Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система з адміністрування футбольних секцій. Back-end

(тема)

Виконав:
студент 4 курсу, групи ПЗПІпз-22-1

Персіанова О.Ю.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. Валенда Н.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Персіановій Олені Юріївні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система з адміністрування футбольних секцій. Back-end

Затверджена наказом по університету від 17.06. 2024р. № 93 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 14.07.2024

3. Вихідні дані до роботи Розробити серверну частину програмного застосунку з адміністрування футбольних секцій в Україні, котрий містить в собі режими адміністратора, менеджера секцій, тренера та опікуна неповнолітніх гравців, з використанням мови програмування Python та викокорівневого фреймворку Django.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2024	*виконано*
2	Створення специфікації ПЗ	22.05.2024	*виконано*
3	Проектування ПЗ	24.05.2024	*виконано*
4	Розробка ПЗ	28.05.2024	*виконано*
5	Тестування ПЗ	05.06.2024	*виконано*
6	Оформлення пояснювальної записки	10.06.2024	*виконано*
7	Підготовка презентації та доповіді	10.07.2024	*виконано*
8	Попередній захист	15.07.2024	*виконано*
9	Нормоконтроль, рецензування	18.07.2024	*виконано*
10	Здача роботи у електронний архів	18.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	19.07.2024	*виконано*

Дата видачі завдання 06 травня 2024р.

Студент (ка)_____ Персіанова О.Ю.____
(підпис)

Керівник роботи _____ _____доц. Валенда Н.А._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 76 стор., 18 рисунків, 4 таблиці, 5 додатків, 20 джерел.

АДМІНІСТРУВАННЯ ФУТБОЛЬНИХ СЕКЦІЙ, ВИСОКОРІВНЕВИЙ ФРЕЙМВОРК DJANGO, МОВА ПРОГРАМУВАННЯ PYTHON, НАВЧАННЯ ФУТБОЛУ В УКРАЇНІ, РЕЛЯЦІЙНА БАЗА ДАНИХ, CELERY, DOCKER, ORM, POSTGRESQL, REST API.

Об’єкт розробки – серверна частина програмного застосунку з адміністрування футбольних секцій в Україні.

Мета розробки – створення серверної частини програмного застосунку, який підтримуватиме чотири користувацькі ролі: адміністратора, менеджера секцій, тренера та опікуна неповнолітніх гравців.

Метод рішення – викокорівневий фреймворк Django та Jetbrains Pycharm, мова програмування Python.

В результаті розробки був створений застосунок, який дозволяє авторизованим користувачам у відповідному режимі (адміністратора, менеджера секцій, тренера та опікуна) взаємодіяти та зберігати дані про себе та інших користувачів, футбольні секції, програми, заняття та команди, а також реєструвати гравців на програми та включати їх в футбольні команди. Також програма дозволяє шукати, фільтрувати інформацію, отримувати статистики реєстрацій на програми та підтримує розсилку листів на пошту тренера про підтвердження футбольної секції навчальної програми.

The object of development is back-end for a software application dedicated to football management and football sections administration.

The purpose of the work is to create a software application that will provide 4 modes of football management: administrator (staff), section manager, coach, and guardian.

Solution method – Django web development framework, JetBrainds Pycharm development environment, and Python programming language.

As a result of the development, an application was created that allows authorized users in the appropriate mode (administrator, section manager, coach or guardian) to interact and save data about themselves and other users, football sections, programs, classes and teams, as well as register players for programs and include them in football teams. Also, the program allows you to search, filter information, receive statistics of registrations for programs and supports the sending of letters to the coach's mail about the confirmation of the football section of the training program.

Я, Персіанова Олена Юріївна, студентка гр. ПЗППп-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система з адміністрування футбольних секцій. Back-end», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі 9
1.1 Аналіз предметної області 9
1.2 Виявлення проблем 11
1.3 Постановка задачі 13
2 Формування вимог до програмної системи 15
2.1 Опис програмної системи 15
2.2 Характеристики користувачів 20
2.3 Функції продукту 23
2.4 Нефункціональні вимоги 25
3 Архітектура та проєктування програмного забезпечення 26
3.1 UML проєктування програмного забезпечення 26
3.2 Проєктування бази даних 30
3.3 Архітектура серверної частини застосунку 30
3.4 Приклади алгоритмів та методів 35
4 Опис прийнятих програмних рішень 38
4.1 Обґрунтування вибору викокорівневого фреймворку 38
4.2 Обґрунтування вибору бази даних 41
4.3 Обґрунтування вибору хмарного рішення зберігання файлів 43
4.4 Обґрунтування вибору сервісу розсилки повідомлень 45
4.5 Огляд реалізації операцій CRUD у фреймворку Django 47
5 Тестування розробленого програмного забезпечення 52
5.1 Проведення API тестування серверної частини 52
5.2 Проведення навантажувального тестування серверної частини 56
Висновки 60
Перелік джерел посилання 61
Додаток А 63
Додаток Б 64
Додаток В 65

Додаток Г 66
Додаток Д 67

ВСТУП

Відомо, що організація та супроводження футбольних занять є важкою та кропітливою роботою, що нерідко лягає на плечі футбольних тренерів. Формування команд, комунікація з гравцями та їхніми батьками або опікунами з різноманітних організаційних питань, координація з менеджерами спортивних секцій - лише кілька прикладів подібної роботи. Для гравців та їхніх батьків або опікунів, в свою чергу, важливим є співпраця саме з найкращими та перевіреними тренерами. Стратегічно важливим є також розвиток футбольної спільноти України та довгострокових взаємозв'язків між усіма зацікавленими сторонами футболу.

Даний обсяг робіт є потенційним об'єктом автоматизації та предметною сферою для програмного застосунку та визначає актуальність розробки платформи з адміністрування футбольних секцій, або платформи з розвитку відносин в сфері спорту. Проектування та програмна реалізація серверної частини програмної системи є метою даної роботи. Під час проектування проведено ER-моделювання, концептуальне та логічне моделювання реляційної БД, а також проведена її фізична реалізація.

В результаті спроєктовано та реалізовано застосунок, який дозволяє взаємодіяти та зберігати дані про користувачів, футбольні секції, програми, заняття та команди, а також реєструвати гравців на програми та включати їх в футбольні команди. Також програма дозволяє шукати, фільтрувати інформацію, отримувати статистики реєстрацій на програми та підтримує задачу з розсилки листів на пошту тренера про підтвердження менеджером футбольної секції навчальної програми.

Застосунок є легким для розуміння та швидкого пристосування, отже людина з будь-яким досвідом може з легкістю його використовувати.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Платформа з адміністрування спортивних секцій надаватиме можливість шукати та організовувати футбольні секції тренерам України, а також реєструватися на футбольні програми гравцям або їхнім опікунам/батькам. Менеджери секцій матимуть змогу адмініструвати свої футбольні секції, підтверджувати програми та футбольних тренерів, що проводять заняття у секціях, тощо. Програмна система буде присвячений розвитку Sport Relationship Management (SRM) в сфері футболу України, що надасть змогу керувати ролями всіх учасників програмної системи та надавати відповідні права [1].

Футбольна секція є однією з основних структурних одиниць системи, що фізично співвідноситься з футбольним полем за зазначеною адресою, яке належить певній організації та де проводять заняття гравці з тренером. Футбольна секція має закріпленого за нею менеджера та виступає фундаментом для проведення тренерами своїх навчальних спортивних програм. Секція може бути основою для проведення занять за кількома програми. Програмою, що складається з занять, керує тренер.

Робота менеджерів секцій пов’язана з: реєстрацією програм тренерів на футбольні секції; адмініструванням футбольних секцій, як-от, підтримання інформації за ними в актуальному стані, як-от, назва, адреса.

Адміністратор виконує наступні функції: створення та редагування інформації користувачів в інтерфейсі адміністратора CRUD операціями над усіма іншими ресурсами, у тому числі всіма футбольними секціями системи та опікунами гравців.

Інформаційні потреби менеджерів секцій є наступними: необхідна інформація про програми та заняття, що проводяться у секції менеджера; необхідна інформація про тренерів, заняття яких проводяться у секції менеджера. Інформаційними потребами адміністраторів системи виступають наступні аспекти:

необхідна інформація про всі футбольні секції системи, опікунів гравців та інші ресурси.

Наразі існують деякі застосунки для пошуку футбольних тренерів футбольних, що можуть виступати прямими конкурентами застосунку. Під час конкурентного аналізу було проаналізовано наступні фактори: способи монетизації; способи залучення користувачів; фактори успішності.

Основний прямий конкурент - сервіс пошуку тренерів “Trener” [2]. Має широкий перелік видів спорту, практично всі види спорту, можна записатися конкретно до тренера, немає тематичної розбивки. Монетизація: плата знімається з тренера за розміщення профілю. Перевага: через те що має вузьку спеціалізацію (пошук тренерів), відсутні прямі конкуренти в Україні.

Прямі конкуренти - дошки об’яв: сервіс пошуку репетиторів “BUKI” [3], сервіс пошуку та доставки товарів “OLX” [4], спеціалізований сервіс пошуку навчальних закладів “Edusearch” [7], сервіс замовлення послуг “Кабанчик” [8]. Програмою, що найбільше відповідає тематиці даного курсового проекту, є веб-сервіс «Trener» [2]. В даному сервісі можуть реєструватися тренери та заповнювати власні профілі (рис.1.1). Анонімні користувачі (потенційно, самі гравці або їхні опікуни/батьки) можуть знайти тренера за видами спорту, містом, районом міста або станцією метро, віком, статтю, вартістю послуг, розкладом (рис. 1.2).

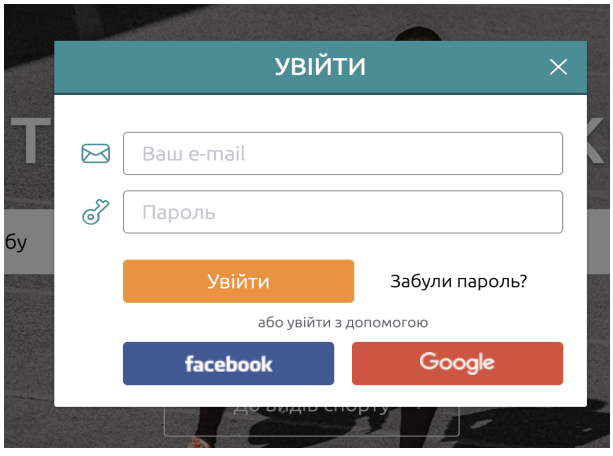


Рисунок 1.1 – Інтерфейс реєстрації та логіну адміна веб-сервісу «Trener» (публічна інформація [2], режим анонімного користувача)

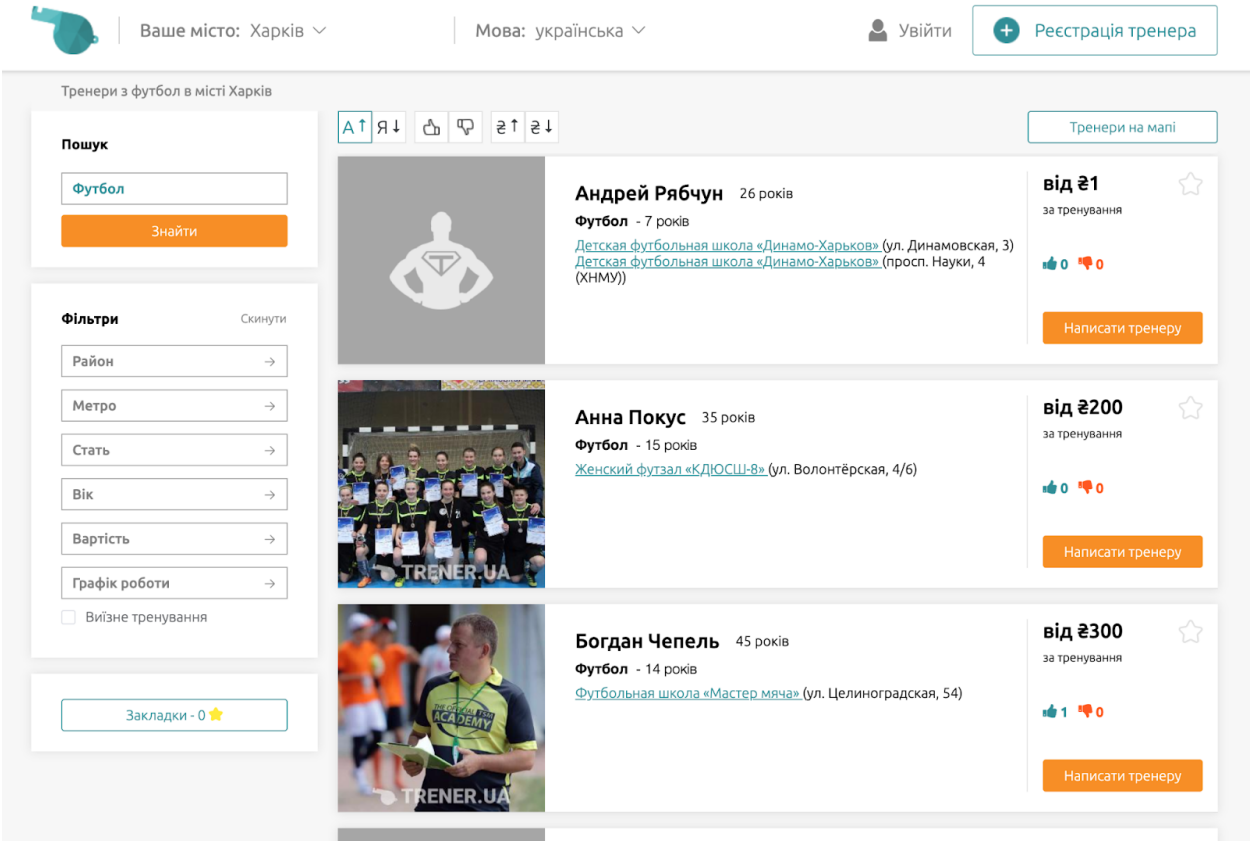


Рисунок 1.2 – Інтерфейс результатів пошуку тренерів у «Trener» (публічна інформація [2], режим анонімного користувача)

Таким чином, перевагами веб-сервісу «Trener» є її зручність у використанні та якісний інтерфейс, підтримка локалізації, покриття багатьох видів спорту, підтримка великої кількість фільтрів пошуку тренерів, оперативний супровід розробниками, а з недоліків – те, що сервіс не розрахований на допомогу тренеру або менеджерам секцій в адмініструванні програм, команд і занять, або гравцям - у плануванні занять.

1.2 Виявлення проблем

Порівняння конкурентів ринку навчання футболу України проведено за рядом факторів крізь призму конкурентних переваг програмного продукту, що розроблятиметься (табл. 1.1).

Таблиця 1.1 - Порівняння прямих конкурентів (таблиця виконана самостійно)

Характеристика	Конкурент				
	Trener [2]	Кабанчик [6]	Buki [3]	Olx [4]	Edusearch [5]
Сфокусованість на набутті навичок (компетентнісний підхід)	+ (частково)	-	-	-	-
Контроль якості послуг тренерів	+ (відгуки)	+ (відгуки)	+ (відгуки)	-	+ (відгуки)
Організаційне супроводження користувачів (тренерів та гравців)	-	-	-	-	-
Зручний календар з планування занять	-	-	-	-	-
Загальний рівень складності UI / UX	помірний	високий	високий	помірний	помірний
Покриття інших видів спорту, наприклад, футзал, теніс, баскетбол.	+	+	+	+	+
Підтримка локалізації	+ (2 мови)	+ (2 мови)	+ (3 мови)	+ (2 мови)	-

Серед запланованих переваг програмної системи варто зазначити наступні:

- сфокусованість на набутті навичок (компетентнісний підхід). Користувач придбаває конкретний курс - відкриту навчальну програму, а не послуги тренера. Реалізація підходу “бачиш, що купуєш”; контроль якості послуг тренерів. Перевірка тренерів на сертифікованість в Українській Асоціації Футболу (UAF): тренер може відкривати програму тільки якщо вони UAF-сертифіковані;
- організаційне супроводження користувачів (тренерів та гравців): наявність менеджерів секцій, які можуть підтримати як гравців, так і тренерів;
- зручний календар з планування занять: консолідація даних: занять для тренерів (їм не треба вести розклад вручну);
- загальний рівень складності UI / UX: низький рівень складності UI / UX функціональності, що вирішує конкретну задачу - адміністрування своїх

футбольних секцій користувачами різноманітних категорій: гравцями, тренерами, менеджерами футбольних секцій - згідно відповідних прав.

Як бачимо, конкуренти не мають певних конкурентних переваг, що заплановано програмною системою. Наприклад, контроль якості послуг тренерів присутній тільки в формі відгуків.

1.3 Постановка задачі

Користувачами додатка є наступні групи: анонімні користувачі, адміністратори, області менеджери секцій, студенти (гравці), їхні батьки або опікуни, сертифіковані тренери.

Проектування програмної системи для адміністрування футбольних секцій в Україні ставить перед собою кілька важливих завдань, необхідних для забезпечення ефективного функціонування системи та задоволення потреб усіх зацікавлених сторін. Ці завдання включають як технічні, так і організаційні аспекти, адже система повинна враховувати різні рівні користувачів, їхні потреби та взаємодію між собою. Основна мета полягає у створенні інтуїтивно зрозумілого, ефективного та надійного додатку, що дозволяє тренерам, менеджерам секцій, гравцям та їх опікунам взаємодіяти у зручний та організований спосіб.

Важливим аспектом є забезпечення високої продуктивності та надійності системи. Це означає, що система повинна працювати стабільно та швидко, обробляючи великі обсяги даних без затримок і збоїв. Вона повинна бути розроблена так, щоб забезпечити максимальну безпеку даних користувачів, запобігаючи несанкціонованому доступу та захищаючи конфіденційну інформацію. Для досягнення цих цілей необхідно використовувати сучасні технології та методи розробки програмного забезпечення, що забезпечать високу якість продукту та його відповідність усім стандартам.

Основними завданнями роботи є наступні:

- визначити функціональні та нефункціональні вимоги програмної системи з адміністрування футбольних секцій;

- спроєктувати базу даних програмної системи;
- створити базу даних за допомогою ORM веб-фреймворку Django;
- реалізувати REST API програмної системи за допомогою веб-фреймворку Django та бібліотек мови програмування Python, включно з авторизацією за ролями адміністратора, менеджера секцій, тренера та опікуна;
- реалізувати функціональність відправки повідомлень на електронну пошту користувачів за допомогою менеджера задач Celery та сервісу Sendgrid;
- провести API тестування за допомогою інструменту Postman та навантажувальне тестування з бібліотекою locust;
- створити середовище розгортання проєкту за допомогою Docker та Docker Compose.

Реалізація даних завдань дозволить повною мірою реалізувати мету роботи, що полягає в створенні повноцінної програмної системи з адміністрування футбольних секцій.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Опис програмної системи

Серверна частина програмної системи адміністрування футбольних секцій реалізована за допомогою веб-фреймворку Django та споріднених бібліотек мови програмування Python. За допомогою Django реалізовано логіку API енпдоїнтів, авторизацію за ролями адміністратора, менеджера секцій, тренера та опікуна.

Врахування специфічних потреб кожної категорії користувачів є важливим аспектом програми. Тренери мають можливість створювати та керувати навчальними програмами, планувати тренування та відстежувати прогрес гравців. Менеджери секцій мають інструменти для ефективного управління секціями. Гравці можуть реєструватися на програми, переглядати розклад тренувань та отримувати сповіщення про зміни. Опікуни мають доступ до інформації про тренування та контактувати з тренерами у разі потреби.

Система забезпечує можливість реєстрації гравців на програми, створені тренерами, додавання гравців до команд, а також управління даними про секції, програми, заняття та команди. Ця функціональність дозволяє тренерам ефективно організовувати свої тренувальні сесії, розподіляти гравців за командами, а також слідкувати за їхнім прогресом. Важливою функцією є пошук і фільтрація інформації, яка дозволяє користувачам швидко знаходити необхідні дані, що значно спрощує управління великим обсягом інформації. Наприклад, тренери можуть швидко знаходити списки гравців, доступні програми або розклади занять, що дозволяє їм більш ефективно планувати свою роботу.

Також особлива увага приділяється автоматизованій розсилці сповіщень електронною поштою про зміни в розкладі занять. Це надзвичайно важливо для підтримки комунікації з гравцями та їх опікунами, оскільки дозволяє їм своєчасно отримувати актуальну інформацію про тренування, зміни в розкладі або інші важливі новини. Такий підхід значно знижує ризик виникнення непорозумінь або пропусків занять через недоотриману інформацію.

Ці можливості сприятимуть підвищенню організованості та зручності для користувачів системи. Завдяки можливості швидко і зручно реєструватися на програми, додавати гравців до команд та керувати розкладом занять, користувачі зможуть ефективніше планувати свій час та ресурси. Автоматизована система сповіщень значно полегшить роботу тренерів, знизивши їхнє навантаження на комунікаційні завдання і дозволяючи їм зосередитися на основних тренувальних процесах. У підсумку, ці функції забезпечать більш високий рівень організації та зручності для всіх учасників футбольних секцій.

Першочерговим завданням було створення бази даних Postgres за допомогою ORM фреймворку Django, що містить інформацію про користувачів системи, таких як тренери, гравці, опікуни, менеджери та адміністратори, а також про програми, секції, заняття та команди. Ця база даних повинна бути надійною, ефективною та легко масштабованою, щоб забезпечити безперебійну роботу системи навіть при збільшенні кількості користувачів та обсягу даних. База даних повинна підтримувати операції з додавання, редагування та видалення даних, забезпечуючи при цьому цілісність та узгодженість інформації. Це означає, що будь-які зміни в даних повинні бути автоматично синхронізовані, щоб уникнути розбіжностей та помилок.

Наприклад, коли новий гравець реєструється в системі, його дані повинні бути негайно доступні для тренерів та менеджерів секцій, які зможуть включити його в відповідні програми та команди. Якщо гравець або тренер оновлює свої дані, зміни повинні бути відображені в реальному часі у всій системі. Це забезпечить, що вся інформація залишається актуальною і точною, що є критичним для ефективного управління футбольними секціями.

Крім того, база даних забезпечує надійний захист даних користувачів, включаючи їх особисту інформацію та дані про їх діяльність. Для цього необхідно впровадити сучасні методи шифрування даних, а також механізми контролю доступу, які дозволяють тільки уповноваженим користувачам отримувати доступ до певної інформації. Це допоможе запобігти несанкціонованому доступу та захистити конфіденційну інформацію користувачів від потенційних загроз.

Було забезпечено підтримку різних типів запитів до бази даних, таких як пошук, фільтрація та сортування даних, щоб користувачі могли швидко знаходити необхідну інформацію. Це може включати пошук гравців за іменем, фільтрацію програм за датою або сортування занять за місцем проведення. Таким чином, добре спроєктована база даних є основою для ефективної роботи всієї системи, забезпечуючи швидкий доступ до точної та актуальної інформації для всіх користувачів.

Кожна секція повинна прив’язана до конкретної фізичної локації футбольного поля з вказівкою точної адреси та відповідального менеджера. Це забезпечить відповідність і точність розташування для проведення тренувань і заходів, а також дозволить уникнути плутанини з місцями проведення занять. Важливо, щоб ця інформація була доступною в базі даних для всіх користувачів системи, що дозволить тренерам і гравцям легко знаходити необхідну секцію та планувати свої дії відповідно до місця проведення занять. Відповідальний менеджер секції буде відігравати ключову роль у координації діяльності, контролю за станом футбольного поля, забезпеченні необхідного обладнання та ресурсів для тренувань.

Тренери можуть створювати та керувати навчальними програмами, планувати тренування та адаптувати розклад занять залежно від потреб. Це включає можливість додавання нових програм, редагування існуючих та управління списками учасників. Тренери також зможуть змінювати розклад занять залежно від потреб команди або індивідуальних гравців, враховуючи їхні особисті потреби та рівень підготовки. Це забезпечить гнучкість у плануванні та проведенні тренувань.

Гравці мають можливість брати участь у тренуваннях, записуючись на програми та входячи до складу команд. Молодші гравці матимуть призначених опікунів, які служитимуть контактною особою для екстрених випадків. Опікуни, які є батьками або легальними опікунами у випадку відсутності батьків, зможуть отримувати сповіщення про розклад тренувань, зміни в програмах та іншу важливу інформацію.

Особлива увага приділяється автоматизації процесів. Система повинна автоматично відправляти сповіщення електронною поштою гравцям та їхнім опікунам про зміни у розкладі занять, а також надавати статистику щодо участі в програмах. Це допоможе знизити навантаження на тренерів та менеджерів, дозволяючи їм зосередитися на організаційних та навчальних аспектах.

Автоматизовані сповіщення забезпечать своєчасне інформування гравців та опікунів про всі зміни, що відбуваються в розкладі занять, включаючи перенесення, скасування або зміну часу тренувань. Це значно знизить ймовірність пропуску занять та допоможе підтримувати високий рівень відвідуваності. Сповіщення будуть надходити безпосередньо на електронну пошту користувачів, що є зручним і надійним способом комунікації.

Автоматизація процесів також сприятиме підвищенню точності та надійності виконання завдань, зменшуючи ризик людських помилок. Це забезпечить більш стабільну та ефективну роботу системи, звільнивши тренерів та менеджерів від рутинних завдань і дозволивши їм зосередитися на більш важливих аспектах їхньої діяльності.

Для забезпечення надійності та безпеки даних система підтримує сучасні методи аутентифікації та авторизації користувачів. Використання JWT (JSON Web Tokens) для авторизації дозволить забезпечити безпечну передачу даних між клієнтською та серверною частинами системи. Токени генеруються сервісом Passage/1Password під час входу користувача в систему та зберігатимуться в браузері користувача, що забезпечить швидкий доступ до ресурсів системи без необхідності повторної аутентифікації на кожен запит.

Токени будуть дійсні протягом обмеженого часу, після закінчення якого користувач буде переведений в анонімний режим. Це забезпечить додатковий рівень безпеки, запобігаючи можливості несанкціонованого доступу до системи після закінчення сеансу користувача. Користувачам потрібно буде повторно аутентифікуватися, щоб отримати новий токен, що знизить ризик викрадення або несанкціонованого використання токенів.

Використання JWT також дозволить зберегти контекст користувача між різними сеансами, забезпечуючи зручність і безперервність роботи. Це включає збереження налаштувань користувача, прав доступу та іншої важливої інформації, що покращить загальний користувацький досвід.

Серверна частина додатку реалізована з використанням фреймворку Python Django, що забезпечує високу продуктивність та зручність розробки. Django, завдяки своїй модульній архітектурі, дозволяє створювати масштабовані та легко підтримувані додатки, де кожен модуль відповідає за певну функціональність. Це сприяє більшій гнучкості та повторному використанню коду, що значно прискорює процес розробки.

Використання Docker для віртуалізації процесів дозволить швидко встановлювати та розгортати додатки у будь-якому середовищі. Docker контейнеризує додаток разом з усіма його залежностями, що забезпечує стабільну роботу незалежно від середовища розгортання. Це значно спрощує процес налаштування та підтримки додатка, оскільки розробники можуть бути впевнені, що додаток буде працювати однаково на всіх етапах розробки, тестування та продуктивного розгортання.

Екосистема Django включає численні вбудовані компоненти, такі як система аутентифікації, адміністративний інтерфейс, ORM (Object-Relational Mapping), які забезпечують готові до використання рішення для стандартних задач. Це дозволяє розробникам зосередитися на логіці додатку, використовуючи перевірені рішення для стандартних задач, що значно підвищує продуктивність і знижує час розробки.

Крім того, Django підтримує створення API за допомогою Django REST Framework, що дозволяє легко створювати і керувати RESTful сервісами. Це забезпечує гнучкість у взаємодії з клієнтськими додатками і іншими сервісами, що підвищує загальну ефективність системи.

Супроводжуваність додатку означає, що код повинен бути добре документованим, структурованим та легко зрозумілим для інших розробників. Це включає написання чистого коду, використання коментарів та докладної документації, що описує функціональність системи, її архітектуру та інтерфейси.

Це значно полегшить процес підтримки та розвитку системи в майбутньому, зменшуючи час на навчання нових членів команди та знижуючи ймовірність помилок.

Переносимість додатку забезпечить можливість його роботи у різних сучасних браузерах (Chrome, Mozilla, Safari, Edge) без обмежень операційної системи. Це досягається використанням стандартних веб-технологій, таких як HTML5, CSS3 та JavaScript, що забезпечують сумісність з різними платформами. Тестування додатку в різних браузерах і на різних пристроях дозволить виявити та усунути можливі проблеми, забезпечуючи стабільну та однакову роботу системи на всіх підтримуваних платформах.

Таким чином, забезпечення доступності, супроводжуваності та переносимості є важливими аспектами розробки програмного забезпечення, що сприяють створенню надійного, зручного та універсального додатку, який буде ефективно працювати в будь-яких умовах.

2.2 Характеристики користувачів

Користувачами додатка є наступні групи: анонімні користувачі, адміністратори, області менеджери секцій, студенти (гравці), їхні батьки або опікуни, сертифіковані тренери.

Заради відповідності принципам OWASP, буде реалізовано керування доступом на основі ролей (англ. Role Based Access Control, RBAC) - розвиток політики вибіркового керування доступом, при якому права доступу суб'єктів системи на об'єкти групуються з урахуванням специфіки їх застосування, утворюючи ролі [7]. Ролі користувачів системи наведено у табл. 2.1.

Наприклад, у менеджерів секцій є доступ до програм власних секцій; у адміністраторів та менеджерів спортивних секцій більше прав, ніж у інших користувачів. Опис правил RBAC [7] наведено у табл. 2.2 та Додатку Г.

Кожна роль буде мати відповідний набір прав, що буде основою для створення авторизації користувачів за ролями.

Таблиця 2.1 - Ролі користувачів системи (таблиця виконана самостійно)

№	Роль	Опис
1	Адміністратор (Staff)	Роль з найбільшою кількістю доступів, що має доступ до Панелі Адміністрування (Django Admin) та CRUD всіх ресурсів (можливі винятки).
2	Менеджер секцій (Section Manager)	Користувач із наданим аккаунтом для адміністрування секцій у регіонах.
3	Тренер (Coach)	Людина, що тренує футбольні команди. Сертифікований в UAF (Українська асоціація футболу) тренер. Користувач, який зареєструвався та надав документальне підтвердження сертифікації. Приклад повноваження: “Сертифікований тренер має право створювати програми для набору студентів”.
4	Опікуни/Батьки (Guardian)	Люди, що є офіційними опікунами або батьками гравця/гравців та реєструють їх в системі.
5	Анонімний користувач	Користувач, який не увійшов в систему та має мінімальні права доступу до системи, наприклад, перегляд останніх новин від UAF, сторінок контактів та “Про нас”. Може мати можливість зареєструватися або залогінитися.
6	Гравець, або Студент (Player)	Гравці футбольних команд, зазвичай до 18 років (саме тому в системі існують опікуни), яких зареєстрували їхні Опікуни/Батьки (Guardian) та прийняли в команди Тренери (Coach). Всі дії за гравців в системі виконують їхні опікуни.

Авторизація за ролями поєднуватиметься з авторизацією за атрибутами (англ. Attribute-Based Access Controle, або ABAC), тобто спочатку буде перевірятися роль користувача, а потім приналежність користувача певній організації або наявність певних властивостей, притаманних їхньому профілю [7]. Наприклад, тренер ніколи не зможе редагувати інформацію про команди інших тренерів, що забезпечуватиметься RBAC, але тренер також не зможе створювати навіть власну команду, якщо вони не завантажили сертифікат (ABAC).

Кожній ролі відповідає набір повноважень, наданих системою. Перевірка даних повноважень буде важливою частиною CRUD, адже кожна операція буде супроводжуватися перевіркою ролі та повноважень користувача. Згідно OWASP, буде реалізовано принцип мінімальних повноважень: за замовченням, у

користувача немає права на виконання операції. Дане право надається явно прописаними правилами RBAC/ABAC.

Таблиця 2.2. - Повноваження користувачів (таблиця виконана самостійно)

№	Повноваження	Опис
1	Зареєструватися в системі (register)	Створити аккаунт/обліковий запис користувача з певною роллю для Role-based access control (RBAC).
2	Залогінитися (login)	Зайти в систему в свій обліковий запис.
3	CRUD Ресурсу “Команди” (Teams)	Створити, редагувати, видалити, переглянути ресурс “Команди” (Teams)
4	CRUD Ресурсу “Склад команди” (TeamsPlayers)	Створити, редагувати, видалити, переглянути ресурс “Склад команди” (TeamsPlayers)
5	CRUD Ресурсу “Заняття” (Lesson)	Створити, редагувати, видалити, переглянути ресурс “Заняття” (Lesson)
6	CRUD Ресурсу “Секції” (Sections)	Створити, редагувати, видалити, переглянути ресурс “Секції” (Sections)
7	CRUD Ресурсу “Програми” (Programs)	Створити, редагувати, видалити, переглянути ресурс “Програми” (Programs)
8	CRUD Ресурсу “Реєстрації на програми” (Enrollments)	Створити, редагувати, видалити, переглянути ресурс “Реєстрації на програми” (Enrollments)
9	CRUD Ресурсу “Призначені опікуни/батьки гравців” (GuardiansPlayers)	Створити, редагувати, видалити, переглянути ресурс “Призначені опікуни/батьки гравців” (GuardiansPlayers)
10	Вийти з системи (logout)	Вийти з системи, перейшовши в режим анонімного користування.

Режими, в яких може працювати ПЗ, є наступними:

- режим анонімного користувача: реєстрація, логін, перегляд сторінок “Про нас”, “Контакти”, інформація щодо працюючих тренерів, місця розташування секцій, програми та заняття;
- режим адміністратора: може виконувати повністю всі CRUD дії над всіма ресурсами, наприклад, створювати та видаляти програми, секції, команди, тощо;

- режим менеджера секцій: призначення тренерів власних футбольних секцій; перегляд та редагування команд власної секції;
- режим тренера: CRUD своїх власних ресурсів: команди, програми, уроки;
- режим опікунів/батьків: реєстрація власних підопічних (гравців) на програми.

Таким чином, програмна система підтримуватиме п’ять ролей, забезпечуючи різноманітну функціональність для кожної з них.

2.3 Функції продукту

Продукт надає як функціональність, яка доступна всім користувачам, так і функції, якими можуть користуватися тільки окремі ролі. Основні функції продукту наведені нижче.

Всі користувачі: перегляд змісту сторінок: Головна, “Про нас” та “Контакти”; перегляд, пошук, сортування та фільтрація згідно мокапів; зміна мови інтерфейсу (англійська, українська).

Роль “Анонімний користувач”: створювати обліковий запис (реєстрація); заходити в систему (логін).

Всі авторизовані користувачі: переглядати та оновлювати власний профіль; вийти з системи.

Роль “Адміністратор”: заходити в систему як адміністратор; створювати, редагувати та видаляти облікові записи інших користувачів; здійснювати створення, редагування та видалення всіх даних.

Роль “Менеджер секцій”: перегляд секцій менеджера (власних секцій); створення та редагування секцій менеджера (власних секцій); видалення секцій менеджера (власних секцій); накладання обмежень на видалення (неможливо видалити секції, за якими зареєстровані програми); призначення тренерів футбольних секцій менеджера (дана дія викликає відправлення повідомлення на електронну пошту тренера зі сповіщенням про призначення їхніх програм на футбольні секції); перегляд програм, заняття за якими проводяться у секції менеджера; перегляд інформації згідно мокапів; пошук за назвою; сортування;

фільтрування за ознакою опублікованості; друк звіту з програм, заняття за якими проводяться у секції менеджера); перегляд занять власної секції; перегляд команд, що займаються у секції менеджера; перегляд інформації згідно мокапів; сортування; фільтрація за ознакою наявності вільних місць; друк звіту з команд, що займаються у секції менеджера; перегляд профілів тренерів.

Роль “Тренер”: завантаження сертифікату Української асоціації футболу (УАФ) [8] на сторінці профілю (сертифікований тренер має право створювати програми для набору студентів); перегляд секцій, де проводяться заняття за програмою тренера; перегляд власних програм; перегляд інформації згідно мокапів; пошук за назвою; сортування; фільтрування за ознакою опублікованості; друк звіту з власних програм; реєстрації гравців на програми (призначення (реєстрація) та виключення гравців з власних програм, перегляд реєстрацій з сортуванням, фільтруванням за віком гравців, пошуком за іменем гравців); перегляд занять власної програми; створення та редагування власних занять (перенесення заняття відправлення повідомлення на електронну пошту гравців зі сповіщенням про перенесення заняття); видалення власних занять (накладання обмежень на видалення (неможливо видалити минулі заняття)); перегляд команд, що займаються у тренера; перегляд інформації; сортування; фільтрація за ознакою наявності вільних місць; друк звіту з команд, що займаються у тренера; створення та редагування команд власної програми (власних команд) - тільки сертифікований тренер має право створювати програми для набору студентів; видалення власних команд (накладання обмежень на видалення (неможливо видалити команди, де вже є гравці); реєстрації гравців у команди (призначення (реєстрація) та виключення гравців з власних команд - обмеження: не можна одного гравця зареєструвати у декілька команд, ні одночасно, ні в різний час); перегляд реєстрацій з сортуванням, фільтруванням за віком гравців, пошуком за іменем гравців; перегляд профілів опікунів та гравців команд, що займаються в тренера.

Роль “Опікун”: призначення опікунів підопічним (гравцям); реєстрація підопічних (гравців) на програми (в програмах є максимальна кількість гравців, що можуть записатися (показник конфігурується у момент створення програми).

Програмна система також має дозволяти формувати звіти. Буде створено наступний звіт: перелік програм з даними про тренування, реєстрації, цінами та датами початку і кінця - з фільтрацією за ознакою опублікованості (Додаток В).

Таким чином, планування конкретних функціональних вимог додатку дозволить ефективно організувати процеси розробки програмного забезпечення та тестування.

2.4 Нефункціональні вимоги

Нефункціональними вимогами серверної частини програмного продукту є безпека та продуктивність.

Безпека забезпечується хешуванням чутливої інформації, як-от, електронних адрес користувачів, а також використанням з'єднання https. Доступ до даних користувача обмежений ролями і сферою відповідальності інших користувачів.

Стосовно продуктивності, відомо, що клієнтський додаток не споживатиме багато обчислювальних ресурсів. Навантаження на серверний додаток планується бути мінімальним, принаймні протягом першого півроку.

На перші півроку планова максимальна кількість одночасних підключень - 100. С кожним наступним місяцем планується збільшувати даний показник на 10-15 підключень, через зростання кількості користувачів завдяки маркетинговим зусиллям. Таким чином, через рік після запуску продукту система має витримувати 200 одночасних користувачів.

Водночас, тривалість завантаження сторінки не має перевищувати 5 секунд за будь-яких умов (за будь-якої кількості одночасних користувачів).

Реалізація даних нефункціональних вимог серверної частини системи дозволить створити сучасний програмний застосунок, що буде відповідати найкращим стандартам галузі та запитам користувачів повною мірою.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування програмного забезпечення

Опис функціонального призначення програмної системи наведено на рис.3.1.

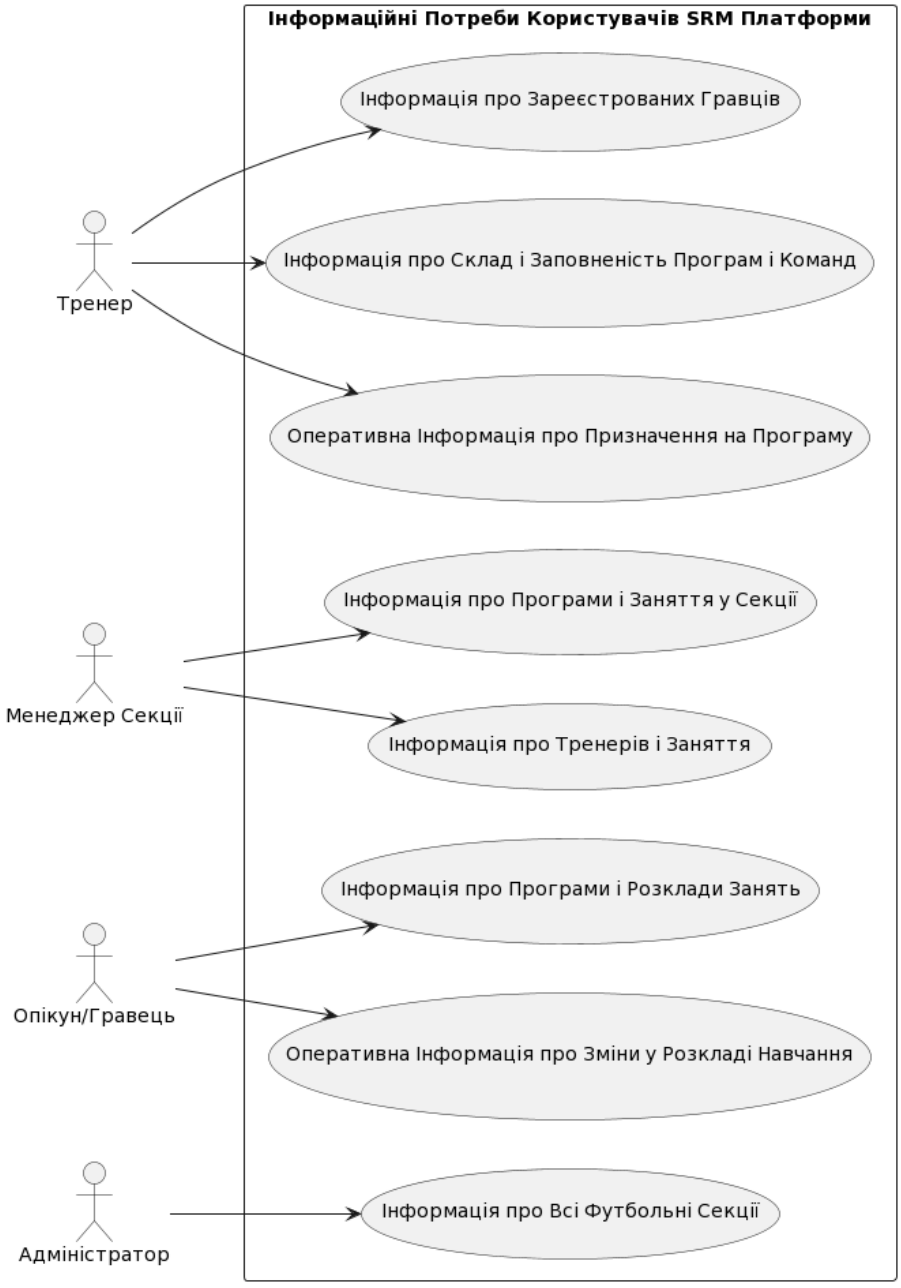


Рисунок 3.1 – Загальна USE-CASE діаграма для опису інформаційних потреб користувачів програмної системи (рисунок виконаний самостійно)

Зазначимо перелік характеристик для кожного з наведених понять. Поняття секції має наступні властивості: назва, регіон, адреса, призначений менеджер. Поняття “програми” включає в себе назву (наприклад, “Футбольний Фенікс”), опис

(наприклад, “Інтенсивна програма для вдосконалення всіх аспектів гри, від фізичної підготовки до психології перемоги.”), обмеження за віком (мінімальний та максимальний вік гравців), ціна в гривнях, ознаку опублікованості, дати старту та початку, дата та час початку та закінчення запису на програму, призначеного тренера, секцію, коментарі (зазвичай, залишені менеджером або тренером), а також вмістимість.

На підставі проведеного аналізу побудуємо загальну діаграму класів (див. рис.3.2).

Рисунок 3.2 – Загальна діаграма класів (рисунок виконаний самостійно)

Заняття мають зазначені атрибути: асоційовану програму, дату та час проведення. Поняття команди - назву та тренера. Поняття користувача – роль (тренер, гравець, опікун, менеджер, адміністратор), юзернейм, ім'я та прізвище, емейл, номер телефона, контакт та адресу опікуна (якщо існує), дату народження. Для спрощення роботи менеджерів під час роботи з інформацією вони повинні мати можливість: сортувати гравців за ім’ям; сортувати програми та команди за назвою, гравців у складі команди та зареєстрованих гравців - за ім’ям гравця - за

власною секцією; отримати перелік власних футбольних сесій з вказанням регіону; отримати перелік реєстрацій гравців на конкретну програму власної секції, включно з інформацією про їхніх опікунів.

Оскільки менеджер секцій або адміністратор призначає програму на секцію, існує необхідність сповіщати тренера про підтвердження секції для їхньої програми. Запропоновану автоматизацію зображено за допомогою діаграми станів (рис.3.3).

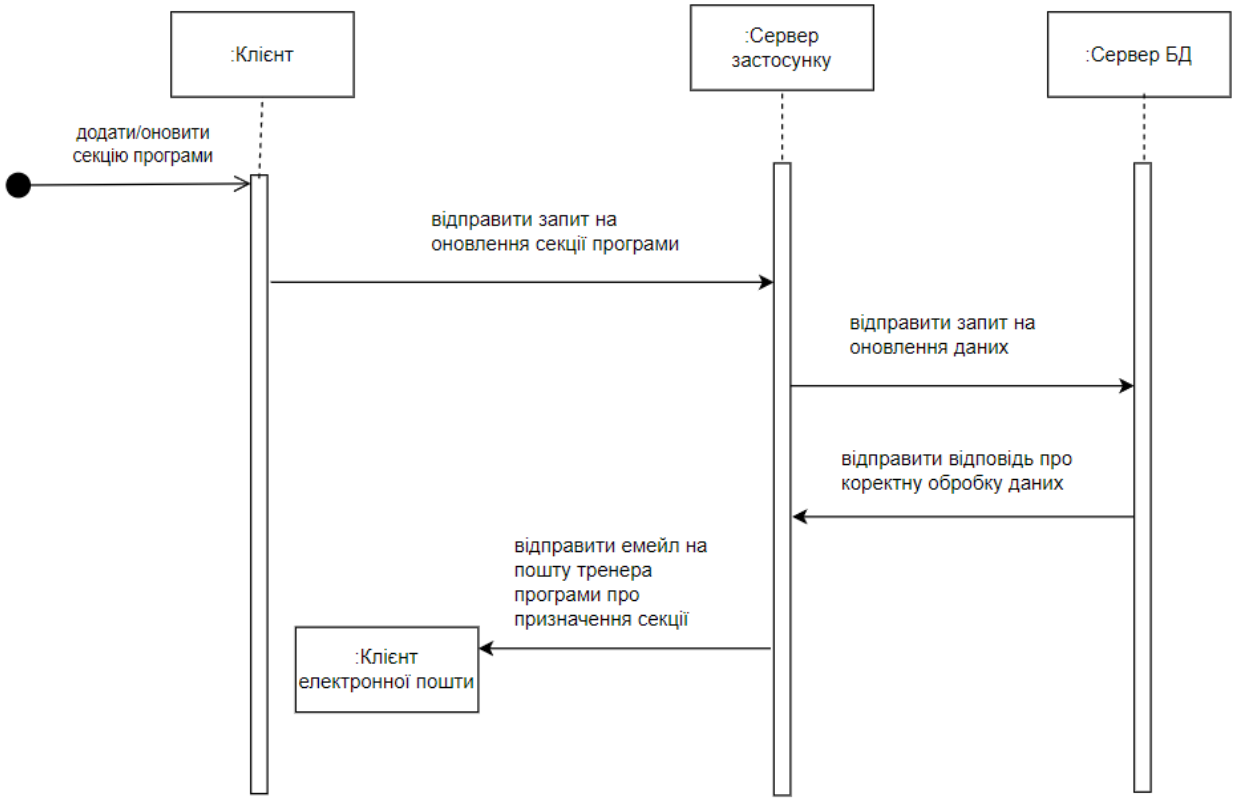


Рисунок 3.3 - Діаграма послідовностей рішення зі сповіщень тренерам про призначення їхніх програм на футбольні секції (рисунок виконаний самостійно)

Опис існуючого документообігу характеризується системою звітів. Звіт з програм - це перелік програм з даними про тренування, реєстрації, цінами та датами початку і кінця - з фільтрацією за ознакою опублікованості.

Лінгвістичні відносини, що існують в програмній системі, повинні нормалізуватися термінологією, що наведена нижче.

Управління відносинами у сфері спорту (англ. Sport Relationship Management, SRM) - це підвид управління відносинам зі споживачами, або CRM (Customer

Relationship Management) як бізнес-моделі та об’єкта досліджень, що спрямований на розвиток довгострокових відносин з усіма гравцями ринку/спільноти спорту/спортивних ігор.

Керування доступом на основі ролей (англ. Role Based Access Control, RBAC) - розвиток політики вибіркового керування доступом, при якому права доступу суб'єктів системи на об'єкти групуються з урахуванням специфіки їх застосування, утворюючи ролі.

Користувач - це особа чи організація, що користується функціональністю програмної системи та може мати обліковий запис в системі.

Тип користувача / Роль користувача - сутність, що визначає вид користувача та його привілеї, частина RBAC; можливі значення - гравець, опікун, тренер, менеджер (секцій), адміністратор.

Менеджер секцій - користувач програмної системи; особа, що займається адмініструванням футбольних секцій (перегляд та призначення тренерів на секції, підтвердження івентів з проведення занять/ігор, тощо).

Секція - регіональна або обласна організаційна одиниця, що має відповідне футбольне поле, назву, закріпленого за собою менеджера.

Програма - структурована навчальна програма, за якою закріплений тренер та яка складається з футбольних занять.

Заняття - урок з футболу, що проводить тренер.

Менеджмент футбольних секцій - це сукупність функціональності з адміністрування футбольних секцій їхніми менеджерами (наприклад, призначення тренерів на секції, управління заняттями).

Адміністрування даних - це сукупність функціональності адміністративного модулю, що передбачає надання адміністраторам програмної системи максимальних привілеїв на CRUD операції з даними.

Отже, в рамках даної роботи було проведено UML проєктування програмного забезпечення.

3.2 Проєктування бази даних

На даному етапі проектування БД доцільно за основу взяти загальну діаграму класів (див.рис.3.2). Була розроблена схема бази даних (Додаток Б).

Реляційна модель бази даних є найбільш ефективною для вирішення задач програмної системи, тому що саме дана модель підходить для додатків, які потребують складних запитів, транзакцій і можливості зберігати великі обсяги даних, які є структурованими в категорії, - все для забезпечення високого ступеню консистентності даних; SRM як підвид CRM є одним з поширених прикладів застосування реляційної моделі БД. Як бачимо на схемі, гравців, опікунів, менеджерів, тренерів та адміністраторів було нормалізовано шляхом об'єднання в одну сутність - користувачі (“users”), що розрізняються за роллю, або типом користувача (“user_type”).

В даному проекті при нормалізації виконується приведення таблиць до третьої нормальної форми для усунення з бази зайвих функціональних залежностей між атрибутами.

В усіх відношеннях всі атрибути є атомарними, тому вони знаходяться в першій нормальній формі (1НФ). При зведенні до другої нормальної форми було доведено, що всі відношення не містять неповних функціональних залежностей, тобто в складі потенційного ключа відсутня менша підмножина атрибутів від якої можна також вивести дану функціональну залежність.

При зведенні до третьої нормальної форми була проведена перевірка на відсутність транзитивних функціональних залежностей. Отже, отримана схема БД знаходиться в 3НФ.

3.3 Архітектура серверної частини застосунку

Програмним продуктом виступає програмна система - веб-сервіс, що надає можливість організовувати програми для навчання грі в футбол, онлайн пошуку та комунікації тренерів і гравців, а також адмініструвати футбольні секції їхнім менеджерам. Архітектурна схема наведена на рис.3.4.

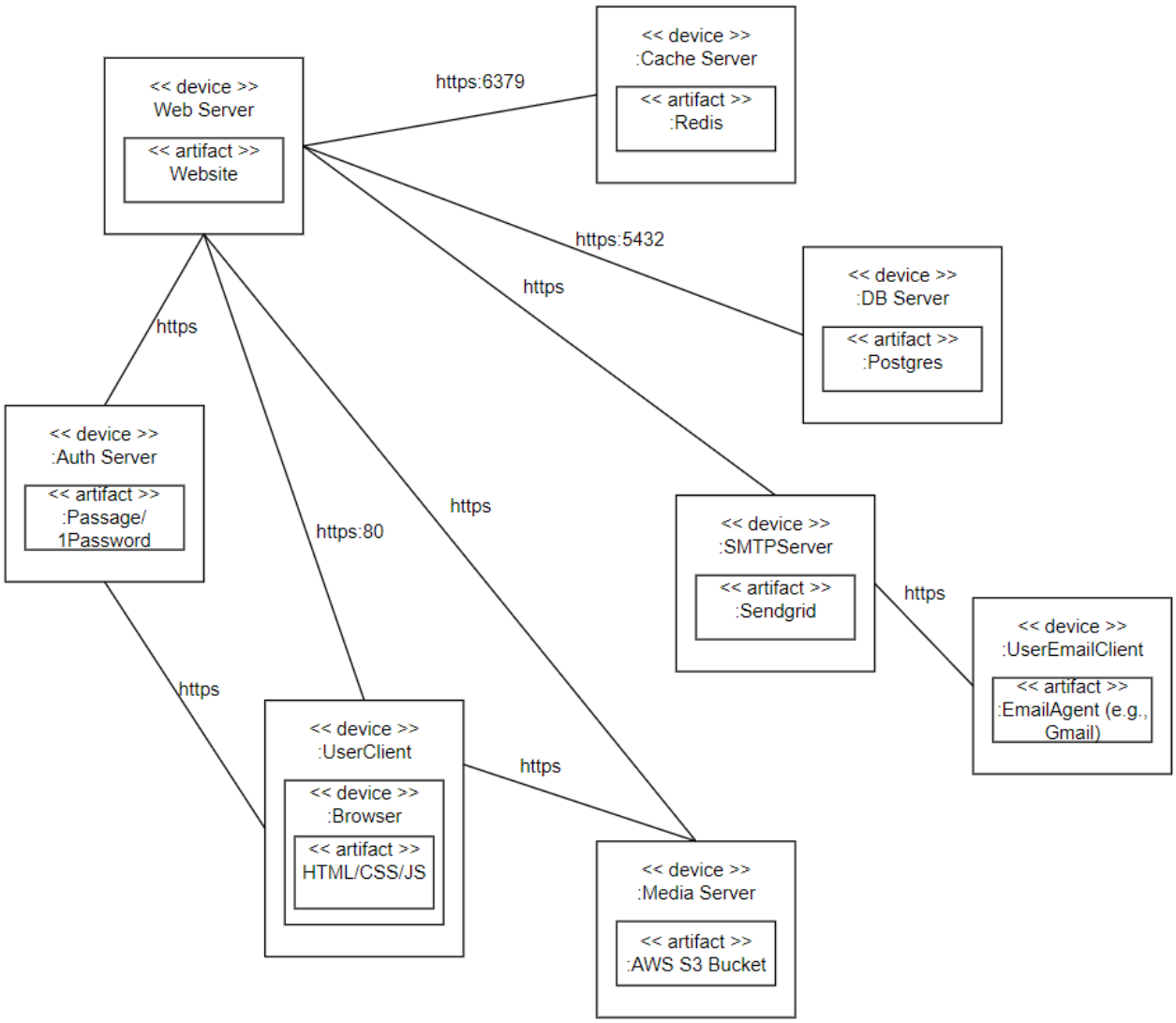


Рисунок 3.4 - Діаграма розгортання (рисунок виконаний самостійно)

В якості веб-фреймворку для розробки серверної частини (бекенду) було використано Django [9], перевагами якого є висока швидкість розробки, легка та швидка інтеграція з іншими технологіями, як-от, Celery [10]; наявність великої кількості бібліотек; обов'язковість модульної організації коду у застосунки (applications), що підвищує якість коду; наявність інтерфейсу адміністрування Django Admin, а також Object Relational Mapper (ORM) для написання SQL запитів для реалізації функціональності CRUD [9].

Додаток націлено на взаємодію з базами даних, отже для їх створення було обрано систему управління реляційними базами даних PostgreSQL. Саме ця СУБД була використана через її здатність забезпечувати високу цілісність даних, легкість

використання та простоту конфігурування. За допомогою налаштувань сеттінгів Django та утіліти DataGrip було здійснено управління і конфігурування компонентів PostgreSQL, виконання довільних SQL запитів, а також відладку та тестування SQL запитів.

Застосунок було спроектовано за специфікацією REST API / OpenAPI - абревіатура від REpresentational State Transfer (передача репрезентативного стану, що дозволяє застосункам бекенду та фронтенду обмінюватися даними через HTTP запити за узгодженими контрактами API [11,12]. Перевагами REST API є легкість у користуванні та швидкість інтеграції, забезпечення можливості паралельної незалежної розробки логіки даних на бекенді та фронтенді кількома розробниками завдяки наявності узгодженої специфікації, наявність досвіду роботи з даною специфікацією у автора даної роботи. Для документування та тестування REST API були використані утіліти Postman [13] та Swagger [14]. Для програмної реалізації бекенду використано бібліотеку з екосистеми Django - Django Rest Framework (DRF) [14].

При запуску застосунку використовується Docker - інструментарій для управління ізольованими Linux-контейнерами, а також та Docker Compose для керування (оркестрації) декількома контейнерами Docker одночасно [15]. Docker інструменти завдяки своїй ефективній технології віртуалізації дозволяють швидко встановлювати та розгортати застосунки та їхні залежності у будь-якому середовищі.

Розсилку листів на пошту гравців зі сповіщенням про перенесення дати/часу заняття реалізовано за допомогою бібліотеки Celery [10] для зменшення тривалості виконання HTTP запитів у REST API (завдяки переведенню операцій у неблокуючий режим у окремий поток - на background), а також сервісу Sendgrid. Celery - інструмент для створення розподілених черг задач. Такі черги дозволяють розвантажити роботу в інший процес і виконувати частину задач у фоновому режимі (якщо вони не залежать від подальших дій користувачів), поки застосунок виконує інші задачі [10]. Redis було використано як брокер та результуючий бекенд для Celery.

Завантаження сертифікатів тренерів, зображень користувачів та інших користувацьких медіа-файлів буде здійснено з використання хмарної технології Amazon S3 [16]: хмарне сховище буде зберігати власне медіа-файли, тоді як бекенд зберігатиме посилання на дані файли.

Електронні листи буде відправлено за допомогою бібліотеки Sendgrid REST API (альтернатива SMTP SDK) та Sendgrid для Django - “django-sendgrid-v5” [17].

Аутентифікацію та авторизацію та буде проведено за допомогою технологій “аутентифікації без паролю” (англ. passwordless) та JSON Web Token (JWT). Аутентифікація буде розроблена на клієнтській частині, що взаємодіє з сервісом Passage/1Password [18]. Авторизація користувачів виконуватиметься на серверній частині завдяки валідації JWT, який міститься у всіх запитах до бекенду, через Passage/1Password та бібліотеку “passage-python” [19].

Логічна структура програмної системи складатиметься з серверної частини (бекенд) та клієнтської частини (фронтенд). Прототип дерева проєкту наведено на рис. 3.5.

Бекенд, який реалізовано за допомогою веб-фреймворка Django, складається з наступних програмних модулів (однойменних Python пакетів, що є за своєю суттю Django застосунками/Django apps) та інших наведених складових елементів.

App “users” (англ. “app”, або “application” - Django застосунок системи, модуль) - модуль структури даних про користувачів (однойменний Python-пакет). Приклад даних зображено на рис. 3.6.

App “sports”: модуль структури даних про навчання футболу (програми, команди, заняття, тощо).

App “api”: модуль, що містить логіку REST API, реалізованої за допомогою Django RESTful Framework. Містить документацію Swagger.

App “frontend” - модуль, що містить логіку URL роутінгу фронтенду в Django, а також власне клієнтський код (Javascript).

App “background”, що містить задачі, виконувані в окремому процесі - у неблокуючому режимі, як-от, розсилку електронних листів (реалізовано за допомогою Celery [10]).

```
├── AUTHORS
├── Dockerfile    <-- логіка побудови Docker середовища
├── LICENCE-DEPENDENCIES.md
├── LICENSE.md
├── README.md            <-- зведена технічна документація проєкту
├── api
├── babel.config.js
├── docker-compose.yml<-- оркестрація Docker контейнерів
├── background        <-- задачі, що виконуються в окремому процесі
├── frontend
├── load_fixtures.sh <-- скрипт з заповнення БД тестовими даними
├── dump_fixtures.sh
├── manage.py
├── node_modules
├── package-lock.json
├── package.json  <-- JavaScript залежності (бібліотеки)
├── postcss.config.js
├── requirements  <-- Python залежності (бібліотеки)
├── resources
├── sports
├── srm_platform  <-- містить конфігураційні файли проєкту
├── static        <-- автозбірка JS/CSS - вимагає Django
├── templates
├── users
├── webpack.config.dev.js
└── webpack.config.js
```

Рисунок 3.5 – Прототип дерева (модульної структури) програми (рисунок виконаний самостійно)

Project setup “srm_platform”: конфігурація рівню проєкту (Python модуль, або файл “srm_platform/settings.py”).

Django admin, логіку якого містить кожен програмний модуль, що керує даними, тобто має логіку “models.py” (модулі “admin.py”).

Для втілення даного додатку буде використано середовища розробки JetBrains (PyCharm та DataGrip), мови програмування та високорівневі фреймворки Python (для програмування серверної частини, або бекенду) та Javascript (для розробки фронтенду), а також високорівневі фреймворк Django.

Серверна частина проєкту також використовує Gunicorn, HTTP сервер Python WSGI для UNIX, що дозволяє управляти пулінгом баз даних (англ. DB Pooling) та покращує продуктивність застосунку в цілому.

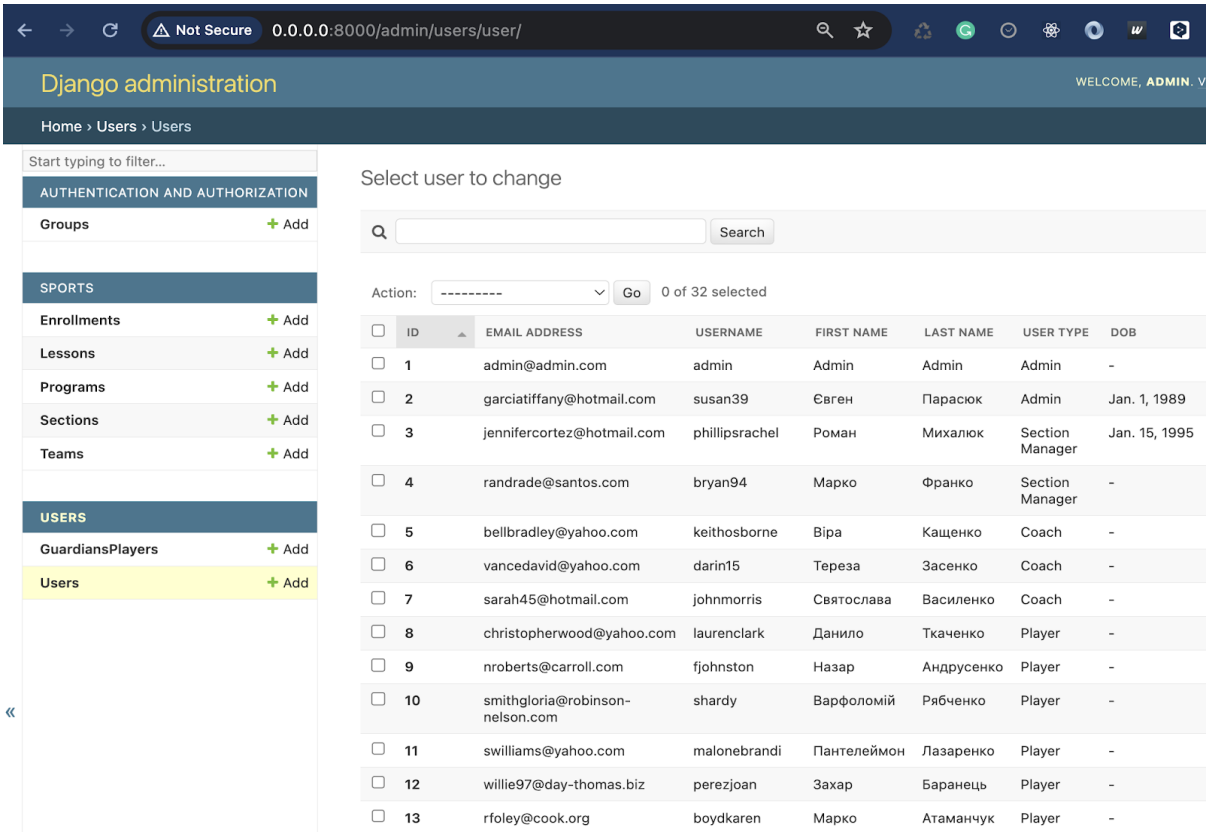


Рисунок 3.6 – Прототип відображення даних модулю “users” в інтерфейсі супер-адміністратора Django Admin (рисунок виконаний самостійно)

Таким чином, було наведено детальний опис програмної реалізації та екосистеми розробки серверної частини програмної системи.

3.4 Приклади алгоритмів та методів

В програмній системі використано MD5 алгоритм для хешування електронних адрес користувачів, які виступають значенням параметру запиту REST API. Даний запит спрямовано на отримання даних користувача з бази даних за їхньої реєстрації або логіну: клієнтська сторона застосунку має зберегти та тримати дані користувача (як-от, ім’я та прізвище для їхнього відображення в графічному інтерфейсі) в браузері протягом всієї тривалості сесії.

Алгоритм шифрування MD5 (англ. Message Digest Algorithm 5) складається з кількох основних етапів, які включають обробку вхідного повідомлення, перетворення його на заздалегідь визначену довжину та генерацію вихідного хешу.

Спочатку вхідне повідомлення розширюється таким чином, щоб його довжина стала кратною 512 бітам, при цьому додається одиничний біт, а потім достатня кількість нульових бітів, щоб досягти потрібної довжини. Потім додається 64-бітове представлення початкової довжини повідомлення. Наступний крок полягає у розбитті розширеного повідомлення на 512-бітні блоки, кожен з яких проходить через чотири основні цикли обробки. У кожному циклі використовується спеціальна побітова операція, зміна порядку байтів і додавання констант. Алгоритм включає чотири етапи обробки: розширення повідомлення, ініціалізація буферів, основна петля обробки та збирання вихідного хешу. Кожен з цих етапів містить специфічні операції, такі як додавання попередніх значень, побітові операції AND, OR, XOR та додавання констант. На завершальному етапі всі блоки повідомлення комбінуються, формуючи остаточний 128-бітовий хеш.

Фрагмент програмної реалізації застосування алгоритму хешування MD5 наведено нижче.

```
from hashlib import md5


class User(AbstractUser, CommonInfo):
    """Custom user model."""

    email = models.EmailField(_("email address"), unique=True)
    email_md5 = models.CharField(max_length=64, unique=True, editable=False,
null=True)

      def save(self, *args, **kwargs):
          self.email_md5 = md5(str(self.email).encode("utf8")).hexdigest()
          super().save(*args, **kwargs)
```

Для програмної реалізації хешування електронної адреси користувача було використано вбудовану Python бібліотеку hashlib. Django модель «Користувач», що відповідає таблиці бази даних “users”, містить спеціально призначене поле – “email_md5”. Дане поле являє собою MD5 хеш оригінального поля – “email” (власне електронної адреси), який визначається кожного разу, коли створюється новий запис в таблиці “users” (тобто коли реєструється новий користувач) – та зберігається в базі даних.

Збереження MD5 хешу в окремому полі дозволяє зекономити ресурси системи на обчисленні MD5 хешу кожного разу, коли здійснюється запит на отримання даних користувача за певною електронною адресою.

Таким чином, MD5-хешування електронних адрес дозволяє підвищити безпеку користувацьких даних, адже хеш забезпечує односторонню функцію, яка гарантує унікальність для різних вхідних даних і є основою криптографічної стійкості MD5.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Обґрунтування вибору викокорівневого фреймворку

Наряду з фреймфорком Django було досліджено фреймфорк Flask, в результаті чого було обрано Django з кількох важливих причин, які випливають із потреб проекту і особистих вподобань в роботі з веб-фреймворками. Перш за все, Django пропонує комплексний набір можливостей "з коробки". Це включає в себе адміністративну панель, ORM для роботи з базами даних, вбудовану систему автентифікації користувачів, форми і валідацію, а також багато інших функцій. Така функціональність дозволяє значно скоротити час розробки, оскільки не потрібно витрачати додатковий час на інтеграцію окремих компонентів або написання основних функцій з нуля.

Окрім цього, Django надає чітку і добре задокументовану структуру проекту, що значно полегшує підтримку і розширення коду. Ця структура дозволяє розробникам слідувати усталеним шаблонам і практикам, що допомагає уникнути плутанини та сприяє більш ефективній розробці. Такий підхід особливо важливий для великих проектів або для командної роботи, де кілька розробників одночасно працюють над одним і тим же кодом. Чітка організація коду допомагає уникати непорозумінь і забезпечує, щоб усі учасники проекту розуміли, як працює кожен його компонент.

Flask, будучи мікрофреймворком, надає більше гнучкості і свободи в організації проекту. Це означає, що розробники мають можливість самостійно вирішувати, як структурувати свій проект і які інструменти використовувати. Така гнучкість може бути корисною для невеликих проектів або для досвідчених розробників, які знають, як ефективно організувати код. Однак, це також означає більше відповідальності за налаштування і інтеграцію додаткових бібліотек та інструментів. Наприклад, для побудови функціоналу автентифікації у Flask потрібно підключати додаткові розширення, тоді як у Django це вбудовано з самого початку, що значно спрощує завдання розробників.

Важливим аргументом на користь Django є його масштабованість. Цей фреймворк добре підходить для побудови як невеликих веб-додатків, так і великих, високонавантажених систем. Завдяки своїм можливостям, таким як кешування, підтримка різних баз даних і вбудовані механізми захисту від поширених веб-загроз, Django забезпечує високу продуктивність і безпеку. Кешування дозволяє зменшити навантаження на сервер і покращити швидкість роботи додатка. Підтримка різних баз даних дозволяє розробникам вибирати найбільш підходящу для їх потреб СУБД, а вбудовані засоби захисту допомагають запобігати атакам на додаток.

Flask, хоча і може бути масштабованим, вимагає більшої роботи для досягнення тих же рівнів продуктивності та безпеки. Для цього розробникам доводиться додатково налаштовувати кешування, інтегрувати засоби захисту та підключати бібліотеки для роботи з базами даних. Наприклад, вбудовані у Django засоби захисту від XSS та CSRF атак значно полегшують завдання забезпечення безпеки додатку. Ці механізми автоматично захищають додаток від найбільш поширених загроз, що дозволяє розробникам зосередитися на інших аспектах проекту.

Крім того, Django має активну і розвинену спільноту, що забезпечує швидку підтримку та доступ до численних ресурсів. Це включає в себе документацію, плагіни, пакети і відповіді на форумах. Наявність великої кількості плагінів та бібліотек означає, що багато завдань можна вирішити швидко і ефективно без необхідності писати код з нуля. Наприклад, якщо розробник стикається з певною проблемою, він може знайти готове рішення у вигляді плагіна або бібліотеки, що значно скорочує час розробки.

Спільнота Flask також активна, але через більшу гнучкість фреймворку можна стикнутися з різноманітними підходами до розв'язання тих самих задач. Це може ускладнити процес розробки, особливо для новачків, які можуть бути не впевнені, який підхід вибрати. Наприклад, для інтеграції з різними базами даних у Flask потрібно вручну вибирати і налаштовувати відповідні бібліотеки, тоді як у Django все це вже інтегровано і добре задокументовано.

Для тих, хто цінує стандартизацію і чіткі правила, Django є кращим вибором. Фреймворк забезпечує єдині підходи до вирішення типових завдань, що сприяє зменшенню кількості помилок і покращенню якості коду. Розробники, які працюють з Django, можуть бути впевнені, що їх код буде відповідати загальноприйнятим стандартам і кращим практикам.

З іншого боку, розробники, які люблять експериментувати і шукати нестандартні рішення, можуть віддати перевагу Flask. Гнучкість цього мікрофреймворку дозволяє створювати унікальні архітектури і використовувати найсучасніші технології. Однак, це також означає, що розробники повинні бути готові до більшої відповідальності і мати глибокі знання у сфері розробки.

Одним із головних критеріїв вибору була потреба у швидкому запуску проекту. Django дозволяє швидко створювати MVP (Minimum Viable Product) завдяки своїм вбудованим інструментам і генераторам коду. Це особливо важливо для стартапів або проектів з обмеженими термінами, де кожна хвилина на рахунку. Flask, з його мінімалістичним підходом, також може бути використаний для швидкого створення прототипів, але це потребує більше часу на налаштування і інтеграцію необхідних компонентів.

Не менш важливим є фактор підтримки і масштабування команди. Django забезпечує високий рівень стандартизації, що полегшує входження нових розробників у проект. Це знижує криву навчання і дозволяє новим членам команди швидко приступати до роботи. Flask, з його гнучкістю, може викликати певні труднощі у нових розробників, оскільки кожен проект може мати унікальну структуру і підходи до вирішення типових завдань.

Авторка також врахувала необхідність підтримки старих версій проекту і можливість легкого оновлення до нових версій. Django дотримується чіткої стратегії випуску оновлень і підтримки старих версій, що забезпечує стабільність і передбачуваність розвитку проекту. Flask, хоча і розвивається активно, не завжди надає таку ж стабільність і передбачуваність, що може створювати додаткові ризики при довготривалій підтримці проекту.

Загалом, вибір на користь Django обумовлений прагненням до швидкого, надійного і масштабованого розвитку веб-додатків. Для проектів, які потребують швидкого запуску з мінімальними налаштуваннями і готовими до використання функціями, Django є ідеальним вибором. Він дозволяє розробникам зосередитися на основних бізнес-логіках і функціональності, тоді як Flask більше підходить для проектів, де потрібна максимальна гнучкість і контроль над кожним аспектом розробки.

4.2 Обґрунтування вибору бази даних

Авторка обрала PostgreSQL замість MySQL, MariaDB, Oracle або SQL Server з кількох ключових причин. По-перше, PostgreSQL відомий своєю відповідністю стандартам SQL і потужними функціональними можливостями. Він підтримує розширені типи даних, такі як JSONB для зберігання і обробки JSON даних, що робить його ідеальним для сучасних веб-додатків, які потребують гнучкості у роботі з даними. Ця функціональність особливо корисна для додатків, що потребують зберігання напівструктурованих даних або інтеграції з різними API.

Крім того, PostgreSQL має потужну систему розширень, що дозволяє розширювати функціональність бази даних без необхідності модифікації ядра. Наприклад, розширення PostGIS додає підтримку географічних інформаційних систем (ГІС), що дозволяє зберігати і обробляти просторові дані. Такі можливості роблять PostgreSQL універсальним вибором для різних типів додатків, від стандартних CRUD-додатків до складних аналітичних систем.

Ще однією важливою причиною вибору PostgreSQL є його відмінна підтримка транзакцій і забезпечення цілісності даних. PostgreSQL підтримує ACID-транзакції, забезпечуючи надійність і стабільність роботи з даними навіть у разі збоїв або високого навантаження. Це особливо важливо для фінансових додатків або будь-яких інших систем, де важлива точність і надійність даних.

Порівнюючи з MySQL, PostgreSQL пропонує більше можливостей для налаштування і оптимізації. Хоча MySQL є популярним вибором завдяки своїй

швидкості і простоті налаштування, PostgreSQL перевершує його у багатьох аспектах, особливо коли мова йде про складні запити і великі обсяги даних. MariaDB, будучи форком MySQL, успадкувала багато його переваг і недоліків, тому для більш складних сценаріїв PostgreSQL залишається кращим вибором.

Oracle і SQL Server, хоча і є потужними системами керування базами даних (СКБД), мають свої обмеження, зокрема ліцензійні обмеження і високу вартість володіння. PostgreSQL, будучи відкритим і безкоштовним, надає розробникам можливість використовувати потужну СКБД без додаткових фінансових витрат. Це особливо важливо для стартапів або невеликих компаній, які обмежені у бюджеті, але потребують надійного і функціонального рішення для зберігання даних.

Ще однією перевагою PostgreSQL є його активна спільнота і велика кількість доступних ресурсів для навчання і підтримки. Це забезпечує швидкий доступ до оновлень, плагінів і відповіді на будь-які питання, що можуть виникнути під час розробки і експлуатації. У порівнянні з Oracle і SQL Server, де підтримка може бути дорогою і обмеженою, PostgreSQL надає відкриту і доступну платформу для всіх розробників.

З точки зору продуктивності, PostgreSQL демонструє відмінні результати у високонавантажених системах. Завдяки ефективному використанню ресурсів і можливостям масштабування, PostgreSQL здатний обробляти великі обсяги даних і високі навантаження з мінімальними затримками. Це робить його чудовим вибором для проектів, які потребують високої продуктивності і надійності.

Python має високоякісні бібліотеки, що дозволяють використовувати PostgreSQL, як-от,

Враховуючи всі ці фактори, вибір на користь PostgreSQL був зроблений з огляду на його потужні функціональні можливості, надійність, відповідність стандартам і відкритість. Для проектів, які потребують гнучкого, масштабованого і стабільного рішення для роботи з даними, PostgreSQL є ідеальним вибором. Він дозволяє зосередитися на розробці бізнес-логіки додатку, залишаючи питання зберігання і обробки даних на надійну і перевірену платформу.

4.3 Обґрунтування вибору хмарного рішення зберігання файлів

Було обрано Amazon S3 для зберігання медіа-контенту замість Azure Storage Account (blob), Google Cloud Storage Buckets або інших рішень, виходячи з кількох вагомих причин. Amazon S3 (Simple Storage Service) є одним із найвідоміших і найнадійніших рішень для зберігання об'єктів, і він надає широкий спектр функцій, які задовольняють різноманітні потреби у зберіганні медіа-контенту.

Amazon S3 використано для збереження зображень профілів користувачів та сертифікатів тренерів. Фрагмент реалізації завантаження файлу наведено нижче (з використанням Python бібліотеки «boto»):

```
class FileUploadView(HandleAPIExceptionMixin, views.APIView):

    def put(self, request, format=None):
        file_ = request.FILES["file"]
        now_timestamp = f"{time()}".replace(".", "")
        filename_no_ext = os.path.splitext(file_.name)[0]
        extension = file_.name.split(".")[1].lower().replace(".", "")
        filename = f"{filename_no_ext}-{now_timestamp}.{extension}"
        contents = file_.read()

        try:
            client = boto3.client(
                "s3",
                aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
                aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY,
            )
            client.put_object(
                Body=contents,
                Bucket=settings.AWS_STORAGE_BUCKET_NAME,
                Key=filename,
            )
        except Exception as exc:
            msg = f"An error happened when uploading a file."
            raise ApiCustomException(message=msg)

        return Response(
            status=status.HTTP_200_OK,
            data={
                "file": os.path.join(settings.MEDIA_PATH, filename)
            }
        )
```

Перш за все, Amazon S3 забезпечує надзвичайно високу надійність і доступність даних. Завдяки реплікації даних у кількох зонах доступності (Availability Zones), дані зберігаються надійно і завжди доступні навіть у разі відмови однієї з зон. Це особливо важливо для додатків, які потребують

безперебійного доступу до медіа-контенту і не можуть дозволити собі втрати даних.

Ще однією важливою перевагою Amazon S3 є його масштабованість. Сервіс дозволяє зберігати необмежені обсяги даних і автоматично масштабуватися в залежності від потреб додатку. Це означає, що незалежно від того, чи потрібно зберігати декілька гігабайтів або петабайти медіа-контенту, Amazon S3 впорається з цим завданням без необхідності додаткових налаштувань або втручань з боку розробників.

Amazon S3 також відзначається своєю безпекою. Сервіс надає вбудовані механізми шифрування даних як під час передачі, так і під час зберігання. Крім того, політики контролю доступу дозволяють точно налаштовувати, хто і які дії може виконувати з медіа-файлами, забезпечуючи високий рівень захисту від несанкціонованого доступу.

Azure Storage Account (blob) і Google Cloud Storage Buckets також є потужними і надійними рішеннями для зберігання даних, але вони мають свої особливості і обмеження. Azure Storage добре інтегрується з іншими сервісами Microsoft і підходить для проектів, які активно використовують екосистему Microsoft. Проте, для проектів, які не обмежуються інфраструктурою Microsoft, Amazon S3 надає більше гнучкості.

Google Cloud Storage пропонує високий рівень продуктивності і надійності, але для багатьох проектів Amazon S3 є більш зручним вибором завдяки своїй популярності і широкій підтримці в різних інструментах і платформах. Багато фреймворків і бібліотек мають вбудовану підтримку Amazon S3, що полегшує інтеграцію і використання сервісу.

Важливо також відзначити, що Amazon S3 пропонує конкурентоспроможну модель ціноутворення. Користувачі сплачують лише за ті ресурси, які вони використовують, що дозволяє ефективно керувати витратами. Гнучка модель оплати на основі обсягу збережених даних, кількості запитів і обсягу переданих даних робить Amazon S3 вигідним вибором як для невеликих проектів, так і для великих компаній з високими вимогами до зберігання даних.

Однією з ключових функцій, яка вигідно відрізняє Amazon S3, є його можливість інтеграції з іншими сервісами AWS (Amazon Web Services). Наприклад, інтеграція з Amazon CloudFront дозволяє легко налаштувати глобальну мережу доставки контенту (CDN) для швидкої і ефективної доставки медіа-контенту користувачам по всьому світу. Інтеграція з AWS Lambda надає можливість автоматизації обробки файлів без необхідності запуску окремих серверів.

Загалом, вибір на користь Amazon S3 був обумовлений прагненням до забезпечення високої надійності, масштабованості і безпеки зберігання медіа-контенту. Це рішення забезпечує гнучкість і ефективність, що дозволяє розробникам зосередитися на основних завданнях додатку, знаючи, що їх дані зберігаються надійно і завжди доступні. Враховуючи всі ці переваги, Amazon S3 є оптимальним вибором для зберігання медіа-контенту у сучасних веб-додатках.

4.4 Обґрунтування вибору сервісу розсилки повідомлень

Для відправки електронної пошти замість MailChimp або Amazon SES було обрано SendGrid, виходячи з кількох ключових причин. SendGrid, будучи спеціалізованою платформою для управління електронною поштою, пропонує широкий спектр функцій, які задовольняють потреби різних типів користувачів, від малого бізнесу до великих корпорацій.

По-перше, SendGrid забезпечує високу надійність і доставлюваність листів. Платформа має потужну інфраструктуру, яка дозволяє обробляти великі обсяги листів без затримок. Це особливо важливо для бізнесів, які відправляють великі обсяги транзакційних і маркетингових листів і не можуть дозволити собі втрати або затримки в доставці.

Ще однією перевагою SendGrid є його зручний і інтуїтивно зрозумілий інтерфейс. Інструменти для створення листів, налаштування кампаній та аналізу результатів розроблені таким чином, щоб навіть користувачі без технічного досвіду могли легко використовувати їх. Крім того, SendGrid надає потужний API, який

дозволяє інтегрувати функціонал відправки електронної пошти у будь-який додаток або систему з мінімальними зусиллями.

Sendgrid використано для розсилки електронних повідомлень опікунам, що сповіщають про зміну розкладу занять. Фрагмент реалізації наведено нижче:

```
@app.task()
def send_email_lesson_rescheduled(**kwargs) -> None:
    """Send email to players (recipients) whose lesson got rescheduled."""
    lesson_id = kwargs.get("lesson_id")
    event_datetime = kwargs.get("event_datetime")
    program_id = kwargs.get("program_id")

    enrollments = (
        Enrollment.objects.filter(program_id=program_id)
        .order_by("player__email")
    )

    if enrollments:
        program_name = enrollments[0].get("program__name")
        recipients = tuple([obj.get("player__email") for obj in enrollments])

        send_email(
            subject="Заняття перенесено",
            message=f"Увага! Перенесено заняття #{lesson_id} програми
'{program_name}'! Новий час: {event_datetime}",
            recipients=recipients,
        )
```

На панелі адміністрування Sendgrid було створено ресурс, який було прив’язано до розроблюваної програмної системи. Кожного разу, коли на серверній частині спрацьовує певний тригер (наприклад, зміна розкладу футбольних занять або призначення секції програми), Sengrid відправляє вказаним адресатам сповіщення на запрограмовані електрону адрес з текстом, який також запрограмовано на серверній частині застосунку.

У порівнянні з MailChimp, який більше орієнтований на маркетингові кампанії і пропонує потужні інструменти для створення і управління розсилками, SendGrid забезпечує більшу гнучкість і функціональність для відправки транзакційних листів. MailChimp дійсно є відмінним вибором для маркетологів, які потребують розширених інструментів для створення красивих і ефективних розсилок. Однак, для бізнесів, які потребують надійного рішення для відправки великого обсягу транзакційних повідомлень, SendGrid є кращим вибором.

Amazon SES (Simple Email Service) також є потужним рішенням для відправки електронної пошти, але його використання може вимагати більше технічних знань і налаштувань. Amazon SES пропонує високу масштабованість і низьку вартість, що робить його привабливим для великих компаній з великими обсягами розсилок. Однак, для менш технічно підкованих користувачів або тих, хто шукає більш зручний і готовий до використання інструмент, SendGrid є більш підходящим варіантом.

Ще однією важливою причиною вибору SendGrid є його потужні інструменти для аналітики і моніторингу. Платформа надає детальні звіти про доставленість листів, відкриття, кліки і інші важливі метрики, що дозволяє користувачам легко відстежувати ефективність своїх кампаній і робити відповідні коригування. Це особливо корисно для бізнесів, які хочуть максимізувати ефективність своїх розсилок і покращити взаємодію з клієнтами.

Крім того, SendGrid має розширені можливості для налаштування і персоналізації листів. Це дозволяє створювати більш цільові і релевантні повідомлення для кожного користувача, що може значно підвищити їх ефективність. Підтримка динамічного контенту і можливість легко налаштовувати різні варіанти розсилок робить SendGrid гнучким і універсальним інструментом для будь-якого бізнесу.

Інтеграція з іншими сервісами і платформами також є сильною стороною SendGrid. Платформа підтримує інтеграцію з численними CRM-системами, маркетинговими платформами і іншими інструментами, що дозволяє легко інтегрувати її у вже існуючі бізнес-процеси. Це робить перехід на SendGrid простим і безболісним, що є важливим фактором для багатьох компаній.

4.5 Огляд реалізації операцій CRUD у фреймворку Django

В рамках даної роботи було реалізовано операції створення, перегляду, редагування та видалення користувацьких профілів, футбольних секцій, програм, команд, занять, а також реєстрації на програми та призначення гравців в команди.

Як вже відомо, серверну частину CRUD реалізовано за допомогою Django / Django Rest Framework. Варто відзначити, що кожний API енпдоїнт має дворівневу перевірку авторизації. По-перше, проводиться перевірка валідності JWT токену через сервіс Passage/1Password (authentication_classes по коду). По-друге, проводиться перевірка RBAC/ABAC, як-от, перевіряється відповідність ролі та атрибутів користувача виконуваним операціям (наприклад, тренер може бачити тільки свої команди). Остання перевірка можлива завдяки реалізації користувацьких секцій, які зберігаються в сервісі Redis. Коли користувач заходить в систему на клієнтській стороні, клієнт виконує запит на валідацію сесії в Redis, а коли користувач виходить з системи – на її інвалідацію. Код валідації сесії наведено нижче:

```
class ValidateUserSessionView(HandleAPIExceptionMixin, views.APIView):
    """Validate a user session in Redis."""

    authentication_classes = (PassageAuthentication,)

    def post(self, request, format=None):
        """Validate a user session."""
        access_token = get_token_auth_header(request)
        hashed_access_token = hash_token(access_token) if access_token else None
        email_hash = request.data.get("email_hash")

        user = User.objects.filter(email_md5=email_hash).first()

        if not user:
            logger.error(f"Broken session for a user {email_hash}")
            raise BrokenSession(message="Broken session")

        session_data = dict(
            user_type=user.user_type,
            user_id=user.id,
        )

        redis_client = RedisNormalClient()
        redis_client.set_value_to_dict(
            hash_name=REDIS_CURRENT_USERS_KEY,
            key=hashed_access_token,
            value=json.dumps(session_data),
        )

        return Response(
            status=200,
            data={"message": "Validated a user session successfully."},
        )
```

Код інвалідації користувацької сесії наведено нижче:

```
class InvalidateUserSessionView(HandleAPIExceptionMixin, views.APIView):
    """Invalidate a user session in Redis."""
```

```
    authentication_classes = (PassageAuthentication,)

    def post(self, request, format=None):
        """Invalidate a user session."""
        access_token = get_token_auth_header(request)
        hashed_access_token = hash_token(access_token) if access_token else None

        redis_client = RedisNormalClient()
        redis_client.delete_value_from_dict(
            hash_name=REDIS_CURRENT_USERS_KEY,
            key=hashed_access_token,
        )

        return Response(
            status=200,
            data={"message": "Invalidated a user session successfully."},
        )
```

Отже, наявність сесій дозволяє зберігати обмежену інформацію про користувача в Redis (як-от, роль користувача) та перевіряти відповідність ролі користувача виконуваній операції або навіть створювати новий ресурс для поточного користувача. Наприклад, розглянемо енпдоїнт створення програми:

```
class ProgramListCreateView(HandleAPIExceptionMixin, generics.ListCreateAPIView):
    """Program list / creation API."""

    queryset = Program.objects.all()
    authentication_classes = (PassageAuthentication,)
    pagination_class = DynamicPageSizePageNumberPagination
    serializer_class = ProgramSerializer

    def create(self, request, *args, **kwargs):
        data = request.data

        if getattr(self.request, "srm_session", None):
            current_user_id = self.request.srm_session.get("user_id")
        else:
            logger.info("No session is present on request.")
            current_user_id = None

        data["coach_id"] = current_user_id

        serializer = self.get_serializer(data=data)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_201_CREATED,
headers=headers)
```

Як бачимо, при створенні програми перевіряється srm_session запиту – сесія, яку було взято з Redis.

Для того, щоб кожний API енпдоїнт мав змогу користуватися даними сесії, було створено глобальну обгортку для запитів (middleware), фрагмент коду якої наведено нижче:

```
class CustomSessionMiddleware(MiddlewareMixin):
    """SRM Custom session middleware."""

    def process_request(self, request):
        access_token = get_token_auth_header(request)
        hashed_access_token = hash_token(access_token) if access_token else None
        redis_client = RedisNormalClient()
        try:
            current_user = redis_client.get_value_from_dict(
                hash_name=REDIS_CURRENT_USERS_KEY,
                key=hashed_access_token,
            )
        except DataError as exc:
            logger.error(
                f"User session seems no not have been validated "
                f"with /auth/user/validate/ upon login; "
                f"{repr(exc)}"
            )
            raise BrokenSession(message="User session seems no not have been
validated.")

        if current_user:
            request.srm_session = loads(current_user)
        else:
            raise BrokenSession(message="User session does not exist.")
```

Знову має місце взаємодія з клієнтом Redis: на кожен запит дана глобальна обгортка бере дані сесії з Redis та приєднує їх до контексту запиту (request.srm_session), який стає доступним API енпдоїнту, до якого звертається запит. Аналогічний код було створено для CRUD над усіма ресурсами (програмами, командами, секціями, заняттями, тощо).

Варто відзначити, що особливістю даної роботи є наявність режиму адміністратора, інтерфейс якої повністю визначено у серверній частині системи за допомогою технології Django Admin. У адміністратора, що виступає супер-користувачем системи та якого створено за допомогою скриптів Django, є привілеї на виконання операцій CRUD над усіма уснуючими ресурсами системи (програмами, командами, тощо). На бекенді у декларативному стилі, який вимагає Django Admin, було задано правила пошуку, сортування та виводу полів ресурсів, як продемонстровано нижче:

```
@admin.register(Team)
class TeamAdmin(admin.ModelAdmin):
    list_display = (
        "id",
        "name",
        "coach",
        "updated_at",
    )
    list_display_links = list_display
    filter_horizontal = ("players",)
    search_fields = (
        "name",
        "coach",
    )
```

Інтерфейс Django Admin, який відповідає наведеному сніпету, наведено на рис. 4.1.

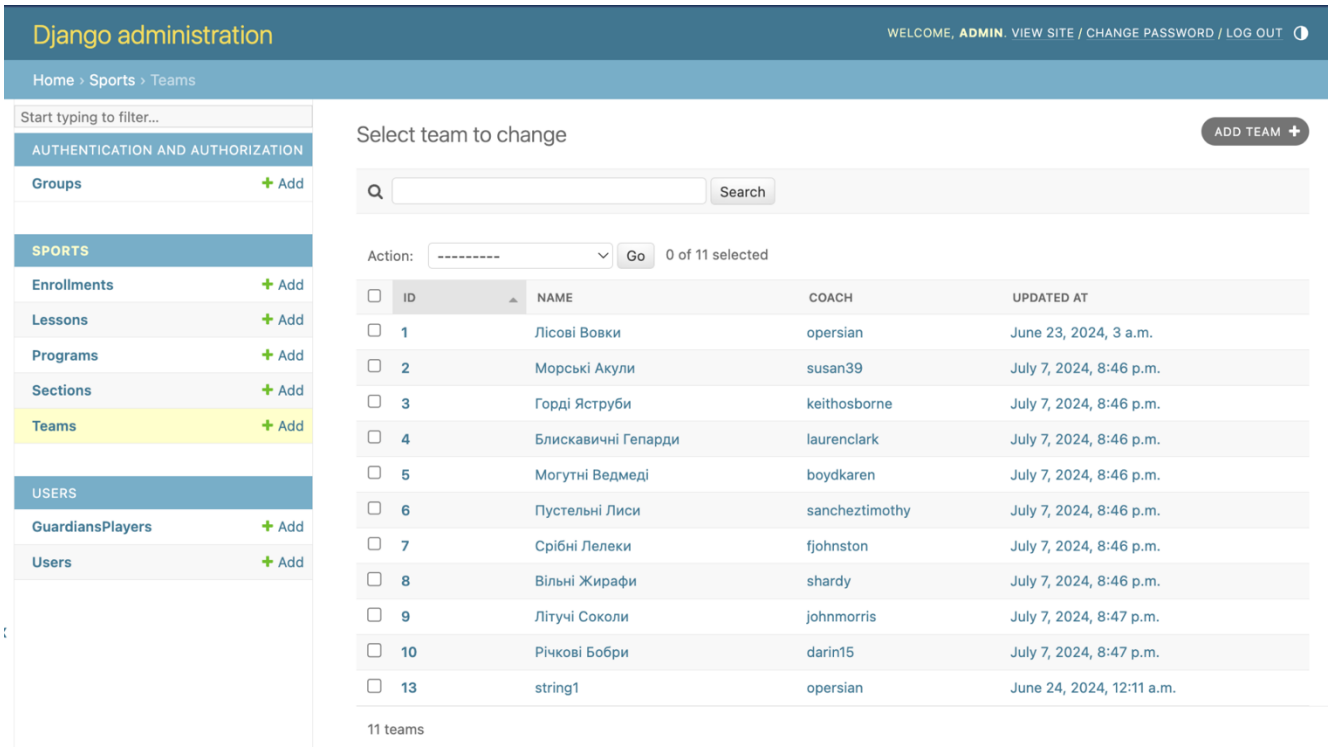


Рисунок 4.1 – Інтерфейс адміністратора у Django Admin: перелік команд (рисунок виконаний самостійно)

Таким чином, було реалізовано серверну частину функціональності CRUD для різноманітних ресурсів (футбольні програми, секції, команди, заняття, тощо), яку було захищено механізмами авторизації (JWT, RBAC/ABAC) та розширено інтерфейсом адміністрування Django Admin.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Проведення API тестування серверної частини

З метою виявлення дефектів та багів, а також покращення якості програмного продукту, проведемо API тестування основних функцій за ролями менеджера секцій, тренера, та опікуна, тобто API тестування RBAC [9]. API тестування буде проведено вручну за допомогою сервісу Postman [13].

Протестуємо оновлення профілів користувачів:

- оновлення власного профілю опікуном, а також оновлення профілю іншого (будь-якого) користувача адміністратором (супер-адміністратором): очікується успіх, код HTTP статусу (далі – просто код) – 200 Success (рис.5.1);
- спроба оновлення профілю іншого тренера тренером: очікується помилка, адже в тренера немає прав редагувати профілі інших користувачів – код помилки авторизації 403 Forbidden (рис.5.2).

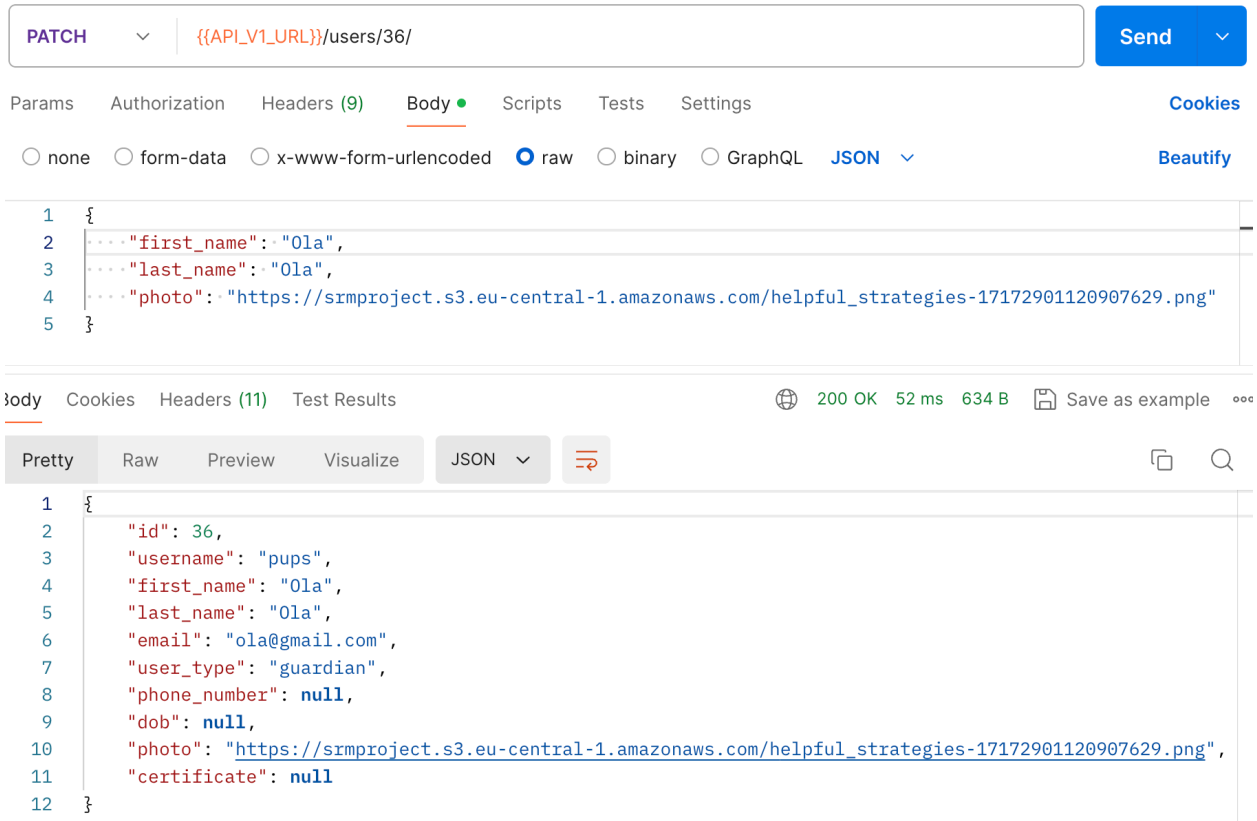


Рисунок 5.1 – Тест «Оновлення власного профілю опікуном» (рисунок виконаний самостійно)

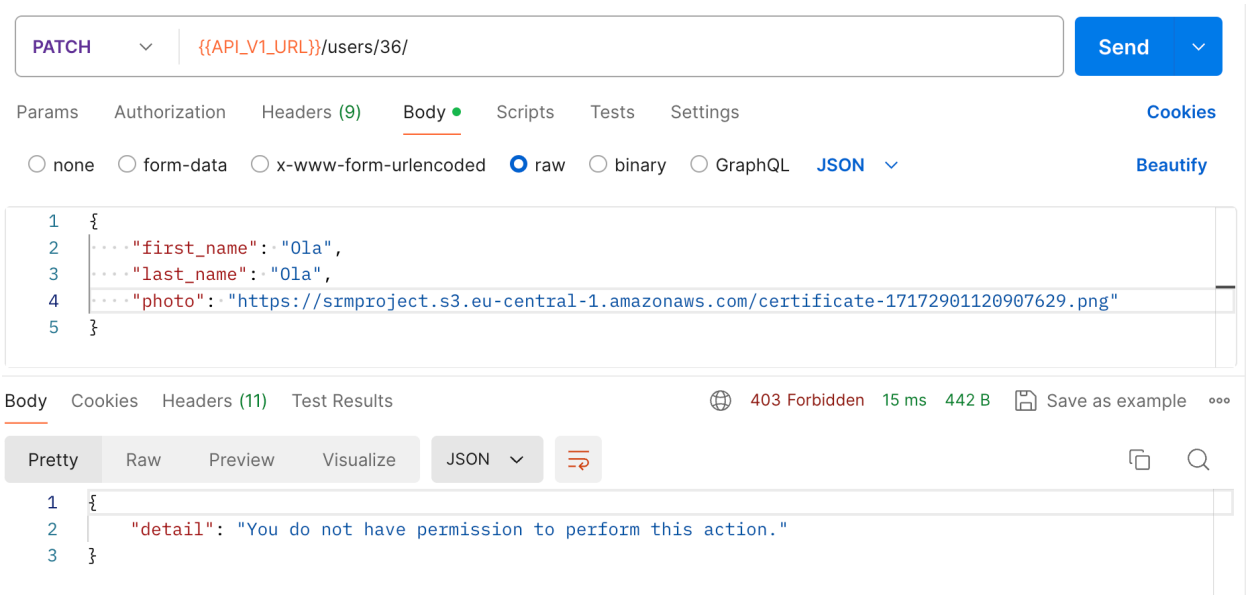


Рисунок 5.2 - Тест «Оновлення профілю іншого тренера тренером» (рисунок виконаний самостійно)

Протестуємо оновлення секцій менеджерами секцій:

- оновлення власної секції менеджером секцій – код 200 (рис.5.3);
- спроба оновлення секції іншого менеджера менеджером секцій – код 403 (рис.5.4).

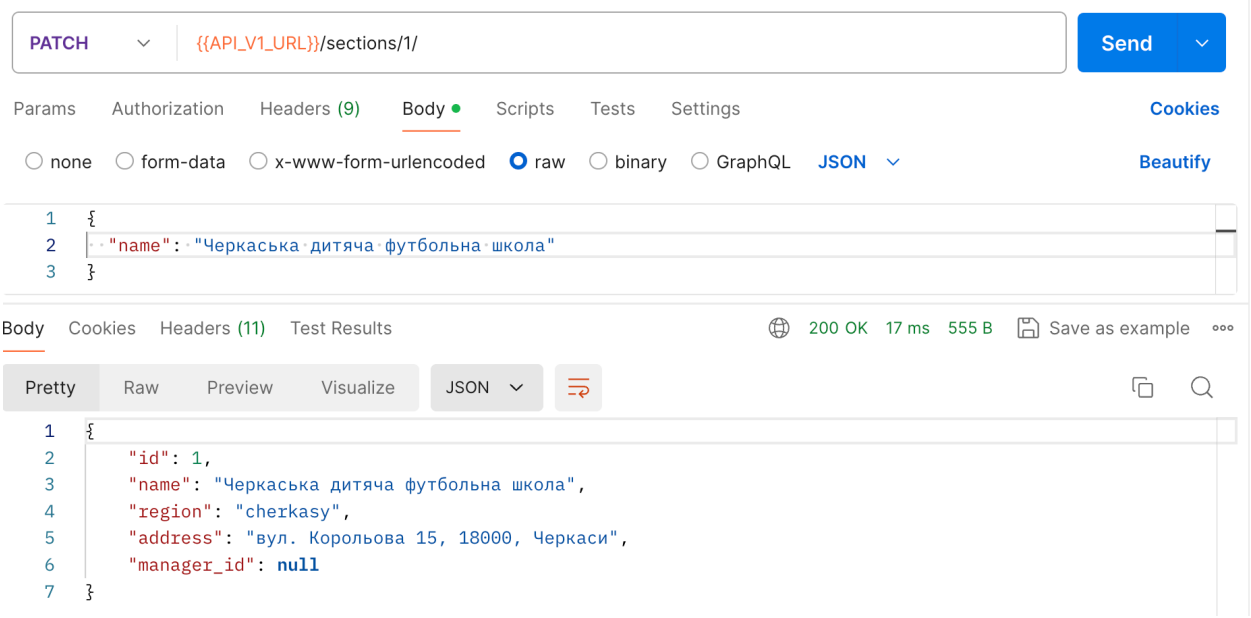


Рисунок 5.3 - Тест «Оновлення власної секції менеджером секцій» (рисунок виконаний самостійно)

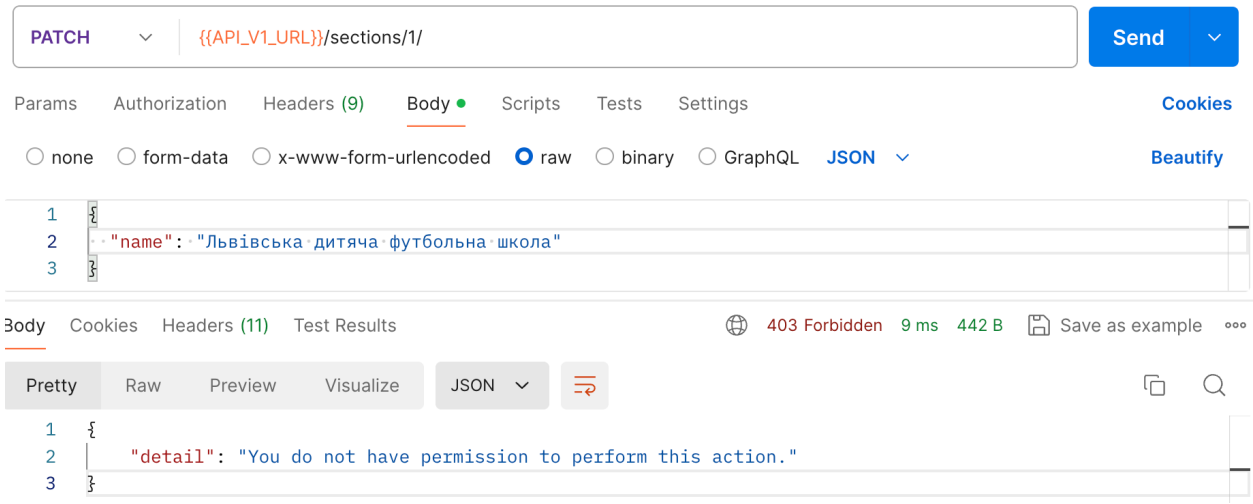


Рисунок 5.4 - Тест «Оновлення секції іншого менеджера менеджером секцій» (рисунок виконаний самостійно)

Таким чином, відслідковуються чіткі правила недопущення користувачам певних ролей або з певними атрибутами (як-от, відсутність «власності» тренера на команду іншого тренера), які застосовуються для всіх режимів, ролей, та ресурсів. Тим часом, продовжимо тестування RBAC.

Протестуємо видалення гравця з команди:

- видалення гравця з власної команди тренером – код успіху 204 No Content (рис.5.5);
- спроба видалення гравця команди іншого тренера тренером – код 403 (рис.5.6).

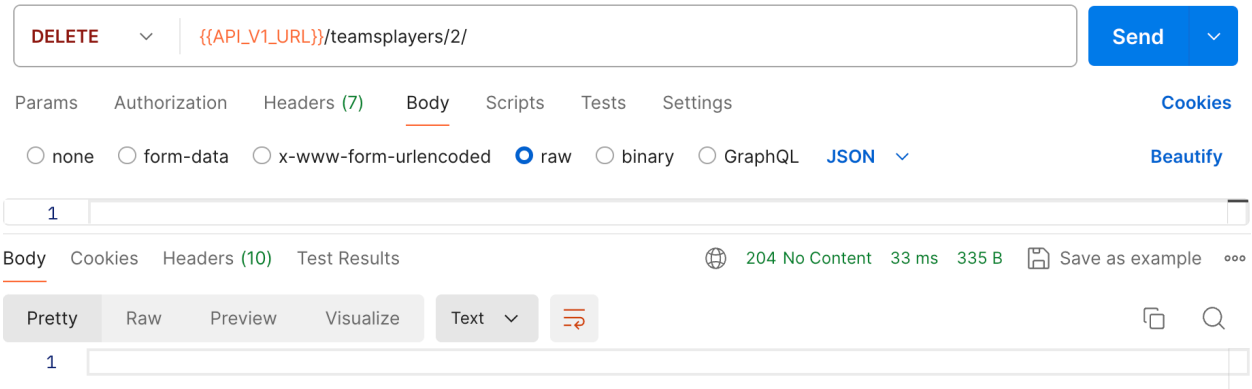


Рисунок 5.5 - Тест «Видалення гравця з власної команди тренером» (рисунок виконаний самостійно)

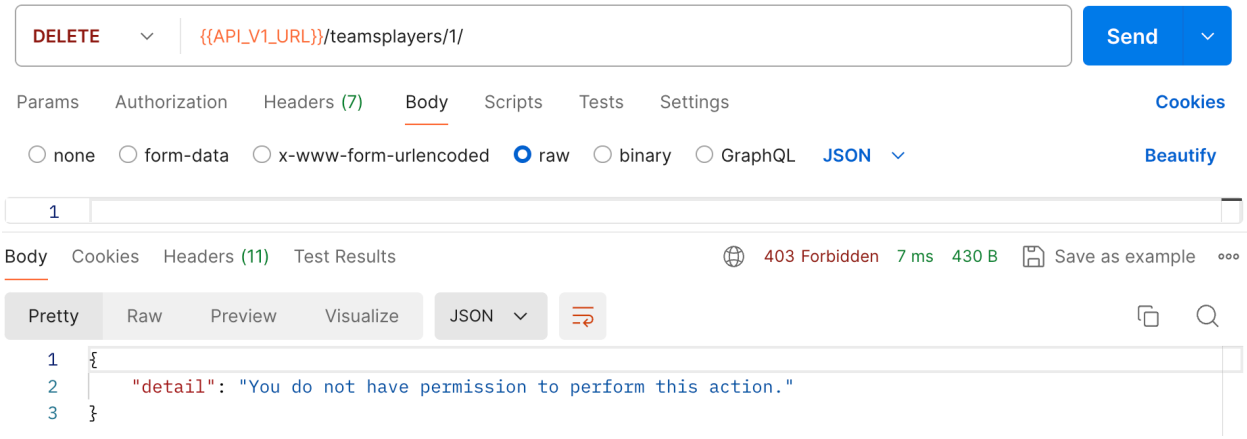


Рисунок 5.6 - Тест «Видалення гравця команди іншого тренера тренером» (рисунок виконаний самостійно)

Протестуємо функцію режиму опікунів:

- призначення підопічного опікуну опікуном – код успіху 201 Created (рис.5.7);
- спроба призначення підопічного іншому опікуну опікуном – код 403 (рис.5.8).

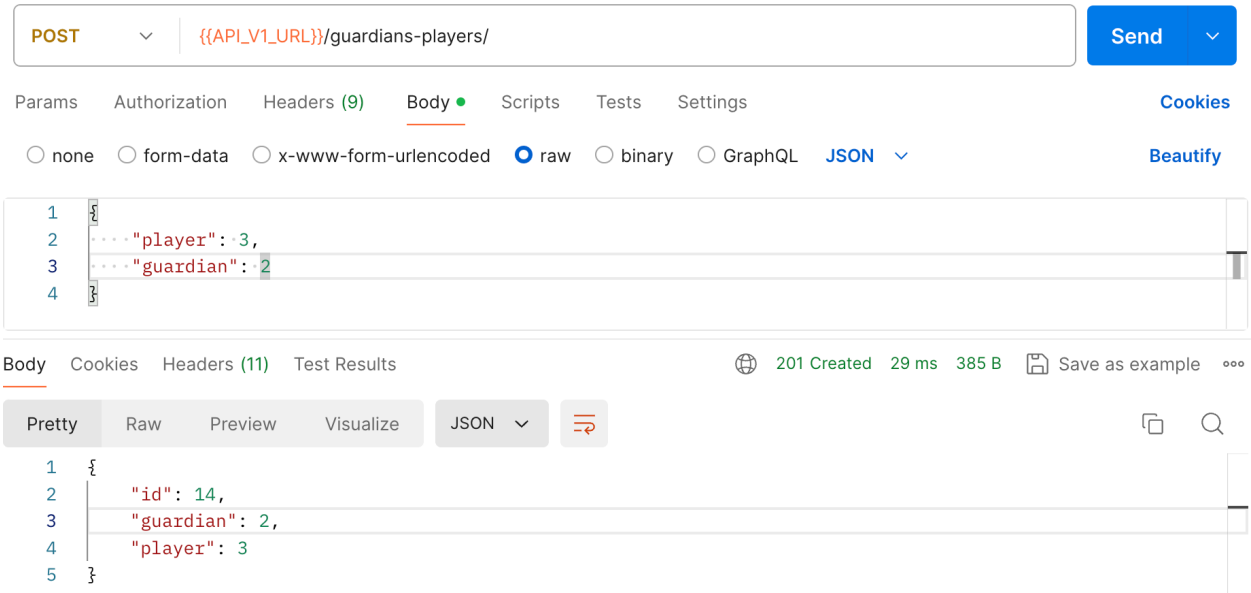


Рисунок 5.7 - Тест «Призначення підопічного опікуну опікуном» (рисунок виконаний самостійно)

Аналогічні правила RBAC накладаються на операції створення, видалення, та перегляду ресурсів.

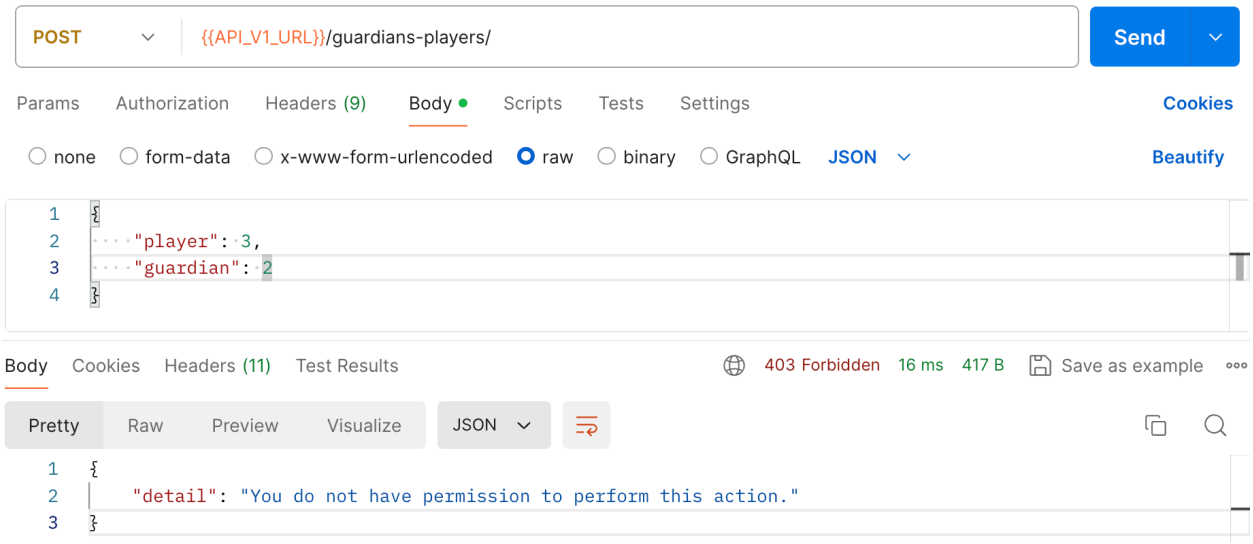


Рисунок 5.8 - Тест «Призначення підопічного іншому опікуну опікуном» (рисунок виконаний самостійно)

Як бачимо, реалізований механізм RBAC відповідає функціональним вимогам стосовно заборони користувачам, що не мають мати змоги виконувати певні операції, здійснювати такі операції.

5.2 Проведення навантажувального тестування серверної частини

З метою підтримання відповідності нефункціональним вимогам, проведемо навантажувальне тестування за допомогою Python бібліотеки “locust” [20]. При створенні тестів була врахована карта запитів, що їх надсилає клієнтська частина для отримання різноманітних даних (програм, секцій, занять, тощо). Запити були згруповані в класи тестів, як-от, ProgramsPageTestCase, TeamsPageTestCase, SectionsPageTestCase, - що відповідає структурі запитів клієнтської сторони за сторінками (програми, секції, команди, тощо).

Навантажувальне тестування було проведено за наступними параметрами: 200 користувачів максимум (що відповідає очікуваній максимальній кількості користувачів, згідно нефункціональних вимов); новий користувач приєднується до системи кожну секунду; об’єктом тестування виступає серверна частина застосунку з використанням gunicorn серверу, що максимально наближене до

реальних умов розгортання застосунку в мережі Інтернет; показники операційної системи серверу – 10-ядерний CPU на базі ARM, RAM 32 Gb.

Приклад реалізації тесту наведено нижче:

```
class PlayersPageTest(HttpUser, BaseLoadTestSetup):
    """Players page load tests."""

    wait_time = between(WAIT_TIME_FROM, WAIT_TIME_TO)

    @tag("players-list")
    @task(3)
    def get_players_list(self):
        """Test players page flow."""
        self._issue_request("/api/v1/users/", method=self.client.get)

    @tag("player-details")
    @task(3)
    def get_player_details(self):
        """Test a player (user) page."""
        resource_id = self._get_random_pk()
        self._issue_request(f"/api/v1/users/{resource_id}",
method=self.client.get)
```

Після проведення навантажувального тестування за допомогою locust були отримані результати, представлені в табл. 5.1.

Таблиця 5.1. Результати навантажувального тестування (таблицю виконано самостійно)

Кількість користувачів, що одночасно користуються застосунком	Тривалість виконання API запиту, мікросекунд		Загальна кількість виконуваних запитів у секунду
	середня	95-й персентиль	
10	22,98	57	1,5
50	23,41	62	10,4
100	24,23	72	23,6
150	28,04	120	35.8
200	31.05	180	50.4

Як бачимо, системної поламки не сталося ні за 100, ні за 200 одночасних користувачів системи, що відповідає нефункціональній вимозі стосовно кількості одночасних підключень (на перші півроку планова максимальна кількість одночасних підключень – 100, а через рік після запуску продукту система має витримувати 200 одночасних користувачів).

Враховуючи, що одна сторінка в середньому складається з 3-5 запитів, для 10 одночасних користувачів тривалість завантаження сторінки триватиме максимум 285 мілісекунд (приблизно), для 50 - 310, для 100 - 360, для 150 – 600, для 200 - 900 (не враховуючи клієнтської частини, що, зазвичай, займає мінімальну частку у загальній тривалості завантаження сторінки в браузері). Отримані результати відповідають нефункціональним вимогам (тривалість завантаження сторінки не має перевищувати 5 секунд за будь-яких умов - за будь-якої кількості одночасних користувачів).

Додатково, було отримані значення показнику загальної кількості виконуваних запитів у секунду, що також підтверджує достатню продуктивність розробленого програмного забезпечення.

Ті самі результати відображені на рис. 5.9, що демонструє фрагмент звіту, згенерованого бібліотекою locust [20].

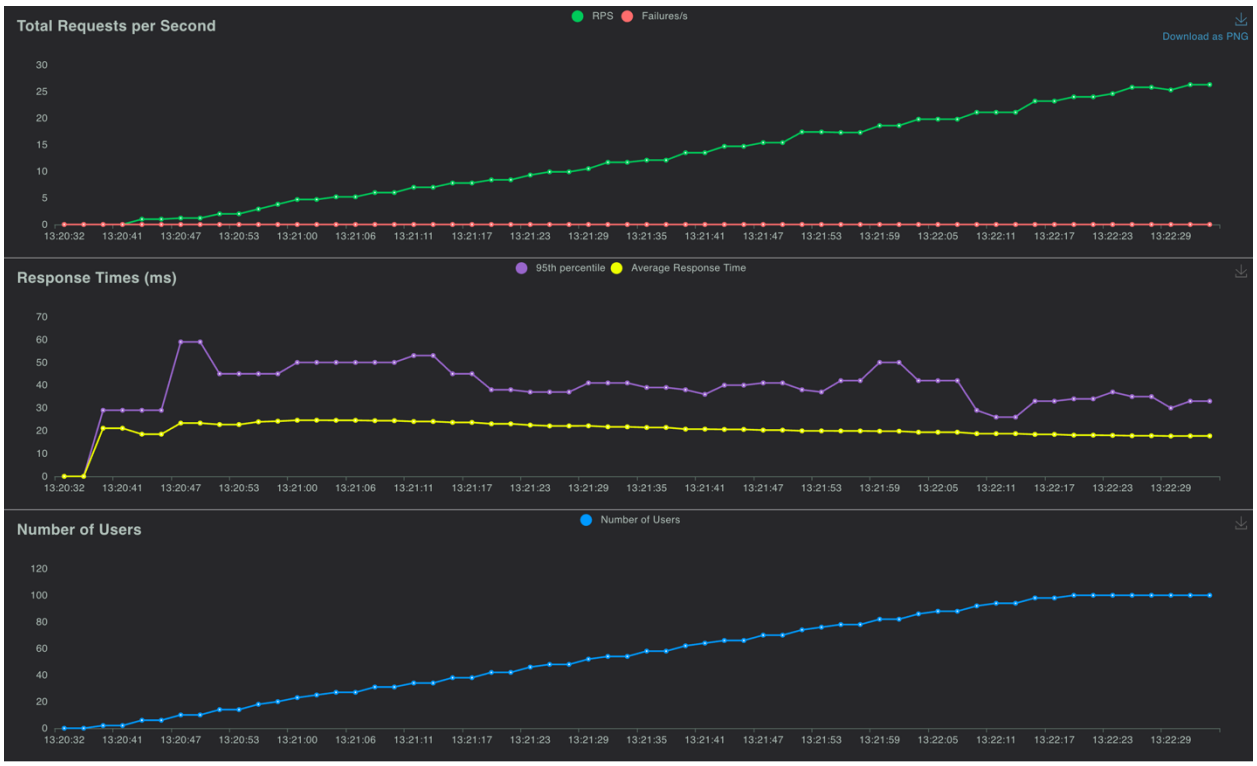


Рисунок 5.9 - Фрагмент звіту locust з результатами навантажувального тестування (рисунок виконаний самостійно)

Таким чином, було проведене API тестування та навантажувальне тестування серверної частини застосунку з адміністрування футбольних секцій, що дозволило

встановити чіткі об’єктивні критерії перевірки відповідності функціональним вимогам режимів менеджера секцій, тренера та опікуна і вимогам авторизації RBAC [7], так само як і нефункціональним вимогам продуктивності системи, а також запевнитися у відповідності системи встановленим вимогам.

ВИСНОВКИ

У результаті роботи було спроєктовано та реалізовано серверну частину програмної системи з управління відносинами у сфері спорту, що дозволяє адмініструвати футбольні секції програм для навчання грі в футбол, а також здійснювати пошук програм і тренерів та комунікації тренерів і менеджерів. Було вивчено логіку зберігання і роботи з інформацією та важливість ретельного ставлення до коректного зберігання та обробки даних.

Під час створення застосунку було розглянуто логіку використання реляційних баз даних та спроєктовано базу даних, проходячи етапи від концептуального моделювання предметної області до проєктування фізичної моделі за допомогою СУБД PostgreSQL. Створення таблиць та виконання запитів здійснено за допомогою Django ORM (Object-Relational Mapping).

Виконання цієї роботи надало можливість закріпити знання з технологій веб-розробки REST API, Django, а також технології віртуалізації Docker. Розуміння даних технологій допоможе успішно реалізовувати поставлені задачі, а також опанувати альтернативні підходи до вирішення аналогічних задач.

Результатом даної роботи є серверна частина системи з адміністрування футбольних секцій, що підтримує режими адміністратора, менеджера секцій, тренера та опікуна неповнолітніх гравців - з використанням реляційних баз даних та логіки об’єктно-орієнтованого програмування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sport customer relationship management, competitive advantage, satisfaction, loyalty, and complaint management. Kinesiologia Slovenica 28 (1). 2022. URL: https://www.researchgate.net/publication/360791494_Sport_customer_relationship_management_competitive_advantage_satisfaction_loyalty_and_complaint_management (дата звернення 10.06.24)

2. Сервіс пошуку тренерів “Trener” (Україна). URL: https://trener.ua/uk/kiev (дата звернення 18.06.24)

3. Сервіс пошуку репетиторів “BUKI” (Україна). URL: https://buki.com.ua/tutors/fitnes-trener/ (дата звернення 22.06.24)

4. Сервіс пошуку та доставки товарів і послуг “OLX” (Україна). URL: https://www.olx.ua/uk/uslugi/obrazovanie/ (дата звернення 01.07.24)

5. Сервіс пошуку навчальних закладів “Edusearch” (Україна). URL: https://edusearch.com.ua/ (дата звернення 16.06.24)

6. Сервіс замовлення послуг “Кабанчик” (Україна). URL: https://kabanchik.ua/ua/kyiv/category/posluhy-treneriv (дата звернення 22.06.24)

7. Role-Based Access Control - RBAC: Enforce Access Controls by OWASP. URL: https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls (дата звернення 19.06.24)

8. Українська асоціація футболу. URL: https://uaf.ua (дата звернення 30.05.24)

9. Django: The web framework for perfectionists with deadlines. URL: https://www.djangoproject.com/ (дата звернення 15.06.24)

10. Celery: Distributed Task Queue. URL: https://docs.celeryq.dev/en/stable/ (дата звернення 14.06.24)

11. OpenAPI Specification: The world's most widely used API description standard. URL: https://www.openapis.org/ (дата звернення 17.06.24)

12. Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine. URL:

https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення 22.06.24)

13. Postman: Build APIs together. URL: https://www.postman.com/ (дата звернення 01.07.24)

14. Django RESTful Framework: powerful and flexible toolkit for building Web APIs. URL: https://www.django-rest-framework.org/ (дата звернення 06.06.24)

15. Docker: Make better, secure software from the start. URL: https://www.docker.com/ (дата звернення 13.06.24)

16. Amazon S3: Object storage built to retrieve any amount of data from anywhere. URL: https://aws.amazon.com/s3/ (дата звернення 19.06.24)

17. Sendgrid Django SDK - django-sendgrid-v5. URL: https://github.com/sklarsa/django-sendgrid-v5 (дата звернення 02.07.24)

18. Unlock a passwordless future with passkeys: Passage by 1Password. URL: https://passage.1password.com/ (дата звернення 03.07.24)

19. Passage / 1Password Python SDK. URL: https://github.com/passageidentity/passage-python (дата звернення 11.06.24)

20. Locust: An open source load testing tool. URL: https://locust.io/ (дата звернення 01.07.24)

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

Дата звіту 7/14/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІпз-22-1_Персіанова_О_Ю_архів

Автор
Персіанова Олена Юріївна

Науковий керівник / Експерт
Вадим Юрійович Нечволод

підрозділ
Харківський національний університет радіоелектроніки

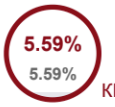
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	¶	1
Білі знаки	ß	1
Парафрази (SmartMarks)	a	18

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КЦ

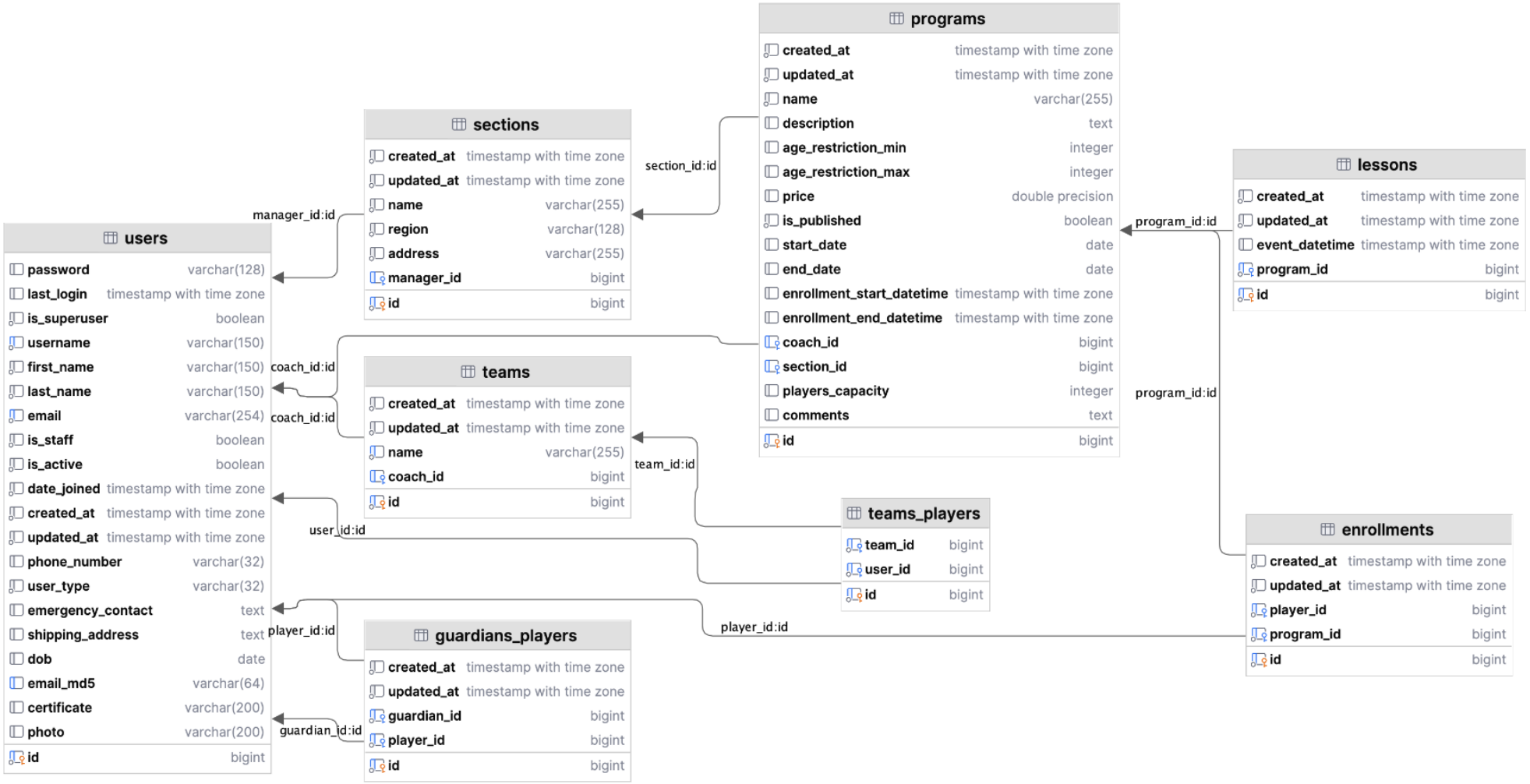
25
Довжина фрази для коефіцієнта подібності 2

12337
Кількість слів

101874
Кількість символів

ДОДАТОК Б

Схема реляційної бази даних



ДОДАТОК В

Приклад звіту

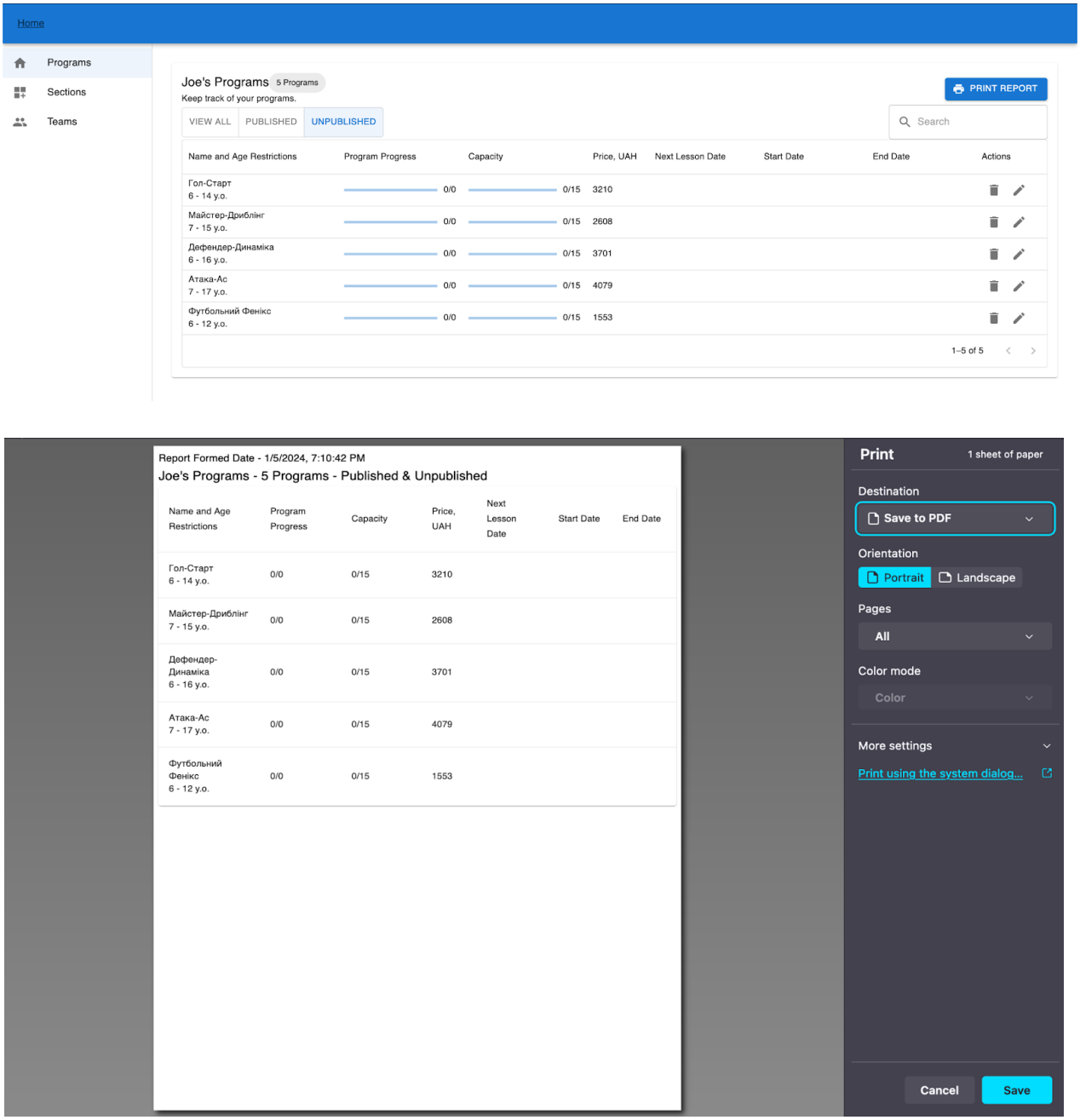


Рисунок В.1 - Перелік програм з даними про тренування (lessons), реєстрації (enrollments), цінами та датами початку і кінця - з фільтрацією за ознакою is_published (опція “Print Report”) (рисунок виконаний самостійно)

ДОДАТОК Г

Матриця ролей

Таблиця Г.1 - Матриця ролей і повноважень

№	Повноваження / Роль	Staff	Coach	Section Manager	Guardian	Anon
1	Зареєструватися в системі (register)					X
2	Профілі інших користувачів	X	X (R, опікуни та гравці власних програм)	X (R, тренери)	X (CRU, власний підопічний)	X (R - тренери)
3	Увійти в систему (login)					X
4	CRUD Ресурсу “Команди” (Teams)	X	X (R - власні)	X (U - add coach; власні)	X (R, власного Player)	
5	CRUD Ресурсу “Склад команди” (TeamsPlayers)	X	X (власні)		X (R, власного Player)	
6	CRUD Ресурсу “Заняття” (Lesson)	X		X (R all, CUD власні)	X (R, власного Player)	X (R)
7	CRUD Ресурсу “Секції” (Sections)	X		X (R all, CUD власні)	X (R, власного Player)	X (R)
8	CRUD Ресурсу “Програми” (Programs)	X	X (власні)	X (R all, U власні)	X (R, власного Player)	X (R)
9	CRUD Ресурсу “Реєстрації на програми” (Enrollments)	X	X (власні)		X (R, власного Player)	
10	CRUD Ресурсу “Призначені опікуни/батьки гравців” (GuardiansPlayers)	X			X (R, власного Player)	
11	Вийти з системи (logout)	X	X	X	X	

ДОДАТОК Д

Слайди презентації

Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

Кваліфікаційна робота
Програмна система з адміністрування футбольних секцій. Back-end

Виконала:
студентка групи ПЗПІпз-22-1
Персіанова О.Ю.

Керівник:
доц. кафедри ПІ
Валенда Н.А.

2024

Мета роботи

Створення програмної системи з адміністрування футбольних секцій, призначеної для полегшення роботи тренерів з футболу та менеджерів футбольних секцій, зокрема, з можливістю формування команд, а також надання інформації батькам та опікунам гравців щодо навчальних програм та занять з футболу.

2

Актуальність

- організація та супроводження футбольних занять є важкою та кропітливою роботою, що нерідко лягає на плечі футбольних тренерів;
- для гравців та їхніх батьків або опікунів важливим аспектом є співпраця саме з найкращими та перевіреними тренерами;
- стратегічно важливим є розвиток футбольної спільноти України та довгострокових взаємозв'язків між усіма зацікавленими сторонами футболу.

3

Аналіз існуючих рішень

Характеристика	Конкурент				
	Trener	Кабанчик	Buki	Olx	Edusearch
Сфокусованість на набутті навичок (компетентнісний підхід)	+ (частково)	-	-	-	-
Контроль якості послуг тренерів	+ (відгуки)	+ (відгуки)	+ (відгуки)	-	+ (відгуки)
Організаційне супроводження користувачів (тренерів та гравців)	-	-	-	-	-
Покриття інших видів спорту, наприклад, футзал, теніс, баскетбол.	+	+	+	+	+

4

Постановка задачі

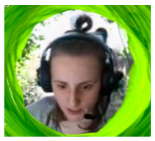
Створити програмну систему з адміністрування футбольних секцій, яка забезпечить роботу 4 груп користувачів:

- адміністратора,
- менеджера секцій,
- тренера,
- опікуна неповнолітніх гравців.

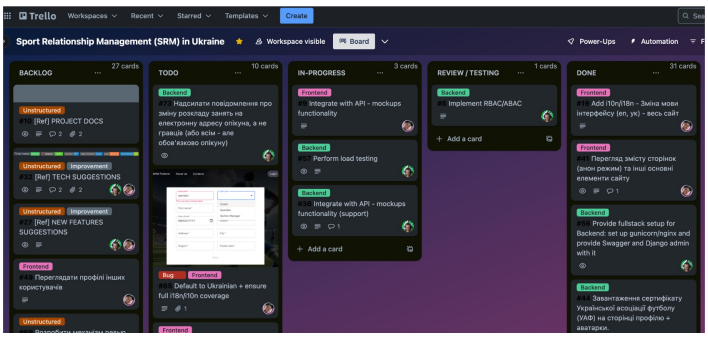
5

Ролі в проєкті

Олена Персіанова	**Сергій Персіанов**
Backend-розробниця	Frontend-розробник
Бізнес-аналітикиня	Власник продукту
Менеджерка проєкту	UI/UX дизайнер
Тестувальниця	Тестувальник



Інструмент для керування проєктами **Trello**



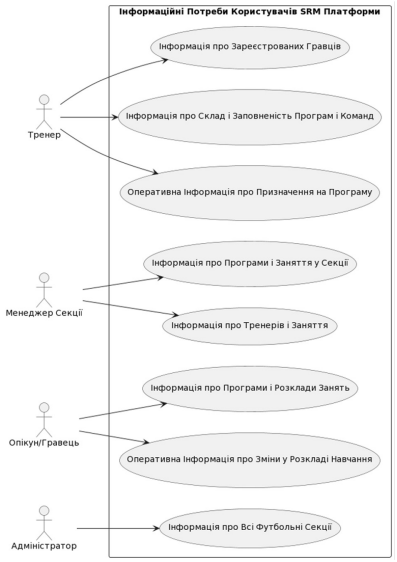
6

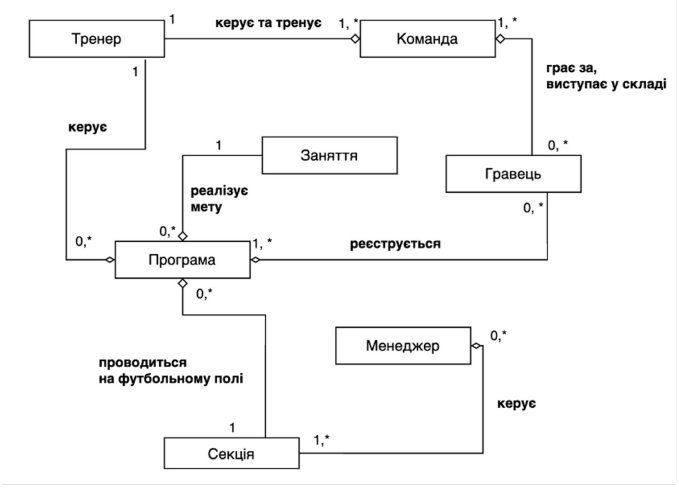
Постановка задачі

- Реалізувати серверну частину програмної системи з адміністрування футбольних секцій.
- Визначити функціональні та нефункціональні вимоги програмної системи з адміністрування футбольних секцій.
- Спроєктувати та реалізувати базу даних програмної системи, використовуючи ORM веб-фреймворку Django.
- Реалізувати REST API програмної системи за допомогою веб-фреймворку Django та бібліотек мови програмування Python, включно з авторизацією за ролями адміністратора, менеджера секцій, тренера та опікуна.
- Реалізувати функціональність відправки повідомлень на електронну пошту користувачів за допомогою інструмента з управління задачами Celery та сервісу Sendgrid.
- Провести API тестування за допомогою інструменту Postman та навантажувальне тестування з бібліотекою locust.
- Створити середовище розгортання проєкту за допомогою Docker та Docker Compose.

7

Загальна USE-CASE діаграма та діаграма класів



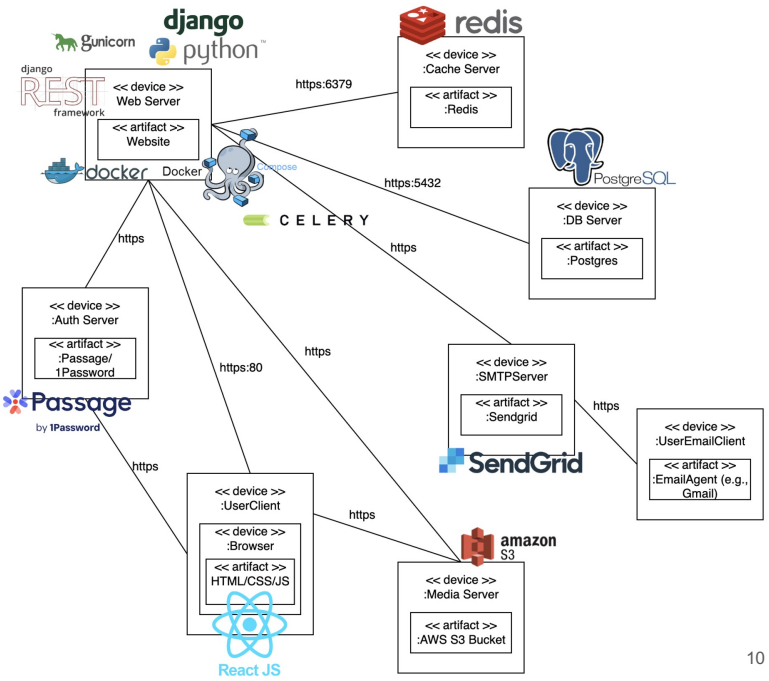


8

Схема БД

DataGrip, JetBrains IDE

Діаграма розгортання та засоби реалізації

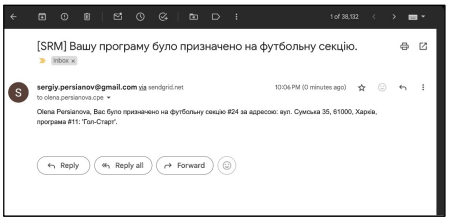


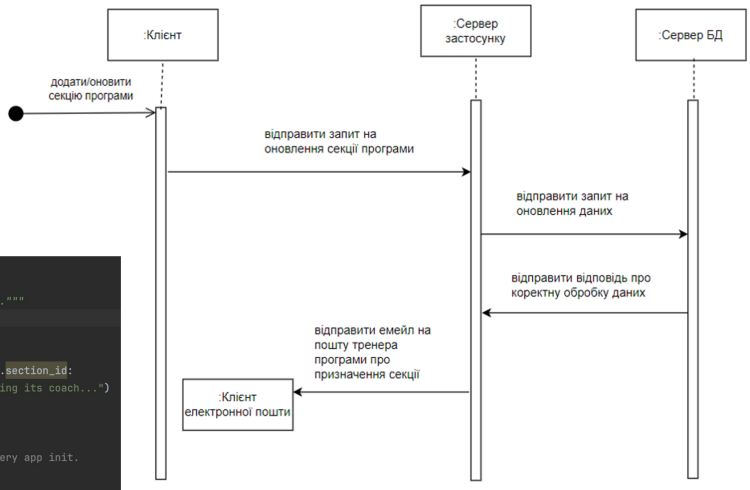
Фрагменти коду API енпдоїнту та URL роутингу

```python
class ProgramListCreateView(HandleAPIExceptionMixin, generics.ListCreateAPIView):
    """Program list / creation API."""

    queryset = Program.objects.all()
    authentication_classes = (PassageAuthentication,)
    pagination_class = DynamicPageSizePageNumberPagination

    def get_serializer_class(self):
        """Provide different serializations depending on the incoming request."""
        if self.request.method == "POST":
            return ProgramSerializer
        else:
            return ProgramAnnotatedSerializer

    def get_queryset(self):
        """Custom get_queryset method."""
        if self.request.method == "GET":
            search = self.request.query_params.get("search")

            if getattr(self.request, "srm_session", None):
                current_user_role = self.request.srm_session.get("user_type")
                current_user_id = self.request.srm_session.get("user_id")
            else:
                logger.info("No session is present on request.")
                current_user_role = None
                current_user_id = None

            # RBAC/ABAC
            if current_user_role == COACH_ROLE:
                # Coach should get theirs only
                queryset = super().get_queryset().filter(coach_id=current_user_id)
            elif current_user_role == GUARDIAN_ROLE:
                # Guardian should get their players'
```

```python
"""API v1 paths."""
from django.urls import path

from api.v1 import views


urlpatterns = [
    path("references/", views.ReferenceView.as_view(), name="reference"),
    path("users/", views.UserListView.as_view(), name="users-list"),
    path("users/<int:pk>/", views.UserDetailView.as_view(), name="users-detail"),
    path("guardians-players/", views.GuardianPlayersView.as_view(), name="assign-guardian-player"),
    path(
        "sections/", views.SectionListCreateView.as_view(), name="sections-list-create"
    ),
    path(
        "sections/<int:pk>/", views.SectionDetailView.as_view(), name="sections-detail"
    ),
    path(
        "programs/", views.ProgramListCreateView.as_view(), name="programs-list-create"
    ),
    path(
        "programs/<int:pk>/", views.ProgramDetailView.as_view(), name="programs-detail"
    ),
    path(
        "programs/<int:pk>/enrollments/",
        views.ProgramEnrollmentListView.as_view(),
        name="programs-enrollments-list",
    ),
    path(
        "enrollments/",
        views.EnrollmentListCreateView.as_view(),
        name="enrollments-list",
    ),
    path(
        "programs/<int:pk>/", views.ProgramDetailView.as_view(), name="programs-detail"
    ),
```

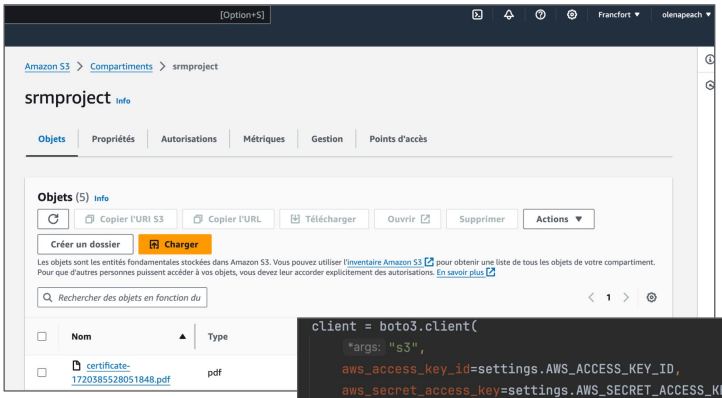

Рішення зі сповіщень тренерам про призначення їхніх програм на футбольні секції



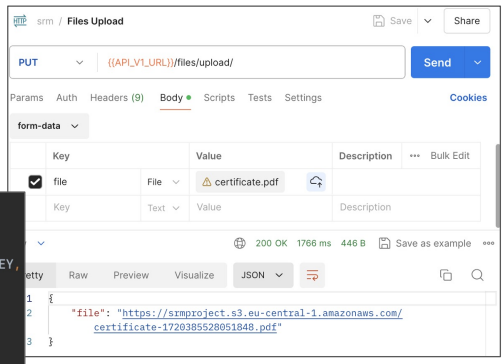



```python
@receiver(pre_save, sender=Program)
def program_coach_assigned_signal(sender, instance: Lesson, **kwargs):
    """Notify a coach when their program gets assigned to a sports section."""
    previous = Program.objects.get(id=instance.id)

    # Only if assigned from None or re-assigned
    if instance.section_id is not None and previous.section_id != instance.section_id:
        logger.debug(f"{instance} was newly-assigned to a section - notifying its coach...")

        program_id = instance.id

        # NOTE: local import: to avoid circular import error caused by Celery app init.
        from background.tasks import send_email_coach_section_assigned
        send_email_coach_section_assigned.delay(program_id=program_id)
```

Завантаження файлів у Amazon S3 Bucket




```
client = boto3.client(
    *args: "s3",
    aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
    aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY,
)
client.put_object(
    Body=contents,
    Bucket=settings.AWS_STORAGE_BUCKET_NAME,
    Key=filename,
)
```

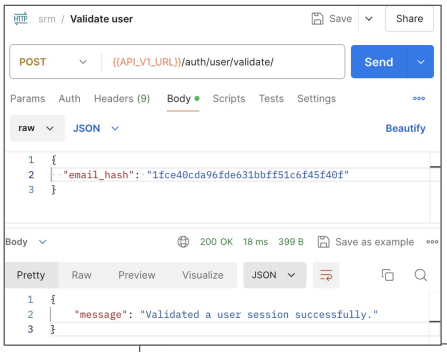


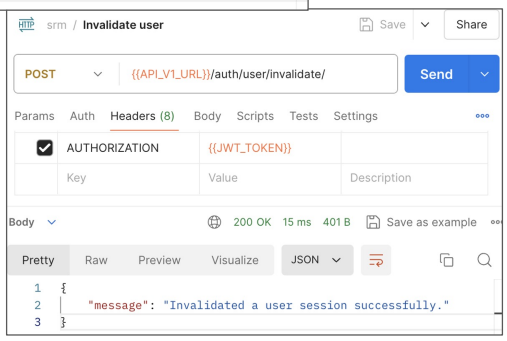
14

Зберігання сесій у Redis

```
def post(self, request, format=None):
    """Validate a user session."""
    access_token = get_token_auth_header(request)
    hashed_access_token = hash_token(access_token)
    email_hash = request.data.get("email_hash")

    user = User.objects.filter(email_md5=email_hash).first()

    redis_client = RedisNormalClient()
    redis_client.set_value_to_dict(
        hash_name=REDIS_CURRENT_USERS_KEY,
        key=hashed_access_token,
        value=dumps(
            dict(
                user_type=user.user_type,
                user_id=user.id,
            )
        ),
    )
```

```
if getattr(self.request, "srm_session", None):
    current_user_role = self.request.srm_session.get("user_type")
    current_user_id = self.request.srm_session.get("user_id")
```





15

Фрагменти Dockerfile та docker-compose.yml

```dockerfile
FROM node:18.19 AS static

ADD . /tmp

WORKDIR /tmp

RUN npm install

RUN npm run build

FROM python:3.12.0

RUN apt-get -y update && \
    apt-get install -y \
    git gcc build-essential && \
    rm -rf /var/lib/apt/lists/*

RUN mkdir /requirements
ADD ./requirements/* /requirements/

WORKDIR /requirements
RUN pip install --upgrade pip && pip install -r dev.txt && rm -rf ~/.cache

RUN apt-get purge -y --auto-remove git
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

ENV PYTHONUNBUFFERED 1

RUN mkdir /app
ADD . /app

WORKDIR /app

COPY --from=static /tmp/frontend/dist frontend/dist
```

```yaml
version: "3.8"

services:
  web:
    container_name: srm_platform
    build:
      context: .
      dockerfile: Dockerfile
    command:
      - /bin/bash
      - -c
      - |
        python manage.py migrate --noinput
        python manage.py runserver 0.0.0.0:8000
    restart: always
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    depends_on:
      - db
    env_file:
      - .env
    stdin_open: true
    tty: true

  db:
    container_name: srm_platform_postgres
    image: postgres:14.2
    env_file:
      - .env
    environment:
      - POSTGRES_DB=$DB_NAME
      - POSTGRES_USER=$DB_USER
      - POSTGRES_PASSWORD=$DB_PASSWORD
    restart: always
    ports:
      - "5432:5432"
```

bac_diploma			Running (5/6)
worker-1	d513330e1789	locustio/locust	Exited
master-1	65f5a9f33bdd	locustio/locust	Running
srm_celery	c632a72ec4d3	bac_diploma-celery	Running
srm_platform	b0335e41ecbe	bac_diploma-web	Running
srm_platform_postgr	dacd95bf8e67	postgres:14.2	Running
srm_redis	f37714a0a510	redis:latest	Running

16

Інтерфейс адміністратора (Django Admin)

Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Sports › Teams

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION
- Groups + Add

SPORTS
- Enrollments + Add
- Lessons + Add
- Programs + Add
- Sections + Add
- Teams + Add

USERS
- GuardiansPlayers + Add
- Users + Add

Select team to change

ADD TEAM +

Search

Action: --------- Go 0 of 11 selected

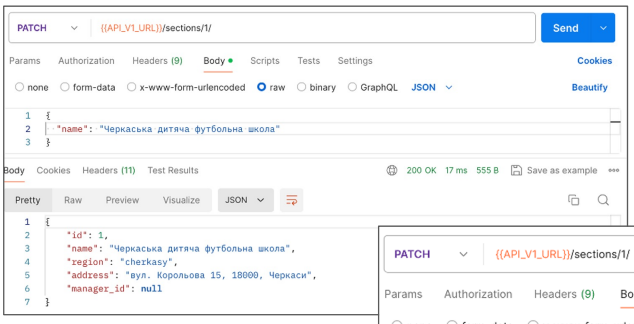
ID	NAME	COACH	UPDATED AT
1	Лісові Вовки	opersian	June 23, 2024, 3 a.m.
2	Морські Акули	susan39	July 7, 2024, 8:46 p.m.
3	Горді Яструби	keithosborne	July 7, 2024, 8:46 p.m.
4	Блискавичні Гепарди	laurenclark	July 7, 2024, 8:46 p.m.
5	Могутні Ведмеді	boydkaren	July 7, 2024, 8:46 p.m.
6	Пустельні Лиси	sancheztimothy	July 7, 2024, 8:46 p.m.
7	Срібні Лелеки	fjohnston	July 7, 2024, 8:46 p.m.
8	Вільні Жирафи	shardy	July 7, 2024, 8:46 p.m.
9	Літучі Соколи	johnmorris	July 7, 2024, 8:47 p.m.
10	Річкові Бобри	darin15	July 7, 2024, 8:47 p.m.
13	string1	opersian	June 24, 2024, 12:11 a.m.

11 teams

```python
@admin.register(Team)
class TeamAdmin(admin.ModelAdmin):
    """Admin interface for Teams."""

    list_display = (
        "id",
        "name",
        "coach",
        "updated_at",
    )
    list_display_links = list_display
    filter_horizontal = ("players",)
    search_fields = (
        "name",
        "coach",
    )
```

Тестування API

POSTMAN

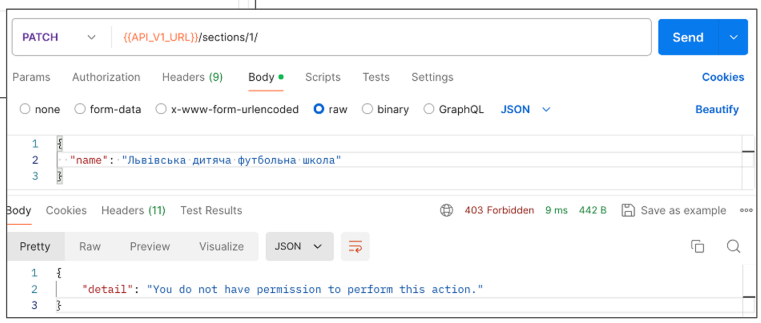


Зміна назви власної команди тренером

200 OK

Спроба змінити назву команди іншого тренера

403 Forbidden



Навантажувальне тестування

LOCUST

Кількість користувачів, що одночасно користуються застосунком	Тривалість виконання API запиту, мікросекунд	Загальна кількість виконуваних запитів у секунду
10	57	1,5
50	62	10,4
100	72	23,6
150	120	35.8
200	180	50.4

```python
class ProgramsPageTestCase(HttpUser, BaseLoadTestSetup):
    """Programs page load tests."""

    wait_time = between(WAIT_TIME_FROM, WAIT_TIME_TO)

    @tag("programs-list")
    @task(3)
    def get_programs_list(self):
        """Test programs page flow."""
        self._issue_request(
            url: "/api/v1/programs/",
            method=self.client.get,
        )

    @tag("program-details")
    @task(3)
    def get_program_details(self):
        """Test a particular program page."""
        resource_id = self._get_random_pk()
        self._issue_request(
            url: f"/api/v1/programs/{resource_id}",
            method=self.client.get,
        )
```

Висновки

- визначено функціональні та нефункціональні вимоги програмної системи з адміністрування футбольних секцій;
- спроєктовано та створено базу даних програмної системи за допомогою ORM веб-фреймворку Django;
- реалізовано та протестовано REST API програмної системи за допомогою веб-фреймворку Django та бібліотек мови програмування Python, включно з авторизацією за ролями адміністратора, менеджера секцій, тренера та опікуна;
- реалізовано функціональність відправки сповіщень на електронну пошту користувачів за допомогою Celery та Sendgrid;
- створено середовище розгортання проєкту за допомогою Docker та Docker Compose.

19