МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ СПОРТИВНИХ ЗМАГАНЬ»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.С. Горбатко
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

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)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Горбатку Олександру Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи для організації та проведення спортивних змагань

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Визначення вимог до системи.
2. Проєктування системи.
3. Програмна реалізація інформаційної системи.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 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.	19.12.2025	
2.	Збір вихідних даних.	16.01.2026	
3.	Обробка методичних та теоретичних	09.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	23.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	15.06.2026	

Студент _____ О.С. Горбатко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи для організації та проведення спортивних змагань»: 65 с., 18 рис., 8 табл., 26 джерел, 1 додаток.

ВЕБДОДАТОК, ІНФОРМАЦІЙНА СИСТЕМА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ОНЛАЙН-ОПЛАТА, ОРГАНІЗАЦІЯ ЗМАГАНЬ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СПОРТИВНІ ЗМАГАННЯ, ТУРНІРНІ СІТКИ, УПРАВЛІННЯ ТУРНІРАМИ.

Об’єкт дослідження – процеси організації та проведення спортивних змагань у клубах і федераціях, зокрема реєстрація учасників, обробка заявок, формування турнірних сіток, ведення результатів і контроль оплати участі.

Мета роботи – розробка та дослідження інформаційної системи для автоматизації повного циклу організації спортивних змагань, реєстрації користувачів, управління клубами, генерації турнірних сіток, фіксації результатів поєдинків і контролю оплати участі.

Методи дослідження – аналіз наукової літератури та предметної області, системний аналіз, об’єктно-орієнтоване та UML-моделювання, проєктування реляційної бази даних, програмна реалізація клієнтської та серверної частини, інтеграція платіжного сервісу Stripe, перевірка функціональності системи.

Основні результати роботи. Проаналізовано предметну область і визначено вимоги до системи управління змаганнями. Побудовано UML-моделі основних бізнес-процесів. Спроєктовано реляційну базу даних, модулі аутентифікації, управління клубами й заявками, адміністрування змагань, генерації турнірних сіток і ведення результат. Інтегровано онлайн-оплату через Stripe.

Проведено перевірку роботи системи яка може бути використане спортивними організаціями для зменшення ручного навантаження на організаторів і підвищення прозорості проведення турнірів.

SUMMARY

Bachelor’s qualifying paper “Development of an Information System for Organizing and Managing Sports Competitions”: 65 pages, 18 figures, 8 tables, 26 references, 1 supplement.

CLIENT-SERVER ARCHITECTURE, INFORMATION SYSTEM, ONLINE PAYMENT, SOFTWARE DEVELOPMENT SECURITY, SPORTS ZMAGAN, TOURNAMENT MANAGEMENT, TOURNAMENTS, WEB APPLICATION, ZMAGAN ORGANIZATION.

Object of the study is the processes of organizing and holding sports competitions in clubs and federations, in particular, registering participants, processing applications, forming tournament grids, maintaining results and controlling participation fees.

The purpose of the work is to develop and study an information system for automating the full cycle of organizing sports competitions, registering users, managing clubs, generating tournament grids, recording match results and controlling participation fees.

Research methods include analysis of scientific literature and the subject area, system analysis, object-oriented and UML modeling, design of a relational database, software implementation of the client and server parts, integration of the Stripe payment service, testing the functionality of the system.

Main results of the work. The subject area was analyzed and the requirements for the competition management system were determined. UML models of the main business processes were built. A relational database, authentication modules, club and application management, competition administration, generating tournament grids and maintaining results were designed. Integrated online payment via Stripe.

A system has been tested that can be used by sports organizations to reduce the manual workload on organizers and increase transparency in tournament holding.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Визначення вимог до системи 11
1.1 Аналіз предметної області та існуючих рішень..... 11
1.2 Опис основних бізнес-процесів..... 12
1.2.1 Процес реєстрації та автентифікації користувачів..... 13
1.2.2 Процес формування складу учасників та валідації..... 14
1.2.3 Робота з клубами та заявками..... 15
1.2.4 Генерація та ведення сітки змагань 15
1.2.5 Фінансовий процес та оплата 16
1.2.6 Проблеми предметної області, які вирішує система..... 16
1.3 Вимоги до системи (функціональні та нефункціональні) 17
1.3.1 Функціональні вимоги..... 17
1.3.2 Нефункціональні вимоги..... 19
2 Проєктування системи..... 21
2.1 Діаграми прецедентів 21
2.2 Діаграми класів предметної області..... 26
2.3 Діаграма послідовності..... 31
2.4 Діаграма діяльності..... 32
2.5 Діаграма компонентів..... 34
2.6 Проєктування бази даних..... 35
2.6.1 Детальний текстовий опис основних сутностей та зв’язків.... 36
2.6.2 Забезпечення цілісності даних 38
2.7 Діаграма розгортання 39
3 Програмна реалізація інформаційної системи 41

3.1 Вибір середовищ розробки та інструментів 41
3.2 Розробка клієнтської частини 42
3.2.1 Структура клієнтського застосунку 42
3.2.2 Приклад компонента авторизації 44
3.3 Розробка серверної частини 46
3.4 Інтеграція з платіжною системою 49
3.5 Організація тестування 50
Висновки 52
Перелік посилань 54
Додаток А Реалізація Front-End частини 57

ВСТУП

У сучасному світі цифровізація є невід’ємною складовою ефективного управління спортивними заходами. Організація змагань, зокрема в рамках спортивних шкіл, клубів і федерацій, вимагає значних адміністративних витрат, пов’язаних із реєстрацією учасників, перевіркою їхніх даних (вік, вага, контактні телефони, страхові поліси), формуванням турнірних сіток та контролем оплати участі. Традиційні методи управління, що спираються на паперову документацію або розрізнені електронні таблиці, є неефективними, підвищують ризик помилок і ускладнюють комунікацію між тренерами, спортсменами та адміністраторами.

Стрімкий розвиток інформаційних і вебтехнологій відкриває можливості для автоматизації цих процесів. Впровадження спеціалізованих вебсистем дозволяє зменшити ручне навантаження на організаторів, забезпечити прозорість етапів підготовки та проведення турнірів і створити зручний цифровий простір для взаємодії учасників, тренерів та суддівського персоналу. Водночас на практиці організатори часто не мають єдиного інструменту, який би поєднував управління клубами, обробку заявок, формування турнірних сіток і фінансові операції. Це обмежує ефективність проведення змагань і знижує якість адміністративного супроводу. Тому розробка комплексної інформаційної системи, що охоплює зазначені функції, є актуальною та має високу практичну цінність.

Метою кваліфікаційної роботи є розробка та дослідження інформаційної системи для автоматизації повного циклу організації та проведення спортивних змагань від реєстрації користувачів до генерації сіток боїв, фіксації результатів і контролю оплати участі.

Для досягнення поставленої мети мені необхідно виконати такі **завдання**:

– провести аналіз предметної області, існуючих підходів до автоматизації спортивних змагань та сформулювати функціональні й

нефункціональні вимоги до інформаційної системи управління змаганнями;

– розробити архітектуру клієнт-серверного вебзастосунку з розмежуванням прав доступу за ролями, побудувати UML-моделі системи та спроєктувати реляційну базу даних для зберігання інформації про користувачів, клуби, змагання, учасників, заявки та поєдинки;
– реалізувати програмну систему spas-tournament, включаючи модулі реєстрації та автентифікації, управління клубами й заявками, адміністрування змагань і учасників, генерації турнірних сіток, ведення результатів та інтеграції онлайн-оплати участі;
– провести перевірку функціональності розробленої системи та оцінити її практичну придатність для автоматизації роботи спортивних організацій.

Об’єктом роботи є процеси організації та проведення спортивних змагань у спортивних школах, клубах і федераціях (управління учасниками, заявками, турнірними сітками та супровідними адміністративними операціями).

Предметом розробки є методи, моделі та програмні засоби автоматизації зазначених процесів на основі клієнт-серверної вебархітектури з розмежуванням прав доступу за ролями (учасник, тренер, адміністратор).

Методи проєктування є аналіз наукової літератури та предметної області, системний аналіз і формалізація функціональних вимог до системи, об’єктно-орієнтоване та UML-моделювання (діаграми варіантів використання, класів, компонентів, послідовностей, діяльності), проєктування реляційної бази даних, програмна реалізація клієнтської (React, Redux) та серверної (Node.js, Express, Prisma, PostgreSQL) частин, інтеграційне тестування функціональності системи та оцінка отриманих результатів.

Структурно кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань і додатка. У першому розділі проведено аналіз

предметної області та постановку завдання. У другому розділі описано процес проєктування архітектури системи, бази даних і користувацького інтерфейсу. У третьому розділі наведено опис реалізації вебзастосунку, тестування функціональності та оцінку ефективності розробленої системи. У додатку наведено фрагменти коду Front-End частини.

1 ВИЗНАЧЕННЯ ВИМОГ ДО СИСТЕМИ

У цьому розділі проводиться детальне дослідження сфери, для якої створюється система, а також обґрунтовую технологічні рішення. У ньому розглядаються особливості організації змагань, аналізуються існуючі рішення та їхні недоліки, а також пояснюється, чому для розробки обрано саме такий технологічний стек. першого підрозділу.

1.1 Аналіз предметної області та існуючих рішень

Предметною областю даного проєкту є автоматизація організації та проведення спортивних змагань, зокрема змагань із єдиноборств та бойових мистецтв. Ключовою особливістю цієї предметної області є висока динамічність процесів та необхідність суворого дотримання регламенту. Організація турнірів охоплює взаємопов’язані організаційні, спортивні та фінансові процеси, які повинні виконуватися узгоджено в часі з дотриманням правил доступу до інформації [19].

У традиційному (паперовому або табличному) форматі ці процеси часто ведуться розрізнено окремо обліковуються списки учасників, окремо заявки клубів, турнірні сітки та фінансові операції [14, 15, 19]. Такий підхід призводить до дублювання даних, високого ризику людської помилки, затримок в обробці інформації та низької прозорості для учасників і глядачів.

Для вирішення цих проблем розробляється комплексна інформаційна система spas-tournament, яка об’єднує всі процеси в єдиний цифровий простір.

У рамках системи виділено три основні ролі з відповідним набором прав та обов’язків (див. рис. 1.1).

Учасник (client) – реєстрація та вхід до особистого кабінету, перегляд доступних спортивних змагань, подача заявки на участь у змаганнях,

редагування особистого профілю в межах встановлених обмежень, оплата реєстраційного внеску.

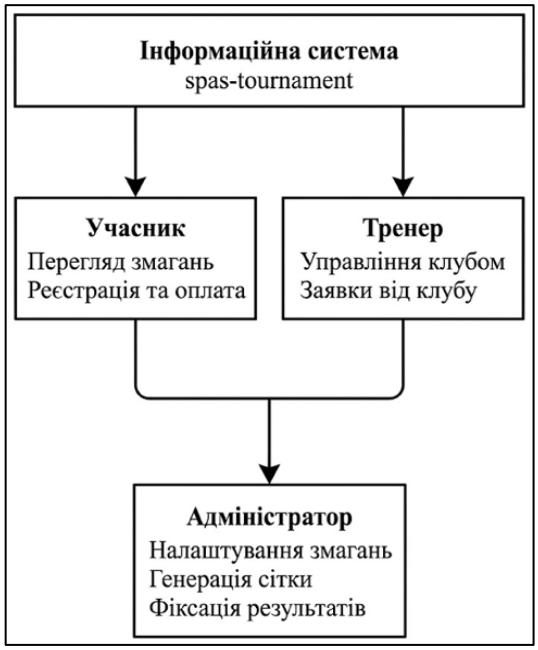


Рисунок 1.1 – Предметні області існуючих рішень

Тренер (coach) – створення та управління власним клубом, запрошення спортсменів до клубу, подача заявок від імені клубу на участь у змаганнях, контроль складу клубу, робота з учасниками та можливість відмітки оплати внесків готівко.

Адміністратор (admin) – створення та налаштування нових змагань, управління користувачами та їхніми ролями, розгляд (затвердження або відхилення) заявок на участь, генерація та перегенерація турнірних сіток, фіксація результатів матчів, контроль фінансових надходжень та аудит історії змін.

1.2 Опис основних бізнес-процесів

Ефективність функціонування будь-якої інформаційної системи визначається тим, наскільки точно вона відображає процеси реального світу.

Організація спортивних змагань, зокрема в сегменті єдиноборств та бойових мистецтв, є комплексною процедурою, яка вимагає високого рівня координації між багатьма учасниками від самих спортсменів та їхніх тренерів до організаторів та суддівської колегії.

У рамках розробки системи spas-tournament було проведено детальне моделювання бізнес-процесів. Нижче наведено детальний опис кожного з етапів, специфікацію взаємодії користувачів та розбір проблем, які виникають під час їхнього виконання традиційними методами.

1.2.1 Процес реєстрації та автентифікації користувачів

Первинним етапом взаємодії користувача з платформою є створення облікового запису. В умовах сучасних вимог до інформаційної безпеки цей процес має бути стандартизованим, але водночас гнучким для підтримки різних ролей.

Система підтримує три ключові актори, кожен з яких має власний життєвий цикл в системі (див. рис. 1.2).

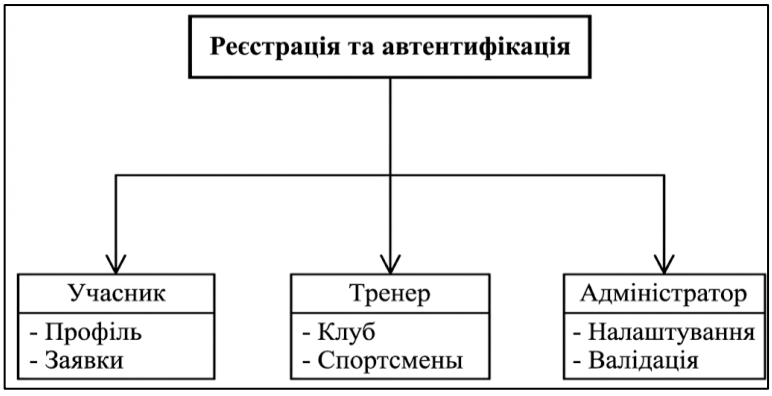


Рисунок 1.2 – Процес реєстрації та автентифікації користувачів

Учасник (client): виступає основним суб’єктом змагань. Процес реєстрації включає введення персональних даних (ПІБ, дата народження, вагова категорія, контактні дані) та створення облікових даних (логін/пароль).

Учасник може переглядати список доступних змагань, подавати заявки, редагувати свій профіль (в межах, які не впливають на допуск до змагань, наприклад, зміна пароля або контактного телефону).

Тренер (coach): відповідає за адміністрування групи спортсменів. Тренер створює клуб, додає до нього учасників, формує командні заявки. Цей актор виконує роль посередника між організатором та спортсменом. Важливою функцією є можливість відмітки оплати внесків готівкою у визначених випадках, що фіксується в системі з обов'язковим зазначенням особи, яка підтвердила операцію.

Адміністратор (admin): має найвищий рівень доступу до системи. Адміністратор створює змагання, налаштовує їхні параметри, затверджує або відхиляє заявки, згенеровані тренерами, ініціює створення турнірних сіток та фіксує підсумкові результати матчів.

1.2.2 Процес формування складу учасників та валідації

Після того, як учасник або тренер подає заявку, починається процес перевірки (валідації) даних. У спортивних змаганнях з єдиноборств цей процес є критично важливим для забезпечення безпеки та чесної конкуренції.

Бізнес-процес валідації включає, перевірка віку, система автоматично порівнює дату народження учасника з регламентом змагань. Перевірка вагової категорії спортсмен розподіляється у відповідну вагову категорію, що виключає участь у невідповідних змаганнях. Обробка статусу заявки заявки може знаходитися в одному з трьох станів Pending (на розгляді), Approved (затверджено), Rejected (відхилено).

У традиційному підході перевірка здійснюється вручну, що вимагає великих витрат часу та призводить до помилок. Автоматизація цього процесу через spas-tournament дозволяє виключити людський фактор на етапі перевірки формальних критеріїв.

1.2.3 Робота з клубами та заявками

Управління заявками є процесом, що забезпечує організацію турнірів великого масштабу. Тренер формує клуб, додаючи до нього спортсменів. Опишемо, як виглядає цей процес.

Створення клубу: тренер реєструє команду, вказує її назву, регіон та контактну інформацію.

Запрошення спортсменів: тренер надсилає запит на приєднання або додає профілі, які вже є в базі даних.

Подача групової заявки: замість того, щоб кожен учасник реєструвався окремо, тренер може подати заявку на змагання одразу для кількох бійців.

Розгляд адміністратором: адміністратор бачить єдиний список заявок від клубу. Він може схвалити всю заявку або відхилити окремого учасника через невідповідність вимогам.

1.2.4 Генерація та ведення сітки змагань

Генерація турнірної сітки – це один із найскладніших етапів з організаційної точки зору. У традиційному спорті помилка при складанні сітки може призвести до скасування результатів змагань [14, 15].

У системі spas-tournament цей процес автоматизовано за деякими правилами, розглянемо їх.

Класичні сітки для змагань на вибування (олімпійська система) створюються пари з урахуванням посіву.

Кругова система для специфічних випадків, наприклад, при участі рівно трьох спортсменів, застосовується кругова логіка (Round Robin), де кожен боєць зустрічається з кожним. Продвиження переможців після завершення матчу адміністратор фіксує результат, і система автоматично просуває переможця до наступного етапу сітки.

1.2.5 Фінансовий процес та оплата

Оплата участі у змаганнях – це процес, який часто потребує синхронізації між різними платіжними системами. У системі реалізовано два шляхи здійснення цього процесу.

Онлайн-оплата (Stripe) тренер або учасник обирає змагання та переходить до оплати [7]. Бекенд генерує PaymentIntent через Stripe API. Після успішної оплати Stripe надсилає webhook-подію, яка змінює статус заявки та фіксує платіж у таблиці PaymentLog.

Готівковий розрахунок у разі командної оплати або спеціальних умов тренер може внести кошти готівкою на місці. Адміністратор або тренер із відповідними правами вручну відмічає статус оплати в системі, супроводжуючи це коментарем.

1.2.6 Проблеми предметної області, які вирішує система

Аналіз існуючих підходів дозволяє виділити основні проблеми, які вдалося вирішити за допомогою розробки системи spas-tournament.

Дублювання та розрізненість даних: у паперовому вигляді інформація часто зберігається в різних таблицях і документах, що призводить до втрати актуальності. Система забезпечує єдиний джерело істини (Single Source of Truth).

Складність обліку оплат: організаторам важко відстежувати, хто з учасників вже сплатив внесок. Автоматизація через webhook від Stripe знижує ризик фінансових помилок.

Відсутність історії (аудиту): у традиційному форматі неможливо відстежити, хто і коли змінив результат або статус учасника. Система логує всі важливі зміни, забезпечуючи повну прозорість для організаторів та учасників.

Людський фактор при створенні сіток, автоматична генерація пар та кругових турнірів виключає ймовірність помилок під час розподілу бійців.

1.3 Вимоги до системи (функціональні та нефункціональні)

Перед початком проєктування архітектури системи було сформовано детальний перелік вимог, що визначають її поведінку, продуктивність та безпеку [20]. Вимоги поділено на функціональні та нефункціональні.

1.3.1 Функціональні вимоги

Функціональні вимоги визначають конкретний спектр завдань, бізнес-правил та сценаріїв використання, які програмний продукт має підтримувати для забезпечення коректної роботи всіх категорій користувачів (акторів). Для досягнення системності декомпозицію функціональної специфікації виконано за відповідними профільними підсистемами.

Підсистема управління користувачами (Auth & Users).

Реєстрація користувачів: забезпечення можливості створення нових облікових записів у системі з обов'язковою валідацією вхідних даних на стороні сервера (перевірка коректності формату електронної пошти, а також довжини і складності пароля) та блокуванням некоректних запитів.

Автентифікація: ідентифікація та автентифікація користувачів із використанням механізму збереження стану сесії на базі токенів JWT (JSON Web Token), що забезпечує генерацію паролізованих токенів доступу (Access/Refresh) для безпечного обміну даними між клієнтською частиною та сервером.

Рольова модель доступу: розмежування прав доступу до ресурсів та операцій системи на основі функціоналу RBAC (Role-Based Access Control). Визначено три основні ролі: admin (повне адміністрування системи), coach (управління спортивним клубом та заявками) та client (спортсмен, з правами перегляду й редагування особистих даних).

Управління профілями: надання можливості модифікації та верифікації персональних даних користувачів (ПІБ, контактна інформація, поточна вагова

категорія) з упровадженням обмеження на редагування після затвердження заявки на змагання.

Підсистема управління клубами та заявками (Clubs & Applications).

Створення клубу: забезпечення тренеру (coach) можливості реєстрації та адміністрування віртуального профілю спортивного клубу із внесенням до бази даних його назви, географічного регіону приналежності та графічної емблеми.

Формування складу: можливість динамічного додавання спортсменів до реєстру конкретного клубу, а також перегляду та фільтрації списку учасників відповідальним тренером.

Управління заявками: реалізація механізму пакетного формування та менеджменту життєвого циклу заявок на участь у змаганнях. Тренер має змогу подавати спільну заявку на групу бійців, а адміністратор системи – змінювати її поточний статус (Pending, Approved або Rejected).

Підсистема проведення змагань та турнірних сіток (Brackets & Matches).

Налаштування турніру: можливість створення й конфігурування картки спортивного турніру адміністратором із визначенням ключових метаданих: дати проведення, регламенту, переліку доступних вагових категорій та фінансових реєстраційних внесків.

Генерація турнірних сіток: автоматизована побудова і розподіл зареєстрованих учасників за парами на основі олімпійської системи (Single Elimination) або за круговою схемою (Round Robin) у випадку наявності трьох спортсменів у ваговій категорії.

Фіксація результатів поєдинків: інструментарій для внесення рахунків та результатів за раундами з подальшим автоматичним просуванням спортсмена-переможця до наступного етапу турнірної сітки.

Платіжна підсистема (Payments).

Інтеграція Stripe API: підключення зовнішнього шлюзу Stripe для безпечного проведення електронних платежів під час оплати реєстраційних внесків за допомогою інструменту PaymentIntent.

Обробка вебхуків: асинхронне перехоплення та верифікація повідомлень від платіжного сервісу за допомогою механізму вебхуків (Webhooks) для підтвердження успішності транзакції та автоматичного оновлення статусу відповідної заявки в системі.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають системні обмеження, архітектурні стандарти та критерії якості, які гарантують високий рівень надійності, захищеності та масштабованості розроблюваного програмного забезпечення. Ключові нефункціональні вимоги розподілено за категоріями.

Вимоги до безпеки та захисту даних (Security).

Криптографічний захист конфіденційних даних користувачів шляхом обов’язкового хешування паролів із використанням стійкого алгоритму bcrypt із адаптивним фактором обчислювальної складності (salt rounds).

Авторизація та захист кінцевих точок (endpoints) серверного API шляхом впровадження проміжного програмного забезпечення (middleware), що здійснює перевірку сигнатури, цілісності та терміну дії JWT-токенів.

Нівелювання ризиків виникнення критичних уразливостей, зокрема захист від ін’єкцій коду (SQL-ін’єкцій) за рахунок абстрагування рівня доступу до бази даних через ORM Prisma (використання параметризованих запитів за замовчуванням) та реалізація XSS-фільтрації вхідного трафіку.

Вимоги до надійності, доступності та цілісності (Reliability & Integrity).

Забезпечення транзакційної цілісності та атомарності комплексних операцій на рівні СКБД [21]. Будь-які пов’язані операції, зокрема одночасна автоматична генерація турнірних сіток та процеси фіксації фінансових платежів, мають виконуватися виключно в межах ізольованих баз даних транзакцій (принцип ACID) для запобігання втрати або розсинхронізації даних у разі виникнення системних збоїв.

Вимоги до продуктивності та ефективності (Performance & Efficiency).

Максимальний час відгуку серверного API (API response time) під час виконання базових операцій створення, читання, оновлення та видалення даних (CRUD) не повинен перевищувати граничного значення у 300–400 мс за нормативного навантаження на систему.

Оптимізація обсягів мережевого трафіку та зниження навантаження на клієнтську частину додатку за рахунок реалізації обов’язкової серверної пагінації та попередньої фільтрації великих масивів даних.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Діаграми прецедентів

Діаграма прецедентів відображає межі системи управління спортивними змаганнями та вказує на основні ролі користувачів (актори) і їхні взаємодії з функціональними блоками. Актор «Учасник» ініціює такі процеси як реєстрація/вхід, редагування профілю, подача заявки в клуб та оплата участі. Актор «Тренер» має ширші повноваження, управління складом учасників клубу та управління самим клубом. Також він може переглядати процеси, пов’язані із заявками. Актор «Адміністратор» має найвищий рівень доступу. Він виконує операції зі створення змагань, керує процесом створення турнірних сіток, ведення результатів поєдинків та контролює фінансові надходження (перегляд платежів). Зовнішній сервіс «Система Stripe» забезпечує безпечну обробку платежів, повертаючи відповіді до підсистеми оплати (див. рис. 2.1).



Рисунок 2.1 – Діаграма варіантів використання

У діаграмі виділено три основні ролі користувачів: учасник (клієнт), тренер та адміністратор. Додатково залучено зовнішню систему Stripe – платіжний сервіс для обробки онлайн-оплати внеску за участь у змаганні. Кожен прецедент описує взаємодію актора з системою для досягнення конкретної цілі, нижче наведено їх детальну характеристику відповідно до реалізованого вебзастосунку.

Прецедент: реєстрація та вхід до системи.

Актори: учасник, тренер, адміністратор.

Користувач відкриває сторінку входу або реєстрації вебзастосунку. При реєстрації він вводить ім’я, електронну пошту, пароль і обирає роль (учасник або тренер, обліковий запис адміністратора створюється окремо). Система надсилає дані на сервер, перевіряє унікальність e-mail, зберігає хеш пароля в базі даних і повертає JWT-токен разом із профілем користувача. При вході користувач вказує e-mail і пароль, сервер звіряє облікові дані та видає токен авторизації. Клієнтська частина зберігає токен і перенаправляє адміністратора в адмін-панель, тренера або учасника – на особистий кабінет. Якщо дані невірні або e-mail уже зайнятий, система відображає повідомлення про помилку без надання доступу. Після успішної авторизації всі подальші запити до API супроводжуються JWT, неавторизований доступ до захищених маршрутів блокується компонентом захисту маршрутів.

Прецедент: перегляд змагань.

Актори: учасник, тренер.

Авторизований користувач переходить до особистого кабінету, де система завантажує з сервера список змагань. Для кожного змагання відображаються назва, дата, тип, статус (майбутнє, поточне, завершене) та розмір вступного внеску. Учасник бачить змагання, до яких може долучитися або в яких уже зареєстрований, тренер використовує цей перелік при формуванні заявок клубу на участь. Перегляд не змінює даних у базі – це інформаційний сценарій, що забезпечує актуальну картину майбутніх і поточних турнірів.

Прецедент: подача заявки на вступ до клубу.

Актори: учасник, тренер.

Учасник може подати заявку на вступ до спортивного клубу двома шляхами. Обрати клуб із доступного списку в інтерфейсі та надіслати заявку – запис потрапляє в чергу на розгляд тренером. Перейти за посиланням або QR-кодом запрошення (/invite/:token), система перевіряє токен, і після входу учасник автоматично приєднується до клубу або отримує інструкції для завершення вступу. До прийняття заявки тренером учасник не вважається повноправним членом клубу для подачі колективних заявок на змагання. При відхиленні заявки учасник отримує відповідний статус у системі.

Прецедент: оплата участі у змаганні.

Актори: учасник, зовнішня система – Stripe.

Учасник у кабінеті бачить свої реєстрації на змагання та статус оплати (не оплачено, оплачено, помилка, звільнено від оплати). Для змагання з ненульовим внеском він ініціює оплату клієнт запитує у сервера створення Payment Intent у Stripe на суму вступного внеску (у копійках, валюта UAH). Сервер перевіряє, що платить сам учасник або адміністратор, що оплата ще не проведена, зберігає ідентифікатор Payment Intent у записі учасника та повертає clientSecret. У модальному вікні Stripe Elements учасник вводить платіжні реквізити, після успішної транзакції Stripe надсилає webhook на сервер, який оновлює статус оплати учасника на «оплачено». Альтернативно тренер або адміністратор може зафіксувати готівкову оплату без Stripe – тоді статус змінюється через окремий API-запит. Якщо змагання безкоштовне або оплата вже здійснена, система відмовляє в повторному створенні платежу.

Прецедент: редагування профілю учасника.

Актори: учасник, тренер.

Учасник відкриває профіль і змінює персональні дані, необхідні для змагань: ім’я, вік, вага, контактні дані, аватар тощо. Зміни зберігаються на сервері та використовуються при розподілі по віково-вагових категоріях і відображенні в списках учасників. Тренер у межах управління клубом може

коригувати вагу окремих членів команди для актуалізації даних перед подачею заявки на турнір. Система не дозволяє стороннім ролям змінювати чужий профіль без відповідних прав доступу.

Прецедент: управління клубом.

Актори: тренер.

Тренер створює спортивний клуб, стаючи його власником. Далі він керує складом: переглядає заявки на вступ, схвалює або відхиляє кандидатів, видаляє членів клубу, редагує їхні параметри (зокрема вагу). Тренер генерує посилання та QR-код запрошення для швидкого набору учасників. Також він формує заявку клубу на участь у змаганні: обирає турнір, відмічає членів клубу, яких реєструє колективно, і надсилає заявку на розгляд адміністратору. Усі операції виконуються через API з перевіркою JWT і ролі coach, учасник без прав тренера до цих функцій доступу не має.

Прецедент: створення та управління змаганнями.

Актори: адміністратор.

Адміністратор у адмін-панелі створює нове змагання, задаючи назву, дату, тип змагання, статус, тип турнірної сітки (олімпійська, подвійна олімпійська, кругова система для трьох учасників у категорії), розмір вступного внеску. Може редагувати параметри існуючого змагання або видаляти його, якщо це допускає бізнес-логіка системи. Створене змагання стає доступним для перегляду учасниками та тренерами і використовується як основа для реєстрації учасників, генерації сітки та обліку платежів.

Прецедент: управління учасниками змагань.

Актори: адміністратор, тренер.

Адміністратор переглядає повний список учасників з фільтрацією за змаганням, статусом оплати, пошуком за іменем. Може додавати учасника вручну, редагувати його дані (вік, вага, клуб, прив’язка до змагання), змінювати статус оплати, видаляти запис. Тренер у своєму кабінеті переглядає учасників, пов’язаних із його клубом або заявками, відкриває детальний профіль, позначає готівкову оплату. Сценарій забезпечує централізований

облік спортсменів на турнірі та контроль готовності до жеребкування (наявність оплати, коректність антропометричних даних).

Прецедент: генерація турнірної сітки боїв.

Актори: адміністратор.

Після реєстрації учасників на змагання адміністратор переходить до розділу турнірної сітки для обраного змагання і запускає генерацію сітки. Сервер отримує список учасників, групує їх за віково-ваговими категоріями, перемішує в межах категорії та будує набір поєдинків відповідно до типу сітки, заданого при створенні змагання: олімпійська система на вильот, подвійна олімпійська (сітка переможців і сітка втішення) або кругова система при трьох учасниках у категорії. Результат зберігається в таблиці поєдинків (Match) з раундами, індексами та мітками категорій. Адміністратор може повторно згенерувати сітку або вручну коригувати розстановку в інтерфейсі редактора сітки (drag-and-drop). Без згенерованої сітки ведення результатів поєдинків неможливе.

Прецедент: ведення результатів поєдинків.

Актори: адміністратор.

Адміністратор на сторінці турнірної сітки переглядає список поєдинків по раундах і категоріях. Для кожного бою він фіксує результат (перемога одного з учасників, нічия за потреби) та переможця. Система оновлює запис поєдинку в базі даних, для олімпійських форматів переможець автоматично просувається в наступний раунд сітки згідно з реалізованою логікою. Сценарій підтримує проведення турніру в реальному часі та формування підсумкової картини змагань без паперових таблиць.

Прецедент: перегляд платежів.

Актори: тренер, адміністратор.

Тренер у розділі учасників бачить статуси оплати по членах клубу та змаганнях: оплачено онлайн, готівкою, очікує оплати, помилка, звільнено. Може ініціювати фіксацію готівкового платежу. Адміністратор має розширений огляд усіх учасників і може змінювати статуси оплати при

необхідності. Детальна інформація про транзакції Stripe зберігається через paymentIntentId і обробку webhook, перегляд у інтерфейсі орієнтований на контроль виконання фінансових умов допуску до змагань. Учасник бачить лише власні платежі в рамках прецеденту оплати (UC-4), тоді як тренер і адміністратор виконують моніторинг у межах цього прецеденту.

2.2 Діаграми класів предметної області

Діаграма класів є ключовою частиною об’єктно-орієнтованого проєктування. Вона визначає статичну структуру системи, показуючи її класи, атрибути, методи та взаємозв’язки між ними (див. табл. 2.1). Для проєктування інформаційної системи spas-tournament було розроблено діаграму класів предметної області, яка описує структуру даних та логіку взаємодії основних сутностей. Детальний опис кожного класу та його ролі в системі (див. рис. 2.2).

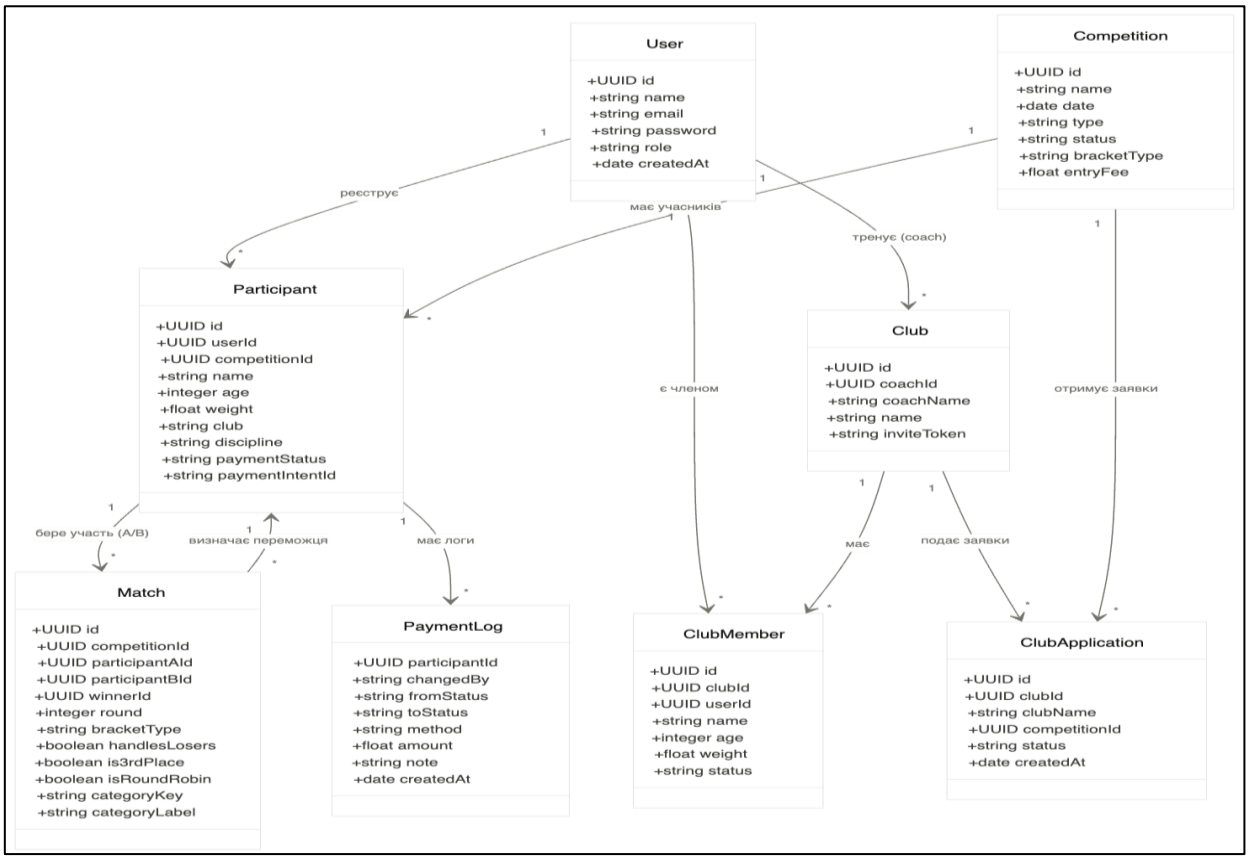


Рисунок 2.2 – Діаграма класів предметної області

Таблиця 2.1 – Клас User

Атрибут	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор користувача (Primary Key)
name	string	Ім’я та прізвище користувача
email	string	Адреса електронної пошти (використовується як логін)
password	string	Хешований пароль для безпеки
role	string	Роль користувача для розмежування доступу (ADMIN, COACH, CLIENT)
createdAt	date	Дата та час створення облікового запису

Цей клас є кореневою сутністю для системи. Він відповідає за глобальний облік користувачів, безпечне збереження авторизаційних даних (паролів) та реалізацію рольової моделі доступу (RBAC). Дані з цієї таблиці використовуються для перевірки прав при спробі створити клуб, подати заявку чи змінити результати матчу (див. табл. 2.2).

Таблиця 2.2 – Клас Participant

Атрибут	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор учасника (Primary Key)
userId	UUID	Зовнішній ключ (Foreign Key), посилання на User
competitionId	UUID	Зовнішній ключ (Foreign Key), посилання на Competition
name	string	Повне ім’я учасника
age	integer	Вік спортсмена (використовується для валідації)
weight	float	Вага спортсмена (для розподілу за ваговими категоріями)
club	string	Назва клубу, який представляє учасник
discipline	string	Спортивна дисципліна (вид єдиноборства)
paymentStatus	string	Поточний статус оплати заявки
paymentIntentId	string	Ідентифікатор платіжної сесії в системі Stripe

Клас описує сутність спортсмена, зареєстрованого на конкретний турнір. Він містить антропометричні дані, які система використовує для автоматичного розподілу по турнірних сітках, а також статус оплати реєстраційного внеску через інтеграцію зі Stripe (див. табл. 2.3).

Таблиця 2.3 – Клас Competition

Атрибут	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор змагання (Primary Key)
name	string	Офіційна назва турніру
date	date	Дата проведення
type	string	Формат або тип змагань
status	string	Поточний стан турніру (заплановано, триває, завершено)
bracket_type	string	Тип турнірної сітки за замовчуванням (олімпійська, кругова)
entryFee	float	Вартість реєстраційного внеску для участі

Сутність для представлення спортивного заходу (турніру, чемпіонату). Вона агрегує в собі всі глобальні правила події: дату проведення, вартість участі, поточний статус (триває чи завершено) та базовий алгоритм побудови турнірних сіток (див. табл. 2.4).

Таблиця 2.4 – Клас Match

Атрибут	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор поєдинку (Primary Key)
competitionId	UUID	Зовнішній ключ (Foreign Key), посилання на Competition
participantAid	UUID	Ідентифікатор першого бійця (Кут А)
participantBid	UUID	Ідентифікатор другого бійця (Кут Б)
winnerId	UUID	Ідентифікатор переможця матчу (може бути null до кінця бою)
round	integer	Номер поточного раунду або етапу сітки
bracket_type	string	Тип сітки для цього конкретного матчу
handlesLosers	boolean	Прапорець для логіки боїв за 3-тє місце / сітки тих, хто програв
isRoundRobin	boolean	Прапорець, який вказує на те, чи є цей етап круговою системою
categoryKey	string	Ключ для ідентифікації категорії (вік/вага)
categoryLabel	string	Текстова мітка категорії

Клас Match безпосередньо керує логікою змагального процесу. Він фіксує пару опонентів (Кут А та Кут Б), номер раунду, тип сітки та ID переможця (див. табл. 2.5).

Таблиця 2.5 – Клас Club

Назва атрибуту	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор клубу (Первинний ключ)
coachId	UUID	Зовнішній ключ, посилання на User (власник-тренер)
coachName	string	Повне ім’я тренера клубу
name	string	Офіційна зареєстрована назва спортивного клубу
inviteToken	string	Унікальний токен/ключ для запрошення нових членів до клубу

Сутність для організації спортивного об’єднання. Вона пов’язує клуб із його творцем (тренером) та генерує унікальний inviteToken (див. табл. 2.6).

Таблиця 2.6 – Клас PaymentLog

Атрибут	**Тип даних**	**Опис**
participantId	UUID	Зовнішній ключ, посилання на Participant
changedBy	string	Ідентифікатор адміна або тренера, який змінив статус платежу вручну
fromStatus	string	Попередній фінансовий статус
toStatus	string	Новий фінансовий статус
method	string	Спосіб оплати (онлайн через Stripe або готівка)
amount	float	Сума операції
note	string	Текстова примітка або коментар до транзакції

Цей клас реалізує склад спортивної команди. Окрім фіксації зв’язку між користувачем та клубом, він дублює базову антропометрію спортсмена, що дозволяє тренеру бачити актуальну вагу та вікову групу своєї команди всередині клубного кабінету (див. табл. 2.7).

Таблиця 2.7 – Клас ClubApplication

Назва атрибуту	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор заяви (Первинний ключ)
clubId	UUID	Зовнішній ключ, посилання на сутність Club

Продовження таблиці 2.7

Назва атрибуту	**Тип даних**	**Опис**
clubName	string	Назва клубу, який подає заявку
competitionId	UUID	Зовнішній ключ, посилання на змагання Competition
status	string	Поточний статус обробки заяви (на розгляді, прийнято)
createdAt	date	Дата та час створення заявки

Сутність для керування командними реєстраціями на турніри. Вона дозволяє тренеру подати єдину заявку від усього клубу на участь у змаганні, яка згодом перевіряється та затверджується організаторами змагань (див. табл. 2.8).

Таблиця 2.8 – Клас ClubMember

Назва атрибуту	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор запису членства (Первинний ключ)
clubId	UUID	Зовнішній ключ, посилання на сутність Club
userId	UUID	Зовнішній ключ, посилання на сутність User
name	string	Повне ім’я члена клубу
age	integer	Вік спортсмена всередині клубу
weight	float	Поточна вага спортсмена всередині клубу
status	string	Поточний статус спортсмена в клубі (активний, на розгляді)

Сутність для ведення фінансового аудиту. Вона логує будь-які зміни платіжних статусів учасників (наприклад, перехід від «не оплачено» до «оплачено готівкою»), фіксуючи відповідального адміністратора та суму операції, що унеможливлює фінансові махінації під час реєстрації.

Описана структура класів формує цілісну й масштабовану концептуальну модель даних системи. Чіткий розподіл сутностей на рівні бази даних забезпечує високу швидкість обробки запитів, мінімізує надмірність даних (дублювання інформації) та гарантує цілісність зв’язків між модулями.

Розроблена специфікація є фундаментальною основою для подальшого етапу проєктування – створення фізичної схеми бази даних (на базі СУБД PostgreSQL) та безпосередньої реалізації об’єктно-реляційного відображення (ORM) на рівні програмного коду бекенд-частини застосунку.

2.3 Діаграма послідовності

Для моделювання послідовності викликів, обміну повідомленнями та хронології виконання бізнес-процесів між компонентами інформаційної системи spas-tournament було розроблено діаграми послідовності (Sequence Diagrams). Ці діаграми дають змогу наочно простежити життєвий цикл запиту, починаючи від дій користувача в інтерфейсі та закінчуючи збереженням даних у базі та отриманням відповідей від зовнішніх API. Розглянемо детально процес генерації та оплати участі в змаганнях, який є критично важливим для забезпечення фінансової прозорості та безперебійності роботи турніру (див. рис. 2.3).

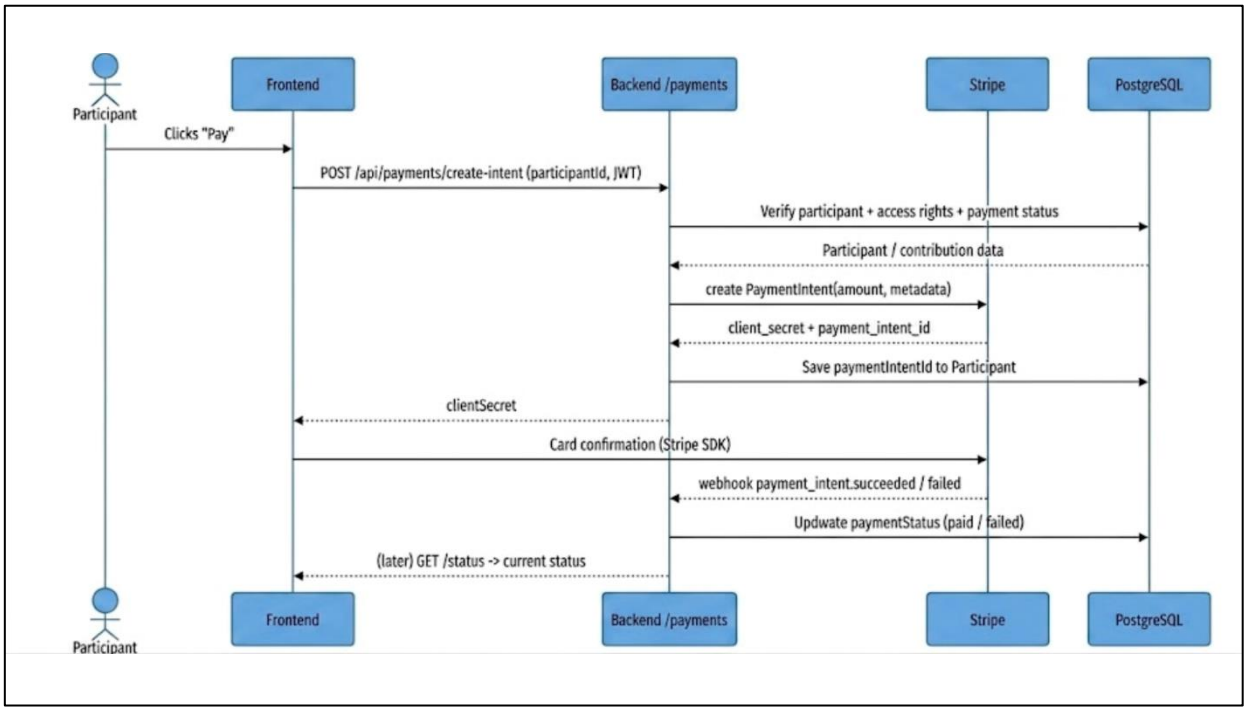


Рисунок 2.3 – Діаграма послідовності

Оплата участі: процес взаємодії компонентів під час проведення платежу поділяється на такі послідовні етапи.

Ініціалізація запиту (клієнтська частина / Frontend): учасник змагань або тренер у вебдодатку обирає подію та натискає кнопку «Оплатити». Фронтенд-компонент формує запит та відправляє його через захищений протокол HTTPS до бекенд-контролера платежів. Перевірка на рівні бекенду (Backend / API). Отримавши запит, бекенд-сервер через middleware проводить автентифікацію користувача, перевіряючи валідність JWT-токена. Система виконує валідацію статусу учасника (перевіряє, чи не заблоковано користувача та чи відповідає його заявка вимогам).

Взаємодія із зовнішнім платіжним шлюзом (Stripe API): бекенд генерує запит на створення наміру оплати (PaymentIntent), вказуючи суму збору та валюту. Запит передається до системи Stripe, яка обробляє дані та повертає статус створення сесії разом із захищеним client_secret. Бекенд-сервіс обробляє відповідь, ініціює збереження paymentIntentId через Prisma ORM до бази даних PostgreSQL для подальшої верифікації.

Завершення транзакції та оновлення інтерфейсу: бекенд передає результати операції на клієнтську частину. Користувач отримує сповіщення про успішний перехід до платіжної форми, а статус заявки змінюється на очікування (Pending). Після підтвердження вебхуком (Webhook) статус змінюється на Approved.

2.4 Діаграма діяльності

Для формалізації та наочної ілюстрації алгоритмів прийняття рішень, розгалуження бізнес-логіки та послідовності виконання операцій під час адміністрування змагань було розроблено діаграму діяльності (Activity Diagram). Ця діаграма відображає переходи між станами системи, починаючи від ініціації змагання та закінчуючи визначенням підсумкових результатів.

На діаграмі детально показано перевірки прав доступу, валідацію статусів учасників перед генерацією матчів, а також процес обробки результатів поєдинків (див. рис. 2.4).

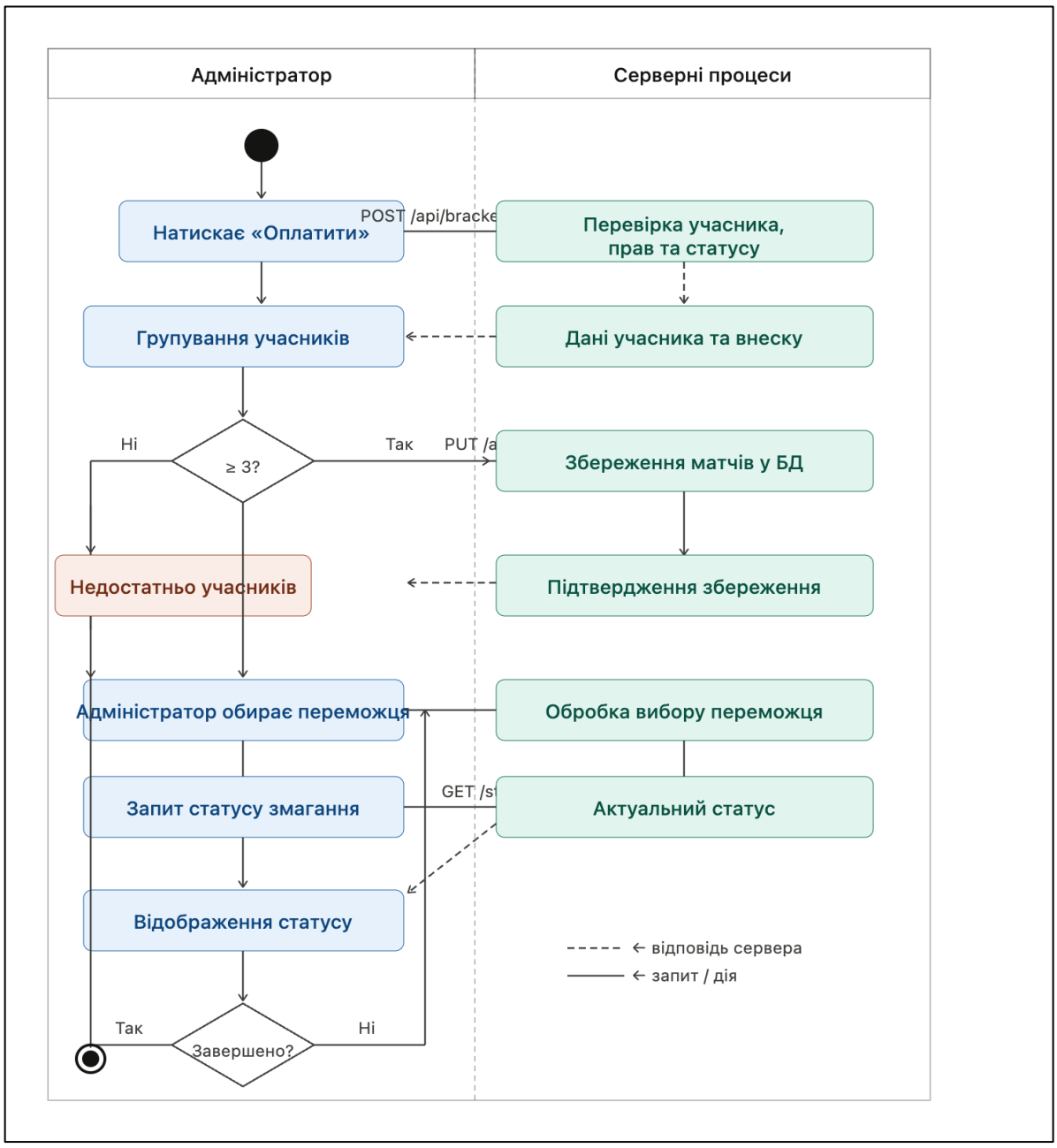


Рисунок 2.4 – Діаграма діяльності

Генерація та ведення турнірної сітки – процес, представлений на діаграмі діяльності, можна поділити на кілька ключових етапів.

Ініціалізація процесу: адміністратор авторизується в системі та переходить до розділу управління змаганнями. Система виконує перевірку

ролі користувача через Auth Middleware. Якщо роль не відповідає рівню ADMIN, процес переривається.

Групування та валідація учасників: система виконує запит до бази даних (через Prisma ORM) для отримання списку затверджених заявок (статус Approved). Відбувається етап групування учасників за ваговими та віковими категоріями.

Точка розгалуження (Decision Node): якщо кількість учасників у певній категорії дорівнює трьом, система автоматично перемикається на алгоритм кругової системи (Round Robin), де кожен боєць зустрічається з кожним. В інших випадках створюється класична сітка змагань на вибування.

Генерація матчів та проведення поєдинків: формується набір записів у таблиці Matches. Дані передаються на клієнтську частину. Після завершення поєдинку адміністратор вносить або підтверджує результат.

Вибір переможця та оновлення даних: система перевіряє статус поєдинку та фіксує ідентифікатор переможця (winnerId). Переможець автоматично переходить до наступного раунду (зміна поля round_number та формування нової пари). Процес завершується (або переходить до фінального етапу), після чого зберігаються зміни в базі даних та формується актуальний статус турніру.

Така деталізація логіки дозволяє уникнути конфліктів при формуванні пар і забезпечує цілісність даних на всіх етапах проведення спортивного турніру.

2.5 Діаграма компонентів

Діаграма компонентів є важливою частиною проєктування фізичної архітектури інформаційної системи spas-tournament (див. рис. 2.5).

Діаграма ілюструє модульну структуру додатку, фізичне пакування коду, залежності між програмними модулями, а також взаємодію між внутрішніми компонентами та зовнішніми сервісами.

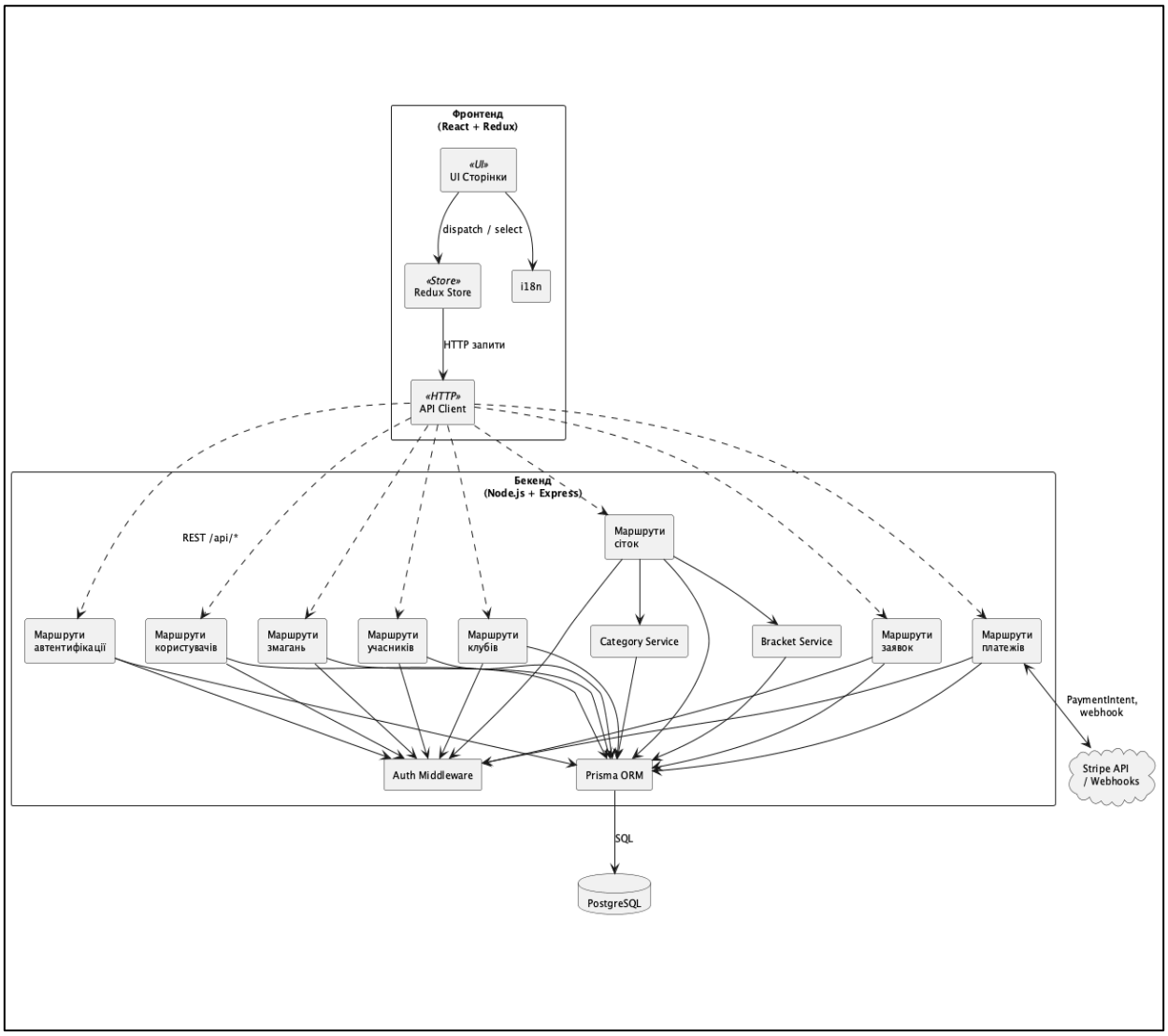


Рисунок 2.5 – Діаграма компонентів

2.6 Проєктування бази даних

Для забезпечення надійного зберігання, високої швидкості доступу та запобігання дублюванню інформації було розроблено реляційну базу даних. Проєктування схеми бази даних базувалося на принципах нормалізації до третьої нормальної форми (3NF) [11], що гарантує цілісність даних, мінімізує надмірність та підтримує складні транзакційні процеси, пов’язані з організацією змагань та фінансовими розрахунками.

Для візуалізації структури зв’язків між таблицями бази даних була наведена діаграма сутностей (див. рис. 2.6).

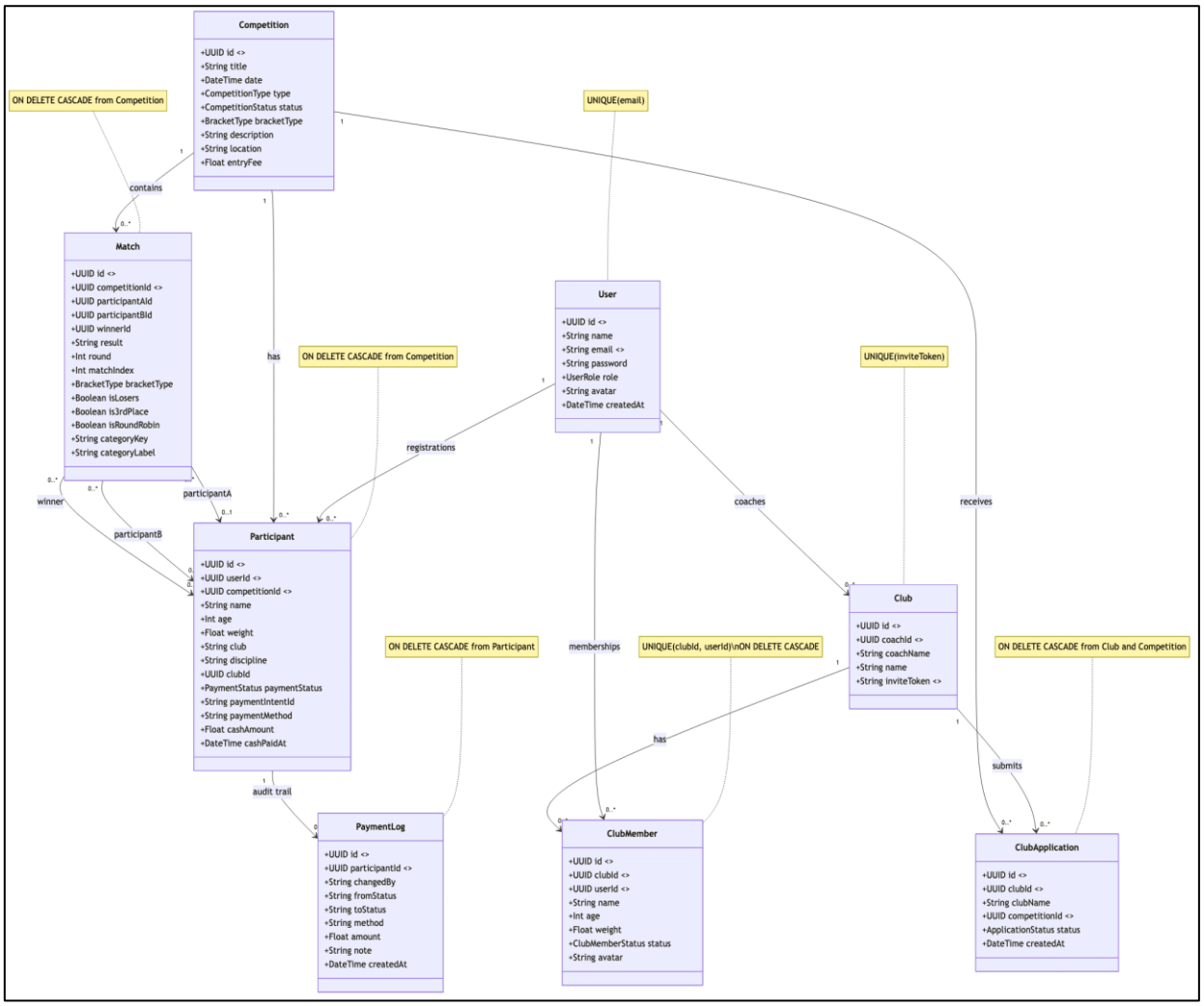


Рисунок 2.6 – Діаграма доменної моделі

2.6.1 Детальний текстовий опис основних сутностей та зв’язків

Нижче наведено детальний опис кожної таблиці бази даних, її полів, типів даних та логіки зв’язків між сутностями.

Базовим елементом архітектуры всієї системи є таблиця Users, яка акумулює облікові дані для проведення автентифікації та динамічного визначення прав доступу на основі рольової моделі. Централізоване керування доступом реалізовано завдяки збереженню даних усіх користувачів в одній сутності, незалежно від їхнього фінального статусу. Унікальний ідентифікатор користувача фіксується у полі id. Електронна пошта користувача зберігається у полі email та виступає унікальним логіном, що унеможливлює повторну

реєстрацію. За безпеку авторизації відповідає поле password_hash, де утримується хеш пароля, згенерований алгоритмом bcrypt. Системна роль, яка визначає рівень доступу до захищених API-маршрутів, задається через поле role зі значеннями адміністратора, тренера або клієнта. Персональні установчі дані представлені полями first_name та last_name, а точний момент створення облікового запису автоматично реєструється полем created_at.

Для збереження специфічних персональних даних безпосередніх учасників змагань призначена таблиця Profiles. Між нею та базовою таблицею Users реалізовано зв’язок типу один-до-одного, де ідентифікатор user_id є одночасно первинним та зовнішнім ключем, що посилається на таблицю користувачів. Це гарантує, що кожен профіль належить лише одному унікальному користувачу. Внутрішня структура таблиці містить поле date_of_birth, необхідне для автоматичної валідації та розподілу спортсменів за віковими групами, поле weight, яке використовується для вагового контролю та сортування за категоріями, а також поле phone для ведення прямого контактного зв’язку.

Менеджмент спортивних об’єднань покладено на таблицю Clubs, яка зберігає офіційну інформацію про команди, що заявляються на турніри. Сутність має логічний зв’язок типу один-до-багатьох з користувачами, які мають роль тренера, оскільки один керівник може виступати засновником та керувати декількома організаціями. Кожен запис клубу містить унікальний ідентифікатор id, офіційну текстову назву клубу у полі name, регіон або місто базування у полі region, а також зовнішній ключ coach_id, який безпосередньо зв’язує клуб із профілем відповідального тренера.

Глобальні параметри змагань, що створюються та моделюються адміністраторами системи, зосереджені в таблиці Tournaments. Вона виступає головним інформаційним вузлом для проведення реєстрацій та містить унікальний ключ поєдинку id. Назва спортивного заходу фіксується у полі title. Часові рамки проведення турніру чітко лімітуються полями start_date та end_date. Поточний стан життєвого циклу події контролюється через поле

status, яке за замовчуванням перебуває в режимі очікування і може приймати значення підготовки, активної фази або завершення. Фінансова складова, а саме сума обов’язкового реєстраційного внеску для спортсменів, визначається полем entry_fee.

Для генерації та динамічного керування інтерактивними турнірними сітками розроблена таблиця Matches, яка зв’язує профілі спортсменів у конкретні пари. Вона має унікальний індекс id та жорстко прив’язана до конкретного турніру через зовнішній ключ tournament_id. Опоненти поєдинку визначаються через поля participant_a_id та participant_b_id, які посилаються на відповідні профілі користувачів. Поточний етап або номер раунду в сітці записується в полі round_number, поточний статус бою фіксується полем status, а поточний рахунок зустрічі, фіксація балів та технічні дії відображаються у полі score.

Фінансовий аудит системи та безпека проведення онлайн-платежів забезпечуються таблицею PaymentLogs, яка веде суворий облік усіх спроб оплати змагальних внесків. Кожен лог транзакції отримує унікальний id. Користувач, який ініціював платіжну сесію, ідентифікується через зовнішній ключ user_id. Точна сума проведеної фінансової операції записується у полі amount, а код валюти транзакції фіксується полем currency. Фінальний результат обробки платежу платіжним шлюзом записується у полі payment_status, а точна часова мітка операції автоматично проставляється полем created_at.

2.6.2 Забезпечення цілісності даних

Для збереження високої якості та надійності даних у базі даних PostgreSQL використовуються механізми, які наведено нижче.

Зовнішні ключі (Foreign Keys): забороняють видалення користувача з роллю COACH, якщо в таблиці Clubs залишаються прив’язані до нього записи.

Це дозволяє уникнути утворення «сирої» інформації та підтримує логічну структуру.

Обмеження UNIQUE: застосовуються до поля email, щоб унеможливити створення кількох облікових записів з однаковою адресою. Використання ENUM гарантує, що в полі ролі можуть бути записані лише визначені розробником значення (ADMIN, COACH, CLIENT), що захищає базу від введення некоректних даних.

Транзакції: забезпечують атомарність операцій при створенні поєдинків у таблиці Matches та логуванні транзакцій у PaymentLogs.

2.7 Діаграма розгортання

Діаграма розгортання відображає фізичну структуру системи та показує, на яких вузлах розміщені програмні компоненти і як вони взаємодіють між собою в середовищі розробки.

Система розгортається локально на одній машині та складається з трьох основних вузлів (див. рис. 2.7): клієнтського застосунку, серверної частини та бази даних.

Клієнтська частина реалізована на React 19 [22] з використанням збирача Vite і запускається на порту 5173.

Серверна частина побудована на основі Node.js і Express [23, 24], працює на порту 3001 та забезпечує обробку HTTP-запитів від клієнта через REST API за адресою /api.

Зв’язок між клієнтом і сервером здійснюється через бібліотеку Axios, а CORS налаштований таким чином, що сервер приймає запити виключно з адреси http://localhost:5173.

База даних PostgreSQL розгорнута на порту 5432, база називається spas_tournament. Доступ до бази даних з серверної частини здійснюється виключно через ORM Prisma, що забезпечує типобезпечну роботу з даними та

спрощує міграції схеми. Автентифікація користувачів реалізована за допомогою JWT-токенів із терміном дії 7 днів.

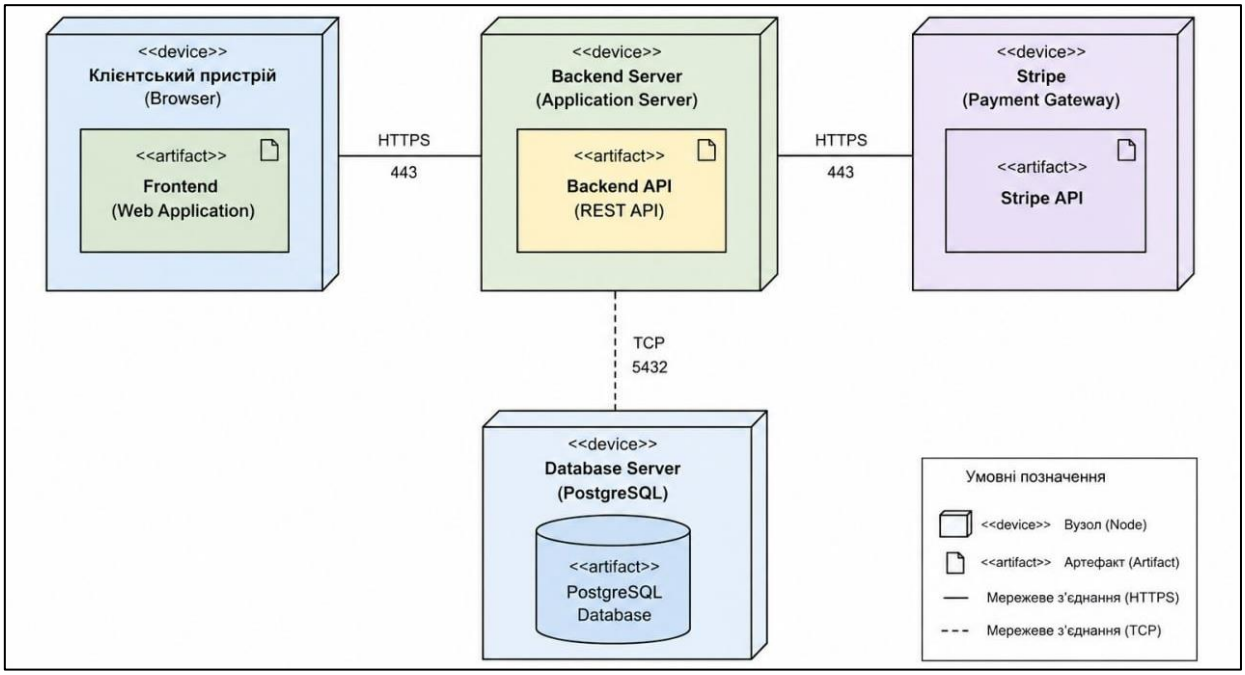


Рисунок 2.7 – Діаграма розгортання

Для обробки платежів система інтегрована із зовнішнім сервісом Stripe. Клієнтська частина використовує бібліотеку Stripe.js із публічним ключем для відображення форми оплати, тоді як серверна частина взаємодіє зі Stripe API через серверний SDK для створення платіжних намірів (PaymentIntent) та обробки вебхуків. На етапі розробки використовуються тестові ключі Stripe.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір середовищ розробки та інструментів

Для реалізації інформаційної системи spas-tournament було сформовано набір інструментів, що забезпечує керованість життєвого циклу програмного продукту, зручність командної взаємодії та контроль якості коду. У якості основного інтегрованого середовища розробки (IDE) використано Visual Studio Code. Даний вибір обумовлений такими перевагами: Нативна підтримка мови TypeScript, розвинена екосистема розширень (ESLint, Prettier, Prisma, REST Client) [6, 2, 10], вбудована інтеграція з системою контролю версій Git, висока продуктивність під час роботи з проєктами, що поєднують клієнтську та серверну частини [16, 17, 18].

Окремою перевагою середовища є зручна навігація по коду, що є критично важливим для багатомодульної архітектури. Для контролю версій застосовано Git, а для централізованого зберігання вихідного коду та організації командної взаємодії – платформу GitHub. Використання цього підходу дозволяє підтримувати історію змін та трасованість прийнятих технічних рішень, застосовувати стратегію гілкування (feature branches) для ізольованої розробки функціоналу, виконувати код-рев’ю за допомогою механізму Pull Request, автоматизувати процеси збирання та перевірки коду (lint, build, test) на етапі безперервної інтеграції (CI).

Для тестування HTTP-інтерфейсів (REST API) використано програмний інструмент Postman. Практична доцільність його застосування полягає в можливості швидкого створення колекцій запитів, параметризації середовищ (розробки та тестування), перевірці кодів відповіді сервера та структури даних. Для системи, що містить аутентифікацію, рольову модель та платіжні сценарії, Postman є ефективним інструментом для проведення регресійного тестування API-контрактів.

Отже, обрана сукупність інструментів формує зріле технологічне середовище для розробки, супроводу та верифікації інформаційної системи спортивного менеджменту.

3.2 Розробка клієнтської частини

Клієнтську частину системи реалізовано на базі сучасного стеку технологій React + TypeScript + Redux Toolkit. Зазначений вибір забезпечує компонентний підхід до побудови інтерфейсу, типобезпечність переданих даних та централізоване керування станом застосунку [1, 2, 3, 22, 24].

3.2.1 Структура клієнтського застосунку

Для забезпечення високого рівня модульності та підтримки принципу поділу відповідальності, було розроблено таку структуру каталогів frontend-модуля:

- app/ – ініціалізація глобального сховища (Store), налаштування маршрутизації та базові хуки;
- pages/ – сторінки інтерфейсу, що відповідають ролям користувачів (учасник, тренер, адміністратор);
- features/ – бізнес-модулі (авторизація, управління учасниками, турніри, турнірні сітки);
- shared/ – перевикористовувані UI-компоненти, API-клієнт та утиліти;
- entities/ – визначення інтерфейсів доменних сутностей.

Практична реалізація описаної структури каталогів frontend-модуля та відображення відповідних інтерфейсів для користувачів системи з різними правами доступу детально продемонстровані нижче. Проєкт охоплює три основні ролі користувачів, для кожної з яких було створено свій окремий

візуальний простір: інтерфейс учасника змагань, який відображає його особисті заходи (див. рис. 3.1), персональний кабінет тренера для управління клубами та заявками (див. рис. 3.2), а також загальна панель управління адміністратора, що слугує для моніторингу всієї системи (див. рис. 3.3).

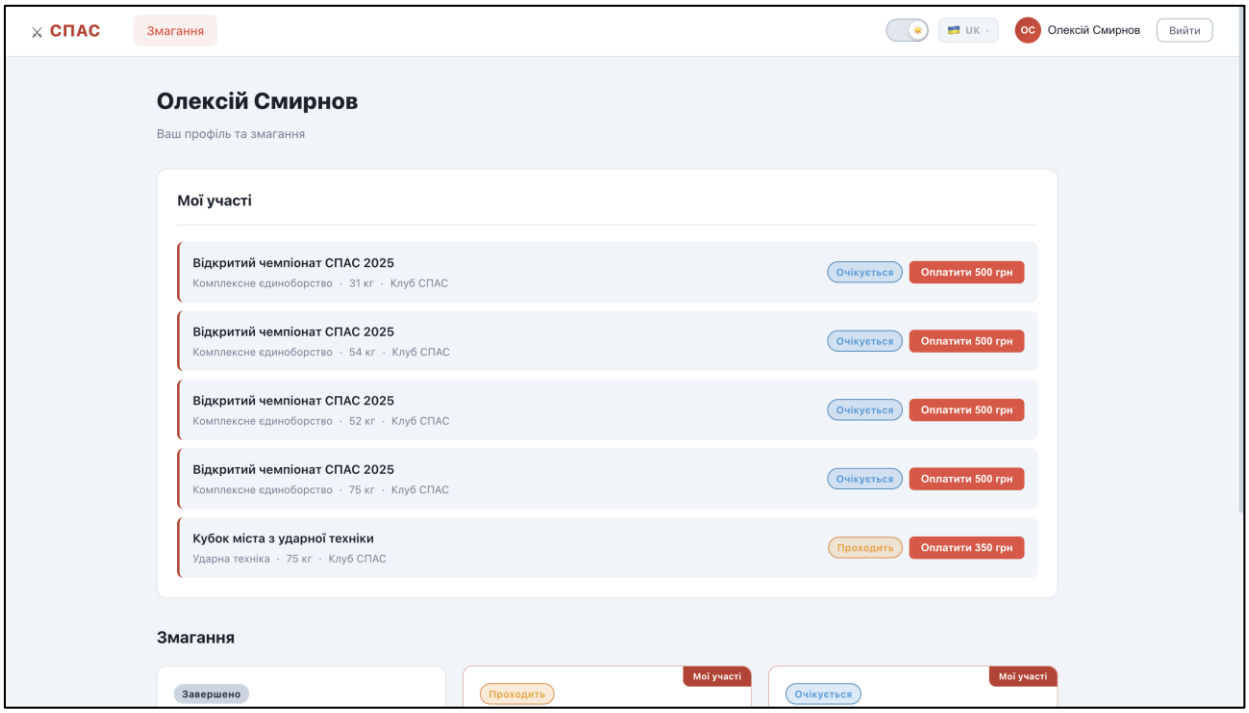


Рисунок 3.1 – Сторінка учасника

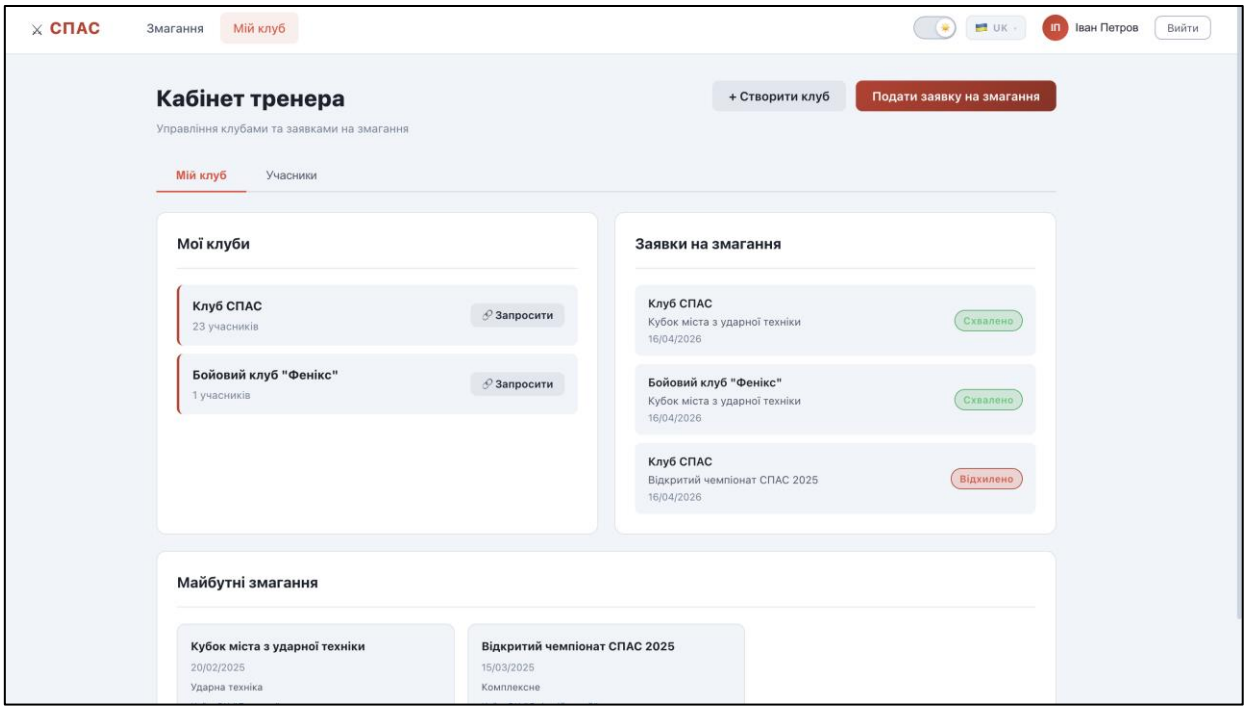


Рисунок 3.2 – Сторінки тренера

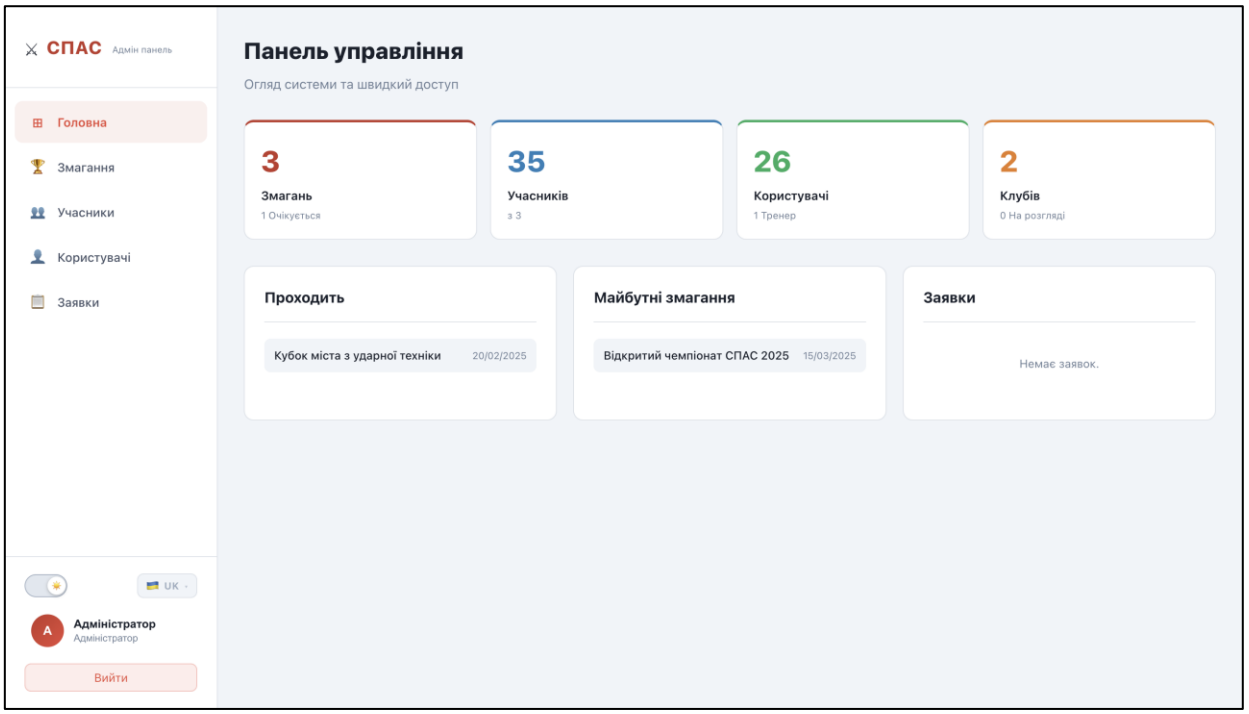


Рисунок 3.3 – Сторінки адміністратора

Кожен екран відображає унікальний набір елементів та функцій, що доступні користувачеві залежно від його прав у системі. При розробці цих сторінок особлива увага приділялася зручності навігації, інтуїтивно зрозумілому розташуванню кнопок та адаптивності елементів, що дозволяє комфортно працювати з програмою на різних пристроях. Такий підхід забезпечує швидкий доступ до потрібної інформації без зайвого навантаження на користувача.

Зазначена структура дозволяє ізолювати бізнес-логіку від презентаційного шару, що істотно спрощує масштабування проєкту.

3.2.2 Приклад компонента авторизації

Нижче наведено приклад реалізації компонента форми входу, написаного з використанням TypeScript та React. Він демонструє класичний приклад реалізації форми авторизації, побудованої на базі React та TypeScript. На рівні коду компонент використовує локальні хуки (наприклад, useState) для

реактивного відстеження введення користувача, статусів завантаження та повідомлень (див. рис. 3.4 – 3.5).

Перед виконанням мережевого запиту вбудована логіка валідації перевіряє коректність даних (наприклад, формат email), блокуючи відправку некоректних форм [8, 9, 12, 13].

```
import React, { useState } from 'react';
import { useAppDispatch, useAppSelector } from '../../app/hooks';
import { loginUser } from '../../features/auth/authSlice';
export const LoginForm: React.FC = () => {
  const dispatch = useAppDispatch();
  const { loading, error } = useAppSelector((s) => s.auth);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const onSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!email.trim() || !password.trim()) { return;}
await dispatch(loginUser({ email, password }));
  };
  return (
    <form onSubmit={onSubmit}>
      <input
        type="email"
        value={email}
        placeholder="Email"
        onChange={(e) => setEmail(e.target.value)}
      />
      <input
        type="password"
        value={password}
        placeholder="Пароль"
        onChange={(e) => setPassword(e.target.value)}
      />
      <button type="submit" disabled={loading}>
        {loading ? 'Вхід...' : 'Увійти'}
      </button>
      {error && <p>{error}</p>}
    </form>
  );
};
```

Рисунок 3.4 – Базова валідація та виклик асинхронних дій

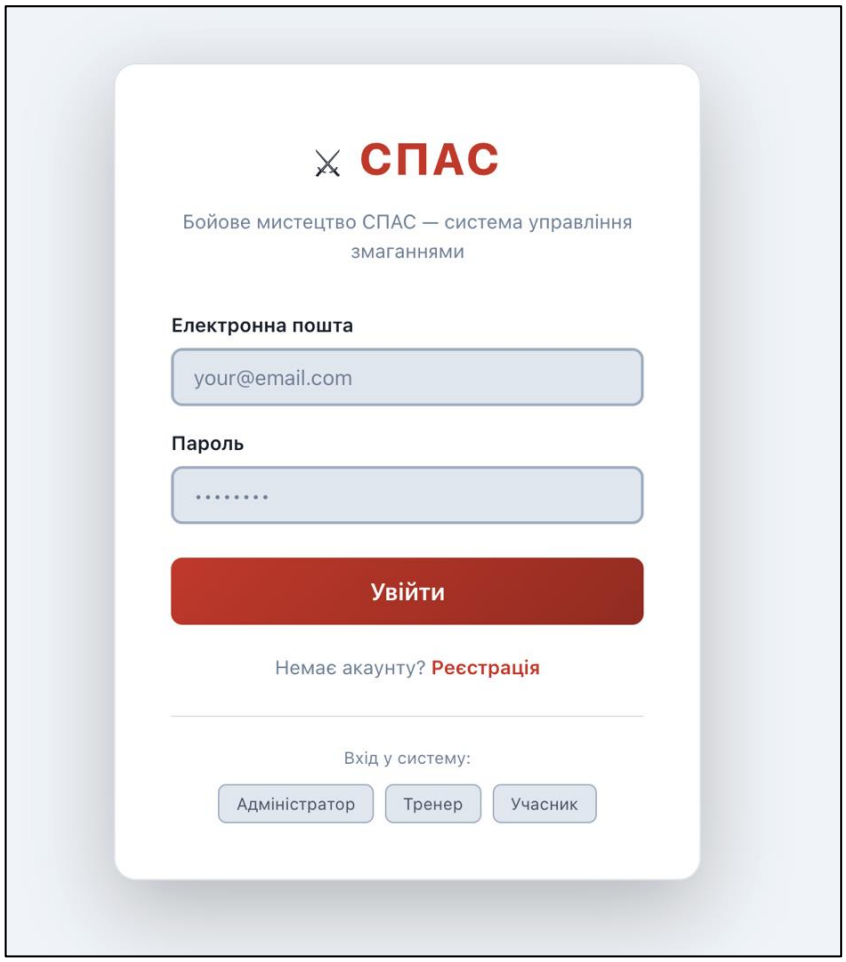


Рисунок 3.5 – Компонент авторизації

Асинхронний шар, реалізований через async/await, забезпечує безпечну взаємодію з сервером, обробляючи очікування мережі, успішний вхід або помилки авторизації.

3.3 Розробка серверної частини

Серверну частину системи реалізовано на базі Node.js + Express + TypeScript [4, 24]. Для досягнення слабкої зв'язності компонентів та забезпечення високого рівня тестованості застосовано пошарову архітектуру:

– routes – визначення маршрутів REST API;
– middleware – реалізація крос-функціональних механізмів (аутентифікація, перевірка прав доступу);

– services – реалізація бізнес-логіки (зокрема генерації турнірних сіток та поєдинків);

– lib – модулі для взаємодії з базою даних (prisma).

Наведений нижче приклад демонструє отримання та створення заявок на участь у турнірах із використанням середовища Express (див. рис. 3.6 – 3.8).

```
import { Router, Request, Response } from 'express';
import { prisma } from '../lib/prisma';
import { authenticate, requireRole } from '../middleware/auth';

const router = Router();
// GET /api/applications?competitionId=...
router.get('/', authenticate, requireRole('admin', 'coach'), async (req: Request, res:
Response) => {
  const { competitionId } = req.query as { competitionId?: string };
  const applications = await prisma.clubApplication.findMany({
    where: competitionId ? { competitionId } : undefined,
    orderBy: { createdAt: 'desc' },
  });
  return res.json(applications);
});
// POST /api/applications
router.post('/', authenticate, requireRole('coach'), async (req: Request, res: Response)
=> {
  const { clubId, competitionId, clubName } = req.body as {
    clubId: string;
    competitionId: string;
    clubName: string;
  };
  if (!clubId || !competitionId || !clubName) {
    return res.status(400).json({ message: 'Некоректні вхідні дані' });
  }
 const created = await prisma.clubApplication.create({
    data: {
      clubId,
      competitionId,
      clubName,
      status: 'pending',
    },
  });
  return res.status(201).json(created);
});
export default router;
```

Рисунок 3.6 – Отримання та створення заявок на участь у турнірах

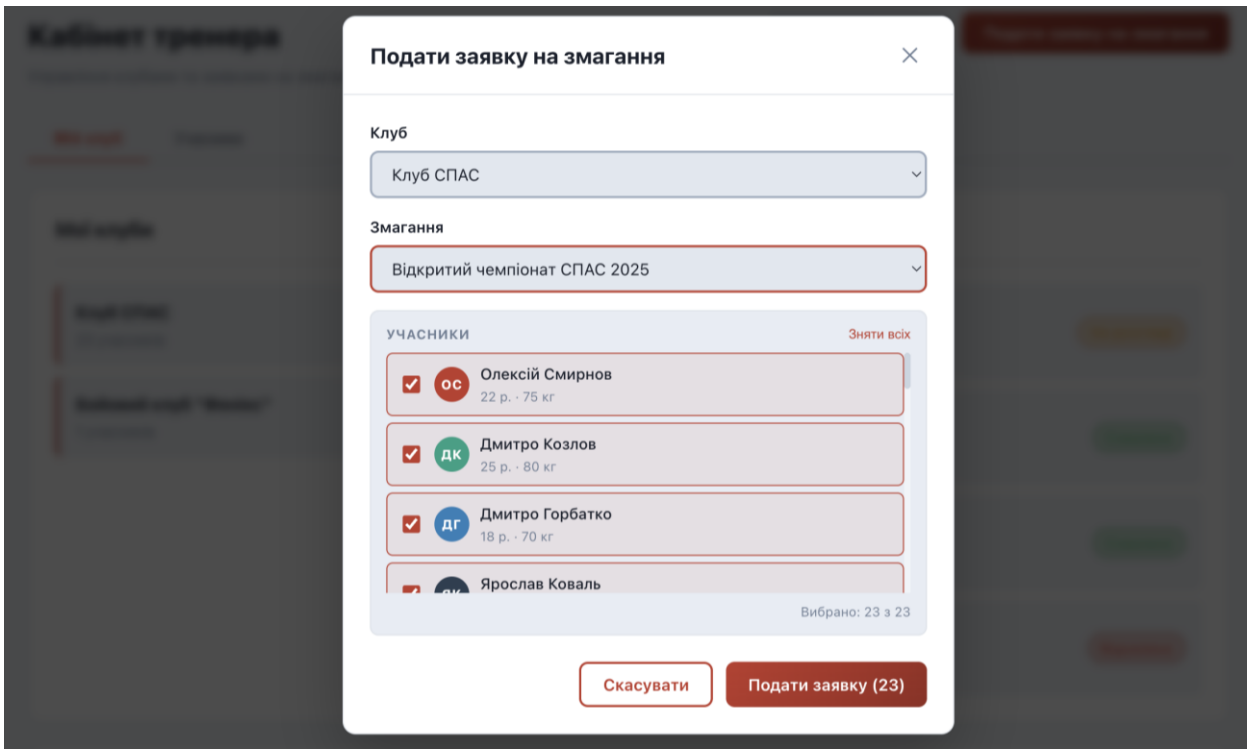


Рисунок 3.7 – UX/UI вигляд отримання та створення заявок на участь у турнірах

```
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
type UserRole = 'client' | 'coach' | 'admin';
declare global {
  namespace Express {
    interface Request {
      user?: { userId: string; role: UserRole }}}}
export function authenticate(req: Request, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Токен відсутній' });
  }
  const token = authHeader.replace('Bearer ', '').trim();
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET!) as {
      userId: string;
      role: UserRole;};
    req.user = { userId: payload.userId, role: payload.role };
    return next();
  } catch {
    return res.status(401).json({ message: 'Невалідний токен' });}}
export function requireRole(...allowed: UserRole[]) {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Неавторизований доступ' })}
    if (!allowed.includes(req.user.role)) {
      return res.status(403).json({ message: 'Недостатньо прав' })}
    return next()}}
```

Рисунок 3.8 – Механізми автентифікації та контролю доступу

Для забезпечення безпеки API-маршрутів розроблено спеціальні проміжні програмні модулі (middleware). Вони відповідають за перевірку валідності токенів та обмеження доступу.

3.4 Інтеграція з платіжною системою

Інтеграція платіжного сервісу Stripe реалізована для забезпечення безпечних, надійних та безперебійних фінансових розрахунків у рамках інформаційної системи spas-tournament. Оскільки система опрацьовує реєстраційні внески та платежі за участь у змаганнях, виникає потреба у жорсткому контролі транзакцій. Процес взаємодії між клієнтським додатком, сервером та платіжною системою побудовано на основі архітектури з використанням безпечного підтвердження намірів (Payment Intents) та асинхронних сповіщень (Webhooks). Процес проведення платежу складається з таких послідовних кроків.

Ініціація оплати з боку клієнта користувач (учасник або тренер) переходить до сторінки оплати в інтерфейсі, де обирає змагання та підтверджує намір внести кошти. Клієнтська частина надсилає захищений запит до бекенду із зазначенням ідентифікатора заявки (competitionId / participantId).

Створення наміру оплати (PaymentIntent) на серверному рівні система перевіряє стан учасника в базі даних PostgreSQL через Prisma ORM. Якщо заявка є дійсною, сервер звертається до Stripe API з використанням секретного ключа (STRIPE_SECRET_KEY) і створює об’єкт PaymentIntent. У запиті зазначається сума, валюта (за замовчуванням ‘EUR’ або інша визначена) та метадані, які дозволяють однозначно ідентифікувати платіж. Stripe повертає на сервер client_secret, який передається на фронтенд [7, 11].

Підтвердження платежу на клієнтській стороні client_secret використовується для відкриття безпечної платіжної форми. Користувач

вводить реквізити картки без збереження даних на серверах нашої системи, що підвищує рівень безпеки та відповідає стандартам PCI DSS.

Оскільки підтвердження платежу є асинхронним процесом, який залежить від банку-емітента, системи карток та самого сервісу Stripe, використання традиційних HTTP-запитів є неефективним. Для цього застосовується механізм вебхуків, який забезпечує гарантовану доставку подій на сервер.

Для забезпечення цілісності фінансових даних накладено такі обмеження:

– перевірка підпису (Signature Verification): метод constructEvent запобігає атакам, під час яких зловмисники можуть спробувати надіслати підроблені запити на сервер (атаки типу Man-in-the-Middle);
– використання транзакцій (Database Transactions): успішні та неуспішні транзакції логуються атомарно: оновлення статусу заявки та створення запису в журналі PaymentLog відбуваються в рамках однієї транзакції PostgreSQL;
– розділення валюти: суми, які надходять зі Stripe у базових одиницях (центах/копійках), обов’язково діляться на 100 для збереження коректного формату даних.

3.5 Організація тестування

Для перевірки працездатності створеного програмного забезпечення було впроваджено комплексний багаторівневий підхід до контролю якості.

Перший рівень включає модульне тестування (Unit Testing) за допомогою фреймворку Jest [25, 26], що дозволяє ізольовано перевірити коректність кожної окремої логічної функції. На наступному етапі інтеграційні тести оцінюють правильність взаємодії API-маршрутів із базою

даних, гарантуючи цілісність інформаційних потоків (див. рис. 3.9). Особлива увага приділена безпеці API, де окремо тестується механізм розмежування прав доступу користувачів відповідно до їхніх системних ролей. Таке поєднання методів забезпечує високу надійність, стабільність роботи та захищеність усього програмного продукту [8].

```
export function resolveWeightCategory(weight: number): string {
  if (weight <= 35) return 'до 35 кг';
  if (weight <= 45) return '36-45 кг';
  if (weight <= 60) return '46-60 кг';
  return 'понад 60 кг';
}
import { resolveWeightCategory } from './categories';
describe('resolveWeightCategory', () => {
  test('повертає "до 35 кг" для ваги 35', () => {
    expect(resolveWeightCategory(35)).toBe('до 35 кг');
  });
  test('повертає "36-45 кг" для ваги 40', () => {
    expect(resolveWeightCategory(40)).toBe('36-45 кг');
  });
  test('повертає "понад 60 кг" для ваги 75', () => {
    expect(resolveWeightCategory(75)).toBe('понад 60 кг');
  });
});
```

Рисунок 3.9 – Приклад модульного тесту

У третьому розділі було описано процес розробки інформаційної системи spas-tournament. Наведено приклади коду для клієнтської та серверної частин, розглянуто особливості безпечної інтеграції з платіжною системою та організацію тестування. Застосування сучасних технологій і підходів до розробки забезпечило високу надійність та готовність програмного продукту до впровадження.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено інформаційну систему spas-tournament для організації та проведення спортивних змагань. Реалізацію здійснено із використанням сучасного технологічного стеку: React, TypeScript і Redux Toolkit (клієнтська частина), Node.js і Express (серверна логіка), Prisma ORM та PostgreSQL (база даних). Поставлену мету досягнуто. Створено вебзастосунок із розмежуванням прав доступу за ролями (учасник, тренер, адміністратор).

Досліджено предметну область організації та проведення спортивних змагань у спортивних школах, клубах і федераціях. Визначено основні бізнес-процеси: реєстрація учасників, перевірка персональних і спортивних даних, управління клубами та заявками, формування турнірних сіток, фіксація результатів поєдинків і контроль оплати участі. Встановлено потребу в автоматизації цих процесів засобами спеціалізованої вебсистеми.

Проаналізовано існуючі підходи до адміністративного супроводу змагань, паперова документація, електронні таблиці та розрізнені програмні засоби. Виявлено, що на практиці організатори часто не мають єдиного інструменту, який поєднує вебдоступ для учасників і тренерів, облік клубів і заявок, генерацію турнірних сіток, ведення результатів та онлайн-оплату внеску.

Проаналізовано предметну область і сформульовано функціональні та нефункціональні вимоги до системи управління змаганнями. Сформовано рольову модель із трьома типами користувачів: учасник, тренер та адміністратор. Побудовано UML-моделі (діаграми варіантів використання, класів, компонентів, послідовностей і діяльності) та виконано опис прецедентів системи.

Розроблено архітектуру клієнт-серверного вебзастосунку з поділом відповідальності між клієнтською частиною, сервером API та базою даних.

Спроєктовано реляційну базу даних із вісьмома основними сутностями, нормалізованими до третьої нормальної форми (3NF), для зберігання профілів користувачів, клубів, змагань, учасників, заявок, поєдинків і даних про платежі; забезпечено цілісність і мінімізацію надмірності даних.

Реалізовано клієнтську частину у вигляді SPA-застосунку на React, TypeScript і Redux Toolkit, а також серверний REST API на Node.js і Express із авторизацією через JWT і ролевою моделлю доступу. Реалізовано програмні модулі системи: автентифікація та реєстрація користувачів; управління клубами й заявками; адміністрування змагань і учасників; генерація турнірних сіток (олімпійська, подвійна олімпійська та кругова системи) і ведення результатів. Система забезпечує повний цикл підготовки та проведення змагань – від реєстрації користувачів до фіксації результатів і контролю оплати.

Забезпечено інтеграцію модуля онлайн-оплати через платіжний сервіс Stripe. Реалізовано створення Payment Intent, обробку webhook-подій та оновлення статусів оплати учасника, передбачено також фіксацію готівкової оплати тренером або адміністратором.

Проведено перевірку функціональності розробленої системи. Перевірено основні сценарії взаємодії – авторизація, робота з клубами та заявками, генерація турнірної сітки, ведення результатів і оплата участі. Підтверджено коректність реалізованих функцій і можливість використання системи в практичній діяльності спортивних організацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Official Documentation. URL: https://react.dev/ (дата звернення: 12.02.2026).

2. Standard ECMA-262 : ECMAScript® 2025 Language Specification : 16th edition / Ecma International. Geneva, 2025. 847 p. URL: https://ecma-international.org/wp-content/uploads/ECMA-262_16th_edition_june_2025.pdf (дата звернення: 12.02.2026).

3. Redux Toolkit Official Documentation. URL: https://redux-toolkit.js.org/ (дата звернення: 18.02.2026).

4. Node.js Official Documentation. URL: https://nodejs.org/ (дата звернення: 20.02.2026)

5. Express.js Official Documentation. URL: https://expressjs.com/ (дата звернення: 20.02.2026).

6. Prisma ORM Official Documentation. URL: https://www.prisma.io/docs/ (дата звернення: 05.03.2026).

7. Stripe API Reference. URL: https://docs.stripe.com/ (дата звернення: 10.03.2026).

8. Jest Testing Framework Documentation. URL: https://jestjs.io/ (дата звернення: 15.03.2026).

9. MDN Web Docs (JavaScript & Web Technologies). URL: https://developer.mozilla.org/ (дата звернення: 10.02.2026).

10. Vite Guide. URL: https://vite.dev/guide/ (дата звернення: 14.02.2026).

11. PostgreSQL 16 Official Documentation. URL: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-US.pdf (дата звернення: 01.03.2026).

12. OWASP Top 10:2021 (The Most Critical Web Application Security Risks). URL: https://owasp.org/www-project-top-ten/ (дата звернення: 18.03.2026).

13. OWASP Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 18.03.2026).

14. Smoothcomp (Tournament Management Software Platform). URL: https://smoothcomp.com/ (дата звернення: 25.02.2026).

15. Challonge (Tournament Brackets Generator). URL: https://www.challonge.com/ (дата звернення: 25.02.2026).

16. Sommerville I. Software Engineering : 11th ed. London : Pearson Education, 2020. 202 p.

17. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms : 4th ed. Cambridge : MIT Press, 2022. 99 p.

18. Jones M., Bradley J. JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force (IETF), 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 20.03.2026).

19. Про фізичну культуру і спорт : Закон України від 24.12.1993 № 3808-XII. Відомості Верховної Ради України. 1994. № 14. Ст. 80. (зі змінами та доповненнями). URL: https://zakon.rada.gov.ua/laws/show/3808-12 (дата звернення: 04.04.2026).

20. Зубков О. В. Проєктування та розробка веб-орієнтованих інформаційних систем : підручник. Київ : КНУТД, 2022. 234 с.

21. Дорогий А. В. Бази даних та інформаційні системи : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 67 с.

22. Banks A., Porcello E. Learning React: Modern Patterns for Developing Real-World Applications : 2nd ed. O’Reilly Media, 2020, 300 p.

23. Casciaro M., Mammino L. Node.js Design Patterns: Design and implement production-grade Node.js applications using Next-generation JavaScript : 3rd ed. Packt Publishing, 2020. 109 p.

24. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack : 2nd ed. O’Reilly Media, 2019. 129 p.

25. Aniche M. Effective Software Testing: A developer’s guide. Manning Publications, 2022. 99 p.

26. Irvine J. Mastering JavaScript Test-Driven Development: TDD with Jest. Packt Publishing, 2021. 109 p.

ДОДАТОК А

Реалізація Front-End частини

А.1 Файл main.tsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { BrowserRouter } from 'react-router-dom';
import { store } from './app/store';
import { AppRouter } from './app/Router';
import { ThemeProvider } from './shared/theme/ThemeContext';
import './styles/global.css';
import './shared/i18n/i18n';


ReactDOM.createRoot(document.getElementById('root')!).render(

  <React.StrictMode>
    <ThemeProvider>
      <Provider store={store}>
        <BrowserRouter>
          <AppRouter />
        </BrowserRouter>
      </Provider>
    </ThemeProvider>
  </React.StrictMode>

);
```

А.2 Файл store.tsx

```
import { configureStore } from '@reduxjs/toolkit';
import authReducer from '../features/auth/authSlice';
import competitionsReducer from
'../features/competitions/competitionsSlice';
import participantsReducer from '../features/participants/participantsSlice';
import clubsReducer from '../features/clubs/clubsSlice';
```

```
import bracketReducer from '../features/bracket/bracketSlice';
import usersReducer from '../features/users/usersSlice';

export const store = configureStore({

  reducer: {
    auth: authReducer,
    competitions: competitionsReducer,
    participants: participantsReducer,
    clubs: clubsReducer,
    bracket: bracketReducer,
    users: usersReducer,
  },
});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;
```

А.3 Файл Router.tsx

```
import React from 'react';
import { Routes, Route, Navigate } from 'react-router-dom';
import { useAppSelector } from './hooks';
import { ProtectedRoute } from './ProtectedRoute';

import { LoginPage } from '../pages/LoginPage/LoginPage';
import { RegisterPage } from '../pages/RegisterPage/RegisterPage';
import { InvitePage } from '../pages/InvitePage/InvitePage';
import { ClientDashboard } from
'../pages/ClientDashboard/ClientDashboard';
import { CoachDashboard } from
'../pages/CoachDashboard/CoachDashboard';
import { AdminDashboard } from
'../pages/admin/AdminDashboard/AdminDashboard';
import { CompetitionPage } from
'../pages/admin/CompetitionPage/CompetitionPage';
import { ParticipantsPage } from
'../pages/admin/ParticipantsPage/ParticipantsPage';
import { BracketPage } from '../pages/admin/BracketPage/BracketPage';
import { UsersPage } from '../pages/admin/UsersPage/UsersPage';
import { ApplicationsPage } from
```

```
'../pages/admin/ApplicationsPage/ApplicationsPage';
import { AdminLayout } from '../shared/ui/Layout/AdminLayout';
import { ClientLayout } from '../shared/ui/Layout/ClientLayout';

const RootRedirect: React.FC = () => {
  const user = useAppSelector((s) => s.auth.currentUser);
  if (!user) return <Navigate to="/login" replace />;
  if (user.role === 'admin') return <Navigate to="/admin" replace />;
  return <Navigate to="/dashboard" replace />;
};

export const AppRouter: React.FC = () => {
  return (
    <Routes>
      <Route path="/" element={<RootRedirect />} />
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
<Route path="/invite/:token" element={<InvitePage />} />

      <Route
        element={
          <ProtectedRoute allowedRoles={['client', 'coach']}>
            <ClientLayout />
          </ProtectedRoute>
        }
      >
        <Route path="/dashboard" element={<ClientDashboard />} />
        <Route
          path="/coach"
          element={
            <ProtectedRoute allowedRoles={['coach']}>
              <CoachDashboard />
            </ProtectedRoute>
          }
        />
      </Route>

      <Route
        path="/admin"
        element={
          <ProtectedRoute allowedRoles={['admin']}>
            <AdminLayout />
          </ProtectedRoute>
        }
```

```
    >
      <Route index element={<AdminDashboard />} />
      <Route path="competitions" element={<CompetitionPage />} />

      <Route path="participants" element={<ParticipantsPage />} />
      <Route path="users" element={<UsersPage />} />
      <Route path="applications" element={<ApplicationsPage />} />
      <Route path="bracket/:competitionId" element={<BracketPage />} />
    </Route>

    <Route path="*" element={<Navigate to="/" replace />} />
  </Routes>
 );
};
```

A.4 Файл ProtectedRoute.tsx

```
import React from 'react';
import { Navigate, useLocation } from 'react-router-dom';
import { useAppSelector } from './hooks';
import type { UserRole } from '../entities/interfaces';

interface ProtectedRouteProps {
 children: React.ReactNode;
 allowedRoles?: UserRole[];
}

export const ProtectedRoute: React.FC<ProtectedRouteProps> = ({
children, allowedRoles }) => {
 const user = useAppSelector((s) => s.auth.currentUser);
 const location = useLocation();
 if (!user) {
   return <Navigate to="/login" state={{ from: location }} replace />;

}
 if (allowedRoles && !allowedRoles.includes(user.role)) {
   if (user.role === 'admin') return <Navigate to="/admin" replace />;
   return <Navigate to="/dashboard" replace />;
 }

 return <>{children}</>;
};
```

А.5 Файл apiClient.ts

```
import axios from 'axios';

const BASE_URL = (import.meta.env.VITE_API_URL as string)

export const apiClient = axios.create({
  baseURL: BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('spas_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

apiClient.interceptors.response.use(
  (response) => response,

  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('spas_token');
      localStorage.removeItem('spas_user');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

А.6 Файл authSlice.ts

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import type { PayloadAction } from '@reduxjs/toolkit';
import type { IUser, UserRole } from '../../entities/interfaces';
import { apiClient } from '../../shared/api/apiClient';
```

```
interface AuthState {
  currentUser: Omit<IUser, 'password'> | null;
  token: string | null;
  loading: boolean;
  error: string | null;

}

const initialState: AuthState = {
  currentUser: JSON.parse(localStorage.getItem('spas_user') || 'null'),
  token: localStorage.getItem('spas_token'),
  loading: false,
  error: null,
};
export const loginThunk = createAsyncThunk(
  'auth/login',
  async (credentials: { email: string; password: string }, { rejectWithValue }) => {
    try {
      const response = await apiClient.post<{ user: IUser; token: string }>(
        '/auth/login',
        credentials
      );
      return response.data;
    } catch (err: unknown) {
      const error = err as { response?: { data?: { message?: string } } };
      return rejectWithValue(error.response?.data?.message || 'Помилка входу');
    }
  }
);

export const registerThunk = createAsyncThunk(
  'auth/register',
  async (data: { name: string; email: string; password: string }, { rejectWithValue }) => {
    try {
      const response = await apiClient.post<{ user: IUser; token: string }>(
        '/auth/register',
        data
      );
      return response.data;
    } catch (err: unknown) {
      const error = err as { response?: { data?: { message?: string } } };
      return rejectWithValue(error.response?.data?.message || 'Помилка реєстрації');
    }
  }
);
```

```
export const updateAvatarThunk = createAsyncThunk(
  'auth/updateAvatar',
  async ({ id, avatar }: { id: string; avatar: string }, { rejectWithValue }) => {
    try {

      const response = await apiClient.patch<Omit<IUser,
'password'>>(`/users/${id}/avatar`, { avatar });
      return response.data;
    } catch (err: unknown) {
      const error = err as { response?: { data?: { message?: string } } };
      return rejectWithValue(error.response?.data?.message || 'Помилка оновлення
фото');
    }
  }
);

export const updateProfileThunk = createAsyncThunk(
  'auth/updateProfile',
  async ({ id, name, avatar }: { id: string; name?: string; avatar?: string | null }, {
rejectWithValue }) => {
    try {
      const response = await apiClient.patch<Omit<IUser,
'password'>>(`/users/${id}/profile`, { name, avatar });
      return response.data;
    } catch (err: unknown) {
      const error = err as { response?: { data?: { message?: string } } };
      return rejectWithValue(error.response?.data?.message || 'Помилка оновлення
профілю');
    }}
);

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    logout(state) {
      state.currentUser = null;
      state.token = null;
      localStorage.removeItem('spas_user');
      localStorage.removeItem('spas_token');
    },
    clearError(state) {
      state.error = null;
    },
    updateUserRole(state, action: PayloadAction<{ userId: string; role: UserRole }>) {
      if (state.currentUser && state.currentUser.id === action.payload.userId) {
```

```
      state.currentUser.role = action.payload.role;
      localStorage.setItem('spas_user', JSON.stringify(state.currentUser));
    }
  },
},
extraReducers: (builder) => {
  builder
    .addCase(loginThunk.pending, (state) => {
      state.loading = true;
      state.error = null;
    })
    .addCase(loginThunk.fulfilled, (state, action) => {
      state.loading = false;
      const { password: _p, ...userWithoutPassword } = action.payload.user;
      state.currentUser = userWithoutPassword;
      state.token = action.payload.token;
      localStorage.setItem('spas_user', JSON.stringify(userWithoutPassword));
      localStorage.setItem('spas_token', action.payload.token);
    })
    .addCase(loginThunk.rejected, (state, action) => {
      state.loading = false;
      state.error = action.payload as string;
    })
    .addCase(registerThunk.pending, (state) => {
      state.loading = true;
      state.error = null;
    })
    .addCase(registerThunk.fulfilled, (state, action) => {
      state.loading = false;
      const { password: _p, ...userWithoutPassword } = action.payload.user;
      state.currentUser = userWithoutPassword;
      state.token = action.payload.token;
      localStorage.setItem('spas_user', JSON.stringify(userWithoutPassword));
      localStorage.setItem('spas_token', action.payload.token);
    })
    .addCase(registerThunk.rejected, (state, action) => {
      state.loading = false;
      state.error = action.payload as string;
    })
    .addCase(updateAvatarThunk.fulfilled, (state, action) => {
      if (state.currentUser && state.currentUser.id === action.payload.id) {
        state.currentUser = { ...state.currentUser, avatar: action.payload.avatar };
        localStorage.setItem('spas_user', JSON.stringify(state.currentUser));
      }
    })
    .addCase(updateProfileThunk.fulfilled, (state, action) => {
```

```
      if (state.currentUser && state.currentUser.id === action.payload.id) {
        state.currentUser = { ...state.currentUser, ...action.payload };
        localStorage.setItem('spas_user', JSON.stringify(state.currentUser));}});
  },
});
export const { logout, clearError, updateUserRole } = authSlice.actions;
export default authSlice.reducer;
```