема на організацію та управління спортивними студіями. Автоматизація бізнес-процесів у спортивних закладах дозволяє підвищити ефективність роботи, зменшити витрати часу на адміністративні завдання та покращити взаємодію з клієнтами.

Зростання популярності здорового способу життя призводить до збільшення кількості спортивних студій, що, у свою чергу, вимагає впровадження сучасних технологічних рішень для ефективного керування. Традиційні методи ведення обліку та організації роботи, такі як використання паперових журналів або електронних таблиць, стають малоефективними через збільшення потоку клієнтів та обсягу даних. Тому необхідність створення автоматизованої інформаційної системи для керування спорт-студією є надзвичайно актуальною.

Мета дослідження – проектування та розробка інформаційної системи для керування спорт-студією, що забезпечить автоматизацію основних бізнес-процесів, зокрема обліку клієнтів, розкладу занять, фінансових операцій та управління персоналом.

Завдання дослідження:

- аналіз предметної області, визначення ключових вимог до інформаційної системи;
- проектування структури та функціональних можливостей системи;
- вибір відповідного програмного забезпечення, технологій та інструментів реалізації;
- розробка та тестування програмного продукту;
- впровадження інформаційної системи та оцінка її ефективності в реальних умовах експлуатації.

Об’єкт дослідження – інформаційна система для керування спорт-студією.

Предмет дослідження – методи та інструментальні засоби, що використовуються для проектування, розробки та тестування інформаційної системи.

Для досягнення поставленої мети використовувалися сучасні методи програмної інженерії, зокрема об’єктно-орієнтований підхід до проектування, реляційні бази даних для збереження інформації, а також методи тестування для забезпечення стабільної та ефективної роботи системи.

Розробка інформаційної системи передбачає використання гнучкої архітектури, що дозволяє адаптувати її до змін у роботі спорт-студії та масштабувати в залежності від потреб. Планується впровадження веб-інтерфейсу для адміністраторів і мобільного додатку для клієнтів, що дасть змогу автоматизувати запис на заняття, відстежувати відвідуваність та здійснювати оплату онлайн.

Запропонована інформаційна система дозволить оптимізувати процеси керування спорт-студією, підвищити якість обслуговування клієнтів і зменшити навантаження на адміністративний персонал. Крім того, система сприятиме підвищенню конкурентоспроможності спорт-студії за рахунок ефективного використання ресурсів та вдосконалення взаємодії з клієнтами. Впровадження автоматизованого підходу до керування забезпечить підвищення рівня сервісу, що стане важливим фактором для залучення нових клієнтів та утримання постійних відвідувачів.

Таким чином, розробка та впровадження інформаційної системи для керування спорт-студією є актуальним та важливим завданням, яке сприятиме покращенню якості обслуговування, ефективності управління та розвитку бізнесу у сфері спортивних послуг.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища (функціональної моделі, процесу діяльності)

Управління спортивними студіями потребує сучасних технологічних рішень для автоматизації бізнес-процесів, підвищення ефективності роботи та покращення взаємодії з клієнтами. У даному розділі розглядається предметне середовище, функціональна модель інформаційної системи, а також процеси діяльності, які мають бути автоматизовані для забезпечення ефективності управління спорт-студією.

Функціональне середовище спортивної студії

Основні компоненти діяльності

Спортивна студія здійснює комплекс заходів, спрямованих на надання послуг з фітнесу та тренувань. Основні напрямки діяльності включають:

- Адміністрування клієнтів – реєстрація нових користувачів, ведення бази даних клієнтів, управління їхніми абонементами та статусами.
- Організація тренувального процесу – складання розкладу занять, бронювання місць на тренування, розподіл тренерів.
- Фінансовий облік – управління оплатами, оформлення рахунків, контроль фінансових операцій.
- Взаємодія з клієнтами – автоматизоване сповіщення про заняття, персоналізовані рекомендації, інтеграція з месенджерами та мобільними додатками.

Учасники предметного середовища

До ключових учасників спортивної студії належать:

- **Адміністратори** – відповідальні за управління студією, контроль фінансів

та загальну організацію роботи.

- **Тренери** – проводять заняття, контролюють фізичну активність клієнтів, ведуть індивідуальні консультації.
- **Клієнти** – користуються послугами студії, записуються на тренування, здійснюють оплату та отримують консультації.

Функціональна модель інформаційної системи

Архітектурний підхід

Для управління спортивною студією розробляється **клієнт-серверна** інформаційна система, яка складається з:

1. Клієнтської частини – веб-додаток, через який користувачі можуть реєструватися, записуватися на заняття, здійснювати оплату.
2. Серверної частини – містить базу даних, обробку запитів, управління ролями користувачів.
3. Адміністративного інтерфейсу – панель для адміністраторів та тренерів, де ведеться управління клієнтами та фінансами.

Основні функції системи

Інформаційна система повинна забезпечити:

- Автоматизоване управління записами на тренування – онлайн-бронювання місць, коригування розкладу відповідно до потреб студії.
- Систему розрахунку вартості послуг – інтеграція платіжних сервісів, контроль оплати та фінансових операцій.
- Контроль відвідуваності – ведення статистики для адміністраторів і тренерів.
- Інтерактивний інтерфейс для клієнтів – персональний кабінет з можливістю перегляду історії тренувань.

- Система повідомлень – автоматизовані нагадування про заняття, консультації через чат-боти.

Технологічний стек

Для реалізації системи застосовуються такі технології:

- React.js – розробка інтуїтивного клієнтського інтерфейсу.
- Node.js та GraphQL – серверна логіка та інтеграція з базою даних.
- JWT-авторизація – безпека входу користувачів.

Опис процесу діяльності

Автоматизація основних процесів

Інформаційна система покликана забезпечити автоматизацію таких операцій:

1. Реєстрація та авторизація користувачів – клієнти та тренери можуть створювати акаунти, входити в систему та отримувати персоналізований доступ.
2. Запис на тренування та управління розкладом – клієнти можуть резервувати місця на заняття, тренери управляють розкладом та коригують тренувальні програми.
3. Фінансовий контроль – автоматизоване виставлення рахунків, моніторинг оплат та інтеграція з онлайн-банкінгом.
4. Контроль ефективності занять – система аналізує відвідуваність клієнтів, формує звіти для тренерів та адміністраторів.
5. Зворотний зв’язок із клієнтами – інтеграція з чат-ботами та месенджерами, автоматизовані повідомлення про зміни у розкладі та нагадування.

Діаграма варіантів використання (Use Case Diagram)

Діаграма варіантів використання демонструє взаємодію між користувачами

системи (акторами) та функціональними можливостями (варіантами використання). На діаграмі представлені:

Актори системи:

- Клієнт - особа, яка користується послугами спортивної студії.
- Тренер - фахівець, який проводить заняття та консультації.
- Адміністратор - відповідальний за управління системою та бізнес-процесами.

Варіанти використання для клієнта:

- Реєстрація/авторизація - процес створення облікового запису та входу в систему.
- Запис на тренування - можливість бронювати місце на заняття.
- Оплата послуг - здійснення платежів за абонементи та тренування.

Варіанти використання для тренера:

- Управління розкладом - можливість створювати та редагувати розклад занять.
- Контроль відвідуваності - ведення обліку присутності клієнтів на тренуваннях.

Варіанти використання для адміністратора:

- Управління клієнтами - робота з базою клієнтів, редагування профілів.
- Фінансовий моніторинг - відстеження платежів та управління фінансами.
- Аналітика і звітність - формування статистичних даних та звітів.

Ця діаграма дозволяє розробникам та стейкхолдерам зрозуміти основні функціональні вимоги системи та ролі користувачів.

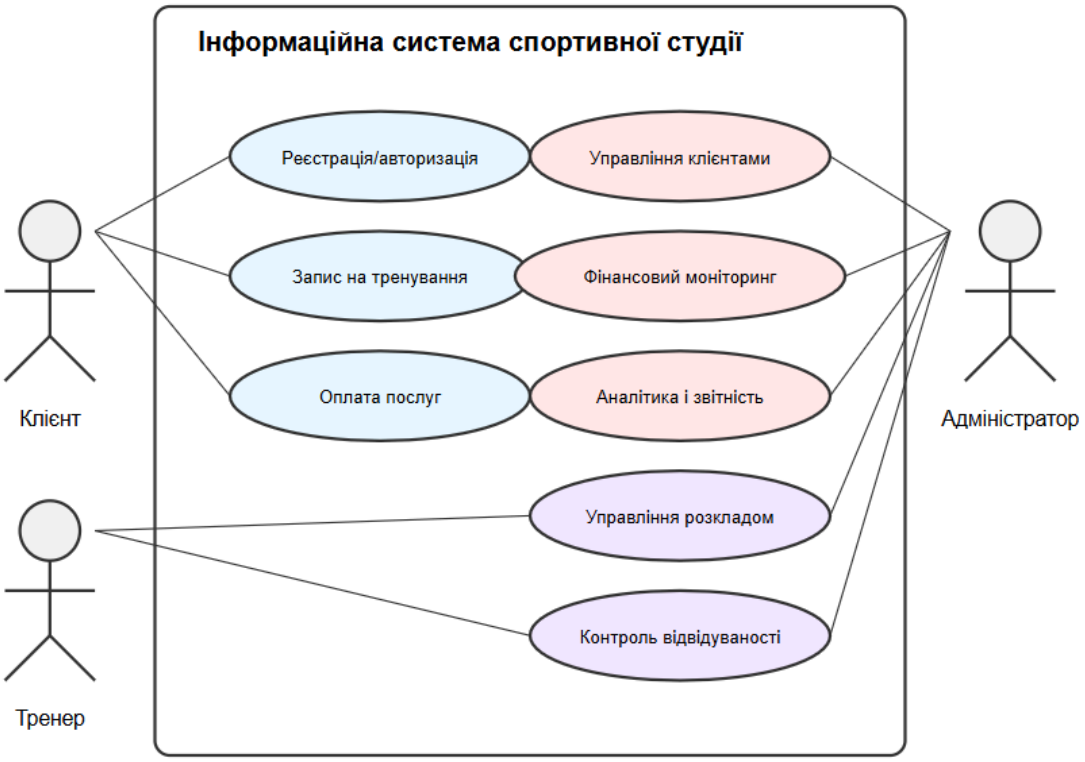

Рис. 1.2.1. Use Case діаграма для інформаційної системи спортивної студії

Джерело: розроблено автором

1.2. Огляд наявних аналогів

Ринок програмного забезпечення для управління спортивними студіями активно розвивається, пропонуючи широкий спектр рішень для автоматизації бізнес-процесів. Основні аналоги можна розділити на три категорії:

- ❖ універсальні CRM-системи,
- ❖ спеціалізовані платформи для фітнес-центрів та індивідуальні розробки.

Розгляд кожної з цих категорій дозволяє визначити їх переваги та недоліки, а також зрозуміти, які функціональні та технологічні рішення слід врахувати під час розробки нової інформаційної системи.

1.2.1 Універсальні CRM-системи

CRM-системи (Customer Relationship Management) широко використовуються у різних сферах бізнесу, включаючи спортивні студії. Вони дозволяють керувати клієнтською базою, вести аналітику відвідувань та автоматизувати комунікацію з клієнтами. Деякі популярні CRM-рішення, які застосовуються у фітнес-індустрії:

- **HubSpot CRM** – загальна система управління клієнтами, що може бути адаптована для фітнес-студій. Вона включає автоматизацію маркетингових кампаній, ведення бази даних клієнтів та відстеження їхньої активності.
- **Zoho CRM** – потужне рішення, яке підтримує інтеграцію з фінансовими сервісами та аналітичними інструментами. Проте, його застосування у сфері фітнесу вимагає додаткової адаптації.
- **Salesforce** – одна з найпопулярніших CRM-систем, що пропонує розширену можливість налаштувань для автоматизації бізнес-процесів.

Переваги універсальних CRM-систем:

- ➢ Широкий функціонал для управління клієнтами
- ➢ Інтеграція з фінансовими та маркетинговими інструментами.
- ➢ Гнучкі налаштування та можливість адаптації.

Недоліки:

- ➢ Відсутність специфічних функцій для управління розкладом тренувань.
- ➢ Складність налаштування для спортивних студій без розробницьких знань.

1.2.2 Спеціалізовані платформи для фітнес-центрів

Окрім CRM-рішень, існують програми, спеціально розроблені для управління фітнес-студіями. Вони мають необхідні функції для адміністрування розкладу,

обліку тренувань та контролю фінансових операцій. Серед таких платформ можна виділити:

- **Mindbody** – одна з найпоширеніших систем для фітнес-центрів. Вона включає інструменти для управління тренуваннями, автоматизації записів та фінансового обліку.
- **Zen Planner** – рішення, орієнтоване на невеликі студії та спортивні клуби. Підтримує функції для контролю відвідуваності, планування тренувань та ведення фінансових звітів.
- **Glofox** – сучасна система для фітнес-студій, яка включає інтуїтивний мобільний додаток для клієнтів, інструменти аналітики та CRM-функції.

Переваги спеціалізованих платформ:

- Наявність вбудованого календаря для тренувань.
- Автоматизація записів та фінансових операцій.
- Спеціалізовані функції для тренерів та адміністраторів.

Недоліки:

- Висока вартість підписки.
- Деякі системи мають обмежені можливості кастомізації.
- Залежність від стороннього програмного забезпечення.

3. Індивідуальні розробки

Деякі спорт-студії створюють власні інформаційні системи, що точно відповідають їхнім бізнес-потребам. Такі рішення можуть бути розроблені із використанням сучасних технологій, таких як React.js, Node.js, GraphQL та JWT для авторизації.

Переваги індивідуальних розробок:

- Повне налаштування під потреби конкретної студії.

- ☐ Можливість інтеграції з внутрішніми корпоративними системами.
- ☐ Відсутність залежності від сторонніх платформ.

Недоліки:

- ☐ Висока початкова вартість розробки.
- ☐ Потреба у підтримці та оновленнях з боку програмістів.

1.3. Постановка задачі

1.3.1 Визначення проблем, які має вирішити розроблювана інформаційна система

Сфера спортивних студій активно розвивається, і разом із цим зростає потреба у впровадженні інформаційних систем для автоматизації процесів управління такими закладами. Основні проблеми, які має вирішити розроблювана система, можна розділити на кілька категорій:

1. Автоматизація адміністративних процесів

Традиційне ручне ведення записів про клієнтів, тренерів, розклад тренувань та фінансові операції є трудомістким та неефективним. Відсутність централізованого підходу до управління спорт-студіями створює ризик помилок, втрати даних та складнощів у контролі фінансів. Розроблювана система повинна забезпечити:

- Централізоване управління базою даних клієнтів.
- Автоматизовану реєстрацію та облік відвідувань.
- Оптимізацію розкладу занять та резервування місць.

2. Покращення взаємодії між студією та клієнтами

У сучасних умовах клієнти очікують високого рівня обслуговування та зручного доступу до послуг студії. Відсутність онлайн-системи реєстрації на тренування, перегляду розкладу та отримання персоналізованих рекомендацій створює

додаткові труднощі для користувачів. Інформаційна система повинна забезпечити:

- Персональний кабінет для користувачів із можливістю перегляду історії занять.
- Автоматичні нагадування про тренування та можливість планування розкладу.
- Інтерактивний інтерфейс для вибору тренерів та програм занять.

3. Контроль ефективності тренувань

Відстеження прогресу клієнтів є важливим аспектом для забезпечення ефективності тренувального процесу. Відсутність інструментів для аналізу статистики занять знижує можливість оптимізації програм тренувань та індивідуального підходу до кожного клієнта. Система повинна містити:

- Механізми аналізу успішності клієнтів та їх активності.
- Інтерактивні звіти для тренерів та адміністраторів.
- Рекомендації на основі даних про відвідуваність.

4. Забезпечення безпеки та конфіденційності даних

Обробка персональних даних клієнтів та фінансових транзакцій потребує високого рівня захисту. Недостатня безпека інформації може призвести до витоку даних або шахрайських дій. Для цього система повинна:

- Використовувати захищені методи авторизації та аутентифікації.
- Мати механізми резервного копіювання даних та їх шифрування.
- Забезпечити контроль доступу на рівні ролей користувачів.

1.3.2. Основні вимоги до інформаційної системи

Враховуючи описані проблеми, розроблювана система повинна відповідати наступним ключовим вимогам:

1. Архітектурні вимоги

- Використання клієнт-серверної моделі для забезпечення гнучкості та масштабованості.
- Впровадження API для інтеграції з веб- та мобільними додатками.
- Можливість обробки великого обсягу даних без втрати продуктивності.

2. Функціональні вимоги

- Автоматизоване управління клієнтською базою, розкладом занять та фінансами.
- Система авторизації з ролями: адміністратор, тренер, клієнт.
- Впровадження механізму онлайн-оплати та звітності.

3. Технологічні вимоги

- Використання сучасних технологій, таких як React.js для фронтенду та Node.js для серверної частини.
- Інтерактивна робота через GraphQL для ефективного обміну даними між клієнтом та сервером.
- Захищена авторизація на основі JWT для безпеки користувачів.

4. Юзабіліті та UX-вимоги

- Інтуїтивний дизайн для зручності взаємодії.
- Адаптивність для підтримки мобільних пристроїв.
- Підтримка мультимовності для розширення аудиторії користувачів.

Таким чином, розробка інформаційної системи для управління спортивною студією повинна враховувати всі ці аспекти, щоб забезпечити ефективну роботу закладу та підвищити рівень обслуговування клієнтів.

ВИСНОВКИ ДО РОЗДІЛУ 1

Розроблена інформаційна система для управління спортивною студією має забезпечити ефективну автоматизацію бізнес-процесів, покращити комунікацію між учасниками студії та оптимізувати взаємодію клієнтів із тренерами. Завдяки сучасним технологіям, система дозволить не лише знизити витрати на адміністрування, а й підвищити якість обслуговування та конкурентоспроможність студії у цифровому середовищі.

Аналіз наявних аналогів показує, що існує широкий спектр програмних рішень для спортивних студій, проте кожне з них має свої обмеження. **C**RM-системи, хоч і потужні, потребують адаптації. Спеціалізовані платформи пропонують зручні функції, але можуть бути дорогими. Індивідуальні розробки надають максимальну гнучкість, але потребують значних ресурсів для їх впровадження.

При розробці нової інформаційної системи варто враховувати переваги кожної категорії, щоб створити продукт, який буде гнучким, функціональним та доступним для спортивних студій будь-якого масштабу.

З огляду на всі розглянуті аналоги, розробка власної інформаційної системи з урахуванням досвіду CRM-систем та спеціалізованих фітнес-платформ виглядає найбільш оптимальним рішенням. Такий підхід дозволить інтегрувати ключові переваги готових рішень, одночасно забезпечивши гнучкість та ефективність роботи.

У майбутньому можна очікувати розширення функціоналу інформаційних систем для спортивних студій, включаючи інтеграцію штучного інтелекту для персоналізованих тренувань, прогнозування поведінки клієнтів та вдосконалення систем безпеки. Успішна реалізація інформаційної системи допоможе покращити бізнес-процеси спортивної студії, підвищити якість обслуговування клієнтів та забезпечити її конкурентоспроможність у сучасному цифровому середовищі.

РОЗДІЛ 2. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

2.1 База даних

База даних є основним компонентом інформаційної системи управління спортивною студією, і її структура була спроектована з урахуванням принципів нормалізації, масштабованості, цілісності даних та відповідності потребам бізнес-логіки. Схема даних побудована таким чином, щоб забезпечити ефективне управління користувачами, тренуваннями, абонементами, розкладом, бронюваннями та іншими ключовими сутностями, що складають основу функціонування системи. Використання цих принципів дозволяє уникнути надмірного дублювання даних, сприяє збереженню цілісності інформації та дозволяє легко додавати нові типи послуг. Це також забезпечує гнучкість для подальшого розвитку системи, зокрема для підтримки групових занять або інтеграції з мобільними застосунками в майбутньому.

Для реалізації цієї бази даних було обрано PostgreSQL — одну з найпопулярніших реляційних баз даних, яка має численні переваги для такого типу інформаційних систем.

Переваги PostgreSQL:

1. PostgreSQL гарантує консистентність даних завдяки потужним механізмам забезпечення цілісності та транзакцій, що критично важливо для великої кількості користувачів і записів.
2. Система може масштабуватися як вертикально, так і горизонтально, що дозволяє ефективно адаптуватися до зростання обсягів даних.
3. PostgreSQL підтримує складні запити, підзапити та агрегацію, що забезпечує гнучкість в обробці і відображенні даних.
4. База даних пропонує високий рівень безпеки через шифрування та багаторівневий доступ, що важливо для захисту чутливих даних.

5. PostgreSQL підтримує додаткові функції та розширення, що дозволяє адаптувати систему до майбутніх потреб і розширень.

2.1.1 Опис структури бази даних та її таблиць

Основні таблиці:

User (Користувачі)

Таблиця виконує роль основного елементу ідентифікації та автентифікації в системі, забезпечує персоналізований доступ до функціоналу залежно від ролі користувача. Дозволяє реалізувати розмежування прав і обов'язків між клієнтами, тренерами та адміністраторами. Сприяє побудові взаємодії між користувачами й іншими підсистемами, зокрема бронюванням тренувань, формуванням замовлень, керуванням розкладом і тренерським складом. Є центральним елементом для моніторингу активності користувачів, аналізу їх участі у тренуваннях та історії взаємодії зі студією.

Таблиця 2.1.1.1. Користувачі

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор користувача.
email	varchar (UNIQUE)	Адреса електронної пошти, яка використовується як логін.
first_name	varchar	Ім'я користувача.
last_name	varchar	Прізвище користувача.
phone	varchar (UNIQUE)	Номер телефону для комунікації.

roles	text[]	Масив ролей (клієнт, тренер, адміністратор тощо).

Особливості**:**

- використовується тип text[] для зберігання кількох ролей;
- поля email та phone індексуються для забезпечення унікальності;
- реалізовано soft-delete (позначення записів як неактивних замість фізичного видалення).

Abonaments (Абонементи)

Ця таблиця керує інформацією про доступні абонементи, які пропонуються клієнтам студії. Вона дозволяє реалізувати модель гнучкої оплати за тренування, де кожен абонемент визначає умови доступу до послуг студії, зокрема кількість тренувань, тривалість дії та вартість. Абонементи є основним інструментом для монетизації послуг, а також для планування фінансових потоків. Вони взаємодіють із підписками користувачів, дозволяючи контролювати їх доступ до тренувань і забезпечувати правильний розподіл ресурсів. Використовується для проведення аналітики популярності абонементів, визначення доходів та планування маркетингових кампаній.

Таблиця 2.1.1.2. Абонементи

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор абонемента.
title	varchar	Назва абонемента.
description	text	Опис переваг і обмежень абонемента.
count_trainings	integer	Кількість тренувань, включених в абонемент.

valid_period	integer	Термін дії в днях.
price	integer	Вартість абонемента.
currency	integer	Код валюти (наприклад, 980 для UAH, 840 для USD).

MembershipPlans (Плани підписки)

Ця таблиця відображає підписки, придбані користувачами на абонементи студії. Вона забезпечує відстеження активних підписок, визначаючи, який абонемент був придбаний, коли він почав діяти і скільки тренувань залишилось. Плани підписки виконують роль зв’язку між користувачами і абонементами, дозволяючи управляти доступом до тренувань, моніторити використання ресурсів і автоматично визначати статус підписки (активна або завершена). Вони також підтримують бізнес-логіку, яка гарантує, що користувач може мати лише обмежену кількість одночасних підписок. Через таблицю забезпечується можливість для аналізу та оптимізації продажів абонементів.

Таблиця 2.1.1.3. Плани підписки

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор.
owner_id	varchar (FK → User.id)	Ідентифікатор користувача, що володіє підпискою.
plan_id	varchar (FK → Abonaments.id)	Посилання на тип абонемента.
start_day	date	Дата початку дії підписки.
trainings_consumed	integer	Кількість використаних тренувань.

status	varchar	Статус: "активний", "завершений", "призупинений".

Training (Тренування)

Ця таблиця описує різні види тренувань, доступні в спортивній студії. Вона дозволяє системі класифікувати та організовувати заняття за типами (групові, індивідуальні) та забезпечує зберігання важливої інформації про кожне тренування, такої як його назва, опис і тип. Тренування є основними одиницями для планування та бронювання, оскільки визначають, які саме види занять будуть доступні для користувачів. Базуючись на цих даних, система може управляти розкладом, забезпечувати правильний розподіл ресурсів (наприклад, тренерів або місць у групах) та надавати користувачам необхідну інформацію для вибору тренувань відповідно до їхніх потреб.

Таблиця 2.1.1.4. Тренування

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор тренування.
title	varchar	Назва заняття (наприклад, "Йога").
description	text	Детальний опис тренування.
type	varchar	Тип: "групове", "індивідуальне".

Schedule (Розклад)

Таблиця Schedule відповідає за зберігання даних про розклад тренувань у спортивній студії. Вона визначає, коли і де відбудеться кожне тренування, хто буде його проводити, а також скільки часу воно триватиме. Завдяки цій таблиці система може організовувати всі заплановані тренування, автоматично

перевіряти доступність тренерів та забезпечувати правильне розподілення часу для кожного заняття. Це дозволяє користувачам зручно обирати з доступних тренувань, а також здійснювати бронювання місць. Розклад є важливим інструментом для ефективного управління часом у студії та для того, щоб уникнути конфліктів у плануванні

Таблиця 2.1.1.5. Розклад

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор розкладу.
date	timestamp	Дата та час проведення.
trainer	varchar (FK → User.id)	Тренер, який проводить тренування.
training	varchar (FK → Training.id)	Тип тренування.
duration	integer	Тривалість (у хвилинах).

ScheduleBooking (Бронювання занять)

Таблиця ScheduleBooking займається фіксацією бронювань на тренування. Вона містить інформацію про те, який користувач забронював місце на певному тренуванні, а також який тип бронювання був обраний (групове чи індивідуальне). Це дає змогу не лише відслідковувати, хто записався на конкретне заняття, а й контролювати кількість доступних місць для кожного тренування. Крім того, система використовує дані цієї таблиці для перевірки можливості бронювання, управління місцями та забезпечення належного сервісу для клієнтів. Завдяки цьому забезпечується правильна організація тренувань і ефективне використання ресурсів спортивної студії.

Таблиця 2.1.1.6. Бронювання занять

Поле	**Тип**	**Опис**
id	varchar (PK)	Унікальний ідентифікатор бронювання.
schedule	varchar (FK → Schedule.id)	Посилання на розклад.
user	varchar (FK → User.id)	Хто здійснив бронювання.
type	varchar	Тип бронювання: "групове", "індивідуальне".

2.1.2 Призначення та обґрунтування використання зв’язків

User → MembershipPlans

Цей зв’язок забезпечує управління підписками кожного користувача. Оскільки кожен користувач може мати кілька активних підписок, таблиця дозволяє:

- Відстежувати, який абонемент було придбано, коли він почав діяти та скільки тренувань залишилося.
- Валідацію: при створенні нового плану система перевіряє, чи не перевищено кількість дозволених активних підписок для конкретного користувача.
- Бізнес-логіка: таблиця використовується для автоматичного визначення статусу підписки (наприклад, "активний" чи "завершений") залежно від кількості використаних тренувань або закінчення терміну дії.

Abonaments → MembershipPlans

Забезпечує зв'язок між типами абонементів та їх використанням. Це дозволяє:

- Валідацію: під час створення нового плану система перевіряє, чи існує вибраний абонемент, чи він активний, та чи підходить для користувача.
- Бізнес-логіка: таблиця дозволяє аналізувати популярність різних абонементів, обчислювати загальний дохід від продажів, а також проводити розрахунки для надання знижок чи бонусів.

User → Schedule

Цей зв’язок реалізує призначення тренерів до розкладу. Таблиця забезпечує:

- Валідацію: при додаванні тренера до розкладу перевіряється його роль (лише користувач із роллю "тренер" може проводити тренування).
- Бізнес-логіка: система може враховувати навантаження на тренера, обчислювати його зайнятість, автоматично пропонувати доступні дати для планування нових тренувань.

Training → Schedule

Цей зв’язок дозволяє планувати тренування за типами. Завдяки цьому:

- Валідація: перевіряється, чи зазначений тип тренування існує у системі.
- Бізнес-логіка: система може групувати розклад за типами тренувань (групове, індивідуальне) та пропонувати користувачам обрати відповідні заняття залежно від їхніх інтересів.

Schedule → ScheduleBooking

Цей зв’язок реалізує функціонал бронювання розкладу. Завдяки цьому:

- Валідація: перевіряється, чи є доступні місця для конкретного розкладу, та чи відповідає тип бронювання можливостям тренування (групове чи індивідуальне).

- Бізнес-логіка: автоматичне закриття можливості бронювання, коли всі місця зайняті, а також сповіщення користувачів про зміни у розкладі.

User → ScheduleBooking

Цей зв’язок дозволяє відстежувати бронювання, здійснені користувачем. Це дає змогу:

- Валідацію: кожен користувач може бронювати лише доступні розклади, враховуючи свої підписки.
- Бізнес-логіка: система може повідомляти користувача про майбутні тренування, формувати історію його участі у заняттях та обчислювати, які послуги залишаються невикористаними.

2.2 Розробка програмного інтерфейсу прикладного рівня та механізмів авторизації

2.2.1 Концепція авторизації та безпеки

Розробка прикладного програмного інтерфейсу (API) для інформаційної системи спортивної студії передбачає впровадження механізмів безпеки для захисту даних користувачів. Авторизація є одним із ключових компонентів цієї системи, оскільки вона визначає, які дії та ресурси доступні певному користувачу на основі його ролі.

Для забезпечення безпеки було обрано метод авторизації на основі токенів JWT (JSON Web Token). Цей підхід є сучасним стандартом, який широко використовується в системах із розподіленою архітектурою.

JSON Web Token: основи та переваги

JSON Web Token (JWT) — це компактний та автономний спосіб передачі інформації між сторонами як JSON-об'єкта, який безпечно підписується за допомогою алгоритму хешування. Основні переваги використання JWT включають:

1. Безпека:
   - JWT підписується криптографічним ключем, що унеможливлює підробку токена.

○ Підпис токена дозволяє перевірити його справжність без необхідності зберігати його на сервері.

2. Масштабованість:

○ Токен зберігає інформацію про користувача, тому сервер не потребує додаткової перевірки в базі даних для кожного запиту.

3. Гнучкість:

○ Токени можуть містити дані про роль користувача, час дії та інші параметри, необхідні для авторизації.

Механізм роботи JWT-авторизації

JWT складається з трьох частин, розділених крапками:

1. Header (заголовок):
   ○ Містить інформацію про алгоритм підпису та тип токена.
2. Payload (корисне навантаження):
   ○ Містить основні дані, такі як ID користувача, його роль та час дії токена.
3. Signature (підпис):
   ○ Забезпечує цілісність токена та підтверджує його справжність.

Етапи роботи JWT у системі

1. Аутентифікація користувача:
   ○ Користувач вводить свої облікові дані (логін і пароль).
   ○ Сервер перевіряє ці дані та створює JWT-токен для авторизованого користувача.
2. Видача токена:
   ○ Після успішної перевірки сервер відправляє клієнту JWT-токен.
   ○ Токен зберігається на стороні клієнта (наприклад, у локальному сховищі або cookies).
3. Авторизація запитів:
   ○ Кожен запит до API містить токен у заголовку Authorization.
   ○ Сервер перевіряє токен на справжність та термін дії.
   ○ Якщо токен валідний, сервер надає доступ до запитуваних ресурсів.
4. Завершення сесії:
   ○ Якщо токен втратив чинність (наприклад, через завершення терміну дії), користувачу необхідно повторно аутентифікуватися.

Особливості впровадження JWT у проєкті

1. Ролі користувачів: У системі реалізовано три типи користувачів: адміністратори, тренери та клієнти. JWT-токен містить інформацію про роль користувача, що дозволяє обмежити доступ до певних ресурсів.
2. Термін дії токена:
   - Для підвищення безпеки встановлено короткий термін дії токенів (наприклад, 1 година).
   - Для оновлення токена використовується механізм рефреш-токенів.
3. Захист конфіденційної інформації:
   - Токени підписуються секретним ключем, який зберігається на сервері.
   - Передача токенів здійснюється через захищений HTTPS-протокол.

Переваги реалізації JWT-авторизації

- Забезпечено високий рівень безпеки без необхідності зберігати сесії на сервері.
- Гнучкість у налаштуванні доступу до ресурсів для різних категорій користувачів.
- Простота інтеграції з клієнтськими додатками та іншими сервісами.

Ця система авторизації дозволяє ефективно керувати доступом користувачів до ресурсів та функціоналу інформаційної системи спортивної студії, забезпечуючи безпеку та зручність у використанні.

2.2.2 Реалізація механізмів Guard та налаштування кастомного Guard для ролей

Для забезпечення надійного контролю доступу до ресурсів API спортивної студії, використано механізм Guard у **NestJS**. Guards відіграють роль фільтрів, що перевіряють запити перед їхньою обробкою контролерами або резолверами. У цьому пункті реалізовано стандартний Guard для перевірки JWT-токенів та кастомний Guard для обмеження доступу на основі ролей користувачів.

Використані бібліотеки та їх призначення:

1. **@nestjs/jwt**

Використовується для роботи з JWT (JSON Web Tokens): створення, підписування та верифікації токенів.

2. **passport-jwt**

Модуль, що забезпечує інтеграцію з бібліотекою Passport для аутентифікації на основі JWT.

3. **crypto**

Генерація криптографічних ключів для підписування токенів.

4. **@nestjs/common**

Забезпечує доступ до основних декораторів (@Injectable, @CanActivate) та класів (UnauthorizedException, ForbiddenException).

5. **@nestjs/core**

Використовується для доступу до метаданих методів через **Reflector**, необхідного для кастомного Guard.

6. **passport**

Бібліотека для створення стратегій аутентифікації.

Реалізація Guard

Guard забезпечує перевірку наявності та валідності JWT-токена в заголовку запиту. Основні етапи роботи стандартного Guard:

1. Перехоплення запиту.
2. Витяг токена із заголовка Authorization.
3. Верифікація токена за допомогою секретного ключа.
4. Передача обробки далі лише за умови успішної перевірки.

Лістинг 2.2.2.1 Базовий Guard для перевірки токена:

```
import { Injectable, CanActivate, ExecutionContext, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
@Injectable()
export class JwtAuthGuard implements CanActivate {
 constructor(private readonly jwtService: JwtService) {}
 canActivate(context: ExecutionContext): boolean {
  const request = context.switchToHttp().getRequest();
  const authHeader = request.headers.authorization;
  if (!authHeader) {
   throw new UnauthorizedException('Token is missing');
  }
  const token = authHeader.split(' ')[1];
  try {
   const decoded = this.jwtService.verify(token);
   request.user = decoded;
   return true;
  } catch (error) {
   throw new UnauthorizedException('Invalid token'); }}}
```

Кастомний Guard для ролей

Для обмеження доступу до певних функцій залежно від ролі користувача (адміністратор, тренер, клієнт) реалізовано кастомний Guard. Цей Guard перевіряє роль користувача, зазначену в токені, перед доступом до ресурсу.

Лістинг 2.2.2.2 Кастомний Guard

```
import { Injectable, CanActivate, ExecutionContext, ForbiddenException } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
@Injectable()
export class RolesGuard implements CanActivate {
 constructor(private readonly reflector: Reflector) {}
 canActivate(context: ExecutionContext): boolean {
  const requiredRoles = this.reflector.get<string[]>('roles', context.getHandler());
  if (!requiredRoles) {
   return true;
  }
  const request = context.switchToHttp().getRequest();
  const user = request.user;
  if (!user || !requiredRoles.includes(user.role)) {
   throw new ForbiddenException('Access denied');
  }
  return true;
 }}
```

Guard навішується на методи контролера або ресолвери за допомогою декораторів @UseGuards та @SetMetadata.

Лістинг 2.2.2.3 Реалізація резолвера з обмеженням доступу за ролями

```
import { Resolver, Query, UseGuards } from '@nestjs/graphql';
import { RolesGuard } from './guards/roles.guard';
import { JwtAuthGuard } from './guards/jwt-auth.guard';
import { SetMetadata } from '@nestjs/common';
@Resolver()
export class UserResolver {
 @Query(() => String)
 @UseGuards(JwtAuthGuard, RolesGuard)
 @SetMetadata('roles', ['admin']) // Доступ лише для адміністратора
 getAdminData() {
  return 'This data is accessible only by admin users.';
 }
 @Query(() => String)
 @UseGuards(JwtAuthGuard, RolesGuard)
 @SetMetadata('roles', ['trainer']) // Доступ лише для тренера
```

```
  getTrainerData() {
    return 'This data is accessible only by trainers.';
  }}
```

Основні переваги механізмів Guard

1. Модульність

Guards є окремими компонентами, які легко інтегруються у будь-які частини системи. Завдяки чіткому відокремленню їх логіки, вони не залежать від бізнес-логіки або інших частин програми, що полегшує їхнє тестування, обслуговування та повторне використання. Це дозволяє легко застосовувати Guards як до конкретних маршрутів, так і до цілих модулів.

2. Гнучкість

Guards можна адаптувати до будь-яких потреб системи. Наприклад:

- Створення кастомних Guards для перевірки специфічних умов доступу (ролі, права, специфічні атрибути користувача).
- Комбінування декількох Guards для складніших сценаріїв авторизації.
- Додавання контекстно-залежних правил доступу, наприклад, перевірка часу, місцезнаходження або стану облікового запису користувача.

3. Безпека

Guards виступають як перша лінія захисту системи:

- Верифікація аутентифікації та авторизації на рівні запиту ще до виконання бізнес-логіки.
- Зменшення ризику несанкціонованого доступу завдяки централізованій перевірці токенів та ролей.
- Унеможливлення доступу до приватних ресурсів без відповідних дозволів.

4. Зручність

Guards дозволяють чітко розділяти логіку перевірки доступу та бізнес-логіку. Це:

● Забезпечує зручність у розробці, адже контролери та резолвери залишаються "чистими" від зайвої перевірки.

● Підвищує читабельність коду, зосереджуючи увагу розробників на основній функціональності програми.

● Полегшує внесення змін у логіку доступу завдяки централізованому розташуванню перевірок у Guards.

5. Універсальність

Guards працюють як у контексті REST API, так і GraphQL:

● У GraphQL Guards інтегруються з резолверами, перевіряючи доступ до конкретних запитів.

● У REST API Guards можна застосовувати до маршрутів контролерів на рівні окремих методів або всього класу.

6. Масштабованість

Guards легко масштабуються разом із системою:

● У великих проєктах з багатьма рівнями доступу можна створювати окремі Guards для кожного рівня або модуля.

● Завдяки чіткій структурі Guards легко інтегруються у мікросервісну архітектуру, забезпечуючи узгоджену політику доступу до ресурсів.

7. Розширюваність

NestJS Guards підтримують декоратори та метадані, що дозволяє їх легко розширювати. Наприклад:

● Використання кастомних метаданих для визначення умов доступу (ролей, атрибутів).

● Інтеграція з іншими модулями, такими як **Passport**, для налаштування стратегій аутентифікації.

Реалізація Guards для контролю доступу на основі токенів та ролей не тільки підвищує безпеку системи, але й сприяє її структурованості, масштабованості та зручності в обслуговуванні. Вони гарантують чіткий розподіл прав доступу, мінімізуючи ризики та забезпечуючи високий рівень захисту даних.

2.4 Впровадження інфраструктурного рівня та бізнес логіки

Впровадження інфраструктурного рівня та бізнес-логіки є однією з найважливіших частин розробки інформаційної системи, оскільки саме ці елементи забезпечують її функціональність та стабільність роботи. У цьому розділі детально розглядаються процеси налаштування ключових компонентів проекту, їх інтеграції та забезпечення взаємодії, що є основою для ефективного управління спортивною студією.

2.4.1 Базове налаштування проекту

Налаштування GraphQL

GraphQL обрано для реалізації API-рівня завдяки його перевагам у гнучкості запитів і можливості мінімізувати надлишкові дані у відповідях. Це забезпечує оптимальну взаємодію між клієнтом і сервером, особливо у складних системах із багатьма типами даних. У рамках проєкту було налаштовано GraphQL API з використанням бібліотеки Apollo Server, інтегрованої через модуль GraphQLModule у NestJS.

Основні етапи налаштування:

1. **Ініціалізація GraphQL-модуля** Для інтеграції GraphQL у проєкт було використано GraphQLModule з підтримкою драйвера ApolloDriver. Це дозволяє швидко налаштувати сервер GraphQL, зокрема для генерації схем та обробки запитів.

Лістинг 2.4.1.1 Реалізація налаштування модуля

```
        @Module({
imports: [
  ConfigModule.forRoot({
    isGlobal: true,
    envFilePath: './configs/.env',
  }),
  MikroOrmModule.forRoot(mikroOrmConfig),
  GraphQLModule.forRootAsync<ApolloDriverConfig>({
    driver: ApolloDriver,
    useFactory: (configService: ConfigService) => ({
      playground: false,
      autoSchemaFile: true,
      sortSchema: true,
      plugins: [ApolloServerPluginLandingPageLocalDefault()],
    }),
    inject: [ConfigService],
```

```
  }),
  AuthModule,
  UsersModule,
  AbonamentsModule,
  MembershipPlansModule,
  TrainingModule,
  ScheduleModule,
  ScheduleTrainingModule,
 ],
 controllers: [AppController],
 providers: [AppService, AppResolver],
})
export class AppModule {}
```

Основні компоненти:

- ApolloServerPluginLandingPageLocalDefault: Забезпечує інтеграцію Apollo Server у середовище NestJS.
- autoSchemaFile: Автоматично генерує файл схеми GraphQL на основі декораторів, використаних у коді.
- Плагін ApolloServerPluginLandingPageLocalDefault: Надає зручний веб-інтерфейс для тестування API під час розробки.

2. **Розробка структури схем.** Схеми GraphQL визначають структуру даних, які можна отримувати або змінювати через API. Основні типи (об'єкти) були створені для відображення сутностей бази даних, таких як:
   - Користувачі (Users): включають ролі (адмін, тренер, клієнт) та особисті дані.
   - Тренери (Trainers): містять інформацію про кваліфікацію та доступність.
   - Заняття (Sessions): включають час, тривалість, типи тренувань тощо.

Лістинг 2.4.1.2 Базовий Guard для перевірки токена:

```
import { Field, ObjectType } from '@nestjs/graphql';

import { BaseType } from '../../base-entity';
import { User } from './user.entity';
import { RoleEnum } from './role.enum';
```

```
@ObjectType()
export class UserType extends BaseType implements User {
 @Field()
 username?: string;

 @Field({ nullable: true })
 firstName?: string;

 @Field({ nullable: true })
 lastName?: string;

 @Field({ nullable: true })
 email!: string;

 @Field({ nullable: true })
 phoneNumber?: string;

 @Field(() => [RoleEnum])
 roles?: RoleEnum[] = [];
}
```

Кожен тип має пов’язану бізнес-логіку, яка реалізується через резолвери.

3. Налаштування запитів і мутацій. Для отримання даних реалізовано запити (Queries), а для внесення змін у базу даних — мутації (Mutations). Запити та мутації відображають функції бізнес-логіки, забезпечуючи можливість інтерактивного управління даними через GraphQL API.

Лістинг 2.4.1.3 Базовий Guard для перевірки токена:

```
        @Resolver(() => User)
export class UserResolver {
 @Query(() => [User])
 getAllUsers() {
  return this.userService.findAll();
 }

 @Mutation(() => User)
 createUser(
  @Args('input')input:CreateUserInput,
 ) {
  return this.userService.create({ name, email, role });
 }
}
```

Використання декораторів:

- @Query: Визначає запит.
- @Mutation: Визначає мутацію.
- @Args: Забезпечує передачу аргументів у запит або мутацію.

4. Тестування GraphQL API. Для перевірки працездатності API використовуються інтерактивні інструменти:

- GraphQL Playground (або його сучасний аналог від Apollo): дозволяє виконувати запити в зручному веб-інтерфейсі.
- Автоматична генерація документації: Схема GraphQL слугує джерелом документації для API, спрощуючи ознайомлення розробників з доступними запитами та мутаціями.

Лістинг 2.4.1.4 GraphQL-запит для отримання даних користувача

```
query = {
getUser {
 username;
 firstName;
 lastName;
 email;
 phoneNumber;
 roles;
    }}
```

Лістинг 2.4.1.5 Результат JSON

```
{
  "data": {
 "getUser": {
  "username": "john_doe",
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "phoneNumber": "123-456-7890",
  "roles": ["user", "trainer"]
    }}
```

Основні переваги використання GraphQL у проєкті:

- Гнучкість: Клієнт отримує саме ті дані, які потрібні, без надлишкових полів.
- Ефективність: Мінімізація кількості запитів до серверу.
- Універсальність: Одна схема підтримує як отримання, так і зміну даних.

● Масштабованість: Легке розширення функціоналу шляхом додавання нових типів, запитів і мутацій.

Таким чином, GraphQL API забезпечує ефективну взаємодію між клієнтом і сервером, відповідаючи потребам сучасної інформаційної системи.

Налаштування системи контролю версій (Git)

Git використовується для управління версіями коду та забезпечення спільної роботи над проектом. Основні кроки налаштування включали:

1. Ініціалізація репозиторію: Проект був доданий до локального репозиторію, а потім синхронізований із віддаленим репозиторієм на GitHub для збереження резервної копії.
2. Організація гілок: Для різних функціональних модулів було створено окремі гілки, що дозволяє незалежно працювати над частинами проекту без ризику зламу основної версії.
3. Налаштування .gitignore: До файлу .gitignore були додані чутливі дані (наприклад, .env) і службові файли, які не повинні потрапити у віддалений репозиторій.
4. Інтеграція Git-хуков: Використано Git-хуки для перевірки якості коду перед комітом, що сприяє підтримці стандартів кодування.

Налаштування бази даних

База даних є критично важливим компонентом системи, відповідальним за збереження та управління даними. Для цього проєкту обрано PostgreSQL як систему управління базами даних завдяки її надійності, масштабованості та потужним можливостям для складних запитів. Для зручності роботи з базою даних використано MikroORM — сучасну ORM, яка забезпечує абстрагований доступ до бази даних і підтримку міграцій.

1. Вибір системи управління базами даних (СУБД)

MikroORM було обрано через її легку інтеграцію з NestJS, підтримку TypeScript, можливість автогенерації міграцій та зручну роботу з об’єктно-реляційним мапінгом. Вона надає потужні інструменти для роботи зі складними структурами даних та гнучке налаштування.

Інтеграція ORM у проект виконується через MikroOrmModule.

Лістинг 2.4.1.6 Конфігурація модуля в AppModule

```
MikroOrmModule.forRoot(mikroOrmConfig),
```

Конфігурація зчитується з файлу mikro-orm.config.ts.

2. Налаштування конфігурації MikroORM

Файл mikro-orm.config.ts містить основні параметри підключення до бази даних, включаючи:

- Тип бази даних (postgresql).
- Дані для підключення (назва бази, користувач, пароль, хост, порт).
- Список сутностей (entities).
- Шляхи для збереження міграцій.

Лістинг 2.4.1.7 Конфігураційний файл

```
import { defineConfig } from '@mikro-orm/core';

        export default defineConfig({
  type: 'postgresql',
  dbName: process.env.POSTGRES_DB,
  user: process.env.POSTGRES_USER,
  password: process.env.POSTGRES_PASSWORD,
  host: process.env.POSTGRES_HOST,
  port: Number(process.env.POSTGRES_PORT),
  entities: [
    BaseDocument,
    User,
    Abonament,
    Training,
    MembershipPlan,
    Schedule,
    ScheduleTraining,
  ],
  migrations: {
    path: './dist/migrations',
          pathTs: './migrations',
    disableForeignKeys: false,   },
});
```

Цей файл містить всі налаштування для з'єднання з базою даних та управління її схемою

3. Створення сутностей (Entities)

Сутності відображають структуру таблиць бази даних. У MikroORM кожна сутність визначається як клас з відповідними анотаціями.

Лістинг 2.4.1.8 Сутність для таблиці користувачів

```
import { Field, ObjectType } from '@nestjs/graphql';
import { Entity, PrimaryKey, Property } from '@mikro-orm/core';
import * as crypto from 'crypto';

import { BaseDocument } from '../../base-entity';
import { userDictionary } from './user.dictionary';

const { email, lastName, firstName, phoneNumber, roles, username, password } =
 userDictionary;
@Entity()
export class User extends BaseDocument {
 @Property({ name: username, nullable: true })
 username?: string;

 @Property({ name: firstName, nullable: true })
 firstName?: string;

 @Property({ name: lastName, nullable: true })
 lastName?: string;

 @Property({  name: email, unique: true })
 email!: string;

 @Property({ name: phoneNumber, nullable: true })
 phoneNumber?: string;

 @Property({ name: password, hidden: true })
 password!: string;

 @Property({ name: roles, type: 'array' })
 roles?: string[] = [];

}
```

4. Налаштування міграцій

Міграції використовуються для управління змінами структури бази даних (додавання нових таблиць, змінення полів тощо). MikroORM дозволяє автоматично створювати та виконувати міграції.

Автогенерація міграцій: Для автоматичного створення файлів міграцій на основі змін у сутностях використовується команда:

```
npx mikro-orm migration:create
```

Це створює файл міграції у папці, визначеній у конфігурації (pathTs).

Команда виконання міграцій:

```
npx mikro-orm migration:up
```

Ця команда застосовує всі невиконані міграції до бази даних.

Лістинг 2.4.1.9 Файл міграції

```
import { Migration } from '@mikro-orm/migrations';

export class Migration20241119220209 extends Migration {

 async up(): Promise<void> {

   this.addSql('create table "user" ("id" uuid not null, "created_at" timestamptz(0) not null, "updated_at" timestamptz(0) not null, "deleted_at" timestamptz(0) null, "username" varchar(255) not null, "first_name" varchar(255) not null, "last_name" varchar(255) not null, "email" varchar(255) not null, "phone_number" varchar(255) not null, "password" varchar(255) not null, "roles" text[] not null, constraint "user_pkey" primary key ("id"));');
  this.addSql('alter table "user" add constraint "user_email_unique" unique ("email");');
 }

 async down(): Promise<void> {
  this.addSql('drop table if exists "user" cascade;');
 }
}
```

5. Переваги використання MikroORM

- Зручність: Легке створення сутностей, генерація схем та виконання міграцій.
- Автоматизація: Автогенерація міграцій на основі змін у сутностях.
- Масштабованість: Підтримка складних реляційних структур і взаємозв'язків.
- Гнучкість: Конфігурація може бути змінена залежно від середовища (розробка, тестування, продакшн).

Налаштування бази даних із використанням MikroORM дозволяє швидко створити, налаштувати та обслуговувати базу даних для інформаційної системи. Завдяки автогенерації міграцій та зручній роботі зі сутностями процес розробки суттєво прискорюється, забезпечуючи стабільність та гнучкість системи.

2.4.2 Налаштування модуля тестування

Для забезпечення високої якості програмного забезпечення, стабільності та надійності системи, у проекті був налаштований модуль тестування. Основним інструментом для написання та виконання тестів обрано **Jest**, що є популярною платформою для тестування JavaScript/TypeScript-додатків. Jest забезпечує високу швидкість виконання тестів, зручний синтаксис і вбудовані інструменти для роботи з асинхронними операціями, модулями та перевіркою результатів.

Етапи налаштування модуля тестування

1. Інсталяція Jest

Для інтеграції Jest у проект було встановлено необхідні залежності:

```
npm install --save-dev jest @nestjs/testing ts-jest @types/jest
```

- jest: Основна бібліотека для тестування.
- @nestjs/testing: Інструменти для тестування у середовищі NestJS.
- ts-jest: Адаптер для роботи Jest з TypeScript.
- @types/jest: Типи для Jest у проектах з TypeScript.

Додатково створено конфігураційний файл Jest для коректної роботи з NestJS і TypeScript.

2. Налаштування конфігурації Jest

Файл jest.config.ts був адаптований для підтримки специфіки проєкту:

Лістинг 2.4.2.1.Конфігурація Jest

```
export default {
moduleFileExtensions: ['js', 'json', 'ts'],
rootDir: './',
testRegex: '.*\\.spec\\.ts$',
transform: {
```

```
  '^.+\\.(t|j)s$': 'ts-jest',
 },
 collectCoverage: true,
          coverageDirectory: './coverage',
          testEnvironment: 'node',
          setupFilesAfterEnv: ['./jest.setup.ts'],
         };
```

У цьому файлі визначено:

- Пошук тестових файлів (*.spec.ts).
- Використання ts-jest для обробки TypeScript.
- Збір покриття коду тестами та збереження звітів у папці coverage.

3. Створення тестових сценаріїв

Для кожного модуля системи були створені окремі тестові файли, наприклад:

- Тестування сервісів (unit-тести): Забезпечує перевірку бізнес-логіки.

Лістинг 2.4.2.2. Юніт-тест для сервісу UserService

```
        import { Test, TestingModule } from '@nestjs/testing';
import { UserService } from './user.service';

describe('UserService', () => {
 let service: UserService;

 beforeEach(async () => {
  const module: TestingModule = await Test.createTestingModule({
   providers: [UserService],
  }).compile();

  service = module.get<UserService>(UserService);
 });

 it('should be defined', () => {
  expect(service).toBeDefined();
 });

 it('should create a user', async () => {
  const user: User = {
              id: 1,
```

```
            username: `john`,
            firstName: `test`,
            lastName: `Sky`,
            email: 'john.doe@example.com',
            role: 'user', };

    jest.spyOn(service, 'create').mockImplementation(() => user);

    expect(await service.create(user)).toEqual(user);
  });
});
```

4. Тестування асинхронних операцій

Оскільки система працює з базою даних і виконує асинхронні операції, було налаштовано спеціальні інструменти для перевірки таких сценаріїв.

Лістинг 2.4.2.3. Тест з асинхронною операцією

```
        it('should fetch user by ID', async () => {
  const user = { id: 1, name: 'John Doe' };
  jest.spyOn(service, 'findOne').mockResolvedValue(user);

  const result = await service.findOne(1);
  expect(result).toEqual(user);
});
```

5. Перевірка покриття коду тестами

Для оцінки рівня покриття коду тестами використовується команда:

```
npx jest --coverage
```

Це генерує звіт у папці `coverage`, який включає:

- **Functions**: Відсоток протестованих функцій.
- **Statements**: Відсоток протестованих рядків коду.
- **Branches**: Покриття умовних конструкцій.

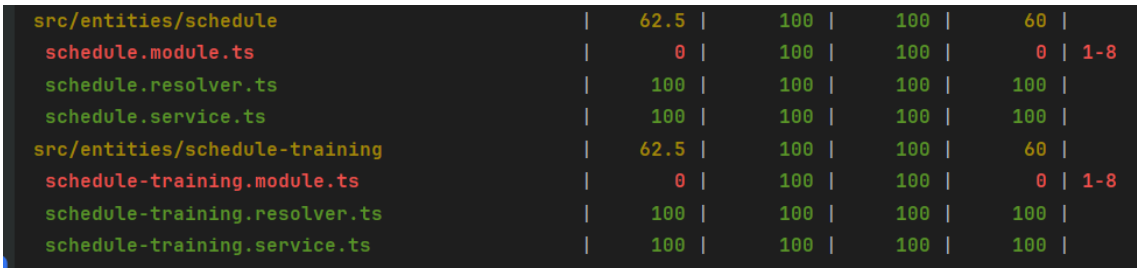

Рис. 2.4.2.1. Покриття тестами модулів

Джерело: розроблено автором

Переваги використання Jest у проекті

1. Швидкість і ефективність: Jest виконує тести швидко завдяки паралельному виконанню.
2. Зручний синтаксис: Простота написання як юніт-, так і інтеграційних тестів.
3. Вбудована підтримка покриття: Автоматичний аналіз покриття коду.
4. Інтеграція з NestJS: Jest легко налаштовується для роботи з модулями та компонентами NestJS.
5. Підтримка асинхронності: Зручна перевірка операцій, які взаємодіють із базою даних чи API.

Таким чином, налаштування модуля тестування дозволяє автоматизувати перевірку функціональності системи, виявляти помилки ще на етапі розробки та забезпечувати високу якість кінцевого продукту.

2.4.3 Реалізація базових компонентів для перевикористання

У процесі розробки інформаційної системи для керування спорт-студією було створено набір загальних компонентів, які дозволяють забезпечити перевикористання коду та спрощують розробку нових модулів. Це включає базову реалізацію сутностей, сервісів, типів запитів та інших елементів, які використовуються в різних частинах системи.

Базова сутність (BaseEntity)

Для уніфікації роботи з базою даних була створена базова сутність BaseEntity, яка включає поля та методи, спільні для всіх таблиць бази даних. До них відносяться:

- ID: Унікальний ідентифікатор запису.
- CreatedAt та UpdatedAt: Поля для зберігання інформації про дату створення та останнього оновлення запису.
- DeletedAt: Поле для логічного видалення записів.

BaseEntity забезпечує стандартизовану структуру, яка значно спрощує додавання нових сутностей і полегшує підтримку цілісності даних у системі.

Data Transfer Object (DTO)

Було створено базовий клас для DTO, що дозволяє стандартизувати передачу даних між різними рівнями системи. Загальні DTO включають:

- PaginationDTO: Використовується для роботи з посторінковими запитами.
- OrderDTO: Дозволяє визначати сортування даних (за зростанням чи спаданням).
- WhereDTO: Використовується для генерації умов пошуку.

Ці DTO створюють універсальний підхід до обробки запитів, що дозволяє уникнути дублювання коду.

Загальні сервіси

Були реалізовані базові сервіси, які містять методи, спільні для роботи з будь-якими сутностями. Основний функціонал включає:

- CRUD-операції: Створення, читання, оновлення та видалення записів.
- Пошук за умовами: Метод для виконання запитів із використанням умов (where).
- Сортування: Універсальний метод для реалізації сортування записів за переданими параметрами.
- Пагінація: Загальний механізм для посторінкового виведення даних.

Завдяки використанню цих сервісів забезпечується стандартизований підхід до роботи з даними, що скорочує час розробки та зменшує ймовірність помилок.

Взаємодія між компонентами

Ці базові елементи інтегровані між собою: BaseEntity використовується як основа для створення нових сутностей, сервіси взаємодіють із DTO для

обробки запитів, а типи запитів забезпечують зручну конфігурацію сортування, фільтрації та пагінації.

Переваги перевикористання

Реалізація базових компонентів для перевикористання надає такі переваги:

- Зниження дублювання коду: Загальні методи та типи зосереджені в одному місці.
- Покращення читабельності та підтримки: Стандартизований підхід робить код зрозумілішим для інших розробників.
- Простота розширення: Додавання нових сутностей чи функціоналу виконується швидше завдяки використанню вже реалізованих компонентів.

Таким чином, базова реалізація компонентів стала основою для ефективної розробки інформаційної системи, дозволивши значно оптимізувати процес створення й інтеграції нових модулів.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було спроектовано та реалізовано серверну частину інформаційної системи спортивної студії. Основою стала база даних, яка забезпечує структуроване зберігання інформації про користувачів, тренування, платежі та відвідування. У підрозділі 2.1 детально описано таблиці бази даних і логічні зв’язки між ними, з акцентом на цілісність даних та використання зовнішніх ключів.

Для прикладного рівня вибрали GraphQL як основний інтерфейс, що дозволяє клієнту гнучко формувати запити та отримувати лише необхідні дані, зменшуючи мережеве навантаження. Це підвищує продуктивність і масштабованість системи.

Підрозділ 2.2 присвячений авторизації з ролями (клієнт, тренер, адміністратор) на базі JWT. Реалізовано механізми захисту GraphQL-запитів за допомогою кастомних Guard-компонентів, які контролюють доступ згідно з ролями.

У підрозділі 2.4 описано налаштування інфраструктури: базове конфігурування проєкту, модуль тестування та сервісні компоненти для повторного використання коду. Особливу увагу приділено бізнес-логіці, що забезпечує ключові процеси — від реєстрації до управління розкладом і фінансами.

В результаті було створено масштабовану, безпечну серверну архітектуру, яка відповідає сучасним вимогам безпеки, ефективності та гнучкості. Реалізація на основі GraphQL дозволила досягти високої адаптивності до змін у клієнтській частині системи, що є особливо важливим для подальшого розвитку інформаційної системи спортивної студії.

РОЗДІЛ 3. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1 Загальні принципи побудови клієнтської частини

Клієнтська частина інформаційної системи для фітнес-студії була реалізована з використанням сучасного JavaScript-фреймворку **React**. Обрання цієї технології обумовлено її перевагами: компонентною структурою, високу продуктивністю та активною підтримкою спільноти розробників. React дозволяє будувати інтерфейс користувача у вигляді набору незалежних компонентів, кожен з яких відповідає за окрему частину сторінки чи функціоналу.

Для розробки інтерфейсу було використано бібліотеку **Chakra UI**, яка забезпечує швидке створення адаптивного та естетичного дизайну завдяки готовим UI-компонентам. Chakra UI підтримує темізацію, контроль стилів за допомогою JavaScript та має потужну систему токенів дизайну.

Керування глобальним станом програми реалізовано за допомогою **Redux Toolkit** — офіційного рекомендованого інструменту для зручної роботи з Redux у сучасних проєктах. Це дозволяє зручно керувати станом користувача, сесією, авторизацією, фільтрами тощо.

Для організації навігації між сторінками застосовано бібліотеку **React Router**, яка забезпечує ефективний механізм маршрутизації з підтримкою динамічних параметрів, вкладених маршрутів та захищених маршрутів (guarded routes) залежно від ролі користувача.

Уся клієнтська частина проєкту організована за принципами SPA (Single Page Application), що дозволяє забезпечити швидке перемикання між сторінками без перезавантаження сайту, зменшуючи навантаження на сервер та підвищуючи зручність взаємодії для користувача.

3.2 Модуль авторизації

Однією з ключових частин реалізації клієнтської частини інформаційної системи фітнес-студії є створення ефективної та безпечної системи авторизації користувачів. Авторизація забезпечує доступ користувачів до функціоналу системи відповідно до їх ролі (адміністратор, тренер або клієнт). Це дозволяє не лише персоналізувати взаємодію з інтерфейсом, а й обмежити доступ до критично важливих функцій та даних.

У рамках клієнтської частини було створено окремі сторінки для авторизації (входу до системи) та реєстрації нового користувача. Користувач вводить електронну пошту та пароль, після чого відбувається перевірка правильності введених даних через запит до сервера. У разі успішної перевірки користувач отримує доступ до системи, а у разі помилки — виводиться відповідне повідомлення. Після входу в систему користувач автоматично перенаправляється на сторінку, що відповідає його ролі.

Особливістю реалізації є підтримка динамічного маршрутизаційного доступу на основі ролей користувача. Наприклад, адміністраторам відкривається інтерфейс управління тренерами, клієнтами та розкладом, тоді як тренери мають доступ лише до власного розкладу тренувань та списку підопічних, а клієнти — до перегляду та запису на заняття.

У системі реалізовано захист маршрутів (так звані guard'и), які перевіряють, чи має користувач право переглядати конкретну сторінку. У разі спроби перейти на сторінку, яка не відповідає ролі користувача або потребує авторизації, система автоматично перенаправляє його на сторінку входу. Це забезпечує базовий рівень безпеки клієнтської частини та запобігає несанкціонованому доступу.

Після успішного входу, система зберігає авторизаційний токен та роль користувача у локальному сховищі браузера та у глобальному стані застосунку. Це дозволяє зберігати сесію користувача навіть після оновлення сторінки або повторного входу до застосунку. При завантаженні клієнтської частини система

автоматично перевіряє наявність даних у сховищі та, за необхідності, відновлює сесію.

У випадку реєстрації нового користувача система пропонує заповнити відповідну форму, яка містить поля для імені, електронної пошти, пароля та, в окремих випадках, ролі (наприклад, якщо реєстрація проводиться адміністратором). Після успішної реєстрації новий користувач одразу авторизується у системі та перенаправляється до відповідного розділу інтерфейсу.

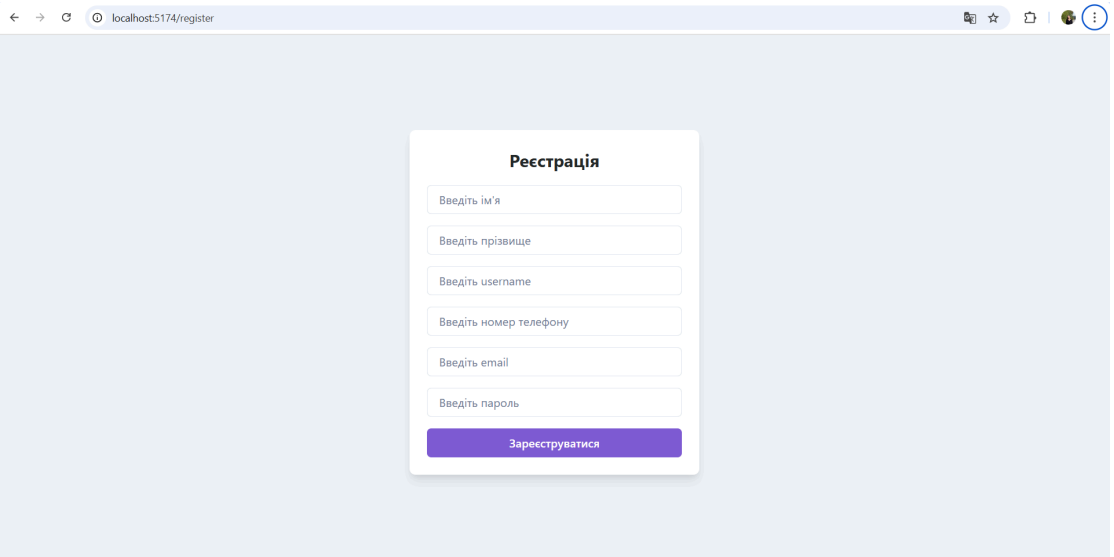

Рис. 3.2.1 Форма реєстрації користувача

Джерело: розроблено автором

Для зручності користувачів також реалізовано функціонал виходу з облікового запису. Після натискання кнопки «Вийти» система очищує локальне сховище, скидає глобальний стан авторизації та перенаправляє користувача на сторінку входу.

Таким чином, реалізація авторизації в клієнтській частині фітнес-системи поєднує сучасні підходи до безпеки, зручність користування, підтримку ролей та захист маршрутів. Це дозволяє не лише контролювати доступ до функцій застосунку, але й забезпечувати належний користувацький досвід для кожної категорії користувачів.

3.3 Ролі та доступи

Важливою складовою побудови інформаційної системи фітнес-студії є впровадження ролей користувачів та визначення рівнів доступу відповідно до цих ролей. Система ролей дозволяє структурувати взаємодію з інтерфейсом, уникати перевантаження користувача зайвим функціоналом та, найголовніше, забезпечити безпеку доступу до критичних даних і функцій.

У реалізованій системі застосовано рольову модель доступу, за якої кожному користувачеві при вході до системи призначається певна роль. На основі цієї ролі система визначає, які частини інтерфейсу мають бути доступні користувачеві, а які — приховані або заборонені.

У клієнтській частині система реалізує механізм фільтрації маршрутів (шляхів переходу в інтерфейсі) залежно від ролі. Це означає, що навіть якщо користувач вручну введе адресу сторінки у браузері, він не отримає доступ до неї, якщо не має відповідного рівня дозволу. Така перевірка виконується на рівні гардів маршрутизатора — спеціальних механізмів, які контролюють, чи має користувач право перейти на певний маршрут.

Механізм доступу працює таким чином: під час входу користувач проходить авторизацію, і система отримує його роль (наприклад, «адміністратор», «тренер», «клієнт»). Ця роль зберігається в глобальному стані застосунку, а також у локальному сховищі браузера, що дозволяє зберігати її навіть після перезавантаження сторінки. Кожен маршрут, що реалізований у системі, має прив’язку до певної ролі або до набору ролей. При переході на маршрут перевіряється, чи відповідає роль користувача дозволеному списку. Якщо ні — система перенаправляє його на доступну сторінку або сторінку з повідомленням про відмову в доступі.

Цей підхід має ряд переваг. По-перше, він підвищує безпеку: користувачі не можуть отримати доступ до функцій, які їм не призначені. По-друге, він забезпечує зручність використання, оскільки кожен бачить лише релевантні

йому елементи інтерфейсу. По-третє, реалізація нових ролей або зміна логіки доступу не потребує значних змін в архітектурі, оскільки рольова логіка зосереджена в одному місці.

Візуалізувати модель доступу можна за допомогою діаграми, яка демонструє, як система маршрутизує запити залежно від ролі користувача.

3.2.1 Адміністратор:

1. Управління користувачами

Фронтенд для цієї частини передбачає створення інтерфейсу для перегляду, редагування, додавання та видалення користувачів. Оскільки адміністратор має доступ до всіх ролей (клієнт, тренер, інший адміністратор), інтерфейс повинний містити:

- Список користувачів: таблиця або карта карток користувачів з основними даними: ім'я, роль, статус (активний/заблокований), кнопки для редагування або видалення.
- Форма додавання користувача: з полями для введення імені, електронної пошти, пароля, ролі (вибір між клієнтом, тренером або адміністратором), статусу активності.
- Форма редагування користувача: з можливістю редагувати ім’я, роль, статус, електронну пошту. Перевірка на коректність введених даних здійснюється на фронтенді через валідацію форми.
- Інтерфейс блокування користувачів: для блокування користувача передбачено кнопку або чекбокс для зміни статусу на "заблокований".

Бізнес-логіка:

При додаванні нового користувача або редагуванні даних фронтенд перевіряє правильність введених даних (наприклад, формат email, довжина пароля). Після успішної валідації форма відправляється на бекенд через API-запит для збереження або оновлення даних у базі.

2. Управління розкладом тренувань

Для управління розкладом тренувань адміністраторами необхідно надати такі функціональні можливості:

- Перегляд списку тренувань: таблиця з інформацією про кожне тренування: дата, час, тип, тренер, кількість доступних місць.
- Форма створення нового тренування: поля для введення дати, часу, типу тренування, вибору тренера з випадаючого списку, кількості доступних місць.

- Редагування тренувань: можливість змінювати дату, час, тренера та кількість місць для вже створених тренувань.
- Скасування тренувань: кнопка для скасування тренування, яка виводить підтвердження. Після скасування система автоматично надсилає повідомлення всім учасникам через API-запит до сервера.

Бізнес-логіка:

Після створення або редагування тренування на фронтенді здійснюється валідація введених даних. Якщо введені дані правильні, відправляється запит на сервер для оновлення розкладу. Для скасування тренування система генерує запит на сервер для зміни статусу тренування в базі даних та сповіщення учасників.

3. Управління абонементами

Інтерфейс для роботи з абонементами передбачає:

- Перегляд існуючих абонементів: таблиця з типами абонементів, цінами, кількістю доступних відвідувань і тривалістю дії.
- Додавання нових абонементів: форма для введення ціни, тривалості дії, кількості відвідувань, додаткових умов.

- Редагування існуючих абонементів: можливість змінювати ці параметри для існуючих абонементів. Кнопка для деактивації старих пропозицій.

Бізнес-логіка:

Форма додавання абонемента перевіряє коректність введених даних, таких як ціна та тривалість. Після перевірки, відправляється запит на сервер для додавання або редагування абонементів у базі даних. Зміни миттєво відображаються в клієнтській частині системи через реактивне оновлення інтерфейсу.

4. Обробка заявок на купівлю абонементів

Процес обробки заявок включає такі функції:

- Перевірка заявок: адміністратор може переглядати заявки на покупку абонементів від клієнтів.
- Підтвердження оплат**и:** після підтвердження оплати адміністратор натискає кнопку для активації абонемента в профілі клієнта. Процес підтвердження оплати вимагає вручну введеного підтвердження, що гарантує точність і контролює активність.

Бізнес-логіка:

Після підтвердження оплати система активує абонемент у базі даних і дозволяє клієнту записуватися на тренування. Запит на активацію абонемента відправляється на сервер, і система надає доступ до функцій.

5. Перегляд аналітики

Для аналітики необхідний інтерфейс:

- Графіки та таблиці: для відображення статистики відвідуваності, заповнюваності груп, кількості тренувань, аналітика активності користувачів.

- Фільтри: можливість вибору періоду (тиждень, місяць, рік) для більш детального аналізу.

Бізнес-логіка:

Після запиту на перегляд статистики на фронтенді система взаємодіє з бекендом для отримання аналітичних даних через API. Відображення результатів здійснюється за допомогою графіків і таблиць. Всі зміни в даних відображаються в реальному часі.

3.2.2 Тренер: Реалізація з точки зору фронтенду

1. Перегляд розкладу тренувань

Тренер має доступ до:

- Інтерфейс розкладу: календар або таблиця, що показує заплановані тренування, з можливістю перегляду деталей (дата, час, назва тренування).
- Перелік учасників**:** для кожного тренування список учасників з можливістю перегляду статусу абонемента (активний/неактивний).

Бізнес-логіка:

Після авторизації тренер отримує дані через API-запит, який фільтрує тренування, закріплені за конкретним тренером. Список учасників оновлюється в реальному часі.

2. Відмітка присутності

Для кожного тренування:

- Чекбокси або кнопки для позначки присутності: для кожного учасника тренер може відмітити, чи був він на занятті.
- Автоматичне оновлення: після відмітки присутності, інформація оновлюється в базі даних, і статус користувача змінюється.

Бізнес-логіка:

Після відмітки присутності, фронтенд відправляє запит на сервер для зміни статусу відвідування конкретного користувача в базі даних.

3. Перегляд історії тренувань

Тренер має доступ до:

- **Історії тренувань:** таблиця з інформацією про проведені тренування, кількість учасників і відсоток заповнюваності.

Бізнес-логіка:

Дані про історію тренувань запитуються через API і відображаються у вигляді таблиць або графіків для аналізу ефективності.

Таким чином, бізнес-логіка на фронтенді забезпечує динамічне оновлення даних і дозволяє користувачам ефективно взаємодіяти з системою через інтуїтивно зрозумілі інтерфейси.

3.2.3 Клієнт: Реалізація з точки зору фронтенду

1. Перегляд доступних тренувань

Клієнт може переглядати список доступних тренувань, де відображаються:

- Дата та час тренування
- Тип тренування (йога, фітнес, аеробіка тощо)
- Ім’я тренера
- Записані учасники та доступні місця

Інтерфейс реалізовано у вигляді таблиці або календаря, з можливістю фільтрації за типом тренування або періодом.

Бізнес-логіка:

Для кожного доступного тренування система автоматично визначає кількість

вільних місць і відображає цю інформацію в реальному часі. Якщо тренування заповнене, воно позначається як недоступне для запису.

2. Запис на тренування

Клієнт має можливість записатися на тренування, якщо для цього є вільні місця:

- **Кнопка "Записатися на тренування"** – з’являється для кожного тренування, якщо є вільні місця.
- Після натискання на кнопку, клієнт вводить необхідні дані (якщо це потрібно), наприклад, вибір абонемента (якщо його ще немає) і підтвердження.

Бізнес-логіка:

При запису перевіряється, чи є у клієнта активний абонемент. Якщо ні, система пропонує оформити абонемент або попереджає про необхідність оплатити тренування. Якщо запис успішний, система оновлює розклад клієнта і додає його до списку учасників тренування.

3. Перегляд особистого кабінету

У клієнта є доступ до:

- **Особистий розклад тренувань:** список всіх записаних на тренування, дата, час, статус (підтверджено, відвідано, пропущено).
- **Історія відвідувань:** історія всіх відвіданих тренувань з детальними відомостями: кількість присутніх, результат тренування (якщо доступно).

Інтерфейс відображає всі ці дані у вигляді таблиці або календаря, з можливістю фільтрації за датами або типами тренувань.

Бізнес-логіка:

Історія тренувань формується автоматично в момент відвідування кожного заняття. Статус відвідування (відвідав/не відвідав) оновлюється після проведення тренування та відображається в особистому кабінеті клієнта.

4. Управління абонементами

Клієнт може:

- **Переглядати наявні абонементи:** інтерфейс для перегляду інформації про активні абонементи, їх статус (активний/неактивний), кількість залишкових відвідувань.
- **Оформити новий абонемент:** у разі відсутності активного абонемента, клієнт може вибрати абонемент і оформити його через онлайн-форму. Оформлення абонемента включає вибір типу абонемента, перевірку наявності активних абонементів і підтвердження оплати.

Бізнес-логіка:

Перед оформленням абонемента система перевіряє, чи є в користувача діючий абонемент. Після успішного оформлення і підтвердження оплати, абонемент активується і клієнт отримує доступ до запису на тренування.

5. Повідомлення та сповіщення

Клієнт отримує різноманітні сповіщення, включаючи:

- **Нагадування про майбутні тренування** через push-сповіщення або email.
- **Інформація про скасування тренування:** якщо тренування було скасоване адміністратором, клієнт отримує сповіщення з деталями.
- **Інформація про підтвердження запису на тренування:** після успішного запису або скасування клієнт отримує повідомлення про статус.

Бізнес-логіка:

Повідомлення надсилаються автоматично за допомогою сервісів сповіщень, які взаємодіють із сервером для відправки даних про зміни (наприклад, зміна часу тренування, скасування, підтвердження запису).

6. Аналіз активності

У клієнта є можливість переглядати свою активність у системі:

- **Статистика відвідувань:** відображення відсотка відвідування тренувань, історія відвідувань.
- **Показники по абонементах:** скільки тренувань клієнт відвідав за поточний абонемент, скільки залишилось відвідувань.

Бізнес-логіка:

Ці показники оновлюються автоматично після кожного тренування і відображаються в реальному часі у розділі активності клієнта.

Загальні аспекти фронтенду для всіх ролей

1. **Роутинг та доступи**

   Для кожної ролі створюються окремі сторінки та доступи через роутінг, при цьому використовуються "гарди" для перевірки ролі користувача перед доступом до певних сторінок (наприклад, адміністратор має доступ лише до адміністративних функцій, а тренер тільки до свого розкладу).

2. **Реактивність інтерфейсу**

   Всі зміни (реєстрація, запис на тренування, зміни в абонементах) автоматично відображаються в інтерфейсі без перезавантаження сторінки, що забезпечується через використання бібліотек для реактивності, таких як React.

3. **Перевірка прав доступу**

   Реалізується через механізм гардів у роутері. Кожен маршрут перевіряє, чи має користувач необхідні права для доступу до цієї сторінки. У разі відсутності доступу, користувач перенаправляється на сторінку входу або на сторінку, відповідну до його ролі.

Таким чином, інтерфейс для клієнта є простим, інтуїтивно зрозумілим, з чітким відображенням усіх доступних можливостей і дозволяє ефективно управляти своїми запитами та тренуваннями.

ВИСНОВКИ ДО РОЗДІЛУ 3

У розділі 3 було детально розглянуто розробку клієнтської частини інформаційної системи для управління спортивною студією. Вибір технології React забезпечив гнучкість і високу продуктивність інтерфейсу користувача завдяки компонентній архітектурі. Застосування бібліотеки Chakra UI сприяло створенню сучасного, адаптивного та привабливого дизайну з урахуванням принципів зручності використання.

Для керування глобальним станом застосунку було впроваджено Redux Toolkit, що дозволяє ефективно обробляти дані користувача, стан авторизації та налаштувань. Навігація реалізована за допомогою React Router, який підтримує динамічні маршрути і розмежування доступу залежно від ролей користувачів, забезпечуючи безпеку та зручність користування.

Клієнтська частина побудована за принципом SPA, що забезпечує швидку роботу без необхідності перезавантаження сторінок, покращуючи користувацький досвід і знижуючи навантаження на сервер.

Особливу увагу приділено реалізації модулю авторизації та ролей, що дозволяє чітко розмежувати права доступу для адміністратора, тренера та клієнта. Кожна роль отримала набір функціональних можливостей, адаптованих до їхніх завдань — від управління користувачами і розкладом тренувань до відмітки присутності та перегляду особистого кабінету.

Реалізація клієнтської частини враховує не лише технічні, а й організаційні аспекти, що сприяють підтримці безпеки, зручності та масштабованості системи. Загалом, створена архітектура забезпечує інтерактивність, швидкодію та простоту подальшого розвитку функціоналу.

Таким чином, розділ 3 заклав міцний фундамент для ефективної роботи з користувачами різних ролей та подальшого розвитку інформаційної системи спортивної студії.

ВИСНОВКИ

У результаті виконаної роботи було підтверджено актуальність та необхідність розробки інформаційної системи для керування спортивною студією в умовах сучасного розвитку цифрових технологій та підвищеного попиту на автоматизацію бізнес-процесів у сфері спортивних послуг. У зв’язку зі зростанням кількості клієнтів, збільшенням обсягів інформації та потребою у швидкій обробці даних традиційні методи ведення обліку, такі як паперові журнали або електронні таблиці, вже не відповідають вимогам ефективного управління. Саме тому автоматизована система керування є логічним і необхідним кроком у напрямі підвищення якості сервісу, оптимізації внутрішніх процесів і зміцнення позицій спортивної студії на ринку.

У межах даної роботи було проаналізовано предметну область, визначено основні проблеми та потреби користувачів, сформульовано технічні й функціональні вимоги до майбутньої інформаційної системи. Було обґрунтовано вибір технологій, архітектурних підходів і програмних засобів для реалізації проєкту, зокрема використання об'єктно-орієнтованого підходу до проєктування, застосування реляційних баз даних для ефективного збереження та обробки інформації, а також інструментів тестування для забезпечення стабільної роботи програмного продукту.

Розроблена система передбачає низку важливих функцій, які дозволяють здійснювати облік клієнтів, формувати та редагувати розклад занять, керувати фінансовими надходженнями та витратами, а також адмініструвати діяльність персоналу. Однією з ключових особливостей є модульність і масштабованість системи, що дозволяє адаптувати її під індивідуальні потреби різних спортивних закладів і легко впроваджувати нові функції у разі необхідності.

Важливим етапом у процесі реалізації стало створення зручного веб-інтерфейсу для адміністраторів та розробка мобільного застосунку для клієнтів. Такий підхід забезпечує простий доступ до системи з будь-якого пристрою,

автоматизує запис на заняття, дозволяє здійснювати онлайн-оплату, переглядати історію відвідувань та взаємодіяти з адміністрацією закладу без необхідності фізичного відвідування студії. Це особливо актуально в умовах цифрової трансформації суспільства та прагнення до максимальної зручності користувача.

Окрім зручності, система забезпечує підвищення ефективності управління. Автоматизація рутинних процесів дозволяє адміністративному персоналу зосередитися на стратегічних завданнях, таких як покращення сервісу, маркетинг, залучення нових клієнтів. У той самий час, система сприяє зменшенню кількості помилок, викликаних людським фактором, підвищенню точності даних та прозорості у фінансових і організаційних питаннях.

Розроблена інформаційна система є універсальним рішенням, яке може бути адаптовано до потреб різних типів спортивних установ – від невеликих фітнес-студій до великих спортивно-оздоровчих центрів. Її гнучкість, масштабованість і орієнтованість на користувача робить її ефективним інструментом для підвищення якості обслуговування, забезпечення конкурентоспроможності та стабільного розвитку бізнесу у сфері спортивних послуг.

Таким чином, результати дослідження та практичної реалізації інформаційної системи доводять доцільність впровадження подібних рішень у спортивній індустрії. Створена система не лише оптимізує управлінські процеси, але й створює додаткову цінність для клієнтів, формуючи позитивний імідж спортивної студії та забезпечуючи високий рівень сервісу.

У підсумку можна зазначити, що поставлена мета дослідження – розробка інформаційної системи для ефективного керування спорт-студією – була повністю досягнута. Усі визначені завдання, включаючи аналіз предметної області, проєктування структури, вибір технологій, реалізацію та тестування програмного продукту, були виконані. Запропонована система має високий

потенціал для подальшого розвитку та впровадження у практичну діяльність, що є важливим кроком до цифровізації бізнес-процесів у сфері спорту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Адаменко О.І. Інформаційні системи та технології: навч. посібник. — К.: Видавничий дім «Києво-Могилянська академія», 2019. — 320 с.
2. Fowler M. Patterns of Enterprise Application Architecture. — Addison-Wesley Professional, 2003. — 560 p.
3. Hartig O., Pérez J. GraphQL: A query language for APIs. — 2018. URL: https://graphql.org/ .
4. Meijer E., Burnett M., Hirschfeld R. GraphQL in Action. — Manning Publications, 2021. — 300 p.
5. Jones B., Smith L. Modern Authentication with JWT. — O'Reilly Media, 2020. — 220 p.
6. Chorafas D.N. Token-based Authentication: Concepts and Practice. — Springer, 2019. — 180 p.
7. React Documentation. React – A JavaScript library for building user interfaces. URL: https://reactjs.org/ .
8. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/ .
9. Chakra UI Documentation. URL: https://chakra-ui.com/ .
10. React Router Documentation. URL: https://reactrouter.com/ .
11. Freeman E., Robson E. Head First Design Patterns. — O'Reilly Media, 2004. — 694 p.
12. Sanderson R. Pro GraphQL. — Apress, 2020. — 250 p.
13. Stenberg M. Secure API Design and Implementation. — Wiley, 2021. — 300 p.
14. Hiebert S., Mann C. Fullstack React: The Complete Guide to ReactJS and Friends. — Fullstack.io, 2017. — 520 p.
15. Norvig P. Artificial Intelligence: A Modern Approach. — Prentice Hall, 2010. — 1152 p.
16. Taft C. Single Page Applications: Building Modern Web Apps with React and Redux. — Packt Publishing, 2019. — 340 p.

17. Microsoft. ASP.NET Core Security Best Practices. URL: https://learn.microsoft.com/en-us/aspnet/core/security/
18. Mozilla Developer Network. Web Security Guidelines. URL: https://developer.mozilla.org/en-US/docs/Web/Security
19. Zakas N.C. Maintainable JavaScript: Writing Readable Code. — O'Reilly Media, 2012. — 120 p.
20. Bogdanov A. Практичний посібник з розробки інформаційних систем. — К.: Видавничий центр «Академія», 2018. — 400 с.
21. Smith J., Jones A. Effective State Management with Redux Toolkit. — Manning, 2021. — 270 p.
22. Pivotal Software. JWT Handbook. URL: https://jwt.io/introduction/.
23. W3C. Web Accessibility Initiative (WAI). URL: https://www.w3.org/WAI/ .
24. Rozum L. Архітектура сучасних інформаційних систем. — Львів: Видавництво ЛНУ, 2020. — 280 с.
25. Rubin H., Anaya M. Testing React Applications: Best Practices. — Packt Publishing, 2022. — 300 p.
26. Abramov D., Clark A. The Redux Docs. URL: https://redux.js.org/introduction/getting-started .
27. Node.js Documentation. URL: https://nodejs.org/en/docs/ .
28. Morgan S. Principles of Secure Software Development. — Wiley, 2019. — 350 p.
29. Welling L., Thomson L. PHP and MySQL Web Development. — Addison-Wesley, 2016. — 660 p.
30. Kurniawan B. React and Redux: Modern Front-End Development. — Apress, 2021. — 320 p.
31. Bostock M. D3.js Data Visualization Fundamentals. — O'Reilly Media, 2017. — 400 p.
32. Croll P., Yoskovitz B. Lean Analytics: Use Data to Build a Better Startup Faster. — O'Reilly Media, 2013. — 220 p.

33. Burns B., Grant B. Kubernetes: Up and Running. — O'Reilly Media, 2018. — 200 p.
34. Bass L., Clements P., Kazman R. Software Architecture in Practice. — Addison-Wesley, 2012. — 560 p.
35. Fitzpatrick B. The Art of Agile Development. — O'Reilly Media, 2010. — 300 p.
36. Eichinger R. Responsive Web Design with Chakra UI. — Packt Publishing, 2022. — 250 p.
37. Meyer B. Object-Oriented Software Construction. — Prentice Hall, 2000. — 900 p.
38. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. — Addison-Wesley, 2010. — 480 p.
39. Lowdermilk T. User-Centered Design: A Developer’s Guide to Building User-Friendly Applications. — Addison-Wesley, 2013. — 300 p.