Sofia Reia fc59875
Diogo Sousa fc59792
Renan Silva fc59802

Relatório CSS: Fase 1

Arquitetura em camadas:

Para a nossa abordagem, optámos por uma arquitetura de 3 camadas, com:

Camada de Serviços:

- Contém os **Rest Controllers** (anotados com ***@RestController***), com a responsabilidade de definir os endpoints, funcionalidades expostas ao exterior, e de coordenar os Mappers e os Serviços, assegurando que qualquer parâmetro que entre num Serviço tenha campos válidos.
- Os **Mappers** (anotados com *@Mapper* da dependência MapStruct) têm a responsabilidade de converter as Entidades para DTOs e vice-versa de modo a não expor toda a informação sobre as entidades para o exterior.

Camada de Lógica de Negócios:

- Contém os **Serviços** (anotados com ***@Service***), que têm a responsabilidade de realizar todas as verificações relacionadas com a Lógica de Negócio e coordenar as mesmas. Contém também as **Entidades** (anotadas com ***@Entity***), cuja responsabilidade é, para além de criar o Schema da BD, ter um conjunto de operações relacionadas com a sua Lógica de Negócio.

- Mais perto da Camada de Dados, está a Camada de Persistência de Dados, que contém os **Repositórios** (anotados com ***@Repository***) e trata das interações com a BD, assim como a conversão entre OOP e o Modelo Relacional.

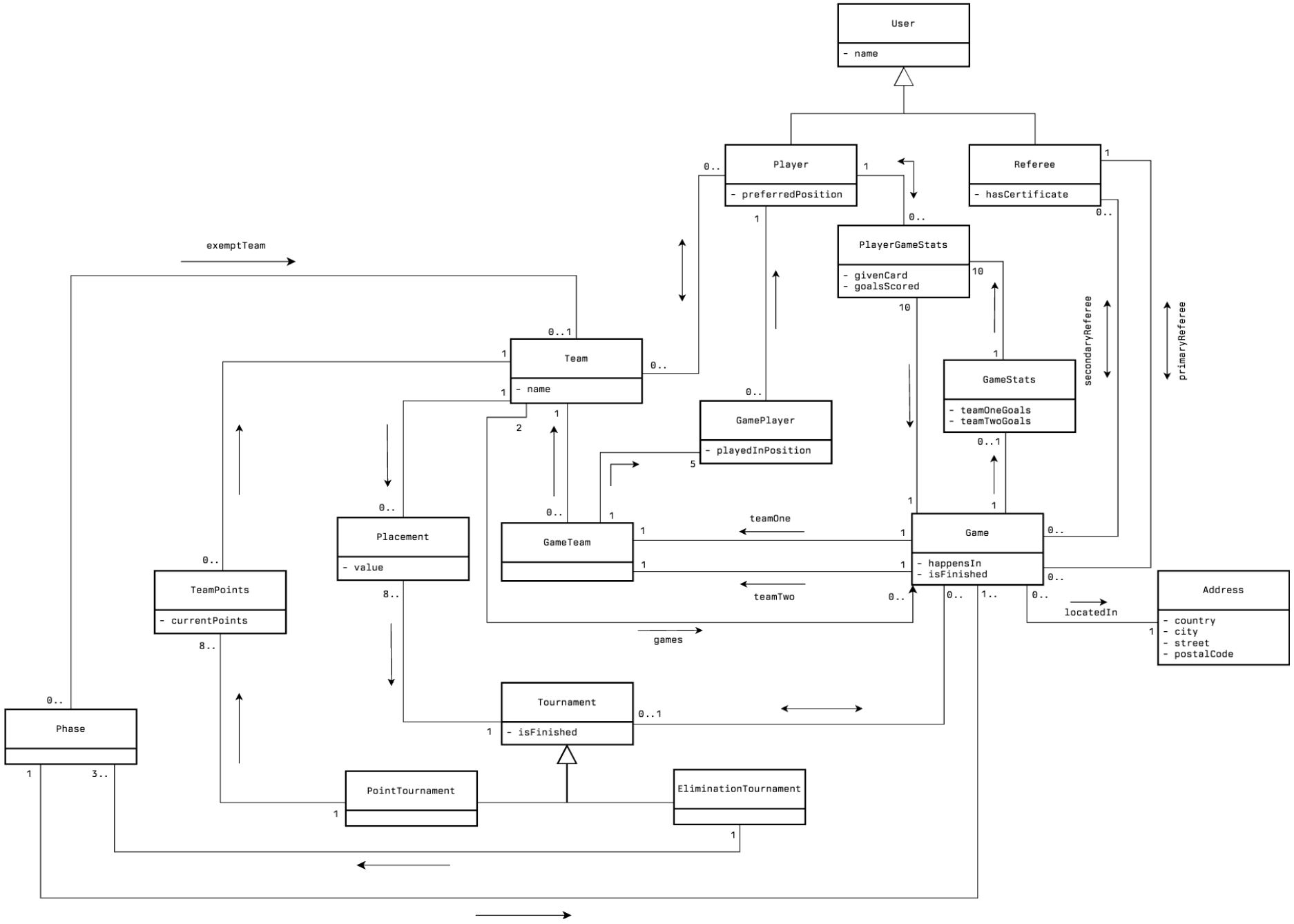
Camada de Dados

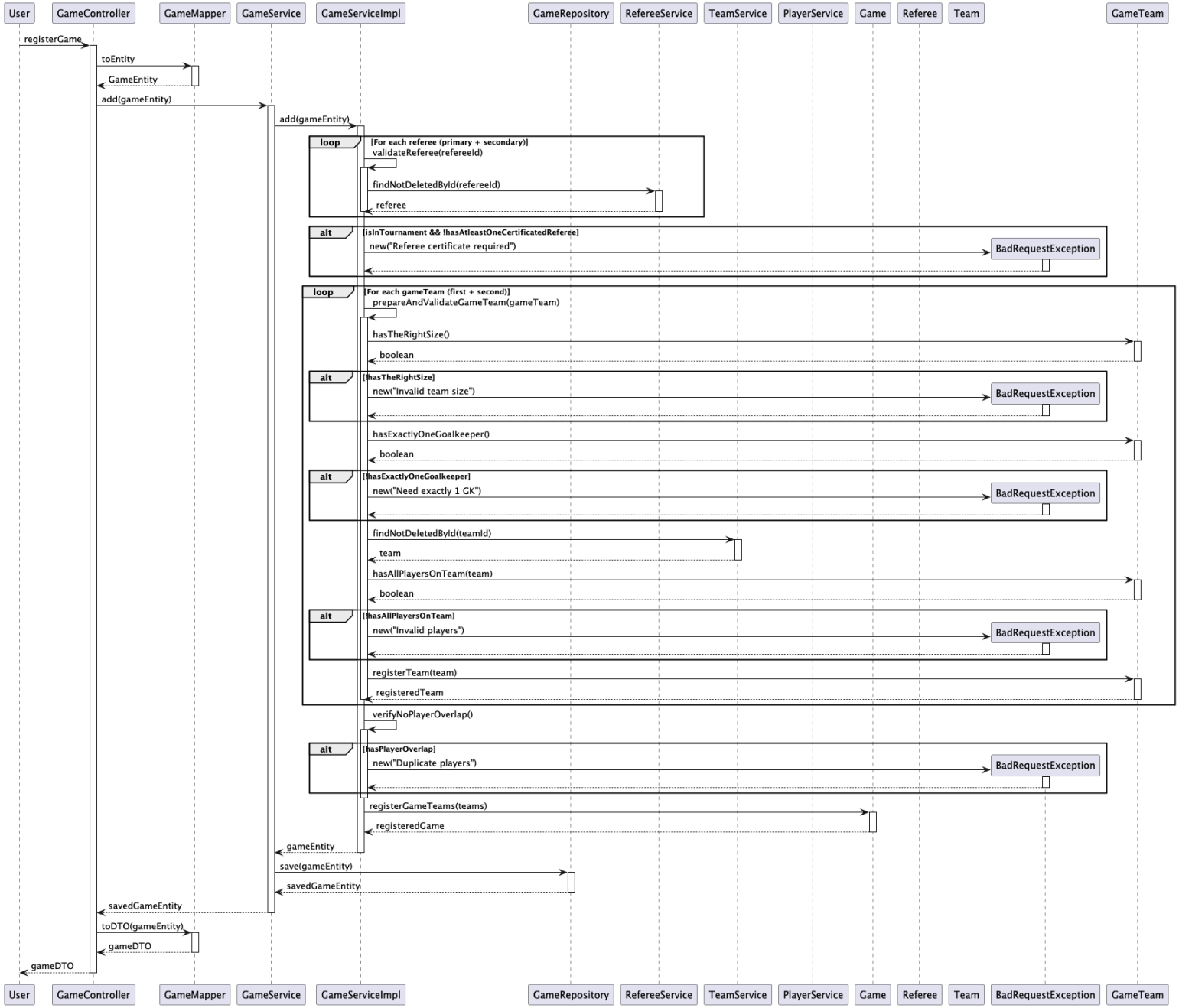
- Contém a **BD** (e todos os dados persistidos pela aplicação)

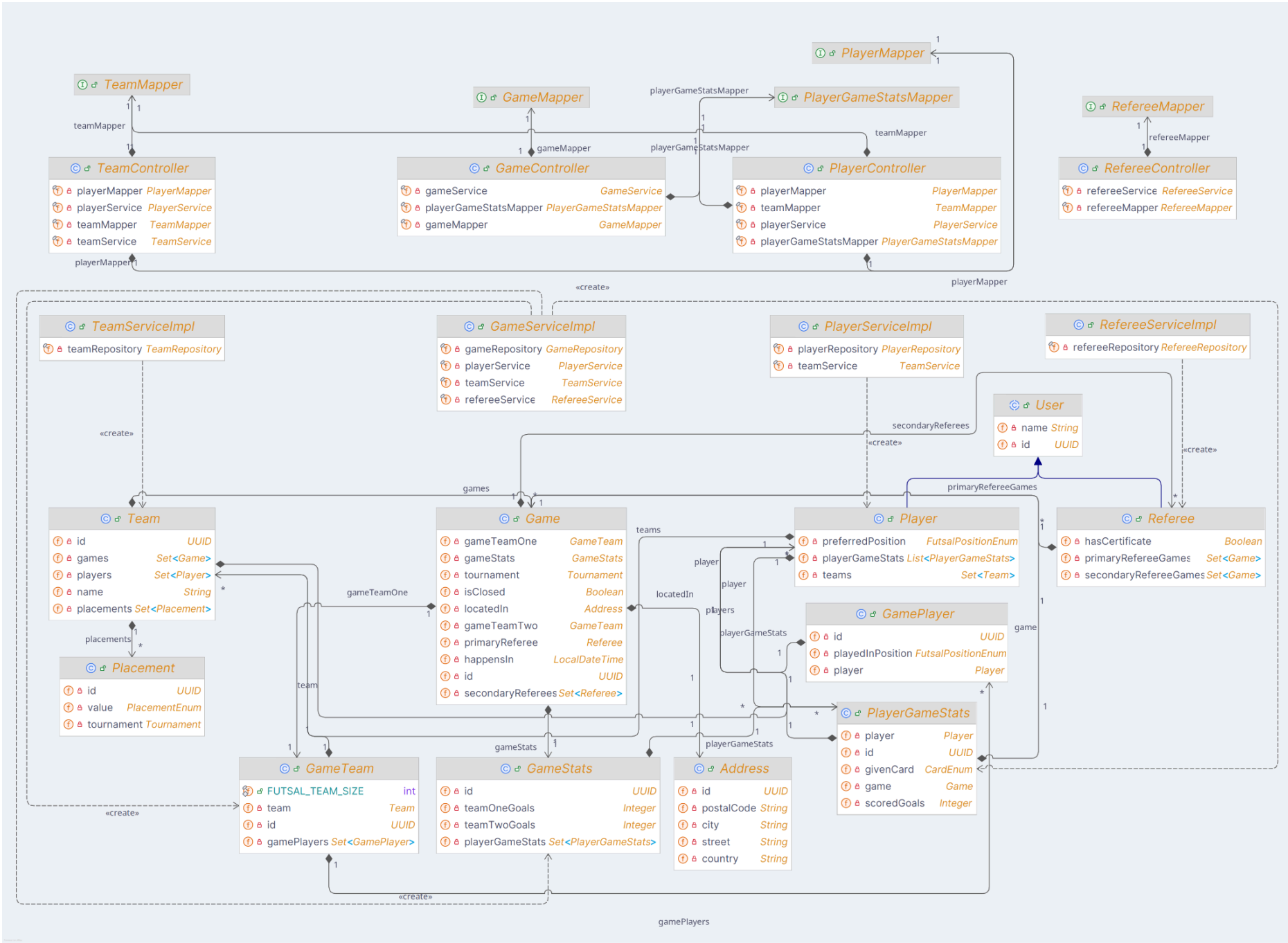
Futuramente, o projeto terá também uma Camada de Apresentação que ainda não está presente nesta fase.

Seguidamente, apresentamos:

1. **Modelo de Domínio**
2. **SSD do Caso de Uso H**
3. **Diagrama de Classes**


User
- name
Player
- preferredPosition
Referee
- hasCertificate
PlayerGameStats
- givenCard
- goalsScored
Team
- name
GamePlayer
- playedInPosition
GameStats
- teamOneGoals
- teamTwoGoals
Placement
- value
GameTeam
Game
- happensIn
- isFinished
TeamPoints
- currentPoints
Address
- country
- city
- street
- postalCode
Tournament
- isFinished
Phase
PointTournament
EliminationTournament
exemptTeam
secondaryReferee
primaryReferee
teamOne
teamTwo
games
locatedIn


User
GameController
GameMapper
GameService
GameServiceImpl
GameRepository
RefereeService
TeamService
PlayerService
Game
Referee
Team
GameTeam
BadRequestException
registerGame
toEntity
GameEntity
add(gameEntity)
add(gameEntity)
loop
[For each referee (primary + secondary)]
validateReferee(refereeId)
findNotDeletedById(refereeId)
referee
alt
[isInTournament && !hasAtleastOneCertificatedReferee]
new("Referee certificate required")
loop
[For each gameTeam (first + second)]
prepareAndValidateGameTeam(gameTeam)
hasTheRightSize()
boolean
alt
[!hasTheRightSize]
new("Invalid team size")
hasExactlyOneGoalkeeper()
boolean
alt
[!hasExactlyOneGoalkeeper]
new("Need exactly 1 GK")
findNotDeletedById(teamId)
team
hasAllPlayersOnTeam(team)
boolean
alt
[!hasAllPlayersOnTeam]
new("Invalid players")
registerTeam(team)
registeredTeam
verifyNoPlayerOverlap()
alt
[hasPlayerOverlap]
new("Duplicate players")
registerGameTeams(teams)
registeredGame
gameEntity
save(gameEntity)
savedGameEntity
savedGameEntity
toDTO(gameEntity)
gameDTO
gameDTO


PlayerMapper
TeamMapper
GameMapper
PlayerGameStatsMapper
RefereeMapper
teamMapper
gameMapper
playerGameStatsMapper
refereeMapper
TeamController
playerMapper PlayerMapper
playerService PlayerService
teamMapper TeamMapper
teamService TeamService
GameController
gameService GameService
playerGameStatsMapper PlayerGameStatsMapper
gameMapper GameMapper
PlayerController
playerMapper PlayerMapper
teamMapper TeamMapper
playerService PlayerService
playerGameStatsMapper PlayerGameStatsMapper
RefereeController
refereeService RefereeService
refereeMapper RefereeMapper
«create»
TeamServiceImpl
teamRepository TeamRepository
GameServiceImpl
gameRepository GameRepository
playerService PlayerService
teamService TeamService
refereeService RefereeService
PlayerServiceImpl
playerRepository PlayerRepository
teamService TeamService
RefereeServiceImpl
refereeRepository RefereeRepository
User
name String
id UUID
secondaryReferees
games
primaryRefereeGames
Team
id UUID
games Set<Game>
players Set<Player>
name String
placements Set<Placement>
Game
gameTeamOne GameTeam
gameStats GameStats
tournament Tournament
isClosed Boolean
locatedIn Address
gameTeamTwo GameTeam
primaryReferee Referee
happensIn LocalDateTime
id UUID
secondaryReferees Set<Referee>
Player
preferredPosition FutsalPositionEnum
playerGameStats List<PlayerGameStats>
teams Set<Team>
Referee
hasCertificate Boolean
primaryRefereeGames Set<Game>
secondaryRefereeGames Set<Game>
Placement
id UUID
value PlacementEnum
tournament Tournament
placements
gameTeamOne
teams
player
players
locatedIn
playerGameStats
game
team
GamePlayer
id UUID
playedInPosition FutsalPositionEnum
player Player
PlayerGameStats
player Player
id UUID
givenCard CardEnum
game Game
scoredGoals Integer
GameTeam
FUTSAL_TEAM_SIZE int
team Team
id UUID
gamePlayers Set<GamePlayer>
GameStats
id UUID
teamOneGoals Integer
teamTwoGoals Integer
playerGameStats Set<PlayerGameStats>
Address
id UUID
postalCode String
city String
street String
country String
gameStats
playerGameStats
gamePlayers

Mapeamento JPA:

Usamos o ***@Entity*** para definir todos os objetos que precisavam de ser persistidos na BD.

Para a herança, neste momento utilizada para os Players e Referees que herdam de User, usamos o ***@Inheritance*****(strategy = InheritanceType.*TABLE_PER_CLASS*)** uma vez que não existem muitas subclasses da Entidade User e existem alguns parâmetros que diferem entre ambos (por exemplo, um Player ter acesso às suas equipas e não diretamente aos jogos, ao contrário dos Referees).

Nas as ligações do Modelo de Domínio em que temos **1..N,** usamos a anotação ***@OneToMany***.

Para **M..N** usamos a anotação ***@ManyToMany*** com ***@JoinTable***.

Para **N..1** usamos a anotação ***@ManyToOne***.

Para **1..1** usamos a anotação ***@OneToOne***.

Para a maioria dos casos, usamos o FetchType default do JPA (***LAZY*** para ***@OneToMany*** e ***@ManyToMany*** e ***EAGER*** para ***@ManyToOne*** e ***@OneToOne***).

Usamos também o padrão **Identity Field** para facilitar o mapeamento entre o Modelo Relacional e OO, usando sempre um field extra anotado com ***@ID*** do tipo UUID, gerado automaticamente pela BD com ***@GeneratedValue*****(strategy = GenerationType.*UUID*).**

Para as entidades principais (User,Team,Game,Tournament) usamos também herança com uma classe extra, **SoftDeleteEntity**, com o ***@MappedSuperclass*** de modo a não criar uma tabela para a mesma, mas todos os seus atributos serem persistidos nas subclasses de modo a encapsular tanto a lógica de "dar softdelete" (com um atributo extra para a hora/dia em que se "dá delete" numa entidade) como a lógica da gestão de concorrência otimista, com o **@Version.**

Nas as entidades fracas (entidades que não têm um repositório específico e que dependem de outras entidades) usamos o **cascade = CascadeType.ALL,** de modo a assegurar que ao fazer alterações à entidade forte, essas mesmas são persistidas para as suas entidades fracas.

Para os enumerados usamos a anotação ***@Enumerated*(EnumType.STRING),** usamos o **EnumType.String** ao invés do **EnumType.Ordinal**, uma vez que neste caso o espaço não é um problema e preferimos ter uma representação mais expressiva do enumerado na BD para facilitar a fase de desenvolvimento.

Para assegurar que as entidades não são persistidas na BD com valores errados, colocámos também anotações como ***@Size***, ***@NotNull***,etc.

Garantias que o sistema oferece relativamente à Lógica de Negócio:

O sistema oferece garantias de concorrência, coerência e persistência de dados.

- **Concorrência:** Assumimos um modelo de gestão de concorrência otimista (pouca probabilidade de ocorrerem problemas de concorrência), colocando a tag *@Version* em todas as entidades principais. Para funcionar corretamente, colocámos um atributo extra que representa a versão de uma entidade num determinado momento, que é utilizado para verificar se essa versão é de facto a mais atualizada e aceitar ou rejeitar as alterações consoante isso.

- **Coerência de dados:** Todos os métodos críticos estão anotados com a tag *@Transactional* de modo a que executem de forma atómica.

- **Persistência de dados:** Foi utilizada a JPA como framework de mapeamento objeto-relacional. As operações são realizadas através de repositórios Spring Data JPA.

DTOs:

Nesta fase, desenvolvemos um DTO para cada entidade forte (Game, Player, Referee e Team) e para PlayerGameStats. Para evitar o problema de relações cíclicas entre DTOs, criamos (para cada classe de DTO principal) DTOs internos especializados que representam as entidades com as quais se relaciona, de modo a fornecer apenas a informação necessária.

Dessa forma, evitamos a ocorrência de ciclos e entregamos dados que podem ser já utilizados na camada de apresentação (a ser desenvolvida na próxima fase), sem a necessidade de fazer re-fetch (no caso de termos apenas IDs para cada entidade que aqui é representada por um DTO especializado).