«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
«КПІ ім. І. Сікорського»
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

КУРСОВА РОБОТА

з дисципліни «Системи баз даних»

на тему *«Онлайн турнірні таблиці»*

Студента III курсу групи КА-97
Спеціальність 122 *Комп'ютерні науки*
Тригуб А. П.

Керівник: Афанасьєва І. В.

Національна оцінка: _____
Кількість балів: _____
Оцінка ECTS: _____

Члени комісії _____ _____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

_____ _____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

_____ _____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2021

ЗМІСТ

АНОТАЦІЯ

Даний документ є пояснювальною запискою до курсової роботи з дисципліни «Системи баз даних» на тему «Онлайн турнірні таблиці». Метою є створення бази даних та бізнес-логіки продукту в цілому. У даній роботі застосовуються базові завдання роботи з базами даних. Пояснювальна записка складається зі вступу, основної частини та висновків.

ANNOTATION

This document is an explanatory note to the course work on the subject “Database systems” on “Online tournament tables”. The course work essence is creating database and business-logic of product. In this work the basic tasks of working with databases are implemented. Explanatory note divided into introduction, main part and conclusions.

ВСТУП

В усі епохи існування суспільства важливу роль в житті людей відіграє дозвілля. Одним із видів відпочинку є перегляд різного роду змагань, турнірів, матчів тощо. Це може бути футбольний матч, хокей, олімпійські ігри, бокс або ж взагалі власноруч організований розважальний або спортивний захід. Основною метою даного курсового проекту є створення інструменту, що дозволив би автоматизувати процес визначення переможців змагань за допомогою турнірною сітки.

Турнірна сітка являє собою простий варіант візуалізації парних матчів між командами з багатьох або одного учасника. Тобто даний інструмент буде зручний саме для малих власноруч організованих змагань або турнірів, наприклад, по шахам або волейболу тощо.

Завданням курсового проекту є проектування бази даних, що змогла б організовано зберігати інформацію про команди, турніри та матчі. А також створення та налаштування бекенду для роботи зі створеною базою даних у зручному для користувача вигляді, а саме API-запитів (Application Programming Interface).

Для реалізації даної курсової роботи були обрані наступні інструменти: реляційна база даних MySQL 5.7, інструмент візуального проектування баз даних MySQL Workbench 8.0 CE, JavaScript оточення для створення серверу застосунку NodeJS, а також додатково бібліотеки даної платформи для налаштування WebAPI, а саме – бібліотека для організації запитів до бази даних Mysql, ORM-бібліотека (Objective-Relational Mapping) Sequelize для швидкого та зручного налаштування моделі бази даних з подальшим зручним інтерфейсом для зв’язку з базою даних, JWT (JSON Web Token, де JSON -

JavaScript Object Notation) для створення токенів авторизованим користувачам та BCrypt для шифрування користувацьких паролів.

Також додатково для перевірки роботоспроможності реалізованого програмного продукту використовувався додаток Postman, що дозволяє зручно створювати та організовувати колекції REST-запитів (Representational State Transfer).

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

Кожна турнірна таблиця повинна складатись з декількох основних структурних елементів:

- основна інформація про захід – назва турніру, дата початку та закінчення проведення;
- інформація про учасників турніру, тобто перелік команд; важливо, що їх кількість повинна бути степенем двійки, адже всі матчі будуть попарними, а тому потрібно 2^n команд;
- інформація про матчі – команди, що приймають участь в ньому та дата проведення, а також результати проведення, тобто остаточний рахунок та переможець.

Основними учасниками бізнес-процесу застосунку будуть гості, зареєстровані користувачі та адміністратори. Кожен тип користувача повинен мати свій обмежений набір дій, що наведений далі у вигляді функціональний вимог до продукту.

Функціональні вимоги реалізованої курсової роботи поділені за категоріями користувачів та мають наступний вигляд:

1. Як незареєстрований користувач (гість) я хочу мати можливість
   1.1. Зареєструватись у системі за допомогою заповнення наступної інформації – юзернейму (набір латинських символів з можливим використанням нижнього підкреслення всередині основного тексту), паролю (будь-який набір символів), імені та прізвища (набір латинських букв)
   1.2. Увійти до особистого кабінету за допомогою коректного юзернейму та паролю
   1.3. Переглядати список активних команд та їх учасників

1.4. Переглядати список активних турнірів, їх учасників та матчі

2. Як зареєстрований та авторизований користувач я хочу мати можливість
   2.1. Створювати власні команди з унікальними назвами та учасниками, що є зареєстрованими користувачами
   2.2. Видаляти власні команди, якщо вони не є учасниками турнірів
   2.3. Переглядати власні запити на створення команд (неактивні команди)
   2.4. Переглядати список власних активних команд
   2.5. Переглядати основну інформацію про команду, навіть, якщо вона поки не є активною
   2.6. Створювати власні турніри з активних команд (навіть, якщо команди не належать даному користувачу)
   2.7. Видаляти власні турніри
   2.8. Переглядати власні запити на створення турнірів (неактивні турніри)
   2.9. Переглядати список власних активних турнірів
   2.10. Переглядати основну інформацію про турнір (учасників та матчі), навіть, якщо він поки не є активним
   2.11. Додавати матчі на перший етап власного активного турніру; для всіх інших етапів матчі формуються автоматично після визначення переможця
   2.12. Визначати результати матчу власного турнірну
   2.13. Встановлювати дату проведення матчу власного турніру
3. Як адміністратор (авторизований користувач з необмеженими правами) я хочу мати можливість
   3.1. Приймати запити на створення команд та турнірів від користувачів
   3.2. Переглядати інформацію про всі неактивні турніри (учасників та матчі) та команди (учасників)
   3.3. Видаляти будь-які (активні/неактивні, власні/користувацькі) команди та турніри
   3.4. Додавати матчі до будь-яких активних турнірів та визначати переможців, а також змінювати дату проведення матчів

Отже, загалом предметну область курсової роботи складають користувачі з різними рівнями прав, а також сутності команд з їхніми користувачами-учасниками, турнірів з їхніми командами-учасниками та матчами.

РОЗДІЛ 2
АРХІТЕКТУРА ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ БД

З врахуванням функціональних вимог до програмного продукту була розроблена реляційна модель, що забезпечуватиме зберігання необхідних даних у форматі сутностей та відношень між ними. Наприклад, користувачі можуть належати командам, так само можна сказати, що команди складаються з користувачів тощо. Це є прикладом відношень між різними сутностями.

Для опису сутностей користувачів та їх прав створені три таблиці – users, role та таблиця для зв’язування user_role, що співвідносить користувача та його роль в системі (звичайний користувач чи адміністратор).

Для сутності команди створена таблиця team, що містить в собі загальну інформацію, а також менеджера, що є користувачем. Також додатково необхідна таблиця, що зв’язує користувача та команду, що робить користувача учасником команди. Варто зазначити, що користувач може бути учасником багатьох команд.

Для сутності турніру створена таблиця tournaments. Додатково для забезпечення необхідних відношень створені таблиці tournament_members та tournament_matchups, що містять в собі інформацію про зв’язки між командами та турнірами (команда є учасником турніру) та матчі турніру відповідно. Варто зазначити, що команда може бути учасником багатьох турнірів та багатьох матчів в рамках одного турніру.

Загалом маємо 8 таблиць, що зображені за допомогою ER-діаграми (entity relationships – зв’язки сутностей) на рисунку 2.1.

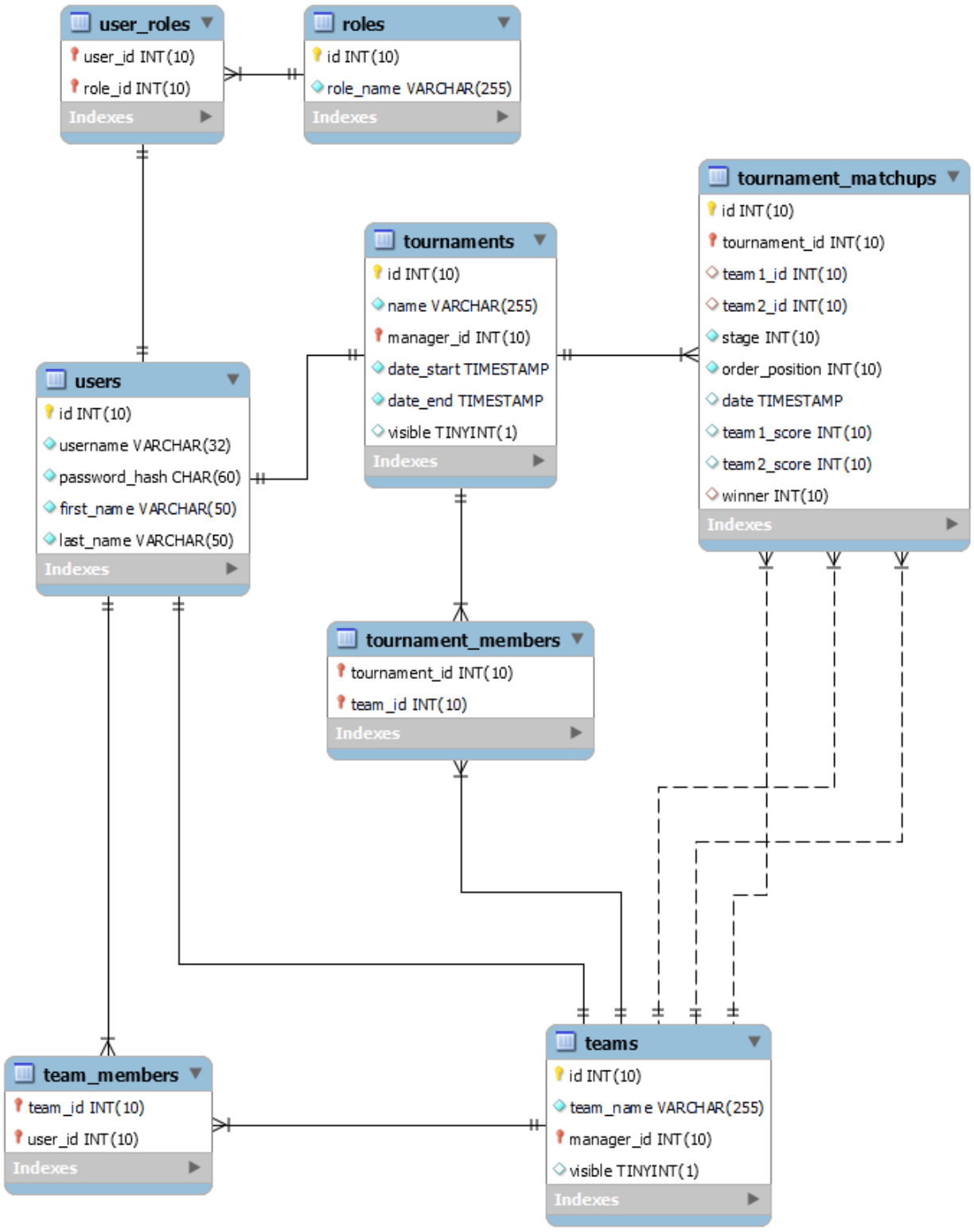

Рис. 2.1 – Діаграма «сутність-зв’язок» БД турнірних таблиць

РОЗДІЛ 3
ДАТАЛОГІЧНА МОДЕЛЬ БАЗИ ДАНИХ

Даталогічна модель бази даних є інструментом для візуалізації потоків даних, що відбуваються в проекті за поточної бізнес-логіки. На кожному рівні деталізації уточнюються процеси, що відносяться до окремих модулів (сервісів) продукту.

Загалом виділяється два основних рівні деталізації – DFD-0 (Data Flow Diagram), що містить в собі основну концепцію продукту, тобто які користувачі з ним взаємодіють та, що це за продукт взагалі; на другому ж рівні вже деталізується основна концепція модулів (сервісів), процесів, що відбуваються між ними та яким чином ці процеси спричиняються, тобто відповідні потоки даних та дій користувачів. Також використовується ще третій рівень DFD-2, який потрібен для деталізації певних компонент системи, тобто яким чином вони функціонують на більш глибокому рівні деталізації.

В рамках даної курсової проекти достатньо двох рівнів деталізації, а відповідні даталогічні моделі зображені на рисунках 3.1 та 3.2.

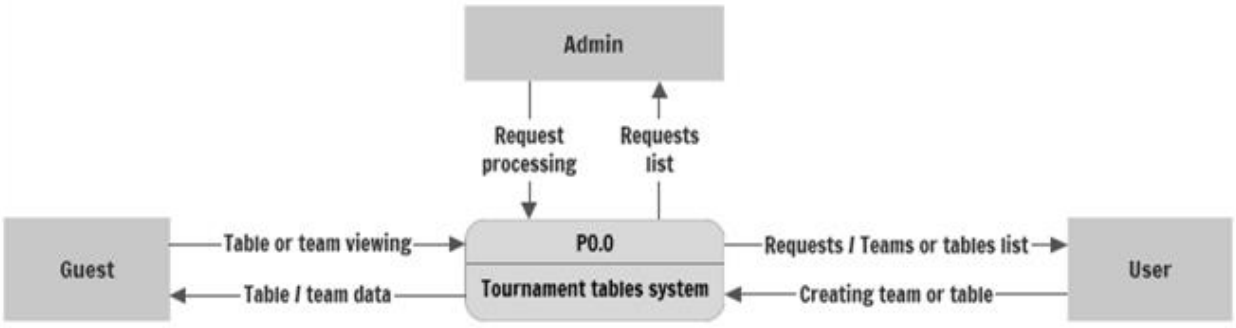

Рис. 3.1 – DFD-0

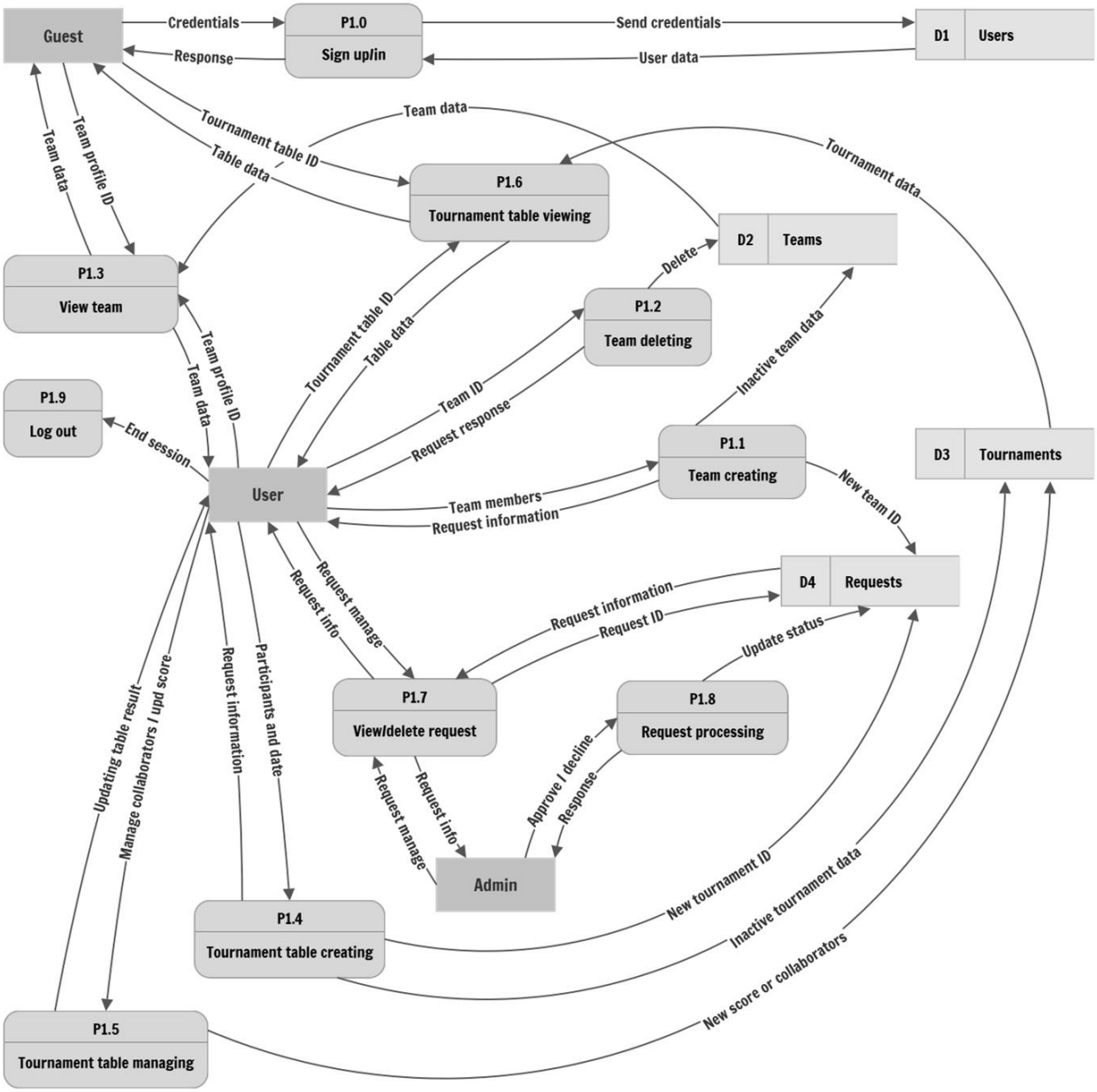
Guest
Credentials
Response
P1.0
Sign up/in
Send credentials
User data
D1
Users
Team data
Team profile ID
Tournament table ID
Table data
P1.6
Tournament table viewing
Team data
Tournament data
D2
Teams
Delete
P1.3
View team
P1.2
Team deleting
Tournament table ID
Table data
Team profile ID
Team data
Team ID
Request response
Inactive team data
P1.9
Log out
End session
User
P1.1
Team creating
D3
Tournaments
Team members
Request information
New team ID
D4
Requests
Request manage
Request info
Request information
Request ID
Update status
P1.7
View/delete request
P1.8
Request processing
Updating table result
Manage collaborators / upd score
Request information
Participants and date
Request manage
Request info
Approve / decline
Response
Admin
P1.4
Tournament table creating
New tournament ID
Inactive tournament data
P1.5
Tournament table managing
New score or collaborators

Рис. 3.2 – DFD-1

РОЗДІЛ 4
ФІЗИЧНА МОДЕЛЬ БАЗИ ДАНИХ. ЗАСОБИ ЗАБЕЗПЕЧЕННЯ ЦІЛОНОСТІ БАЗИ ДАНИХ

Реалізована фізична модель бази даних повністю відповідає наведеній ER-діаграмі в розділі 2 даної пояснювальної записки. Кожна таблиця в результаті була описана за допомогою SQL-запитів, що створюють таблиці із заданими полями та типами даних.

Для забезпечення цілісності бази даних в кожній таблиці визначенні primary key (основний ключ) та foreign keys (стороні ключі), що забезпечують необхідну конзистентність бази даних. Тобто, наприклад, не можна видати роль користувачеві, якого не існує, або не можна створити турнір учасниками якого будуть не існуючі команди тощо.

Реляційна система керування базами даних MySQL забезпечує ефективні механізми збереження та індексації даних за допомогою визначення основних та сторонніх ключів. Навіть з врахуванням того, що дана СКБД є безкоштовною. Проте в рамках даної курсової роботи можливостей даної СКБД є цілком достатньо, адже основні операції, що виконуються в процесі роботи продукту – це так звані CRUD операції (create, read update, delete).

Основний скрипт для створення таблиць та налаштування зв’язків між ними наведений у Додатку А. Усі інші запити до бази даних виконуються за допомогою бібліотеки Sequalize, що за допомогою коду, наведеного у Додатку Б, що представляє модель даних, автоматично створює необхідні SQL-запити до бази даних. Дана ORM також має ефективні методи забезпечення цілісності бази, а саме валідацію вхідних даних запитів, налаштування зв’язків між таблицями One-to-One (один до одного), One-to-Many (один до багатьох) тощо.

РОЗДІЛ 5
ОПИС ПРОГРАМНОГО ПРОДУКТУ

Увесь інтерфейс системи реалізований лише у вигляді зовнішніх REST-запитів. За допомогою GET-запитів отримується інформація про турнірні таблиці, їхніх учасників та матчі, про команди та їх учасників. За допомогою POST-запитів інформація вносить до бази даних. За допомогою PUT-запитів оновлюються записи в базі даних – наприклад, оновлення записів про матчі, а саме дати проведення, переможців тощо. За допомогою DELETE-запитів видаляються турнірні таблиці, команди тощо.

Приклади взаємодії з реалізованим API наведені на рисунках 5.1-5.9 з додатку Postman, більш детальна документація з прикладами усіх можливих запитів та відповідних помилок, що можуть виникнути, доступна за посиланням - https://documenter.getpostman.com/view/18816873/UVRBompL.

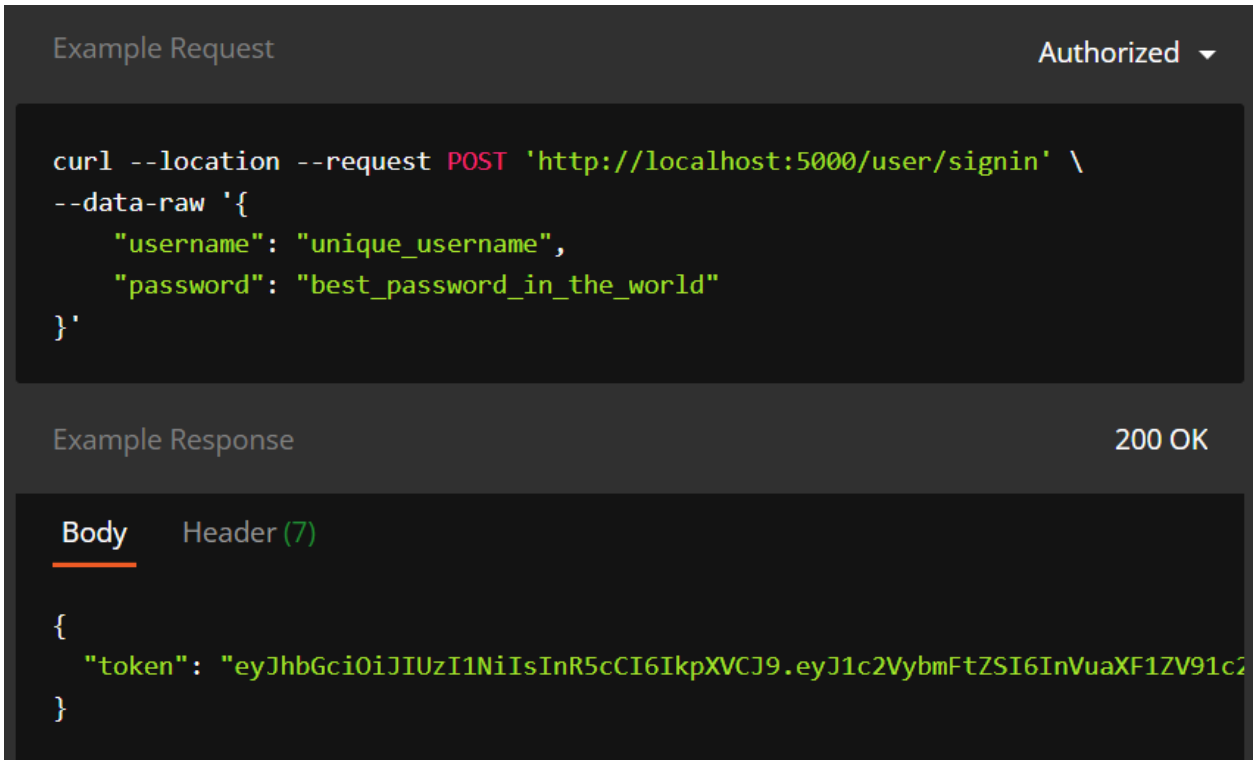

Рис. 5.1 – Успішна авторизація користувача

Example Request 201 Created

```
curl --location --request POST 'http://localhost:5000/user/signup' \
--data-raw '{
    "username": "unique_username",
    "password": "best_password_in_the_world",
    "first_name": "First",
    "last_name": "Last"
}'
```

Example Response 201 Created

Body Header (7)


```
{
  "user": {
    "id": 1
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVuaXF1ZV91c2Vyb
}
```


Рис. 5.2 – Успішна реєстрація користувача

Example Request 201 Created

```
curl --location --request POST 'http://localhost:5000/user/team/create' \
--data-raw '{
    "team_name": "Sympathetic Cats",
    "member_ids": [2, 10]
}'
```

Example Response 201 Created

Body Header (7)


```
{
  "team": {
    "id": 2
  }
}
```


Рис. 5.3 – Успішне створення команди користувачем

Example Request — Already Exists

```
curl --location --request POST 'http://localhost:5000/user/team/create' \
--data-raw '{
    "team_name": "Sympathetic Cats",
    "member_ids": [2, 10]
}'
```

Example Response — 400 Bad Request

Body | Header (7)

```
{
  "error": "name of team must be unique"
}
```

Рис. 5.4 – Помилка при створенні команди з неунікальною назвою

Example Request — 200 OK

```
curl --location --request GET 'http://localhost:5000/admin/teams/1'
```

Example Response — 200 OK

Body | Header (7)

```
{
  "teams": [
    {
      "id": 2,
      "team_name": "Sympathetic Cats",
      "manager_id": 1,
      "visible": 0
    },
    {
      "id": 7,
```

View More

Рис. 5.5 – Перегляд адміністратором запитів на створення команд

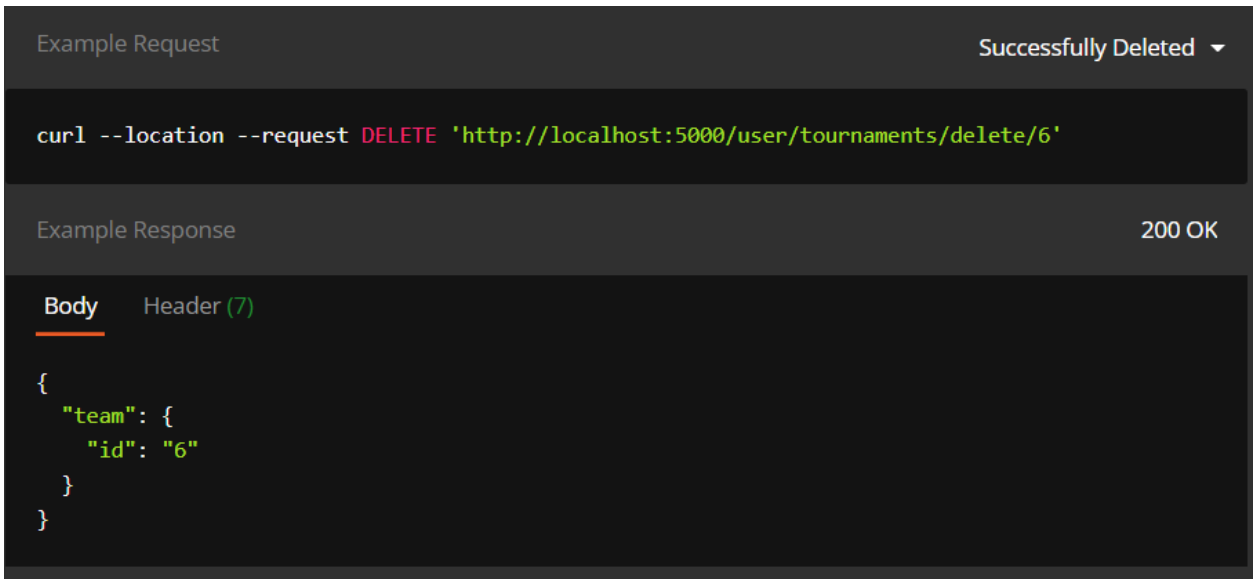

Рис. 5.6 – Успішне видалення команди власником

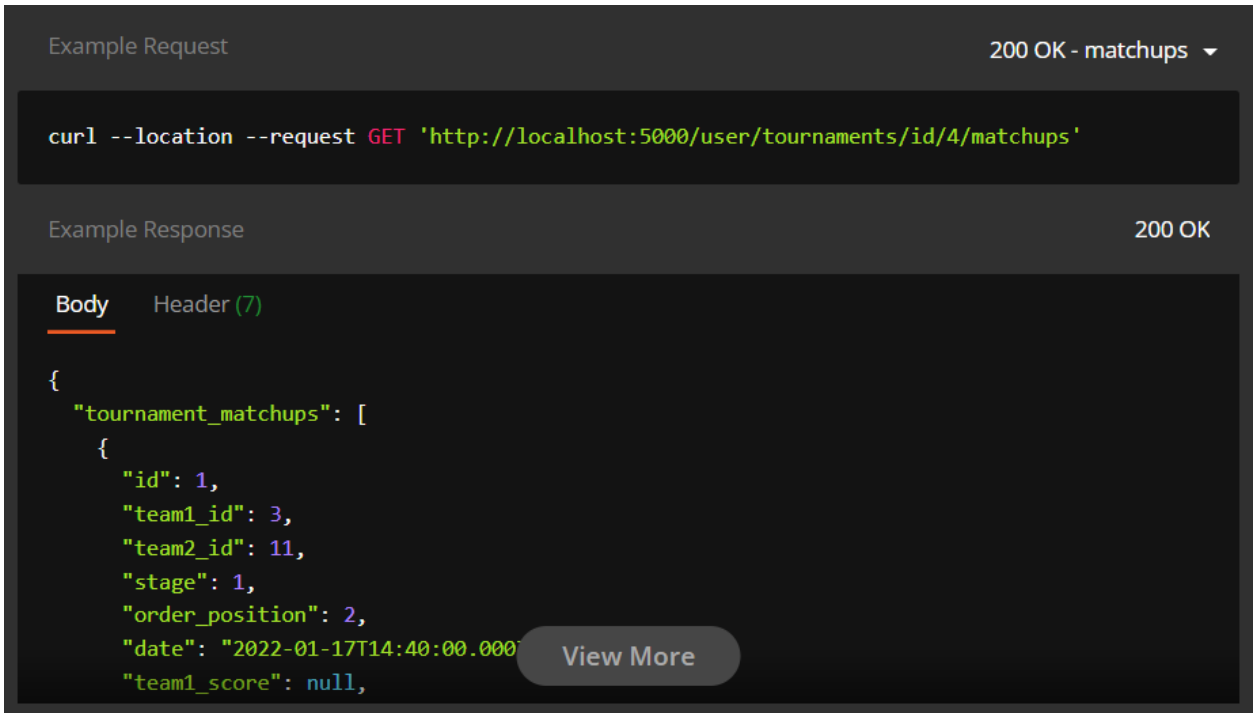

Рис. 5.7 - Перегляд матчів турніру

Example Request — 200 OK - new matchup

```
curl --location --request POST 'http://localhost:5000/user/tournaments/id/4/matchups/addToFirstStage' \
--data-raw '{
    "team1_id": "16",
    "team2_id": "15",
    "date": "2022-01-17 19:30:00",
    "order_position": 3
}'
```

Example Response — 200 OK

Body | Header (7)


```
{
  "matchup": {
    "id": 18,
    "tournament_id": 4,
    "team1_id": "16",
    "team2_id": "15",
    "stage": 1,
    "order_position": 3,
    "date": "2022-01-17T17:30:00.000Z"
  }
```


Рис. 5.8 – Успішне додавання матчу до першого етапу турніру

Example Request — Successfully Updated

```
curl --location --request PUT 'http://localhost:5000/user/tournaments/id/4/matchups/setWinner' \
--data-raw '{
    "team1_score": 5,
    "team2_score": 7,
    "matchup_id": "1"
}'
```

Example Response — 201 Created

Body | Header (7)


```
{
  "matchup": {
    "id": 19,
    "tournament_id": 4,
    "stage": 2,
    "order_position": 1,
    "team2_id": 11
  }
}
```


Рис. 5.9 – Встановлення переможця матчу

ВИСНОВКИ

В ході виконання курсової роботи була проаналізована предметна область теми «Онлайн турнірні таблиці», встановлені основні функціональні вимоги до системи, визначені методи та засоби реалізації поставленої задачі.

В результаті були створені чотири сервіси, що відповідають за функціональність турнірних таблиць, менеджмент команд, функціонал адміністратора та користувача. А також 8 таблиць реляційної моделі.

Отримані сервіси та API взаємодії з ними повністю забезпечує увесь необхідний функціонал. А тому поставлена задача курсової роботи виконана. Також в ході виконання роботи були здобуті практичні навички роботи з реляційною базою даних MySQL, а також освоєні технології для налаштування бекенду та технологія ORM, що забезпечує зручний інтерфейс взаємодії з базою даних.

Увесь поточний функціонал системи задокументований у вигляді REST-запитів за допомогою застосунку Postman. Документація з чіткими прикладами запитів доступна за посиланням наведеним у Розділі 5.

З подальших покращень є необхідність розробити графічний веб-інтерфейс. А також збільшення функціоналу – наприклад, можливість редагувати склад команди, переглядати історію матчів гравця тощо. Але на даний момент всі функціональні вимоги є виконані та отримана система є повноцінною.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. MySQL Documentation : веб-сайт. URL: https://dev.mysql.com/ (дата звернення 20.12.2021)
2. NodeJS Документація : веб-сайт. URL: https://nodejs.org/uk/docs/ (дата звернення 20.12.2021)
3. Sequelize Documentation : веб-сайт. URL: https://sequelize.org/master/ (дата звернення 20.12.2021)
4. Postman : веб-сайт. URL: https://learning.postman.com/docs/publishing-your-api/documenting-your-api/ (дата звернення 20.12.2021)
5. Leonard Richardson, Mike Amundsen RESTful Web APIs. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2013.
6. Ентоні Моллінаро SQL Збірник рецептів. Санкт-Петербург, Москва, 2009.

ДОДАТОК А

```
CREATE TABLE IF NOT EXISTS `users` (
    `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `username` VARCHAR(32) NOT NULL,
    `password_hash` CHAR(60) NOT NULL,
    `first_name` VARCHAR(50) NOT NULL,
    `last_name` VARCHAR(50) NOT NULL,
    CONSTRAINT pk_user_id PRIMARY KEY (`id`),
    CONSTRAINT uq_username UNIQUE (`username`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `roles` (
    `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `role_name` VARCHAR(255) NOT NULL,
    CONSTRAINT pk_role_id PRIMARY KEY (`id`),
    CONSTRAINT uq_role_name UNIQUE (`role_name`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `tournaments` (
    `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    `manager_id` INT UNSIGNED NOT NULL,
    `date_start` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    `date_end` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    `visible` TINYINT(1) DEFAULT 0,
    CONSTRAINT pk_tournament_id PRIMARY KEY (`id`),
    CONSTRAINT uq_tournament_name UNIQUE (`name`),
    CONSTRAINT fk_tournaments_users FOREIGN KEY
(`manager_id`) REFERENCES users (`id`)
```

```
)  ENGINE=INNODB CHARACTER SET=UTF8 COLLATE =
UTF8_UNICODE_CI;

CREATE TABLE IF NOT EXISTS `teams` (
    `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `team_name` VARCHAR(255) NOT NULL,
    `manager_id` INT UNSIGNED NOT NULL,
    `visible` TINYINT(1) DEFAULT 0,
    CONSTRAINT pk_team_id PRIMARY KEY (`id`),
    CONSTRAINT uq_team_name UNIQUE (`team_name`),
    CONSTRAINT fk_teams_users FOREIGN KEY (`manager_id`)
REFERENCES users (`id`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `user_roles` (
    `user_id` INT UNSIGNED NOT NULL,
    `role_id` INT UNSIGNED NOT NULL,
    CONSTRAINT pk_user_role PRIMARY KEY (`user_id`,
`role_id`),
    CONSTRAINT fk_user_roles_users FOREIGN KEY (`user_id`)
REFERENCES users (`id`) ON DELETE CASCADE,
    CONSTRAINT fk_user_roles_roles FOREIGN KEY (`role_id`)
REFERENCES roles (`id`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `tournament_members` (
    `tournament_id` INT UNSIGNED NOT NULL,
    `team_id` INT UNSIGNED NOT NULL,
    CONSTRAINT pk_tournament_member PRIMARY KEY
(`tournament_id`, `team_id`),
    CONSTRAINT fk_tournament_members_tournaments FOREIGN
KEY (`tournament_id`)
```

```
                                                    REFEREN
CES tournaments (`id`) ON DELETE CASCADE,
    CONSTRAINT fk_tournament_members_teams FOREIGN KEY
(`team_id`) REFERENCES teams (`id`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `tournament_matchups` (
    `id` INT UNSIGNED AUTO_INCREMENT,
    `tournament_id` INT UNSIGNED NOT NULL,
    `team1_id` INT UNSIGNED NULL,
    `team2_id` INT UNSIGNED NULL,
    `stage` INT UNSIGNED NOT NULL,
    `order_position` INT UNSIGNED NOT NULL,
    `date` TIMESTAMP NULL,
    `team1_score` INT UNSIGNED NULL,
    `team2_score` INT UNSIGNED NULL,
    `winner` INT UNSIGNED NULL,
    CONSTRAINT pk_tournament_matchup PRIMARY KEY (`id`),
    CONSTRAINT fk_tournament_matchups_tournaments FOREIGN
KEY (`tournament_id`)
                                                    REFERENCE
S tournaments (`id`) ON DELETE CASCADE,
    CONSTRAINT fk_tournaments_matchups_team1 FOREIGN KEY
(`team1_id`)
                                                REFERENCES
teams (`id`),
    CONSTRAINT fk_tournaments_matchups_team2 FOREIGN KEY
(`team2_id`)
                                                REFERENCES
teams (`id`),
    CONSTRAINT fk_tournaments_matchups_winner FOREIGN KEY
(`winner`)
```

```
                                                    REFERENCES
teams (`id`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `team_members` (
    `team_id` INT UNSIGNED NOT NULL,
    `user_id` INT UNSIGNED NOT NULL,
    CONSTRAINT pk_team_member PRIMARY KEY (`team_id`,
`user_id`),
    CONSTRAINT fk_team_members_teams FOREIGN KEY
(`team_id`) REFERENCES teams (`id`) ON DELETE CASCADE,
    CONSTRAINT fk_team_members_users FOREIGN KEY
(`user_id`) REFERENCES users (`id`)
) ENGINE=INNODB CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;
```

ДОДАТОК Б

```
const User = sequelize.define('user', {
    id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
    }, username: {
        type: DataTypes.STRING(32),
        allowNull: false,
        unique: true,
        validate: {
            is: /^[a-z]+[a-z_]*[a-z]+$/i
        }
    }, password_hash: {
        type: DataTypes.CHAR(60),
        allowNull: false,
    }, first_name: {
        type: DataTypes.STRING(50),
        allowNull: false,
        validate: {
            isAlpha: true
        }
    }, last_name: {
        type: DataTypes.STRING(50),
        allowNull: false,
        validate: {
            isAlpha: true
        }
    }
}, {
    timestamps: false
```

```
});

const UserRole = sequelize.define('user_roles', {
    user_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: User,
            key: 'id'
        }
    }, role_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: Role,
            key: 'id'
        }
    }
}, {
    timestamps: false
});

const Role = sequelize.define('roles', {
    id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
    }, role_name: {
        type: DataTypes.STRING(255),
        allowNull: false,
```

```
        unique: true,
        validate: {
            isAlpha: true
        }
    }
}, {
    timestamps: false
});

const Team = sequelize.define('teams', {
    id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
    }, team_name: {
        type: DataTypes.STRING(255),
        allowNull: false,
        unique: true
    }, manager_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: User,
            key: 'id'
        }
    }, visible: {
        type: DataTypes.TINYINT(1),
        allowNull: false,
        defaultValue: 0
    }
}, {
```

```
    timestamps: false
});

const TeamMember = sequelize.define('team_members', {
    user_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: User,
            key: 'id'
        }
    }, team_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: Team,
            key: 'id'
        }
    }
}, {
    timestamps: false
});

const Tournament = sequelize.define('tournaments', {
    id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
    }, name: {
        type: DataTypes.STRING(255),
```

```
        allowNull: false,
        unique: true
    }, manager_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        allowNull: false,
        references: {
            model: User,
            key: 'id'
        }
    }, date_start: {
        type: DataTypes.DATE,
        allowNull: false,
        validate: {
            isDate: true
        }
    }, date_end: {
        type: DataTypes.DATE,
        allowNull: false,
        validate: {
            isDate: true
        }
    }, visible: {
        type: DataTypes.TINYINT(1),
        allowNull: false,
        defaultValue: 0
    }
}, {
    timestamps: false
});

const TournamentMember =
sequelize.define('tournament_members', {
    tournament_id: {
```

```
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: Tournament,
            key: 'id'
        }
    }, team_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        allowNull: false,
        references: {
            model: Team,
            key: 'id'
        }
    }
}, {
    timestamps: false
});

const TournamentMatchup =
sequelize.define('tournament_matchups', {
    id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false
    }, tournament_id: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        allowNull: false,
        references: {
            model: Tournament,
            key: 'id'
```

```
    }
}, team1_id: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    references: {
        model: Team,
        key: 'team_id'
    }
}, team2_id: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    references: {
        model: Team,
        key: 'team_id'
    }
}, stage: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    allowNull: false
}, order_position: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    allowNull: false
}, date: {
    type: DataTypes.DATE,
    validate: {
        isDate: true
    }
}, team1_score: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    validate: {
        isInt: true
    }
}, team2_score: {
    type: DataTypes.INTEGER(10).UNSIGNED,
    validate: {
        isInt: true
```

```
        }
    }, winner: {
        type: DataTypes.INTEGER(10).UNSIGNED,
        references: {
            model: Team,
            key: 'team_id'
        }
    }
}, {
    timestamps: false
});

User.hasOne(UserRole, {
    as: 'role', sourceKey: 'id', foreignKey: 'user_id',
onDelete: 'cascade'
});

Tournament.hasMany(TournamentMember, {
    as: 'members', sourceKey: 'id', foreignKey:
'tournament_id', onDelete: 'cascade', hooks: true
});
Tournament.hasMany(TournamentMatchup, {
    as: 'matchups', sourceKey: 'id', foreignKey:
'tournament_id', onDelete: 'cascade', hooks: true
});

Team.hasMany(TeamMember, {
    as: 'members', sourceKey: 'id', foreignKey: 'team_id',
onDelete: 'cascade', hooks: true
});
```