ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА WEB-СЕРВІСУ «Календар спортивних змагань» ДЛЯ ОНЛАЙН-РЕЄСТРАЦІЇ НА ЗМАГАННЯ МОВОЮ PYTHON»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Лілія ПЕТРУСЬ
(підпис)

Виконала: здобувачка вищої освіти групи ПД-41

Лілія ПЕТРУСЬ

Керівник: Олена НЕГОДЕНКО
к.т.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Петрусь Лілії Андріївні

1. Тема кваліфікаційної роботи: : «Розробка web-сервісу «Календар спортивних змагань» для онлайн-реєстрації на змагання мовою Python»
керівник кваліфікаційної роботи к.т.н., доц., доцент кафедри ІПЗ Олена НЕГОДЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація мови програмування Python, офіційна документація PyCharm Pro, науково-технічна література, пов’язана з розробкою web-сервісів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд аналогів та аналіз існуючих рішень.
2. Опис основних вимог до web-сервісу «Календар спортивних змагань» для онлайн-реєстрації на змагання.
3. Проектування та розробка web-сервісу.
4. Тестування web-сервісу.

5. Перелік графічного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідження,
2. Задачі дипломної роботи.
3. Аналіз аналогів.
4. Вимоги до програмного забезпечення.
5. Програмні засоби реалізації.
6. Діаграма варіантів використання актора «Гість».
7. Діаграма варіантів використання актора «Учасник».
8. Діаграма варіантів використання актора «Організатор».
9. Діаграма варіантів використання актора «Адміністратор».
10. Схема бази даних.
11. Карта web-сервісу.
12. Екранні форми.
13. Екранні форми.
14. Екранні форми.
15. Відео роботи web-сервісу.
16. Апробація результатів дослідження.
17. Висновки

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	Виконано
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	Виконано
3	Огляд та аналіз процесів організації та проведення спортивних змагань	14.03-19.03.2024	Виконано
4	Проектування web-сервісу календаря спортивних змагань з онлайн-реєстрацією на змагання	20.03-27.03.2024	Виконано
5	Програмна реалізація web-сервісу	20.03-25.04.2024	Виконано
6	Тестування web-сервісу	26.04-29.04.2024	Виконано
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	Виконано
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	Виконано
9	Попередній захист роботи	13.05-31.05.2024	Виконано

Здобувачка вищої освіти _____ Лілія ПЕТРУСЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 36 рис., 25 джерел.

Мета роботи – спрощення процесу адміністрування спортивних змагань за допомогою веб-сервісу, створеного мовою Python.

Об'єкт дослідження – процес адміністрування спортивних змагань.

Предмет дослідження – веб-сервіс для адміністрування спортивних змагань.

Короткий зміст роботи: У роботі проведено опис та аналіз існуючих рішень за допомогою яких, учасник змагань може зареєструватися на обране спортивне змагання. Ключові проблеми, що можна виділити, пов'язані з поширенням інформації про спортивні змагання, простота та доступність реєстрації на змагання, обмеженість керування організатором реєстраціями учасників. Це призводить до потреби створення онлайн-сервісу «Календар спортивних змагань» з онлайн-реєстрацією та оплатою. Проаналізовано веб-сервіси календарів спортивних змагань: Timing Events, Ukraine Sport Events, ВсіПробіги. Розроблено та програмно реалізовані ключові функціональні можливості, зокрема: створення, редагування, видалення реєстрацій учасників організатором, авторизація на сервісі за допомогою соцмереж, редагування та анулювання реєстрацій на спортивні змагання учасниками змагань самостійно. Проведено функціональне та модульне тестування додатку. У роботі використано Python, фреймворки Django та Django Rest Framework для реалізації REST API, СУБД PostgreSQL для зберігання даних, JavaScript, фреймворк Vue.js, HTML, CSS, Bootstrap (CSS фреймворк). Розгортатися Web-сервіс буде у хмарному сервісі AWS (Amazon Web Services).

Сферою використання сервісу є організація спортивних змагань та реєстрація на них.

КЛЮЧОВІ СЛОВА: WEB-СЕРВІС, PYCHARM, PYTHON, VUEJS, СПОРТИВНЕ ЗМАГАННЯ, ОРГАНІЗАТОР, УЧАСНИК, РЕЄСТРАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Фреймворк – це готова модель, заготовка в програмуванні для швидкої розробки, на основі якої можна писати власний код.

IDE – Integrated Development Environment – інтегроване середовище розробки. Програмне рішення для розробки програмного забезпечення. Має в своєму складі редактор коду, інструменти збірки та налагодження програм.

REST -Representational State Transfer – передача репрезентативного стану. Це архітектурний стиль взаємодії компонентів розподіленого додатку в мережі.

API – Application Programming Interface – програмний інтерфейс додатку – набір способів та правил взаємодії однієї комп'ютерної програми з іншими.

SQL – Structured Query Language – структурована мова запитів – мова, за допомогою якої відбуваються маніпуляції з даними у реляційних базах даних.

ORM – Object-Relational Mapping – об'єктно-реляційне відображення – технологія, яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, суть якої в створенні «віртуальної об'єктної бази даних».

PIP – система керування пакетами, які написані мовою Python. Використовується для установки та керування пакетами.

NPM – Node Package Manager – стандартний менеджер пакетів Node.js. Використовується для установки та керування пакетами.

SPA – Single Page Application – одно сторінковий додаток.

DOM – Document Object Model – об'єктна модель документа, яку створює браузер в пам'яті на основі HTML-коду, отриманого від серверу.

HTML – Hyper Text Markup Language – мова гіпертекстової розмітки для перегляду веб-сторінок в браузере.

CSS – Cascading Style Sheets – каскадні таблиці стилів – формальна мова опису зовнішнього вигляду веб-сторінки.

URL – Uniform Resource Locator – адреса ресурсу в мережі Internet.

URI – Uniform Resource Identifier – уніфікований ідентифікатор ресурсу – послідовність символів, що ідентифікує абстрактний чи фізичний ресурс.

JSON – JavaScript Object Notation – текстовий формат обміну документами, оснований на JavaScript.

UML – Unified modeling language – універсальна мова моделювання.

CRUD – Created Read Update Delete – створення, читання, оновлення, видалення.

Скорочення:

СКБД – система керування базами даних.

БД – база даних.

ПЗ – програмне забезпечення.

ВСТУП

У наш час відслідковується велика зацікавленість гаджетами, а також малорухливий спосіб життя, це має згубний вплив на здоров’я нації. Заняття спортом сприяє збереженню та зміцненню ментального, фізичного, морального здоров’я. Також, беручи участь у спортивних змаганнях, люди вчяться об'єднувати всю свою енергію, справлятися з невдачами, концентруватися, долати всі труднощі і досягати перемоги. Ці навички допомагають людям у всіх сферах життя. Бажання організаторів мати якісні, гнучкі інструменти управління спортивними змаганнями, які допоможуть їм організовувати та проводити спортивні змагання, а також бажання учасників отримувати актуальну інформацію про змагання та легко і швидко реєструватися на них, вимагають створення платформ, які гарантують таку взаємодію. Для реалізації такої платформи було створено веб-сервіс. Актуальність даної роботи можна визначити стрімким розвитком веб-сервісів.

Метою роботи: спрощення процесу адміністрування спортивних змагань за допомогою веб-сервісу, створеного мовою Python.

Об’єкт дослідження: процес адміністрування спортивних змагань.

Предмет дослідження: веб-сервіс для адміністрування спортивних змагань.

Для реалізації поставленої мети потрібно виконати наступні задачі:

1) Проаналізувати існуючі аналоги web-сервісів для онлайн-реєстрації на змагання, визначити переваги та недоліки.

2) Розробити функціональні та нефункціональні вимоги до web-сервісу для онлайн-реєстрації на змагання.

3) Дослідити та вибрати засоби розробки web-сервісу, розробити модель архітектури web-сервісу та його дизайн.

4) Розробити web-сервіс на основі обраних інструментів та визначених вимог.

5) Провести тестування web-сервісу для онлайн-реєстрації на змагання.

Практичне значення результатів роботи: даний web-сервіс спростить реєстрацію на спортивні змагання для учасників змагань та допоможе організаторам змагань більше ефективно організовувати та проводити змагання за рахунок повного доступу до керування змаганнями та реєстраціями на них.

1 АНАЛІЗ ОРГАНІЗАЦІЇ СПОРТИВНИХ ЗМАГАНЬ ТА РЕЄСТРАЦІЇ НА ЗМАГАННЯ

1.1 Аналіз процесів організації спортивних змагань та реєстрації на змагання

Спорт (цей термін походить від давньофранцузького слова *disport* – «дозвілля», «розвага») – це діяльність, організована за певними правилами, що полягає в зіставленні фізичних та інтелектуальних досягнень людей. Спорт – це змагання за певними правилами. Зазвичай це специфічний вид фізичної або інтелектуальної активності, яку здійснюють з метою порівняння окремих здібностей, а також покращення здоров'я, отримання морального чи матеріального задоволення, прагнення до удосконалення та слави. Спорт, на відміну від фізичної культури чи фізичної активності, передбачає участь в офіційних змаганнях, за яку спортсмен може отримати матеріальну винагороду, титули, звання, нагороди. Крім, власне змагань, складовою спорту є підготовка до них у формі тренувальних занять [1].

Спортивні змагання – це своєрідна модель людських взаємовідносин, які реально існують у світі: боротьби, перемог і поразок, спрямованості до постійного вдосконалення, досягнення найвищого результату, творчих, престижних і матеріальних цілей [2].

З іншого боку, спортивні змагання – це об'єктивний спосіб демонстрації досягнутого рівня підготовленості, спосіб оцінки і порівняння досягнень окремих спортсменів і команд. Вони є системо утворюючим фактором спорту. В свою чергу системо утворюючим фактором змагань є результат.

В залежності від мети, завдань, форм організації, складу учасників змагання поділяються на різні види, а саме:

- за значенням (підготовчі, відбіркові, головні);
- за масштабами (районні, місцеві, регіональні, континентальні);
- за вирішенням завдань (контрольні, класифікаційні, відбіркові, показові);
- за характером організації (відкриті, закриті, традиційні, матчеві, кубкові);

- за формою заліку (особисті, командні, особисто-командні);

- за віковими категоріями учасників (дитячі, юнацькі, для дорослих, для ветеранів);

- за статтю (серед чоловіків або жінок);

- за професійною орієнтацією учасників (шкільні, студентські та інші) [2].

Організація і проведення масових спортивних заходів потребує багато зусиль та ресурсів від організатора для поширення інформації про захід, забезпечення простого і ефективного процесу реєстрації учасників на змагання, мінімізації накладок та помилок, можливості оперативної зміни деяких умов реєстрації. Також організатор потребує наявності оперативної інформації (в тому числі і фінансової) про хід реєстрації на змагання. В той же час для учасників змагань важливо мати свіжу інформацію про спортивні змагання (бажано зібрану на одному ресурсі) та можливість мати просту і доступну систему онлайн-реєстрації на змагання з можливістю онлайн-оплати. Також для учасників важливою є можливість керувати своєю реєстрацією – робити зміну деяких параметрів реєстрації (наприклад, змінити дистанцію, докупити додаткові опції, тощо), анулювати реєстрацію на змагання, передавати свою реєстрацію іншому учаснику. Інколи перед організатором постає задача реєстрації учасників, так як на деякі змагання компанії чи спортивні клуби подають списки людей, які будуть приймати участь у змаганні. Також не поодинокі ситуації, коли організатор мусить коригувати реєстрації учасників (зміна дистанції, анулювання реєстрації, зміна стартового номера і т.п.), так як не всі учасники можуть зробити це самостійно через різні причини. Також важливо наголосити, що у спортивного змагання може бути більше одного користувача-організатора.

1.2 Аналіз існуючих аналогів

Перед розробкою web-сервісу потрібно провести аналіз існуючих рішень, провести оцінку їх переваг та недоліків.

Було обрано 3 web-ресурси для дослідження – Timing Events. Ukraine Sport Events та ВсіПробіги.

Timing Events [3] було засновано в 2016 році. Сервіс надає комплексне обслуговування спортивного заходу з використанням системи електронного хронометражу. Компанією був запущений сервіс онлайн-реєстрації учасників на спортивні змагання TicketMe [4]. На рис. 1.1 показано головну сторінку сервісу Timing Events.

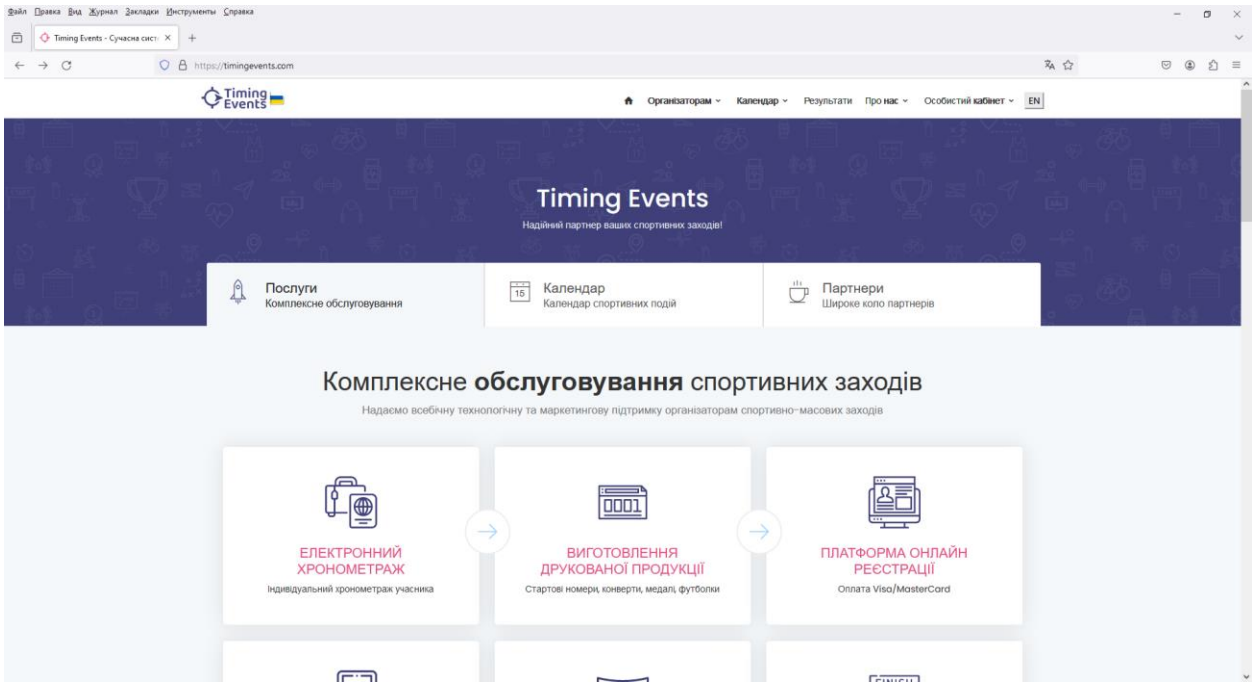


Рис. 1.1 Головна сторінка сервісу Timing Events

Окрім самого календаря спортивних подій (на рис. 1.2 зображений календар спортивних заходів, на рис. 1.3 показана сторінка спортивного змагання), сервісу онлайн-реєстрації на спортивні заходи та системи електронного хронометражу, сервіс надає послуги виготовлення друкованої продукції, продаж та оренду обладнання, забезпечення інфраструктури.

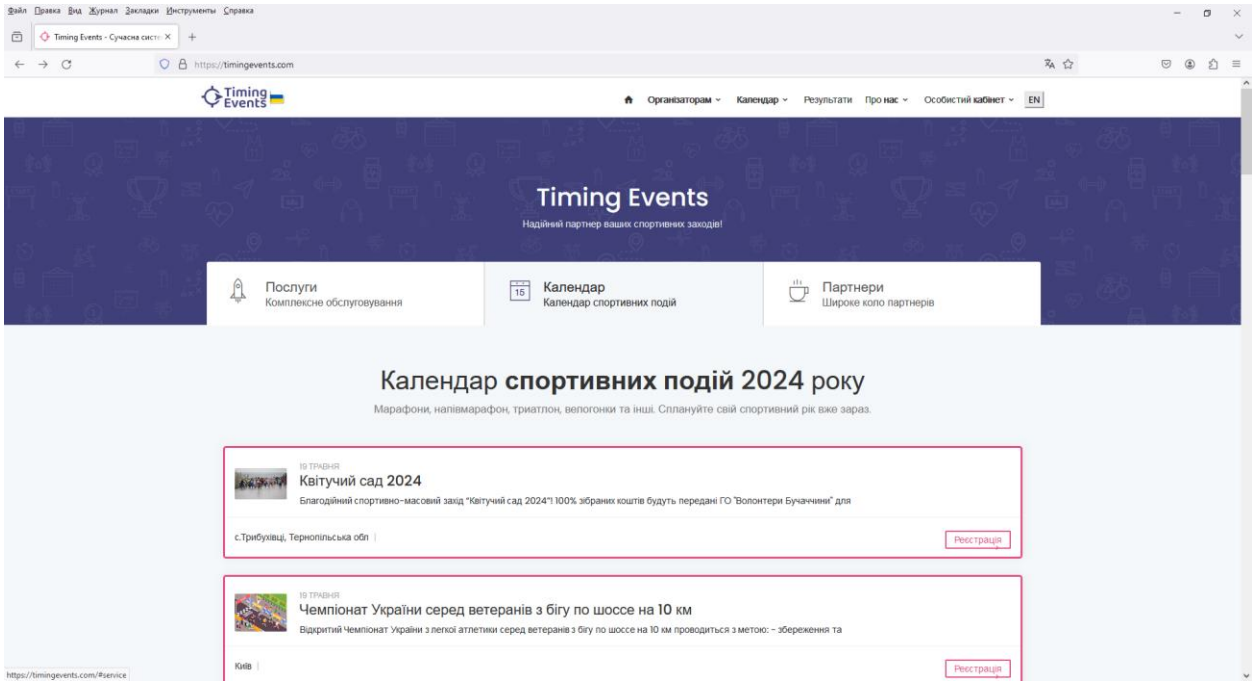


Рис. 1.2 Календар спортивних змагань сервісу Timing Events

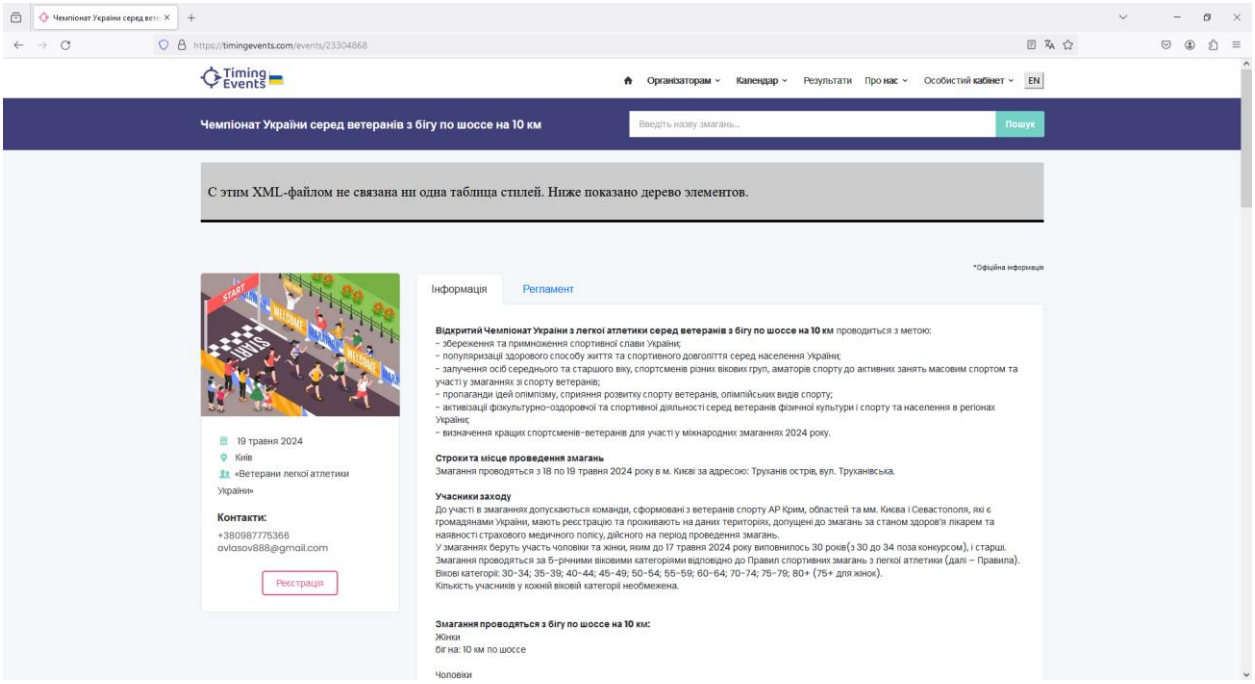


Рис. 1.3 Сторінка спортивного змагання на сервісі Timing Events

Реєстрація та оплата проходить на хмарному сервісі TicketMe. На ньому також продубльований календар змагань. На рис. 1.4 зображена сторінка реєстрації на спортивне змагання.

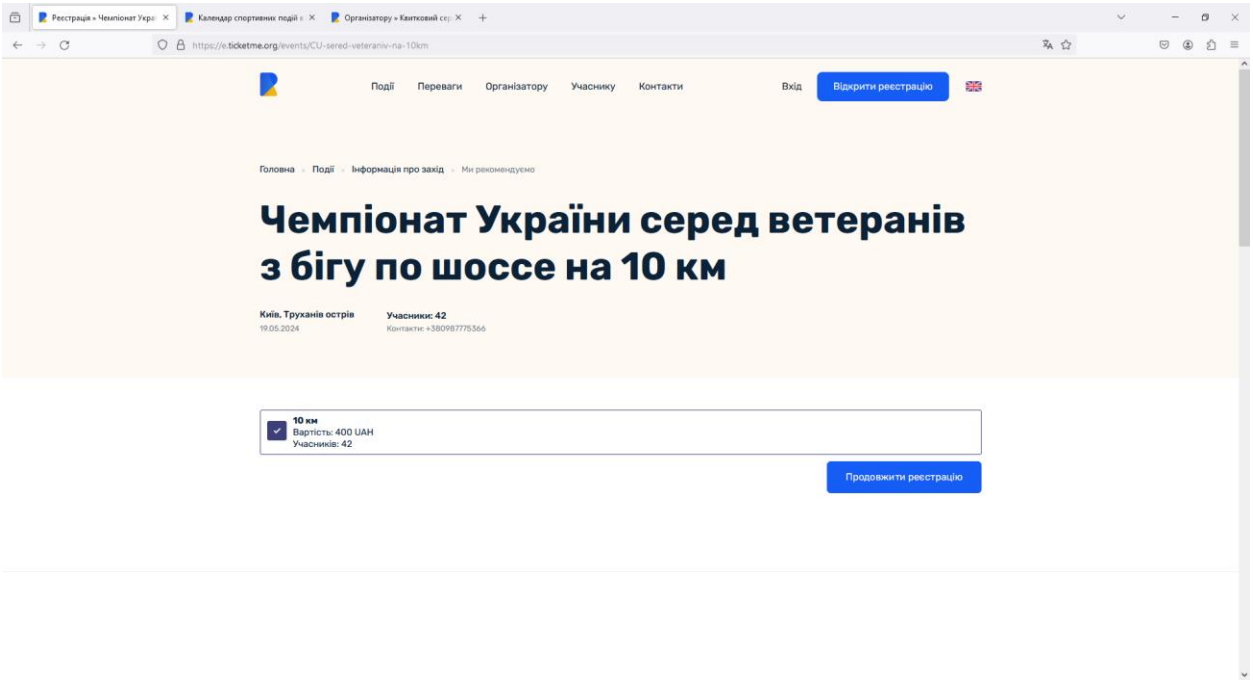


Рис 1.4 Сторінка реєстрації на спортивне змагання

Серед переваг сервісу – завдяки електронному хронометражу результати змагань миттєво з'являються на сервісі і також можуть надсилатися учаснику на email або за допомогою смс.

Недоліки сервісу:

- помилки відображення в інтерфейсі (див. рис. 1.3);
- учасник не може самостійно редагувати параметри своєї реєстрації на змагання;
- учасник не може самостійно передати свою реєстрацію іншому учаснику;
- організатор має повний доступ до керування реєстрацією тільки якщо є партнером;
- неможливо створити аккаунт на сервісі без реєстрації на спортивне змагання;
- неможливий вхід на сервіс за допомогою аккаунтів соцмереж або аккаунта Google.

Далі дослідимо сервіс Ukraine Sport Events [5]. На рис. 1.5 показана головна сторінка сервісу. Сервіс дозволяє реєстрацію на будь-які види змагань. Доступний вхід на сервіс за допомогою аккаунтів соцмереж. Надаються послуги електронного хронометражу, результати змагань миттєво доступні онлайн і також надсилаються на електронну пошту учаснику або за допомогою смс. Доступна генерація ваучера на отримання стартового номеру та генерація диплому учасника. Серед недоліків сервісу – помилки локалізації, учасник не може самостійно редагувати чи анулювати свою реєстрацію. На даний час календар сервісу на поточний рік пустий.

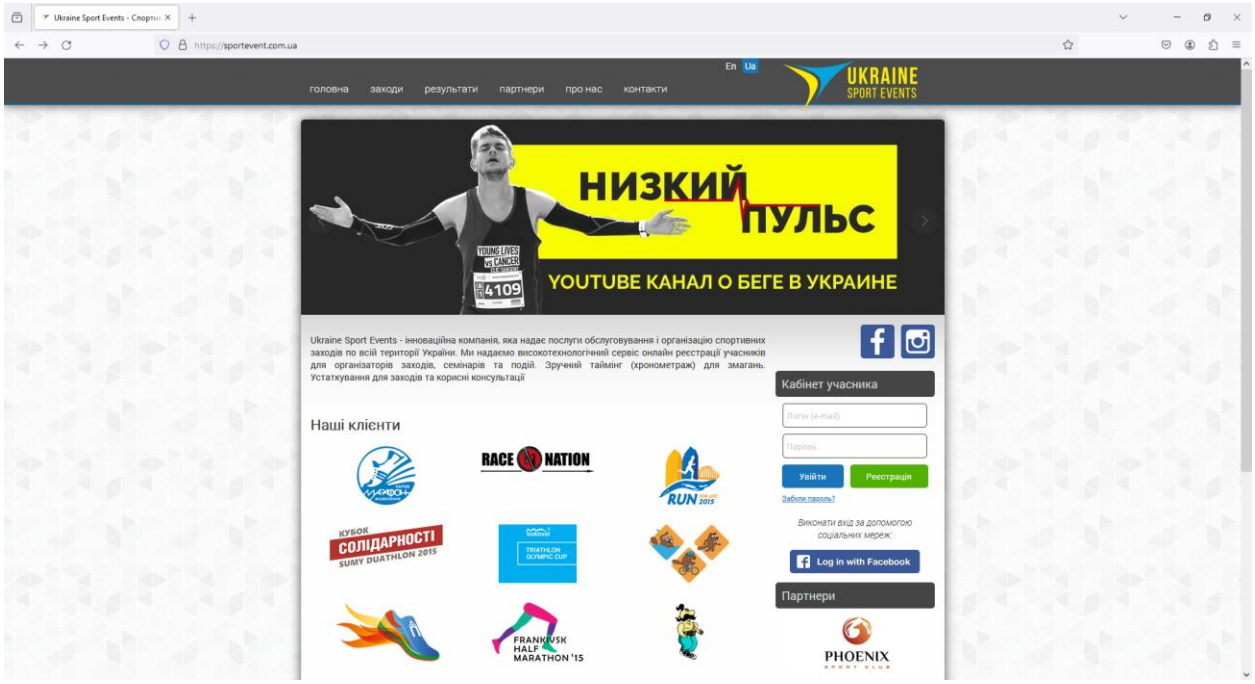


Рис. 1.5 Головна сторінка сервісу Ukraine Sport Events

Наступний досліджуваний web-сервіс – ВсіПробіги [6]. Головна сторінка сервісу показана на рис. 1.6.

Сервіс має як переваги так і недоліки. До переваг можемо віднести можливість організатора повністю керувати налаштуваннями спортивного заходу. Також організатору доступна генерація промокодів на реєстрацію на змагання.

Але сервіс має і ряд недоліків:

- недоступна реєстрація на всі види змагань (сервіс більш направлений на бігові види змагань);
- відсутність можливості підключення електронного хронометражу;
- неможливість редагування/анулювання реєстрації на змагання учасником самостійно;
- учасник не може самостійно передати свою реєстрацію іншому учаснику;
- учасник не може самостійно вибрати стартовий номер;
- учаснику недоступно отримання результатів змагання на електронну пошту або за допомогою смс;
- сервіс має частково адаптивний дизайн;
- недоступний вхід за допомогою аккаунтів соцмереж.

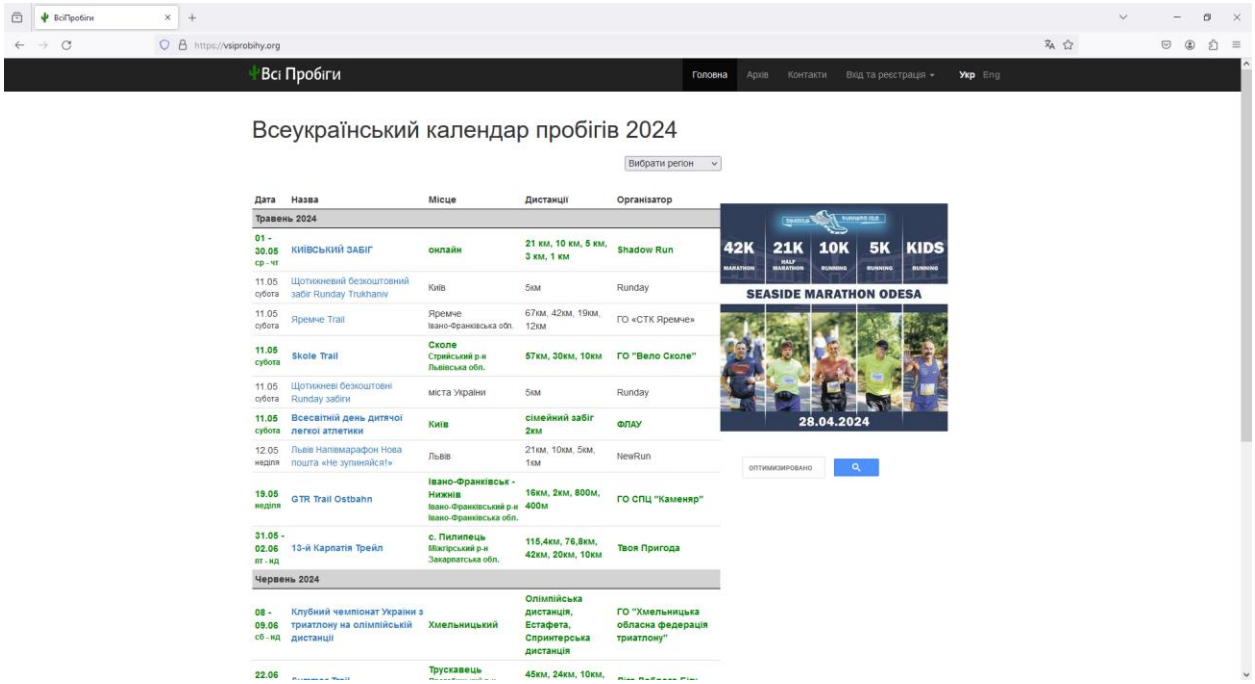


Рис. 1.6 Головна сторінка ВсіПробіги

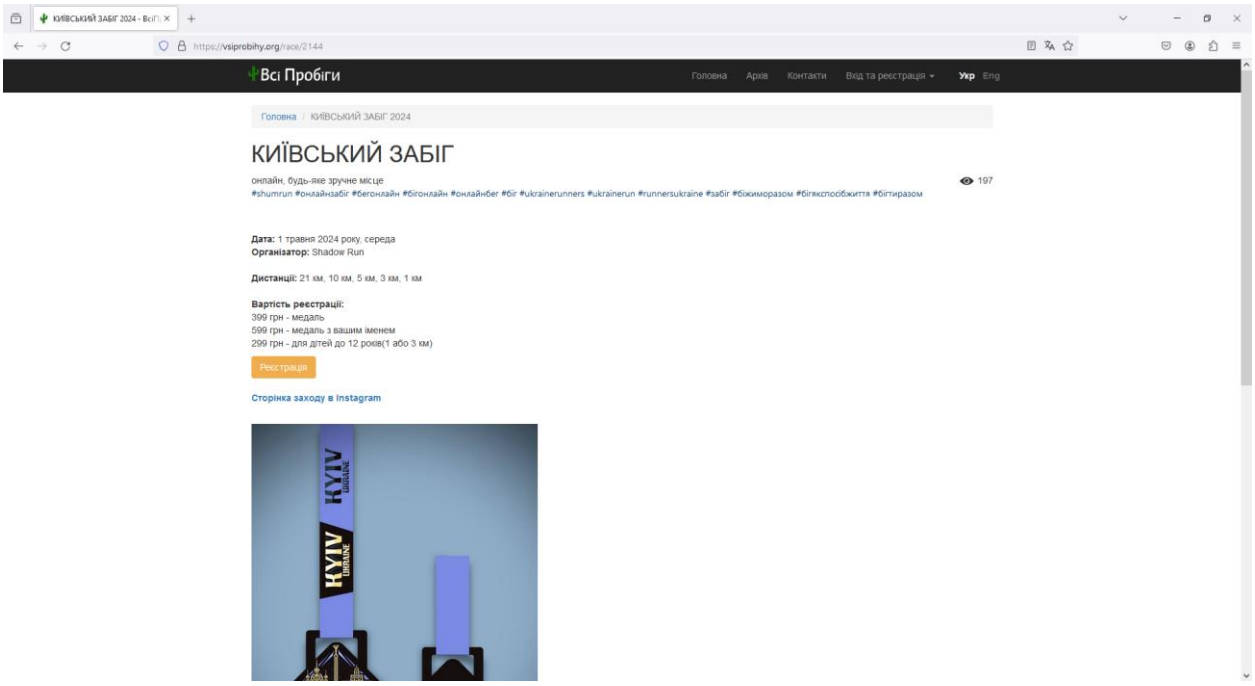


Рис. 1.7 Сторінка змагання на сервісі ВсіПробіги

В таблиці 1.1 наведено порівняння вище досліджених існуючих аналогів web-сервісу «Календар спортивних змагань».

Таблиця 1.1 – Зведена таблиця з характеристиками існуючих аналогів web-сервісу «Календар спортивних змагань»

Критерії порівняння	**Timing Events**	**Ukraine Sport Events**	**ВсіПробіги**	**Sports Events**
1	2	3	4	5
Реєстрація на всі види змагань	+	+	-	+
Результат змагання учаснику на email або смс	+	+	-	+
Можливість підключення системи електронного хронометражу	+	+	-	+
Повний доступ для керування налаштуваннями реєстрації	тільки для партнерів	інформація недоступна	для всіх організаторів	для всіх організаторів
Редагування/ анулювання учасником своєї реєстрації	-	-	-	+
Можливість передати свою реєстрацію іншому учаснику	-	-	-	+
Самостійний вибір стартового номеру учасником	-	-	-	+
Зручність реєстрації на веб-сервісі	Аккаунт створюється після заповнення форми реєстрації на спортивний захід	Можливість реєстрації та входу за допомогою соцмереж	Класична реєстрація та вхід за допомогою email та паролю	Можливість реєстрації та входу за допомогою соцмереж або аккаунту Google

Продовження таблиці 1.1 – Зведена таблиця з характеристиками існуючих аналогів web-сервісу «Календар спортивних змагань»

1	2	3	4	5
Використання з мобільних пристроїв	+	-	-	+

1.3 Вимоги до програмного забезпечення

По результатах проведеного аналізу організації спортивних змагань та реєстрації на них і дослідження існуючих аналогів можна сформувати функціональні та нефункціональні вимоги до web-сервісу.

Функціональні вимоги до web-сервісу:

1. Організатор:
   - повинен мати можливість створювати, редагувати та видаляти спортивне змагання;
   - повинен мати можливість створення аккаунту учасника, редагування даних учасника;
   - може зареєструвати учасника на змагання, редагувати дані реєстрацій учасників на змагання та видаляти реєстрації на змагання.
2. Учасник:
   - може переглядати список спортивних змагань, детальну інформацію по змаганню та результати спортивного змагання;
   - може реєструватись на спортивні змагання та здійснювати їх онлайн-оплату;
   - повинен мати можливість анулювати свою реєстрацію або передати свою реєстрацію іншому учаснику.
3. Веб-сервіс:
   - повинен забезпечувати реєстрацію на всі види спортивних змагань;
   - надає можливість реєстрації та входу на сервіс як за допомогою логіну та паролю, так і за допомогою аккаунту Google або соцмереж.

Нефункціональні вимоги:

1. Web-сервіс має бути доступний і повністю працездатний 24/7.
2. Час завантаження сторінки не повинен перевищувати 2 секунди.
3. У системі має бути присутня авторизація користувачів, обмеження доступу згідно ролей користувачів.
4. Система має надавати гарантію конфіденційності даних користувачів, вся інформація про користувача має використовуватися лише за призначенням.
5. Система повинна мати інтуїтивно зрозумілий та зручний інтерфейс користувача.

2 ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Вибір засобів розробки серверної частини

Існує широкий вибір мов програмування для розробки серверної частини web-сервісів. Найбільш поширені – це Python, Java, C#.

Java та C# мають більший поріг входу, більш складний синтаксис та у світі веб-розробки більш орієнтовані на розробку складних рішень Enterprise класу. Мова Python більше підходить для розробки e-commerce рішень.

Станом на травень 2024 року по рейтингу TIOBE [7] мова Python займає перше місце із зростанням на 2,88% відносно травня 2023 року.

Python - проста у вивченні та потужна мова програмування. Вона має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис та динамічна типізація разом із інтерпретованою природою роблять його ідеальною мовою для написання сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Інтерпретатор Python та обширна стандартна бібліотека доступні у вихідному або двійковому вигляді для всіх основних платформ на web-сайті Python [8] і можуть вільно поширюватися. Цей же сайт містить дистрибутиви та вказівники на багато безкоштовних сторонніх модулів Python, програм та інструментів, а також додаткову документацію.

Інтерпретатор Python легко розширюється за допомогою нових функцій і типів даних, реалізованих у C або C++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для програм, які підтримують плагіни [9].

Python підвищує продуктивність праці розробників в багато раз в порівнянні з компільованими і статично типізованими мовами програмування C, C++ та Java. Код Python звичайно займає від однієї третьої до однієї п’ятої частини розміру еквівалентного коду C++ або Java. В результаті приходиться менше набирати на клавіатурі, менше відладжувати і менше внаслідок супроводжувати. Крім того, програми на Python запускаються негайно, без тривалих кроків компіляції та зв’язування, які потрібні в ряді інших інструментів, що додатково підвищує швидкість роботи програмістів.

Разом з Python поставляється велика колекція попередньо зібраної функціональності, яка називається стандартною бібліотекою. Стандартна бібліотека підтримує множину рішень програмних задач прикладного рівня починаючи від співставлення тексту і закінчуючи сценаріями для мереж. До того ж, Python можна розширювати бібліотеками власної розробки і обширним набором прикладного ПЗ, яке створене сторонніми розробниками. Область стороннього ПЗ

для Python пропонує інструменти, які призначені для конструювання веб-сайтів, чисельного програмування, доступу до послідовних портів, розробки ігор та багато іншого [10].

Мову програмування Python використовують такі відомі компанії, як Google, DropBox (в цій компанії працює творець мови Python Гвідо ван Россум), Facebook, Instagram, Netflix, Reddit.

Враховуючи швидкість розробки, лаконічність синтаксису, наявність великої кількості бібліотек та гарну документацію було вирішено обрати мову Python для розробки серверної частини сервісу.

Для реалізації серверної частини сервісу був обраний фреймворк Django. Django дуже популярний Python фреймворк. Є і інші фреймворки, але фактично поєднання Python + Django є класичним .

Django – це високорівневий фреймворк Python, який заохочує швидку розробку та чистий, прагматичний дизайн. Створений досвідченими розробниками, він впорається з більшою частиною клопоту веб-розробки, тому можна зосередитися на написанні свого додатка без потреби винаходити колесо. Фреймворк безкоштовний та з відкритим кодом [11].

Проект Django складається із одного або декількох додатків, які разом утворюють повноцінний web-додаток. Кожний додаток представляє якусь функціональність або групу функціональностей. Тому можливо виділити задачу або групу задач в окремий модуль і розробляти їх незалежно від інших. Внаслідок цього є можливість переносити додаток із одного додатка в інший незалежно від іншої функціональності проекту.

Django використовують такі компанії, як Mozilla, NASA, Pinterest, Bitbucket, Instagram.

Для реалізації REST API було обрано бібліотеку DRF (Django Rest Framework) [12], яка легко інтегрується з Django. Вона працює з моделями Django і дозволяє створювати гнучкі та потужні API.

В якості СКБД для проекту обрано PostgreSQL. PostgreSQL – це потужна об’єктно-реляційна база даних з відкритим вихідним кодом, активна розробка якої триває понад 35 років, завдяки чому вона заслужила міцну репутацію надійності та продуктивності [13].

2.2 Вибір засобів реалізації клієнтської частини

Для реалізації фронтенд частини веб-сервісу використовуються такі, де-факто стандартні, технології як HTML 5, CSS 3 та JavaScript. Також для єдиної стилізації всіх веб-сторінок було обрано бібліотеку Bootstrap. Bootstrap – це потужний, багатофункціональний інтерфейсний інструментарій, який допомагає розробникам створювати адаптивні, а також орієнтовані на мобільні пристрої web-

сайти. Bootstrap надає готові компоненти та інструменти, які допомагають розробникам створювати узгоджені та зручні веб-інтерфейси [14].

Однією з переваг Bootstrap є його гнучкість. Розробники можуть змінювати його компоненти CSS та JavaScript згідно своїх вимог. Також він включає ряд плагінів, які розробники можуть використовувати для розширення функціональності. Також перевагою Bootstrap являється адаптивний дизайн. Bootstrap побудований на основі адаптивної сітки, що гарантує коректний вигляд сайту на будь-якому пристрої.

Bootstrap включає в собі готові компоненти, що спрощує для розробника створення зручних веб-інтерфейсів. Це такі компоненти, як форми, кнопки, меню навігації, модальні вікна, оповіщення та інші.

JavaScript є потужною мовою програмування, можна сказати одним із стандартів де-факто у сферу веб-розробки. JavaScript став важливим інструментом для надання динамічності та інтерактивності веб-сторінкам. Мова постійно розвивається, регулярно додаються нові функції.

Для реалізації клієнтської частини в якості JavaScript фреймворку був обраний фреймворк Vue.js [15]. Це прогресивний фреймворк для створення користувацьких інтерфейсів. Він повністю підходить для реалізації одно сторінкових (SPA) додатків. Vue.js характеризується швидкість налаштування та легкістю освоєння, що робить його гарним вибором для тих розробників, які недостатньо знають JavaScript або яким необхідно швидко створити веб-додаток.

Важливою концепцією Vue.js є компоненти. Вони являють собою придатні до повторного використання об’єкти. Структурно це дозволяє збирати великі додатки з маленьких кусочків. Це пов’язано з тим, що майже будь-який інтерфейс можна представити як дерево компонентів.

Додаток з кореневого екземпляру Vue, який опціонально вбудований в дерево вкладених, повторно використовуваних компонентів. Vue.js реактивно пов’язує DOM та дані, які ми хочемо відобразити. Завдяки такій реалізації взаємодії напряму з HTML більше не потрібно. HTML є точкою входу, а все інше відбувається всередині створеного екземпляру Vue.

При створенні SPA додатків для співставлення компонентів додатку з маршрутами, які доступні по заданому URL, використовують бібліотеку Vue Router.

2.3 Вибір середовища розробки

Для розробки використовувались IDE PyCharm Professional Edition та WebStorm компанії JetBrains [16]. Це одні з найкращих IDE для програмування на мові Python та JavaScript відповідно. PyCharm має дві версії – безкоштовну Community Edition та платну Professional Edition. Остання має значно розширені можливості, як от підтримка мов Python, SQL, а також Python фреймворків Django,

FastAPI, Flask. WebStorm підтримує мову JavaScript, а також JavaScript фреймворки Vue.js, Angular, React. Обидва IDE основані на IntelliJ IDEA.

Так, як PyCharm та WebStorm повноцінні IDE, то вони мають «з коробки» всі необхідні інструменти для розробки, рефакторингу, профілювання та відладки коду. Крім того, присутня вбудована підтримка Git та Docker. На рис. 2.1 показаний приклад вікна IDE PyCharm.

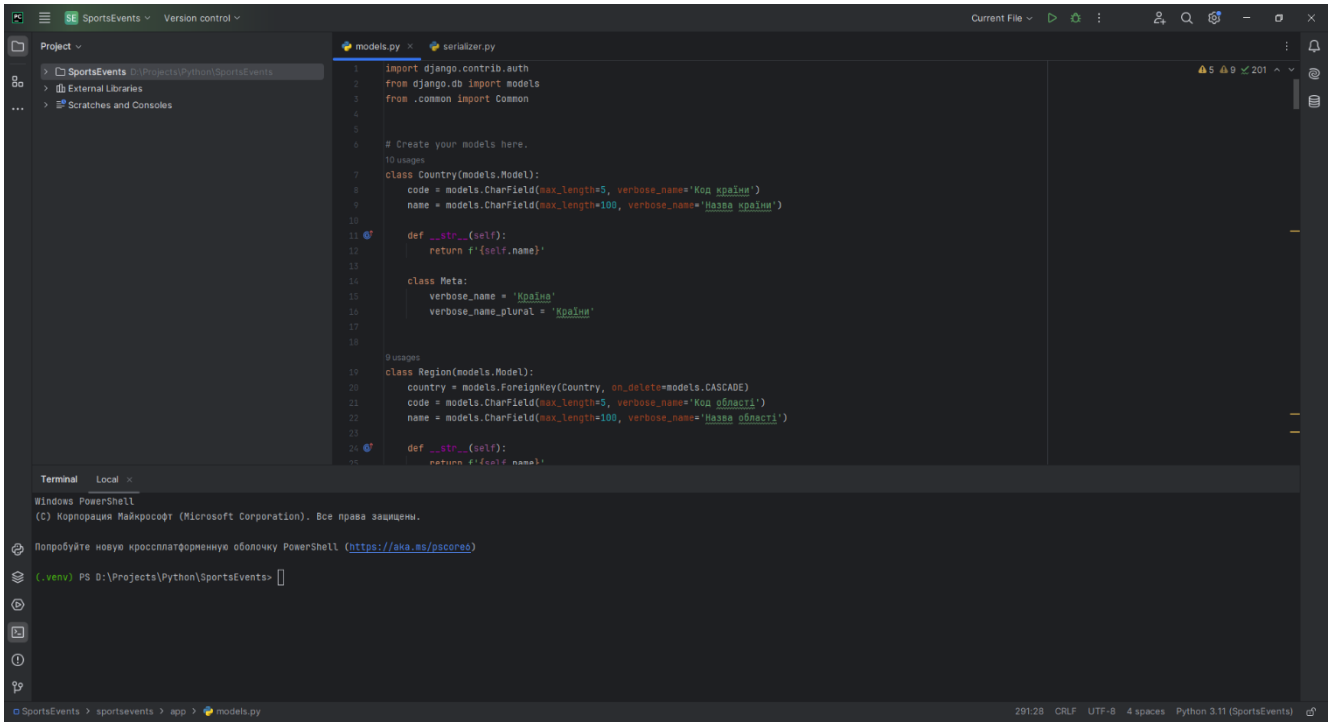


Рис. 2.1 Приклад вікна IDE PyCharm

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Діаграми прецедентів

Діаграма прецедентів являє собою UML діаграму, що зображує відношення між актором та прецедентом в системі. Суть діаграми в тому, що проектована система показується у вигляді множин сутностей та акторів, які взаємодіють з системою при допомозі варіантів використання. Кожним варіантом використання описується набір дій, який використовує система при взаємодії з користувачем.

Основні елементи діаграми прецедентів:

1) Актор – сутність, яка взаємодіє з системою. Позначається на діаграмі як фігурка «чоловічка». Під фігуркою вказується назва актора. Актором може виступати не тільки людина, але і будь-яка інша система.

2) Прецедент – це набір послідовних дій, які виконує система для отримання актором певного результату. Стандартним позначенням являється еліпс, всередині якого вказана назва прецеденту.

Між актором та прецедентом можуть існувати різні відносини, які описують взаємодію між ними.

Можна виділити такі відносини:

1. Асоціація (association) – визначає зв’язок між актором і прецедентом. Позначається суцільною лінією.
2. Розширення (extend) – визначає взаємозв’язок одного прецеденту з іншим, більш загальним. Позначається пунктирною лінією в напрям від прецеденту, що є розширенням, до вихідного прецеденту. Підписується ключовим словом «extend».
3. Включення (include) – вказує, що поведінка одного прецеденту є складовою частиною поведінки іншого. Позначається пунктирною лінією від базового прецеденту до того прецеденту, що включається. Підписується ключовим словом «include».

Веб-сервіс «Календар спортивних змагань» має чотири актори, які взаємодіють з системою. Це такі актори, як Гість, Учасник, Організатор та Адміністратор.

Актор Гість – це неавторизований користувач. У нього максимально обмежений доступ до ресурсу. Він може переглядати календар спортивних змагань та детальну інформацію по кожному змаганню, а також результати спортивного змагання. Також для нього доступні функції авторизації, якщо користувач раніше вже створював аккаунт, або реєстрації на веб-сервісі. На рис. 3.1 представлена діаграма прецедентів актора Гість.

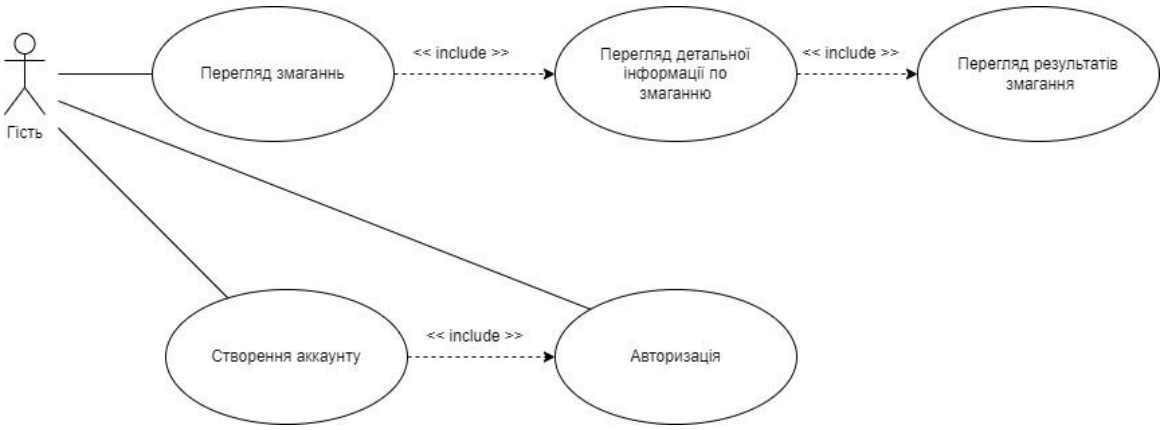


Рис. 3.1 Діаграма прецедентів актора Гість

Актор Учасник – це авторизований користувач. Як і гостю, учаснику доступний перегляд змагань, детальної інформації по змаганню та результатів змагання. Крім цього для цього актора доступна реєстрація на змагання, яка також включає в себе редагування або видалення реєстрації на змагання та передачу своєї реєстрації іншому учаснику. Учасник може провести оплату реєстрації як в кінці самого процесу реєстрації на змагання, так і окремою дією. Також актору Учасник доступне редагування даних свого профілю користувача. На рис. 3.2 представлена діаграма прецедентів актора Учасник.

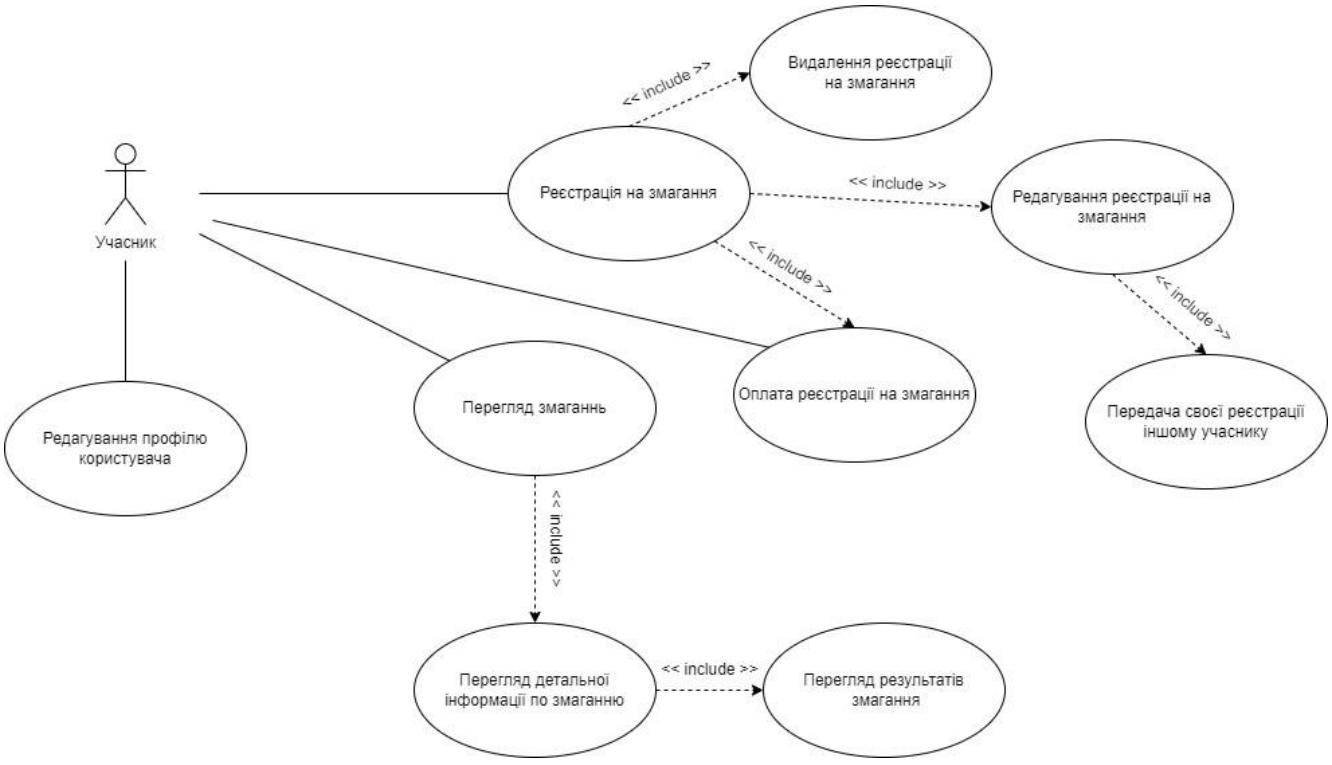


Рис. 3.2 Діаграма прецедентів актора Учасник

Актор Організатор (на рис. 3.3 приведена діаграма прецедентів даного актора) – це також авторизований користувач, якому доступні як всі функції актора Учасник, так і наступні функції (нижче під спортивними змаганнями маються на увазі змагання, створені саме цим конкретним організатором; доступ до спортивних змагань інших організаторів тільки на перегляд):

- створення спортивного змагання в календарі;
- редагування спортивного змагання, що також включає в себе внесення результатів змагання;
- видалення спортивного змагання з календаря змагань;
- створення аккаунтів учасників;
- реєстрація інших учасників на змагання;
- редагування реєстрацій учасників, в тому числі з передачею реєстрації на змагання іншому учаснику;
- видалення реєстрацій на спортивне змагання учасників.

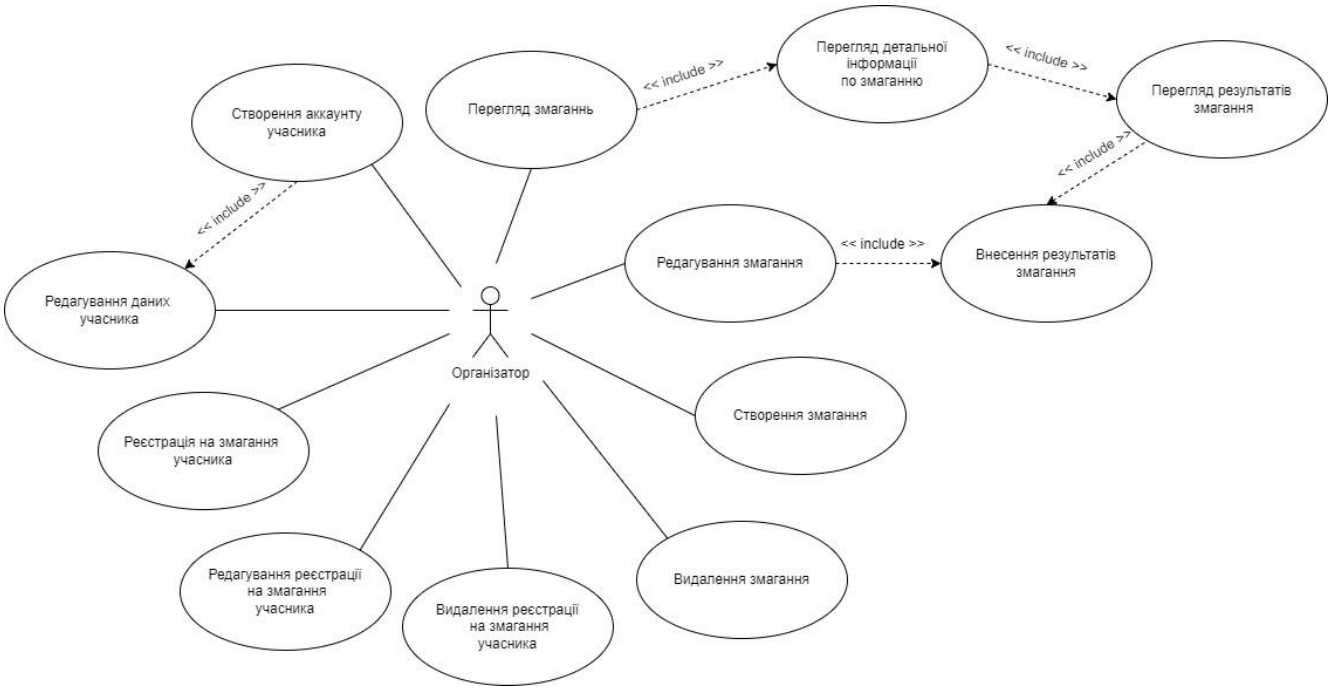


Рис. 3.3 Діаграма прецедентів актора Організатор

Актор Адміністратор – це авторизований користувач, який має повний доступ до всіх функцій веб-сервісу. Актор Адміністратор має доступ до всіх функцій, які доступні попереднім акторам, крім цього він має доступ на редагування спортивних змагань всіх організаторів. Актор Адміністратор може змінювати ролі всіх користувачів та редагувати їх права, а також може блокувати користувачів. На рис. 3.4 наведена діаграма прецедентів даного актора.

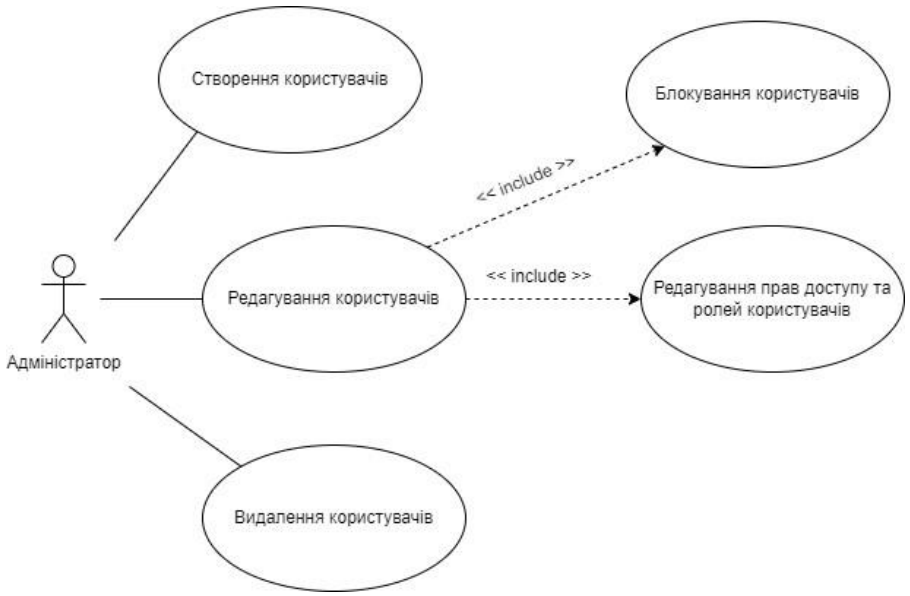


Рис. 3.4 Діаграма прецедентів актора Адміністратор

3.2 Діаграма класів

Діаграма класів – це важливий інструмент для опису програмного забезпечення. Вона надає можливість візуалізації структури системи, компонентів і зв’язків між ними з точки зору класів і їх властивостей. На етапі проектування зазвичай не так важливо, на якій мові буде розроблюватись програмне забезпечення. Цінність діаграми класів також в тому, що на ній можна відобразити тільки важливі сутності. На рис. 3.5 приведено діаграму основних класів. Це класи моделей Django.

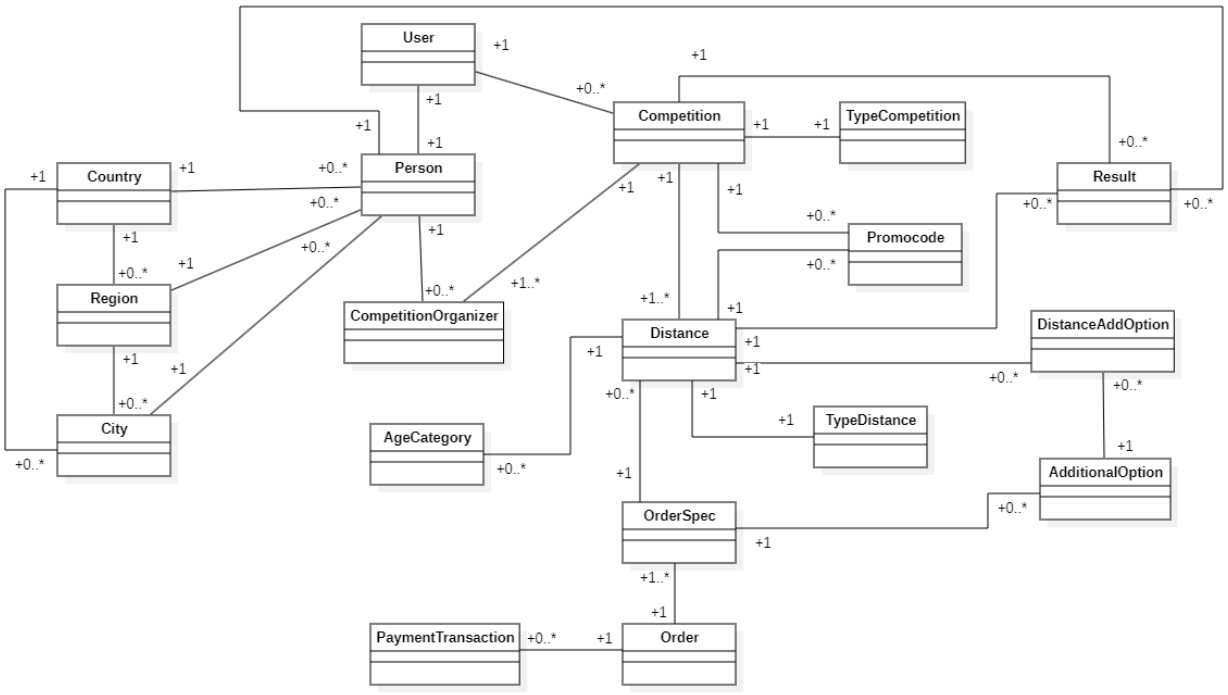


Рис.3.5 Діаграма основних класів веб-сервісу

3.3 Схема бази даних

Django ORM генерує схему бази даних на основі класів моделей Django за допомогою механізму міграцій. Міграції – це спосіб Django розповсюджувати зміни, які були внесені до класів моделей, в схему бази даних. При створенні таблиць в базі даних Django ORM до імені класів моделей додає префікс, який є іменем додатку Django, і таким чином формує унікальну назву таблиці на рівні бази даних. Схема бази даних зображена на рис. 3.6.

База даних складається з наступних таблиць (в списку і на схемі бази даних не вказані системні таблиці Django, окрім auth_user):

- app_country – довідник країн.
- app_region – довідник областей (або регіонів)
- app_city – довідник міст (населених пунктів)
- auth_user – довідник користувачів - системна таблиця Django. Вносити зміни в таку таблицю не бажано. Тому для можливості збереження розширеної інформації по користувачу було створено таблицю app_person.
- app_person – таблиця розширеної інформації користувача (профіль користувача).
- app_typecompetition – довідник типів спортивних змагань.
- app_competition – таблиця спортивних змагань.
- app_competitionorganizer – таблиця організаторів спортивних змагань. Так, як у спортивного змагання може бути більше одного організатора, то ця таблиця використовується для зв’язування таблиць app_competition та app_person.

- app_additionaloption – довідник додаткових опцій.

- app_typedistance – довідник типів дистанцій (або етапів) спортивного змагання.

- app_distance – таблиця дистанцій (етапів) спортивного змагання.

- app_distanceaddoption – таблиця додаткових опцій дистанції (етапу) змагання.

- app_agecategory – таблиця вікових категорій дистанцій (етапів) змагання.

- app_promocode – таблиця промокодів.

- app_order – таблиця замовлень.

- app_orderspec – таблиця специфікацій замовлень. Реєстрація на змагання оформлюється в системі як замовлення. В записах специфікації замовлення зберігається інформація про деталі реєстрації (дистанція, стартовий номер) та замовлені додаткові опції (це може бути медаль, футболка і т.п., залежить від конкретного змагання).

- app_paymenttransaction – таблиця платіжних транзакцій.

- app_result – таблиця результатів змагання.

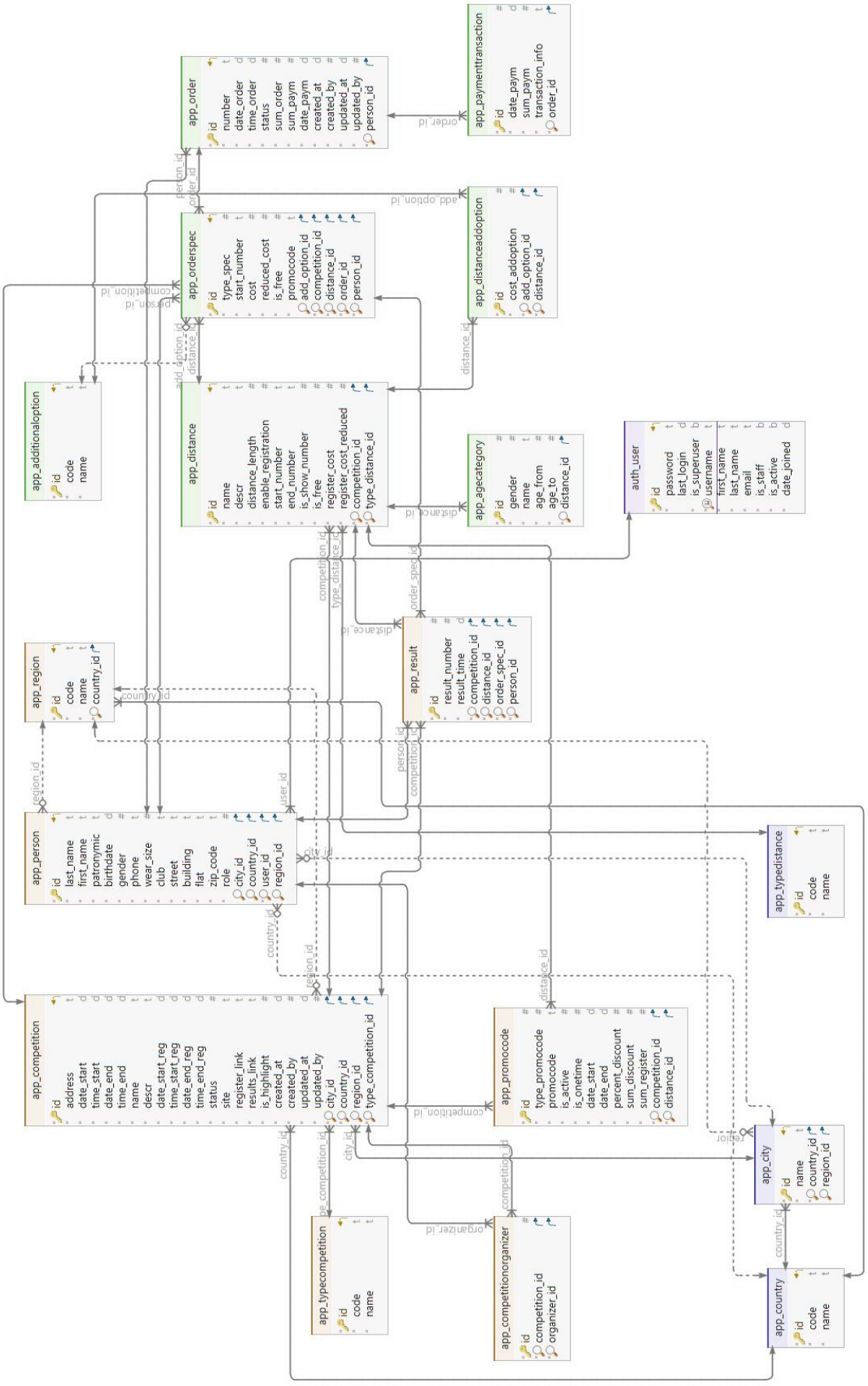


Рис. 3.6 Схема бази даних

3.4 Архітектура програмного додатку

Для реалізації web-сервісу було обрано клієнт-серверна архітектура, що складається з трьох рівнів – інтерфейс користувача (клієнтська частина), серверна частина та база даним з REST API для зв'язку між серверною та клієнтською частинами.

REST API (також ще називають RESTful API або RESTful web API) – це архітектурний стиль взаємодії компонентів розподіленого додатку в мережі. Рой Філдінг у 2000 році у своїй докторській дисертації виклав концепцію REST [17]. Він виділив 5 обов'язкових обмежень для створення розподілених REST застосунків та одне необов'язкове:

1. Модель клієнт-сервер – розмежування потреб є принципом, що є в основі цього обмеження. При розподілі потреб інтерфейсу клієнту від потреб сервера підвищується переносимість коду клієнтського на інші платформи. При спрощенні серверної частини покращується масштабування.

2. Відсутність стану – протокол взаємодії між сервером та клієнтом не зберігає якогось сесійного стану після запиту та відповіді. За потреби – стан може зберігатися на клієнті. Тільки в цьому випадку користувач відв'язаний від конкретного сервера, що, в свою чергу, дає можливість масштабуватися і без проблемно балансувати навантаження між серверами.

3. Кешування – кожен із клієнтів та проміжні вузли між клієнтами та сервером мають змогу кешувати відповіді сервера. У кожному запиті має явно вказуватися про можливість кешування відповіді та отримання відповіді із існуючого кешу. Вірне використання кешування дозволяє видалити надлишкові взаємодії клієнта та сервера. А це, в свою чергу, покращує швидкість та розширюваність системи.

4. Уніфікований інтерфейс. На уніфіковані інтерфейси накладаються чотири умови – ідентифікація ресурсів; маніпулювання інтерфейсами через представлення; само описові повідомлення; гіпермедіа, як двигун стану застосунку. Кожний URI є ідентифікатором або одиночного ресурсу, або колекції ресурсів. Необхідні дії задаються за допомогою використання відповідного HTTP методу (GET, POST, PATCH, PUT, DELETE).

5. Шари абстракцій – клієнт може взаємодіяти з сервером не напряму, а через проміжні вузли або шари. При цьому клієнт може не підозрювати про їх існування (окрім випадків передачі конфіденційної інформації). Проміжні сервери можуть виконувати кешування та можуть виконувати балансування навантаження.

6. Запитування коду (code on demand) – іноді відповіді від REST API можуть містити виконуваний код (аплети Java або сценарії) і в таких випадках код має виконуватися тільки на вимогу.

Філдінг вказував [16], що додатки, які не відповідають даним вимогам, не можуть рахуватись REST-додатками. Якщо ж всі умови виконані, то на його думку, додаток отримає наступні переваги – надійність, продуктивність, масштабованість, прозорість системи взаємодії, простоту інтерфейсів, портативність компонентів, легкість внесення змін, здатність еволюціонувати, пристосовуватися до нових вимог.

Загальна архітектура сервісу зображена у вигляді діаграми пакетів на рис. 3.7.

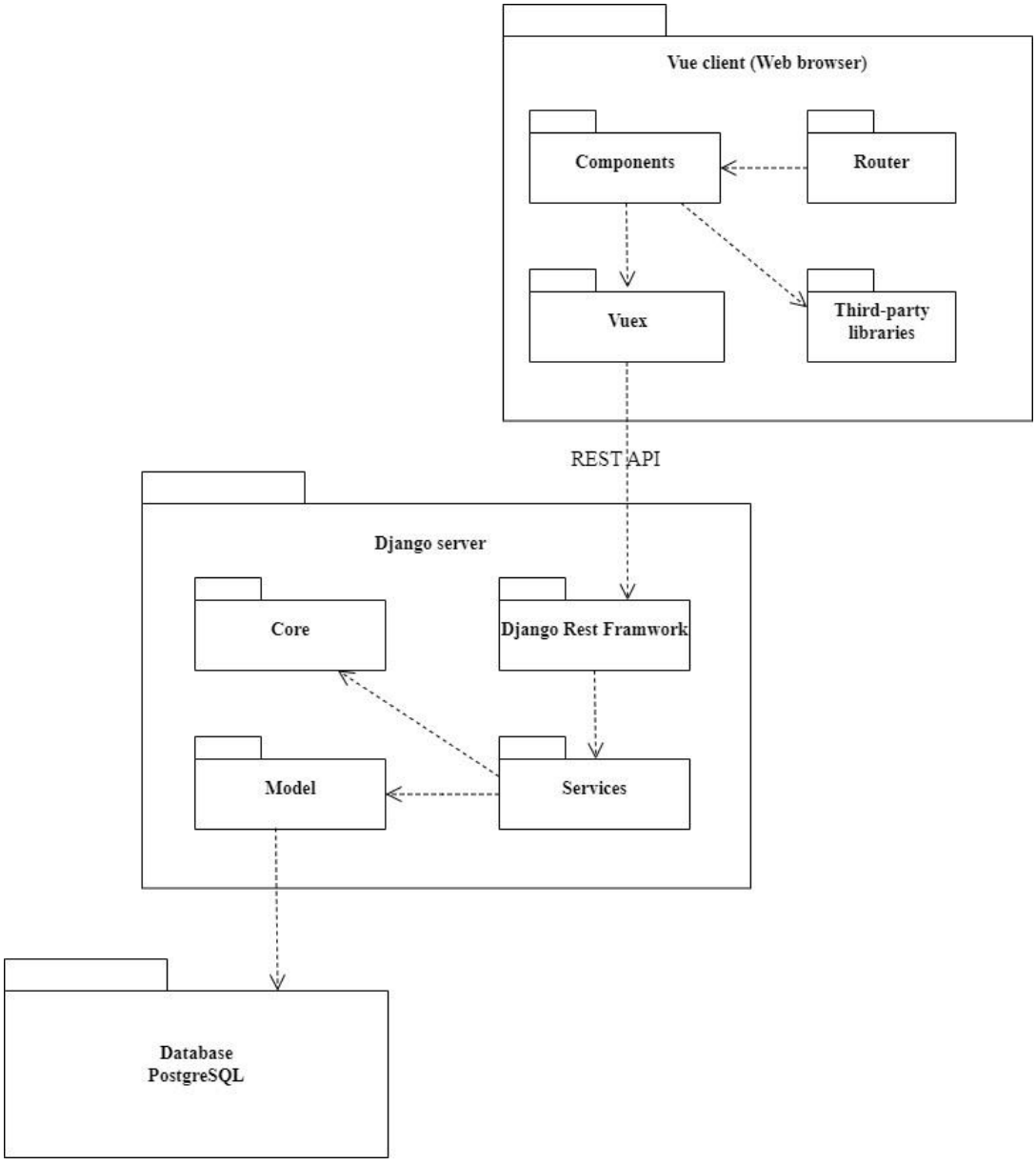


Рис. 3.7 Діаграма пакетів

3.5 Карта web-сервісу

На рис. 3.8 показана карта web-сервісу.

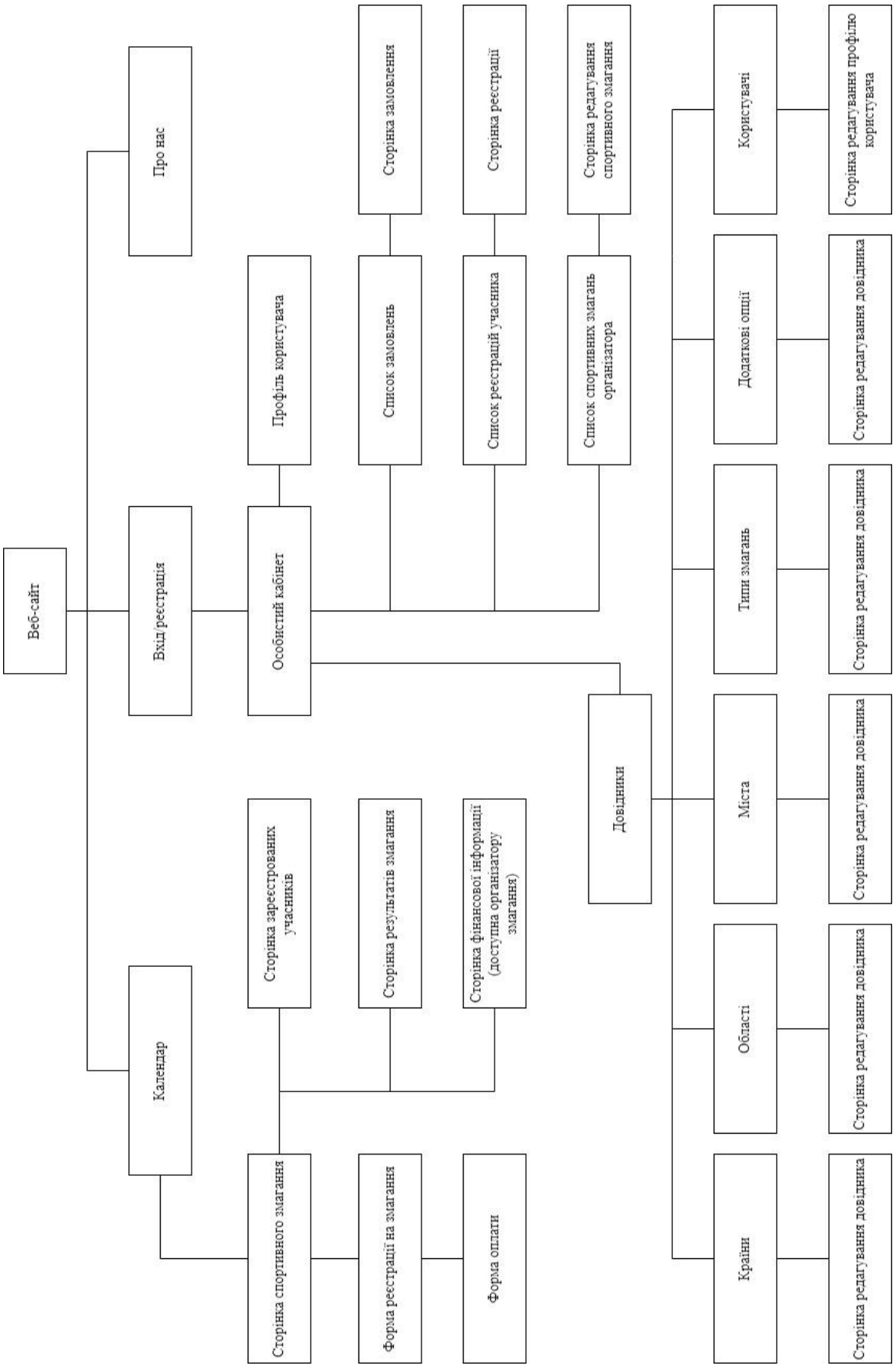


Рис. 3.8 Карта web-сервісу

Карта веб-сайту – це файл чи сторінка, яка містить список всіх доступних сторінок веб-сайту. Цей список, як правило, має структуровану ієрархію, що відображає посилання між сторінками. Карта веб-сайту використовується пошуковими системами Google, Bing та іншими для розуміння структури веб-сайту при індексації сторінок сайту. Це надає можливість пошуковій системі оновлювати свою базу даних посилань на веб-сайт та показувати його у списку результатів пошуку.

Карта сайту є корисною також і для веб-розробників і користувачів, оскільки вона надає огляд загальної структури сайту.

Карта сайту може бути в XML форматі, який прийнятий пошуковими системами, або в HTML форматі, який більш придатний для перегляду людиною.

3.6 Розробка основних модулів web-сервісу

Першим кроком створимо проєкт у середовищі PyCharm. Вид вікна показано на рис. 3.9. Робимо вибір розташування проєкту, розташування каталогу віртуального оточення та версію інтерпретатора Python.

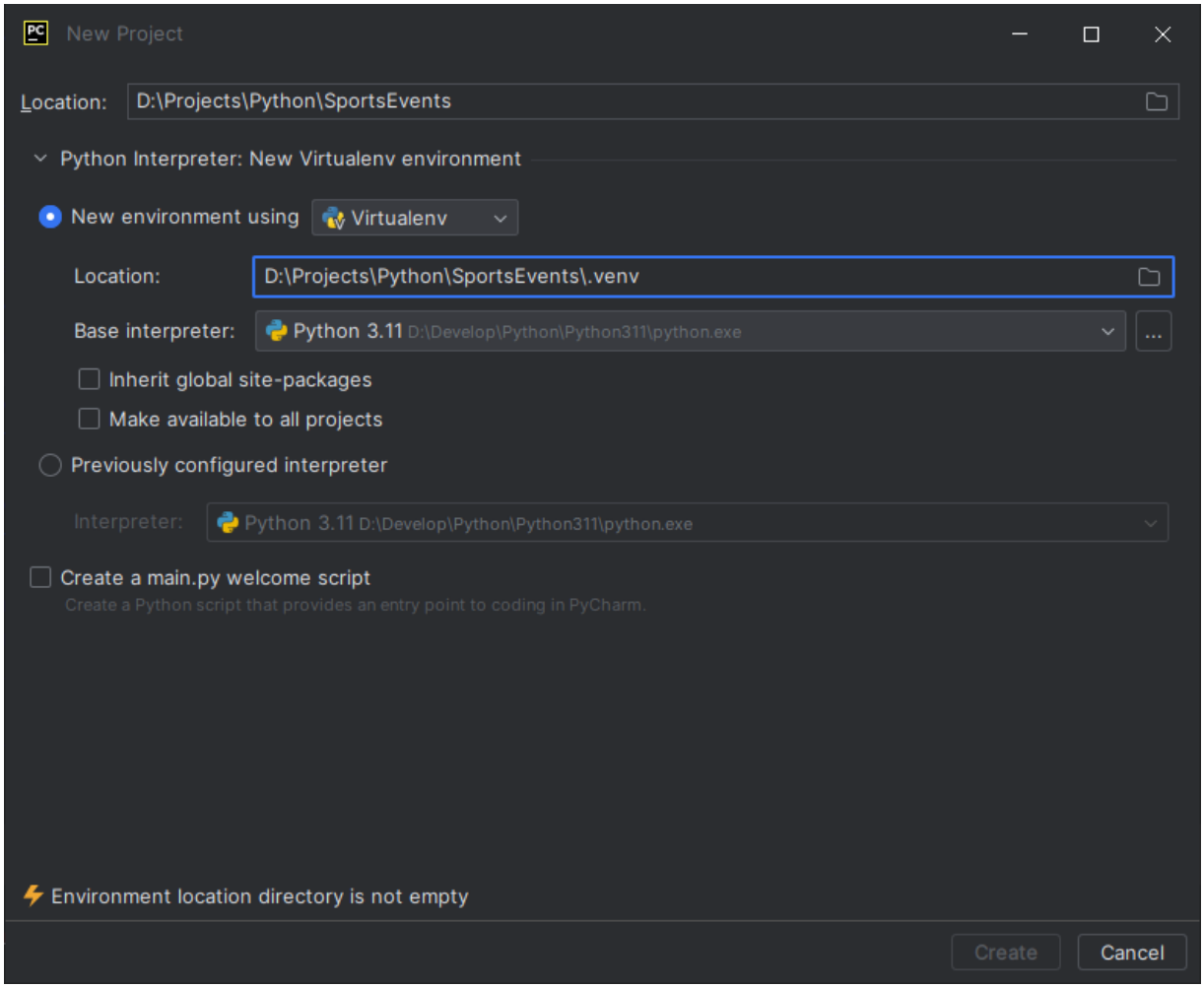


Рис. 3.9 Вікно створення нового проєкту PyCharm

Далі встановимо фреймворк Django та бібліотеку Django Rest Framework. Для цього у вбудованому терміналі PyCharm виконаємо наступні команди:

pip install django
pip install djangorestframework
pip install markdown
pip django-filter

Тепер створимо проєкт Django командою
django-admin startproject sportsevents
Далі переходимо в каталог sportsevents і створимо додаток:
python manage.py startapp app
На рис. 3.10 зображена структура проєкта серверної частини в PyCharm.

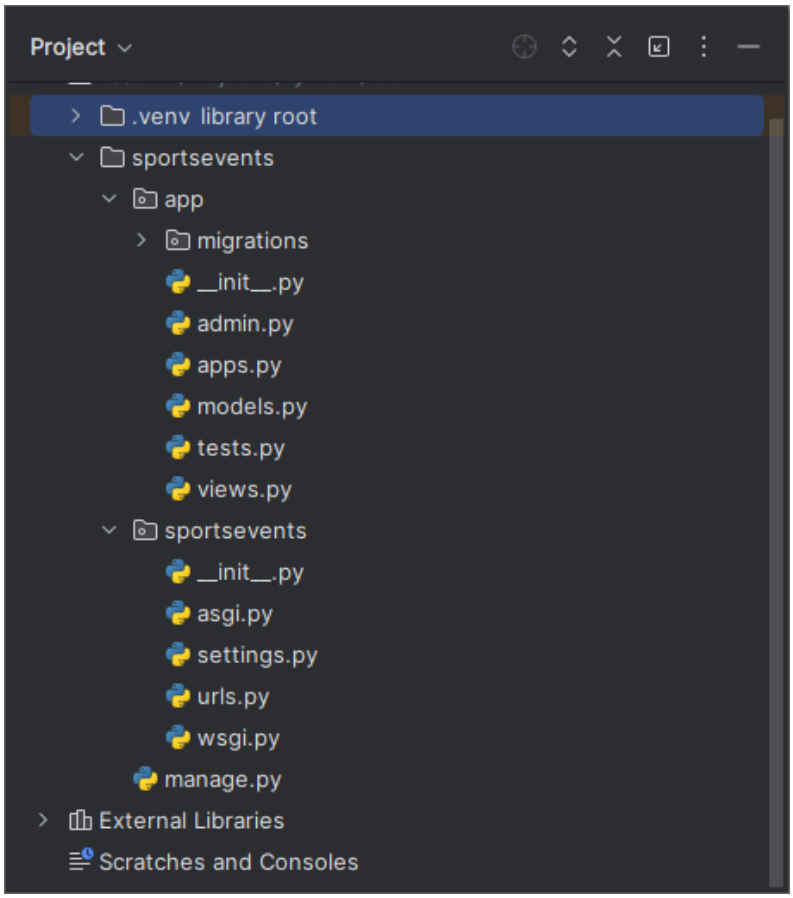


Рис. 3.10 Структура проєкта серверної частини

Тепер потрібно зробити налаштування в файлі settings.py, а саме підключити створений додаток та Django Rest Framework. Для цього в секцію INSTALLED_APPS додаємо рядки ‘rest_framework’ та ‘app’.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'app'
]
```

Далі зробимо налаштування з’єднання з базою даних, зробивши відповідні налаштування в секції DATABASES файлу settings.py.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'sportsevents',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Зроблено початкове налаштування Django та DRF. Можна приступати до розробки моделей нашого додатку.

Модель Django – це єдине і кінцеве джерело інформації про ваші дані. Вона містить основні поля та поведінку даних, які ви зберігаєте. Ціль в тому, щоб визначити вашу модель даних в одному місці і автоматично отримувати з неї дані. Це включає в себе міграції, які повністю виходять із файлу ваших моделей і являють собою історію, яку Django може перегорнути для того, щоб оновити схему бази даних для відповідності її поточному стану моделей додатку [11].

Деякі поля моделей фактично передбачають вибір із списку значень (так/ні, розміри футболок і т.д.) було винесено в окремий клас Common в файл common.py.

Файл models.py показано на рис. 3.11.

Після розробки моделей потрібно підготувати та виконати міграції. Робимо це за допомогою команд:

python manage.py makemigrations

python manage.py migrate

В результаті виконання міграцій будуть створені об’єкти бази даних. Далі створимо супер користувача (адміністратора) Django:

python manage.py createsuperuser

Після запуска вищеозначеної команда потрібно ввести ім’я користувача, email та пароль. На рис. 3.12 зображено таблиці, створені за допомогою міграцій.

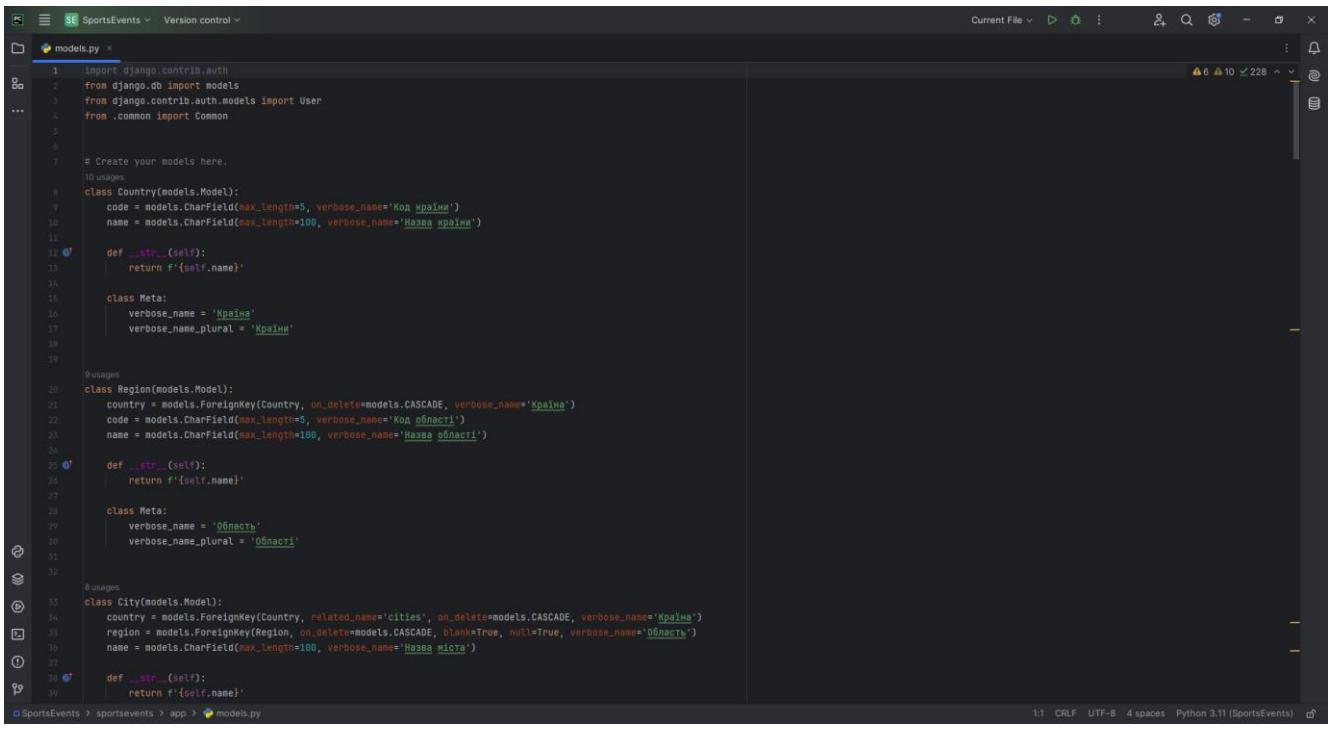


Рис. 3.11 Приклад файлу models.py з кодом моделей

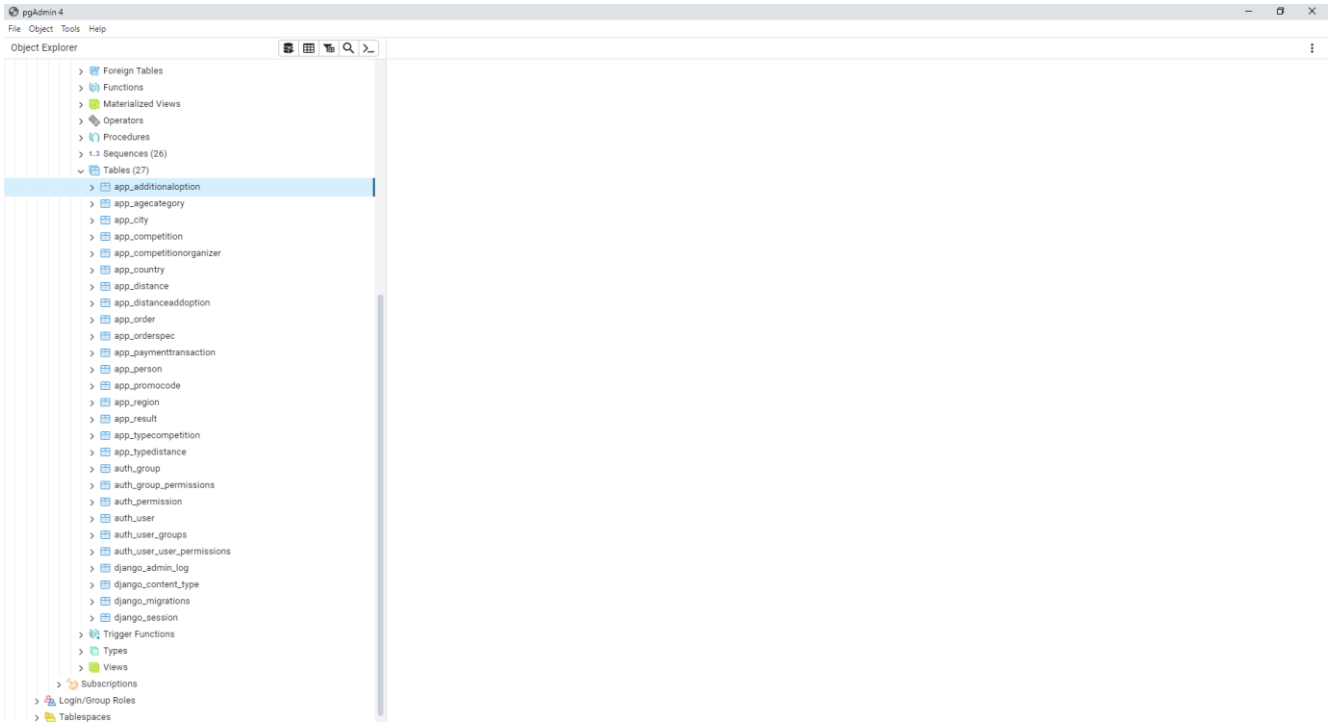


Рис. 3.12 Вікно pgAdmin4 із списком таблиць

На рис. 3.13 відображено вікно із полями таблиці спортивних змагань.

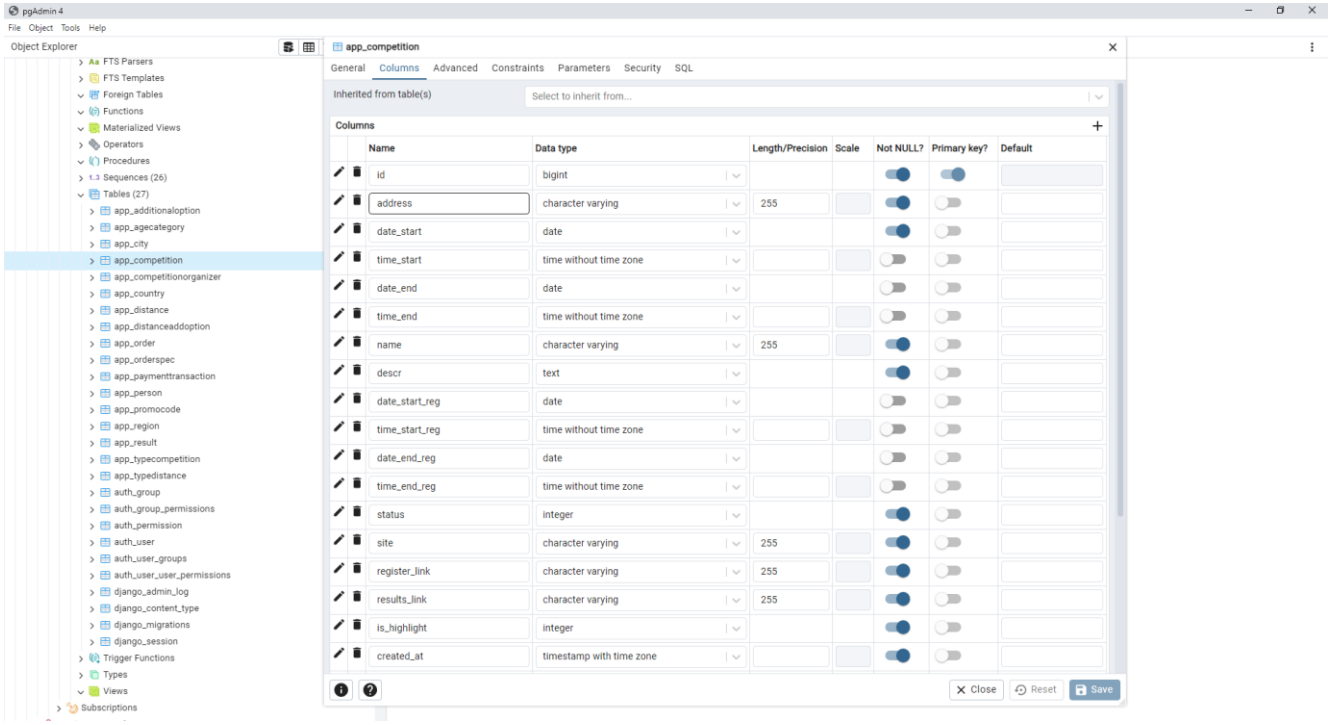


Рис. 3.13 Структура таблиці app_competition

Для можливості авторизації за допомогою JWT токена потрібно підключити в файлі бібліотеку rest_framework_simplejwt.

Приступаємо до реалізації REST API. Спочатку визначимо серіалізатори до наших моделей. Для цього створимо модуль serializers.py. Приклад файлу serializers.py відображено на рис. 3.14.

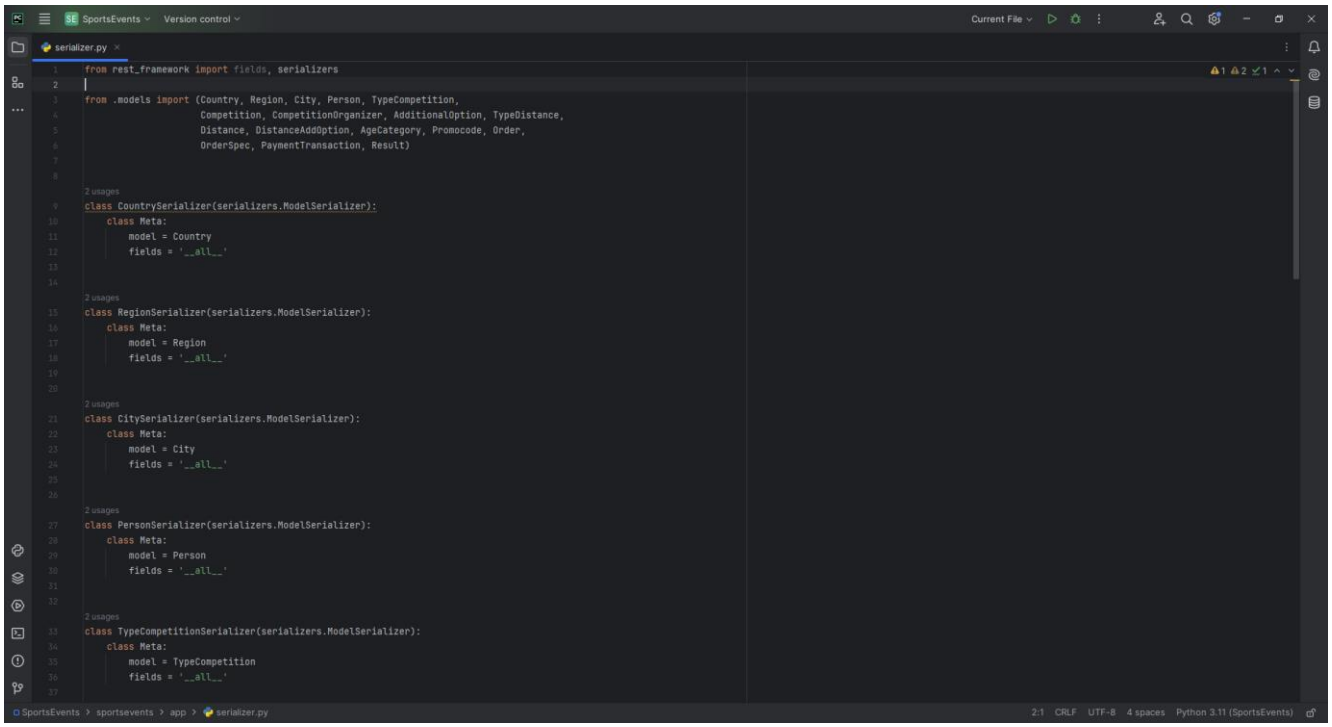


Рис. 3.14 Приклад файлу serializers.py

Так, як Django використовує модель рівневої архітектури MTV (Model – Template - View), яка є близькою до архітектури MVC, то рівень View в Django приблизно відповідає рівню Controller архітектури MVC. Рівень View в Django інкапсулює логіку, яка відповідає за обробку запиту користувача та повернення йому відповіді. Отже, напишемо код «видів», об’єднавши всю загальну поведінку CRUD в класи ViewSets, які будуть наслідувати клас ModelViewSet. Також для авторизації за допомогою JWT токенів додаємо в код «видів» базові дозволи. На рис. 3.15 приклад файлу views.py.

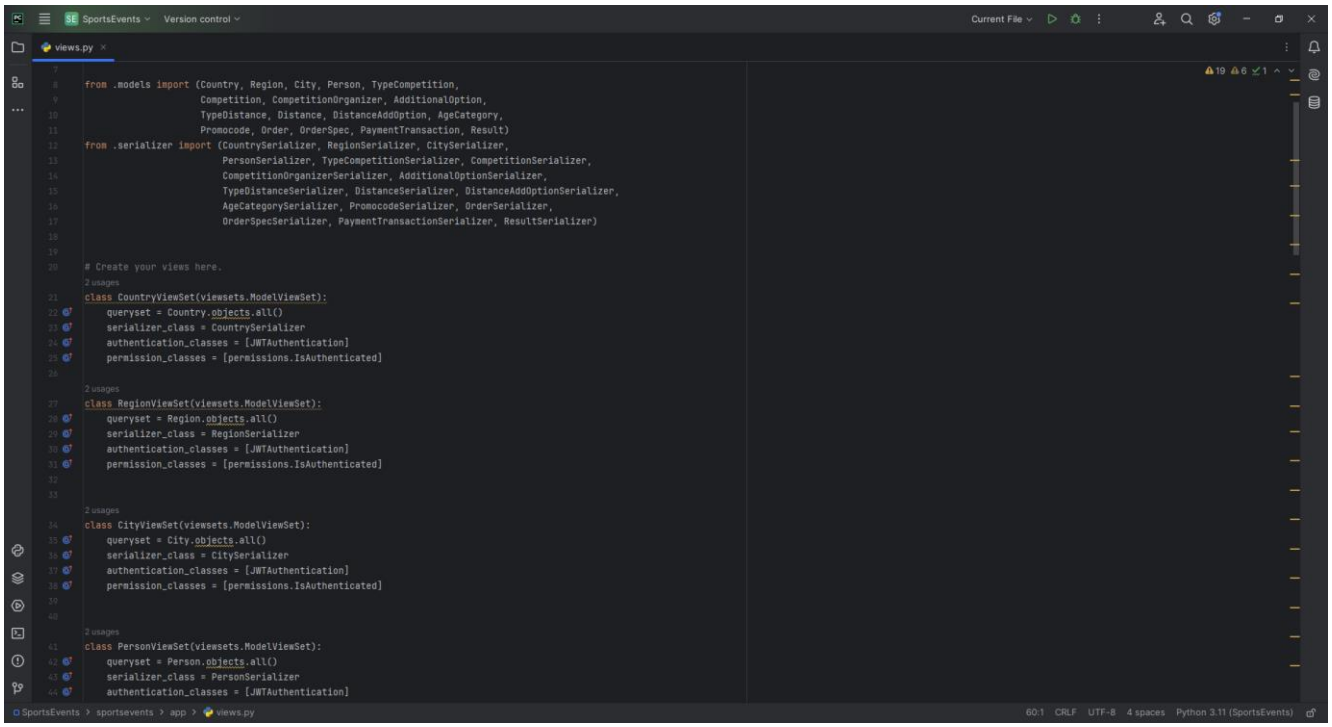
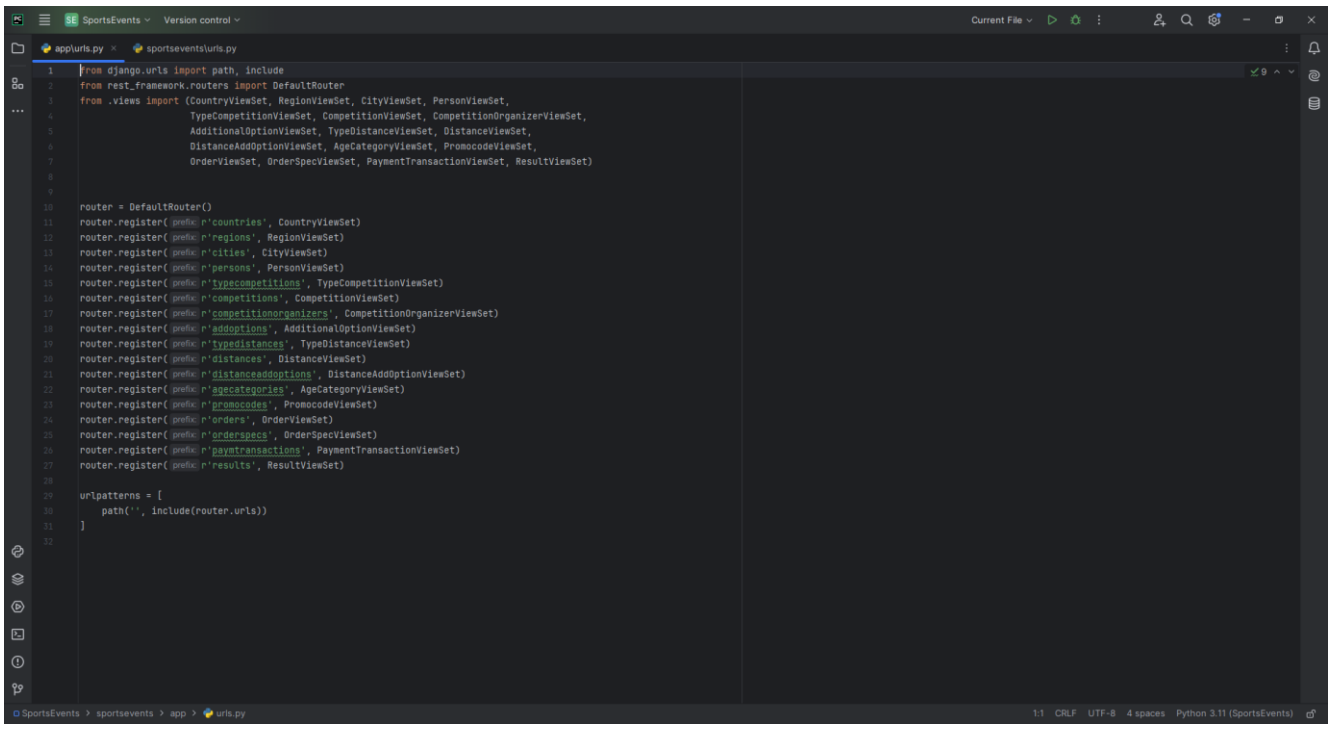


Рис. 3.15 Приклад файлу views.py

Наступним кроком потрібно налаштувати маршрутизацію. В каталозі додатку створимо файл urls.py. Оскільки ми групували загальну поведінку в набори «видів» - класи ViewSets, то є можливість автоматично генерувати URL конфігурацію. Для цього зареєструємо набори «видів» у класі маршрутизатора. Приклад коду наведено на рис. 3.16.

Наступним кроком потрібно підключити цей urls.py у глобальний файл маршрутів проєкту. Це наведено на рис. 3.17.

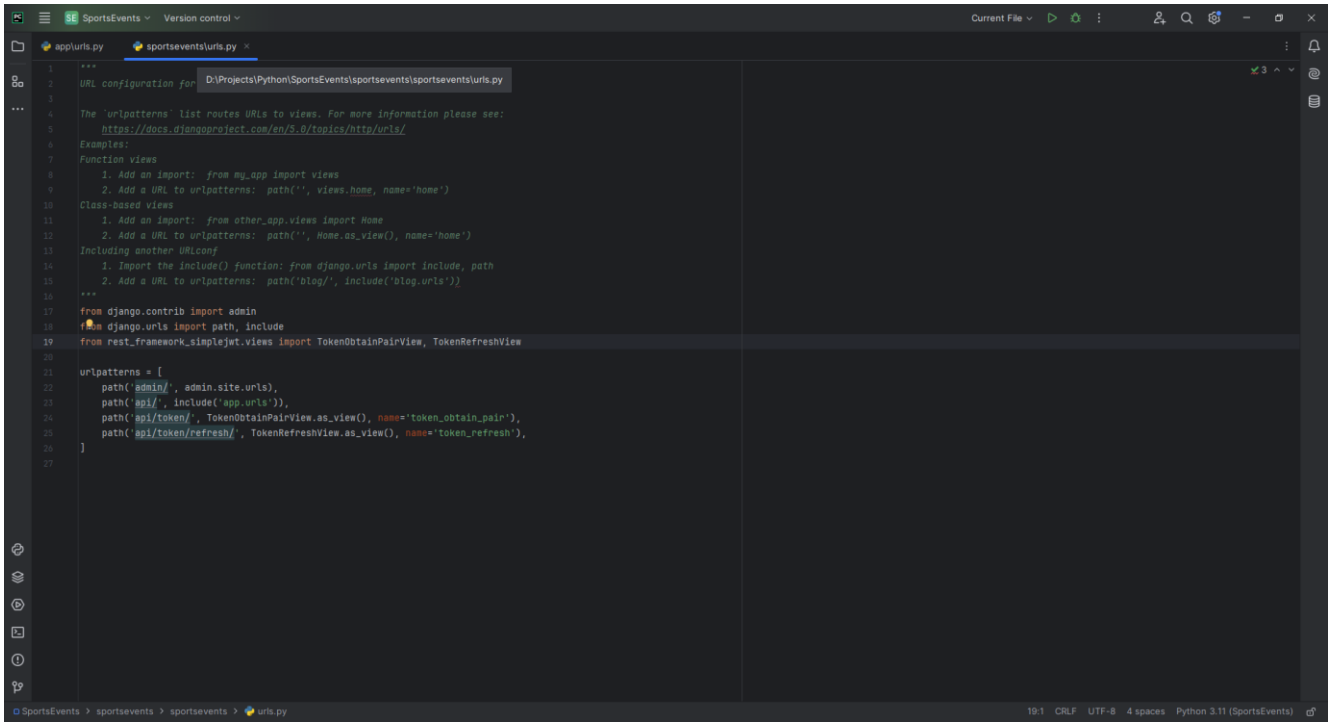


```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import (CountryViewSet, RegionViewSet, CityViewSet, PersonViewSet,
                    TypeCompetitionViewSet, CompetitionViewSet, CompetitionOrganizerViewSet,
                    AdditionalOptionViewSet, TypeDistanceViewSet, DistanceViewSet,
                    DistanceAddOptionViewSet, AgeCategoryViewSet, PromocodeViewSet,
                    OrderViewSet, OrderSpecViewSet, PaymentTransactionViewSet, ResultViewSet)


router = DefaultRouter()
router.register(prefix r'countries', CountryViewSet)
router.register(prefix r'regions', RegionViewSet)
router.register(prefix r'cities', CityViewSet)
router.register(prefix r'persons', PersonViewSet)
router.register(prefix r'typecompetitions', TypeCompetitionViewSet)
router.register(prefix r'competitions', CompetitionViewSet)
router.register(prefix r'competitionorganizers', CompetitionOrganizerViewSet)
router.register(prefix r'addoptions', AdditionalOptionViewSet)
router.register(prefix r'typedistances', TypeDistanceViewSet)
router.register(prefix r'distances', DistanceViewSet)
router.register(prefix r'distanceaddoptions', DistanceAddOptionViewSet)
router.register(prefix r'agecategories', AgeCategoryViewSet)
router.register(prefix r'promocodes', PromocodeViewSet)
router.register(prefix r'orders', OrderViewSet)
router.register(prefix r'orderspecs', OrderSpecViewSet)
router.register(prefix r'paymtransactions', PaymentTransactionViewSet)
router.register(prefix r'results', ResultViewSet)

urlpatterns = [
    path('', include(router.urls))
]
```

Рис. 3.16 Приклад коду urls.py додатку



```python
"""
URL configuration for D:\Projects\Python\SportsEvents\sportsevents\sportsevents\urls.py

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('app.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

Рис. 3.17 Підключення маршрутів додатку у глобальний urls.py

Нарешті, налаштуємо пагінацію. Пагінація дає можливість контролювати кількість об’єктів, які повертаються у відповіді на запит і будуть розміщені на сторінці. Для налаштування пагінації додаємо у секцію REST_FRAMEWORK файлу settings.py параметри DEFAULT_PAGINATION_CLASS та PAGE_SIZE. Також потрібно додати налаштування дозволів, правил валідації паролів та

параметрів JWT токенів у секції AUTH_PASSWORD_VALIDATORS, REST_FRAMEWORK та SIMPLE_JWT.

Запустимо вбудований web-сервер Django командою
python manage.py runserver

На рис. 3.18 результат виконання команди. Сервер успішно запустився. Веб-сервіс доступний за адресою http://127.0.0.1:8000.

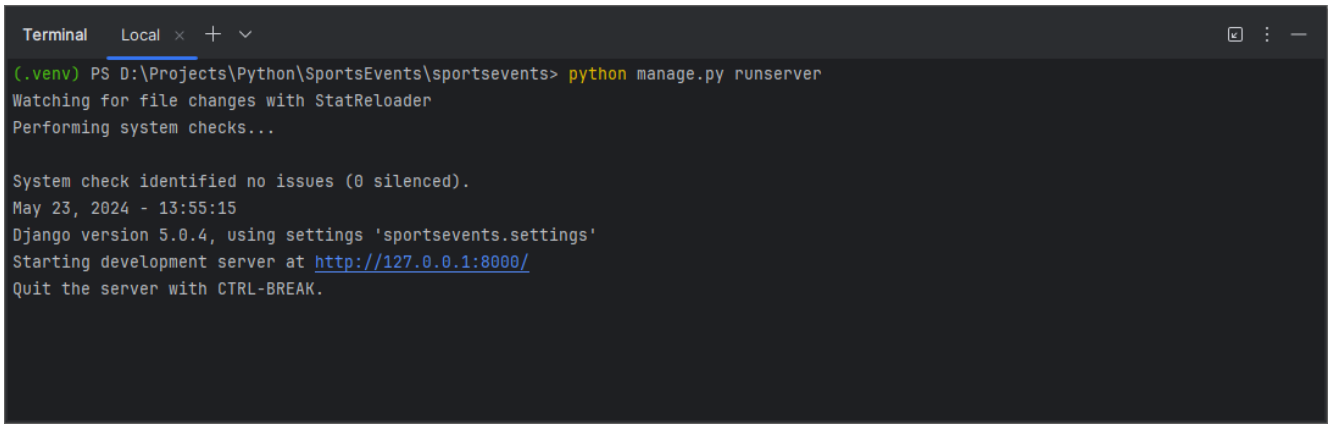


Рис. 3.18 Запуск вбудованого web-сервера Django

REST API доступно за адресою http://127.0.0.1:8000/api/ - перевіримо його доступність. На рис. 3.19 результат переходу по вказаному URL.

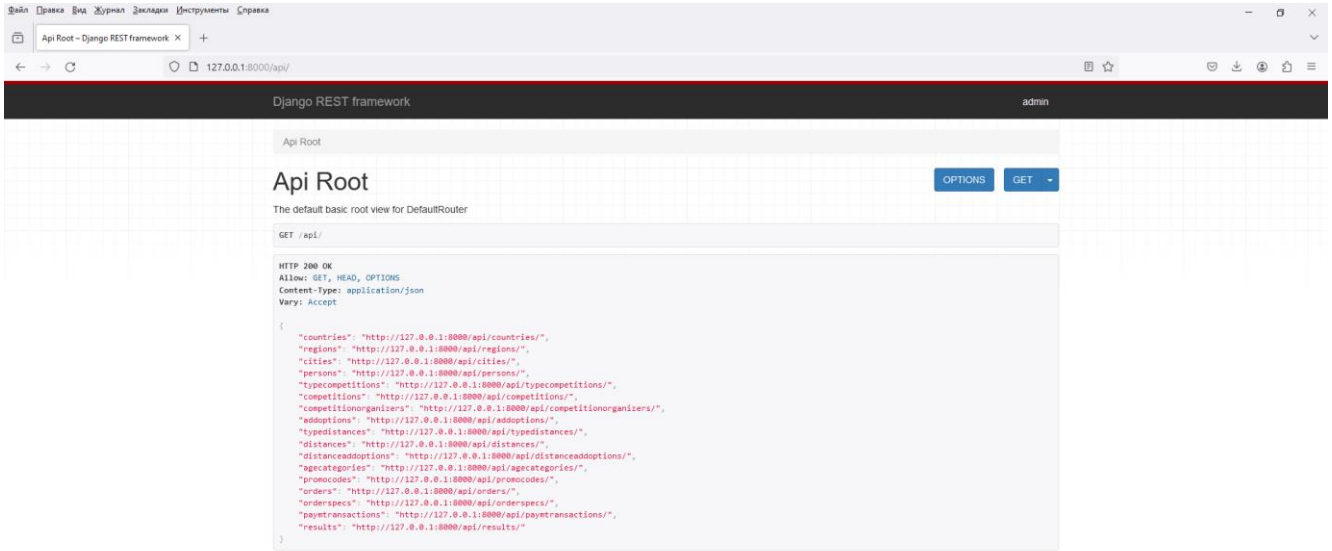


Рис. 3.19 API web-сервісу

Перевіримо доступність ендпоінту, наприклад спортивних змагань. Перейдемо за адресою http://127.0.0.1:8000/api/competitions, натиснувши на

відповідне посилання. Результат відображено на рис. 3.20. Отримано відповідь із списком спортивних змагань у форматі JSON.

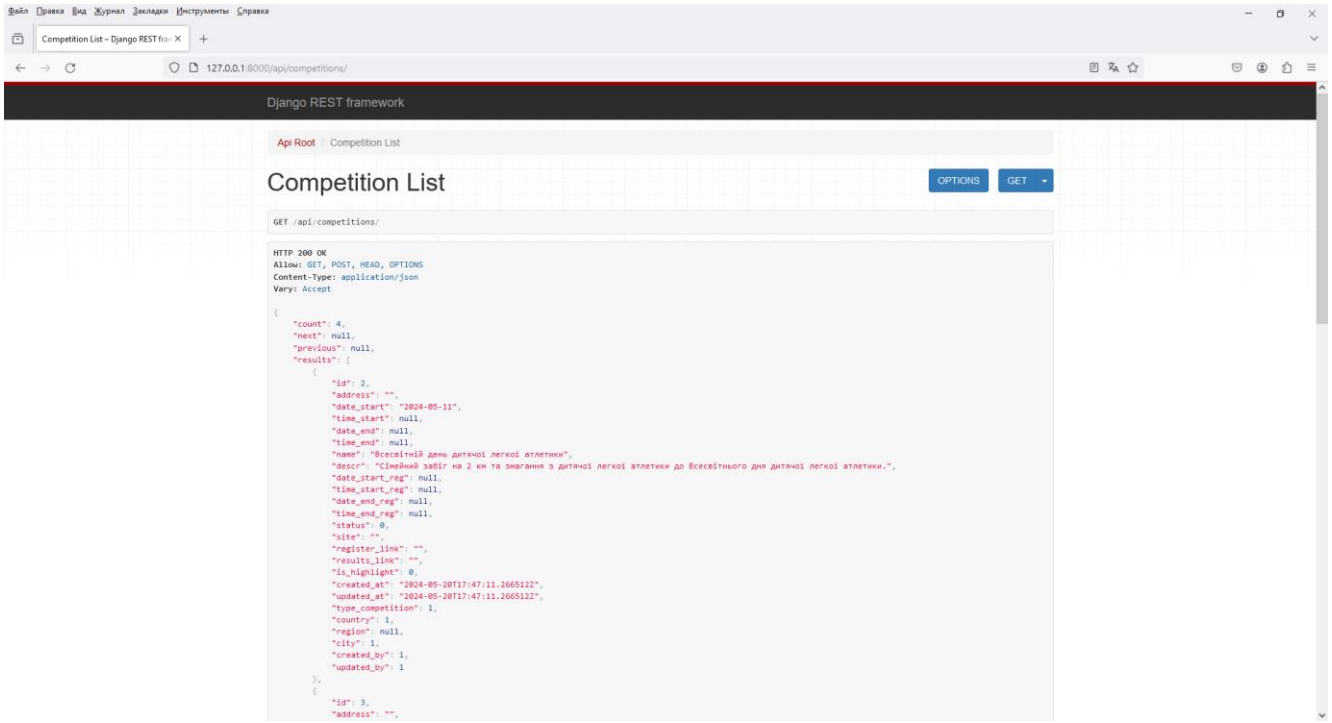


Рис.3.20 Список спортивних змагань

4 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Тестування програмного забезпечення – це процес перевірки коректності функціонування програмного забезпечення, відповідності його роботи поставленим вимогам. Це важливий і невід'ємний етап розробки ПЗ, який допомагає знайти і виправити помилки до випуску ПЗ на ринок.

Щоб впевнитися, що API працює правильно, проведемо його тестування. В якості інструменту будемо використовувати Postman. Так як з web-сервісом будуть взаємодіяти як авторизовані, так і не авторизовані користувачі, в першу чергу перевіримо отримання JWT токену.

Спочатку спробуємо отримати токен для користувача. Створюємо POST запит до ресурсу /api/token. У вкладці Body встановлюємо тип тіла запиту як JSON та вказуємо username і password. Пароль введемо невірний і перевірим отримання токену для такого випадку. На рис. 4.1 відображено результат запиту на отримання токену з введеним невірним паролем.

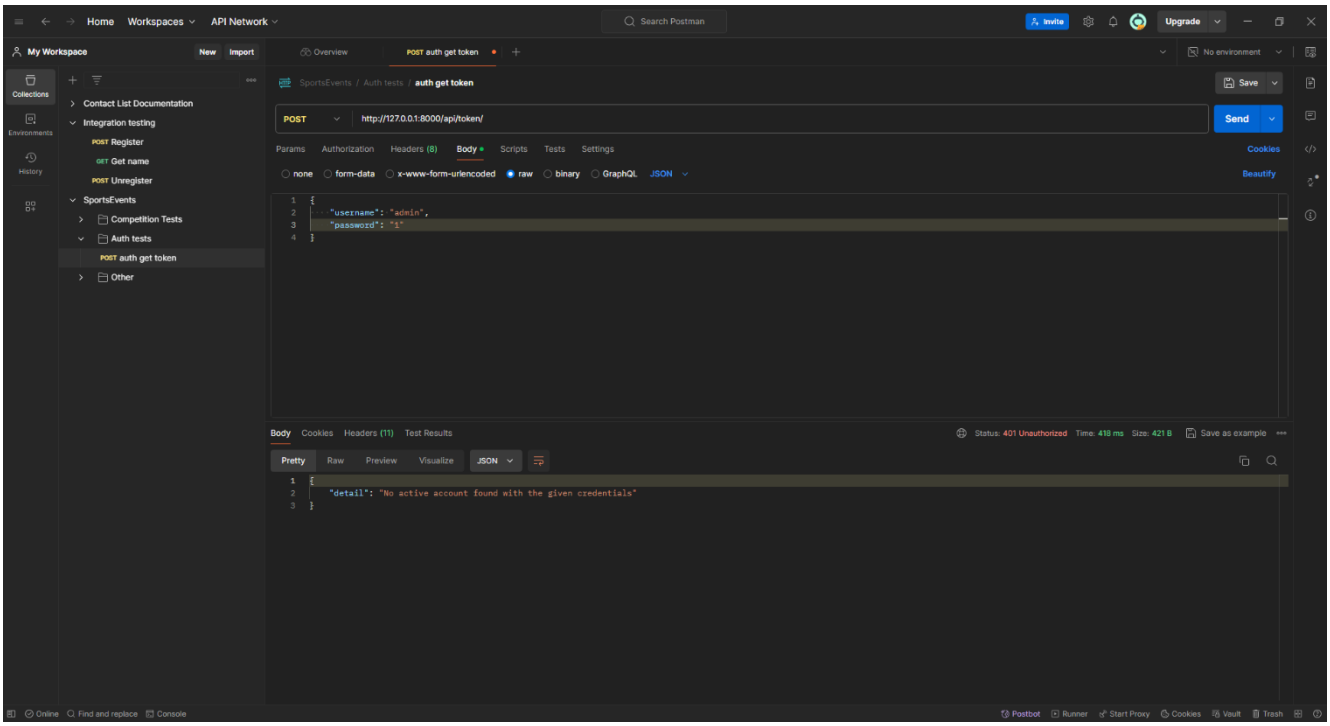


Рис. 4.1 Результат запиту на отримання токену з невірними вхідними параметрами

В результаті виконання такого запиту від API отримуємо відповідь зі HTTP статусом 401 та повідомленням, що користувача з таким логіном та паролем не знайдено. Тест пройдено.

Тепер відправимо запит з вірними логіном та паролем. На рис. 4.2 показано результати виконання такого запиту.

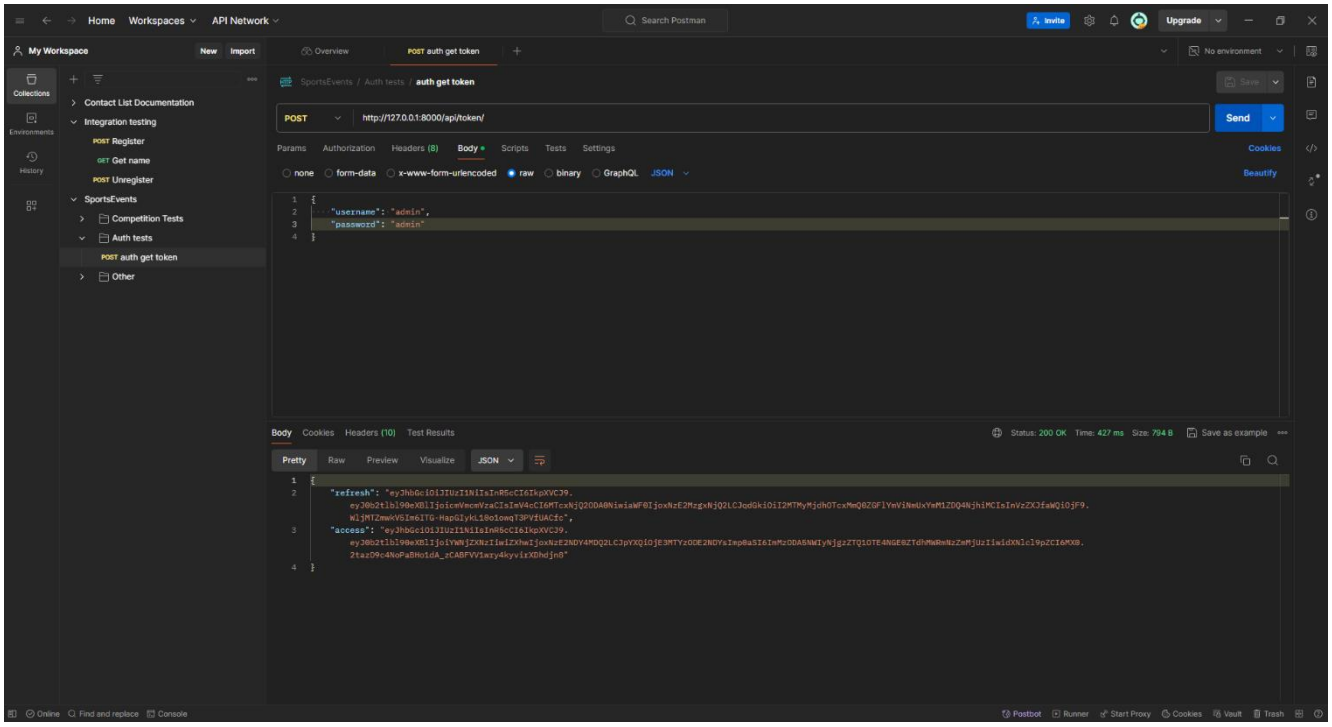


Рис. 4.2 Результат запиту на отримання токену з вірними вхідними параметрами

В результаті виконання запиту отримуємо відповідь зі статусом 200 OK та в тілі відповіді отримуємо JWT токен.

Тепер на прикладі сутності «Спортивне змагання» протестуємо CRUD операції. Наведемо перелік запитів до API для тестування:

- GET /competitions – отримати всі змагання;
- GET /competitions/{id} – отримати змагання по ID;
- POST /competitions – створити змагання;
- PUT /competitions/{id} – оновити змагання;
- DELETE /competitions/{id} – видалити змагання.

Неавторизовані користувачі та авторизовані учасники мають право тільки переглядати спортивні змагання. Організатори та адміністратори можуть також створювати, редагувати та видаляти змагання.

Створимо GET запит до ресурсу /api/competitions на отримання списку змагань неавторизованим користувачем. На рис. 4.3 представлений результат виконання запиту. У відповіді на запит сервер повернув JSON з усіма змаганнями. Тест пройдено.

Протестуємо отримання спортивного змагання по ID неавторизованим користувачем. Створимо GET запит до ресурсу /api/competitions/3. Результат відображено на рис. 4.4. В результаті отримано JSON з змаганням з ID = 3.

Наступний тест – створення змагання неавторизованим користувачем. Виконаємо POST запит до ресурсу /api/competitions в тілі якого передаємо JSON з даними змагання. На рис. 4.5 показаний результат виконання запиту. Отримано

відповідь із статусом 401 і повідомленням “Authentication credentials were not provided”. Тест пройдено.

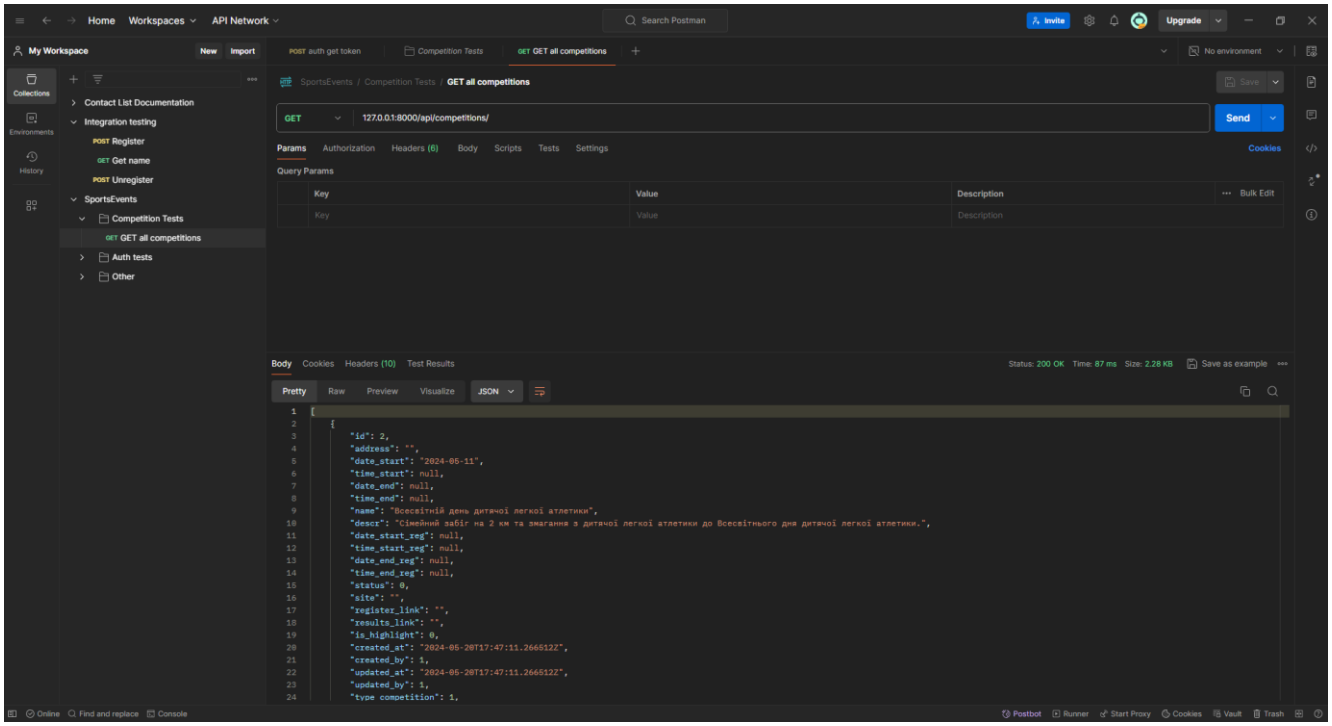


Рис. 4.3 Результат виконання GET запиту на отримання всіх спортивних змагань неавторизованим користувачем

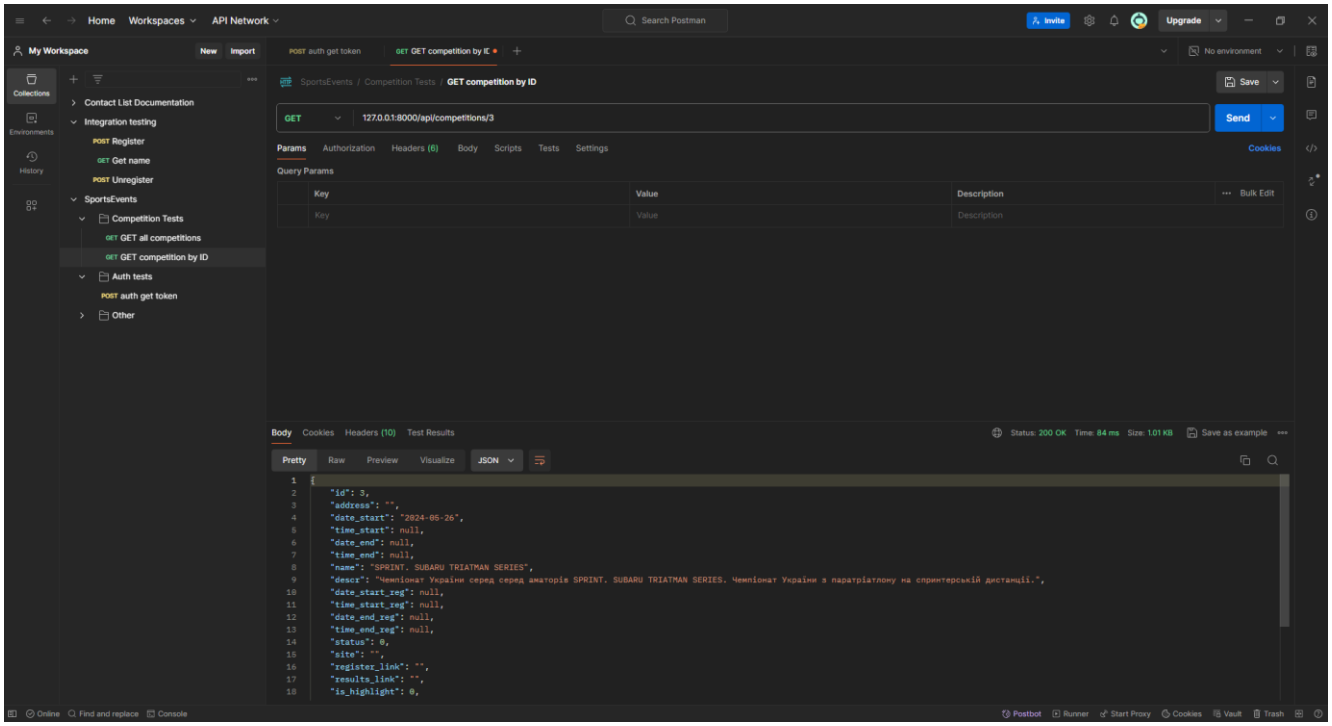


Рис. 4.4 Результат отримання спортивного змагання по ID неавторизованим користувачем

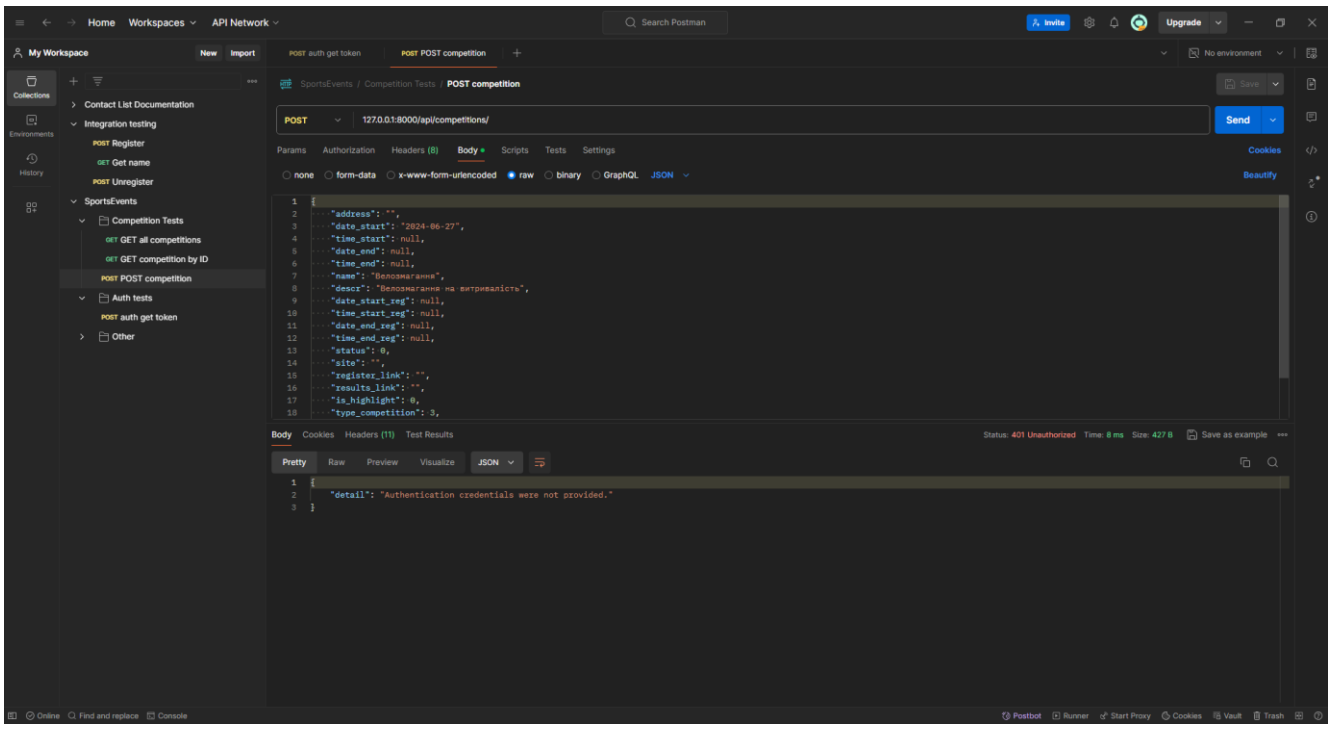


Рис. 4.5. Результат виконання запиту на створення змагання неавторизованим користувачем

Протестуємо створення спортивного змагання авторизованим користувачем із правами на створення змагання. В попередній запит на закладці «Authorization» прописуємо раніше отриманий токен. Результат виконання запиту на рис. 4.6. Отримано відповідь із статусом 201 Created і з JSON із даними по створеному змаганню в тілі відповіді. Тест пройдено.

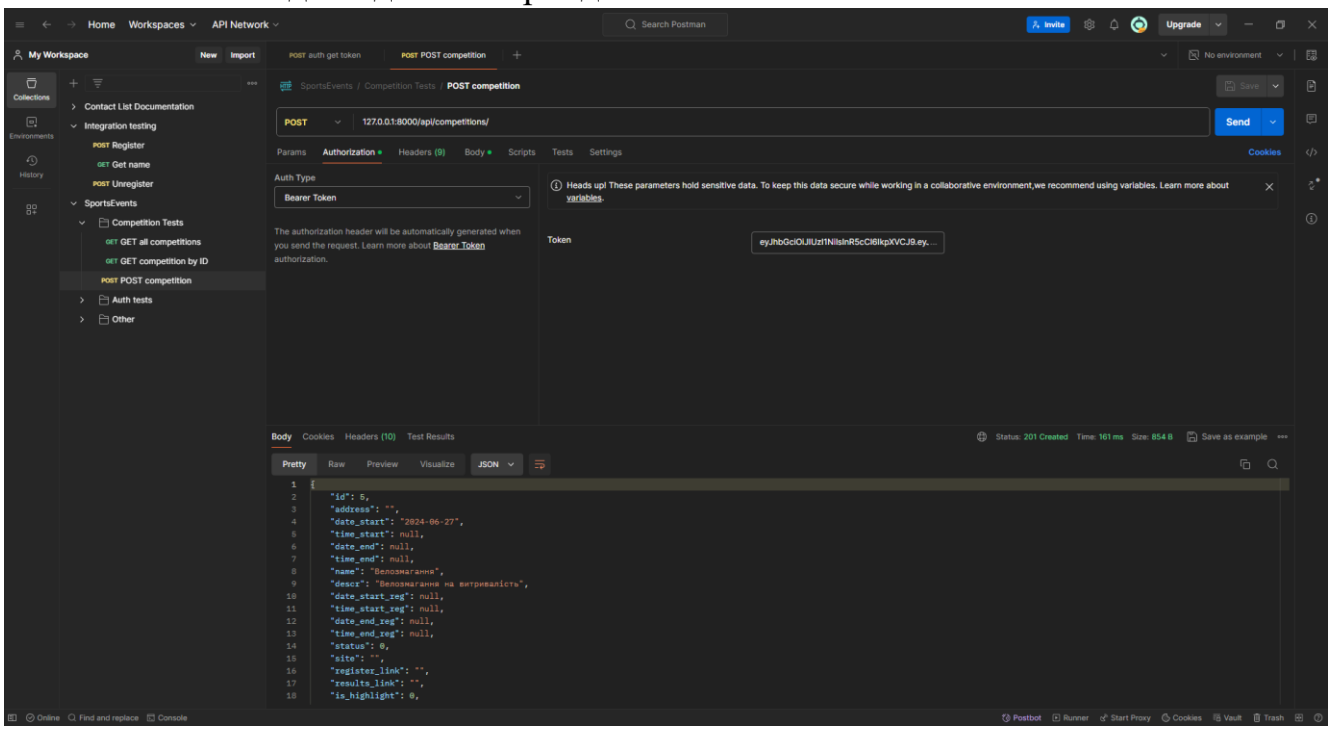


Рис. 4.6. Результат виконання запиту на створення змагання авторизованим користувачем

Протестуємо оновлення змагання. Потрібно створити PUT запит до ресурсу /api/competitions/5. Результат виконання запиту на рис. 4.7. Отримано відповідь із статусом 200 і з JSON з оновленими даними в тілі запиту. Тест пройдено.

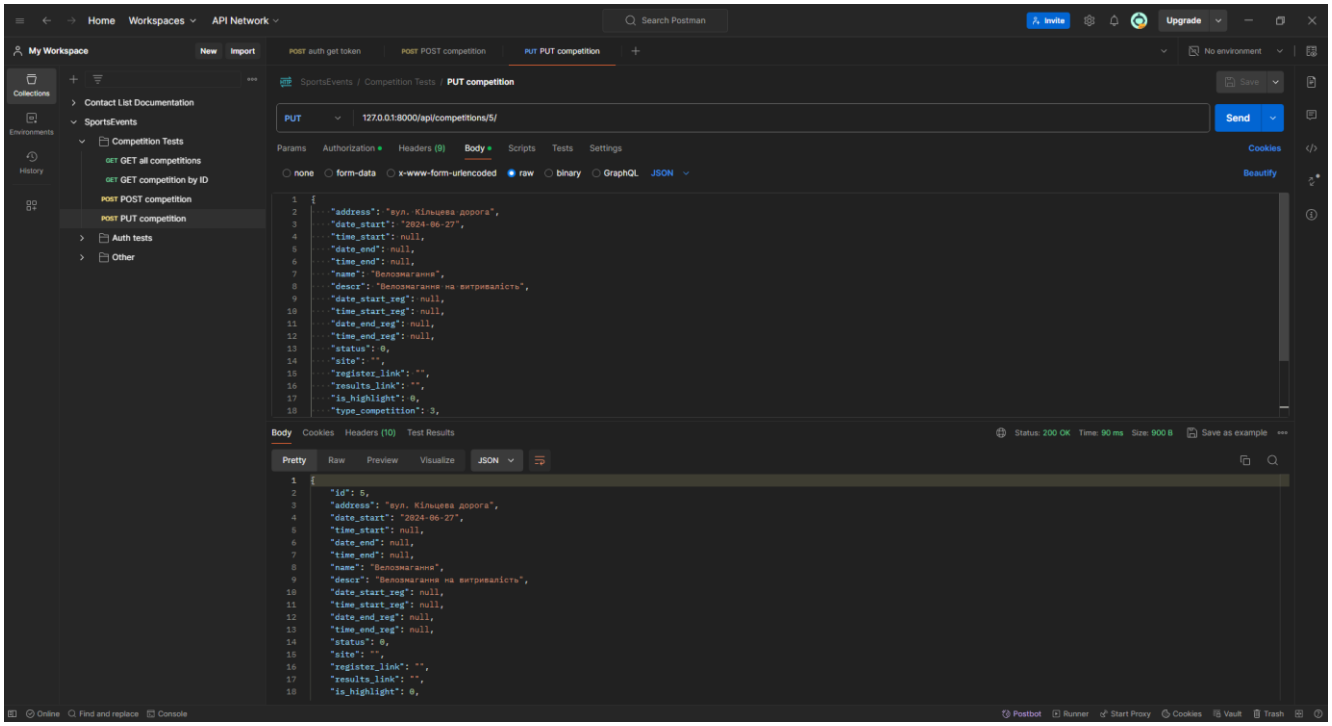


Рис. 4.7 Результат виконання запиту на оновлення спортивного змагання

Протестуємо видалення змагання. Створимо DELETE запит до ресурсу /api/competitions/5. Результат виконання запиту на рис. 4.8.

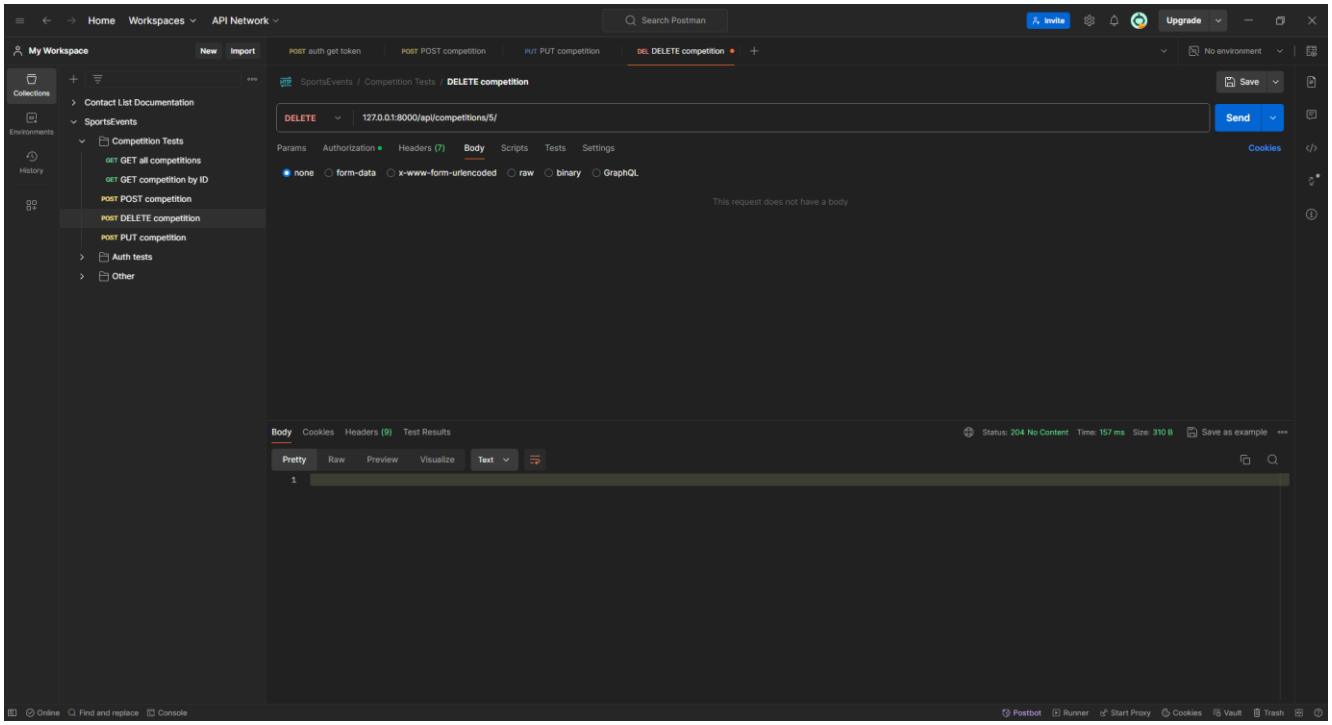


Рис. 4.8 Результат виконання запиту на видалення спортивного змагання

Отримано відповідь із статусом 204 No content. Перевіримо також виконання запиту GET запитом до ресурсу /api/competitions/5. Результат виконання запиту показано на рис. 4.9. Отримано відповідь із статусом 404 Not found. Тест пройдено.

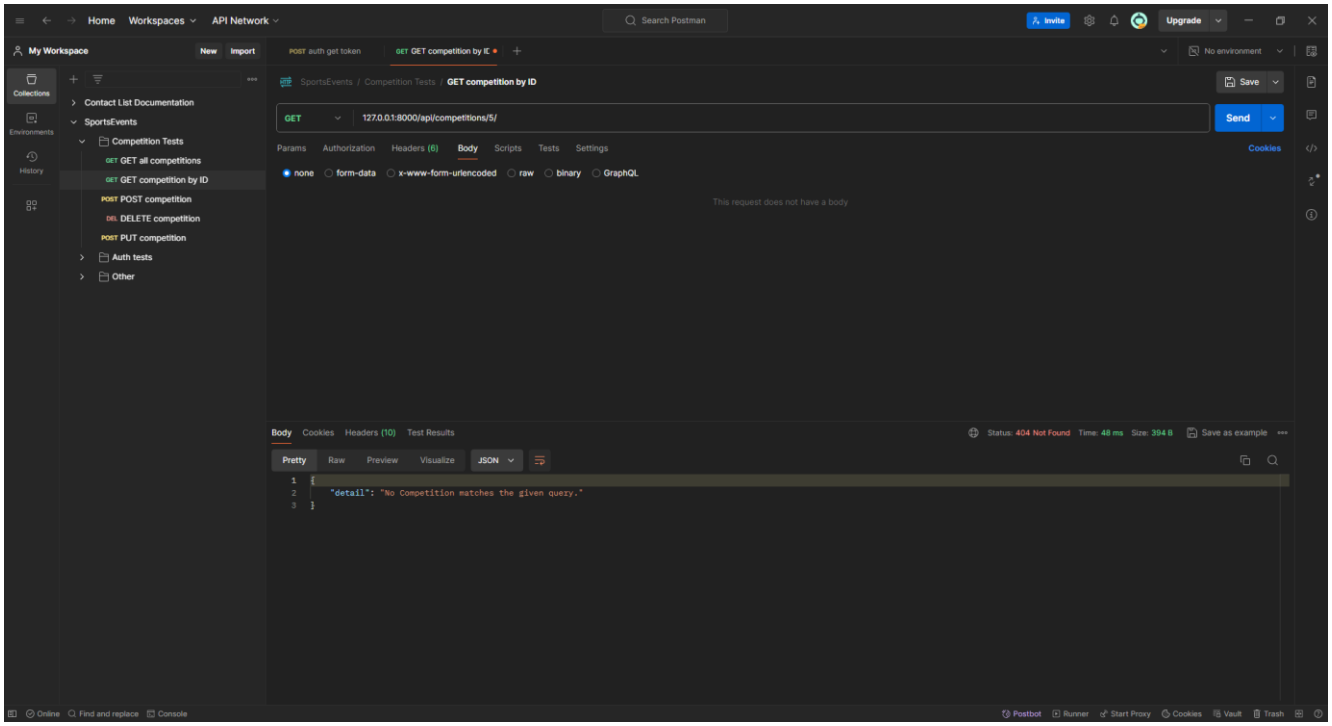


Рис. 4.9 Результат виконання GET запиту з ID видаленого змагання

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено web-сервіс «Календар спортивних змагань» для онлайн-реєстрації на змагання мовою Python.

В ході виконання поставленої мети було:

1. Проведено аналіз web-сервісів для онлайн-реєстрації на змагання. Проаналізовано три існуючих web-сервісів для онлайн-реєстрації на змагання: Timing Events, Ukraine Sport Events та ВсіПробіги, виділено їх основні переваги та недоліки.
2. На основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги до web-сервісу.
3. Проаналізовано існуючі засоби розробки web-сервісів та обрано наступні: фреймворки Django, Django Rest Framework та Vue.js, база даних PostgreSQL, середовища розробки PyCharm та WebStorm, Git та GitHub для контролю версій.
4. Спроектовано клієнт-серверну архітектуру web-сервісу, що ділить застосунок на три частини: серверну, клієнтську та базу даних. Архітектура web-сервісу була спроектована з урахуванням всіх функціональних та нефункціональних вимог.
5. Розроблено web-сервіс для онлайн-реєстрації на змагання із застосуванням патерну Model-View-Template для розробки серверної частини, модульного підходу для розробки клієнтської частини та REST API для зв’язку між серверною та клієнтською частинами.
6. Проведено тестування web-сервісу та перевірено виконання функціональних та нефункціональних вимог.

Робота пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в ІКТ». За результатами апробації опубліковано тези доповідей «Розробка веб-сервісу «Календар спортивних змагань» з онлайн-реєстрацією на змагання з використанням мови Python» та «Переваги використання хмарних сервісів Amazon Web Services при розробці веб-сервісу «Календар спортивних змагань»».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Організація масових спортивних заходів і показових виступів: навч. посіб./ укладачі В.А. Товт, Н.В. Семаль, Е.М. Сивохоп – Ужгород: ТОВ «РІК-У», 2023. - 142 с. https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/49859/1/%D0%9E%D0%9C%D0%A1%D0%97%20%D0%9F%D0%92%20-%2012%20%D1%81..pdf
2. Хіменес Х.Р. Змагання і змагальна діяльність в спорті. Лекція з навчальної дисципліни «Теорія і методика підготовки кваліфікованих спортсменів». Львівський державний університет фізичної культури. 2015 р. [Електронний ресурс] – Режим доступу до ресурсу: https://repository.ldufk.edu.ua/bitstream/34606048/3762/1/%D0%A2%D0%B5%D0%BC%D0%B0%20%E2%84%965%20%D0%97%D0%9C%D0%90%D0%93%D0%90%D0%9D%D0%9D%D0%AF%20%D0%86%20%D0%97%D0%9C%D0%90%D0%93%D0%90%D0%9B%D0%AC%D0%9D%D0%90%20%D0%94%D0%86%D0%AF%D0%9B%D0%AC%D0%9D%D0%86%D0%A1%D0%A2%D0%AC%20%D0%92%20%D0%A1%D0%9F%D0%9E%D0%A0%D0%A2%D0%86.pdf
3. Timing Events [Електронний ресурс] – Режим доступу до ресурсу: https://timingevents.com
4. TicketMe [Електронний ресурс] – Режим доступу до ресурсу: https://e.ticketme.org
5. Ukraine Sport Events [Електронний ресурс] – Режим доступу до ресурсу: https://sportevent.com.ua
6. ВсіПробіги [Електронний ресурс] – Режим доступу до ресурсу: https://vsiprobihy.org
7. TIOBE Index [Електронний ресурс] - Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/
8. Офіційний web-сайт Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org
9. The Python Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/tutorial/index.html
10. Lutz M. Learning Python, Fifth Edition / Mark Lutz. – Sebactopol, CA 95472: O'Reilly Media, Inc, 2013.
11. Офіційний web-сайт Django [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com
12. Офіційний web-сайт Django Rest Framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.django-rest-framework.org/
13. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org
14. Bootstrap Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com
15. VueJS Documentation [Електронний ресурс] – Режим доступу: https://vuejs.org

16. JetBrains official site [Електронний ресурс] – Режим доступу: https://www.jetbrains.com
17. Fielding R. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] / Roy Fielding // University of California, Irvine. – 2000. – Режим доступу до ресурсу: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm
18. Wilkes M. Advanced Python Development: Using Powerful Language Features in Real-World Applications / Matthew Wilkes. – Berkeley, CA: Apress Berkeley, CA, 2020. – 605 с.
19. Amazon Web Services Documentation [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/
20. CSS Reference [Електронний ресурс] – Режим доступу: https://cssreference.io
21. GitHub Documentation [Електронний ресурс] – Режим доступу: https://docs.github.com
22. HTML Reference [Електронний ресурс] – Режим доступу: https://htmlreference.io
23. JavaScript Documentation [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
24. Flanagan D. JavaScript: The Definitive Guide, 7th Edition / David Flanagan. – Sebastopol, CA 95472: O\'Reilly Media, Inc., 2020. – 706 с.
25. Frain B. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques / Ben Frain. – Birmingham, UK: Packt Publishing, 2020. – 408 с.

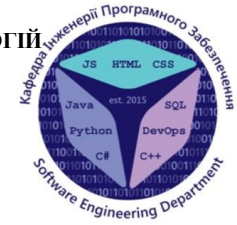
ДОДАТОК А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



РОЗРОБКА WEB-СЕРВІСУ «КАЛЕНДАР СПОРТИВНИХ ЗМАГАНЬ» ДЛЯ ОНЛАЙН -РЕЄСТРАЦІЇ НА ЗМАГАННЯ МОВОЮ PYTHON

Виконала студентка 4 курсу
групи ПД-41
Петрусь Лілія Андріївна

Керівник роботи
К.т.н., доц., доцент кафедри ІПЗ Негоденко Олена Василівна

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу адміністрування спортивних змагань за допомогою веб-сервісу, створеного мовою Python.

Об’єкт дослідження – процес адміністрування спортивних змагань.

Предмет дослідження – веб-сервіс для адміністрування спортивних змагань.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі аналоги web-сервісів для онлайн-реєстрації на змагання, визначити переваги та недоліки.
2. Розробити функціональні та нефункціональні вимоги до web-сервісу для онлайн-реєстрації на змагання.
3. Дослідити та вибрати засоби розробки web-сервісу, розробити модель архітектури web-сервісу та його дизайн.
4. Розробити web-сервіс на основі обраних інструментів та визначених вимог.
5. Провести тестування web-сервісу для онлайн-реєстрації на змагання.

АНАЛІЗ АНАЛОГІВ

Критерії порівняння	Timing Events	Ukraine Sport Events	ВсіПробіги	Sports Events
Реєстрація на всі види змагань	+	+	-	+
Результат змагання учаснику на email або смс	+	+	-	+
Можливість підключення системи електронного хронометражу	+	+	-	+
Повний доступ для керування налаштуваннями реєстрації	тільки для партнерів	інформація не доступна	для всіх організаторів	для всіх організаторів
Редагування/анулювання учасником своєї реєстрації	-	-	-	+
Можливість передати свою реєстрацію іншому учаснику	-	-	-	+
Самостійний вибір стартового номеру учасником	-	-	-	+
Зручність реєстрації на веб-сервісі	Аккаунт створюється після заповнення форми реєстрації на спортивний захід	Можливість реєстрації та входу за допомогою соцмереж	Класична реєстрація та вхід за допомогою email та паролю	Можливість реєстрації та входу за допомогою соцмереж або аккаунту Google
Використання з мобільних пристроїв	+	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні

1.Організатор:
- повинен мати можливість створювати, редагувати та видаляти спортивне змагання;
- повинен мати можливість створення аккаунту учасника, редагування даних учасника;
- може зареєструвати учасника на змагання, редагувати дані реєстрацій учасників та видаляти реєстрації на змагання.

2. Учасник:
- може переглядати список спортивних змагань, детальну інформацію по змаганню та результати спортивного змагання;
- може реєструватись на спортивні змагання та здійснювати їх онлайн-оплату;
- повинен мати можливість анулювати свою реєстрацію або передати свою реєстрацію іншому учаснику.

3. Веб-сервіс:
- повинен забезпечувати реєстрацію на всі види спортивних змагань;
- надає можливість реєстрації та входу на сервіс як за допомогою логіну та паролю, так і за допомогою аккаунту Google або соцмереж.

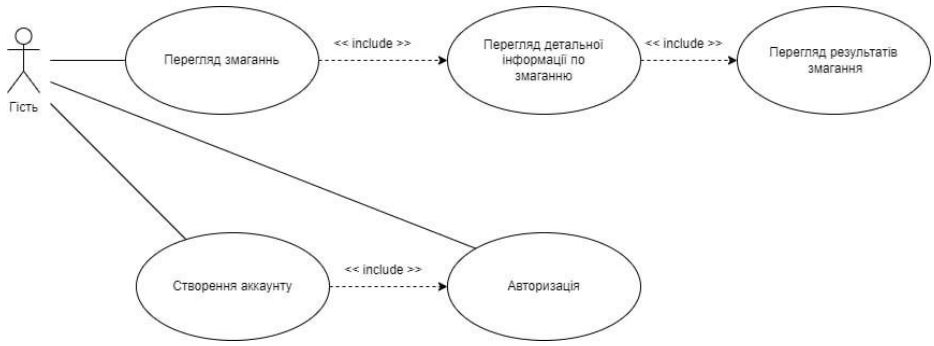
Нефункціональні

1. Web-сервіс має бути доступний і повністю працездатний 24/7.
2. Час завантаження сторінки не повинен перевищувати 2 секунди.
3. У системі має бути присутня авторизація користувачів, обмеження доступу згідно ролей користувачів.
4. Система має надавати гарантію безпеки та конфіденційності даних користувачів, вся інформація користувача має використовуватися лише за призначенням.
5. Система повинна мати інтуїтивно зрозумілий та логічний інтерфейс.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

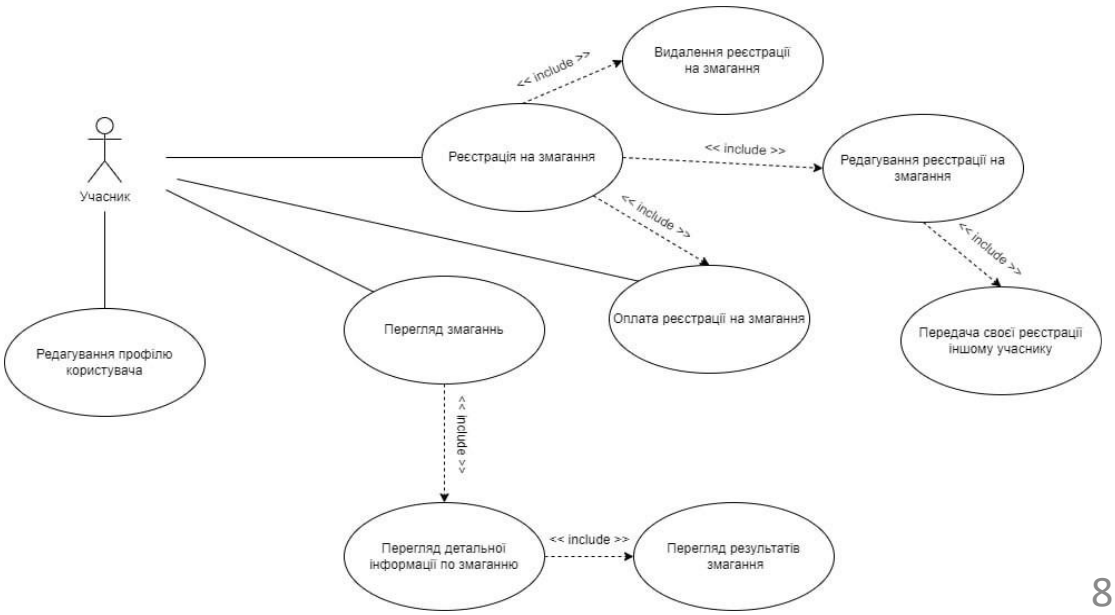
6

Діаграма прецедентів актора «Гість»



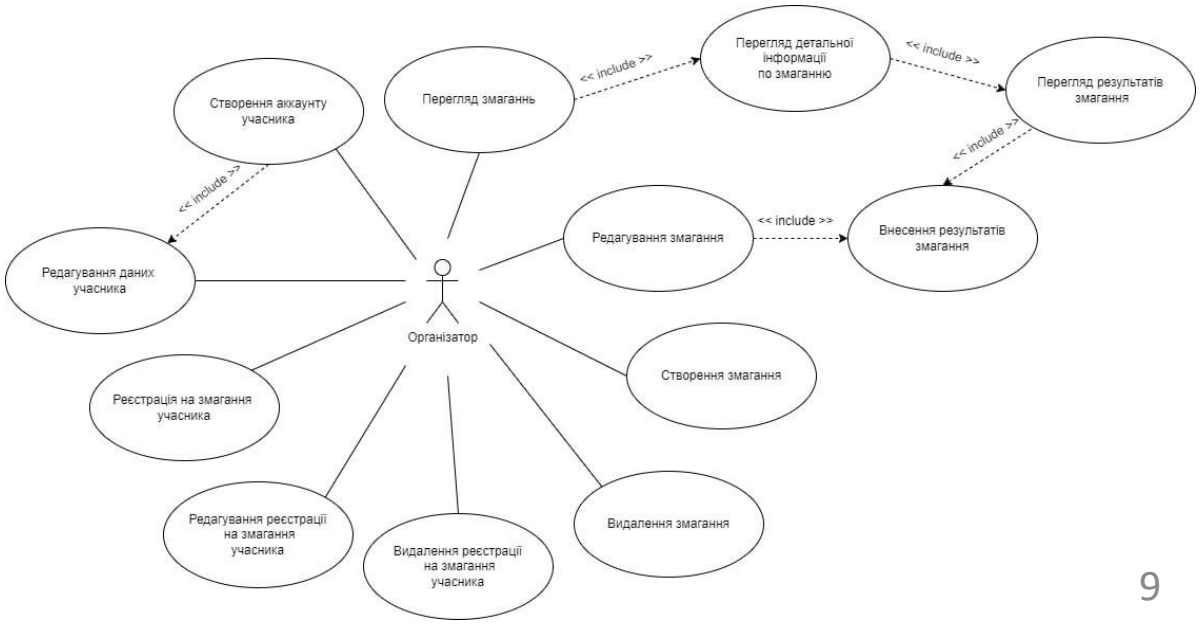
7

Діаграма прецедентів актора «Учасник»



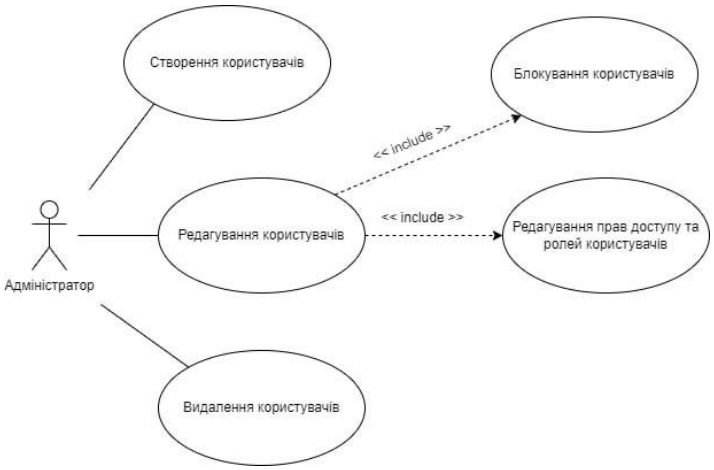
8

Діаграма прецедентів актора «Організатор»



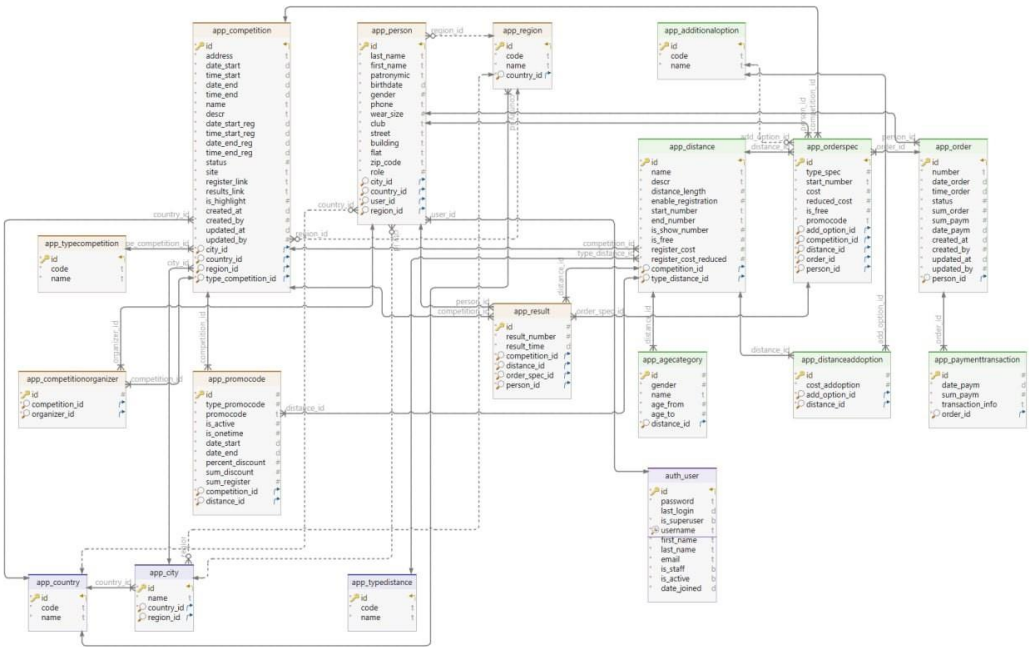
9

Діаграма прецедентів актора «Адміністратор»



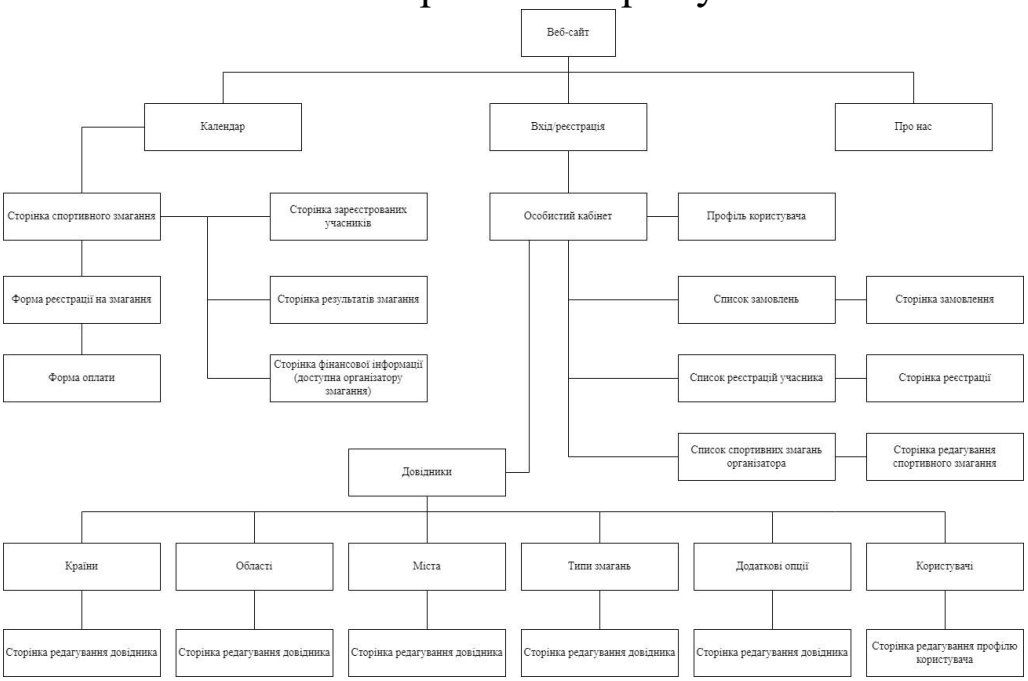
10

Схема бази даних

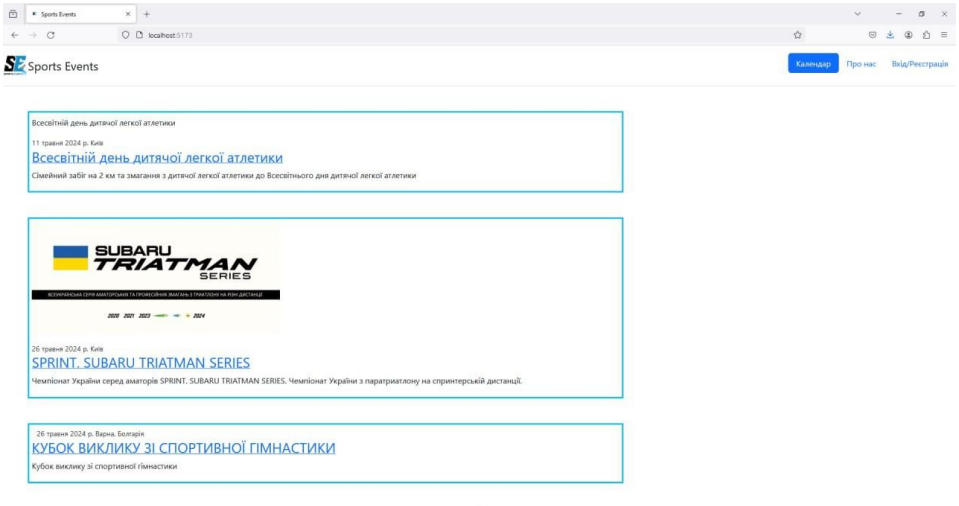


11

Карта web-сервісу

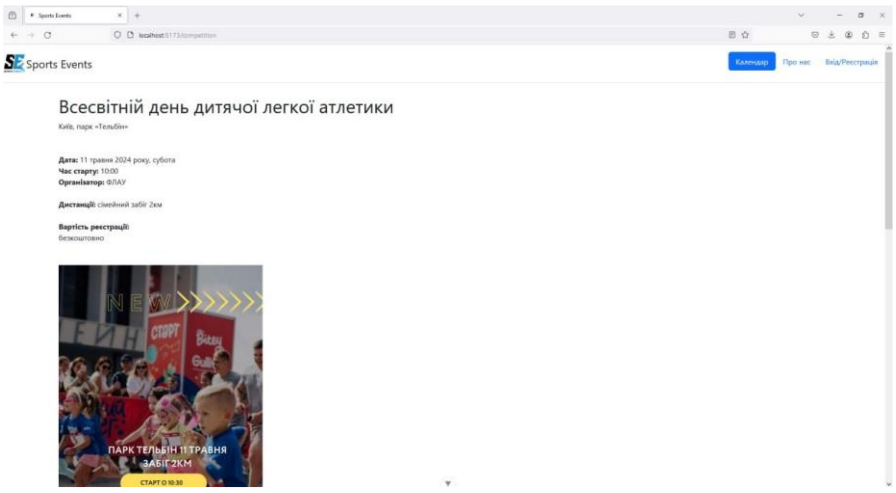


ЕКРАННІ ФОРМИ



Сторінка з календарем змагань

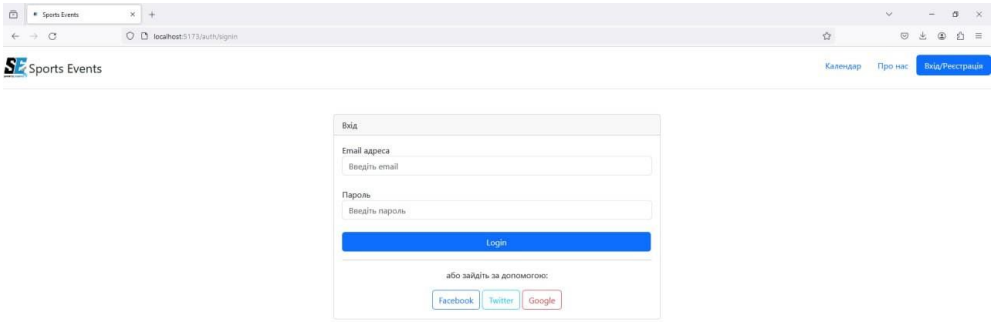
ЕКРАННІ ФОРМИ



Сторінка спортивного заходу

14

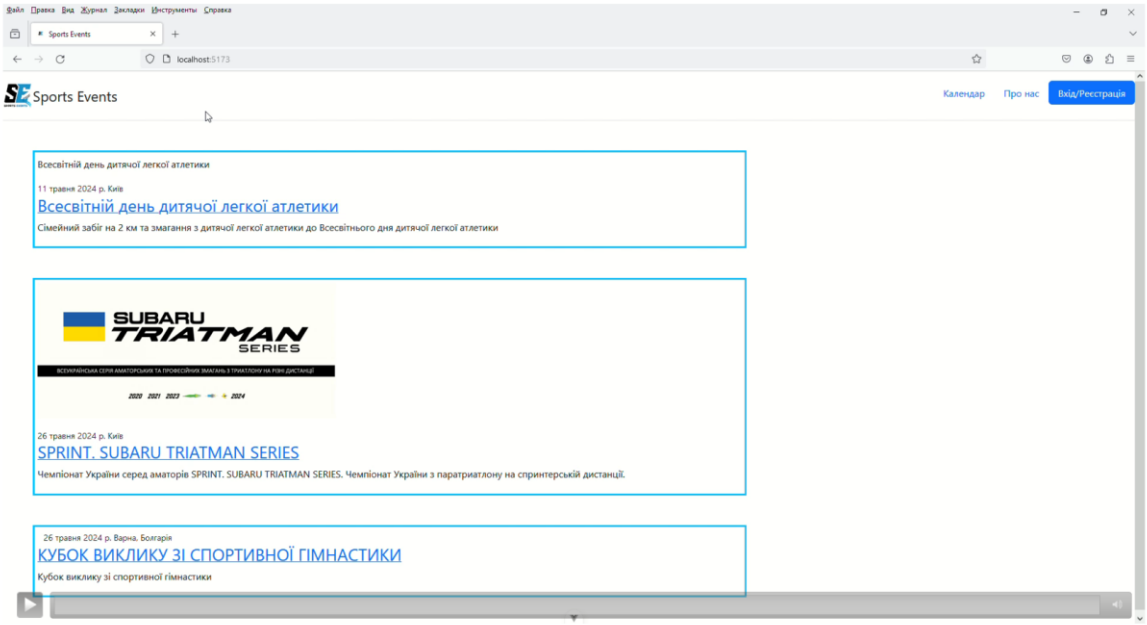
ЕКРАННІ ФОРМИ



Форма авторизації

15

ВІДЕО РОБОТИ WEB-СЕРВІСУ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Петрусь Л. А., Негоденко О. В. Розробка веб -сервісу «Календар спортивних змагань» з онлайн - реєстрацією на змагання з використанням мови Python: Матеріали опубліковано на IV Всеукраїнській науково -технічній конференції «Застосування програмного забезпечення в ІКТ» Збірник тез. 24. 04. 2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024 с. 432-433

2. Петрусь Л. А., Негоденко О. В. Переваги використання хмарних сервісів Amazon web services при розробці веб -сервісу «Календар спортивних змагань» : Матеріали опубліковано на IV Всеукраїнській науково -технічній конференції «Застосування програмного забезпечення в ІКТ» Збірник тез. 24. 04. 2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024 с.

ВИСНОВКИ

1. Проведено аналіз web-сервісів для онлайн-реєстрації на змагання. Проаналізовано три існуючих web-сервісів для онлайн-реєстрації на змагання: Timing Events, Ukraine Sport Events та ВсіПробіги, виділено їх основні переваги та недоліки.
2. На основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги до web-сервісу.
3. Проаналізовано існуючі засоби розробки web-сервісів та обрано наступні: фреймворки Django, Django Rest Framework та Vue.js, база даних PostgreSQL, середовища розробки PyCharm та WebStorm, Git та GitHub для контролю версій.
4. Спроектовано клієнт-серверну архітектуру web-сервісу, що ділить застосунок на три частини: серверну, клієнтську та базу даних. Архітектура web-сервісу була спроектована з урахуванням всіх функціональних та нефункціональних вимог.
5. Розроблено web-сервіс для онлайн-реєстрації на змагання із застосуванням патерну Model-View-Template для розробки серверної частини, модульного підходу для розробки клієнтської частини та REST API для зв'язку між серверною та клієнтською частинами.
6. Проведено тестування web-сервісу та перевірено виконання функціональних та нефункціональних вимог.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

app/admin.py

```
from django.contrib import admin
from .models import (Country, Region, City, Person, TypeCompetition,
    Competition, CompetitionOrganizer, AdditionalOption,
    TypeDistance, Distance, DistanceAddOption, AgeCategory,
    Promocode, Order, OrderSpec, PaymentTransaction, Result)


# Register your models here.
admin.site.register(Country)
admin.site.register(Region)
admin.site.register(City)
admin.site.register(Person)
admin.site.register(TypeCompetition)
admin.site.register(Competition)
admin.site.register(CompetitionOrganizer)
admin.site.register(AdditionalOption)
admin.site.register(TypeDistance)
admin.site.register(Distance)
admin.site.register(DistanceAddOption)
admin.site.register(AgeCategory)
admin.site.register(Promocode)
admin.site.register(Order)
admin.site.register(OrderSpec)
admin.site.register(PaymentTransaction)
admin.site.register(Result)
```

app/apps.py

```
from django.apps import AppConfig


class AppConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
```

```
    name = 'app'
```

app/common.py

```
class Common:
    EMPTY = 0
    MALE = 1
    FEMALE = 2
    GENDER_CHOICE = [
        (EMPTY, 'не обрано'),
        (MALE, 'чол'),
        (FEMALE, 'жін'),
    ]
    YESNO_CHOICE = [
        (0, 'Ні'),
        (1, 'Так'),
    ]
    SIZE_CHOICE = [
            (0, 'не обрано'),
            (1, 'XS'),
            (2, 'S'),
            (3, 'M'),
            (4, 'L'),
            (5, 'XL'),
            (6, 'XXL'),
            (7, 'XXXL'),
        ]
    TYPE_PROMOCODE_CHOICE = [
        (0, 'не обрано'),
        (1, 'процент знижки'),
        (2, 'сума знижки'),
        (3, 'сума'),
    ]
    STATUS_CHOICE = [
        (0, 'не обрано'),
        (1, 'новий'),
        (2, 'опублікований'),
        (3, 'реєстрацію відкрито'),
        (4, 'реєстрація завершена'),
        (5, 'відмінений'),
        (6, 'завершений'),
```

```
    ]
    ORDER_STATUS_CHOICE = [
        (0, 'не обрано'),
        (1, 'новий'),
        (2, 'сплачено'),
        (3, 'підтверджено адміністратором'),
        (4, 'відмінено'),
    ]
    TYPE_ORDER_ITEM_CHOICE = [
        (0, 'не обрано'),
        (1, 'реєстрація'),
        (2, 'додаткова опція'),
    ]
    USER_ROLE_CHOICE = [
        (0, 'не обрано'),
        (1, 'учасник'),
        (2, 'організатор'),
        (3, 'адміністратор'),
    ]
```

app/models.py

```
import django.contrib.auth
from django.db import models
from django.contrib.auth.models import User
from .common import Common


# Create your models here.
class Country(models.Model):
    code = models.CharField(max_length=5, verbose_name='Код країни')
    name = models.CharField(max_length=100, verbose_name='Назва країни')

    def __str__(self):
        return f'{self.name}'

    class Meta:
        verbose_name = 'Країна'
        verbose_name_plural = 'Країни'
```

```
class Region(models.Model):
    country = models.ForeignKey(Country, on_delete=models.CASCADE, verbose_name='Країна')
    code = models.CharField(max_length=5, verbose_name='Код області')
    name = models.CharField(max_length=100, verbose_name='Назва області')

    def __str__(self):
        return f'{self.name}'

    class Meta:
        verbose_name = 'Область'
        verbose_name_plural = 'Області'


class City(models.Model):
    country = models.ForeignKey(Country, related_name='cities', on_delete=models.CASCADE, verbose_name='Країна')
    region = models.ForeignKey(Region, on_delete=models.CASCADE, blank=True, null=True, verbose_name='Область')
    name = models.CharField(max_length=100, verbose_name='Назва міста')

    def __str__(self):
        return f'{self.name}'

    class Meta:
        verbose_name = 'Місто'
        verbose_name_plural = 'Міста'


class Person(models.Model):
    last_name = models.CharField(max_length=100, verbose_name='Прізвище')
    first_name = models.CharField(max_length=100, verbose_name='Ім\'я')
    patronymic = models.CharField(max_length=100, verbose_name='По батькові')
    birthdate = models.DateField(verbose_name='Дата народження')
    gender = models.IntegerField(choices=Common.GENDER_CHOICE, default=0, verbose_name='Стать')
    phone = models.CharField(max_length=20, verbose_name='Телефон')
    wear_size = models.IntegerField(choices=Common.SIZE_CHOICE, default=0, verbose_name='Розмір одягу')
    club = models.CharField(max_length=255, verbose_name='Клуб', blank=True)
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Користувач')
    country = models.ForeignKey(Country, on_delete=models.PROTECT, verbose_name='Країна', blank=True, null=True)
    region = models.ForeignKey(Region, on_delete=models.PROTECT, verbose_name='Область', blank=True, null=True)
    city = models.ForeignKey(City, on_delete=models.PROTECT, verbose_name='Місто', blank=True, null=True)
    street = models.CharField(max_length=255, verbose_name='Назва вулиці', blank=True)
    building = models.CharField(max_length=10, verbose_name='Номер будинку', blank=True)
    flat = models.CharField(max_length=10, verbose_name='Номер квартири', blank=True)
    zip_code = models.CharField(max_length=10, verbose_name='Поштовий індекс', blank=True)
    role = models.IntegerField(choices=Common.USER_ROLE_CHOICE, default=0, verbose_name='Роль')

    def __str__(self):
        return f'{self.last_name} {self.first_name} {self.patronymic} {self.birthdate}'

    class Meta:
        verbose_name = 'Учасник'
        verbose_name_plural = 'Учасники'


class TypeCompetition(models.Model):
    code = models.CharField(max_length=5, verbose_name='Код')
    name = models.CharField(max_length=50, verbose_name='Назва')

    def __str__(self):
        return f'{self.code} {self.name}'

    class Meta:
        verbose_name = 'Тип змагання'
        verbose_name_plural = 'Типи змаганнь'


class Competition(models.Model):
```

```
    type_competition = models.ForeignKey(TypeCompetition, on_delete=models.PROTECT, verbose_name='Тип змагання')
    country = models.ForeignKey(Country, on_delete=models.PROTECT, verbose_name='Країна')
    region = models.ForeignKey(Region, on_delete=models.PROTECT, verbose_name='Область', blank=True, null=True)
    city = models.ForeignKey(City, on_delete=models.PROTECT, verbose_name='Місто')
    address = models.CharField(max_length=255, verbose_name='Адреса змагання', blank=True)
    date_start = models.DateField(verbose_name='Дата початку')
    time_start = models.TimeField(verbose_name='Час початку', blank=True, null=True)
    date_end = models.DateField(verbose_name='Дата закінчення', blank=True, null=True)
    time_end = models.TimeField(verbose_name='Час закінчення', blank=True, null=True)
    name = models.CharField(max_length=255, verbose_name='Назва')
    descr = models.TextField(verbose_name='Детальний опис', blank=True)
    date_start_reg = models.DateField(verbose_name='Дата початку реєстрації', blank=True, null=True)
    time_start_reg = models.TimeField(verbose_name='Час початку реєстрації', blank=True, null=True)
    date_end_reg = models.DateField(verbose_name='Дата закінчення реєстрації', blank=True, null=True)
    time_end_reg = models.TimeField(verbose_name='Час закінчення реєстрації', blank=True, null=True)
    status = models.IntegerField(choices=Common.STATUS_CHOICE, default=0, verbose_name='Статус змагання')
    site = models.CharField(max_length=255, verbose_name='Сайт', blank=True)
    register_link = models.CharField(max_length=255, verbose_name='Посилання на сторінку реєстрації', blank=True)
    results_link = models.CharField(max_length=255, verbose_name='Посилання на сторінку результатів', blank=True)
    is_highlight = models.IntegerField(choices=Common.YESNO_CHOICE, default=0, verbose_name='Виділяти у списку змагань')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата створення')
    created_by = models.ForeignKey(User, related_name='created_%(class)s_set', on_delete=models.PROTECT, null=True,
                                   blank=True)
```

```
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Дата оновлення')
    updated_by = models.ForeignKey(User, related_name='updated_%(class)s_set', on_delete=models.PROTECT, null=True,
                                   blank=True)

    def __str__(self):
        return f'{self.date_start} {self.name}'

    class Meta:
        verbose_name = 'Змагання'
        verbose_name_plural = 'Змагання'


class CompetitionOrganizer(models.Model):
    competition = models.ForeignKey(Competition, on_delete=models.CASCADE)
    organizer = models.ForeignKey(Person, on_delete=models.PROTECT)

    def __str__(self):
        return f'{self.competition.name} {self.organizer.last_name} {self.organizer.first_name} {self.organizer.patronymic}'

    class Meta:
        verbose_name = 'Організатор змагання'
        verbose_name_plural = 'Організатори змаганнь'


class AdditionalOption(models.Model):
    code = models.CharField(max_length=5, verbose_name='Код')
    name = models.CharField(max_length=255, verbose_name='Назва')

    def __str__(self):
        return f'{self.code} {self.name}'

    class Meta:
        verbose_name = 'Додаткова опція'
        verbose_name_plural = 'Додаткові опції'


class TypeDistance(models.Model):
    code = models.CharField(max_length=5, verbose_name='Код')
```

```
    name = models.CharField(max_length=100, verbose_name='Назва')

    def __str__(self):
        return f'{self.code} {self.name}'

    class Meta:
        verbose_name = 'Тип дистанції'
        verbose_name_plural = 'Типи дистанцій'


class Distance(models.Model):
    competition = models.ForeignKey(Competition, on_delete=models.CASCADE, verbose_name='Змагання')
    type_distance = models.ForeignKey(TypeDistance, on_delete=models.PROTECT, verbose_name='Тип дистанції')
    name = models.CharField(max_length=255, verbose_name='Назва')
    descr = models.TextField(verbose_name='Детальний опис', blank=True)
    distance_length = models.IntegerField(verbose_name='Довжина дистанції в м', blank=True)
    enable_registration = models.IntegerField(choices=Common.YESNO_CHOICE, default=0,
        verbose_name='Дозволити реєстрацію')
    start_number = models.CharField(max_length=50, verbose_name='Початковий стартовий номер', blank=True)
    end_number = models.CharField(max_length=50, verbose_name='Кінцевий стартовий номер', blank=True)
    is_show_number = models.IntegerField(choices=Common.YESNO_CHOICE, default=0,
        verbose_name='Показувати стартовий номер')
    is_free = models.IntegerField(choices=Common.YESNO_CHOICE, default=0,
        verbose_name='Безкоштовно')
    register_cost = models.FloatField(verbose_name='Вартість реєстрації', default=0)
    register_cost_reduced = models.FloatField(verbose_name='Вартість пільгової реєстрації', default=0)

    def __str__(self):
        return f'{self.competition.name} - {self.name}'

    class Meta:
        verbose_name = 'Дистанція'
        verbose_name_plural = 'Дистанції'
```

```
class DistanceAddOption(models.Model):
    distance = models.ForeignKey(Distance, on_delete=models.CASCADE, verbose_name='Дистанція')
    add_option = models.ForeignKey(AdditionalOption, on_delete=models.PROTECT, verbose_name='Додаткова опція')
    cost_addoption = models.FloatField(verbose_name='Вартість', default=0)

    def __str__(self):
        return f'{self.distance.name} - {self.add_option.name}'

    class Meta:
        verbose_name = 'Додаткова опція дистанції'
        verbose_name_plural = 'Додаткові опції дистанцій'


class AgeCategory(models.Model):
    distance = models.ForeignKey(Distance, on_delete=models.CASCADE)
    gender = models.IntegerField(choices=Common.GENDER_CHOICE, default=Common.EMPTY,
        verbose_name='Стать')
    name = models.CharField(max_length=50, verbose_name='Назва')
    age_from = models.IntegerField(verbose_name='Вік з')
    age_to = models.IntegerField(verbose_name='Вік по')

    def __str__(self):
        return f'{self.distance.name} - {self.gender} з {self.age_from} по {self.age_to}'

    class Meta:
        verbose_name = 'Вікова категорія'
        verbose_name_plural = 'Вікові категорії'


class Promocode(models.Model):
    competition = models.ForeignKey(Competition, on_delete=models.CASCADE, verbose_name='Змагання')
    distance = models.ForeignKey(Distance, on_delete=models.CASCADE, verbose_name='Дистанція')
    type_promocode = models.IntegerField(choices=Common.TYPE_PROMOCODE_CHOICE, default=0,
```

```
        verbose_name='Тип промокоду')
    promocode = models.CharField(max_length=20, verbose_name='Промокод')
    is_active = models.IntegerField(choices=Common.YESNO_CHOICE, default=0,
        verbose_name='Активний')
    is_onetime = models.IntegerField(choices=Common.YESNO_CHOICE, default=0,
        verbose_name='Одноразовий')
    date_start = models.DateField(verbose_name='Дата початку дії', blank=True,
null=True)
    date_end = models.DateField(verbose_name='Дата закінчення дії', blank=True,
null=True)
    percent_discount = models.FloatField(verbose_name='% знижки', default=0)
    sum_discount = models.FloatField(verbose_name='Сума знижки', default=0)
    sum_register = models.FloatField(verbose_name='Сума реєстрації', default=0)

    def __str__(self):
        return f'{self.distance.name} - {self.promocode}'

    class Meta:
        verbose_name = 'Промокод'
        verbose_name_plural = 'Промокоди'


class Order(models.Model):
    number = models.CharField(max_length=100, verbose_name='Номер замовлення')
    date_order = models.DateField(verbose_name='Дата замовлення')
    time_order = models.TimeField(verbose_name='Час замовлення', blank=True)
    status = models.IntegerField(choices=Common.ORDER_STATUS_CHOICE,
default=0,
        verbose_name='Статус замовлення')
    sum_order = models.FloatField(verbose_name='Сума замовлення', default=0)
    sum_paym = models.FloatField(verbose_name='Сплачена сума', default=0)
    date_paym = models.DateTimeField(verbose_name='Дата сплати', blank=True)
    person = models.ForeignKey(Person, on_delete=models.PROTECT,
verbose_name='Учасник')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Створено
в')
    created_by = models.BigIntegerField(verbose_name='Створив')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Оновлено в')
    updated_by = models.BigIntegerField(verbose_name='Оновив')

    def __str__(self):
```

```
        return f'{self.number}, {self.date_order}, {self.status}'

    class Meta:
        verbose_name = 'Замовлення'
        verbose_name_plural = 'Замовлення'


class OrderSpec(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE, verbose_name='Замовлення')
    person = models.ForeignKey(Person, on_delete=models.PROTECT, verbose_name='Учасник')
    type_spec = models.IntegerField(choices=Common.TYPE_ORDER_ITEM_CHOICE, default=0,
                                    verbose_name='тип специфікації замовлення')
    competition = models.ForeignKey(Competition, on_delete=models.PROTECT, verbose_name='Змагання')
    distance = models.ForeignKey(Distance, on_delete=models.PROTECT, verbose_name='Дистанція')
    start_number = models.CharField(max_length=50, verbose_name='Стартовий номер', blank=True)
    add_option = models.ForeignKey(AdditionalOption, on_delete=models.PROTECT, blank=True, null=True, verbose_name='Додаткова опція')
    cost = models.FloatField(verbose_name='Вартість', default=0)
    reduced_cost = models.FloatField(verbose_name='Знижена вартість', default=0)
    is_free = models.IntegerField(choices=Common.YESNO_CHOICE, default=0, verbose_name='Безкоштовно')
    promocode = models.CharField(max_length=20, verbose_name='Промокод', blank=True)

    def __str__(self):
        return f'{self.order.number}, {self.type_spec}, {self.competition}, {self.distance}'

    class Meta:
        verbose_name = 'Специфікація замовлення'
        verbose_name_plural = 'Специфікації замовленнь'


class PaymentTransaction(models.Model):
    order = models.ForeignKey(Order, on_delete=models.DO_NOTHING)
    date_paym = models.DateTimeField(verbose_name='Дата сплати')
```

```
    sum_paym = models.FloatField(verbose_name='Сума сплати')
    transaction_info = models.TextField(verbose_name='Інформація про транзакцію', blank=True)

    def __str__(self):
        return f'{self.order.number}, {self.date_paym}, {self.sum_paym}'

    class Meta:
        verbose_name = 'Платіжна транзакція'
        verbose_name_plural = 'Платіжні транзакції'


class Result(models.Model):
    competition = models.ForeignKey(Competition, on_delete=models.PROTECT, verbose_name='Змагання')
    distance = models.ForeignKey(Distance, on_delete=models.PROTECT, verbose_name='Дистанція')
    person = models.ForeignKey(Person, on_delete=models.PROTECT, verbose_name='Учасник')
    order_spec = models.ForeignKey(OrderSpec, on_delete=models.PROTECT, verbose_name='Реєстрація')
    result_number = models.IntegerField(verbose_name='Результат', blank=True, null=True)
    result_time = models.TimeField(verbose_name='Час результату', blank=True, null=True)

    def __str__(self):
        return f'{self.competition.name}: {self.distance.name}, {self.result_number}'

    class Meta:
        verbose_name = 'Результат змагання'
        verbose_name_plural = 'Результати змагань'
```

app/permissions.py

```
from rest_framework import permissions
from .models import Person, CompetitionOrganizer
from .common import Common


class IsOrganizer(permissions.BasePermission):
```

```
    def has_permission(self, request, view):
        return request.user and request.user.is_authenticated

    def has_object_permission(self, request, view, obj):
        return Person.objects.filter(user=request.user, role=2).exists()
```

app/serializer.py

```
from rest_framework import fields, serializers

from .models import (Country, Region, City, Person, TypeCompetition,
                     Competition, CompetitionOrganizer, AdditionalOption, TypeDistance,
                     Distance, DistanceAddOption, AgeCategory, Promocode, Order,
                     OrderSpec, PaymentTransaction, Result)


class CountrySerializer(serializers.ModelSerializer):
    class Meta:
        model = Country
        fields = '__all__'


class RegionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Region
        fields = '__all__'


class CitySerializer(serializers.ModelSerializer):
    class Meta:
        model = City
        fields = '__all__'


class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        fields = '__all__'
```

```
class TypeCompetitionSerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeCompetition
        fields = '__all__'


class CompetitionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Competition
        fields = '__all__'


class CompetitionOrganizerSerializer(serializers.ModelSerializer):
    class Meta:
        model = CompetitionOrganizer
        fields = '__all__'


class AdditionalOptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = AdditionalOption
        fields = '__all__'


class TypeDistanceSerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeDistance
        fields = '__all__'


class DistanceSerializer(serializers.ModelSerializer):
    class Meta:
        model = Distance
        fields = '__all__'


class DistanceAddOptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = DistanceAddOption
        fields = '__all__'
```

```
class AgeCategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = AgeCategory
        fields = '__all__'


class PromocodeSerializer(serializers.ModelSerializer):
    class Meta:
        model = Promocode
        fields = '__all__'


class OrderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order
        fields = '__all__'


class OrderSpecSerializer(serializers.ModelSerializer):
    class Meta:
        model = OrderSpec
        fields = '__all__'


class PaymentTransactionSerializer(serializers.ModelSerializer):
    class Meta:
        model = PaymentTransaction
        fields = '__all__'


class ResultSerializer(serializers.ModelSerializer):
    class Meta:
        model = Result
        fields = '__all__'
```

app/urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
```

```
from .views import (CountryViewSet, RegionViewSet, CityViewSet, PersonViewSet,
                    TypeCompetitionViewSet, CompetitionViewSet,
CompetitionOrganizerViewSet,
                    AdditionalOptionViewSet, TypeDistanceViewSet, DistanceViewSet,
                    DistanceAddOptionViewSet, AgeCategoryViewSet, PromocodeViewSet,
                    OrderViewSet, OrderSpecViewSet, PaymentTransactionViewSet,
ResultViewSet)


router = DefaultRouter()
router.register(r'countries', CountryViewSet)
router.register(r'regions', RegionViewSet)
router.register(r'cities', CityViewSet)
router.register(r'persons', PersonViewSet)
router.register(r'typecompetitions', TypeCompetitionViewSet)
router.register(r'competitions', CompetitionViewSet)
router.register(r'competitionorganizers', CompetitionOrganizerViewSet)
router.register(r'addoptions', AdditionalOptionViewSet)
router.register(r'typedistances', TypeDistanceViewSet)
router.register(r'distances', DistanceViewSet)
router.register(r'distanceaddoptions', DistanceAddOptionViewSet)
router.register(r'agecategories', AgeCategoryViewSet)
router.register(r'promocodes', PromocodeViewSet)
router.register(r'orders', OrderViewSet)
router.register(r'orderspecs', OrderSpecViewSet)
router.register(r'paymtransactions', PaymentTransactionViewSet)
router.register(r'results', ResultViewSet)

urlpatterns = [
    path('', include(router.urls))
]
```

app/views.py

```
from django.shortcuts import render
from rest_framework import viewsets
from rest_framework import permissions
from rest_framework_simplejwt.authentication import JWTAuthentication
from rest_framework.exceptions import PermissionDenied
from .permissions import IsOrganizer
```

```
from .models import (Country, Region, City, Person, TypeCompetition,
                     Competition, CompetitionOrganizer, AdditionalOption,
                     TypeDistance, Distance, DistanceAddOption, AgeCategory,
                     Promocode, Order, OrderSpec, PaymentTransaction, Result)
from .serializer import (CountrySerializer, RegionSerializer, CitySerializer,
                         PersonSerializer, TypeCompetitionSerializer, CompetitionSerializer,
                         CompetitionOrganizerSerializer, AdditionalOptionSerializer,
                         TypeDistanceSerializer,                    DistanceSerializer,
DistanceAddOptionSerializer,
                         AgeCategorySerializer, PromocodeSerializer, OrderSerializer,
                         OrderSpecSerializer, PaymentTransactionSerializer, ResultSerializer)


# Create your views here.
class CountryViewSet(viewsets.ModelViewSet):
    queryset = Country.objects.all()
    serializer_class = CountrySerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]

class RegionViewSet(viewsets.ModelViewSet):
    queryset = Region.objects.all()
    serializer_class = RegionSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class CityViewSet(viewsets.ModelViewSet):
    queryset = City.objects.all()
    serializer_class = CitySerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class PersonViewSet(viewsets.ModelViewSet):
    queryset = Person.objects.all()
    serializer_class = PersonSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]
```

```
class TypeCompetitionViewSet(viewsets.ModelViewSet):
    queryset = TypeCompetition.objects.all()
    serializer_class = TypeCompetitionSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class CompetitionViewSet(viewsets.ModelViewSet):
    queryset = Competition.objects.all()
    serializer_class = CompetitionSerializer
    authentication_classes = [JWTAuthentication]
#    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def perform_create(self, serializer):
        serializer.save(created_by=self.request.user)

    def perform_update(self, serializer):
        serializer.save(updated_by=self.request.user)

    def get_permissions(self):
        if self.action == 'create':
            self.permission_classes = [IsOrganizer]
        else:
            self.permission_classes = [permissions.IsAuthenticatedOrReadOnly]
        return super().get_permissions()

class CompetitionOrganizerViewSet(viewsets.ModelViewSet):
    queryset = CompetitionOrganizer.objects.all()
    serializer_class = CompetitionOrganizerSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class AdditionalOptionViewSet(viewsets.ModelViewSet):
    queryset = AdditionalOption.objects.all()
    serializer_class = AdditionalOptionSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]

class TypeDistanceViewSet(viewsets.ModelViewSet):
    queryset = TypeDistance.objects.all()
```

```
    serializer_class = TypeDistanceSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class DistanceViewSet(viewsets.ModelViewSet):
    queryset = Distance.objects.all()
    serializer_class = DistanceSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class DistanceAddOptionViewSet(viewsets.ModelViewSet):
    queryset = DistanceAddOption.objects.all()
    serializer_class = DistanceAddOptionSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class AgeCategoryViewSet(viewsets.ModelViewSet):
    queryset = AgeCategory.objects.all()
    serializer_class = AgeCategorySerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class PromocodeViewSet(viewsets.ModelViewSet):
    queryset = Promocode.objects.all()
    serializer_class = PromocodeSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class OrderViewSet(viewsets.ModelViewSet):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class OrderSpecViewSet(viewsets.ModelViewSet):
```

```
    queryset = OrderSpec.objects.all()
    serializer_class = OrderSpecSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class PaymentTransactionViewSet(viewsets.ModelViewSet):
    queryset = PaymentTransaction.objects.all()
    serializer_class = PaymentTransactionSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticated]


class ResultViewSet(viewsets.ModelViewSet):
    queryset = Result.objects.all()
    serializer_class = ResultSerializer
    authentication_classes = [JWTAuthentication]
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
```

sportsevents/urls.py

```
"""
URL configuration for sportsevents project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('app.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

sportsevents/settings.py

```
"""
Django settings for sportsevents project.

Generated by 'django-admin startproject' using Django 5.0.4.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""
from datetime import timedelta
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-af06v7o^a5ys8d$1gy1g-_jw5%qw6pwebtex953khg0u(hbuzt'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_simplejwt',
#    'rest_framework.authtoken',
    'app',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'sportsevents.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
```

```
    },
]

WSGI_APPLICATION = 'sportsevents.wsgi.application'


'''REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [],
    'TEST_REQUEST_DEFAULT_FORMAT': 'json'
}'''

# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': BASE_DIR / 'db.sqlite3',
#     }
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'sportsevents',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_PARSER_CLASSES': [
        'rest_framework.parsers.JSONParser',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework_simplejwt.authentication.JWTAuthentication'
    ],
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=1),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
    'SLIDING_TOKEN_LIFETIME': timedelta(days=30),
    'SLIDING_TOKEN_REFRESH_LIFETIME_LATE_USER': timedelta(days=1),
    'SLIDING_TOKEN_LIFETIME_LATE_USER': timedelta(days=30),
}

# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True
```

```
USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```