КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

Тема «Система керування спортивними змаганнями»

Виконав:

студент групи АІ-235

Мормиш Микола Валерійович

Керівник:

Годовиченко М.В.

Одеса-2025

ЗМІСТ

[ВСТУП 2](#_Toc200735281)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 4](#_Toc200735282)

[2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6](#_Toc200735283)

[2.1 Проєктування структури даних 7](#_Toc200735284)

[2.2 Опис архітектури застосунку 8](#_Toc200735285)

[3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 9](#_Toc200735286)

[3.1 Реалізація моделей 9](#_Toc200735287)

[3.2 Реалізація репозиторіїв 10](#_Toc200735288)

[3.3 Реалізація сервісів 11](#_Toc200735289)

[3.4 Реалізація контролерів 12](#_Toc200735290)

[3.5 Реалізація аутентифікації 14](#_Toc200735291)

[3.6 Конфігурація безпеки 15](#_Toc200735292)

[4 Тестування та налагодження 18](#_Toc200735293)

[ВИСНОВКИ 20](#_Toc200735294)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 21](#_Toc200735295)

# ВСТУП

У сучасному світі організація спортивних змагань вимагає ефективного керування великими обсягами інформації: реєстрацією команд, контролем складу гравців, фіксацією результатів матчів, формуванням турнірних таблиць та аналітикою. Автоматизація цих процесів дозволяє підвищити прозорість, зменшити ймовірність помилок при обробці даних і значно спростити роботу організаторів. Системи керування спортивними змаганнями стають необхідними для спортивних шкіл, федерацій, клубів та організаторів різноманітних турнірів, оскільки забезпечують централізований доступ до інформації, її швидке оновлення та зручність аналізу.

Метою цієї курсової роботи є розробка веб-застосунку для керування спортивними змаганнями, який дозволяє організовувати турніри, управляти командами, гравцями, фіксувати матчі та результати, а також отримувати різноманітну статистику.

Для досягнення цієї мети необхідно виконати такі завдання:

– проаналізувати предметну область та визначити основні бізнес-процеси;

– спроєктувати структуру даних і архітектуру програмного забезпечення;

– реалізувати REST API для управління змаганнями, командами, гравцями, матчами та статистикою;

– забезпечити зручний доступ до даних та їхню цілісність;

– реалізувати механізми автентифікації та авторизації користувачів;

– протестувати систему та розгорнути її на віддаленому сервері.

Предметна область охоплює процес організації та проведення спортивних змагань, що включає взаємодію декількох основних компонентів. Змагання, або турніри, є подіями, у межах яких відбуваються матчі між командами. Команди виступають учасниками таких змагань і складаються з гравців, кожен з яких належить до певної команди. Матчі – це спортивні ігри, що проходять між двома командами в рамках конкретного турніру; за результатами цих матчів фіксуються рахунки, формується турнірна таблиця та збирається статистика. Окрім цього, важливим елементом є участь, яка відображає зв'язок між командою та турніром й фіксує сам факт участі конкретної команди у певному змаганні.

Для реалізації системи було обрано сучасний технологічний стек, що забезпечує стабільність, безпеку та масштабованість рішення. Java є однією з найпоширеніших платформ для розробки корпоративних веб-застосунків і вирізняється широкою підтримкою інструментів. Використання Spring Boot дозволяє значно спростити налаштування застосунку, забезпечити модульність та швидку розробку веб-додатків[1] із вбудованою підтримкою REST[2]. Для взаємодії з реляційною базою даних застосовується Java Persistence API, що надає стандартні засоби для реалізації Object-Relational Mapping і спрощує роботу із збереженням, отриманням та оновленням даних[3]. Для скорочення обсягу шаблонного коду використовується бібліотека Lombok, яка автоматично генерує гетери, сетери, конструктори та інший службовий код, що підвищує читабельність та підтримуваність проєкту[4]. Реалізація безпечної автентифікації та авторизації користувачів здійснюється за допомогою JSON Web Token, що дозволяє організувати гнучкий контроль доступу до ресурсів на основі токенів[5]. Використання вказаних технологій дозволяє створити сучасний, безпечний, масштабований та зручний у підтримці веб-застосунок для ефективного керування спортивними змаганнями.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Завдання полягає у розробці інформаційної системи, яка дозволяє організовувати спортивні змагання, управляти командами та гравцями, фіксувати результати матчів, а також формувати статистику та турнірні таблиці. Система має забезпечити можливість роботи для різних категорій користувачів: організаторів змагань, тренерів та адміністраторів команд, а також надати інтерфейси для отримання інформації про перебіг турніру та результати. Нижче приведений опис основних бізнес-процесів:

а) організація змагань, організатор створює новий турнір, задає його параметри (назва, дати, місце проведення). До змагання додаються команди-учасниці;

б) управління командами, система дозволяє створювати, редагувати або видаляти команди. Кожна команда має назву, місто, список гравців;

в) управління гравцями, для кожної команди ведеться перелік гравців із вказанням їхніх особистих даних (ПІБ, дата народження, позиція). Гравець може бути учасником лише однієї команди;

г) проведення та фіксація матчів, в рамках турніру організатор створює розклад матчів, призначає команди-учасниці, фіксує дату проведення. По завершенні матчу вноситься рахунок, на підставі якого оновлюються турнірна таблиця та статистика;  
ґ) аналіз та статистика, система автоматично формує турнірні таблиці, статистику команд і гравців, дозволяє переглядати розклад і результати матчів, визначати переможців.

У розроблюваній системі всі користувачі мають однакові права доступу до функціоналу після проходження автентифікації. Ролі «організатор», «тренер», «глядач» у цьому контексті не виділяються програмно, а використовуються лише для опису можливих сценаріїв взаємодії зі системою. Таким чином, будь-який авторизований користувач може створювати турніри, додавати або редагувати команди й гравців, формувати розклад матчів, фіксувати результати, переглядати турнірну таблицю та статистику. Діаграма варіантів використання зображено на рисунку 1.1.

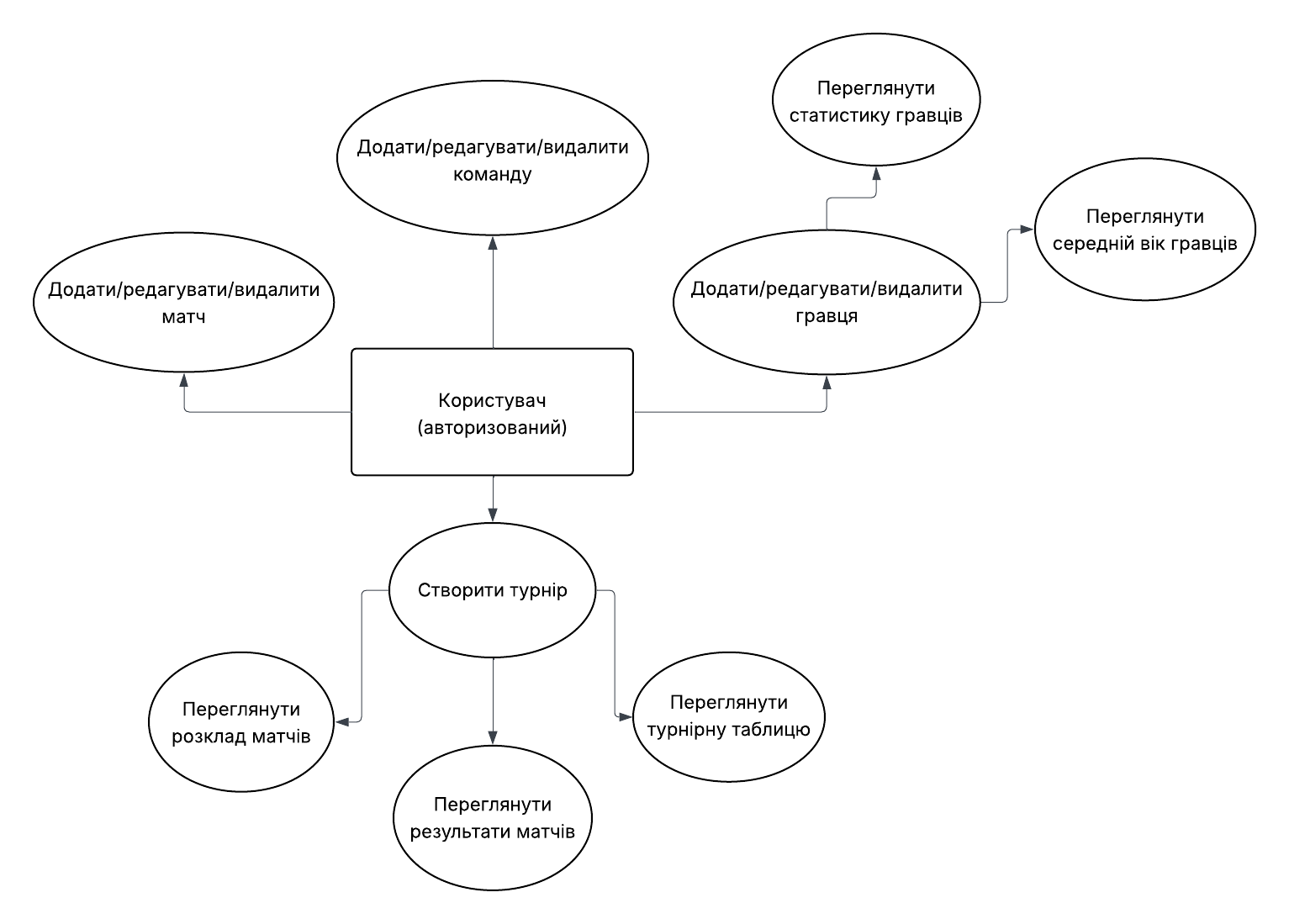


Рисунок 1.1 – UML-діаграма варіантів використання

# 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування програмного забезпечення є одним із ключових етапів розробки інформаційної системи, оскільки саме на цьому етапі визначаються основні компоненти, їхні взаємозв’язки і структура даних. Чітке та продумане проєктування дозволяє закласти надійну основу для подальшої реалізації, масштабування та підтримки програмного продукту. У межах даної системи особливу увагу приділено моделюванню предметної області, а також визначенню взаємодії між різними сутностями, що відображають реальні процеси організації й проведення спортивних змагань.

Для досягнення гнучкості й розширюваності застосунку обрано багаторівневу архітектуру, яка передбачає розподіл відповідальності між окремими шарами: контролерів, сервісів і репозиторіїв. Це дозволяє забезпечити чітке розділення логіки роботи з даними, бізнес-логіки та обробки клієнтських запитів, що, у свою чергу, підвищує підтримуваність і тестованість коду. Актуальним є і використання сучасних підходів до проєктування REST API, що забезпечує простий та уніфікований інтерфейс для взаємодії з клієнтською частиною або зовнішніми системами.

Під час проєктування структури даних враховано особливості предметної області: команда бере участь у багатьох змаганнях, але кожний гравець належить лише одній команді; матч проводиться між двома командами в рамках певного турніру; участь команди в турнірі фіксується окремою сутністю Participation. Реалізація зв’язків між сутностями за допомогою JPA-анотацій дозволяє зберігати цілісність даних та спрощує роботу з реляційною базою даних. Окремо слід зазначити, що в системі передбачено зберігання детальної статистики гравців і команд, що відкриває можливості для розширення функціоналу у майбутньому (наприклад, автоматичне формування рейтингів тощо).

Таким чином, продумане проєктування забезпечує не лише відповідність вимогам замовника, а й створює передумови для подальшого розвитку й масштабування системи.

## 2.1 Проєктування структури даних

У даній системі виділяються такі основні сутності: турнір, команда, гравець, матч та участь. Турнір – це змагання, яке має унікальний ідентифікатор, назву, дати початку й завершення, а також місце проведення. Команда бере участь у різних змаганнях і складається з гравців; для кожної команди фіксуються її назва та місто. Гравець входить до складу лише однієї команди і містить інформацію про повне ім’я, дату народження та ігрову позицію. Матч проводиться у межах певного турніру між двома командами; для кожного матчу зберігається рахунок і дата проведення. Окремо виділяється сутність Participation, яка фіксує участь певної команди у конкретному турнірі та служить проміжною ланкою для реалізації зв’язку між командами і змаганнями. Все це зображено на рисунку 1.2 для кращої візуалізації структури проєкту та зв’язків між елементами.

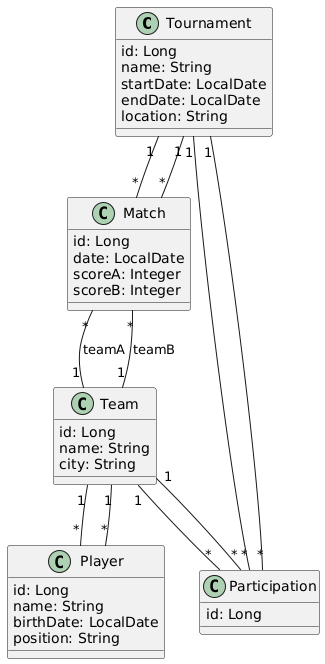


Рисунок 1.2 – Діаграма класів проєкту

## 2.2 Опис архітектури застосунку

Архітектура застосунку реалізована на основі принципів багаторівневого підходу, який дозволяє чітко розділити відповідальність між окремими компонентами системи та забезпечити гнучкість, масштабованість і простоту тестування. Основними рівнями архітектури є Controller, Service та Repository[6].

Рівень Controller відповідає за прийом і обробку HTTP-запитів від клієнтів. Контролери отримують дані ззовні, викликають відповідні сервіси та повертають відповіді у форматі JSON. На цьому шарі здійснюється валідація введених даних і формування HTTP-відповідей.

Рівень Service містить бізнес-логіку застосунку. Сервіси реалізують основні операції над сутностями, координують взаємодію між різними компонентами, обробляють запити від контролерів, виконують додаткову валідацію, калькуляцію чи обробку даних перед збереженням у базу або віддачею клієнту.

Рівень Repository реалізує взаємодію з базою даних. Репозиторії є інтерфейсами, які розширюють стандартні можливості JPA, і дозволяють виконувати CRUD-операції, а також складніші вибірки, через запити або специфікації.

Залежності між компонентами побудовані таким чином, що контролери взаємодіють лише із сервісами, сервіси – з репозиторіями, а репозиторії – з базою даних. Це дозволяє мінімізувати зв'язаність коду та спрощує подальшу підтримку і тестування окремих частин системи.

# 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Реалізація програмного продукту ґрунтується на сучасних підходах до розробки веб-застосунків на основі Spring Boot, що дозволяє забезпечити модульність, розширюваність та простоту підтримки коду. Всі основні компоненти системи реалізовано у вигляді окремих пакетів для моделей, entity-класів та DTO, репозиторіїв, сервісів і контролерів. Такий підхід сприяє дотриманню принципів SOLID та забезпечує чисту архітектуру застосунку.

## 3.1 Реалізація моделей

Для відображення предметної області розроблені entity-класи для основних сутностей: Tournament, Team, Player, Match, Participation. Всі класи містять необхідні поля, анотації JPA (@Entity, @Id, @ManyToOne, @OneToMany, @JoinColumn тощо) для коректного відображення зв’язків у реляційній базі даних. Для спрощення і скорочення коду активно використовується бібліотека Lombok (@Data, @NoArgsConstructor, @AllArgsConstructor, @Builder), яка автоматично генерує гетери, сетери, конструктори та інші службові методи. Надано приклад реалізації моделі у вигляді entity-класу у проєкті:

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Player {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private LocalDate birthDate;

private String position;

private Integer stats;

@ManyToOne

@JoinColumn(name = "team\_id")

@JsonIgnore

private Team team;

@JsonProperty("teamId")

public Long getTeamId() {

return team != null ? team.getId() : null;

}

}

Клас Player є JPA-сутністю, яка відповідає гравцю у системі та відображається у відповідній таблиці бази даних. Він містить основні поля: ідентифікатор, ім’я, дату народження, позицію, статистику та зв’язок з командою через зовнішній ключ. Анотації Lombok забезпечують автоматичне генерування гетерів, сетерів і конструкторів, а JPA-нотації визначають зв’язки між сутностями та правила збереження у базі. При серіалізації у JSON команда не включається повністю, натомість окремо передається лише її ідентифікатор, що робить API-відповіді компактними та уникнення рекурсії.

## 3.2 Реалізація репозиторіїв

Доступ до даних реалізовано за допомогою інтерфейсів-репозиторіїв, що розширюють JpaRepository та CrudRepository. Для кожної основної сутності створено окремий репозиторій, що забезпечує виконання стандартних CRUD-операцій (створення, читання, оновлення, видалення). У разі потреби для складніших вибірок або кастомних фільтрів застосовуються власні методи з використанням @Query або методів за іменем. Такий підхід дозволяє легко масштабувати систему та швидко додавати новий функціонал. Надано приклад реалізації репозиторію у проєкті:

public interface ParticipationRepository extends JpaRepository<Participation, Long> {

@Query("SELECT p.team FROM Participation p WHERE p.tournament.id = :tournamentId")

List<Team> findTeamsByTournamentId(Long tournamentId);

@Query("SELECT p.tournament FROM Participation p WHERE p.team.id = :teamId")

List<Tournament> findTournamentsByTeamId(Long teamId);

}

Інтерфейс ParticipationRepository є репозиторієм для роботи з сутністю Participation і розширює JpaRepository, що дозволяє виконувати стандартні CRUD-операції над таблицею участей команд у турнірах. Окрім базових методів, тут визначено два додаткових запити: перший (findTeamsByTournamentId) повертає список команд, які беруть участь у заданому турнірі, а другий (findTournamentsByTeamId) дозволяє отримати всі турніри, в яких брала участь певна команда. Обидва методи реалізовані через JPQL-запити з використанням анотації @Query, що забезпечує гнучкий і ефективний пошук потрібних даних у базі.

## 3.3 Реалізація сервісів

Бізнес-логіка системи зосереджена у сервісах, які взаємодіють із репозиторіями та виконують необхідні обчислення, перевірки та координацію дій між різними сутностями. Сервіси відповідають за виконання таких операцій, як створення турнірів, додавання команд і гравців, фіксація результатів матчів, формування турнірної таблиці, обробка статистики тощо. Завдяки чіткій структурі та розмежуванню відповідальностей, сервіси легко тестуються та підтримуються. Для ін’єкції залежностей використовується анотація @Service, а також механізми Spring Dependency Injection. Надано приклад реалізації сервісу у проєкті:

public class TeamService {

private final TeamRepository teamRepository;

private final MatchRepository matchRepository;

@Autowired

public TeamService(TeamRepository teamRepository,

MatchRepository matchRepository) {

this.teamRepository = teamRepository;

this.matchRepository = matchRepository;

}

public Team addTeam(Team team) {

return teamRepository.save(team);

}

public List<Team> getAllTeams() {

return teamRepository.findAll();

}

public Team updateTeam(Long id, Team updatedTeam) {

Optional<Team> optionalTeam = teamRepository.findById(id);

if (optionalTeam.isPresent()) {

Team team = optionalTeam.get();

team.setName(updatedTeam.getName());

team.setCity(updatedTeam.getCity());

return teamRepository.save(team);

} else {

throw new RuntimeException("Команда не знайдена: " + id);

}

}

public boolean deleteTeam(Long id) {

if (teamRepository.existsById(id)) {

teamRepository.deleteById(id);

return true;

} else {

return false;

}

}

public List<Match> getMatchesByTeam(Long teamId) {

return matchRepository.findByTeamId(teamId);

}

public Double getAverageAge(Long teamId) {

return teamRepository.findAveragePlayerAgeByTeamId(teamId);

}

}

Клас TeamService є сервісом, який реалізує бізнес-логіку для роботи з командами у системі. Він взаємодіє з репозиторіями команд і матчів для виконання основних операцій: додавання нової команди, отримання списку всіх команд, оновлення даних певної команди, видалення команди (з перевіркою наявності), а також отримання списку матчів для заданої команди і розрахунку середнього віку гравців у команді. Конструктор класу використовує анотацію @Autowired для впровадження залежностей, а основна логіка організована у вигляді окремих методів, що забезпечує чистоту коду, зручність тестування та повторного використання.

## 3.4 Реалізація контролерів

У системі реалізовані REST-контролери для обробки HTTP-запитів до основних сутностей: Tournament, Team, Player, Match, Participation. Контролери відповідають за прийом та валідацію вхідних даних, виклик відповідних сервісів та формування структурованої відповіді у форматі JSON. Для кожної сутності передбачено базові CRUD-операції, такі як створення, отримання, оновлення, видалення. Контролери використовують анотації Spring (@RestController, @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping), що забезпечує зручну маршрутизацію запитів. Завдяки розмежуванню відповідальностей контролери не містять бізнес-логіки, а лише делегують обробку даних відповідним сервісам. Надано приклад реалізації контролеру у проєкті:

@RequestMapping("/participations")

public class ParticipationController {

private final ParticipationService participationService;

@Autowired

public ParticipationController(ParticipationService participationService) {

this.participationService = participationService;

}

@PostMapping

public ResponseEntity<Participation> addParticipation(@RequestBody Participation participation) {

Participation createdParticipation = participationService.addParticipation(participation);

return ResponseEntity.status(HttpStatus.CREATED).body(createdParticipation);

}

@GetMapping("/tournament/{tournamentId}/teams")

public List<Team> getTeamsByTournament(@PathVariable Long tournamentId) {

return participationService.getTeamsByTournament(tournamentId);

}

@GetMapping("/team/{teamId}/tournaments")

public ResponseEntity<List<Tournament>> getTournamentsByTeam(@PathVariable Long teamId) {

List<Tournament> tournaments = participationService.getTournamentsByTeam(teamId);

if (tournaments != null && !tournaments.isEmpty()) {

return ResponseEntity.ok(tournaments);

} else {

return ResponseEntity.notFound().build();

}

}

}

Клас ParticipationController є REST-контролером, який відповідає за обробку HTTP-запитів, пов’язаних з участю команд у турнірах. Він містить три основних методи: додавання нової участі (POST-запит), отримання списку команд, що беруть участь у певному турнірі (GET-запит), та отримання всіх турнірів, у яких брала участь певна команда (GET-запит). Контролер делегує виконання основної бізнес-логіки сервісу ParticipationService, а також формує коректні HTTP-відповіді: у разі успішного створення повертає статус 201 (Created), при отриманні списків – статус 200 (OK) або 404 (Not Found), якщо даних немає. Така структура забезпечує зручний і сучасний API для управління участю команд у турнірах.

## 3.5 Реалізація аутентифікації

У проєкті реалізовано сучасну систему аутентифікації на основі Spring Security та JSON Web Token, що включає реєстрацію, логін і захист усіх приватних REST-ендпоінтів. Реєстрація здійснюється через ендпоінт “/api/auth/signup”. Користувач надсилає логін, e-mail і пароль. Система перевіряє унікальність логіну та пошти, після чого пароль безпечно хешується за допомогою BCrypt і зберігається в базі даних. Новий користувач отримує роль `ROLE\_USER` за замовчуванням. Аутентифікація виконується через ендпоінт “/api/auth/signin”. Користувач надсилає логін і пароль. Система перевіряє їх валідність і, у разі успіху, генерує JWT-токен. Токен повертається клієнту у відповіді.

Для захисту приватних ендпоінтів застосовується JWT-аутентифікація. Клієнт передає токен у заголовку Authorization. Фільтр JwtAuthenticationFilter перехоплює кожен запит, перевіряє валідність токена, дістає ім'я користувача і підвантажує його деталі через CustomUserDetailsService. У разі успіху створюється аутентифікований контекст у Spring Security. Якщо токен невалідний або відсутній – повертається статус 401.

## 3.6 Конфігурація безпеки

У проєкті використовується сучасний підхід до захисту REST API на основі Spring Security та JWT[7]. Аутентифікація та авторизація реалізовані через набір ключових компонентів, які забезпечують безпечний доступ до приватних ресурсів системи. Для реалізації цієї архітектури задіяні наступні основні компоненти:

а) AuthController – контролер для реєстрації та логіну користувачів, який генерує JWT при успішній аутентифікації;

б) JwtTokenProvider – відповідає за генерацію, парсинг і валідацію JWT-токенів;

в) JwtAuthenticationFilter – фільтр, який обробляє кожен запит і перевіряє токен у заголовку Authorization;

г) CustomUserDetailsService – підвантажує інформацію про користувача та його ролі для Spring Security;

ґ) SecurityConfig – налаштовує політики доступу, підключає всі необхідні фільтри й забезпечує інтеграцію компонентів безпеки між собою.

Нижче наведено частину ключового класу конфігурації безпеки – SecurityConfig:

@Autowired

private CustomUserDetailsService userDetailsService;

@Autowired

private JwtAuthenticationEntryPoint unauthorizedHandler;

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

У першому блоці розташовані ін’єкція залежностей і створення необхідних бінів. У ньому підключаються такі основні сервіси як CustomUserDetailsService для отримання користувача з бази даних, а також JwtAuthenticationEntryPoint, який відповідає за обробку неавторизованих запитів. Окремо створюється бін власного JWT-фільтра, який буде шукати та перевіряти токени у кожному запиті, і бін енкодера паролів на основі алгоритму BCrypt. Завдяки цьому паролі користувачів зберігаються у надійному хешованому вигляді, а всі механізми взаємодії з користувачами та токенами працюють централізовано та безпечно. Далі приведено провайдер і менеджер аутентифікації:

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService);

authProvider.setPasswordEncoder(passwordEncoder());

return authProvider;

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {

return authConfig.getAuthenticationManager();

}

Тут показано налаштування провайдера та менеджера аутентифікації. Провайдер аутентифікації DaoAuthenticationProvider використовує сервіс користувачів і енкодер паролів для перевірки облікових даних під час логіну, інтегруючись зі Spring Security. Менеджер аутентифікації AuthenticationManager виступає центральною точкою для запуску процесу аутентифікації, зокрема використовується у контролерах для логіну користувача. Це дає змогу гнучко контролювати процес перевірки логінів і паролів, а також легко масштабувати систему безпеки при потребі. Далі показано безпосередньо основну конфігурація безпеки:

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

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

.exceptionHandling(exception -> exception.authenticationEntryPoint(unauthorizedHandler))

.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

.authorizeHttpRequests(auth -> auth

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

.requestMatchers("/api/public/\*\*").permitAll()

.anyRequest().authenticated()

);

http.authenticationProvider(authenticationProvider());

http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

return http.build();

}

У цьому блоці розташована основна конфігурація безпеки, яка визначається у методі securityFilterChain. Тут вимикається CSRF-захист, оскільки REST API із stateless токенами не потребує цього механізму. Встановлюється спеціальний обробник для помилок аутентифікації, щоб для неавторизованих запитів API повертав коректний статус і зрозуміле повідомлення. Сесії у застосунку повністю вимикаються: сервер не зберігає жодної інформації про користувача між запитами, і вся ідентифікація виконується через JWT. Вільний доступ дозволяється лише до ендпоінтів для реєстрації, логіну та публічних ресурсів, тоді як усі інші запити доступні тільки після успішної аутентифікації. До ланцюжка фільтрів Spring додається кастомний JWT-фільтр, який перевіряє кожен запит на наявність і валідність токена ще до того, як Spring Security застосує стандартні механізми аутентифікації. Така конфігурація гарантує, що усі приватні частини API надійно захищені і доступ до них можливий лише з валідним JWT, що робить систему сучасною, безпечною та масштабованою.

# 4 Тестування та налагодження

Для перевірки працездатності основних компонентів системи у проєкті були розроблені модульні тести із використанням бібліотек JUnit 5 та Mockito. Тестування охоплює сервіси аутентифікації, роботи з користувачами, а також генерацію і валідацію JWT токенів. Зокрема, у класі CustomUserDetailsServiceTest перевіряється завантаження користувача по імені та реакція на відсутнього користувача, а у JwtTokenProviderTest – генерація і валідація JWT токенів, а також обробка невалідних токенів. Клас SportsManagementApplicationTests забезпечує перевірку базового запуску додатку та коректного підняття контексту Spring.

Для ручного тестування REST API використовувався Postman. Основні сценарії перевірки включали аутентифікацію користувача, операції з командами, турнірами та матчами через відповідні ендпоінти API, такі як /api/auth/login, /teams/id, /tournaments/id, /matches/id. Усі запити надсилалися з відповідними HTTP-методами, що дозволило перевірити коректність обробки як валідних, так і невалідних даних. На рисунку 4.1 показана спроба увійти в створений обліковий запис.

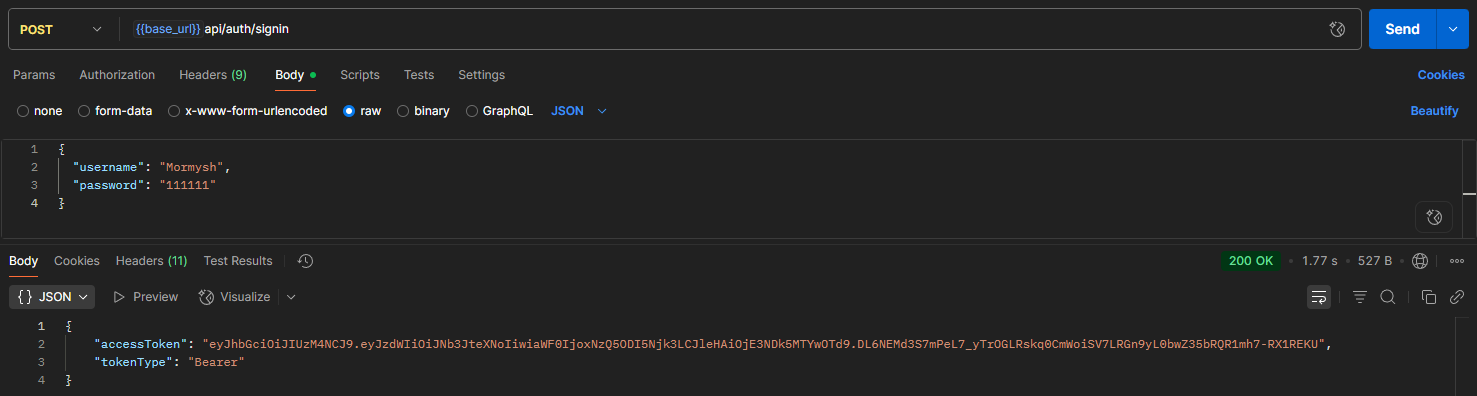


Рисунок 4.1 – Успішний вхід у програмі Postman

На рисунку 4.2 показано спробу отримати всі матчі турніру 1, щоб запит пройшов успішно використовується отриманий токен.

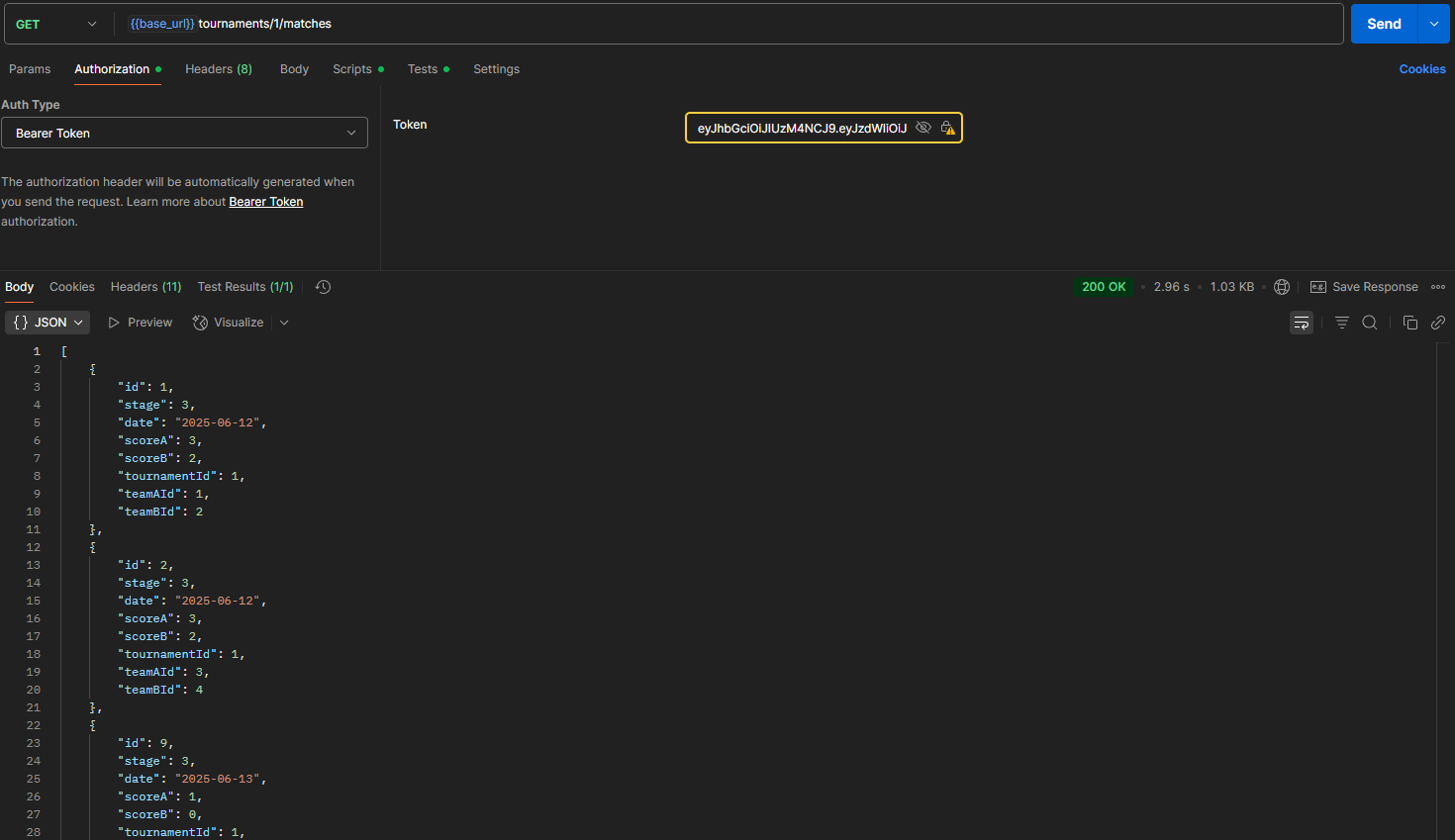


Рисунок 4.2 – Успішний запит у програмі Postman

Проте, якщо намагатися отримати матчі без токену, як на рисунку 4.3, повернеться помилка 401.

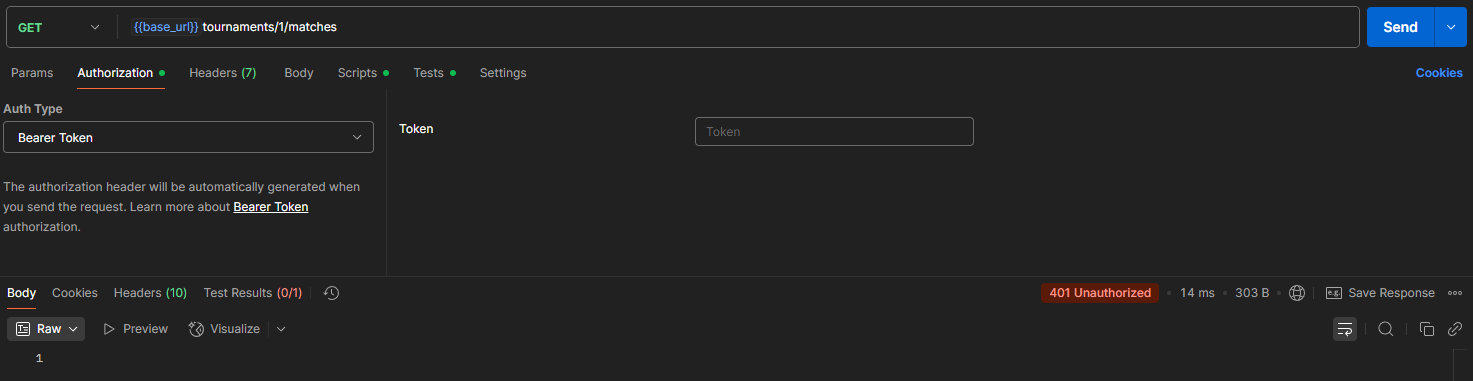


Рисунок 4.3 – Неуспішний запит у програмі Postman

Результати тестування показали, що юніт-тести виконуються успішно та базова логіка сервісів працює коректно. При тестуванні з Postman основні ендпоінти повертають очікувані відповіді та статус-коди. Операції створення, оновлення, видалення та отримання сутностей виконуються відповідно до функціональних вимог.

# ВИСНОВКИ

У процесі виконання курсової роботи було реалізовано веб-застосунок, який відповідає сучасним вимогам до автоматизації організації спортивних змагань. Розроблена система дозволяє ефективно вирішувати ключові завдання предметної області, а саме: реєструвати команди та гравців, організовувати й адмініструвати турніри, фіксувати результати матчів, формувати турнірні таблиці і отримувати статистичні дані. Всі бізнес-процеси, що були визначені на етапі аналізу предметної області, отримали своє відображення у структурі даних і логіці роботи застосунку.

Було спроєктовано і реалізовано багаторівневу структуру, яка забезпечує розділення відповідальностей, спрощує підтримку і подальший розвиток системи. В основі взаємодії з клієнтами лежить REST API, що забезпечує стандартизований і зручний доступ до функціоналу застосунку. Для збереження та обробки даних використано сучасний підхід ORM та JPA, що дозволяє легко працювати з реляційною базою даних.

Було проведене модульне та інтеграційне тестування основних компонентів системи, а також ручна перевірка роботи API за допомогою Postman, дозволили виявити і усунути низку помилок на ранніх етапах розробки. Одержані результати підтверджують коректність і надійність реалізованої функціональності, відповідність системи вимогам предметної області та заявленим у вступі цілям і завданням.

Розроблений програмний продукт є гнучкою основою для подальшого розвитку. У майбутньому систему можна розширити новими функціями, інтегрувати з іншими сервісами, додати розширені аналітичні можливості та додаткові механізми безпеки. Таким чином, мета курсової роботи досягнута, а розроблена система повністю відповідає сучасним тенденціям автоматизації управління спортивними змаганнями.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 20 Найкращих практик Spring Boot : технічний блог. URL: <https://www.javaguides.net/2024/05/java-spring-boot-best-practices.html> (дата звернення: 13.06.2025).
2. REST API Найкращі практики : інформаційний ресурс. URL: https://restfulapi.net/rest-api-best-practices/ (дата звернення: 13.06.2025).
3. Введення до Java Persistence API (Release 7) : офіційна документація Oracle. URL: https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm (дата звернення: 13.06.2025).
4. Overview (Lombok) – Project Lombok : офіційна документація. URL: https://projectlombok.org/api/ (дата звернення: 13.06.2025).
5. JSON Web Token Введення : офіційний сайт JWT. URL: https://jwt.io/introduction (дата звернення: 13.06.2025).
6. Spring Guides : офіційна документація. URL: https://spring.io/guides (дата звернення: 13.06.2025).
7. Spring Security Reference Documentation : офіційна документація. URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 13.06.2025).