# Tournament Management System

# SDLC Of The Project

## Planning phase:

### Define Objectives and Scope:

* + Objective: Organize a college tournament that promotes competition, collaboration, and showcases diverse talents among students.
  + Scope:
    - Specify the number of events: 5 events per team or individual.
    - Clarify the distinction between team and individual events.
    - Set the maximum number of participants: 4 teams with 5 members each (20 members) or 20 individual participants.
    - Develop a flexible and effective web application to manage registrations, event scheduling, and scoring.

### Requirements:

* + Identify the technology needed for online registrations and registration systems for virtual events.
  + Gather the resources and technology background of potential individuals involved in the development process.
  + Utilize the technologies needed to achieve the customer requirements and satisfy his needs
  + Create a fully functional web application with registration, login, event management and score management systems, admin dashboard, CRUD operations given to the admin to control: Events, Users, Questions and Teams. And much more.
  + Create a user friendly interface for the users to access the application.
  + Create a user friendly interface for the admin to access the application.

## Analysis phase:

### Requirements Gathering:

* + Identify the max number of events per team or individual.
  + decide the scoring system for the events.
  + Identify the max number of participants per team or individual.
  + Clarify how points are calculated for each event based on their performance in the event.
  + Identify the data needed from the participants and teams.
  + Discuss the nature of the events and their scope.

## Design Phase:

### System Architecture:

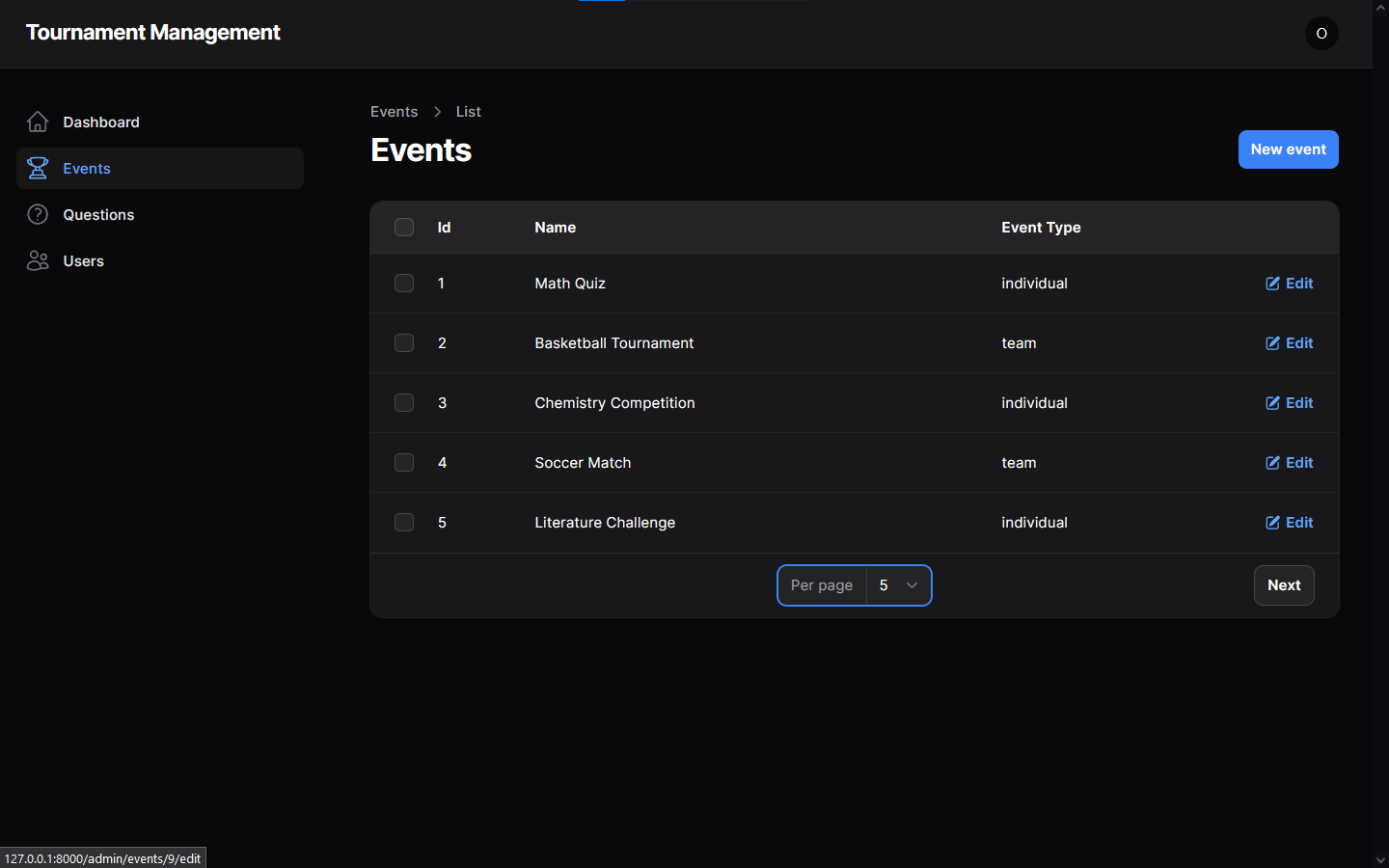
* + I will be using the MVC architecture for the project.
  + The system will be divided into 3 layers:
    - Model: This layer will contain the database and the models for the application.
    - View: This layer will contain the templates and the views for the application.
    - Controller: This layer will contain the controllers for the application.
  + I designed a scoring system that evaluates the performance of the participants in the event.
  + The scoring system will be based on the number of questions answered correctly by the participants.
  + The scoring system is flexible and can be changed according to the needs of the event.
  + I Implemented a registration and login system for the users with authentication and authorization.
  + I Implemented a dashboard for the admin to manage the events, users, questions and teams.
  + I implemented a CRUD operations for the admin to control the events, users, questions and teams.
  + I made protected routes and access tokens to ensure the security of the application.
  + I hashed the password of the user to ensure the security of the application.

### Database Design:

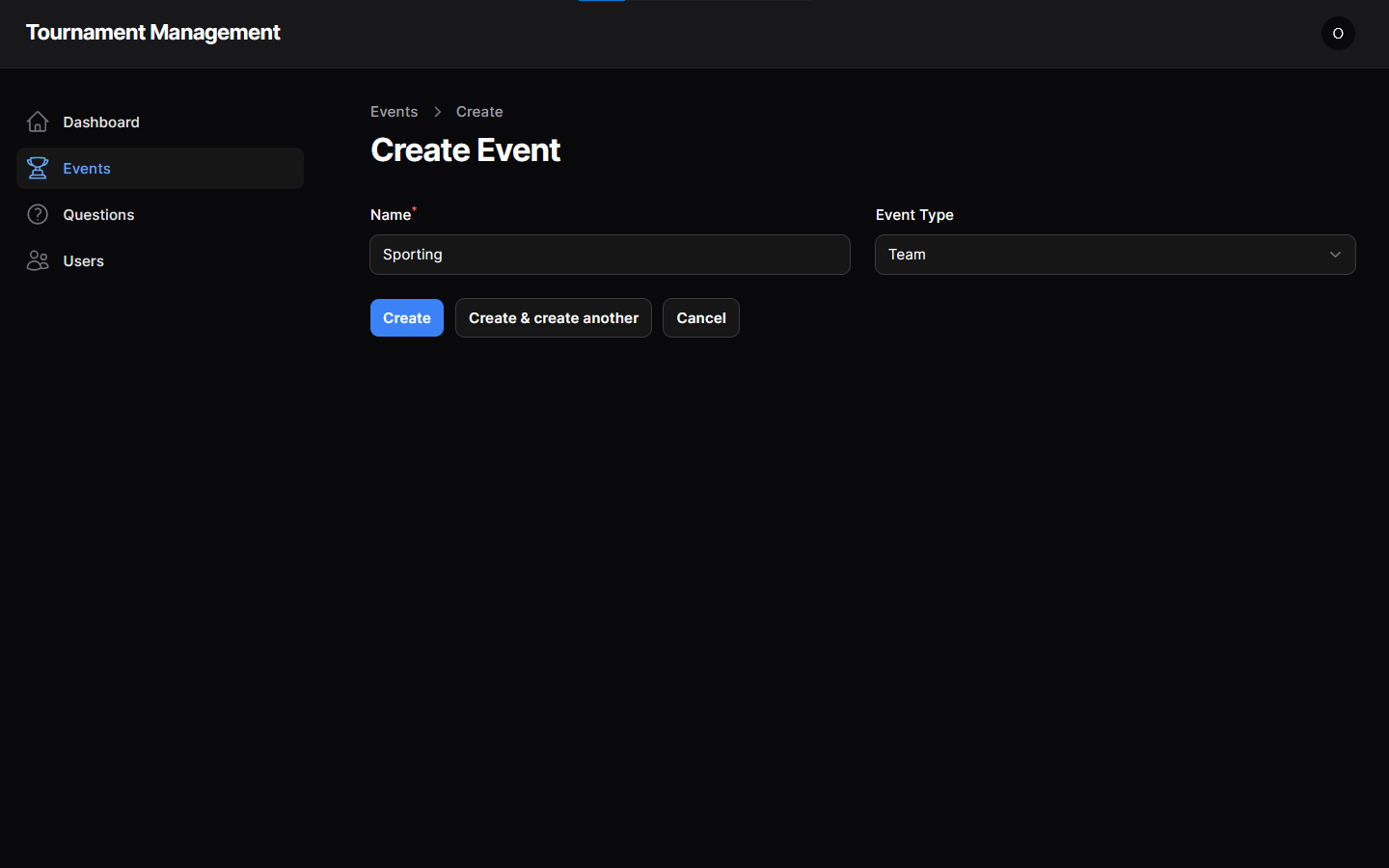
* + I designed a database for the application with the following tables:
    - Users:
      * id
      * name
      * email
      * password (hashed)
      * role (enum: admin, user)
      * team\_members (json string containing the IDs of the users of the team)
      * created\_at
      * updated\_at
    - Events:
      * id
      * name
      * type (enum: team, individual)
      * created\_at
      * updated\_at
    - Questions:
      * id
      * event\_id
      * question
      * answer
      * created\_at
      * updated\_at
    - Event\_Participants:
      * id
      * event\_id
      * user\_id
      * score
      * created\_at
      * updated\_at

### User Interface (UI) Design:

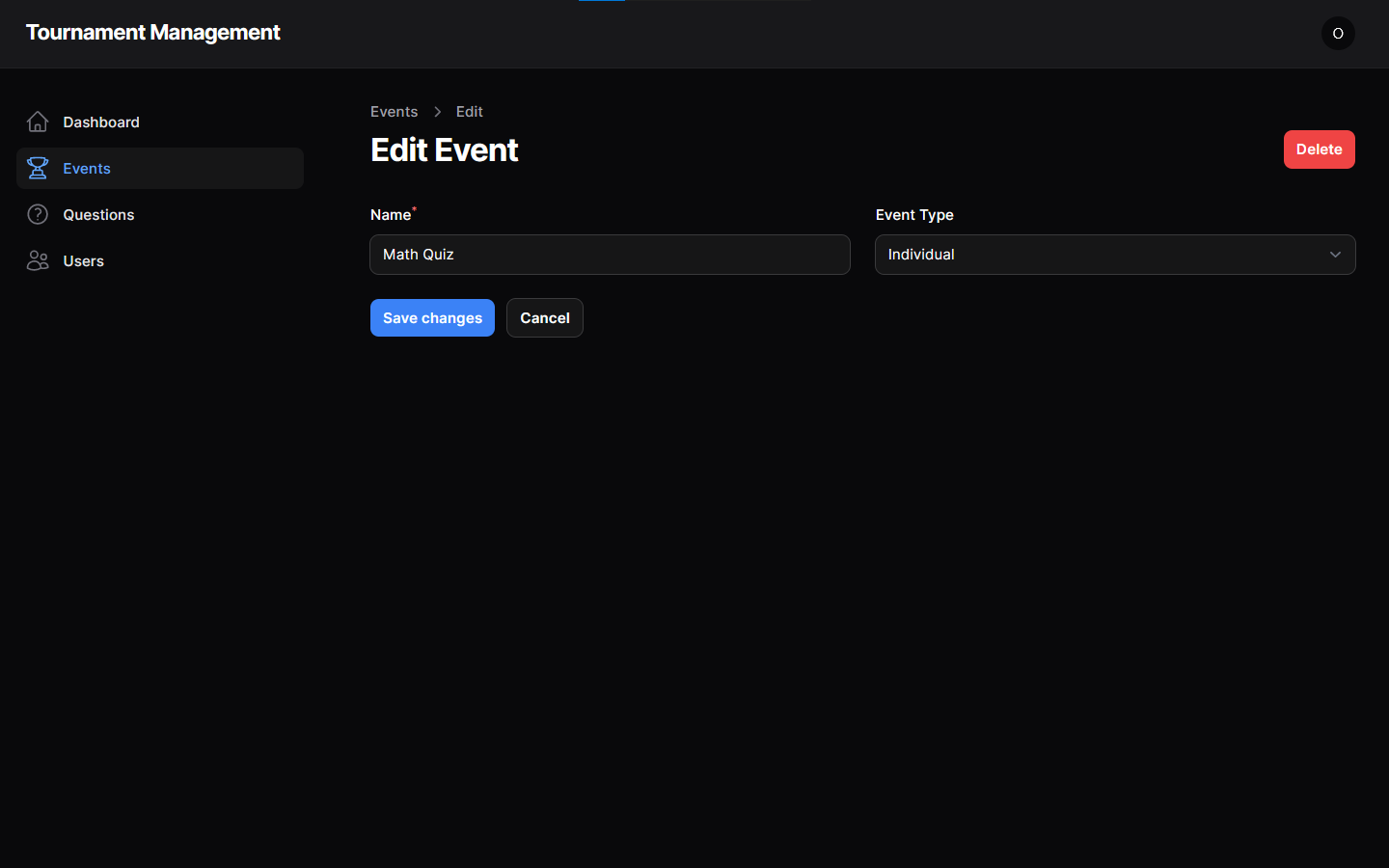
* + I designed a user friendly interface for the users to access the application.
  + I designed a simple registration form for the users to register themselves.
  + I designed a login form for the users to login to the application.
  + I included validation checks on the data with user-friendly error messages.
  + I designed a page for the user to browse the available events and register on them rather as team or as individual with the ability to invite team members to participate with him.
  + I designed a page for the user to view the scores of the participants in the event.
  + I designed a page for the user to view the details of the event.
  + I designed a user friendly interface for the admin to access the application.
  + I designed a login form for the admins to login to the application.
  + I designed a versatile dashboard for the admin to manage the events, users, questions and teams.
  + examples of the admin dashboard:
  + Events:
    - View all Events:



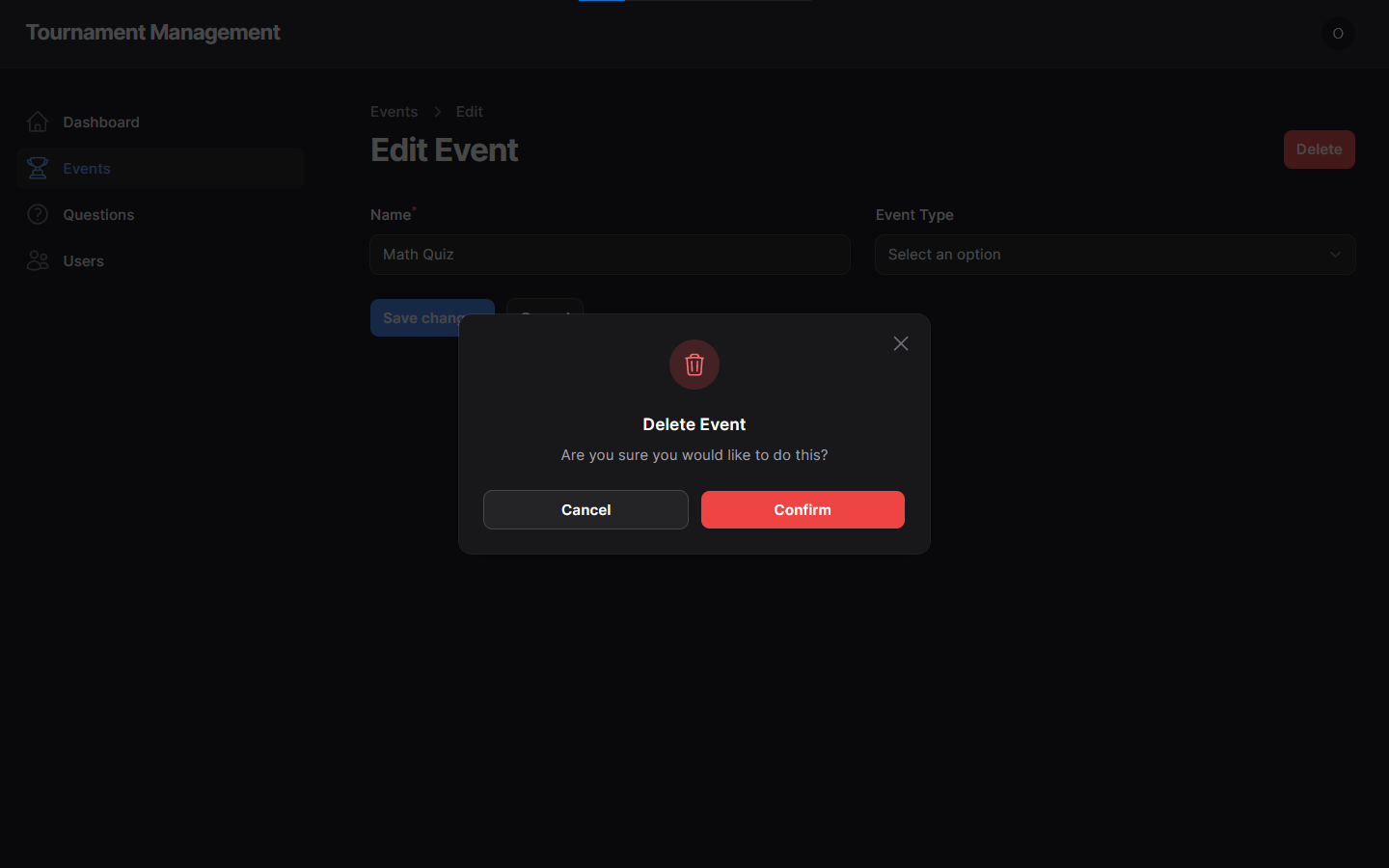
* + - Add new Event:



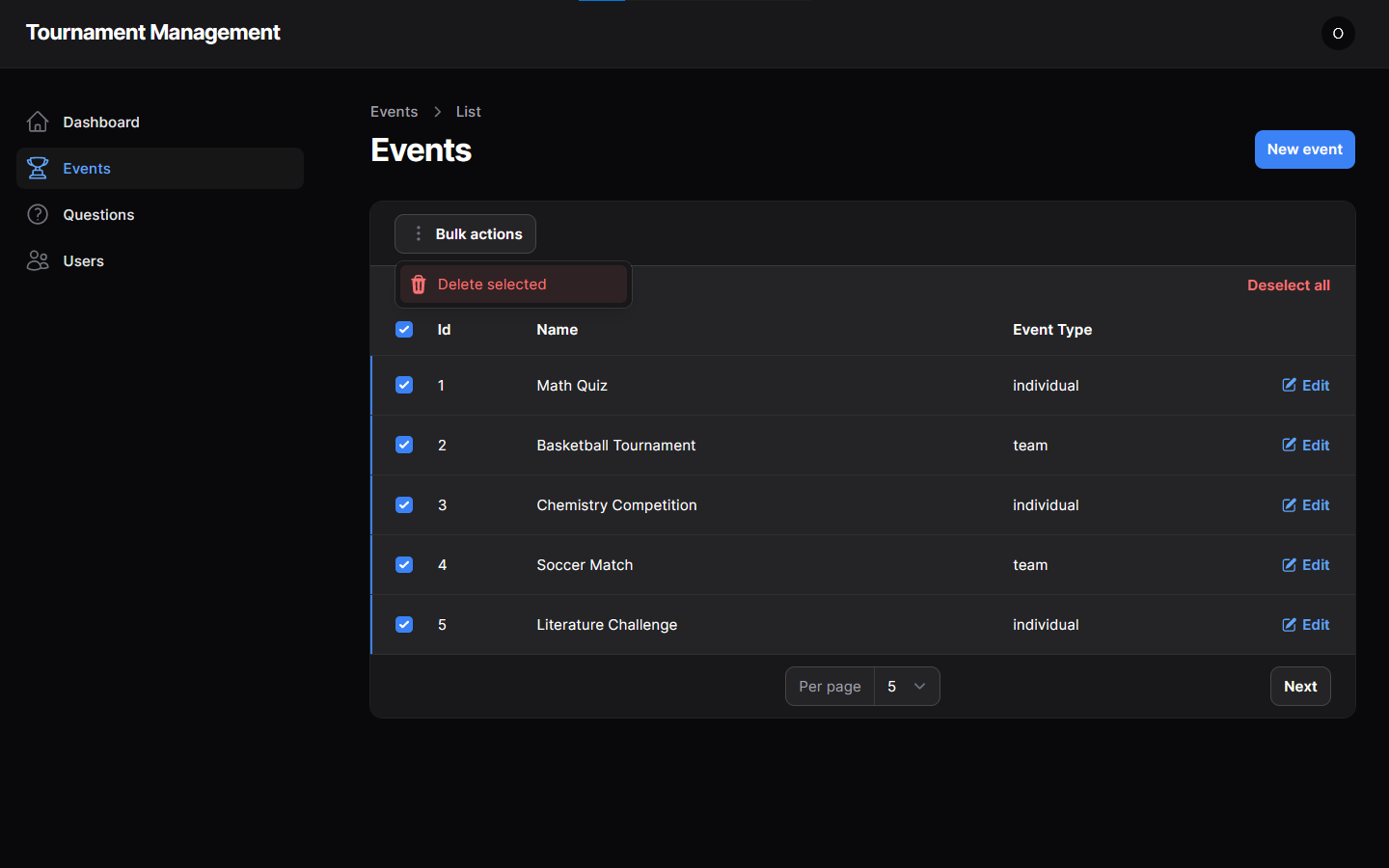
* + - Edit Event:



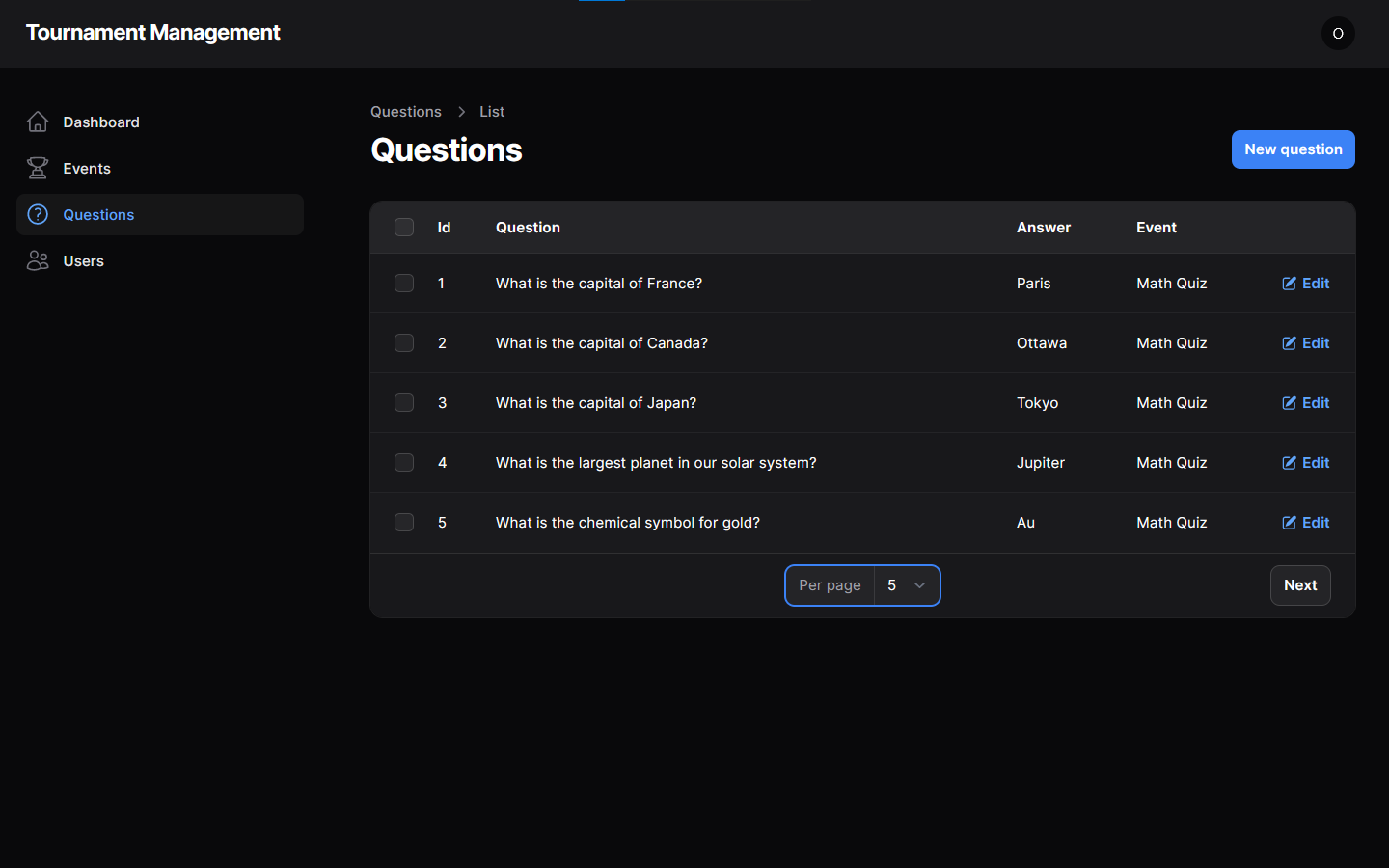
* + - Delete Event:



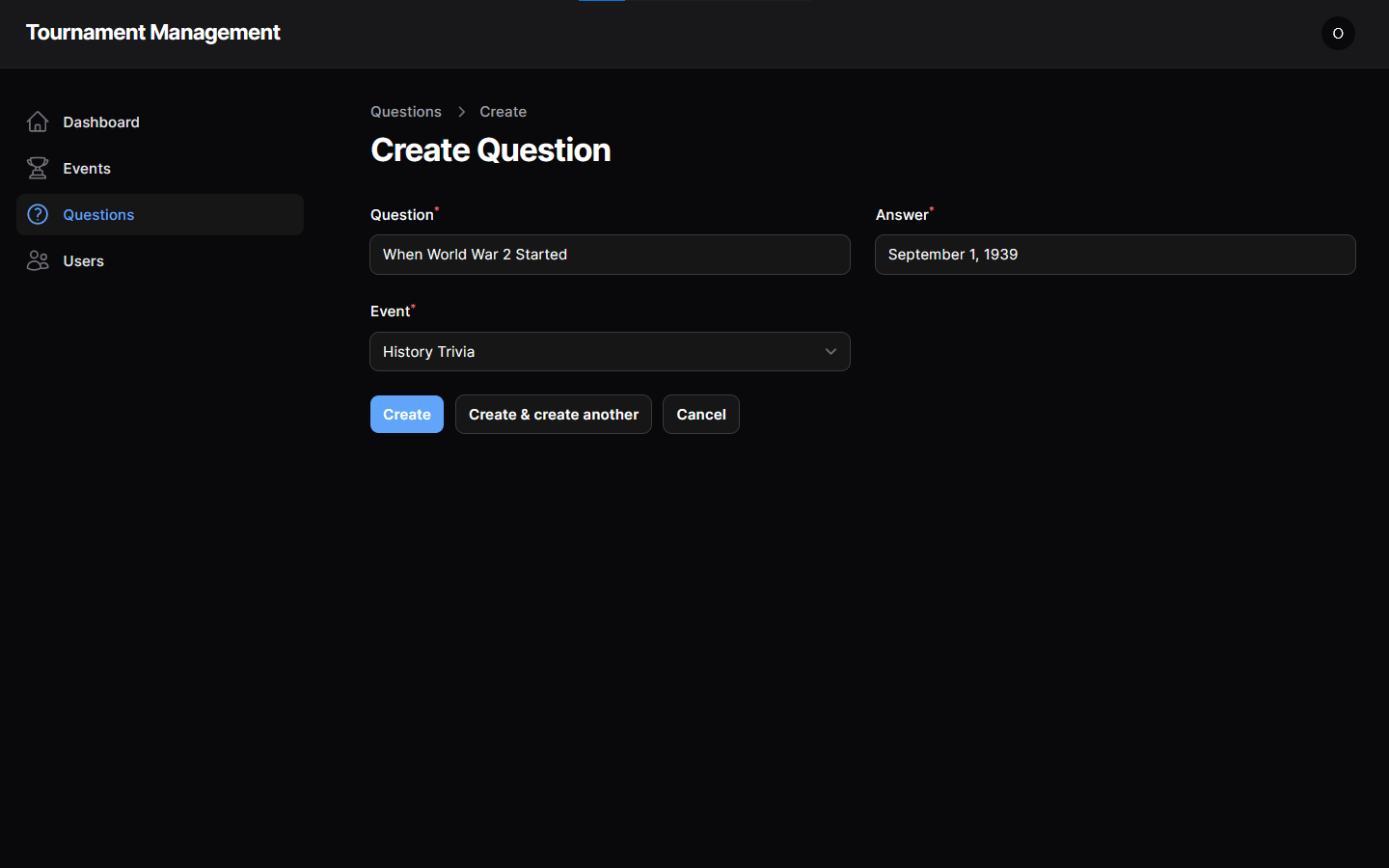
* + - Mass Delete Events:



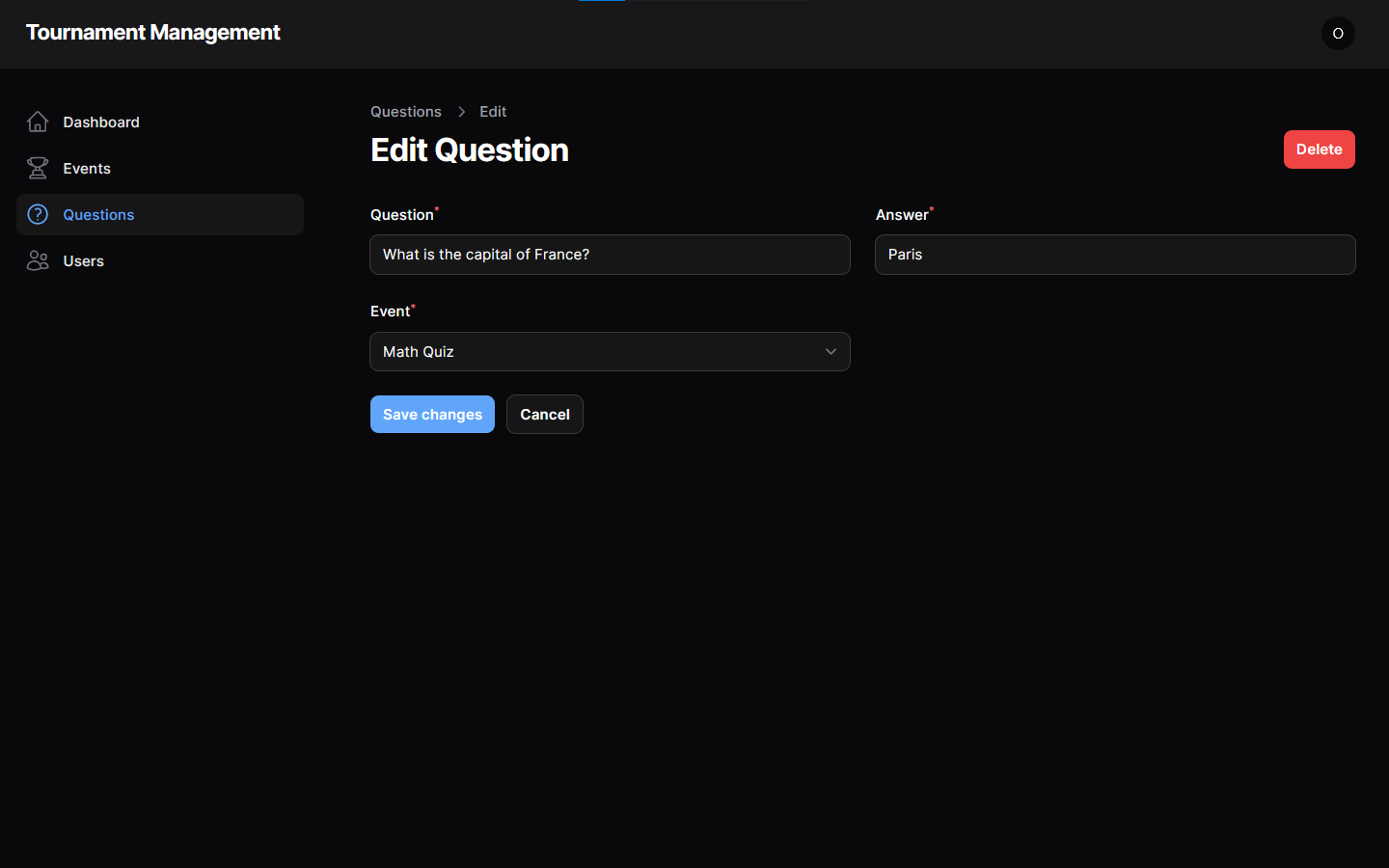
* + Questions:
    - View all Questions:



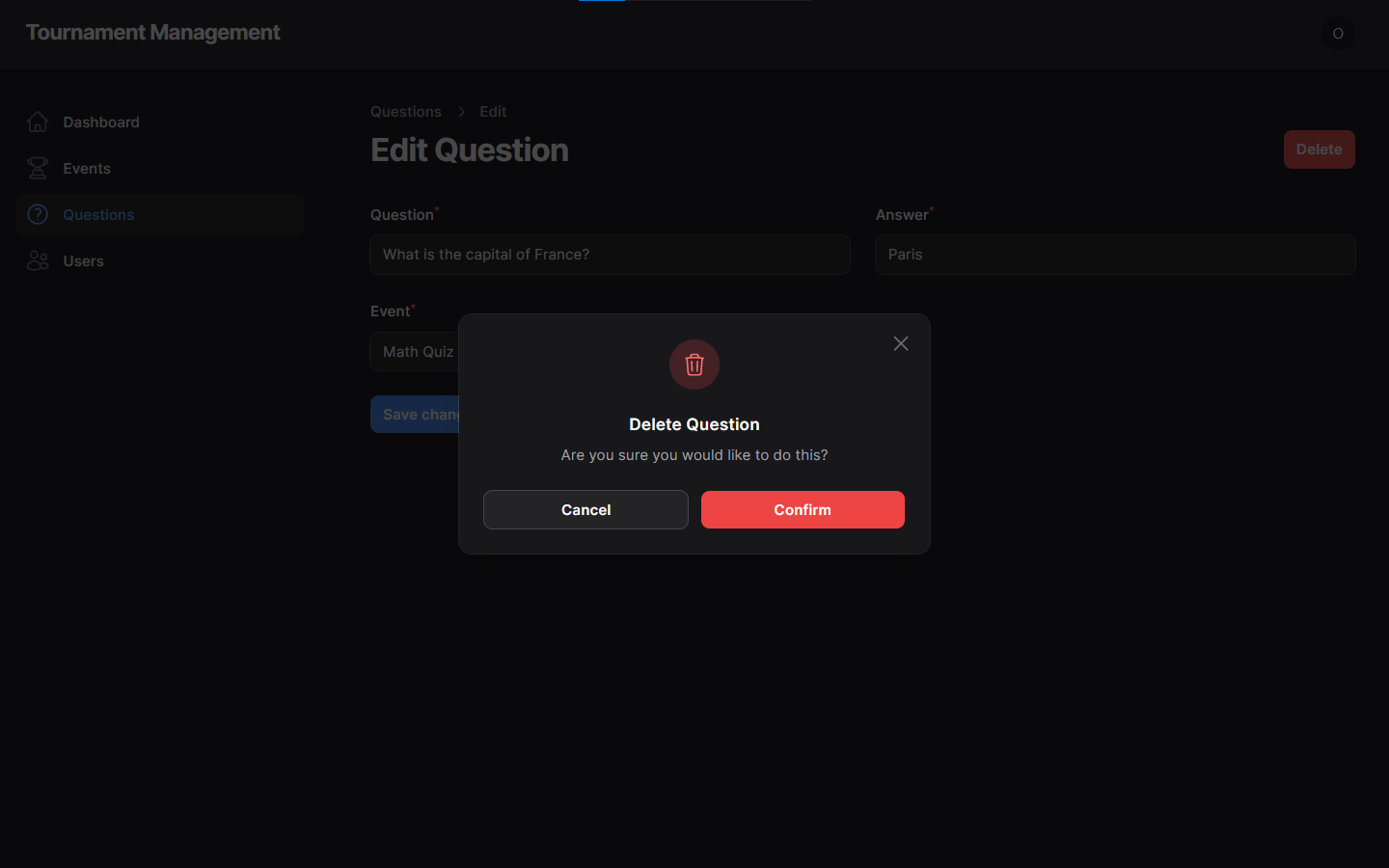
* + - Add new Question:



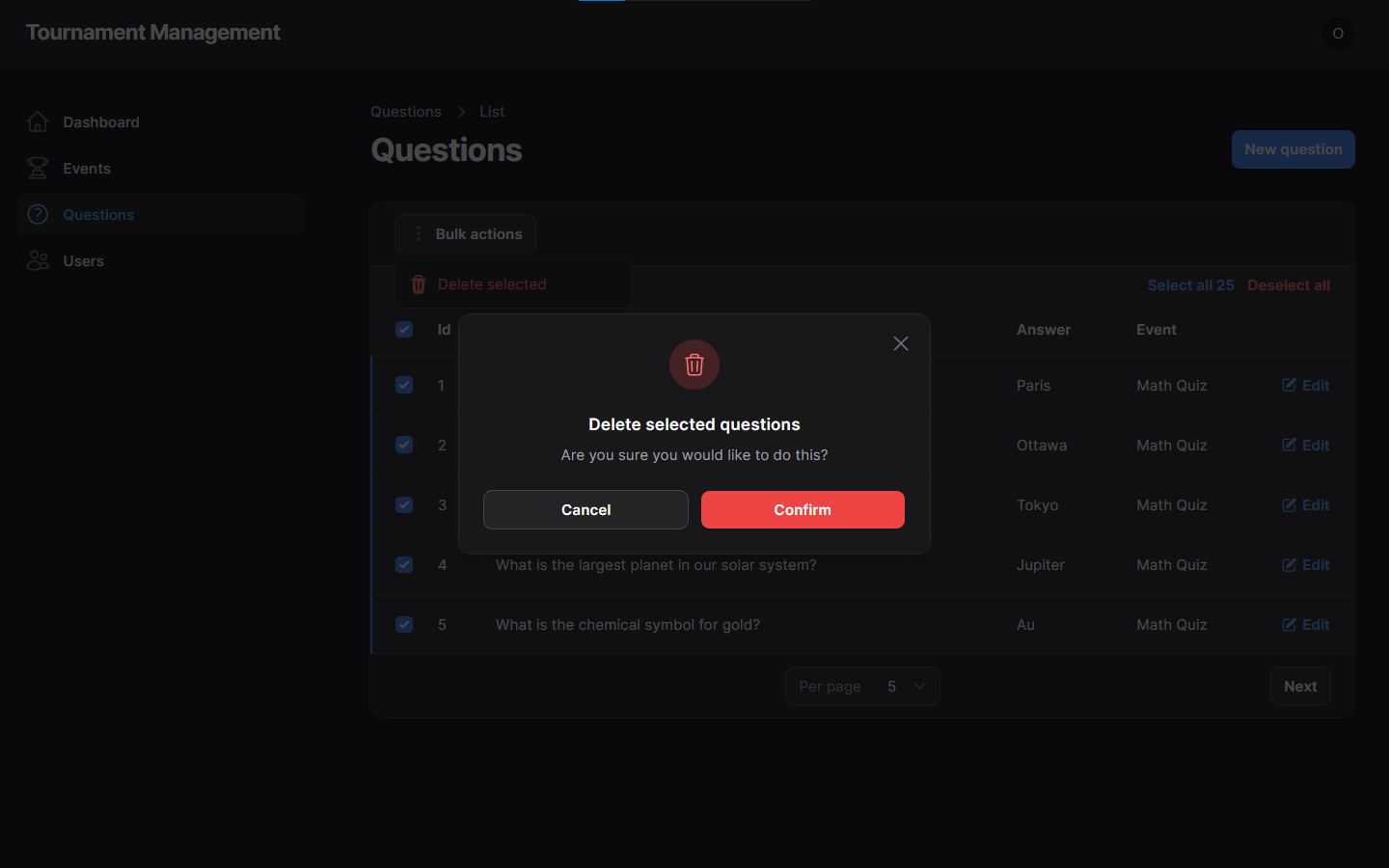
* + - Edit Question:



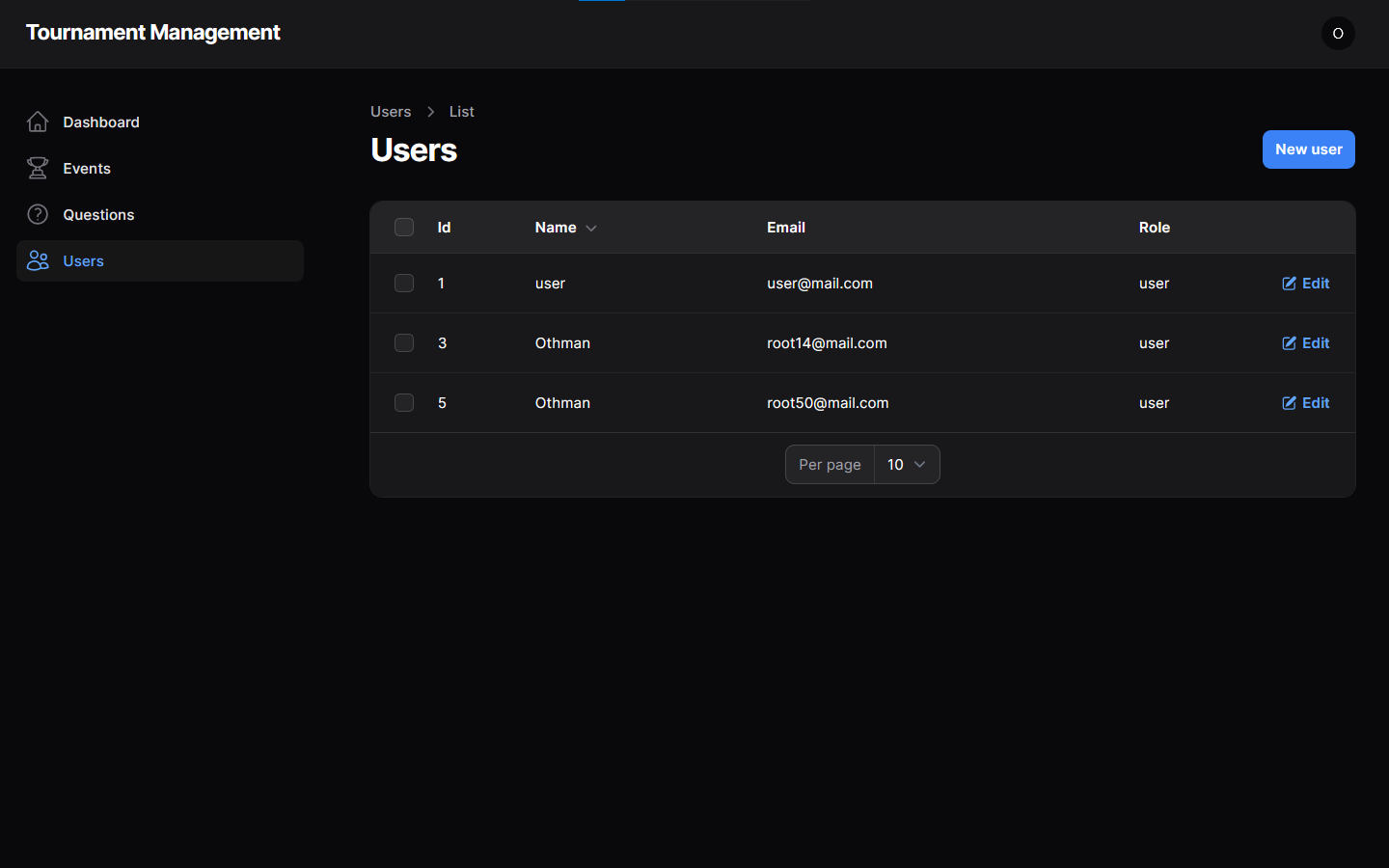
* + - Delete Question:



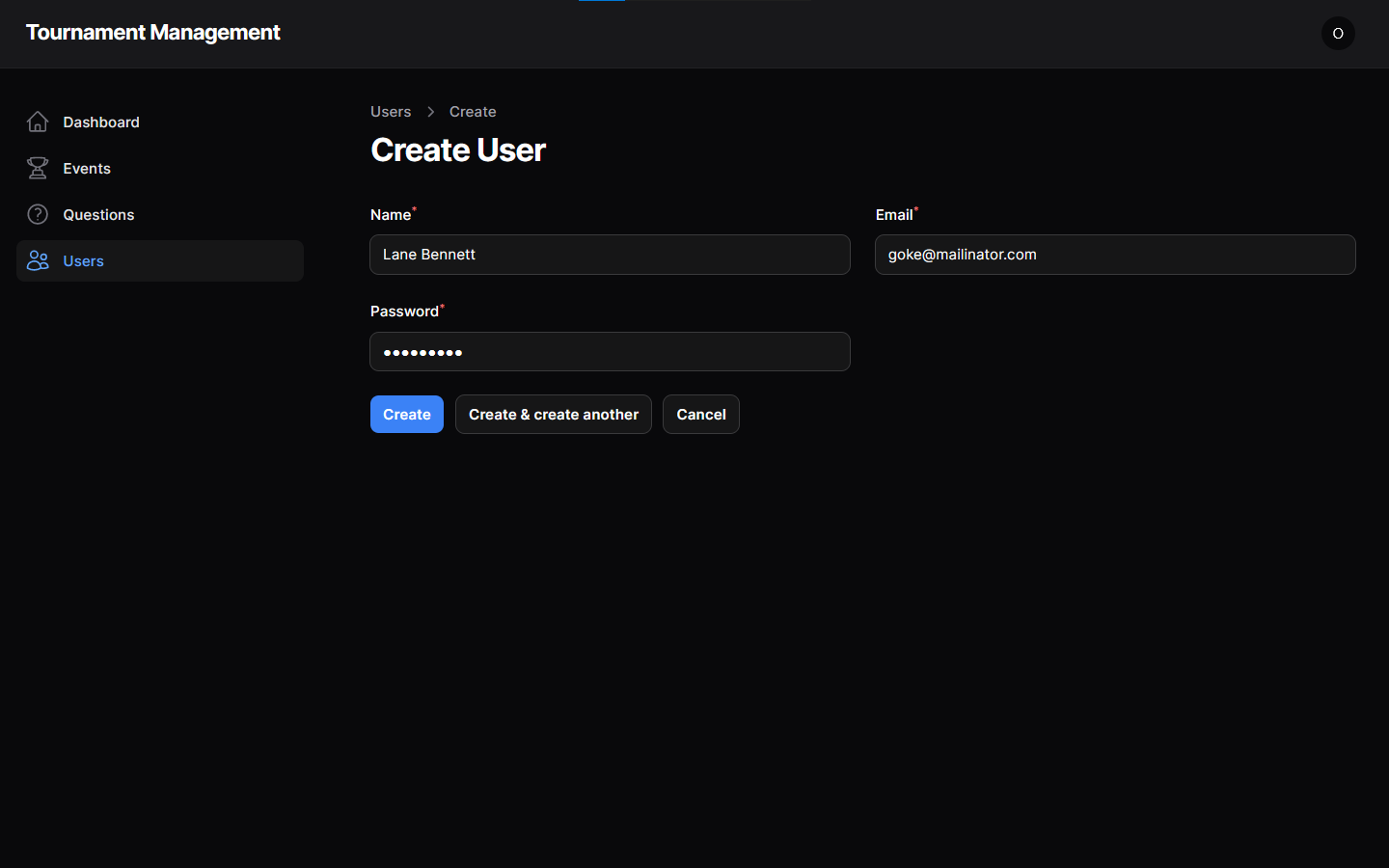
* + - Mass Delete Questions:



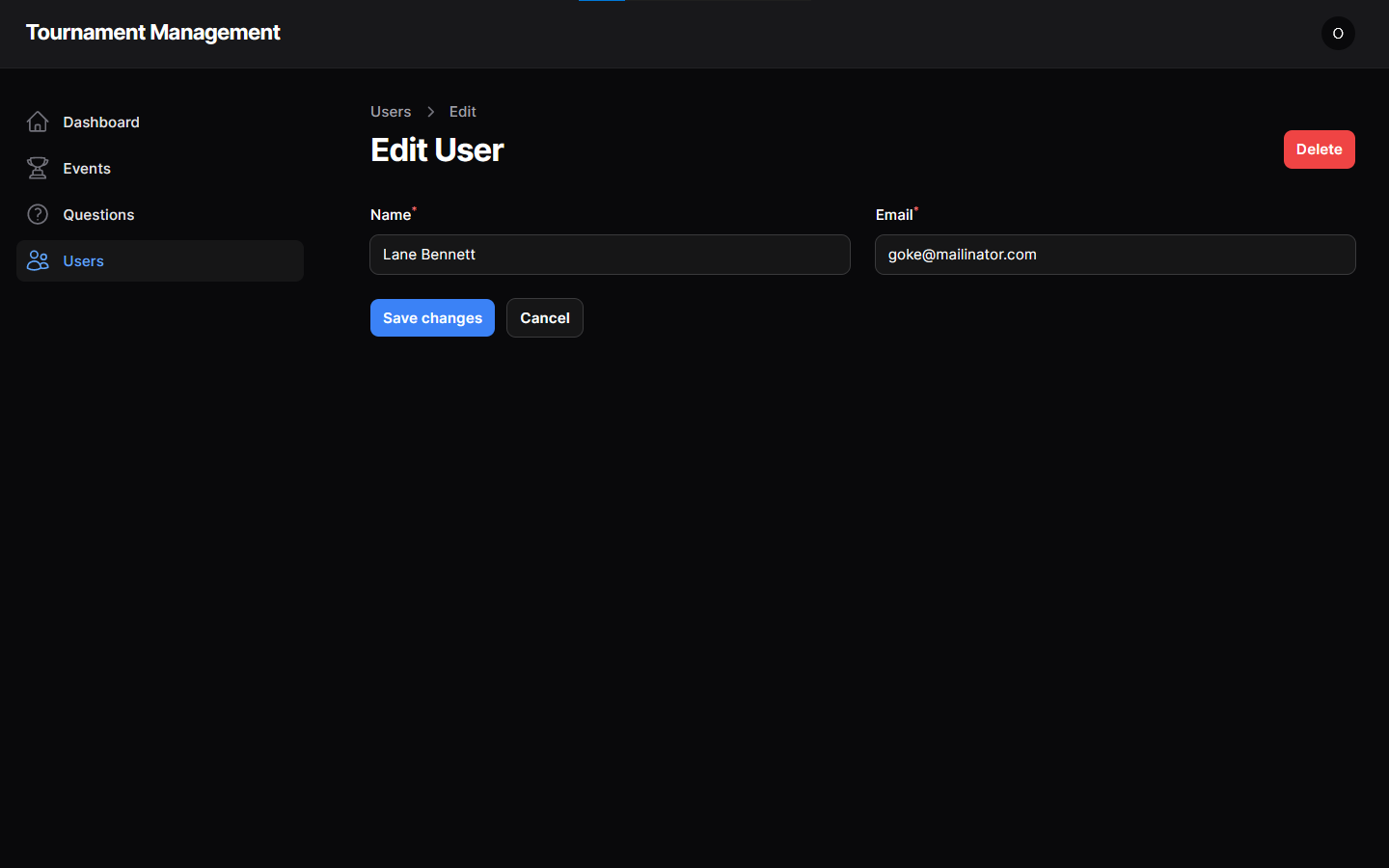
* + Users:
    - View all Users:



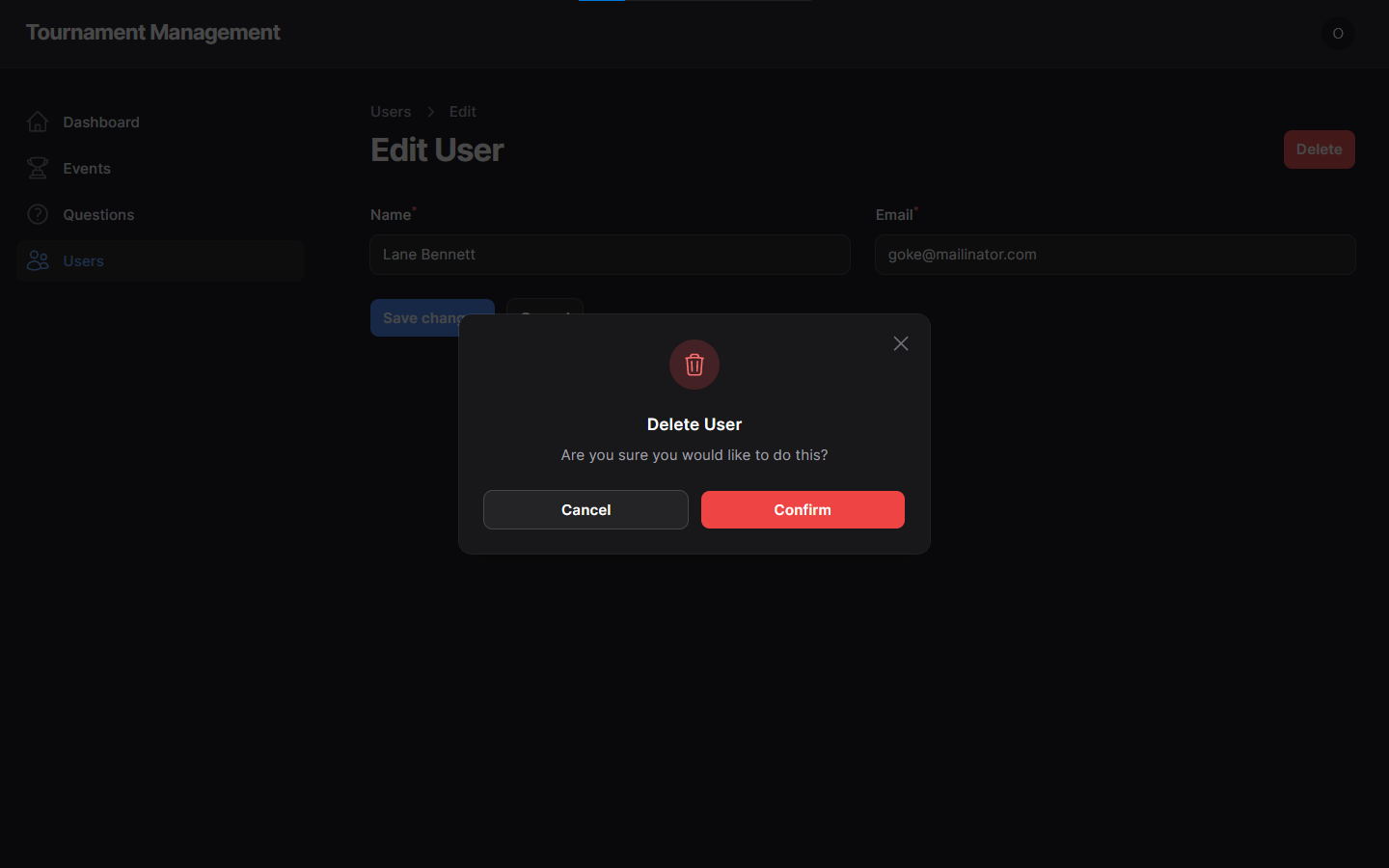
* + - Add new Users:



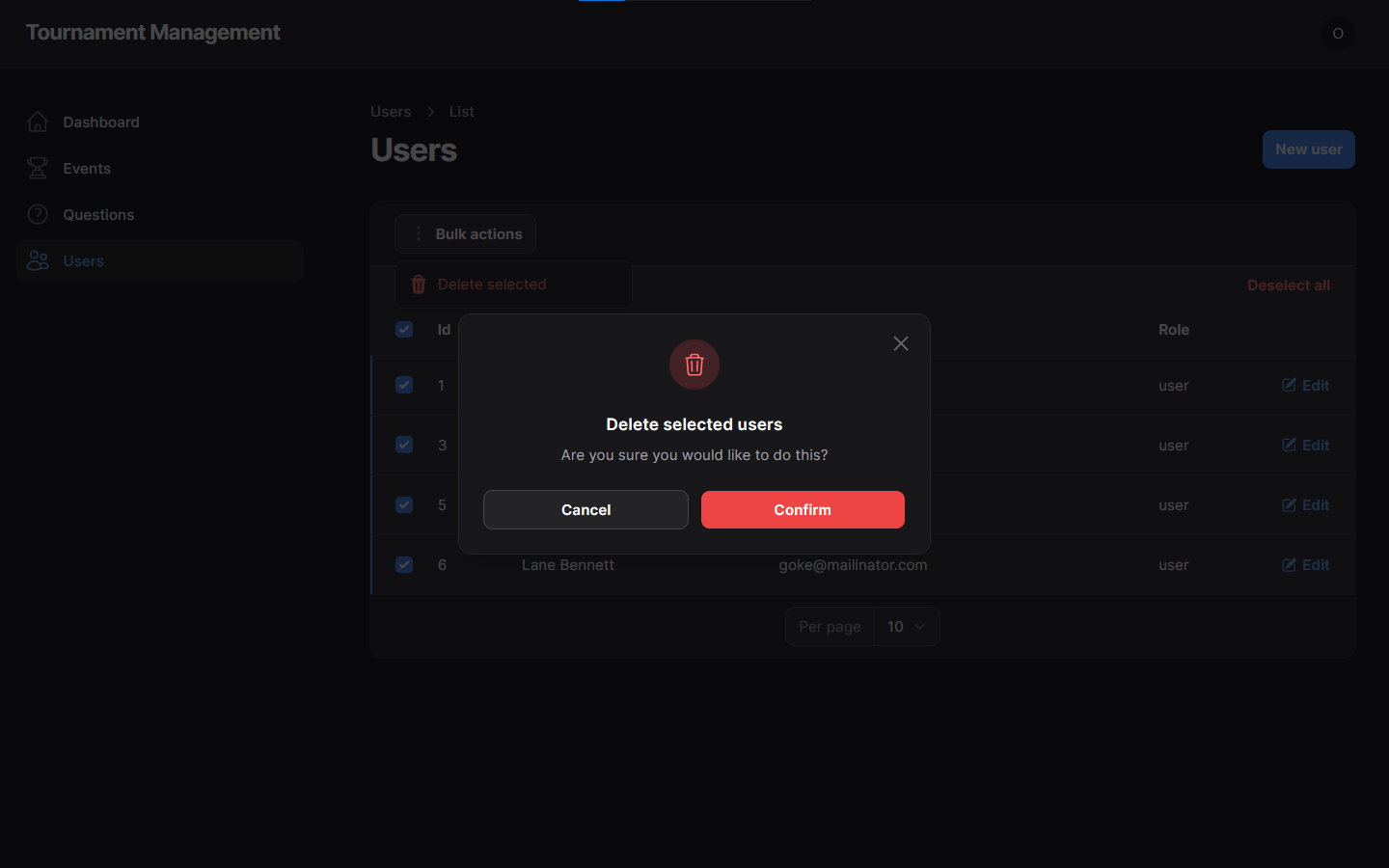
* + - Edit Users:



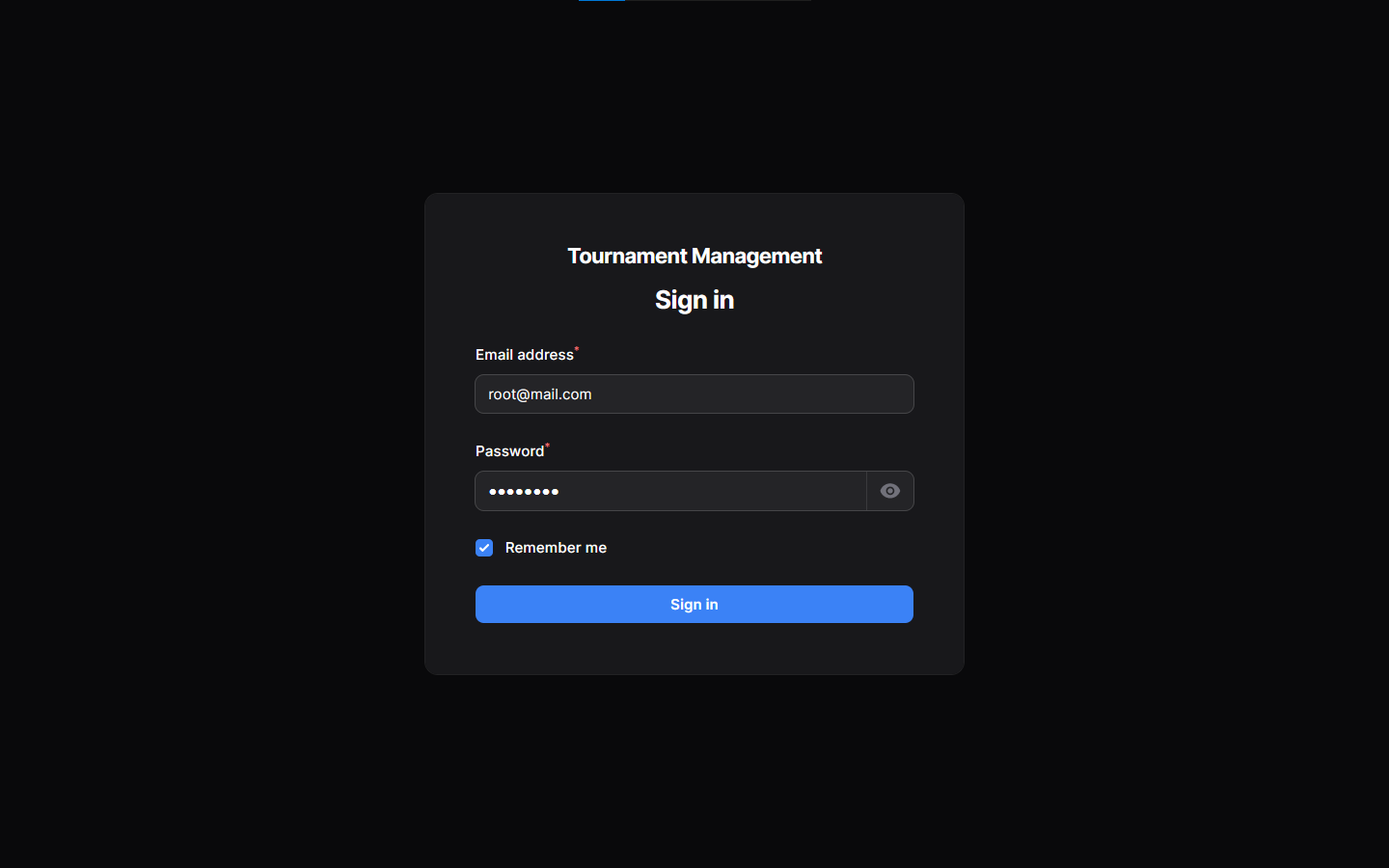
* + - Delete Users:



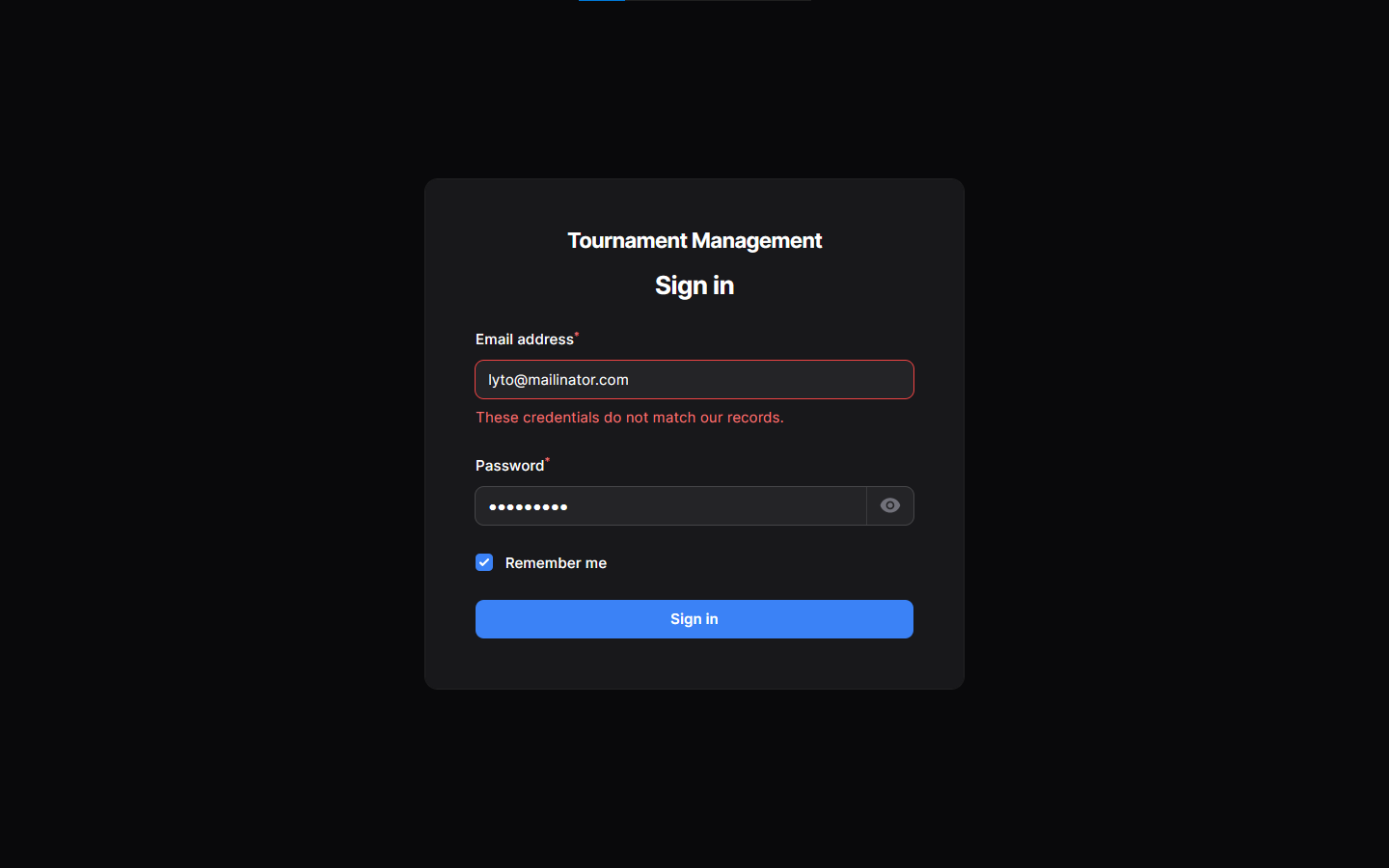
* + - Mass Delete Users:



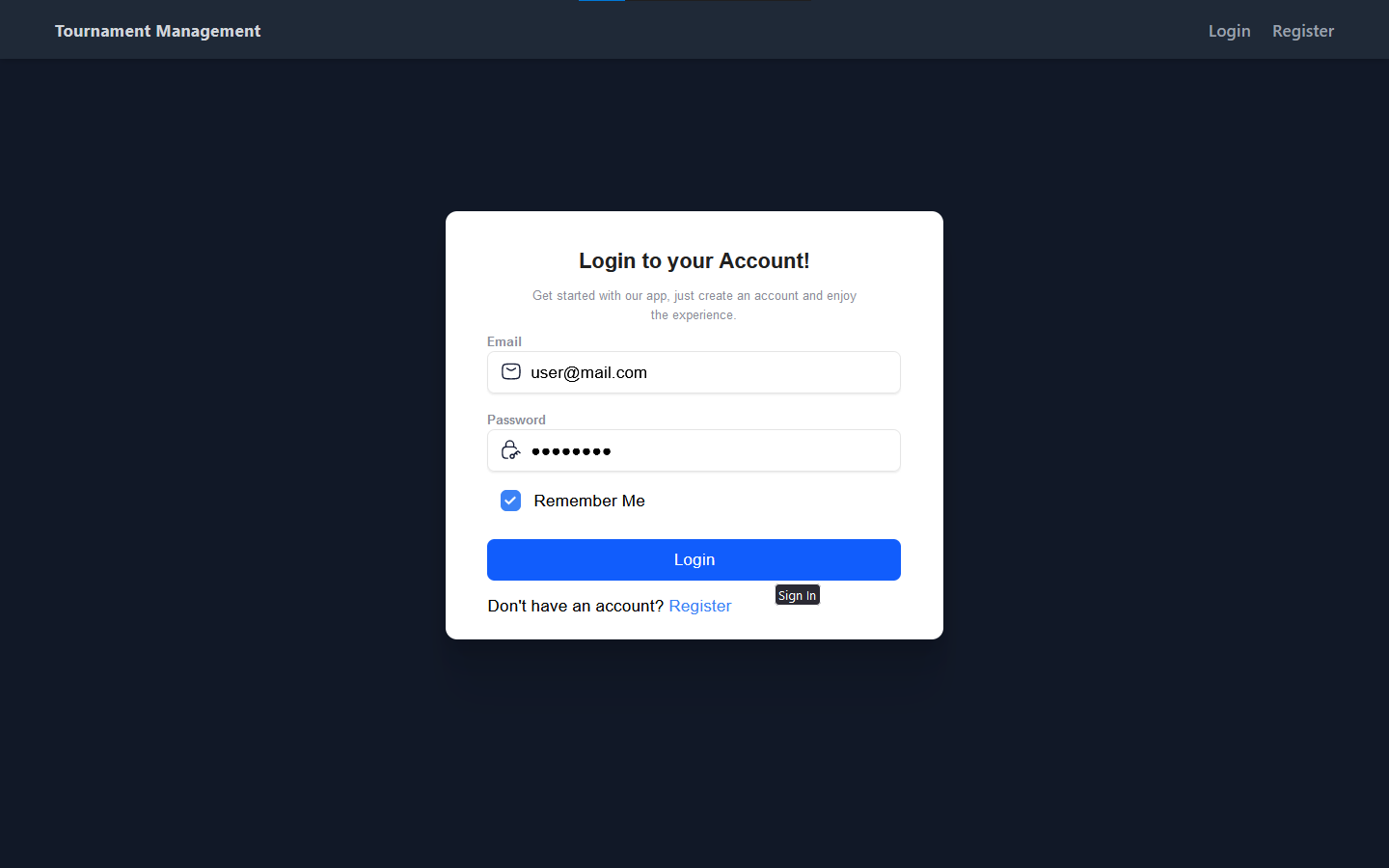
* examples of admin login:



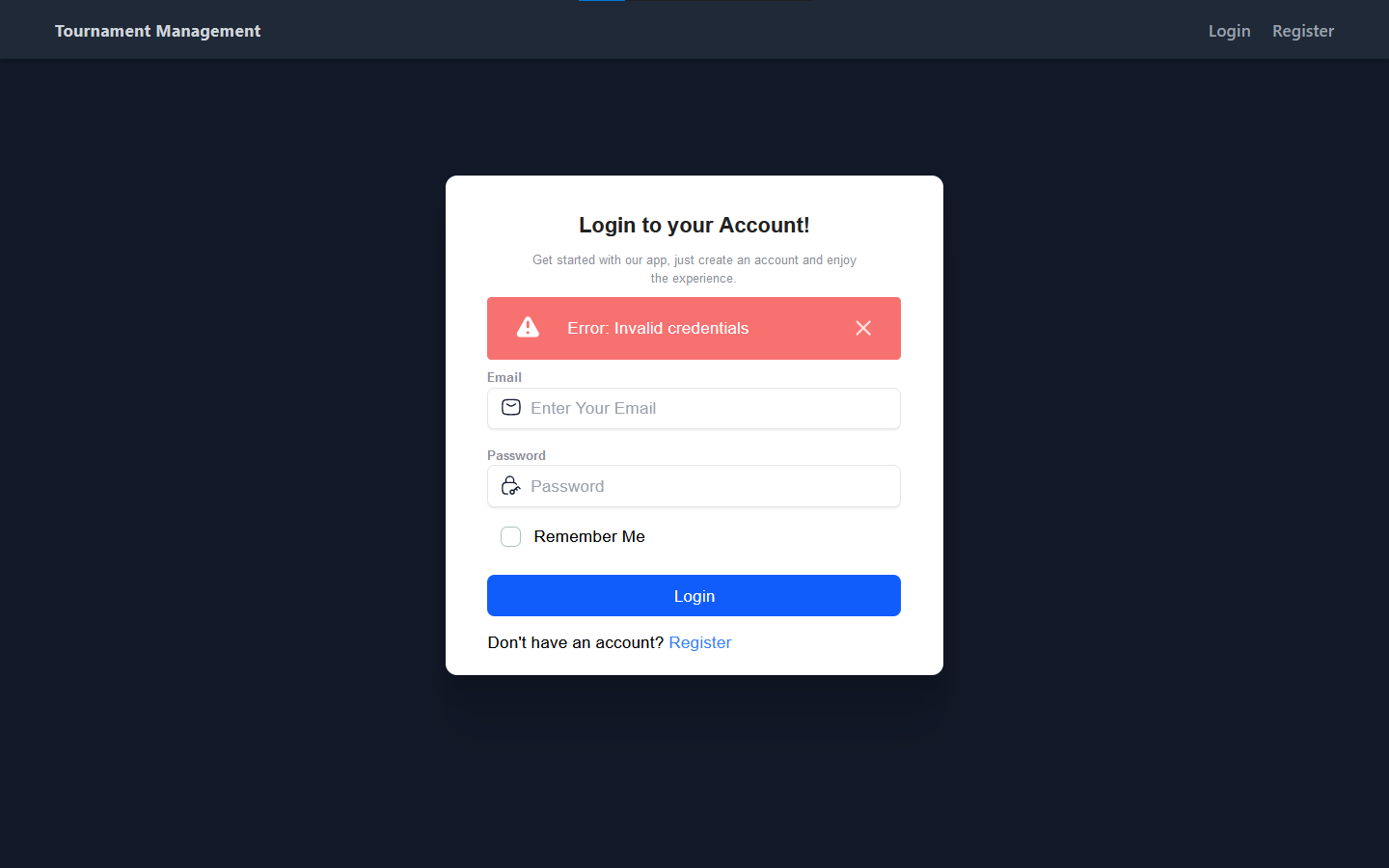
* examples of validation messages:



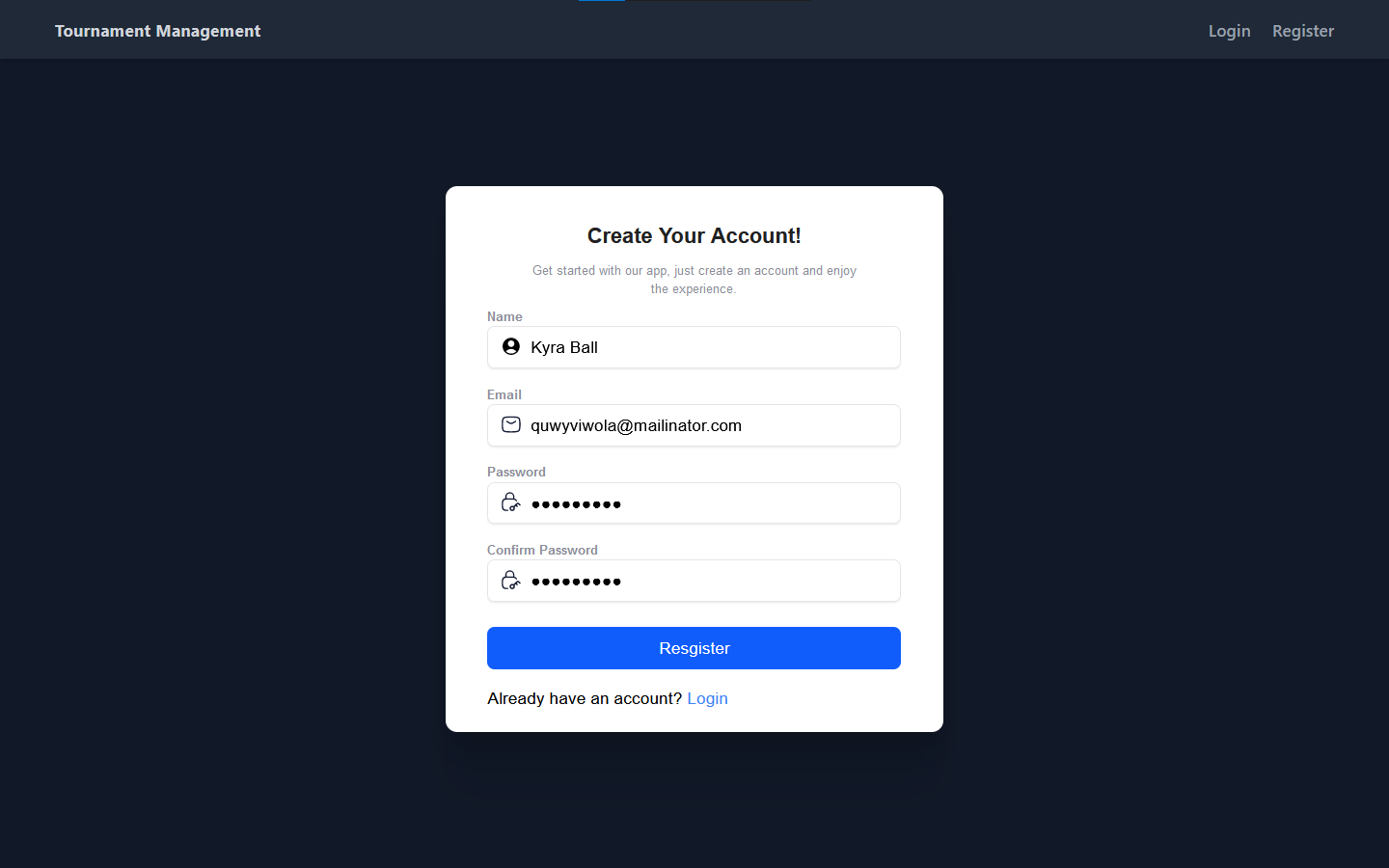
* examples of user login:

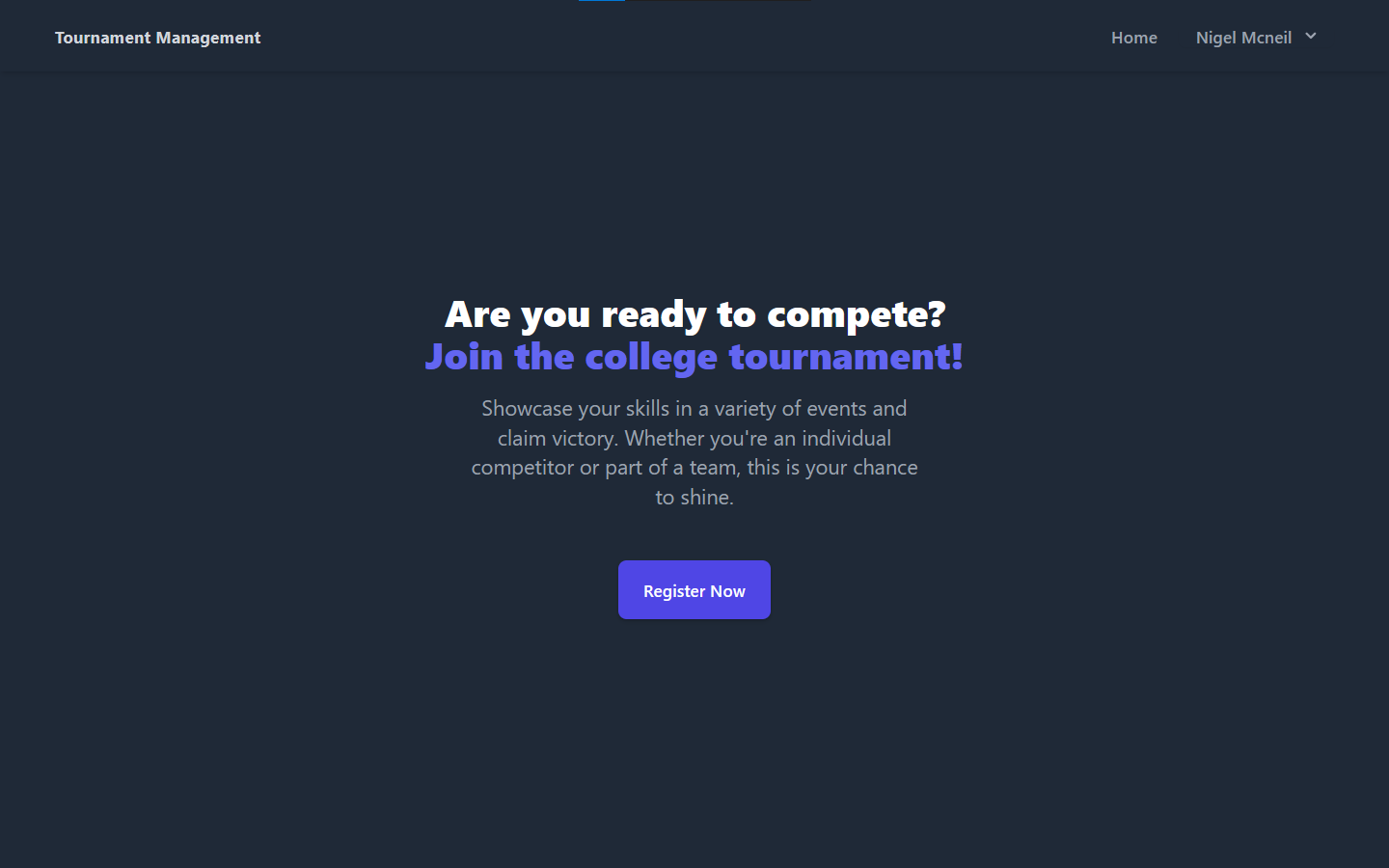
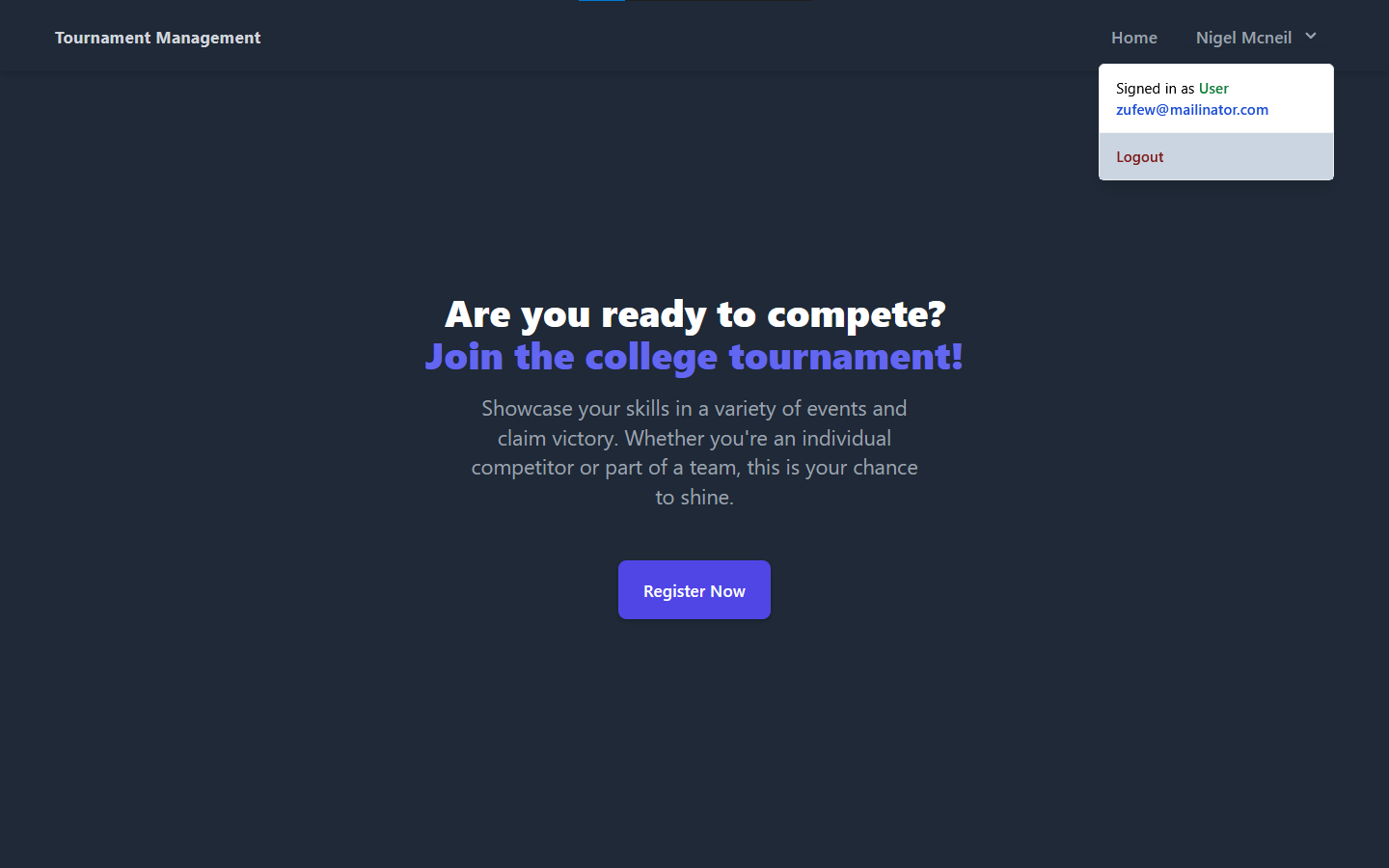


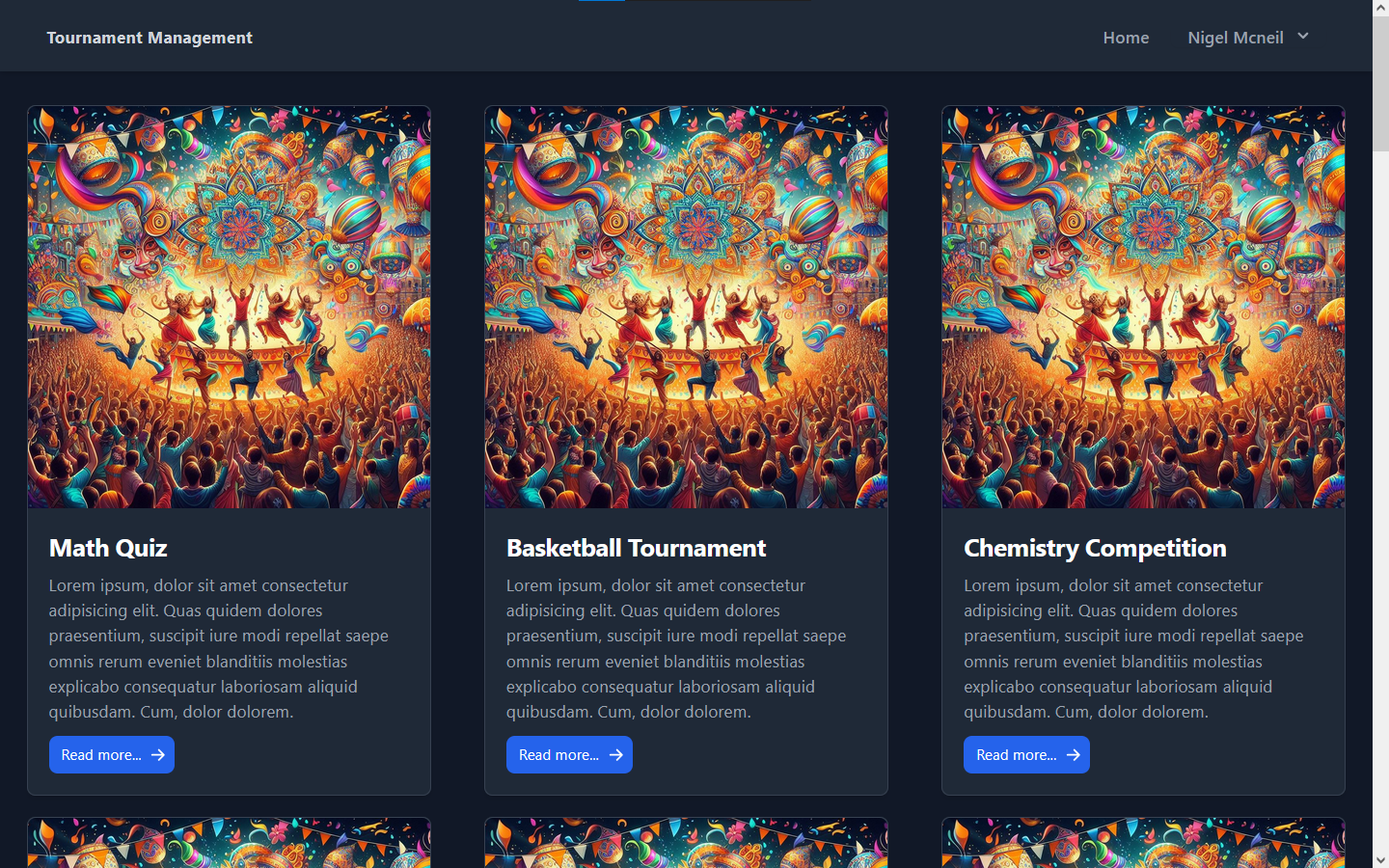
* examples of validation messages:



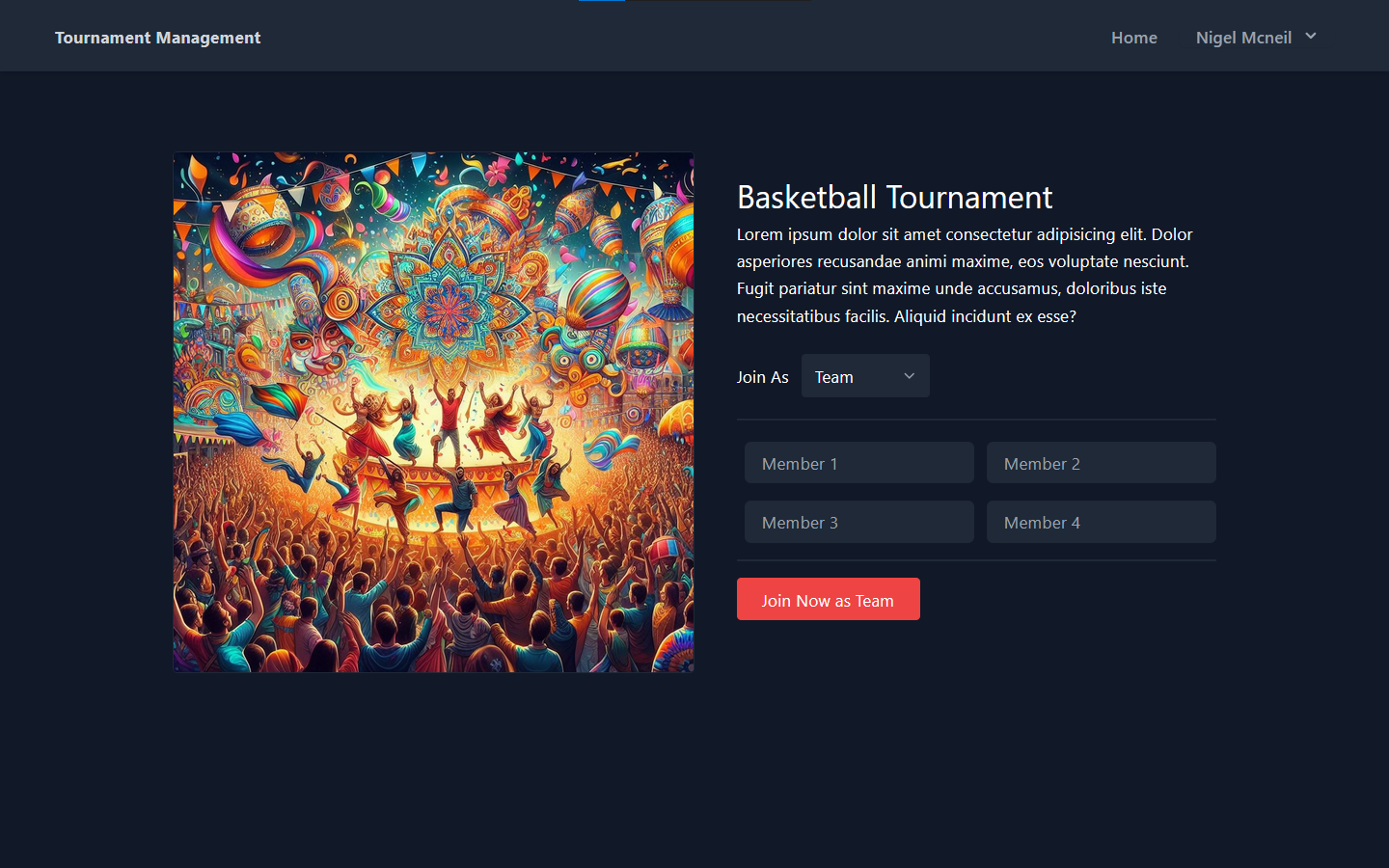
* examples of user registration:



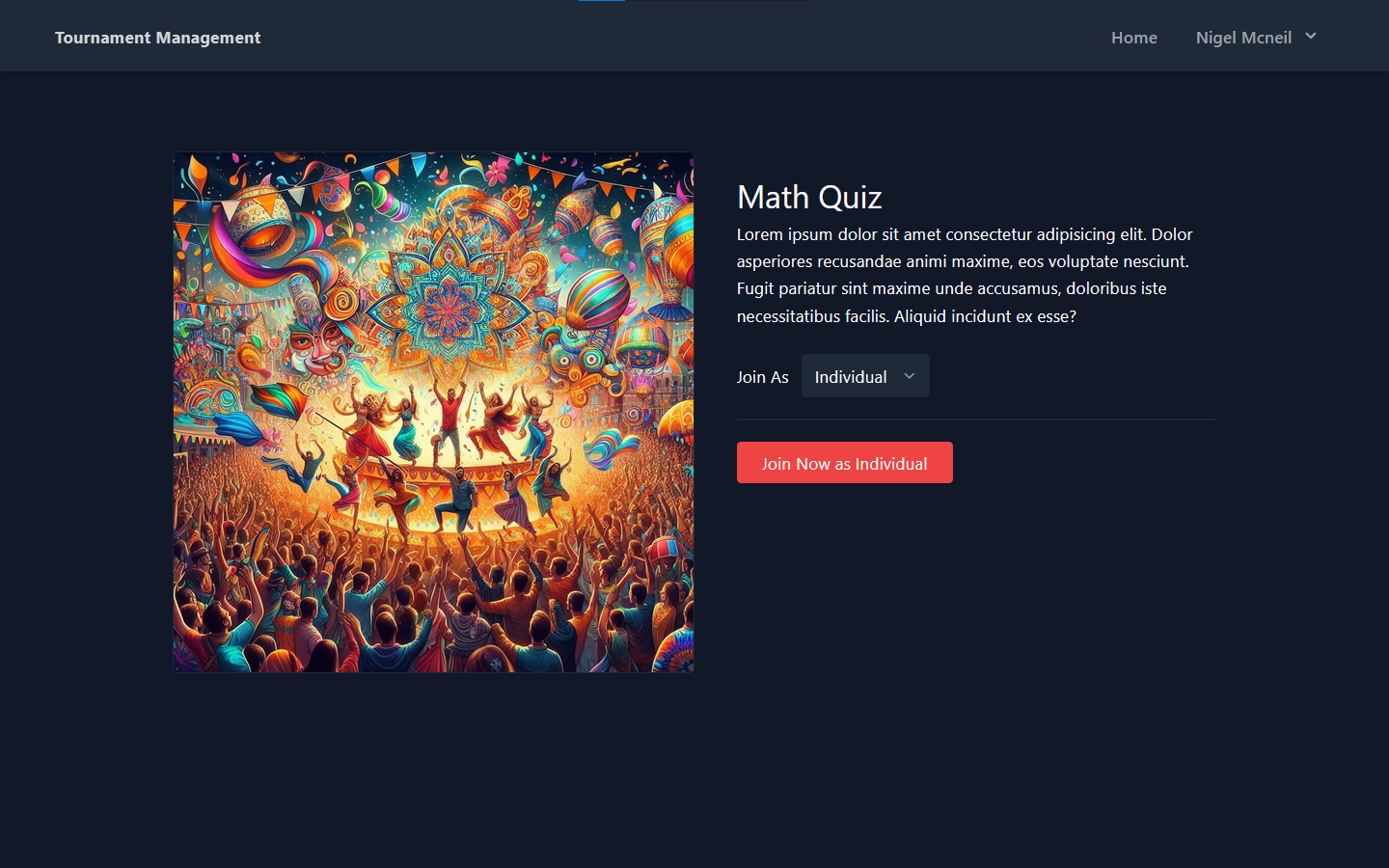
* examples of Landing page:
* 
* 
* examples on the events page:



* examples on the event details page:
  + - Team event:



* + - Individual event:



## Implementation Phase:

### Coding:

#### I used the following technologies for the project:

* HTML (Blade Template)
* CSS (Tailwindcss)
* JavaScript
* PHP (Laravel Framework)
* MySQL
* Git
* GitHub

#### I used the following libraries for the project:

#### Frontend technologies:

* + Tailwindcss: A utility-first CSS framework for rapidly building custom user interfaces.
  + Hero icons: Beautiful hand-crafted SVG icons, by the makers of Tailwind CSS.
  + Laravel Livewire: A full-stack framework for Laravel that provides a simple, yet powerful way to build dynamic UIs.

#### Backend technologies:

* + Filament: A collection of beautiful full-stack components. For designing the admin dashboard.
  + Eloquent: An object-relational mapper (ORM) that makes it enjoyable to interact with your database.
  + the Auth facade: A simple authentication API that provides a variety of common authentication tasks.

#### User Authentication:

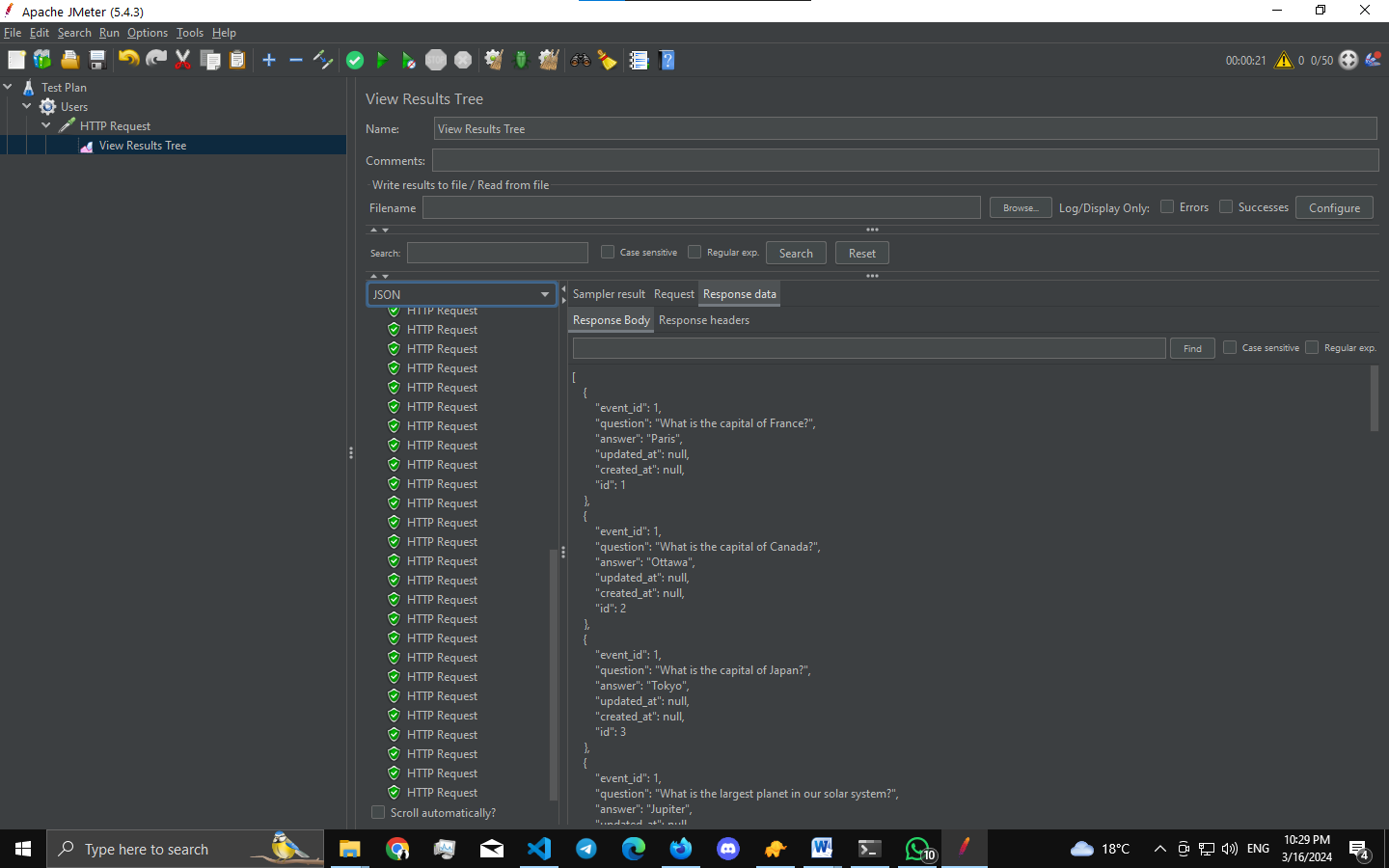
* I have implemented a secure user authentication system in Laravel, utilizing hashed passwords for user security.
* I have utilized Laravel's built-in session-based authentication tokens to ensure secure user authentication.

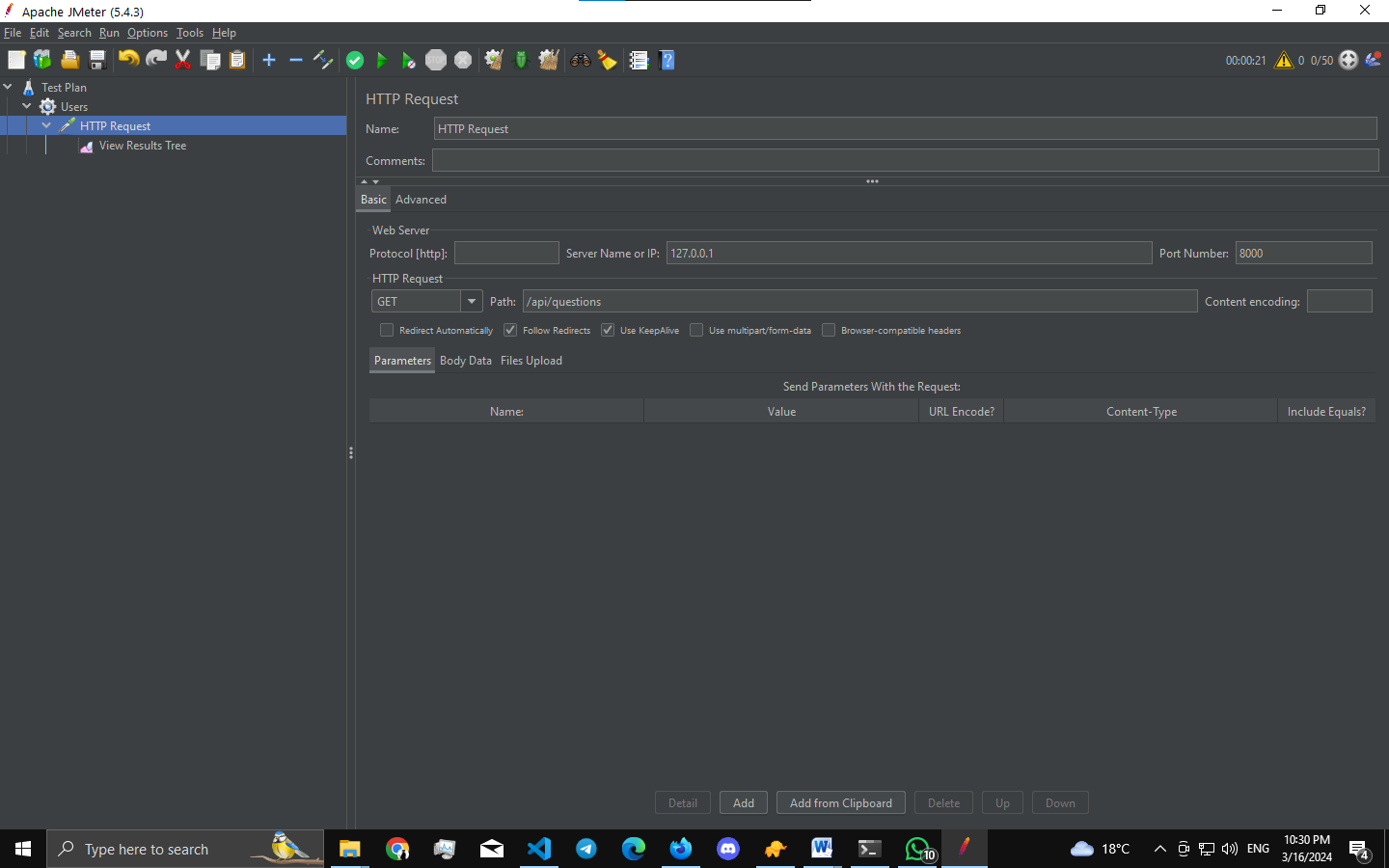
#### User Authorization:

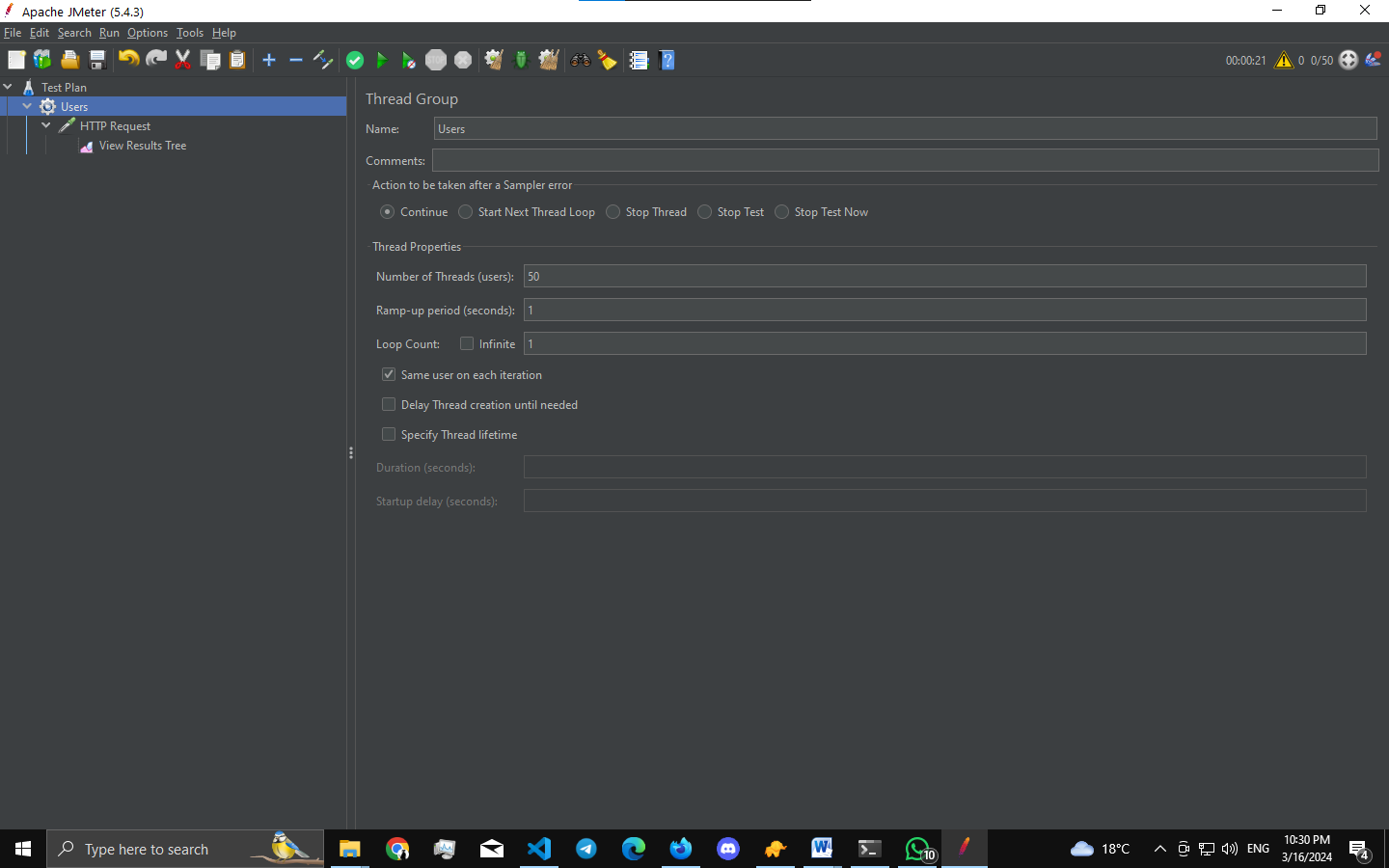
* I have implemented middlewares to protect user and admin routes in the application.
* I have implemented a middleware that ensures only authenticated users with the appropriate roles (user or admin) can access their respective routes. Unauthorized access attempts are redirected to the login page, thereby enhancing the security of the application.
* I have implemented a middleware to check if the user is guest (has logged in or registered) or not.

## Testing Phase:

* Performed integration tests for backend components, including routes, controllers, and database interactions
* Ensured proper integration of Laravel Livewire components with backend APIs in frontend integration tests.
* Evaluated system performance under different loads using tools like JMeter for testing APIs performance.
* examples on testing using JMeter:







## Deployment phase:

* Deploying the app may need additional costs which isn't available with me right now.
* there are lots of ways to deploy laravel apps I will mention two of them:
  + Laravel Vapor: serverless deployment platform for Laravel, powered by AWS. Launch your Laravel infrastructure on Vapor and fall in love with the scalable simplicity of serverless.
  + Envoyer: deploys your PHP applications with zero downtime. Just push your code, and let Envoyer deliver your application to one or many servers without interrupting a single customer.

# Maintenance Phase

### Bug Fixes:

* Continuously monitor the system for any reported bugs or issues.
* Utilize Laravel's debugging tools like Telescope to identify the root causes of problems.
* Implement and test bug fixes using Laravel's testing suite (PHPUnit), ensuring that they do not introduce new issues.

### Performance Monitoring:

* Utilize monitoring tools like Laravel Horizon or third-party services to track the system's performance.
* Identify and address any performance bottlenecks or areas of improvement using Laravel's profiling and optimization techniques.

### Security Updates:

* Stay informed about security updates for Laravel framework and other dependencies used in the project.
* Apply security patches promptly by updating Laravel and its dependencies to the latest versions.

### Database Optimization:

* Analyze and optimize database queries using Laravel's Eloquent ORM to improve overall system performance.
* Implement indexing and caching strategies using Laravel's caching mechanisms and query optimization techniques.

# System Design Document

## Technologies Used:

* Laravel for the backend
* Livewire for frontend interactivity
* Filament for administration dashboard
* MySQL with Eloquent ORM for database interaction
* Tailwind CSS for styling

## System Architecture:

### User Authentication:

* **Task:** Implement user authentication for participants and admins.
* **Algorithm:** Use Laravel's built-in access tokens for token-based authentication.
* **Data Storage:** Store user credentials securely in the MySQL database.

### Tournament Registration:

* **Task:** Allow participants to register as individuals or teams.
* **Algorithm:** Design a registration form that supports both individual and team entries.
* **Data Storage:** Store participant information, team details, and event registrations in the database.

### Team Management:

* **Task:** Create functionality for managing teams and their members.
* **Algorithm:** Implement CRUD operations for teams, allowing participants to create teams.
* **Data Storage:** Store team information and member details in the database.

### Event Management:

* **Task:** Develop a system for creating and managing events.
* **Algorithm:** Design a dynamic form for event creation, considering different event types.
* **Data Storage:** Store event details, including type, participants, and scoring rules in the database.

### Scoring System:

* **Task:** Define and implement a scoring system for events.
* **Algorithm:** Allow flexibility in assigning points based on event types and ranks.
* **Data Storage:** Store event results and points awarded to participants in the database.

### Result Calculation:

* **Task:** Calculate overall tournament results based on individual and team performances.
* **Algorithm:** Sum up individual and team points to determine final rankings.
* **Data Storage:** Store final results in the database for future reference.

### Suggestions for Point System:

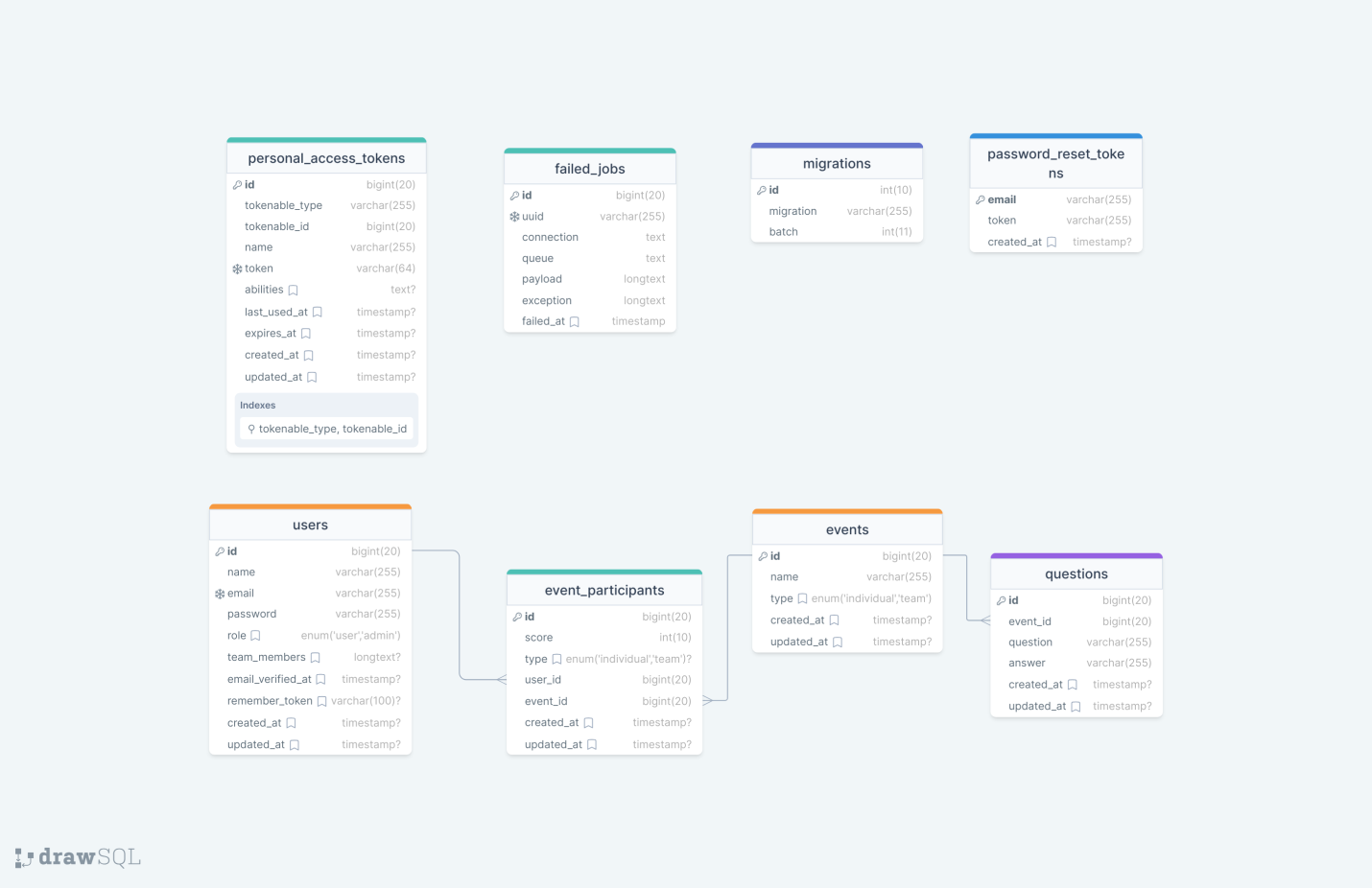
* **Task:** Propose a point system for events.
* **Algorithm:** Consider event difficulty, participation, and performance to allocate points.
* **Data Storage:** Store suggested point systems for each event type in the database.

### Single Event Registration:

* **Task:** Allow participants to register for a single event only.
* **Algorithm:** Implement a registration option that restricts participants to a single event.
* **Data Storage:** Update participant profiles to reflect the chosen event.

## Diagrams:

### Entity Relationship Diagram (ERD):



# Analysis

## Authentication and Authorization:

* **Utilize Laravel access tokens** for secure authentication, providing a stateless mechanism for user identity verification.
* Implement **password hashing** using Laravel's built-in encryption mechanisms to enhance security.
* Employ **role-based access control** to manage permissions for different user types (users, administrators).

## Database Design:

* Choose **MySQL as the database** due to its robustness and compatibility with Laravel's Eloquent ORM.
* Use **Eloquent ORM** to define schema models, ensuring consistency in data storage.
* Consider indexing and caching strategies for efficient data retrieval, especially when dealing with large amounts of participant and event data.

## Scalability:

* Design the system to handle potential future growth in terms of participants, teams, and events.
* Consider load balancing strategies for the Laravel server to distribute incoming requests efficiently.

## Event Management:

* Define a modular event structure that supports both individual and team events.
* Utilize a flexible design to handle different event types within the same system.
* Allow dynamic creation and modification of events, accommodating the college's need for flexibility in event types and rules.

## Scoring System:

* Develop a flexible scoring algorithm that can be easily adjusted based on the college's preferences.
* Allow for real-time calculation and display of points, providing an engaging and competitive environment for participants.

## Single-Event Entry Option:

* Implement a feature that allows participants to enter for one event only.
* Ensure a smooth user experience, aligning data capture and registration processes with the single-event entry option.

## Frontend Design (Livewire and Tailwind CSS):

* Create a responsive and intuitive user interface using **Livewire components** that caters to both participants and administrators.
* Implement client-side routing for a seamless and dynamic user experience.
* Utilize **Tailwind CSS** for styling to ensure consistency and ease of maintenance.

## Backend (Laravel with Eloquent ORM):

* Design RESTful APIs for efficient communication between the frontend and backend using **Laravel's routing system**.
* Implement middleware for authentication and authorization, utilizing **Laravel access tokens**.
* Utilize environment variables for secure configuration management.

## Testing and Debugging:

* Implement a robust testing strategy, including unit tests, integration tests, and end-to-end tests.
* Utilize debugging tools provided by Laravel to identify and address issues during development and deployment.

# Advantage and drawbacks

### Laravel (Backend):

* Advantages:
  + **Eloquent ORM**: Provides a simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table.
  + **MVC Architecture**: Laravel follows the MVC pattern, ensuring clarity between logic and presentation. This design pattern allows changes to be made to the UI without affecting the business logic and vice versa.
* Drawbacks:
  + **Learning Curve**: Laravel comes with its own directory structure which could be overwhelming for new developers.

### Livewire:

* Advantages:
  + **Simplicity**: Livewire allows you to build dynamic interfaces directly from Laravel blade views, simple directives are used in the blade views.
* Drawbacks:
  + **Performance**: Livewire can be slower compared to a solution implemented in Vue.js or React.js.

### Filament:

* Advantages:
  + **Admin Panel**: Filament is a set of tools for rapidly building beautiful TALL stack interfaces.
* Drawbacks:
  + **Limited Community Support**: As Filament is relatively new, the community support might not be as extensive as other established packages.

### MySQL using Eloquent ORM:

* Advantages:
  + **ACID Compliance**: MySQL is ACID compliant which ensures the reliability of transactions.
  + **Popularity**: MySQL is widely adopted, and there exists a large community of developers who can help answer questions or resolve issues.
* Drawbacks:
  + **Scaling**: While MySQL can be scaled, it's not as easy to do as with some NoSQL databases.

### Autoprefixer, PostCSS, PostCSS-Nesting:

* Advantages:
  + **Browser Compatibility**: Autoprefixer adds vendor prefixes to your CSS so that styles work across different browsers.
  + **Advanced Transformations**: PostCSS allows you to use tomorrow’s CSS syntax today, and PostCSS-Nesting enables nested CSS rules.
* Drawbacks:
  + **Additional Build Steps**: These tools add additional steps to your build process.

### Axios:

* Advantages:
  + **Simplicity**: Axios provides a simple API for making HTTP requests and works great with Vue.js.
* Drawbacks:
  + **Overhead**: For simple projects, the additional features of Axios might be unnecessary.

### Laravel Vite Plugin, Tailwind CSS, and Vite:

* Advantages:
  + **Performance**: Vite provides a faster and leaner development experience for modern web projects. It works great with Vue.js and Laravel.
  + **Utility-First**: Tailwind CSS is a utility-first CSS framework which is highly customizable and works very well with modern JavaScript frameworks.
* Drawbacks:
  + **Learning Curve**: There might be a learning curve associated with understanding these tools and how to configure them properly.

## Integration:

* **Laravel + Livewire + Filament**: The combination of Laravel, Livewire, and Filament can provide a seamless stack for building web applications.
* **Eloquent ORM with MySQL**: Eloquent ORM simplifies interaction with MySQL, providing a convenient and expressive way to model data.

## Pre-defined Code:

* **For user authentication**, you can leverage existing libraries such as Laravel's built-in authentication, saving development time.
* **Various Laravel packages** are available for handling HTTP requests, user authentication, and event handling.

## Assets:

* **Livewire Components**: Utilize existing Livewire component libraries for UI elements and styling.
* **Sports and Academic Challenges**: If predefined challenges/events exist, they can be integrated directly into the system, possibly as JSON files.

# Review and Feedback

## Initial Design Overview

* Presented the initial design to the team, highlighting the use of Laravel for the backend, Livewire for dynamic interfaces, Filament for admin interfaces, MySQL with Eloquent ORM for database interactions, and Laravel Vite Plugin, PostCSS, PostCSS-Nesting, Tailwind CSS, and Vite for frontend development.
* Emphasized the flexibility and scalability of the chosen technologies for handling the tournament's diverse requirements.

## Functional Flow

* Walked through the entire functional flow of the application, detailing how participants can register as individuals or teams, the structure of the tournament, and the scoring system based on event rankings.
* Discussed the seamless integration of team and individual events, ensuring a smooth user experience.

## Security Measures

* Highlighted the use of Laravel's built-in authentication and security features, emphasizing the importance of data security, especially when handling user information and tournament results.

## User Experience

* Discussed the user interface implemented with Livewire and Tailwind CSS, focusing on the user experience and intuitive navigation for both participants and administrators.
* Demonstrated the responsive design to ensure accessibility on various devices.

## Scalability and Performance

* Addressed potential scalability concerns, particularly considering the dynamic nature of the tournament with varying numbers of individual participants and team sizes.
* Discussed strategies for optimizing database queries and ensuring efficient server-side processing with Laravel and Eloquent ORM.

## Feedback Collection

* Encouraged team members to provide feedback on the overall design, functionality, and user experience.
* Actively sought suggestions for improvements and alternative approaches to any potential bottlenecks or challenges.

## Points System Discussion

* Initiated a discussion on the undecided points system for events, encouraging team members to share their thoughts on fair and engaging scoring methods.
* Collected feedback on proposed point distributions and explored creative alternatives.

## Single Event Entry Consideration

* Discussed the possibility of entering for a single event only and gathered opinions on how to implement this feature without disrupting the overall tournament structure.

## Documentation Review

* Ensured that the project documentation was comprehensive, covering technical aspects, API documentation, and user guides.
* Encouraged feedback on the clarity and completeness of the documentation.

## Iterative Refinement

* Incorporated feedback into iterative design refinements, addressing any identified areas for improvement.
* Maintained an open line of communication to discuss design decisions and make adjustments based on collective input.

# Test Plan

## 1. Unit Testing:

### a. Frontend (Livewire, Tailwind CSS, Vite):

* **Components**: Test each Livewire component for rendering, state management, and proper handling of user interactions.
* API Integration: Ensure that API calls return the expected responses.

### b. Backend (Laravel, Eloquent ORM):

* **Routes**: Test each API endpoint for proper request handling and response.
* **Controllers**: Verify that controllers handle business logic correctly.
* **Database Operations**: Test CRUD operations on the database using Eloquent ORM.
* Authentication and Authorization:
  + **Laravel Access Tokens**: Test token generation, verification, and expiration.

## 2. Integration Testing:

### a. Frontend-Backend Integration:

* Ensure that Livewire components interact correctly with backend APIs.

### b. Database Integration:

* Verify that data is properly stored and retrieved from the MySQL database.

## 3. End-to-End Testing:

### a. User Flows:

* Test common user flows, such as user registration, event creation, event participation, etc.
* Verify that users can enter as individuals or teams, and events are recorded accurately.

### b. Security:

* Test for security vulnerabilities, such as SQL injection, cross-site scripting, and ensure that sensitive information is properly secured.

## 4. Performance Testing:

### a. API Performance:

* Test the API endpoints under various loads to ensure optimal performance.

### b. UI Responsiveness:

* Verify that the UI remains responsive, even under heavy loads.

## 5. User Acceptance Testing (UAT):

### a. College Requirements:

* Ensure that the system meets the college's expectations for tournament management.
* Test the scoring system for events and suggest improvements if necessary.

## 6. Documentation:

### a. Error Logging:

* Implement proper error logging mechanisms throughout the application.
* Document any errors encountered during testing, including their causes and suggested repairs.

### b. Test Results:

* Document the results of each test, including successful tests and any identified issues.

## 7. Feedback Loop:

### a. Communicate with Stakeholders:

* Regularly communicate with the college to gather feedback on the system during testing.
* Incorporate any feedback into the testing process.

# Repairing the Errors

## Identifying and Documenting Errors

* Reviewed test reports and identified the errors encountered during testing.
* Documented each error with a clear description, including steps to reproduce, expected behavior, and observed issues.

## Prioritizing and Categorizing Errors

* Prioritized errors based on their severity and impact on the overall functionality.
* Categorized errors into frontend and backend issues for a more focused approach.

## Version Control

* Ensured that the codebase was under version control using a tool like Git to track changes.

## Backend Repairs

* Addressed backend issues related to Laravel, Eloquent ORM, and MySQL.
* Applied necessary fixes to APIs, ensuring proper data flow and security measures.
* Documented changes made to the backend codebase, including updated code snippets and explanations.

## Frontend Repairs

* Fixed frontend errors related to Livewire components and their interactions.
* Ensured proper rendering of UI elements, data binding, and user interactions.
* Documented changes made to the frontend codebase, providing updated Livewire component snippets and explanations.

## Revising Authentication Mechanism

* Verified and improved the implementation of Laravel's built-in authentication for secure user authentication.
* Checked and fixed issues related to user authorization and access control.

## Database Fixes

* Reviewed and corrected database interactions using Eloquent ORM to ensure data consistency and integrity.
* Applied necessary changes to resolve issues related to data retrieval and storage.

## Testing and Debugging

* Conducted thorough testing after implementing the repairs to ensure the elimination of identified errors.
* Used debugging tools to trace and resolve any remaining issues.

## Documentation of Repairs

* Created comprehensive documentation detailing each repair made, including code changes, explanations, and potential impacts on other parts of the system.
* Ensured that the documentation is clear, concise, and accessible for future reference.

## Retesting

* Conducted a series of retests to validate the effectiveness of the repairs.
* Documented the results of retesting, including any new issues identified and successfully addressed.

# Documented Errors

## Data Sharing Between Components

* **Reasons for the Error**:
  + Components might not be properly communicating due to incorrect implementation of state management or data passing in Livewire.
* **Suggestions for Repair**:
  + Ensure that you are managing state correctly within your Livewire components. Livewire offers a simple and straightforward way to manage state within its components.
  + Double-check the data binding in your Blade views. Livewire uses a simple syntax for two-way data binding between Blade views and Livewire components.
  + Utilize Laravel's service container appropriately for handling dependencies and services. This can help ensure that your components have access to the resources they need.

## Linking User Information Across Components

* Reasons for the Error:
  + Inadequate authentication implementation.
  + Insufficient use of context or global state for user information.
* Suggestions for Repair:
  + Implement a robust authentication system using Laravel's built-in authentication.
  + Utilize Laravel's service container or a global state management solution to store and share user information across components.
  + Ensure that the user data is properly fetched and stored upon authentication.

# Suggestions for Future Reference

## Centralized State Management

* Implementation:
  + Utilize Laravel's service container for centralized state management.
  + Design a state structure that effectively caters to the needs of your application.

## Consistent Authentication

* Implementation:
  + Ensure that user authentication is handled consistently across all relevant components.
  + Use a secure authentication method like Laravel's built-in authentication.

## Effective Use of Laravel Features

* Implementation:
  + Familiarize yourself with Laravel's service container and use it appropriately.
  + Understand the lifecycle methods and apply them as needed for efficient component behavior.

## Documentation and Comments

* Implementation:
  + Maintain clear and concise documentation for each component and functionality.
  + Use comments to explain complex logic or functionalities for future reference.

# Conclusion

* The Tournament Management System has reached an 85% completion stage, with a commitment to addressing identified errors and achieving 100% in the near future. Leveraging Laravel, Livewire, Filament, MySQL with Eloquent ORM, Autoprefixer, Axios, Laravel Vite Plugin, PostCSS, PostCSS Nesting, Tailwind CSS, and Vite, the project demonstrates meticulous planning, effective design, and successful implementation.
* Key highlights include:
* Robust Testing: Thorough unit, integration, end-to-end, and performance testing ensured the system's reliability and functionality.
* Comprehensive Documentation: Detailed documentation covering system architecture, API documentation, user guides, and error logs facilitates seamless project management and future enhancements.
* Thoughtful Deployment: The deployment phase was carefully executed, ensuring smooth transitions and minimal downtime.
* The system design document provides insights into the chosen technologies, system architecture, and scalability considerations, laying a strong foundation for future enhancements and expansions. The review and feedback processes, along with a detailed test plan, ensured continuous improvement throughout the development lifecycle.
* Moving forward, the project team is dedicated to:
* Refining Existing Features: Addressing identified errors and optimizing existing features for improved performance and user experience.
* Implementing Single-Event Entry Option: Adding functionality to allow participants to enter for individual events only, enhancing flexibility and user engagement.
* Resolving Outstanding Issues: Addressing any remaining issues to ensure the system meets the college's event management needs seamlessly.
* The commitment to reaching 100% completion underscores the team's dedication to delivering a robust Tournament Management System aligned with college event management requirements.