# Project Description

Tournament directors have long needed a solution that will streamline the organization and management of sport events. MatchScore aims to solve that problem by providing all the tools required.

## Features:

* Organizing events
  + Match– a one-off event
  + Tournament – knockout or league format
* Managing events
  + Mark the score
  + Change the date
  + Update the participants
* Player profiles
  + Name, Country, Sports Club
* Score tracking
  + Tournament and player statistics
* Registration
  + Allows event organization and management (if approved)
  + Associate with player profile

# Functional Requirements

### Match

* When it will be played (must)
* Participants – at least two (must) or more (should)
* Match format
  + Time limited (must) – (e.g., duration of 60 minutes)
  + Score limited (must) – (e.g., first to 9 points)

### Tournament

* Participants (must)
* Title (must)
* Matches that are part of it (must)
* Format
  + Knockout (must)
  + League(must) – requires scoring for loss, draw and, win
* Tournament match format (must)
* Prize for top finishers (should)

### Player Profiles

* Full Name (only letters, spaces, dashes) (must)
* Country (must)
* Sports Club (should)

### Registered Users

* Email (must)
* Password (must)
* Option to associate with a player profile (must)

## Create a tournament

Requires a list of participants, containing their full names, and format.

1. Example #1 - create a **knockout** tournament with Alice Green, Bob Brown, Chuck Bing and Don Black.
   1. The system generates the following tournament scheme and **randomly** assigns participants to matches:

Diagram

Description automatically generated

* 1. The system tries to link any of the four participants to an existing profile by their full name. If a profile is not found, one will be automatically created with no country and no sports club.
  2. The system automatically updates the next match when a new score is added:

Diagram

Description automatically generated

1. Example #2 – create a **league** tournament with Alice Green, Bob Brown, Chuck Bing and Don Black. **Scoring** is 0 pts for loss, 1 for draw, 2 for win
   1. The system calculates that at least three rounds are required and randomly creates matches for each round: Diagram, table

      Description automatically generated
   2. After the first round, after scoring is added for the following matches:
      * + Bob Brown – Don Black 1 : 2;
        + Alice Green – Charlie Bing 3 : 3;

The intermediate standings are as follows:

* + - * Don Black - 2 pts
      * Alice Green - 1 pts
      * Charlie Bing - 1 pts
      * Bob Brown - 0 pts

## Manage a player profile

* Automatic creation – when creating a tournament, if a participant cannot be matched to an existing profile (must)
* Manual creation – by admin or director (should)
* Directors can edit any player profile if not linked to a user (should)
* If linked to a user, only they can edit it (should)

## Registration

* Anyone can register with an email (unique in the system) and a password (must)
* After registration
  + Promote-to-director request – if approved by admin, the registered user can organize and manage tournaments (must)
  + Link-profile request – if approved by admin, the player profile is associated with the registered user (must)
    - Receives email notification when added to a tournament (should)

## Administration

* Admin user is predefined in the system (must)
* Can view and approve or decline promote-to-director and link-profile requests (must)
* Can manage any resource – CRU (should)
  + Only admins can delete (could)

## Third-Party Service

* Integrate with <https://dev.mailjet.com/email/guides/send-api-v31/> for email notifications (should)
* Notifications:
  + Promote-to-director request approved/declined (should)
  + Link-profile request approved/declined (should)
  + Added to tournament (should)
  + Added to match (should)
  + Match changed – date and or score (could)

## Score and Tournament Tracking

* Information is available without any authentication (must)
* View any past, present or future tournament (must)
* View any match (must)
* View any player profile (should)
  + Tournaments played / won (should)
  + Matches played / won (should)
  + Most often played opponent (should)
  + Best opponent – win/loss ratio (should)
  + Worst opponent – win/loss ratio (should)

## REST API

Provide a RESTful API that supports the full functionality of the system. (must)

# Technical Requirements

## General

* Follow REST API design [best practices](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy) when designing the REST API (see Appendix)
* Use tiered project structure (separate the application in layers)
* You should implement proper exception handling and propagation
* Try to think ahead. When developing something, think – “How hard would it be to change/modify this later?”

## Database

The data of the application must be stored in a relational database. You need to identify the core domain objects and model their relationships accordingly. Database structure should avoid data duplication and empty data (normalize your database).

Your repository must include two scripts – one to create the database and one to fill it with data.

## Git

Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first and the people who contributed. Contributions from all team members must be evident through the git commit history! The repository must contain the complete application source code and any scripts (database scripts, for example).

Provide a link to a GitLab repository with the following information in the README.md file:

* + Project description
  + Link to the hosted project (if hosted online)
  + Instructions on how to set up and run the project locally
  + Images of the database relations (must)

## Optional Requirements

Besides all requirements marked as should and could, here are some more *optional* requirements:

* Integrate your project with a Continuous Integration server (e.g., GitLab’s own) and configure your unit tests to run on each commit to your master branch
* Host your application's backend in a public hosting provider of your choice (e.g., AWS, Azure, Heroku)
* Use branches while working with Git

# Teamwork Guidelines

Please see the Teamwork Guidelines document.

# Appendix

* + [Guidelines for designing good REST API](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy)
  + [Guidelines for URL encoding](http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/)
  + [Always prefer constructor injection](https://www.vojtechruzicka.com/field-dependency-injection-considered-harmful/)
  + [Git commits - an effective style guide](https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn)
  + [How to Write a Git Commit Message](https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/)

# Legend

* Must – Implement these first.
* Should – if you have time left, try to implement these.
* Could – only if you are ready with everything else give these a try.