Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут штучного інтелекту та робототехніки

Кафедра штучного інтелекту та аналізу даних

Лучанецький Роман Валерійович,

студент групи AI-231

Курсова робота з дисципліни

«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

для здобувачів першого (бакалаврського) рівня вищої освіти

за спеціальністю 122 Комп’ютерні науки

освітня програма «Комп’ютерні науки»

Затверджено на засіданні

кафедри інформаційних систем

протокол №1 від 28 серпня

2024 р. Одеса – 2025

АНОТАЦІЯ

Лучанецький Р.В. Система керування футбольними турнірами: курсова робота з дисципліни «Об’єктно-орієнтоване програмування» за спеціальністю «122 Комп’ютерні науки» / Лучнецький Роман Валерійович; керівник Микола Анатолійович Годовиченко. – Одеса : Нац. ун-т «Одеська політехніка», 2025. – 35 с.

Курсова робота містить основну текстову частину на 35 сторінках, список використаних джерел з 5 найменувань на 1 сторінці, додатки на 1 сторінкі.

Розглянуто постановку задачі щодо автоматизації обліку футбольних турнірів, команд, гравців та детальної статистики матчів. Створено веб-застосунок на базі Spring Boot із застосуванням архітектурного стилю REST. Реалізовано моделі Tournament, Team, Player, Match, Participation, налаштовано схему бази даних PostgreSQL і виконано деплой сервісу на платформу Render. Запропоновані запити для більш детального тестування проєкту.

Ключові слова: Spring Boot, база даних, REST API, Render, Java.

Вступ

Мета курсової роботи.

Систематизувати, поглибити та практично закріпити знання з дисципліни «Об’єктно-орієнтоване програмування», а також засвоїти навички самостійної розробки серверної частини прикладного програмного забезпечення засобами Java та фреймворку Spring Boot.

Актуальність теми.

Сучасні інформаційні технології відіграють ключову роль в автоматизації та оптимізації процесів управління бібліотечними системами. Потреба у цифрових рішеннях цього напряму стабільно зростає, що забезпечує незмінну актуальність розробленого програмного продукту.

Завдання, виконані в межах проєкту:

- Створення RESTful веб-сервісу з повною підтримкою CRUD-операцій.
- Зберігання та обробка даних у реляційній базі даних.
- Коректна взаємодія компонентів застосунку завдяки чіткому поділу на шари (Controller → Service → Repository).
- Розробка архітектури на основі Spring Framework із використанням Spring Data JPA, Lombok, MapStruct та інших сучасних інструментів.
- Реалізація моделі предметної області з такими сутностями: Author, Book, Genre, Reader, Loan, User.

Створення захищеного REST API, що забезпечує:

- додавання, перегляд, оновлення та видалення інформації;
- автентифікацію та авторизацію користувачів на основі Spring Security та JWT.

Аналіз предметної області

Система керування спортивними змаганнями призначена для автоматизації ключових процесів організації турнірів: реєстрації команд і гравців, формування календаря матчів, збирання результатів та побудови турнірних таблиць. Автоматизація зменшує ризик помилок ручного обліку, пришвидшує прийняття управлінських рішень і підвищує прозорість змагань для всіх учасників.

Таблиця 1.1 - ключові сутності та їхні атрибути

Сутність	Основні атрибути	Зв’язки
Tournament	id, name, startDate, endDate, location	1 ← Participation → N
Team	id, name, city	1 ← Player N 1 ← Participation → N
Player	id, name, birthDate, position	N → 1 (Team)
Match	id, date, scoreA, scoreB	N → 1 (Tournament), 2 × 1 (Team)
Participation	id	N → 1 (Team), N → 1 (Tournament)

Логічні обмеження:

- Команда може брати участь у багатьох турнірах.
- Гравець належить лише одній команді.
- Матч належить одному турніру і включає дві різні команди.

Таблиця 1.2 - Типові проблеми галузі

Проблема	Наслідки
Помилки ручного введення	Дублювання гравців/команд, перекручені статистичні дані.
Складність формування календаря	Накладання матчів у часі, нерівномірне навантаження на команди.
Відсутність оперативної аналітики	Керівництво не бачить поточний рейтинг команд та індивідуальну статистику гравців у режимі реального часу.

Під час розроблення проєкту, було створено такі сутності та їхні атрибути:

Сутності:

Tournament:

• id: Long

• name: String

• startDate: LocalDate

• endDate: LocalDate

• location: String

Team:

• id: Long

• name: String

• city: String

Player:

• id: Long

• name: String

• birthDate: LocalDate

• position: String

• team: Team

Match:

• id: Long

• tournament: Tournament

• teamA: Team

• teamB: Team

• date: LocalDate

• scoreA: Integer

• scoreB: Integer

Participation:

• id: Long

• team: Team

• tournament: Tournament

Для кожної сутності системи створено окремий контролер, що надає набір REST-ендпоінтів за принципами CRUD і відповідає рекомендаціям RESTful-архітектури. Усі запити та відповіді передаються у форматі JSON. Доступ до більшості ресурсів захищено механізмом Bearer JWT.

AuthController

// Зареєструвати користувача – @PostMapping("/register")

// Увійти (отримати JWT) – @PostMapping("/login")

// Вийти (очистити JWT cookie) – @PostMapping("/logout")

TournamentController

// Створити новий турнір – @PostMapping

// Отримати всі турніри – @GetMapping

// Оновити турнір за ID – @PutMapping("/{id}")

// Видалити турнір за ID – @DeleteMapping("/{id}")

// Отримати турнірну таблицю – @GetMapping("/{id}/standings")

// Отримати список переможців – @GetMapping("/winners")

TeamController

// Додати команду – @PostMapping

// Отримати всі команди – @GetMapping

// Оновити команду за ID – @PutMapping("/{id}")

// Видалити команду за ID – @DeleteMapping("/{id}")

// Отримати всі матчі команди – @GetMapping("/{teamId}/matches")

PlayerController

// Додати гравця – @PostMapping

// Отримати гравців конкретної команди – @GetMapping("/team/{teamId}")

// Отримати середній вік гравців команди – @GetMapping("/average-age/{teamId}")

// Отримати статистику гравців команди – @GetMapping("/statistics/{teamId}")

// Оновити гравця за ID – @PutMapping("/{id}")

// Видалити гравця за ID – @DeleteMapping("/{id}")

ParticipationController

// Зареєструвати команду в турнірі – @PostMapping

// Отримати всі команди турніру – @GetMapping("/tournament/{tournamentId}")

// Отримати всі турніри команди – @GetMapping("/team/{teamId}")

MatchController

// Додати матч – @PostMapping

// Отримати матчі турніру – @GetMapping("/tournament/{tournamentId}")

// Отримати розклад турніру – @GetMapping("/tournament/{tournamentId}/schedule")

// Оновити матч за ID – @PutMapping("/{id}")

// Видалити матч за ID – @DeleteMapping("/{id}")

Реалізація програмного продукту

В даному проєкту реалізовані такі моделі:

User:

- id : Long — первинний ключ
- username : String — унікальний логін
- password : String — хеш пароля (Bcrypt)

```
package com.example.courseworkLuchnetskyi.model;


import jakarta.persistence.*;
import lombok.*;


@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;
}
```

Tournament:

- id : Long
- name : String
- startDate : LocalDate / endDate : LocalDate
- location : String
- matches : List<Match> — @OneToMany(mappedBy = "tournament")

```
package com.example.courseworkLuchnetskyi.model;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDate;
import java.util.List;
import com.example.courseworkLuchnetskyi.model.Match;

@Entity
@Table(name = "tournaments")
@Data
@NoArgsConstructor
```

```
@AllArgsConstructor
public class Tournament {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private LocalDate startDate;

    private LocalDate endDate;

    private String location;

    @OneToMany(mappedBy = "tournament", cascade = CascadeType.ALL)
    private List<Match> matches;
}
```

Team:

- id : Long
- name : String
- city : String

- players : List<Player> — @OneToMany(mappedBy = "team", cascade = ALL)

```
package com.example.courseworkLuchnetskyi.model;


import jakarta.persistence.*;
import lombok.*;
import java.util.List;
import com.example.courseworkLuchnetskyi.model.Player;


@Entity
@Table(name = "teams")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Team {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String city;
```

```
    @OneToMany(mappedBy = "team", cascade = CascadeType.ALL)
    private List<Player> players;
}
```

Player:

- id : Long
- name : String
- birthDate : LocalDate
- position : String
- team : Team — @ManyToOne
- participations : List<Participation> — зв’язок із голами/матчами

```
package com.example.courseworkLuchnetskyi.model;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDate;
import java.util.List;

@Entity
@Table(name = "players")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Player {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private String name;

    private LocalDate birthDate;

    private String position;

    @ManyToOne
    private Team team;

    @OneToMany(mappedBy = "player")
    private List<Participation> participations;

    public int getMatchesPlayed() {
      return participations != null ? participations.size() : 0;
    }

    public int getGoalsScored() {
      return participations != null
        ?
participations.stream().mapToInt(Participation::getGoals).sum()
        : 0;
    }

}
```

Match:

- id : Long
- tournament : Tournament — @ManyToOne
- teamA : Team, teamB : Team — дві різні команди (@ManyToOne)
- date : LocalDate
- scoreA : Integer, scoreB : Integer

```
package com.example.courseworkLuchnetskyi.model;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDate;

@Entity
@Table(name = "matches")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Match {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Tournament tournament;

    @ManyToOne
    private Team teamA;

    @ManyToOne
    private Team teamB;

    private LocalDate date;
```

```
    private Integer scoreA;

    private Integer scoreB;
}
```

Participation:

- id : Long
- team : Team — @ManyToOne
- tournament : Tournament — @ManyToOne
- player : Player — автор гола в матчі
- goals : int — кількість забитих голів у конкретному матчі

```
package com.example.courseworkLuchnetskyi.model;

import jakarta.persistence.*;
import lombok.*;
import com.example.courseworkLuchnetskyi.model.Team;
import com.example.courseworkLuchnetskyi.model.Tournament;

@Entity
@Table(name = "participations")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Participation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
```

```
    private Team team;

    @ManyToOne
    private Tournament tournament;

    @ManyToOne
    private Player player;

    private int goals;
}
```

JwtUtil:
утиліта для створення, читання та валідації JWT-токенів

```
package com.example.courseworkLuchnetskyi.config;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.JwtException;
import io.jsonwebtoken.io.Decoders;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import javax.crypto.SecretKey;
import java.util.Date;

@Component
public class JwtUtil {

  @Value("${jwt.secret}")
  private String jwtSecret;
```

```
  @Value("${jwt.expiration}")
  private int jwtExpirationMs;

  private SecretKey getSigningKey() {
    byte[] keyBytes = Decoders.BASE64.decode(jwtSecret);
    return Keys.hmacShaKeyFor(keyBytes);
  }

  public String generateToken(String email) {
    return Jwts.builder()
      .setSubject(email)
      .setIssuedAt(new Date())
      .setExpiration(new Date(System.currentTimeMillis() +
jwtExpirationMs))
      .signWith(getSigningKey(), SignatureAlgorithm.HS256)
      .compact();
  }

  public String getEmailFromJwt(String token) {
    return Jwts.parserBuilder()
      .setSigningKey(getSigningKey())
      .build()
      .parseClaimsJws(token)
      .getBody()
      .getSubject();
  }

  public boolean validateJwt(String token) {
    try {
```

```
      Jwts.parserBuilder()
        .setSigningKey(getSigningKey())
        .build()
        .parseClaimsJws(token);
      return true;
    } catch (JwtException | IllegalArgumentException e) {
      return false;
    }
  }
}
```

SecurityConfig

загальна конфігурація Spring Security 6. Дозволяє тільки /api/auth/** без авторизації, інші REST-шляхи — під JWT. Підключає фільтр JwtAuthenticationFilter, задає BCrypt та DaoAuthenticationProvider.

```
package com.example.courseworkLuchnetskyi.config;


import
com.example.courseworkLuchnetskyi.security.JwtAuthenticationFilter;
import
com.example.courseworkLuchnetskyi.service.UserDetailsService;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationPr
ovider;
```

```
import
org.springframework.security.config.annotation.authentication.confi
guration.AuthenticationConfiguration;

import
org.springframework.security.config.annotation.web.builders.HttpSec
urity;

import
org.springframework.security.config.annotation.web.configuration.En
ableWebSecurity;

import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import
org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import
org.springframework.security.web.authentication.UsernamePasswordAut
henticationFilter;


@Configuration

@EnableWebSecurity

public class SecurityConfig {


  @Autowired private JwtAuthenticationFilter
jwtAuthenticationFilter;

  @Autowired private UserDetailsService userDetailsService;


  @Bean

  public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {

    http.csrf(csrf -> csrf.disable())

      .authorizeHttpRequests(auth -> auth

        .requestMatchers("/api/auth/**").permitAll()

        .requestMatchers("/api/tournaments/**", "/api/teams/**",
```

```
                        "/api/players/**",
"/api/participations/**",
                        "/api/matches/**").authenticated()
        .anyRequest().denyAll())
      .logout(logout -> logout.logoutSuccessUrl("/").permitAll())
      .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
      .exceptionHandling(e -> e.authenticationEntryPoint(
          (req, res, ex) ->
res.sendError(HttpServletResponse.SC_UNAUTHORIZED)));
    return http.build();
  }

  @Bean public PasswordEncoder passwordEncoder() { return new
BCryptPasswordEncoder(); }
  @Bean
  public AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider p = new DaoAuthenticationProvider();
    p.setUserDetailsService(userDetailsService);
    p.setPasswordEncoder(passwordEncoder());
    return p;
  }
  @Bean
  public AuthenticationManager
authenticationManager(AuthenticationConfiguration cfg) throws
Exception {
    return cfg.getAuthenticationManager();
  }
}
```

JwtAuthenticationFilter

єдиний фільтр витягує JWT із cookie, перевіряє валідність і ставить аутентифікацію в SecurityContextHolder. Працює до стандартного UsernamePasswordAuthenticationFilter.

```
package com.example.courseworkLuchnetskyi.security;


import com.example.courseworkLuchnetskyi.config.JwtUtil;

import
com.example.courseworkLuchnetskyi.service.UserDetailsService;

import java.io.IOException;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.Cookie;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import
org.springframework.security.authentication.UsernamePasswordAuthent
icationToken;

import
org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.util.Arrays;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;


@Component

public class JwtAuthenticationFilter extends OncePerRequestFilter {


  private final JwtUtil jwtUtil;
```

```
  private final UserDetailsService userDetailsService;

  @Autowired

  public JwtAuthenticationFilter(JwtUtil jwtUtil,
UserDetailsService userDetailsService) {

    this.jwtUtil = jwtUtil;

    this.userDetailsService = userDetailsService;

  }


  @Override

  protected void doFilterInternal(HttpServletRequest req,
HttpServletResponse res,

                                  FilterChain chain) throws
ServletException, IOException {

    String jwt =
Arrays.stream(Optional.ofNullable(req.getCookies()).orElse(new
Cookie[0]))

      .filter(c -> c.getName().equals("JWT_TOKEN"))

      .map(Cookie::getValue).findFirst().orElse(null);


    if (jwt != null && jwtUtil.validateJwt(jwt)) {

      String email = jwtUtil.getEmailFromJwt(jwt);

      UserDetails details =
userDetailsService.loadUserByUsername(email);

      UsernamePasswordAuthenticationToken auth =

        new UsernamePasswordAuthenticationToken(details, null,
details.getAuthorities());

      SecurityContextHolder.getContext().setAuthentication(auth);

    }

    chain.doFilter(req, res);

  }

}
```

CourseworkLuchanetskyi.java

стартує Spring Boot 3 застосунок.

```
@SpringBootApplication
public class CourseworkLuchanetskyi {
  public static void main(String[] args) {
    SpringApplication.run(CourseworkLuchanetskyi.class, args);
  }
}
```

```
MatchService / MatchServiceImpl
public interface MatchService {
    MatchResponseDto createMatch(MatchRequestDto dto);
    List<MatchResponseDto> getMatchesByTournament(Long
tournamentId);
    MatchResponseDto updateMatch(Long id, MatchRequestDto dto);
    void deleteMatch(Long id);
    List<MatchScheduleDto> getTournamentSchedule(Long
tournamentId);
}
```

інкапсулює всі правила створення/оновлення матчів (перевірка існування турніру й команд, мапінг DTO). getTournamentSchedule() формує легку DTO-пачку для фронту: id матчу, назви команд, дата.

```
ParticipationService / ParticipationServiceImpl
@Service @RequiredArgsConstructor
class ParticipationServiceImpl implements ParticipationService {
  private final ParticipationRepository partRepo;
  private final TeamRepository teamRepo;
  private final TournamentRepository tourRepo;
```

```
  private final ParticipationMapper mapper;


  @Override

  public ParticipationResponseDto
addParticipation(ParticipationRequestDto dto) {

      Participation p = mapper.toEntity(dto);

      p.setTeam(teamRepo.findById(dto.teamId()).orElseThrow(() ->
new RuntimeException("Team not found")));


p.setTournament(tourRepo.findById(dto.tournamentId()).orElseThrow((
) -> new RuntimeException("Tournament not found")));

      return mapper.toDto(partRepo.save(p));

  }

  /* getTeamsByTournament, getTournamentsByTeam → резолвлять списки
Participation */

}
```

єдина точка, де команда «підв’язується» до турніру; саме тут можна додати бізнес-валидацію (ліміт команд, дедлайни реєстрації тощо).

PlayerService / PlayerServiceImpl

```
public interface PlayerService {

  PlayerResponseDto createPlayer(PlayerRequestDto dto);

  List<PlayerResponseDto> getPlayersByTeam(Long teamId);

  PlayerResponseDto updatePlayer(Long id, PlayerRequestDto dto);

  void deletePlayer(Long id);

  TeamAverageAgeDto getAverageAgeForTeam(Long teamId);

  List<PlayerStatisticsDto> getPlayerStatisticsByTeam(Long teamId);

}
```

централізує логіку щодо гравців — від CRUD до «аналітики на льоту» (середній вік, статистика).

TeamService / TeamServiceImpl

```
List<TeamMatchDto> getMatchesForTeam(Long teamId) {

   List<Match> matches = matchRepo.findByTeamAIdOrTeamBId(teamId,
teamId);

   return matches.stream().map(m -> { … }).toList();

}
```

дозволяє тренеру чи фанатам швидко глянути, як команда грала проти опонентів.

TournamentService / TournamentServiceImpl

```
if (scoreA > scoreB) { teamAStats.setWins(+1);
teamAStats.setPoints(+3); }

else if (scoreA < scoreB) { /* зворотне */ }

else { teamAStats.setDraws(+1); teamBStats.setDraws(+1);
teamAStats.setPoints(+1); teamBStats.setPoints(+1); }
```

повна «міні-аналітика» всередині сервісу без окремих SQL-агрегацій, щоб швидко піднятися на початковому етапі. У майбутньому можна винести у нативні запити або materialized view.

UserDetailsService

```
@Service

class UserDetailsService implements
org.springframework.security.core.userdetails.UserDetailsService {

  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

      User user = userRepository.findByUsername(username)

                  .orElseThrow(() -> new
UsernameNotFoundException("Not found"));

      return new
org.springframework.security.core.userdetails.User(

              user.getUsername(), user.getPassword(), List.of());
```

```
    }
}
```

«клей» між власною таблицею users і Spring Security (DaoAuthenticationProvider). Повертає об’єкт UserDetails зі списком ролей (поки порожнім).

1 Додати змагання:

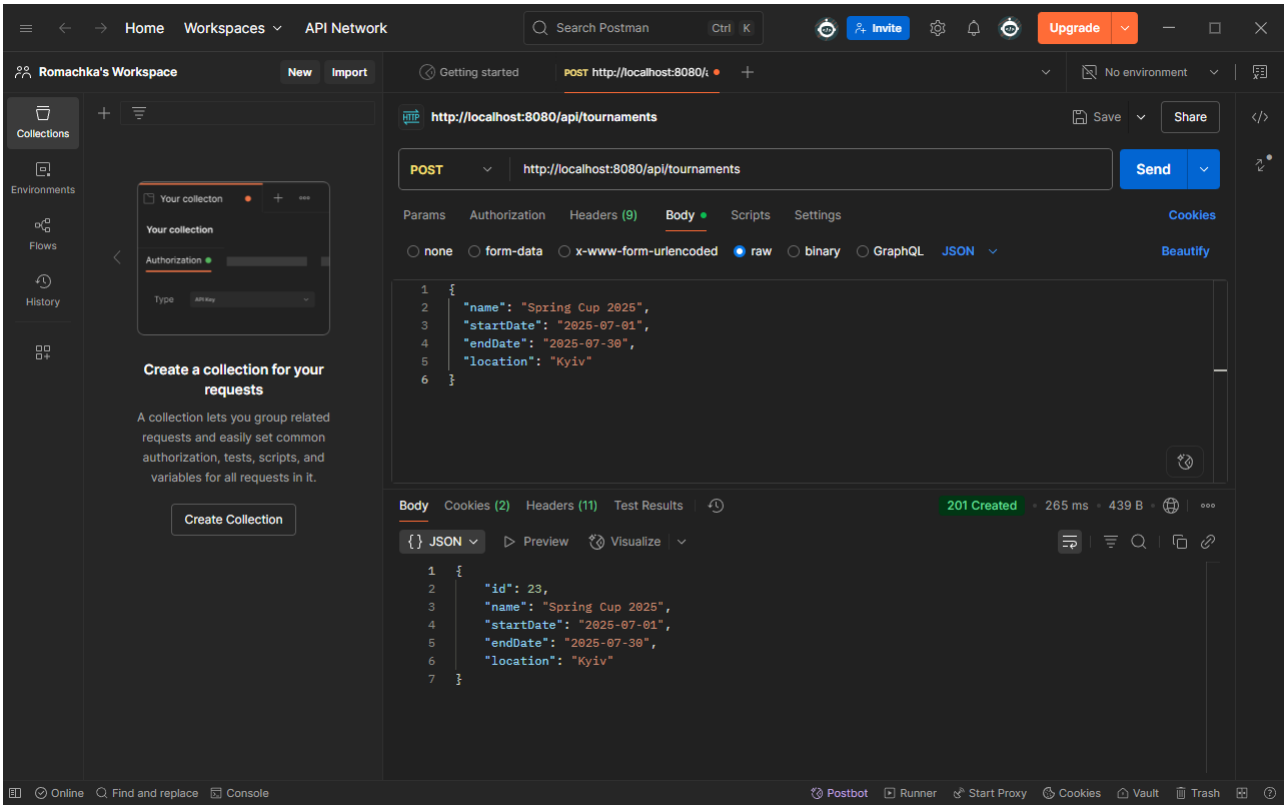

Рисунок 2.1

2 Отримати всі змагання:

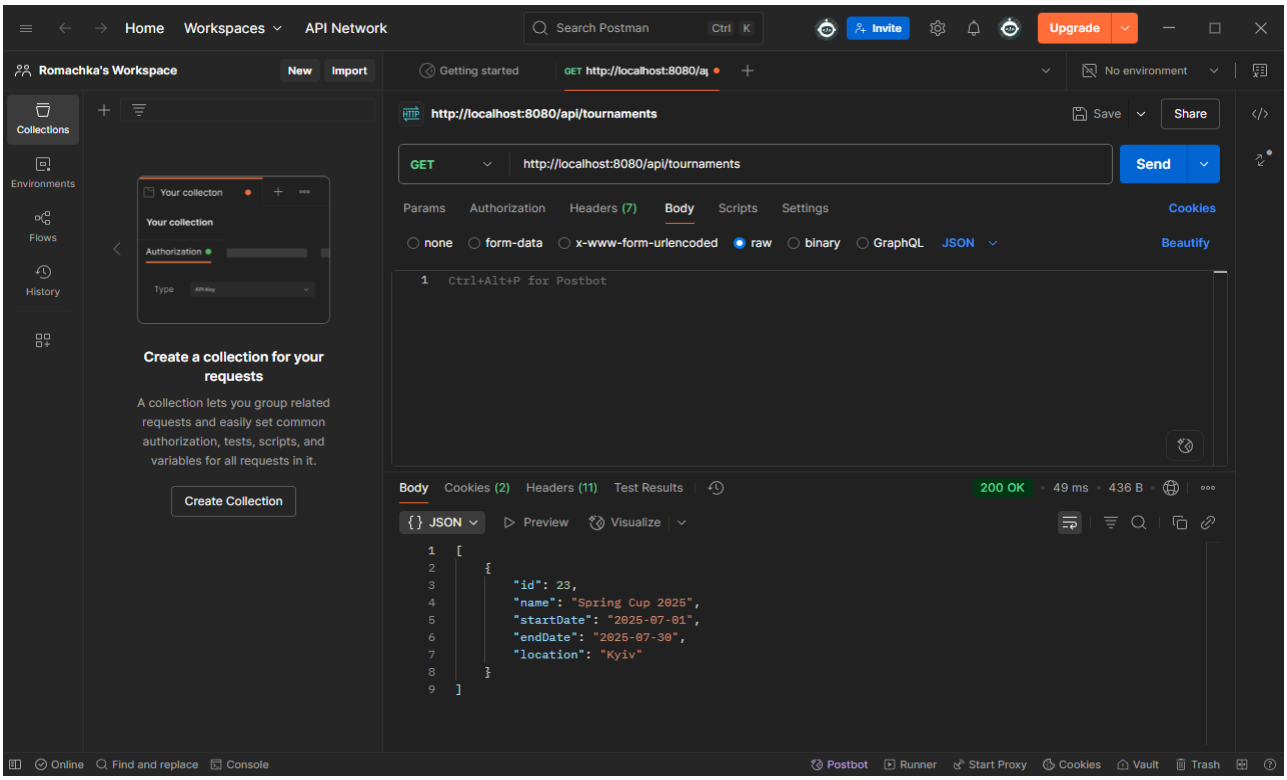

Рисунок 2.2

2 Оновити змагання:

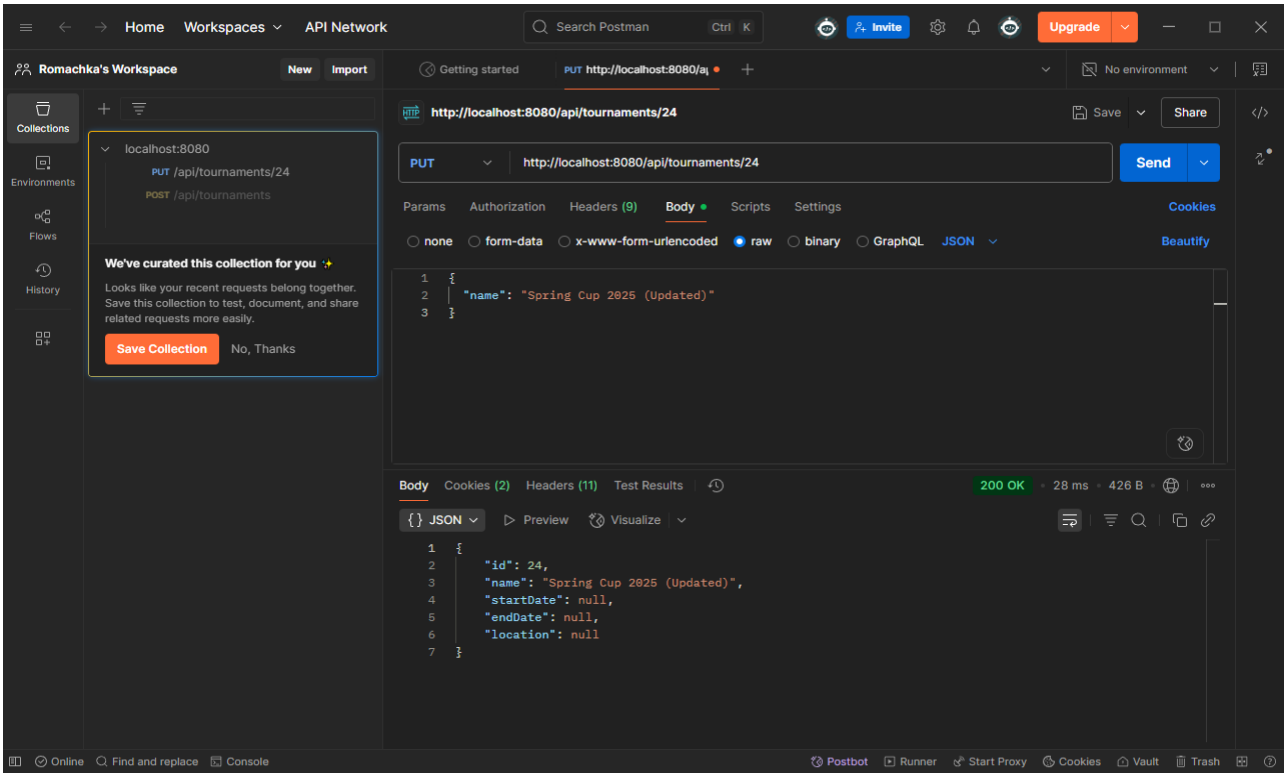

Рисунок 2.3

3 Видалити змагання:

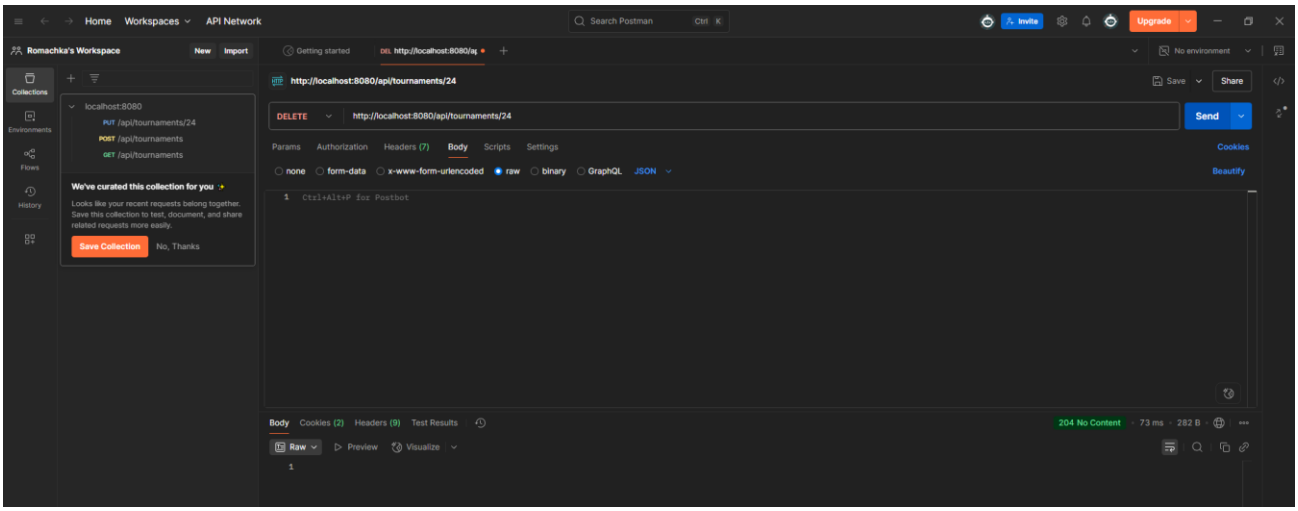

Рисунок 2.4

4 Додати команду

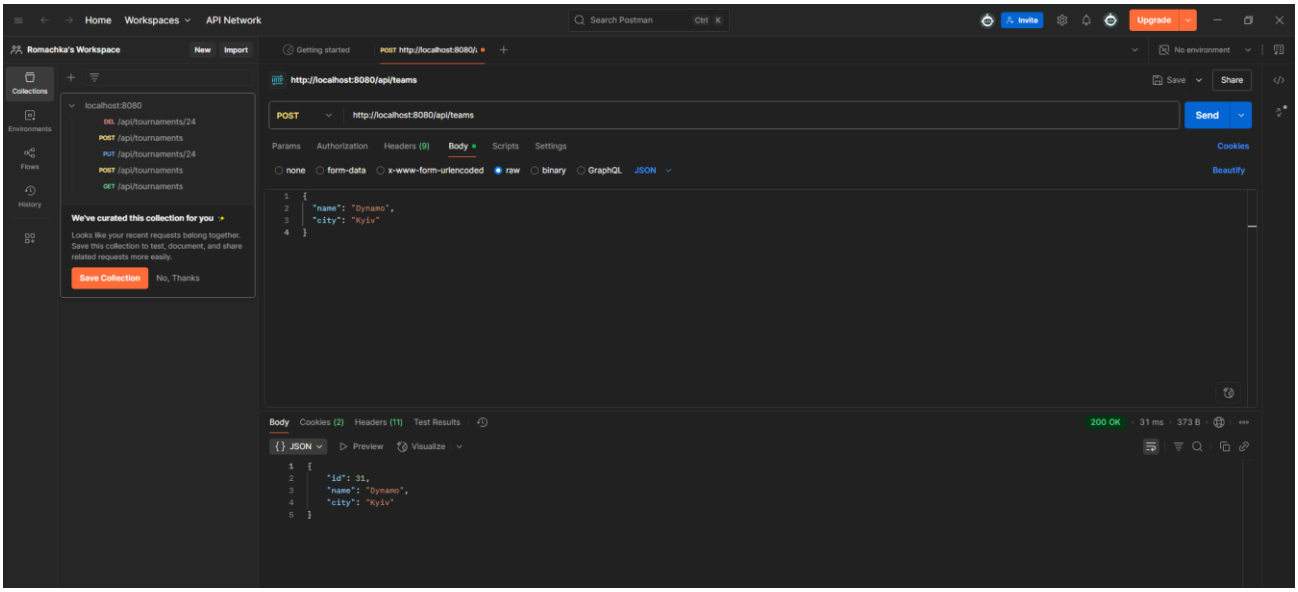

Рисунок 2.5

5 Отримати всі команди:

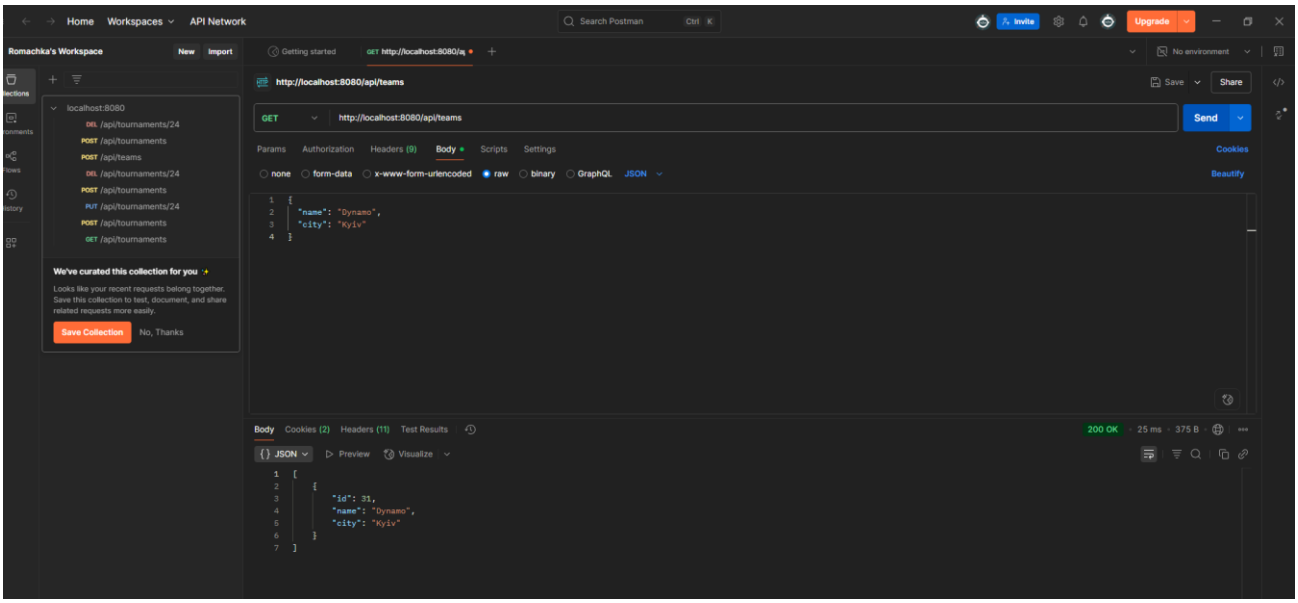

Рисунок 2.6

6 Оновити команду:

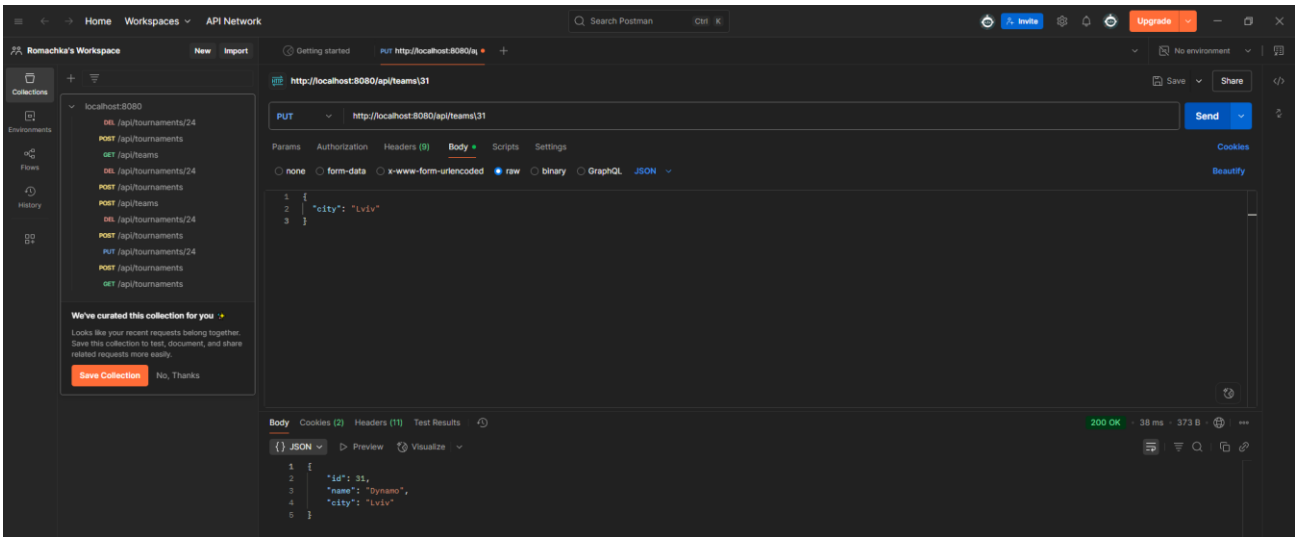

Рисунок 2.7

7 Видалити команду

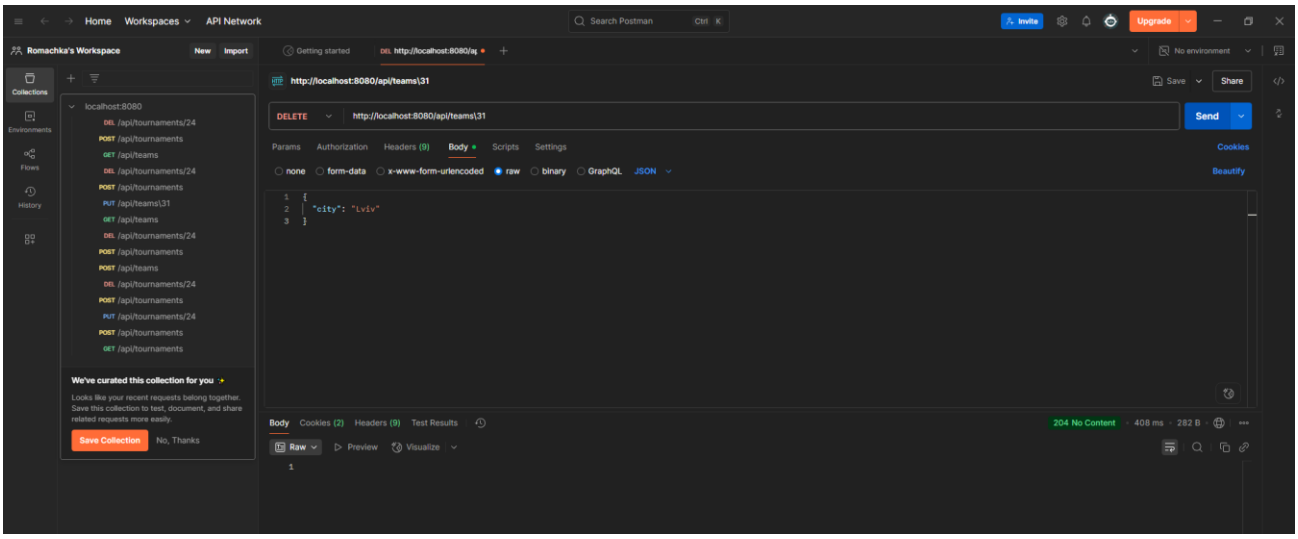

Рисунок 2.8

8 Додати гравця

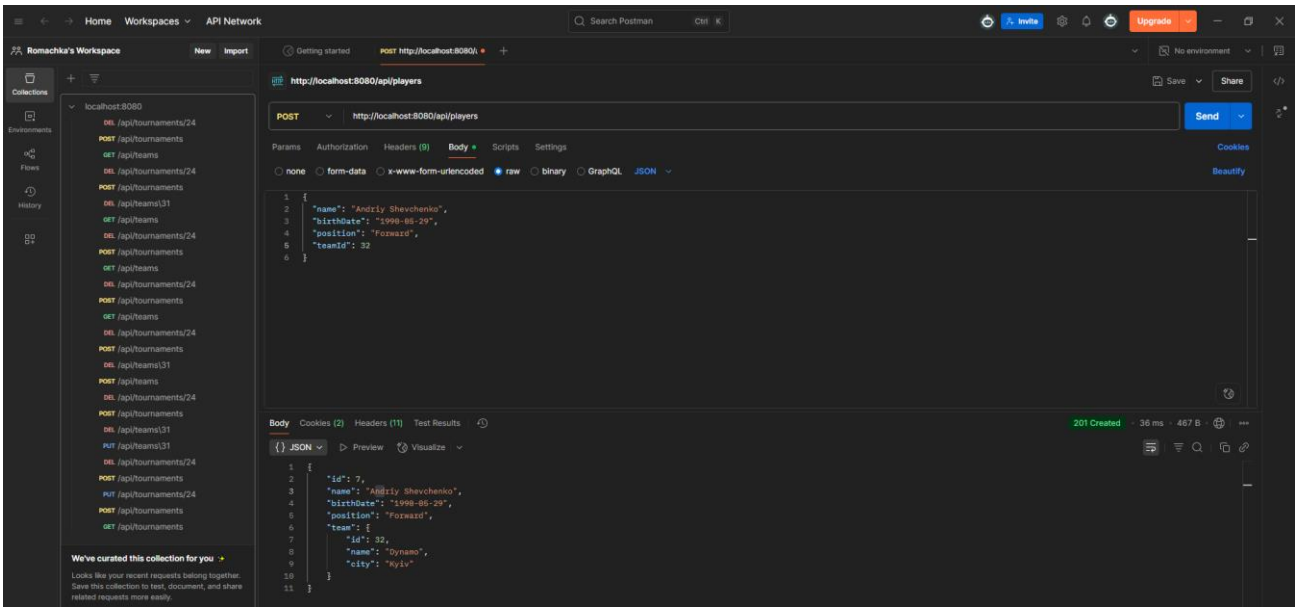

Рисунок 2.9

9 Отримати гравців команди

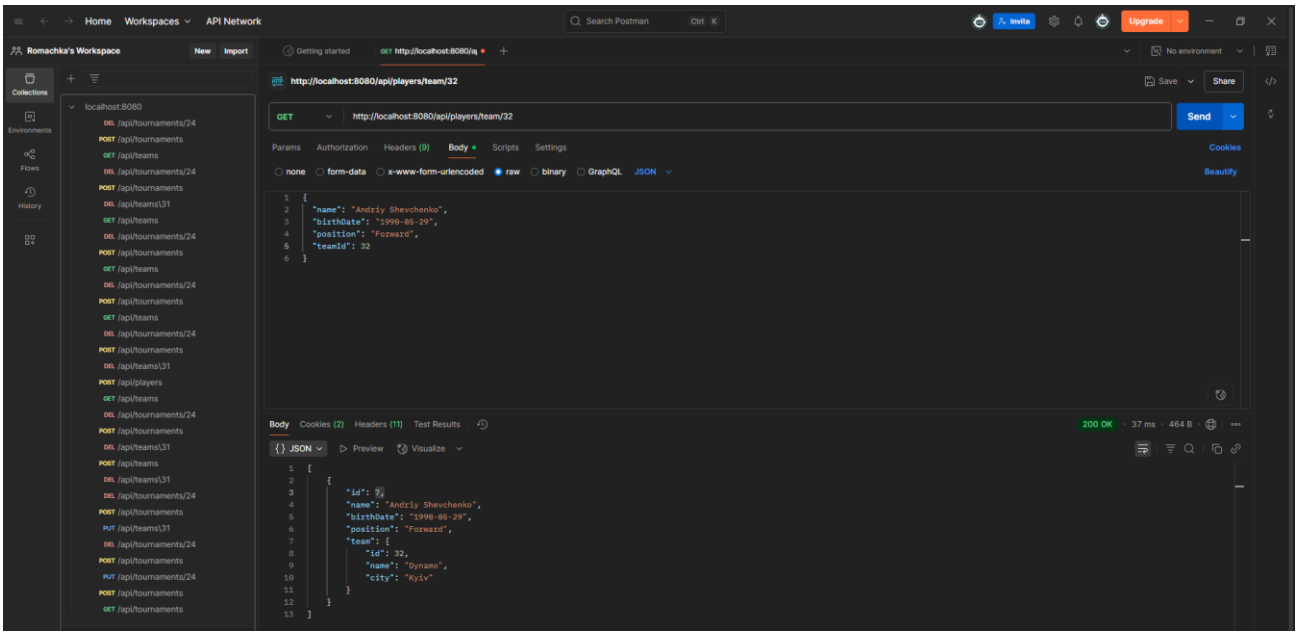

Рисунок 2.11

10 Оновити дані гравця

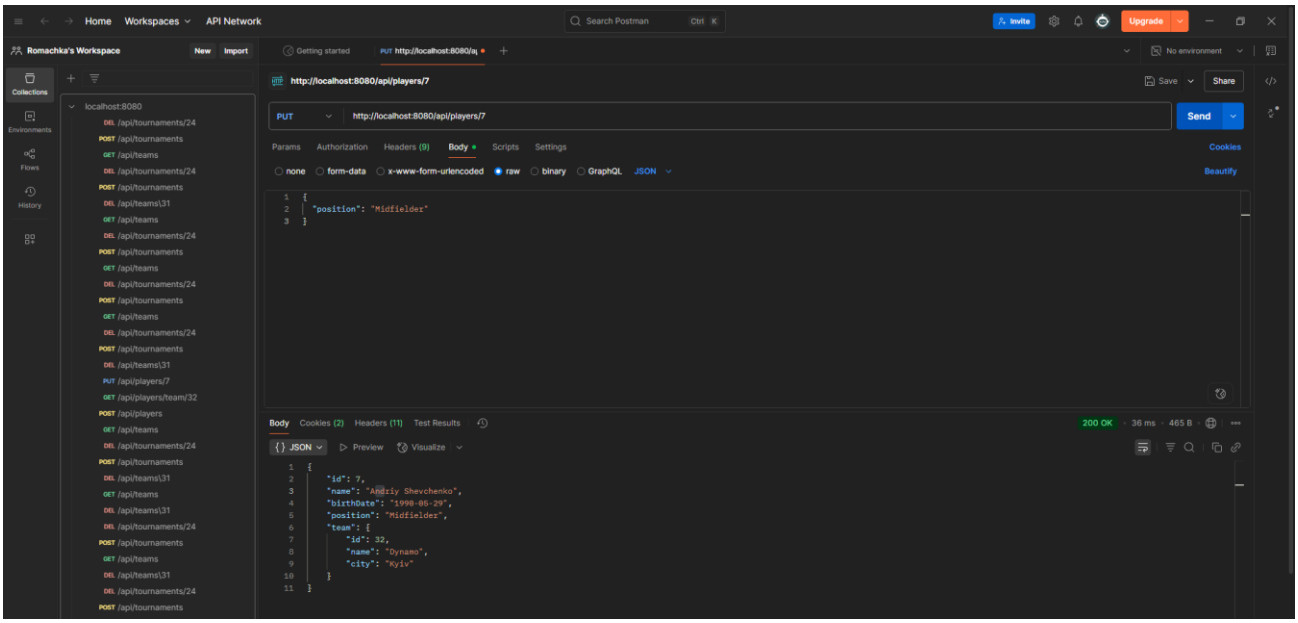

Рисунок 2.12

11 Видалити гравця

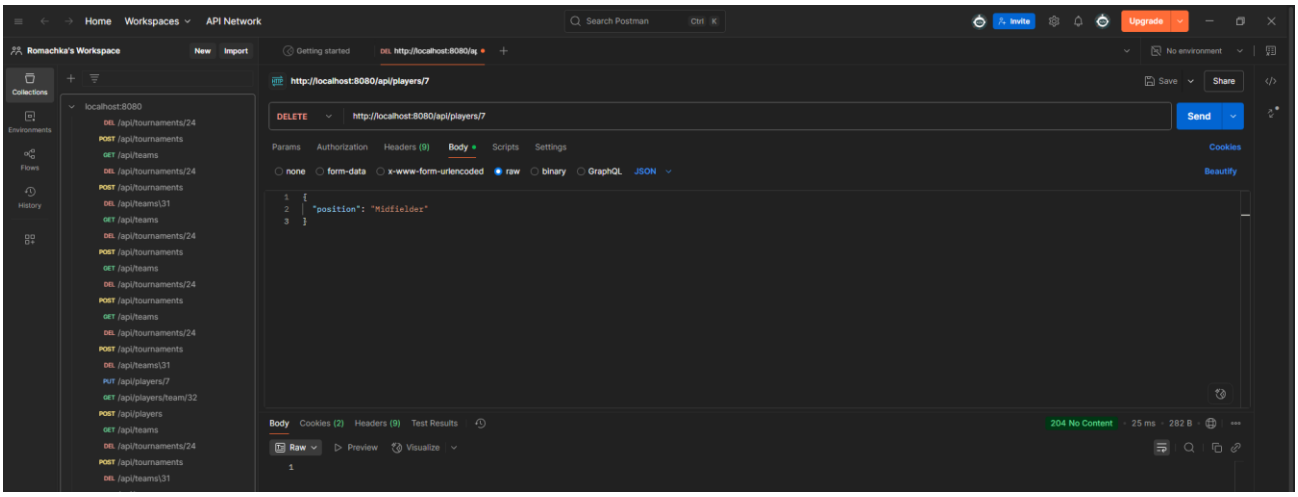

Рисунок 2.12

12 Додати матч

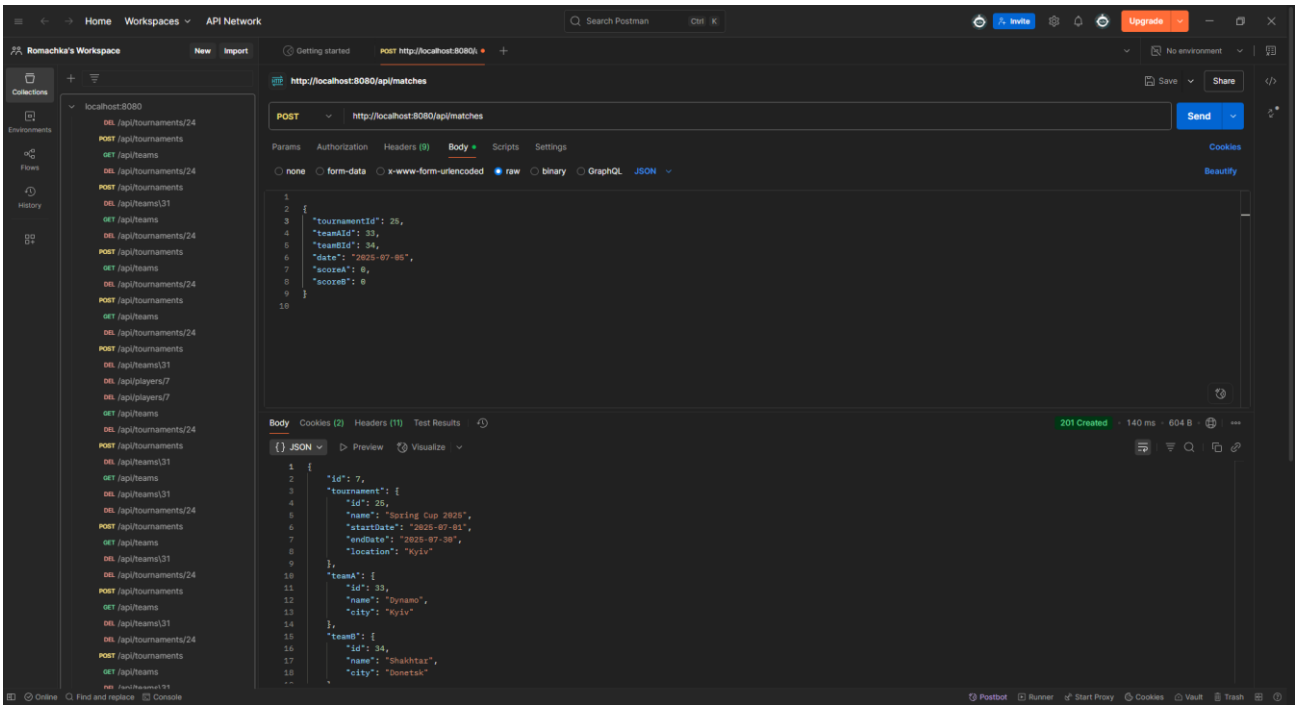

Рисунок 2.13

13 Отримати матчі змагання

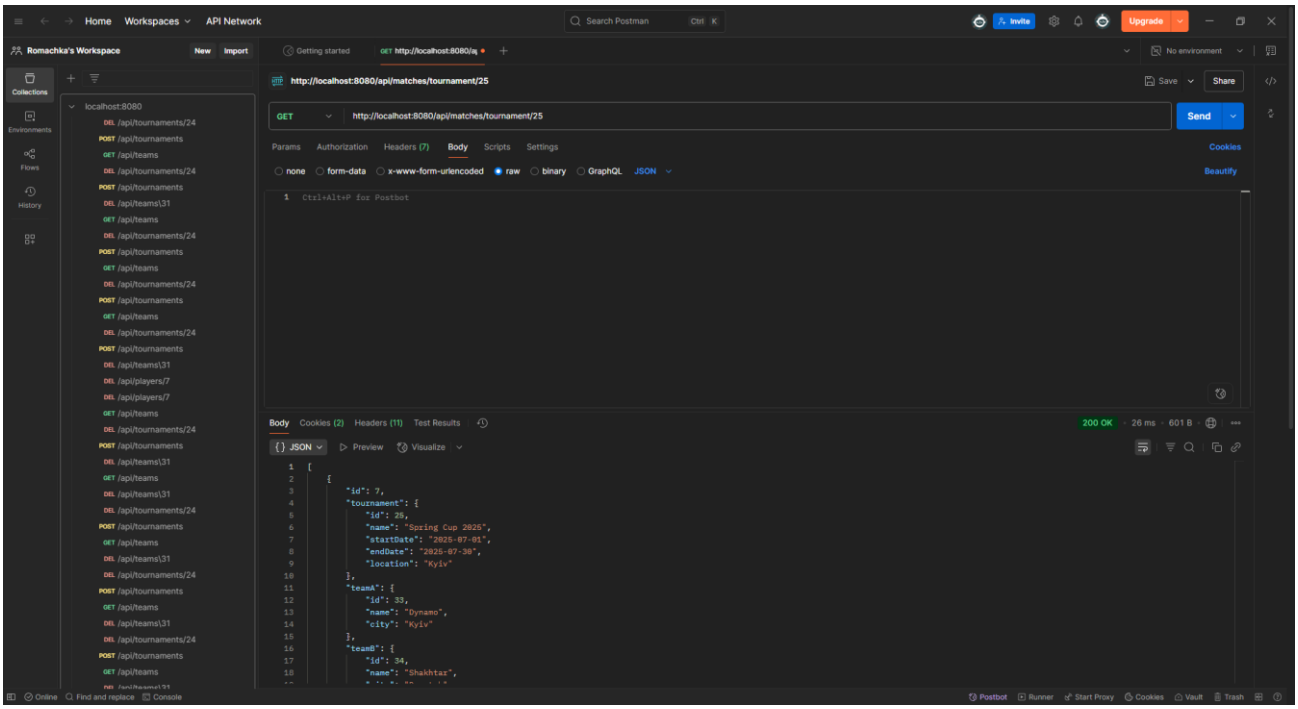

Рисунок 2.14

14 Оновити матч

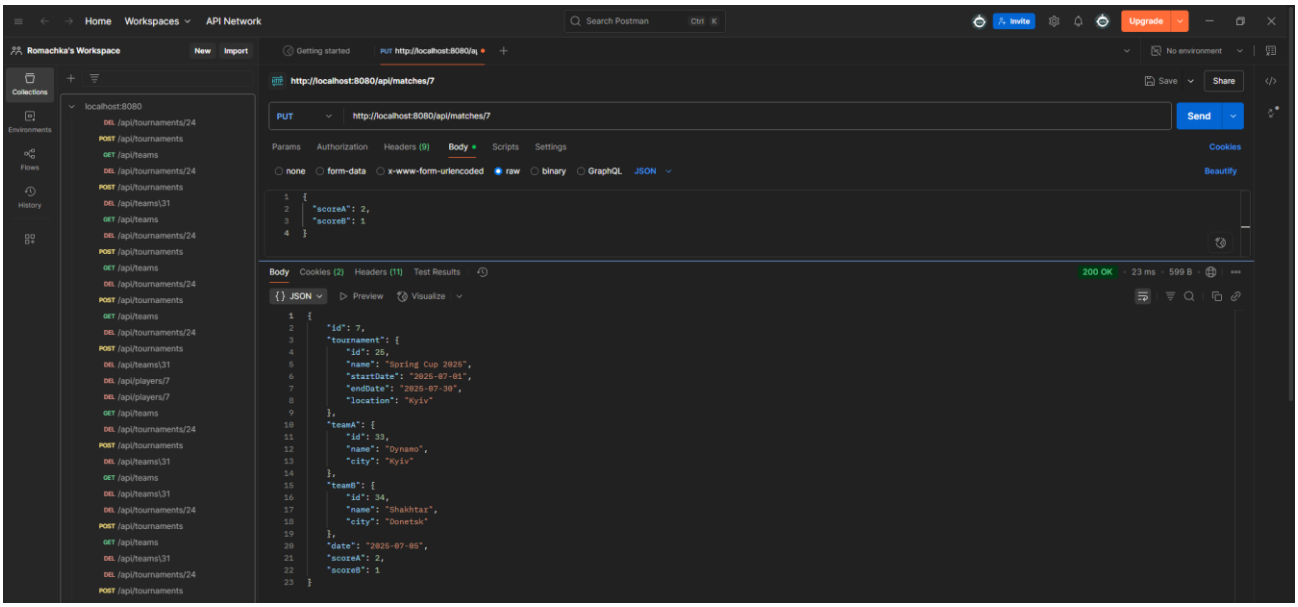

Рисунок 2.15

15 Видалити матч

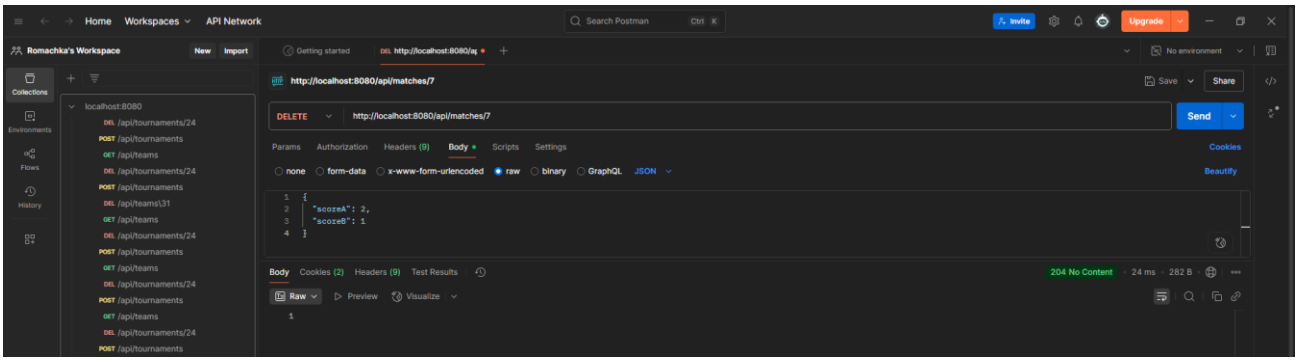

Рисунок 2.16

16 Додати участь команди у змаганні

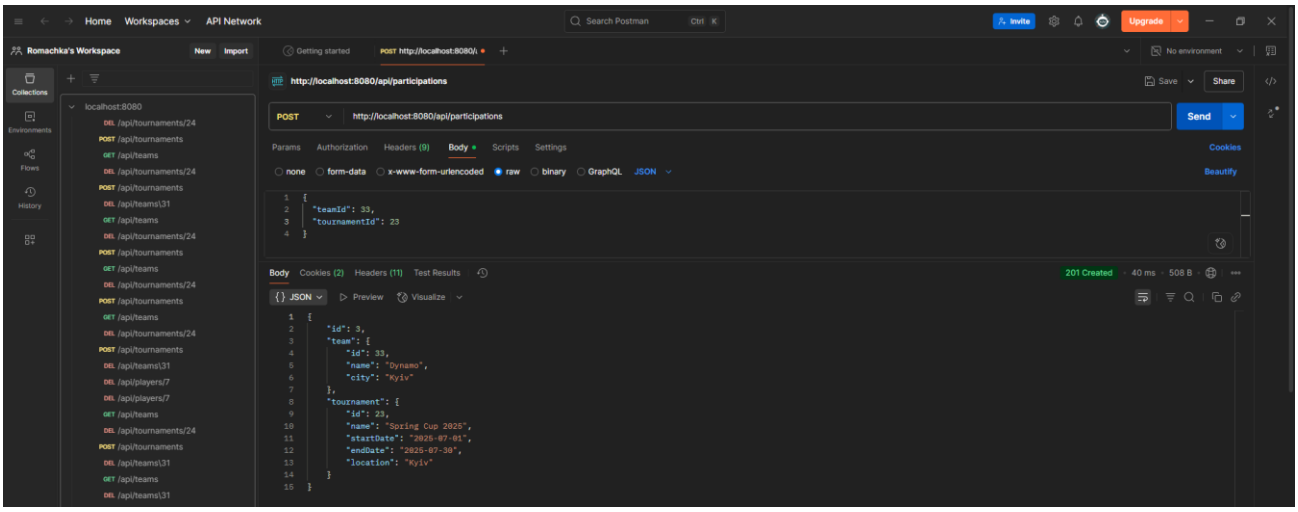

Рисунок 2.17

17 Отримати всі команди змагання

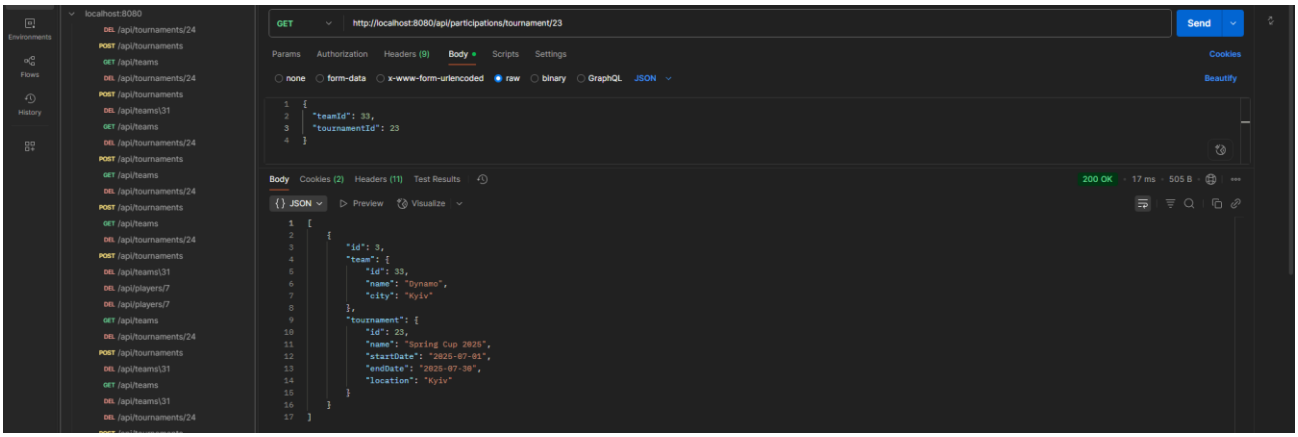

Рисунок 2.18

18 Отримати всі змагання команди

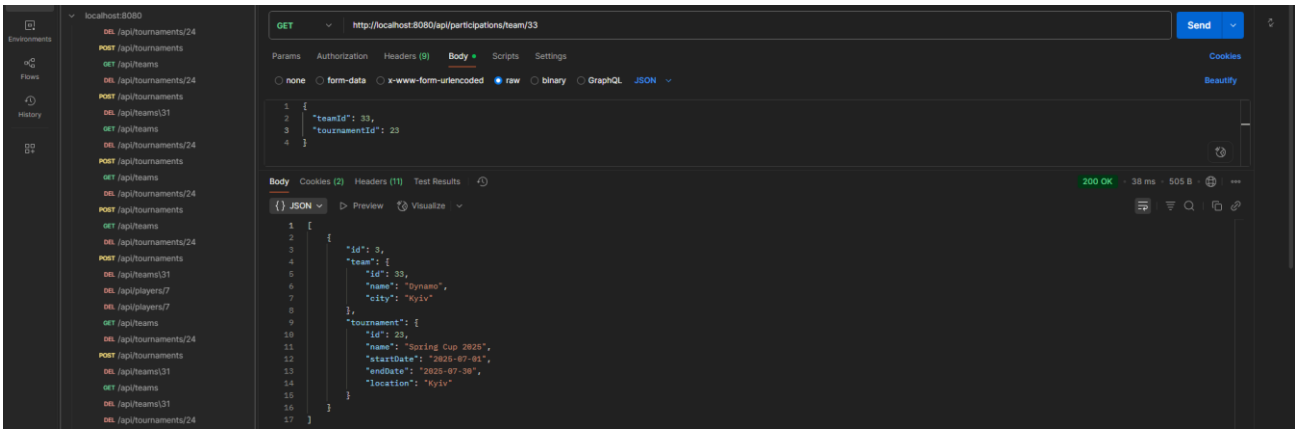

Рисунок 2.19

19 Отримати турнірну таблицю

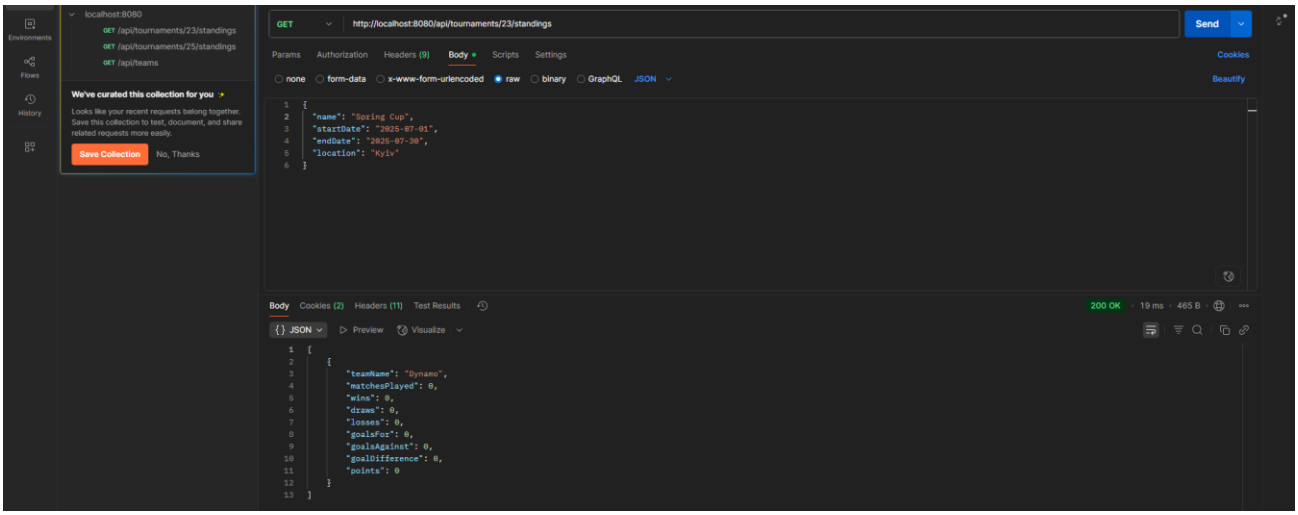

Рисунок 2.20

20 Отримати статистику гравців команди

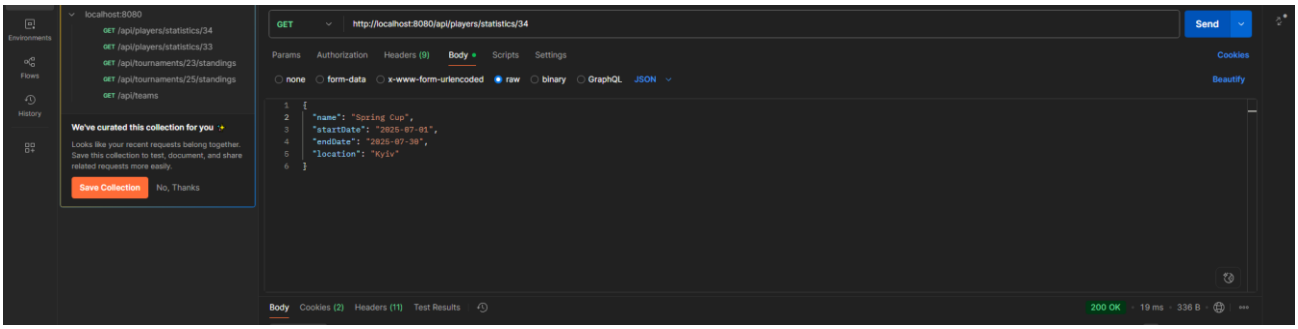

Рисунок 2.21

21 Отримати список матчів команди

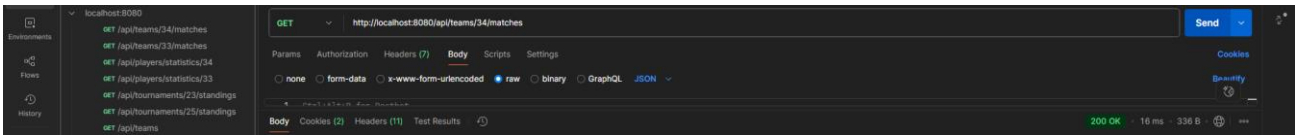

Рисунок 2.22

22 Отримати розклад матчів змагання

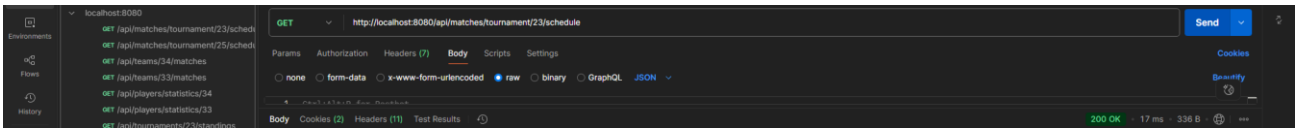

Рисунок 2.23

23 Отримати список переможців

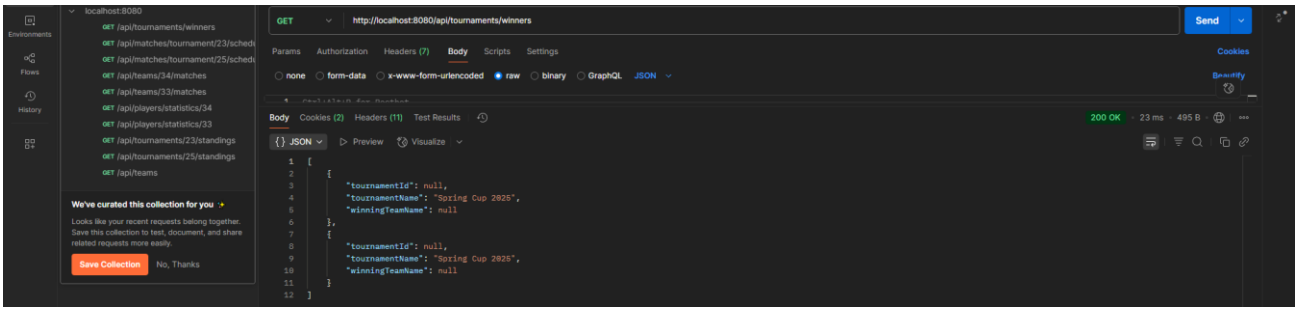

Рисунок 2.24

24 Отримати середній вік гравців команди

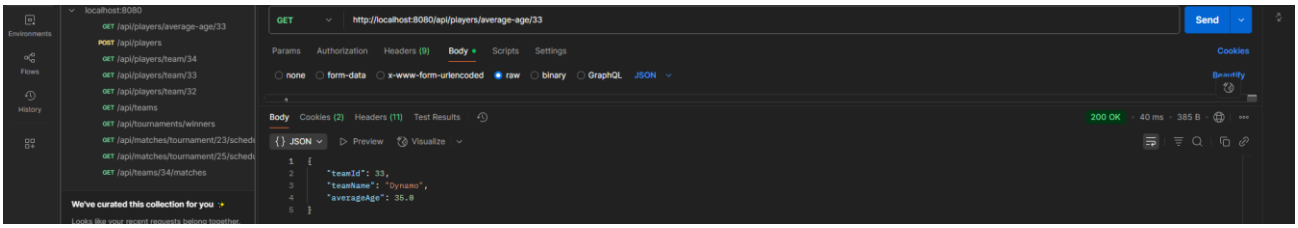

Рисунок 2.25

Висновок

Під час виконання курсової роботи було здійснено ґрунтовну розробку та тестування системи керування спортивними змаганнями на базі Spring Boot. Головною метою стало створення RESTful API, яке автоматизує ключові процеси турніру: реєстрацію команд і гравців, формування календаря матчів, підрахунок очок та надання аналітики у реальному часі.

Ключові результати

- Спроєктовано реляційну БД із шістьма сутностями (Tournament, Team, Player, Match, Participation, User), між якими налаштовано зв’язки та цілісні обмеження.
- Реалізовано 25 REST-запитів, що охоплюють всі CRUD-операції та розширену аналітику: турнірні таблиці, розклади, статистику гравців, список переможців.
- Забезпечено безпеку: аутентифікація через JWT-токени, шифрування паролів Bcrypt, фільтр JwtAuthenticationFilter і роль UserDetailsService.
- Налаштовано MapStruct-мапери та DTO-шар, що відділяє модель БД від зовнішнього API й спрощує подальшу еволюцію схеми.
- Документовано API через інтеграцію зі Swagger/OpenAPI, що полегшує інтеграцію фронтенд-команд.

Курсова робота дозволила поглибити практичні знання у:

- Об’єктно-орієнтованому проєктуванні та побудові багатошарової архітектури (Controller → Service → Repository).
- Розробці серверних застосунків зі Spring Boot 3, Spring Data JPA, Spring Security та HikariCP.
- Інтеграції з PostgreSQL, налаштуванні міграцій, роботі з транзакціями та оптимізації запитів.

Отриманий прототип демонструє, як за допомогою сучасного Java-стека можна швидко розгорнути повнофункціональну систему для спортивних організаторів: зменшити ручну працю, мінімізувати ризик помилок обліку та забезпечити прозору статистику для тренерів, гравців і вболівальників

Список використаних джерел

Spring Boot Reference Documentation. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (звернення: 13.06.2025).

PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/docs/ (звернення: 13.06.2025).

REST API Tutorial. URL: https://restfulapi.net/ (звернення: 13.06.2025).

Додаток

Посилання на Github де розміщений повний код проєкту :

https://github.com/romanlucanetskij/Chupapi/tree/main

Рендер:
https://dashboard.render.com/web/srv-d15vqcndiees73eiljdg/deploys/dep-d15vqcvdiees73eilk2g?r=2025-06-13%4010%3A27%3A03%7E2025-06-13%4010%3A37%3A32