Code of Duty

Football Tournament Management Website
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
09/Dec/23	<1.0>	Part 1, 2, 3 & 4.1, 4.12, 4.13	Nguyễn Tuấn Đạt
		Part 4.2, 4.5	Bùi Đình Bảo
		Part 4.3, 4.4 & package diagrams	Nguyễn Đình Ánh
		Part 4.6, 4.7, 4.10	Lê Minh Huy
		Part 4.9, 4.11	Triệu Hoàng Thiên Ân

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

- This Software Architecture document provides readers with an overview of the overall system architecture. It presents various models of the system architecture from different perspectives.
- The document is intended to summarize and communicate critical decisions made when selecting the software architecture for the system.

1.2 Scope

- This Software Architecture document is used during the development of the Football Tournament Management Website.

1.3 Overview

- This Software Architecture document includes:
    - Architectural Goals and Constraints: The goals and constraints of the software architecture.
    - Use-case Model: Contains the use-case model.
    - Logical View: Provides a list of components in the system and their relationships, including package diagrams and class diagrams.
    - Deployment: Describes the method of deploying the components of the application.
    - Implementation View: Outlines the structure of directories containing code and how components are implemented.

2. Architectural Goals and Constraints

2.1 Applicable Standards

The website must adhere to industry-standard web development best practices, recommendations published by the World Wide Web Consortium (W3C), including HTML5 and CSS3 standards.

2.2 System Requirements

The server component of the website shall operate under the Linux operating system.

The website shall operate on any personal computer with at least 1.5 GHz CPU or better.

The website shall not require more than 100 MB RAM and 20 MB Disk Space.

The website should be compatible with major web browsers, including Chrome, Firefox, Safari, and Edge.

The website must be accessible on various devices, including desktop computers, tablets, and mobile devices.

2.3 Performance Requirements

The website should provide fast response times to ensure a smooth user experience. Pages must load within 2 seconds on average.

The website shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.

The website should have an uptime of at least 99.9%.

Data storage and retrieval operations should take no more than 200 milliseconds on average.

The website shall complete 80% of all purchases within 2 minutes.

Automated notifications must be sent within 30 seconds of the relevant event.

For input fields: maximum 30 data fields, no complex data calculations, no interaction with external systems, can store data directly to the DB, and no storage of large content files such as: images, videos, files exceeding 3MB.

2.4 Environmental Requirements

None.

2.5 Quality Ranges

Availability: The website shall be available 24 hours a day, 7 days a week, upgrade a maximum of once within 3 months, and downtime must not exceed 1 hour per year.
Usability: The website shall be easy-to-use and shall be appropriate for the target market of small to middle football tournament organizers. Moreover, the website should adapt to different screen sizes and resolutions for a consistent user experience on devices with screen sizes from 320x480 pixels to 1920x1080 pixels.
Usability: The System shall include online help for the user, with an average time to find information not exceeding 2 minutes. The time for training tournament organizers' IT admin to use the website should be less than 2 hours.
Maintainability: The website shall be designed for ease of maintenance by team Tournament data shall be modifiable without restarting the web server.

2.6 Constraints

The system shall not require any hardware development or procurement.
The number of available tournaments is limited to those organized by the given organizer.
The website relies on a third-party payment gateway for financial transactions. The choice of payment gateway is constrained by the availability and compatibility of external payment processing services.

2.7 Documentation Requirements

This section describes the documentation requirements of the Football Tournament Management Website.

2.7.1 User manual

The User Manual provides comprehensive guidance for users, including organizers, referees, teams, and spectators. The User Manual shall include:

- System Usage
- Minimum System Requirements
- Installation
- Logging In and Out
- Feature Descriptions
- Customer Support

The User Manual should consist of 50 to 100 pages and follow a standardized format from TA. It will be available in both hardcopy and digital formats through online help.

2.7.2 Online help

Online Help offers immediate assistance to users for each website function. Each topic covered in the User Manual will also be accessible through the online help system.

2.7.3 Installation Guides, Configuration, and Read Me File

The Installation Guide for the server component shall include:

- Minimum System Requirements
- Installation Instructions
- Configuring Organization-Specific Parameters
- How to Initialize the Database
- Customer Support Information
- How to Order Upgrades
- The ReadMe File shall be available for display following installation. The ReadMe File will also reside on disk and be available for viewing at any time by the user. The ReadMe File shall include:

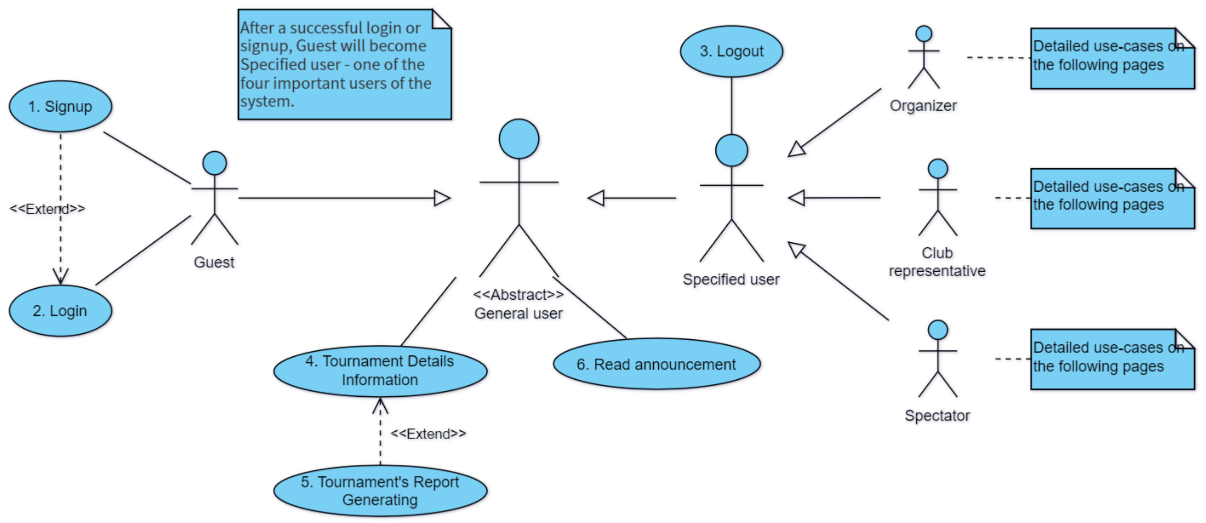
The ReadMe File shall be available for display following installation. The ReadMe File will also reside on disk and be available for viewing at any time by the user. The ReadMe File shall include:

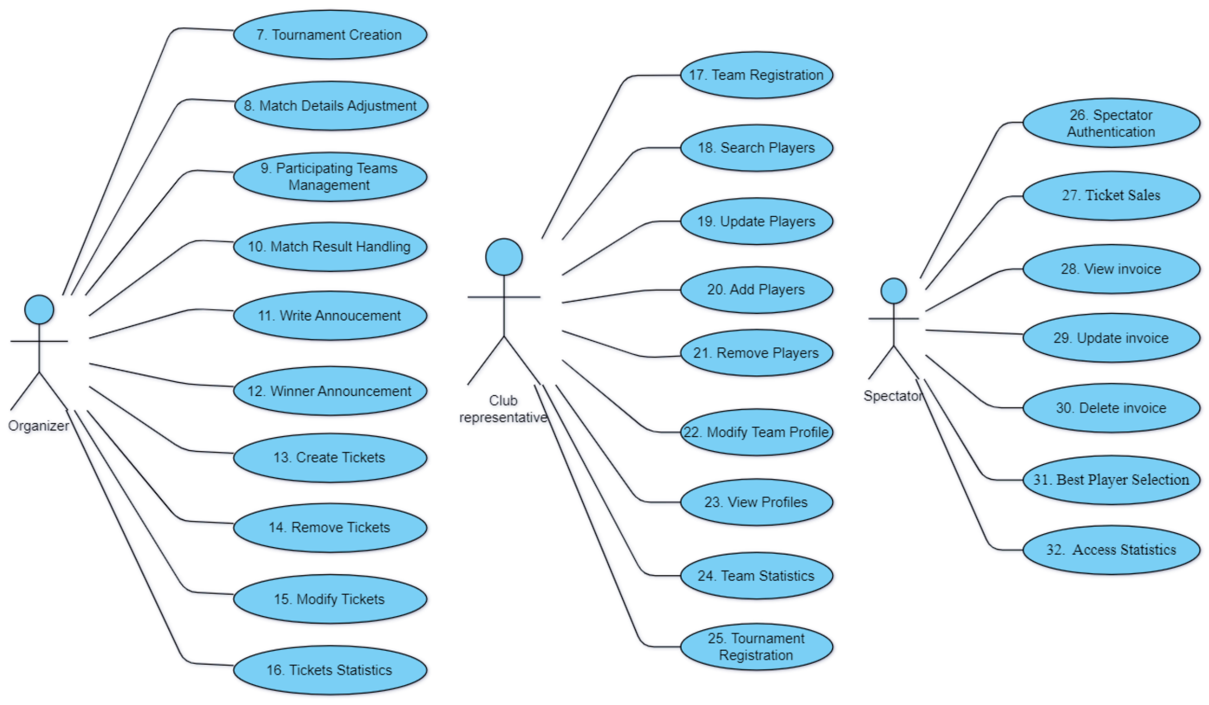
- New release features
- Known bugs and workarounds.

2.7.4 Labeling and Packaging

The Football Tournament Management website's branding, including the official logo, shall be prominently displayed on user documentation and splash screens.
As the initial releases are primarily for specific organizations or institutions and not the general market, there are no plans for extensive product marketing literature, product packaging, or promotional materials at this stage.

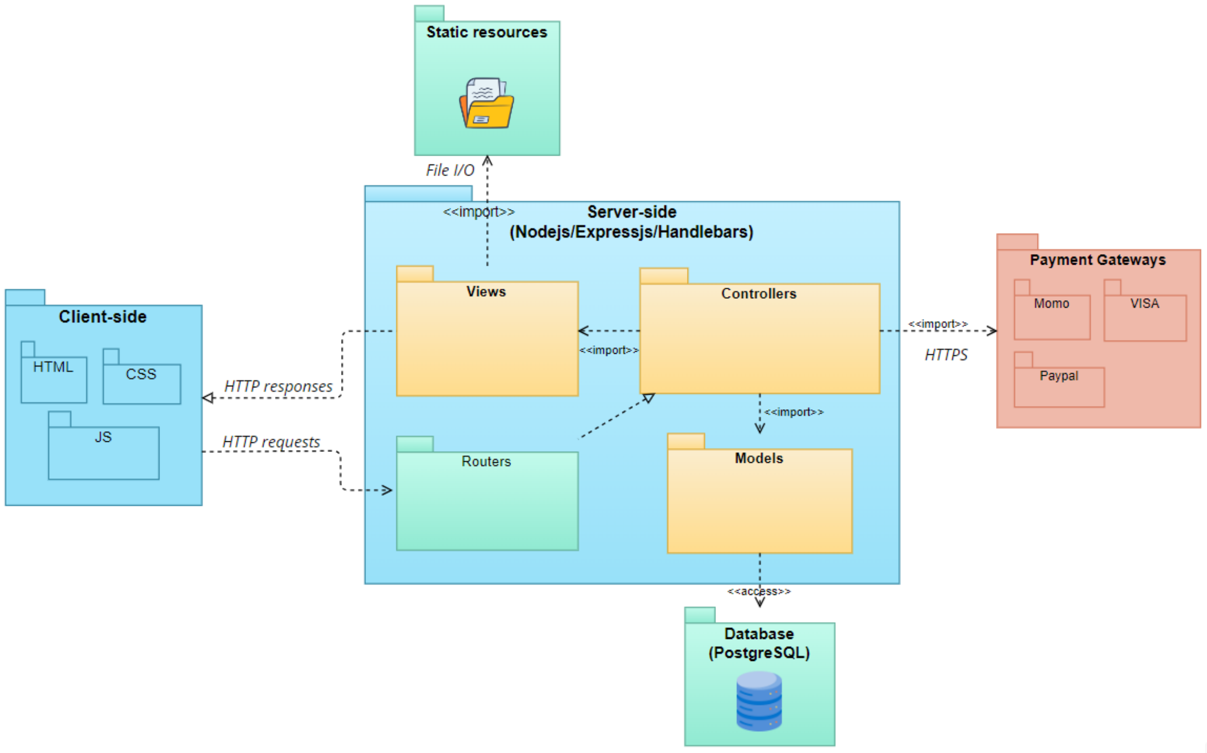
3. Use-Case Model

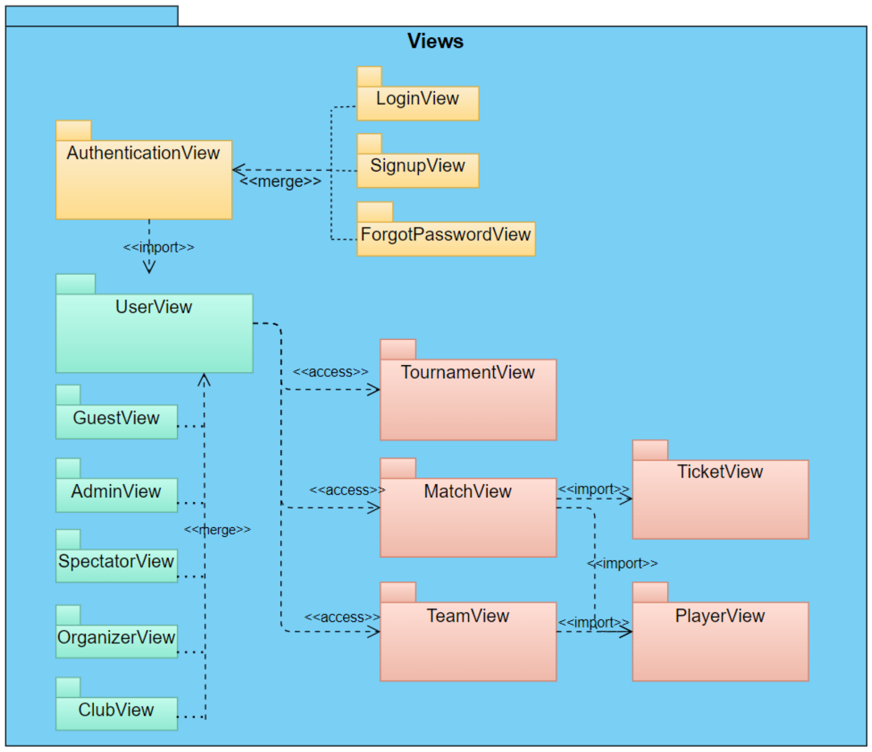

After a successful login or signup, Guest will become Specified user - one of the four important users of the system.
1. Signup
<<Extend>>
2. Login
Guest
<<Abstract>> General user
4. Tournament Details Information
<<Extend>>
5. Tournament's Report Generating
6. Read announcement
3. Logout
Specified user
Organizer
Club representative
Spectator
Detailed use-cases on the following pages
Detailed use-cases on the following pages
Detailed use-cases on the following pages

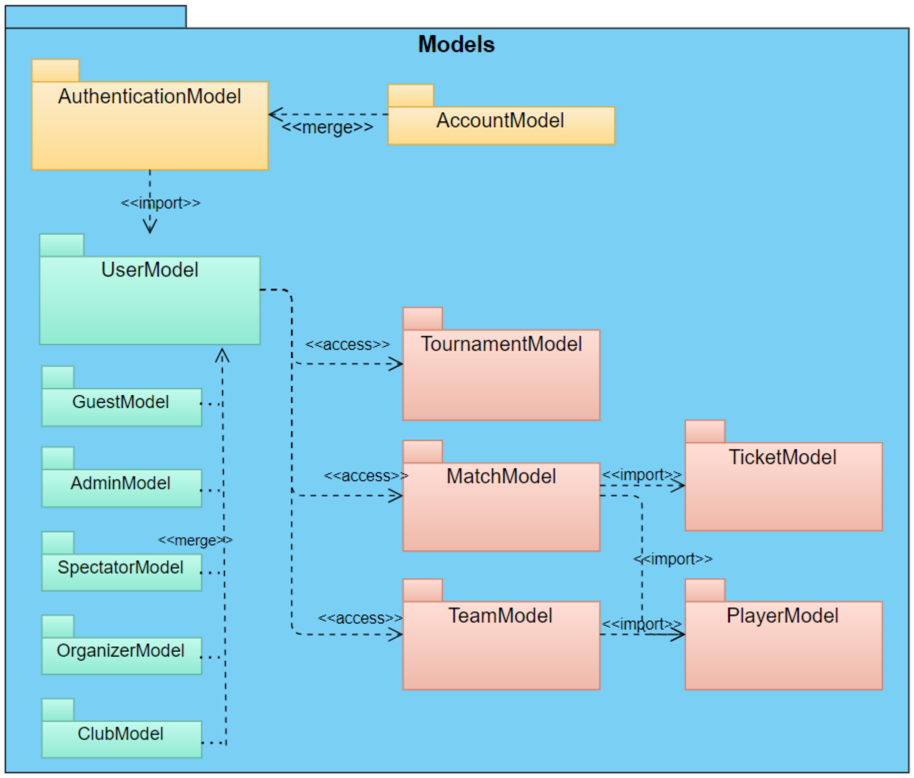


4. Logical View

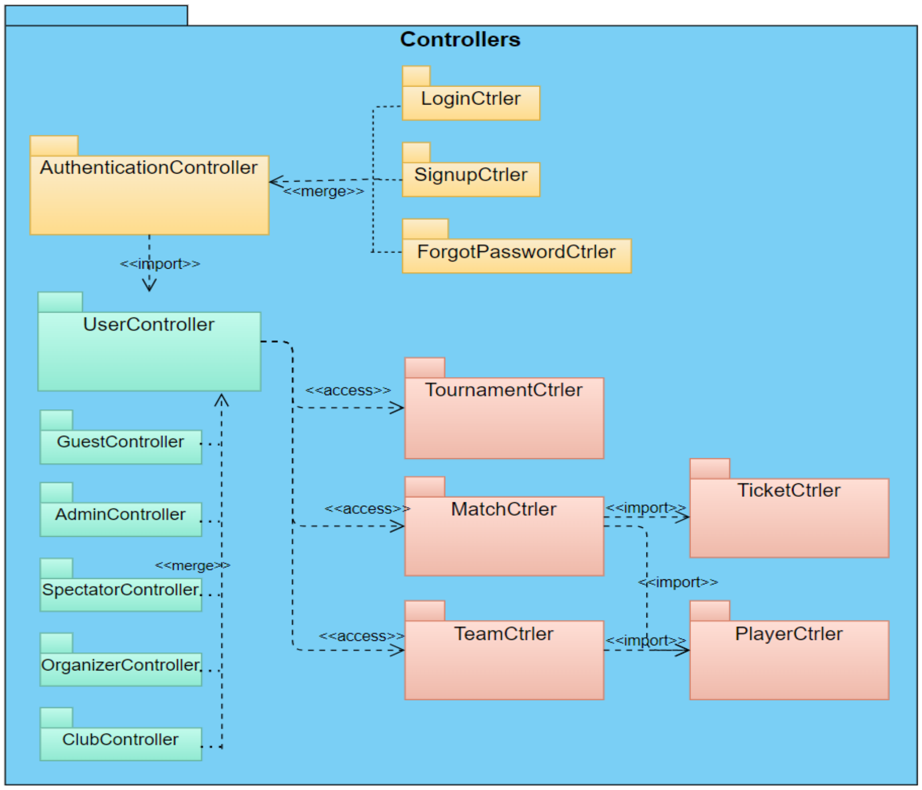
The logic view package diagram is illustrated as below:







Models
AuthenticationModel
AccountModel
<<merge>>
<<import>>
UserModel
GuestModel
AdminModel
<<merge>>
SpectatorModel
OrganizerModel
ClubModel
<<access>>
TournamentModel
<<access>>
MatchModel
<<import>>
TicketModel
<<import>>
<<access>>
TeamModel
<<import>>
PlayerModel

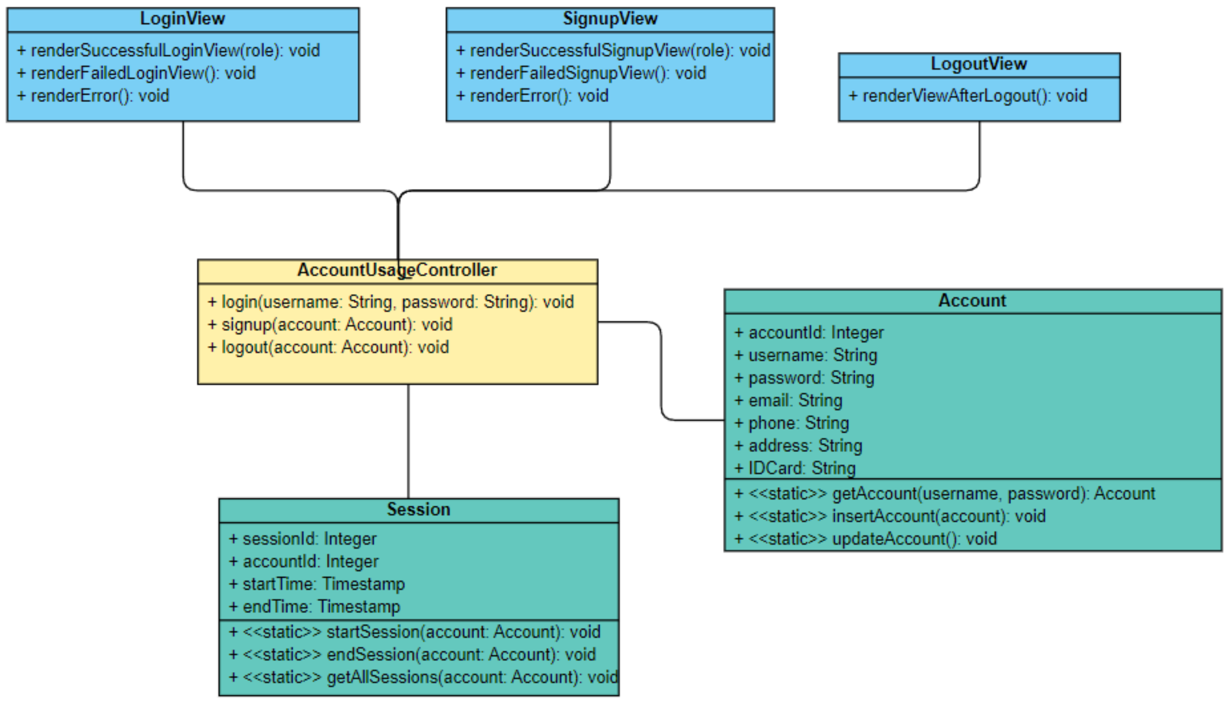


Controllers
LoginCtrler
SignupCtrler
ForgotPasswordCtrler
AuthenticationController
<<merge>>
<<import>>
UserController
GuestController
AdminController
<<merge>>
SpectatorController
OrganizerController
ClubController
<<access>>
TournamentCtrler
<<access>>
MatchCtrler
<<import>>
TicketCtrler
<<import>>
<<access>>
TeamCtrler
<<import>>
PlayerCtrler

4.1 Component: Registration and operation of personal accounts

4.1.1 Class diagram

\


LoginView
+ renderSuccessfulLoginView(role): void
+ renderFailedLoginView(): void
+ renderError(): void
SignupView
+ renderSuccessfulSignupView(role): void
+ renderFailedSignupView(): void
+ renderError(): void
LogoutView
+ renderViewAfterLogout(): void
AccountUsageController
+ login(username: String, password: String): void
+ signup(account: Account): void
+ logout(account: Account): void
Account
+ accountId: Integer
+ username: String
+ password: String
+ email: String
+ phone: String
+ address: String
+ IDCard: String
+ <<static>> getAccount(username, password): Account
+ <<static>> insertAccount(account): void
+ <<static>> updateAccount(): void
Session
+ sessionId: Integer
+ accountId: Integer
+ startTime: Timestamp
+ endTime: Timestamp
+ <<static>> startSession(account: Account): void
+ <<static>> endSession(account: Account): void
+ <<static>> getAllSessions(account: Account): void

4.1.2 LoginView
- renderSuccessfulLoginView(): render view which will be displayed in the case of guest logins successfully.
- renderFailedLoginView(): render view which will be displayed in the case guest logins failed.
- renderErrorView(): render view which will be displayed in an exception situation.

4.1.3 SignupView
- renderSuccessfulLoginView(): render view which will be displayed in the case of guest signup successfully.
- renderFailedLoginView(): render view which will be displayed in the case guest signup failed.
- renderErrorView(): render view which will be displayed in an exception situation.

4.1.4 SignupView
- renderViewAfterLogout(): render guest home view after user click logout.

4.1.5 AccountUsageController
- login(): Check whether username and password is valid; if it is, call the renderSuccessfulLoginView function; otherwise, call the renderFailedView function.
- signup(): Check whether the account is valid; if it is, call the renderSuccessfulSignupView() function; otherwise, call the renderFailedView() function.
- logout(): Log out of the account and call the renderViewAfterLogout function.

4.1.6 Account
- accountId: Integer variable storing id of this account.
- username: String variable storing username of this account.
- password: String variable storing password of this account.
- email: String variable storing user's email.
- phone: String variable storing user's phone.
- address: String variable storing user's address.

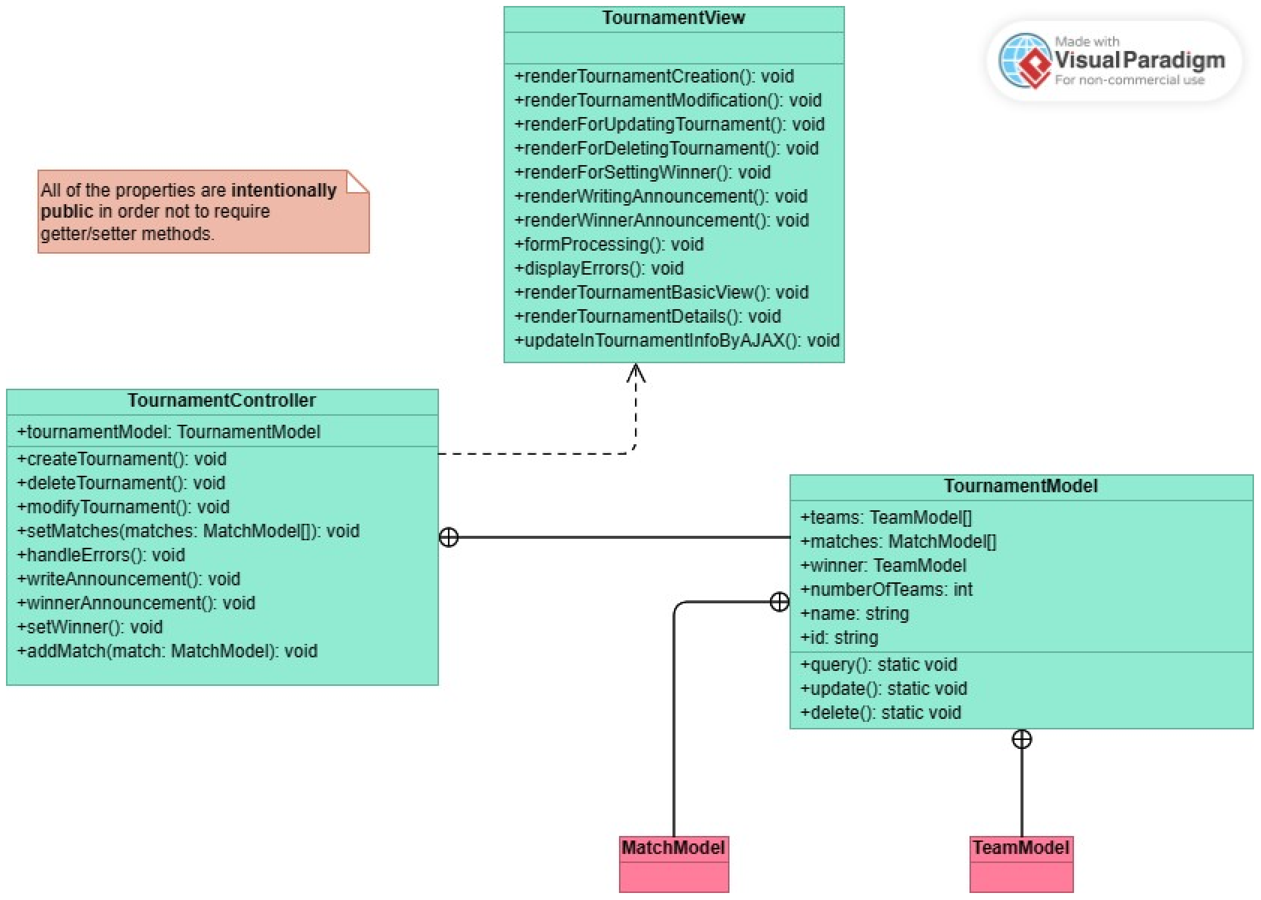
- IDCard: String variable storing user's Identity card number.

4.1.7 Session

- sessionId: Integer variable storing id of this session.
- accountId: Integer variable storing id of this account.
- startTime: Timestamp variable containing the beginning time of the work session.
- endTime: Timestamp variable containing the ending time of the work session.
- startSession(): Create a new session.
- endSession(): End current session.
- getAllSessions(): Get all sessions of a given account.

4.2 Component: Tournament management (Organizer)

4.2.1 Class Diagram


TournamentView
+renderTournamentCreation(): void
+renderTournamentModification(): void
+renderForUpdatingTournament(): void
+renderForDeletingTournament(): void
+renderForSettingWinner(): void
+renderWritingAnnouncement(): void
+renderWinnerAnnouncement(): void
+formProcessing(): void
+displayErrors(): void
+renderTournamentBasicView(): void
+renderTournamentDetails(): void
+updateInTournamentInfoByAJAX(): void
Made with Visual Paradigm For non-commercial use
All of the properties are intentionally public in order not to require getter/setter methods.
TournamentController
+tournamentModel: TournamentModel
+createTournament(): void
+deleteTournament(): void
+modifyTournament(): void
+setMatches(matches: MatchModel[]): void
+handleErrors(): void
+writeAnnouncement(): void
+winnerAnnouncement(): void
+setWinner(): void
+addMatch(match: MatchModel): void
TournamentModel
+teams: TeamModel[]
+matches: MatchModel[]
+winner: TeamModel
+numberOfTeams: int
+name: string
+id: string
+query(): static void
+update(): static void
+delete(): static void
MatchModel
TeamModel

4.2.2 TournamentView - graphical user interface

- Method renderTournamentCreation(): return void type, display the GUI for creating a new tournament, include in some information like: a set of teams in the tournament, a set of matches happening in the tournament, the number of teams in the tournament, the specific id and the name of the tournament.
- Method renderTournamentModification(): return void type, redisplay the interface for tournament modification, as the rendering for tournament creation immediately after initializing the tournament.
- Method renderForUpdatingTournament(): return void type, redisplay the interface for tournament modification, as the rendering for tournament creation but it is more flexible than modifying, allowing users to modify even if the status of the tournament is ended up.
- Method renderForDeletingTournament(): return void type, display a popup window for users to make sure

that the tournament (and all of the data related to) will be removed from the database, and expect to select a specific tournament before this function.
- Method renderForSettingWinner(): return void type, show the webpage for users to set the final team winner of the tournament and the tournament status will be ended up as well.
- Method renderWritingAnnouncement(): return void type, the interface allows users to write a new announcement about the tournament to update the information for the spectators.
- Method renderWinnerAnnouncement(): return void type, the interface allows users to write a final announcement to let the spectator know about the team winner, and the tournament will be ended as well.
- Method formProcessing(): return void type, process the form inputs.
- Method displayErrors(): return void type, process to alert notification about errors.
- Method renderTournamentBasicView(): return void type, display the item related to a tournament in a list of tournaments (because the UI will be designed by component-based language: handlebars).
- Method renderTournamentDetails(): return void type, show the details of the tournament corresponding to.
- Method updateInTournamentInfoByAJAX(): return void type, automatically update the live-information of a tournament without refreshing the webpage.

Expected to be written with Express-Handlebars, HTML, CSS, JS.

4.2.3 TournamentController - the logic processing part
- Attribute tournamentModel: type TournamentModel, the model of the currently indicated tournament.
- Method createTournament(): return void type, help to initialize a new tournament in the database (include all the football teams joining in the tournament, reset the number of teams attribute as well).
- Method deleteTournament(): return void type, help to remove a specific tournament from the database.
- Method modifyTournament(): return void type, support modifying the information of the tournament based on the users inputs.
- Method setMatches(match: MatchModel): return void type with parameter matches (an array of MatchModel type), used to set the match for the indicated ticket.
- Method handleErrors(): return void type, handle the errors in the business flow.
- Method writeAnnouncement(): return void type, support to write an announcement for the tournament.
- Method winnerAnnouncement(): return void type, support to write an announcement for the tournament.
- Method setWinner(): return void type, allow the users to set the winner of the tournament, which will be ended immediately after that as well.
- Method addMatch(): return void type with parameter match of MatchModel type, this function helps the users to add a new match to the current tournament if necessary.

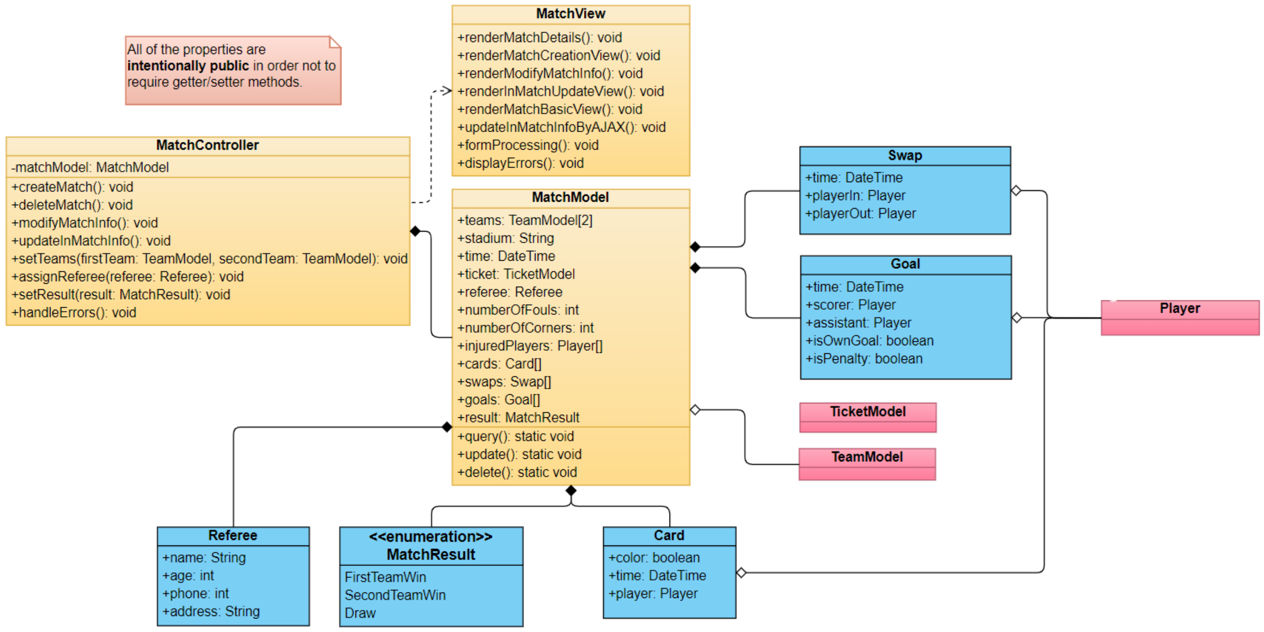
Expected to be written with Node.js, Express.js.

4.2.4 TournamentModel - the data part
- Attribute teams: type array of TeamModel, contain a set of all the teams joining in the tournament.
- Attribute matches: type array of MatchModel, contain a set of all the matches happening in the tournament.
- Attribute winner: type TeamModel, containing the information of the team winner of the tournament, could be null if the tournament still happens.
- Attribute numberOfTeams: type int, contain the number of football teams in the tournament.
- Attribute name: type string, store the name of the tournament.
- Attribute id: type string, store the id of the tournament, in order to distinguish it from other tournaments.
- Method static query(): supports connecting to the database and getting the necessary data.
- Method static update(): supports connecting to the database and updating the necessary records.
- Method static delete(): supports connecting to the database and deleting the necessary records.

4.2.5 Other related classes and note
- MatchModel: have a relationship with TournamentModel due to the attribute matches.
- TeamModel: have a relationship with TournamentModel due to the attribute teams.
- Note: All of the properties are intentionally public in order not to require getter/setter methods.

4.3 Component: Match management (Organizer)

4.3.1 Class diagram



4.3.2 MatchView - graphical user interface

- renderMatchDetails(): display the details of a match.
- renderMatchCreationView(): display the page to create a match.
- renderModifyMatchInfo(): display the page to modify basic information of a match.
- renderMatchUpdateView(): display the page to update the in-match (live) information of a match.
- renderMatchBasicView(): display the item related to a match in a list of matches (because the UI will be designed by component-based language: handlebars).
- updateInMatchInfoByAJAX(): automatically update the live-information of a match without refreshing the webpage.
- formProcessing(): process the form inputs.
- displayErrors(): process to alert notification about errors.
- Expected to be written with Express-Handlebars, HTML, CSS, JS.

4.3.3 MatchController - the logic processing part

- matchModel: the model of the currently indicated match.
- createMatch(): create a new match in a tournament.
- deleteMatch(): delete the indicated match for recreating (as longer as the match hasn't taken place).
- modifyMatchInfo(): modify the basic information of a match.
- updateInMatchInfo(): update the live information of a match.
- setTeams(firstTeam: TeamModel, secondTeam: TeamModel): organize 2 teams in a match.
- assignReferee(referee: Referee): assign referee for that match.
- setResult(result: MatchResult): set the match result (which team wins, or draw), then update the scores and statistics in the tournament.
- handleErrors(): handle the errors in the business flow.
- Expected to be written with Node.js, Express.js.

4.3.4 MatchModel - the data part

- teams: 2 teams have registered in the tournament, and have full conditions to meet each other at that round.
- stadium, time: place and time of the match.
- ticket: if the match doesn’t require a ticket, this field is nullable.
- referee: information about the main referee.
- numberOfFouls, numberOfCorners: number of fouls and number of corner kicks in the match.
- injuredPlayers: an array of players who are swapped out because of being injured.

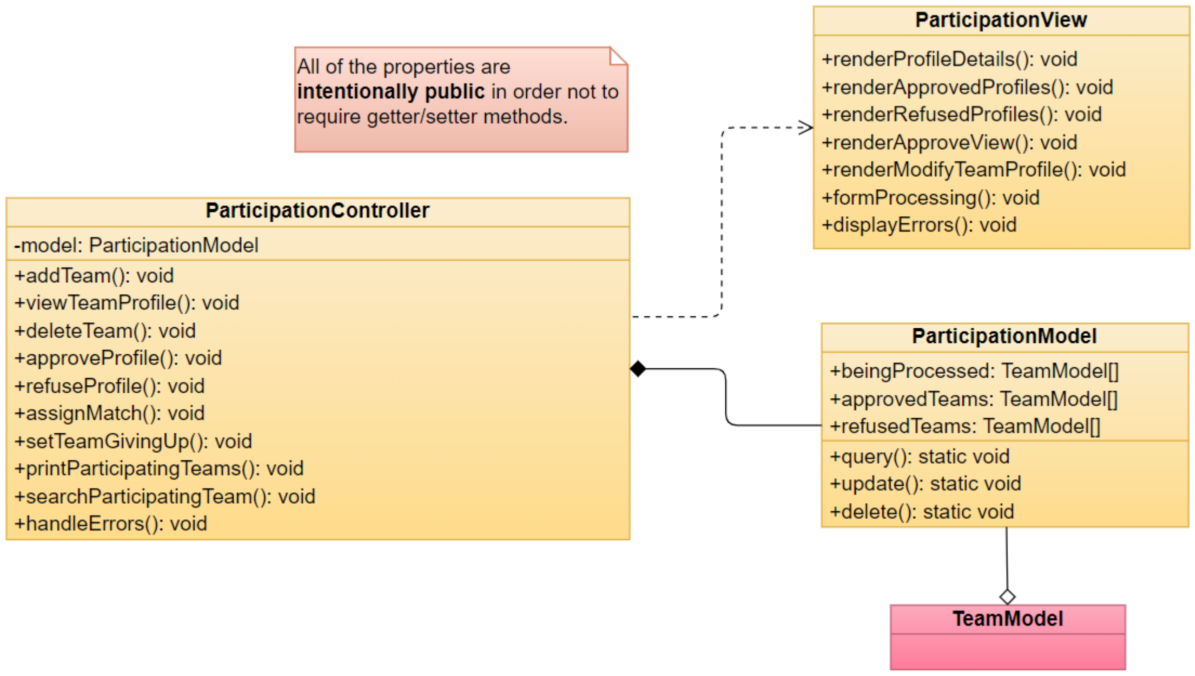
- cards: an array of cards (yellow and red) that are given in the match and information about them.
- swaps: a record about who is swapped in/out and when.
- goals: all of the goals to be made in the match, including penalties.
- result: after the match is over, the result will be updated to know which team wins, loses or a draw match.
- query(): this method supports connecting to the database and getting the necessary data.
- update(): this method supports connecting to the database and updating the necessary records.
- delete(): this method supports connecting to the database and deleting the necessary records.

4.3.5 Others

- The other classes take the supporting roles for the key classes (Model, View, Controller) in the current software component.
- About TicketModel, Player/Footballer and TeamModel, these classes have been/will be defined in the previous/next component sections.

4.4 Component: Management of participating football teams (Organizer)

4.4.1 Class diagram


All of the properties are **intentionally public** in order not to require getter/setter methods.
ParticipationView
+renderProfileDetails(): void
+renderApprovedProfiles(): void
+renderRefusedProfiles(): void
+renderApproveView(): void
+renderModifyTeamProfile(): void
+formProcessing(): void
+displayErrors(): void
ParticipationController
-model: ParticipationModel
+addTeam(): void
+viewTeamProfile(): void
+deleteTeam(): void
+approveProfile(): void
+refuseProfile(): void
+assignMatch(): void
+setTeamGivingUp(): void
+printParticipatingTeams(): void
+searchParticipatingTeam(): void
+handleErrors(): void
ParticipationModel
+beingProcessed: TeamModel[]
+approvedTeams: TeamModel[]
+refusedTeams: TeamModel[]
+query(): static void
+update(): static void
+delete(): static void
TeamModel

4.4.2 ParticipationView - graphical user interface

- renderProfileDetails(): display the page to show the profile of a team sending registration to the tournament.
- renderApprovedProfiles(): display the teams who have been accepted to participate in the tournament.
- renderRefusedProfiles(): display the teams who have been rejected to participate in the tournament.
- renderApproveView(): display the page to validate the profile of the registering teams.
- renderModifyTeamProfile(): display the page to modify some basic information about teams.
- formProcessing(): handle the form inputs and validations.
- displayErrors(): display the errors to the users.

4.4.3 ParticipationController - the logic part

- model: the ParticipationModel that contains the data and helps to retrieve data from DB.
- addTeam(): add a team to the tournament without requiring registering.
- deleteTeam(): delete a team from the tournament.
- approveProfile(): approve the profile of a registering team.

- refuseProfile(): reject the profile of a registering team.
- assignMatch(): if the matches are not automatically organized, the work can be done manually.
- setTeamGivingUp(): if there's any teams quitting the tournament, set their states: Give up.
- printParticipatingTeams(): print all the teams that meet all conditions to participate in the tournament.
- searchParticipatingTeams(): search the teams participated by name.
- handleErrors(): handle any errors occurred.

4.4.4 ParticipationModel - the data part

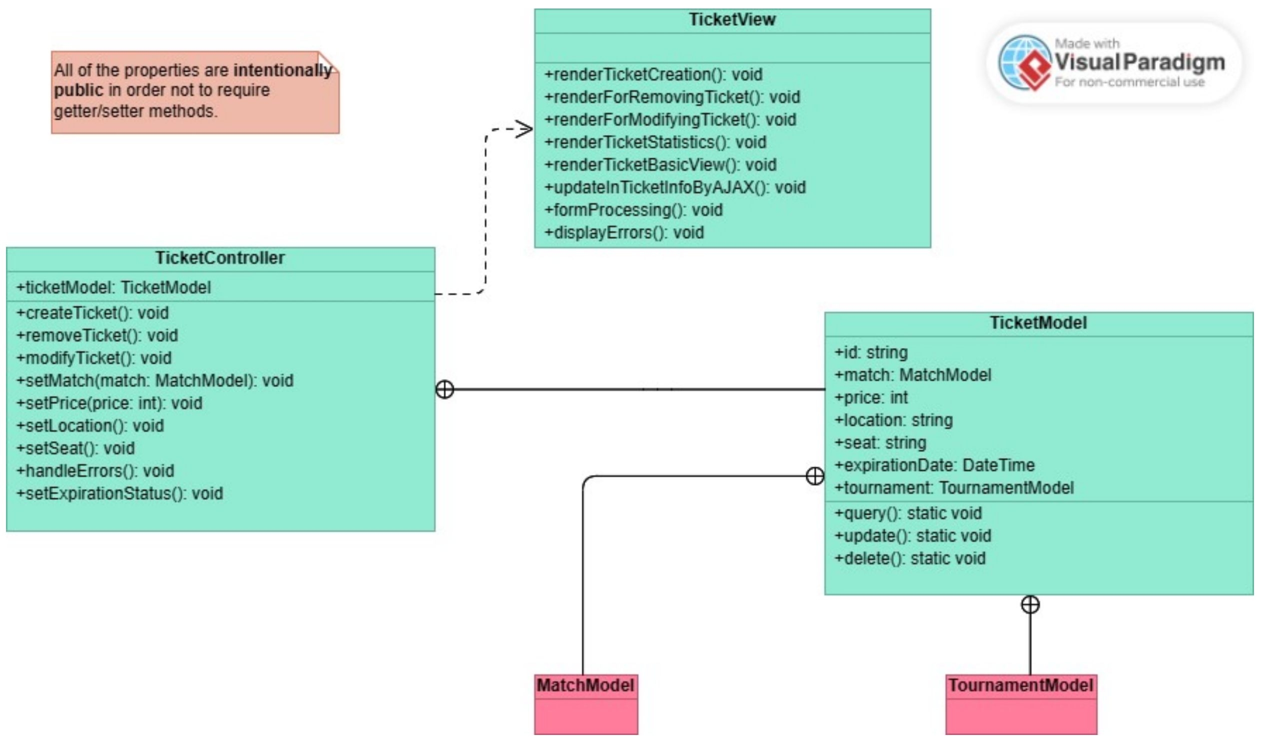
- beingProcessed: all of the registering teams, whose profiles that haven't been checked yet.
- approvedTeams: all of the teams whose profiles that have been approved.
- refusedTeams: all of the teams whose profiles that have been rejected.
- query(): supporting method to retrieve data from DB.
- update(): supporting method to update data in DB.
- delete(): supporting method to delete data in DB.

4.4.5 Others

- TeamModel: a team related to the indicated tournament. More details are defined in other component sections.

4.5 Component: Management of tournament tickets (Organizer)

4.5.1 Class diagram


All of the properties are intentionally public in order not to require getter/setter methods.
TicketView
+renderTicketCreation(): void
+renderForRemovingTicket(): void
+renderForModifyingTicket(): void
+renderTicketStatistics(): void
+renderTicketBasicView(): void
+updateInTicketInfoByAJAX(): void
+formProcessing(): void
+displayErrors(): void
Made with Visual Paradigm For non-commercial use
TicketController
+ticketModel: TicketModel
+createTicket(): void
+removeTicket(): void
+modifyTicket(): void
+setMatch(match: MatchModel): void
+setPrice(price: int): void
+setLocation(): void
+setSeat(): void
+handleErrors(): void
+setExpirationStatus(): void
TicketModel
+id: string
+match: MatchModel
+price: int
+location: string
+seat: string
+expirationDate: DateTime
+tournament: TournamentModel
+query(): static void
+update(): static void
+delete(): static void
MatchModel
TournamentModel

4.5.2 TicketView - graphical user interface

- Method renderTicketCreation(): return void type, display the interface for ticket creation, include in some information like: the ID of the ticket, the price, the match of the ticket, the location of the match, the seat info for customers, the expirationDate of the ticket (after the date of the match), the tournament of the ticket (optional).
- Method renderForRemovingTicket(): return void type, display a pop up window for users to make sure that the ticket will be removed from the database, and expect to select a specific ticket before this function.
- Method renderForModifyingTicket(): return void type, redisplay the interface for ticket modification, as the rendering for ticket creation.

- Method renderTicketStatistics(): return void type, show the statistics of tickets corresponding to.
- Method renderTicketBasicView(): return void type, display the item related to a ticket in a list of tickets (because the UI will be designed by component-based language: handlebars).
- Method updateInTicketInfoByAJAX(): return void type, automatically update the expiration status of the ticket without refreshing the webpage.
- Method formProcessing(): return void type, process the form inputs.
- Method displayErrors(): return void type, process to alert notification about errors.

  Expected to be written with Express-Handlebars, HTML, CSS, JS.

4.5.3 TicketController - the logic processing part

- Attribute ticketModel: type TicketModel, the model of the currently indicated ticket.
- Method createTicket(): return void type, help to initialize a new ticket in the database.
- Method removeTicket(): return void type, help to delete a specific ticket from the database.
- Method modifyTicket(): return void type, support modifying the information of the ticket based on the users inputs.
- Method setMatch(match: MatchModel): return void type with parameter match of MatchModel type, used to set the match for the indicated ticket.
- Method setPrice(price: int): return void type with parameter price of int type, support users to set the price for the ticket.
- Method setLocation(): return void type, allow the users to set the location of the match into the ticket.
- Method setSeat(): return void type, allow the users to set the seat information into the ticket.
- Method handleErrors(): return void type, handle the errors in the business flow.
- Method setExpirationStatus(): return void type, support to modify the expiration status of the ticket.

  Expected to be written with Node.js, Express.js.

4.5.4 TicketModel - the data part

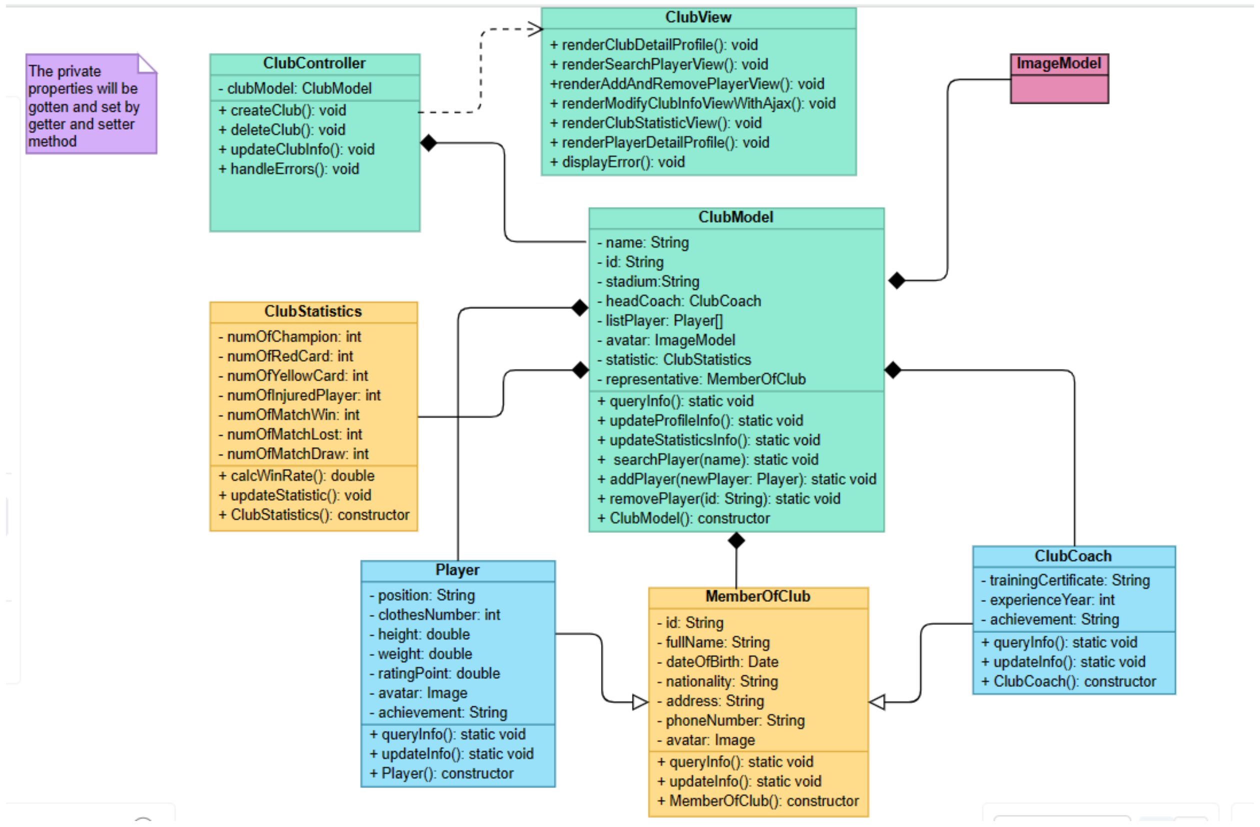
- Attribute id: type string, contain the id of the ticket to distinguish it to other tickets.
- Attribute match: type MatchModel, contain the match info of the ticket.
- Attribute price: type int, contain the price of the ticket.
- Attribute location: type string, contain the location of the match in the ticket.
- Attribute seat: type string, contain the seat information of the ticket.
- Attribute expirationDate: type DateTime, contain the expiration date of the ticket, used to know the current status of the ticket.
- Attribute tournament: type TournamentModel, contain the information of the tournament in the ticket, this information could be null (optional).
- Method static query(): supports connecting to the database and getting the necessary data.
- Method static update(): supports connecting to the database and updating the necessary records.
- Method static delete(): supports connecting to the database and deleting the necessary records.

4.5.5 Other related classes and note

- MatchModel: have a relationship with TicketModel due to the attribute match.
- TournamentModel: have a relationship with TicketModel due to the attribute tournament.
- Note: All of the properties are intentionally public in order not to require getter/setter methods.

4.6 Component: Football team management (club)

4.6.1 Class diagram


The private properties will be gotten and set by getter and setter method
ClubController
- clubModel: ClubModel
+ createClub(): void
+ deleteClub(): void
+ updateClubInfo(): void
+ handleErrors(): void
ClubView
+ renderClubDetailProfile(): void
+ renderSearchPlayerView(): void
+renderAddAndRemovePlayerView(): void
+ renderModifyClubInfoViewWithAjax(): void
+ renderClubStatisticView(): void
+ renderPlayerDetailProfile(): void
+ displayError(): void
ImageModel
ClubModel
- name: String
- id: String
- stadium:String
- headCoach: ClubCoach
- listPlayer: Player[]
- avatar: ImageModel
- statistic: ClubStatistics
- representative: MemberOfClub
+ queryInfo(): static void
+ updateProfileInfo(): static void
+ updateStatisticsInfo(): static void
+ searchPlayer(name): static void
+ addPlayer(newPlayer: Player): static void
+ removePlayer(id: String): static void
+ ClubModel(): constructor
ClubStatistics
- numOfChampion: int
- numOfRedCard: int
- numOfYellowCard: int
- numOfInjuredPlayer: int
- numOfMatchWin: int
- numOfMatchLost: int
- numOfMatchDraw: int
+ calcWinRate(): double
+ updateStatistic(): void
+ ClubStatistics(): constructor
Player
- position: String
- clothesNumber: int
- height: double
- weight: double
- ratingPoint: double
- avatar: Image
- achievement: String
+ queryInfo(): static void
+ updateInfo(): static void
+ Player(): constructor
MemberOfClub
- id: String
- fullName: String
- dateOfBirth: Date
- nationality: String
- address: String
- phoneNumber: String
- avatar: Image
+ queryInfo(): static void
+ updateInfo(): static void
+ MemberOfClub(): constructor
ClubCoach
- trainingCertificate: String
- experienceYear: int
- achievement: String
+ queryInfo(): static void
+ updateInfo(): static void
+ ClubCoach(): constructor

4.6.2. ClubView - user interface

- renderClubDetailProfile(): display detailed profile of club includes some information: name of club, home stadium, coach, list of players, logo of club, statistics on competition activities in the tournament.
- renderSearchPlayerView(): display a page for searching players in the club in order to view or modify information.
- renderAddAndRemovePlayerView(): display a page for representatives of clubs that can add new players when they join clubs or remove players when they move to another club.
- rederModifyClubInfoViewWithAjax(): display a list of detailed information of the club that allows the manager of the club to edit and modify information. Using Ajax helps the current page not to be reloaded.
- renderClubStatisticView(): display information about statistics on competition activities of the club in the tournament such as number of joining matches, number of champions, number of injured players, number of win matches, lost matches, etc.
- renderPlayerDetailProfile(): display detailed information of player such as full name, date of birth, nationality, avatar, compete position, clothes number, height, weight, etc.
- displayError(): display toast to notify error has occurred.

4.6.3. ClubController - processing logic of business flow

- clubModel: the model of the currently indicated club.
- createClub(): create a new club in order to register to participate in the tournament.
- deleteClub(): destroy club for many unexpected reasons like not having enough finances to operate or give up the tournament.
- updateClubInfo(): update club information of the club when it has changed.
- handleError(): handle when error has occurred.

4.6.4. ClubModel - data model:

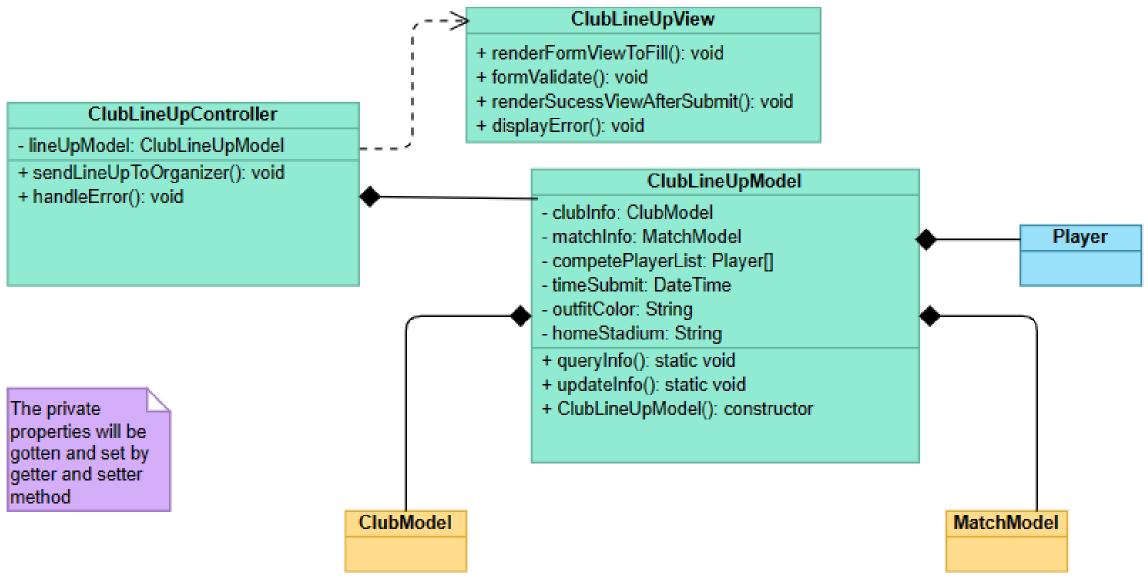
- name: name of club
- id: identification of club
- stadium: home stadium of club
- headCoach: coach of club
- listPlayer: list of players currently on the club's payroll.
- avatar: logo of club
- statistic: information of statistics on competition activities in the tournament.
- representative: a person who manages activities of the club.
- ClubModel(): constructor method for creating a new object of ClubModel.
- queryInfo(): supports connecting to databases to get necessary information.
- updateProfileInfo(): supports connecting to databases and updating information when these have been changed by the club manager.
- updateStatisticsInfo(): supports connecting to databases and updating statistics according to time and number of competition parameters.
- searchPlayer(name): supports connecting to databases and finding the player that matches with the name parameter.
- addPlayer(newPlayer): supports connecting to databases and adding a new record of a new player.
- removePlayer(id): supports connecting to databases and removes a record that matches with the id parameter. This method will be used when a player moves to another club or retires.

4.6.5. Others:

- The other classes take the supporting roles for the key classes (Model, View, Controller) in the current software component.
- About ImageModel, this class will be defined in other components in our software architecture.

4.7 Component: Registration of the lineup for participation in matches in the tournament (club)

4.7.1 Class diagram


ClubLineUpView
+ renderFormViewToFill(): void
+ formValidate(): void
+ renderSucessViewAfterSubmit(): void
+ displayError(): void
ClubLineUpController
- lineUpModel: ClubLineUpModel
+ sendLineUpToOrganizer(): void
+ handleError(): void
ClubLineUpModel
- clubInfo: ClubModel
- matchInfo: MatchModel
- competePlayerList: Player[]
- timeSubmit: DateTime
- outfitColor: String
- homeStadium: String
+ queryInfo(): static void
+ updateInfo(): static void
+ ClubLineUpModel(): constructor
Player
The private properties will be gotten and set by getter and setter method
ClubModel
MatchModel

4.7.2 ClubLineUpView - user interface

- renderFormViewToFill(): display a form for spectators to fill in their votes and send to organizers of the

tournament.
- formValidate(): receive and validate data from user input.
- renderSuccessViewAfterSumbit(): display status after sending registration of list players successfully.
- displayError(): display toast to notify error has occurred.

4.7.3 ClubLineUpController - processing logic of business flow

- lineUpModel: the model of the currently indicated line up activity.
- sendLineUpToOrganizer(): send information about registration of the club to the organizer for considering and storing.
- handleError(): handle when error has occurred.

4.7.4 ClubLineUpModel - data model

- clubInfo: detailed information of club
- matchInfo: information of the match that club registers for the squad.
- competePlayerList: list of players registered by club to compete in the indicated match.
- timeSubmit: the time that the club sends registration to the organizer of the tournament.
- outfitColor: information about competition outfit for the indicated match.
- homeStadium: stadium will be used in the indicated match.
- ClubLineUpModel(): constructor method for creating a new object of ClubLineUpModel.
- queryInfo(): supports connecting to databases and getting information for registration purposes.
- updateInfo(): supports connecting to databases and updating some information before registration activity.

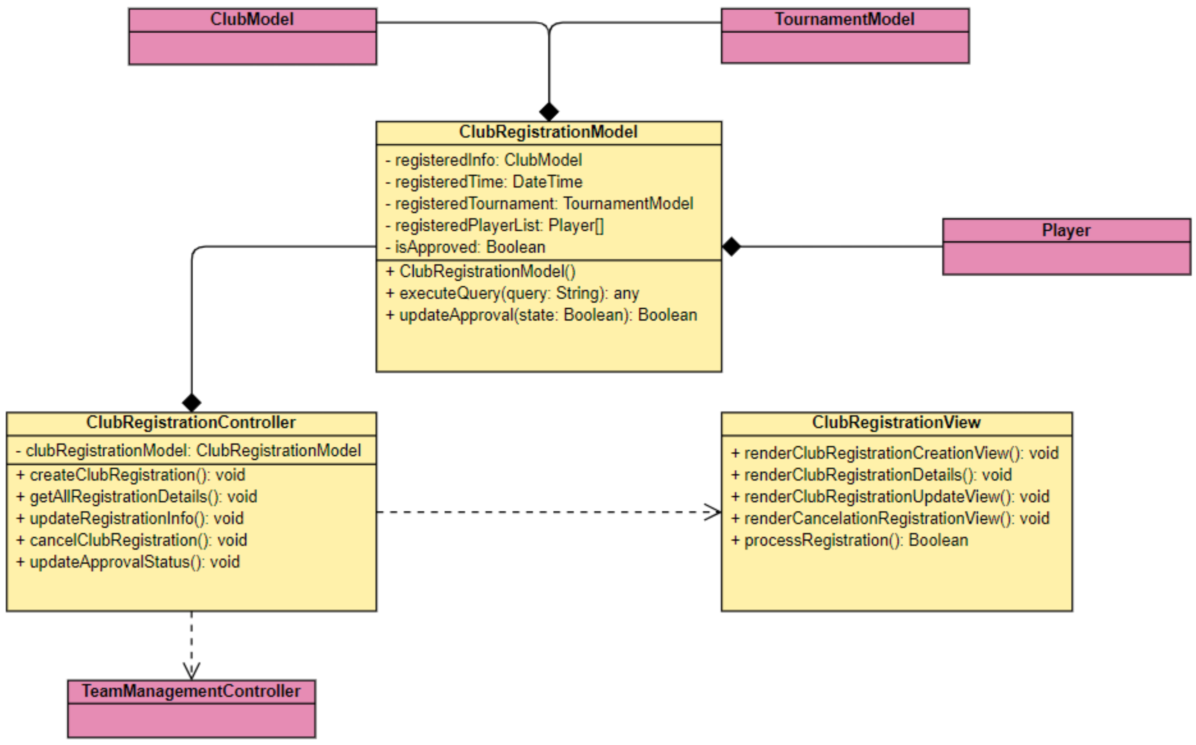
4.7.5 Others

- The other classes take the supporting roles for the key classes (Model, View, Controller) in the current software component.
- About ClubModel, MatchModel, Player, these classes will be defined in other components in our software architecture.

4.8 Component: Tournament participation registration (club)

4.8.1 Class diagram

(Because of insufficient blank, the diagram is put on the next page).


ClubModel
TournamentModel
ClubRegistrationModel
- registeredInfo: ClubModel
- registeredTime: DateTime
- registeredTournament: TournamentModel
- registeredPlayerList: Player[]
- isApproved: Boolean
+ ClubRegistrationModel()
+ executeQuery(query: String): any
+ updateApproval(state: Boolean): Boolean
Player
ClubRegistrationController
- clubRegistrationModel: ClubRegistrationModel
+ createClubRegistration(): void
+ getAllRegistrationDetails(): void
+ updateRegistrationInfo(): void
+ cancelClubRegistration(): void
+ updateApprovalStatus(): void
ClubRegistrationView
+ renderClubRegistrationCreationView(): void
+ renderClubRegistrationDetails(): void
+ renderClubRegistrationUpdateView(): void
+ renderCancelationRegistrationView(): void
+ processRegistration(): Boolean
TeamManagementController

4.8.2. ClubRegistrationView - user interface

- renderClubRegistrationCreationView(): display the form for a club representative to create a new registration to a specific tournament.
- renderClubRegistrationDetails(): display page for detailed information of the registration form.
- renderClubRegistrationUpdateView(): display a form for club representative to update information.
- renderCancelationRegistrationView(): display an announcement to cancel a registration, which means delete the registration.
- processRegistration(): Ensure and validate the information of the registration form.

4.8.3. ClubRegistrationController - processing logic of business flow

- createClubRegistration(): make a new Club registration to a specific tournament.
- getAllRegistrationDetails(): return all needed information about the registration to the view.
- updateRegistrationInfo(): handle the updated information given by the form from view.
- cancelClubRegistration(): cancel registration, which means remove all data about this registration from the database.
- updateApprovalStatus(): update the value of the isApproved property, which means confirm whether the registration has been approved or not.

4.8.4. ClubRegistrationModel - data model:

- registeredInfo: hold all related information about the club which wants to register to a tournament.
- registeredTime: save time data when the registration is created.
- registeredTournament: hold all related information about the tournament that the club wants to register to.
- registeredPlayerList: hold all related information about all the players that will be selected to compete in the tournament.
- isApproved: save the state whether the organizer of the tournament has approved for this registration or not.

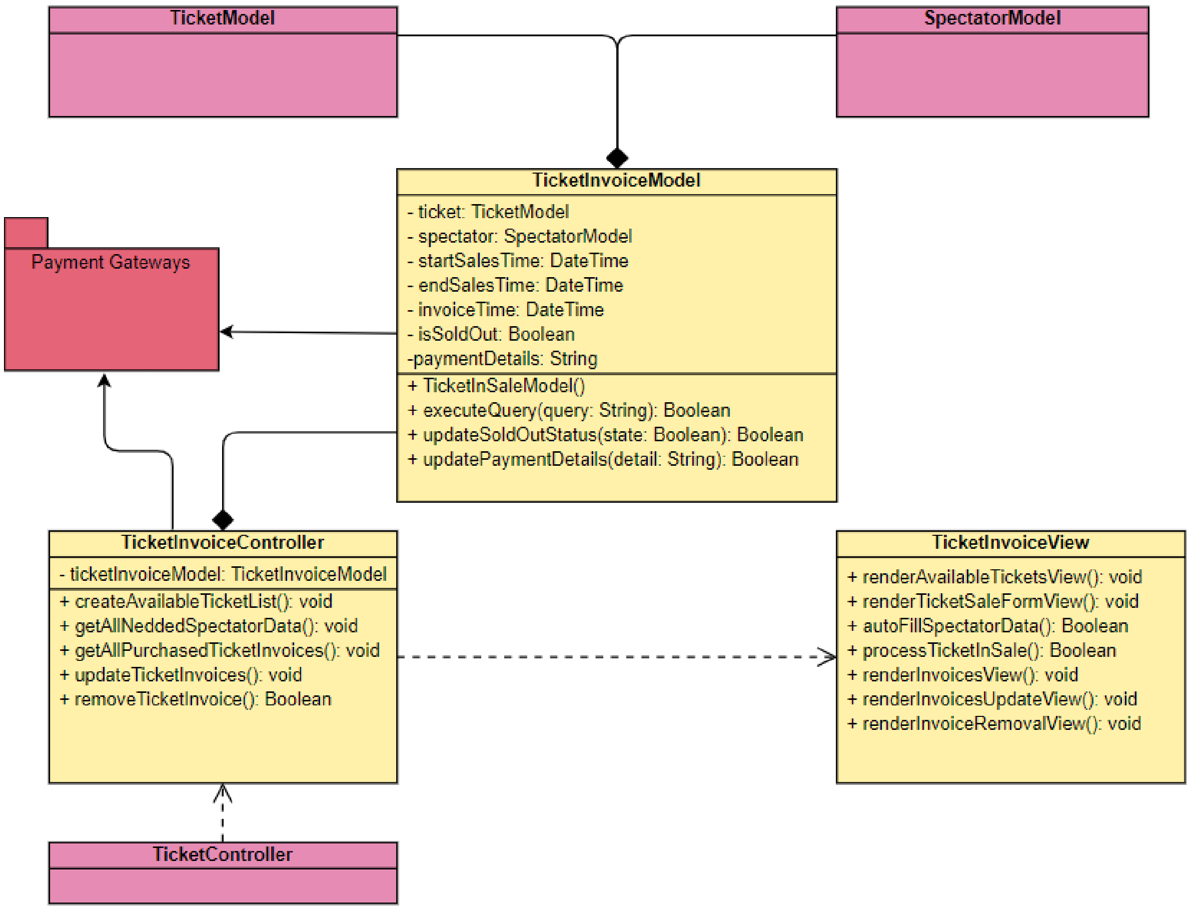
- ClubRegistrationModel(): a constructor for creating a new ClubRegistrationModel object.
- executeQuery(): execute the action given by the query string.
- updateApproval(): update the isApproval property.

4.8.5. Others:

- In this component, the other classes take the supporting roles for these 3 classes: Model - View - Controller.
- The player list in the registration must be a subset of the player list in the specific Club, also the number of players must meet the tournament's standard.

4.9 Component: Purchase and storage of match tickets (spectator)

4.9.1 Class diagram



TicketModel
SpectatorModel
TicketInvoiceModel
- ticket: TicketModel
- spectator: SpectatorModel
- startSalesTime: DateTime
- endSalesTime: DateTime
- invoiceTime: DateTime
- isSoldOut: Boolean
-paymentDetails: String
+ TicketInSaleModel()
+ executeQuery(query: String): Boolean
+ updateSoldOutStatus(state: Boolean): Boolean
+ updatePaymentDetails(detail: String): Boolean
Payment Gateways
TicketInvoiceController
- ticketInvoiceModel: TicketInvoiceModel
+ createAvailableTicketList(): void
+ getAllNeddedSpectatorData(): void
+ getAllPurchasedTicketInvoices(): void
+ updateTicketInvoices(): void
+ removeTicketInvoice(): Boolean
TicketInvoiceView
+ renderAvailableTicketsView(): void
+ renderTicketSaleFormView(): void
+ autoFillSpectatorData(): Boolean
+ processTicketInSale(): Boolean
+ renderInvoicesView(): void
+ renderInvoicesUpdateView(): void
+ renderInvoiceRemovalView(): void
TicketController

4.9.2. TicketInvoiceView - user interface

- renderAvailableTicketsView(): display the page containing a list of tickets, each with the basic information for a ticket of a match, and the number of the remaining tickets.
- renderTicketSaleFormView(): display a form for the spectator to purchase a specific ticket.
- autoFillSpectatorData(): fill all the data relating to the spectator automatically by the data in the system database.
- processTicketSale(): Ensure and validate the information of the ticketInSale form.
- renderInvoicesView(): display a page containing a list of ticket invoices purchased by the spectator, each with all information filled by the ticketInSale form.
- renderInvoicesUpdateView(): display a form for the spectator to update the information inside the ticketInSale

form.
- renderInvoiceRemovalView(): display an announcement to cancel the ticket invoice, which means delete the invoice.

4.9.3. TicketInvoiceController - processing logic of business flow

- renderAvailableTicketList(): make a new ticket invoice for a specific spectator..
- getAllNeededSpectatorData(): return the related data of the specific spectator which will be filled in the ticketInSale form.
- getAllPurchasedTicketInvoices():return all the ticket invoices of a specific spectator.
- updateTicketInvoice(): handle the updated information given by the form from updateView.
- removeTicketInvoice(): remove all data about this invoice from the database.

4.9.4. TicketInvoiceModel - data model:

- ticket: hold all related information about the ticket which the spectator wants to purchase.
- spectator: hold all related information about the spectator who wants to purchase a specific ticket.
- startSalesTime: the time when the ticket is starting being purchased.
- endSalesTime: the time when the ticket cannot be purchased anymore.
- invoiceTime: the time when the ticket invoice is created.
- isSoldOut: the state whether the number of the specific type of this ticket is out of stock.
- paymentDetails: hold all related information about the payment methods, payment account, …
- TicketInvoiceModel(): constructor for creating a new TicketInvoiceModel object.
- executeQuery(): execute an action given by the query string.
- updateSoldOutStatus(): update the isSoldOut property.
- updatePaymentDetails(): update the paymentDetails property.

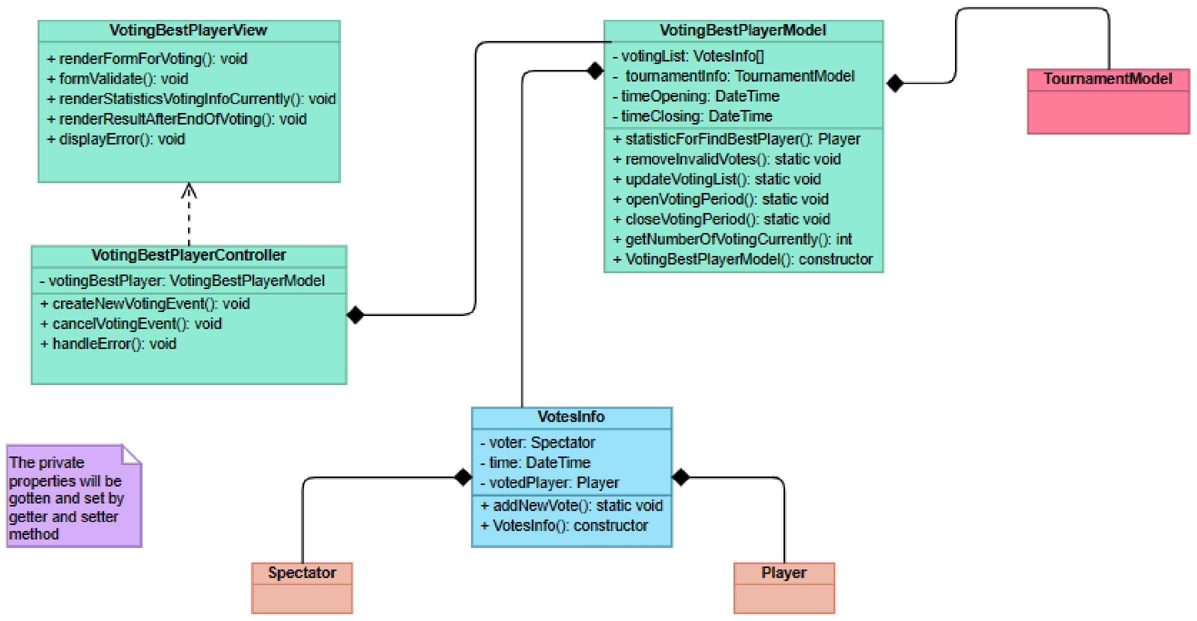
4.9.5. Others:

- In this component, the other classes take the supporting roles for these 3 classes: Model - View - Controller.
- The invoiceTime must be between the startSalesTime and endSalesTime, also these 2 properties must synchronize with the data in the specific TicketModel.
- The isSoldOut state must be updated continuously, in order to handle the ticketInSale synchronously.

4.10 Component: Footballer voting (spectator)

4.10.1 Class Diagram

(Because of insufficient blank, the diagram is put on the next page).


VotingBestPlayerView
+ renderFormForVoting(): void
+ formValidate(): void
+ renderStatisticsVotingInfoCurrently(): void
+ renderResultAfterEndOfVoting(): void
+ displayError(): void
VotingBestPlayerController
- votingBestPlayer: VotingBestPlayerModel
+ createNewVotingEvent(): void
+ cancelVotingEvent(): void
+ handleError(): void
VotingBestPlayerModel
- votingList: VotesInfo[]
- tournamentInfo: TournamentModel
- timeOpening: DateTime
- timeClosing: DateTime
+ statisticForFindBestPlayer(): Player
+ removeInvalidVotes(): static void
+ updateVotingList(): static void
+ openVotingPeriod(): static void
+ closeVotingPeriod(): static void
+ getNumberOfVotingCurrently(): int
+ VotingBestPlayerModel(): constructor
TournamentModel
VotesInfo
- voter: Spectator
- time: DateTime
- votedPlayer: Player
+ addNewVote(): static void
+ VotesInfo(): constructor
The private properties will be gotten and set by getter and setter method
Spectator
Player

4.10.2. VotingBestPlayerView - user interface:

- renderFormForVoting(): display a form for spectators to fill in and send their votes.
- formValidate(): receive and validate data from user input.
- renderStatisticsVotingInfoCurrently(): display statistics about voting currently includes the number of votes, number of players who have been voted.
- renderResultAfterEndOfVoting(): display the result player who has the highest number of votes and this player will get Best Player Awards.
- displayError(): display toast to notify error has occurred.

4.10.3. VotingBestPlayerController: processing logic of business flow

- votingBestPlayer: the model of the currently indicated Voting Event.
- createNewVotingEvent(): create a new voting event for the organizer to find the best player through spectator voting.
- cancelVotingEvent(): stop voting event for some unexpected reasons such as the tournament has been paused or canceled.
- handleError(): handle when error has occurred.

4.10.4 VotingBestPlayerModel - data model:

- votingList: array of votes received from spectators.
- tournamentModel: information of the current tournament in the voting event.
- timeOpening/timeClosing: time that voting event opens and closes.
- VotingBestPlayerModel(): constructor method for creating a new object of VotingBestPlayerModel.
- statisticForFindBestPlayer(): filter and sort the number of votes to find the player with the highest number of votes.
- removeInvalidVotes(): remove list of votes that is invalid such as votes not in the voting period.
- updateVotingList(): update list of votes when receive a new valid vote.
- openVotingPeriod(): open event for spectators can vote for their favorite player.
- closeVotingPeriod(): close event when tournament has finished in order to statistics to find best player.
- getNumberOfVotingCurrently(): get the number of votes currently for spectators known before filling in the voting form.

4.10.5. Others:

- The other classes take the supporting roles for the key classes (Model, View, Controller) in the current software component.
- About Spectator, Player and TournamentModel, these classes will be defined in other components in our software architecture.

4.11 Component: Confirmation of information for spectators

4.11.1 Class Diagram

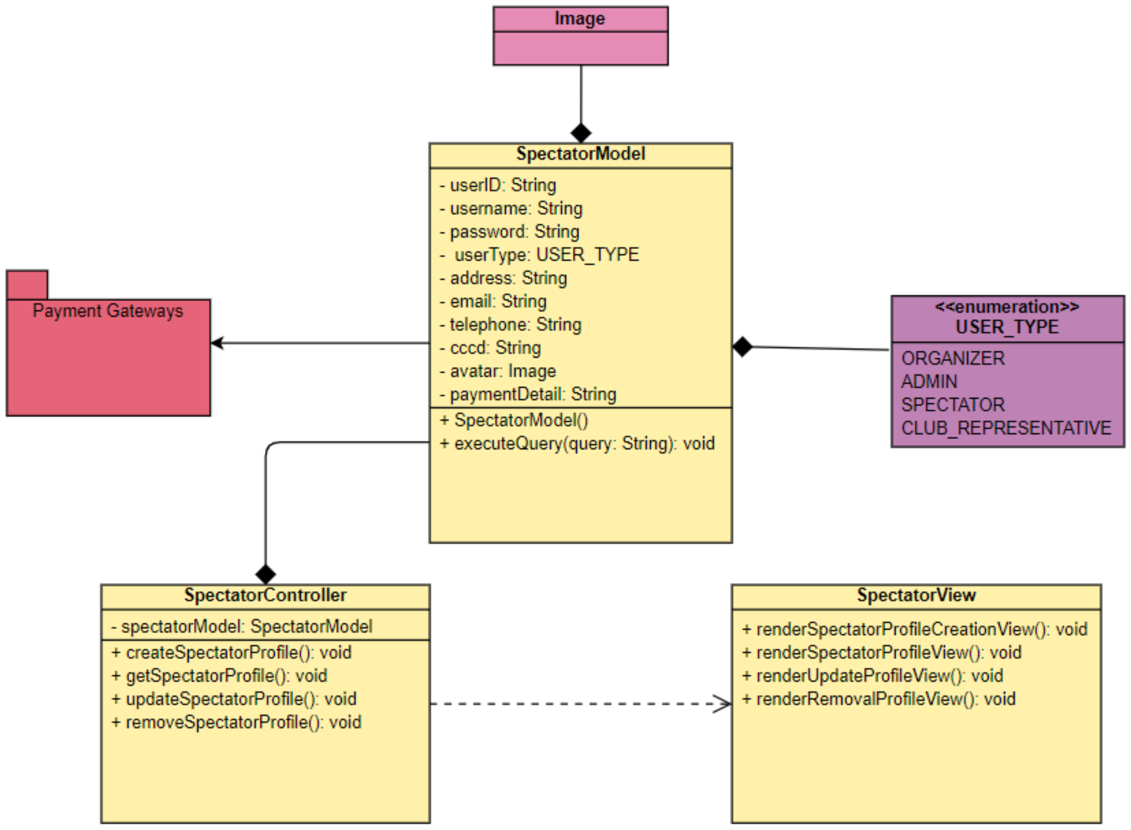


Image
SpectatorModel
- userID: String
- username: String
- password: String
- userType: USER_TYPE
- address: String
- email: String
- telephone: String
- cccd: String
- avatar: Image
- paymentDetail: String
+ SpectatorModel()
+ executeQuery(query: String): void
Payment Gateways
<<enumeration>>
USER_TYPE
ORGANIZER
ADMIN
SPECTATOR
CLUB_REPRESENTATIVE
SpectatorController
- spectatorModel: SpectatorModel
+ createSpectatorProfile(): void
+ getSpectatorProfile(): void
+ updateSpectatorProfile(): void
+ removeSpectatorProfile(): void
SpectatorView
+ renderSpectatorProfileCreationView(): void
+ renderSpectatorProfileView(): void
+ renderUpdateProfileView(): void
+ renderRemovalProfileView(): void

4.11.2. SpectatorView - user interface:

- renderSpectatorProfileCreationView(): display a form for a spectator to fill in and complete their own profile.
- renderSpectatorProfileView(): display a page containing the specific profile of the spectator.
- renderUpdateProfileView(): display a form for a spectator to update the information inside the profile form.
- renderRemovalProfileView(): display an announcement to remove the profile of a specific spectator.

4.11.3. SpectatorController: processing logic of business flow

- createSpectatorProfile(): make a new profile for a spectator if they don't have one.
- getSpectatorProfile(): return all the related data of the spectator's profile if any.
- updateSpectatorProfile(): handle the updated information given by the form from updateView.
- removeSpectatorProfile(): remove all data about this profile from the database.

4.11.4 SpectatorModel - data model:

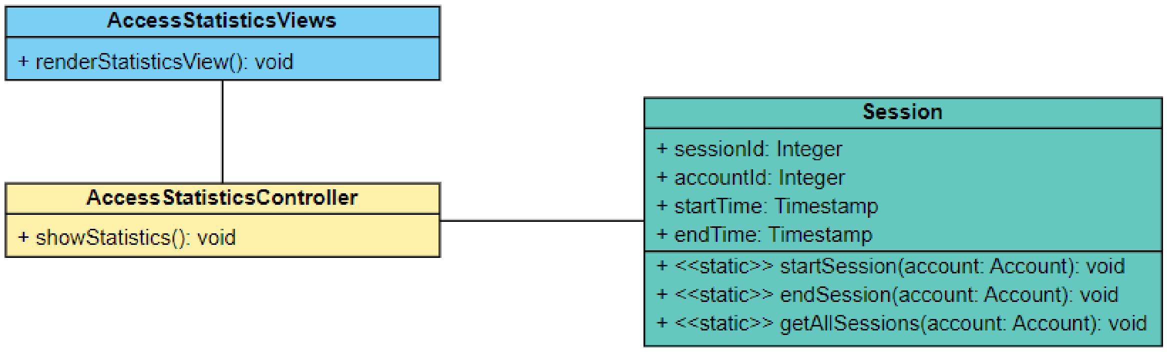
- userID: a unique id to separate a specific spectator with another.
- username: a string containing the name which the spectator wants the system to call them by.
- password: a hashed string containing the password for the spectator account to authenticate.
- userType: an instance of the enum class USER_TYPE, used to separate all users into manageable types.
- address: a string containing the spectator's address, in order to confirm the authentication.
- email: a string containing the spectator's email, in order to confirm the authentication and to send mail for any latest announcement.
- telephone:a string containing the spectator's telephone number, in order to confirm the authentication.
- cccd: a string containing the spectator's citizen identification card number, in order to confirm the authentication.
- avatar: an instance of the Image class, which contains the data for a spectator image.
- paymentDetails: a string containing the spectator's payment information, such as: methods, account, …
- SpectatorModel(): a constructor for creating a new Spectator.
- executeQuery(): execute an action given by the query string.

4.11.5. Others:

- The other classes take the supporting roles for the key classes (Model, View, Controller) in the current software component.
- A spectator can only have a maximum of 1 profile at a time.
- The userType must be a SPECTATOR.

4.12 Component: Access statistics (spectator)

4.12.1 Class diagram


AccessStatisticsViews
+ renderStatisticsView(): void
AccessStatisticsController
+ showStatistics(): void
Session
+ sessionId: Integer
+ accountId: Integer
+ startTime: Timestamp
+ endTime: Timestamp
+ <<static>> startSession(account: Account): void
+ <<static>> endSession(account: Account): void
+ <<static>> getAllSessions(account: Account): void

4.12.2 AccessStatisticsViews
- renderStatisticsView(): Render a view displaying the user's access level.

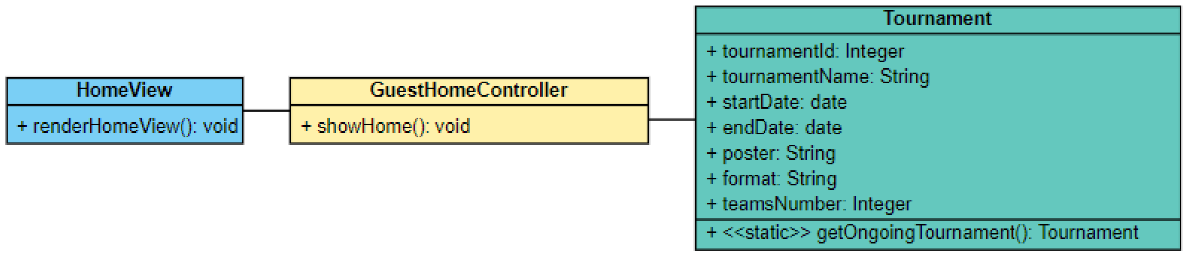
4.12.3 AccessStatisticsController
- showStatistics():Compile statistics from the data provided by the session model and call the renderStatisticsView function.

4.12.4 Session
- sessionId: Integer variable storing id of this session.
- accountId: Integer variable storing id of this account.
- startTime: Timestamp variable containing the beginning time of the work session.
- endTime: Timestamp variable containing the ending time of the work session.
- startSession(): Create a new session.
- endSession(): End current session.
- getAllSessions(): Get all sessions of a given account.

4.13 Component: Information view provided by the website for guests (guest)

4.13.1 Home Page

Class diagram



HomeView

- renderHomeView(): Render home view displaying tournament information.

GuestHomeController

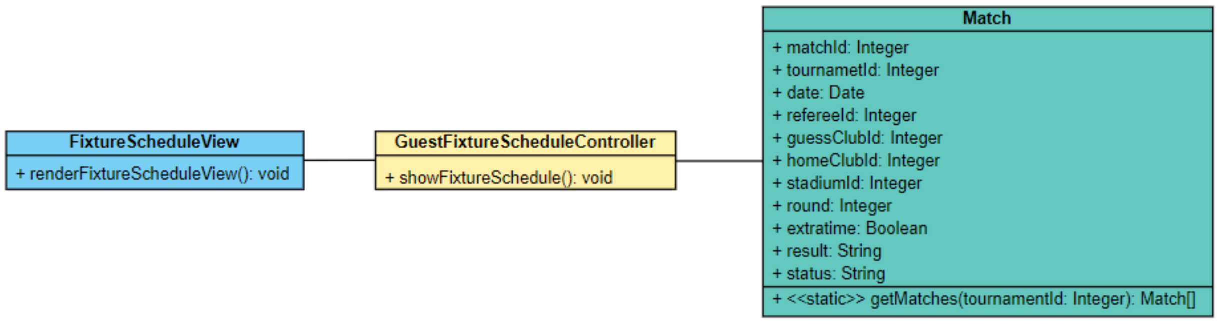
- showHome(): Get information from Tournament model and call renderHomeView().

Tournament:

- tournamentId: Integer variable storing id of the tournament.
- tournamentName: String variable storing name of the tournament.
- startDate: Date variable storing beginning date of the tournament.
- endDate: Date variable storing ending date of the tournament.
- poster: String variable storing path of poster.
- format: String variable storing format of the tournament.
- teamsNumber: Integer variable storing number of teams participating in the tournament.
- getOngoingTournament(): Get a Tournament object that is on going.

4.13.2 Class diagram

Class diagram



FixtureSchduleView

- renderFixtureScheduleView(): Render view displaying fixture schedule of the tournament.

GuestFixtureScheduleController

- showFixtureSchedule(): Get data from Match model and call renderFixtureScheduleView().

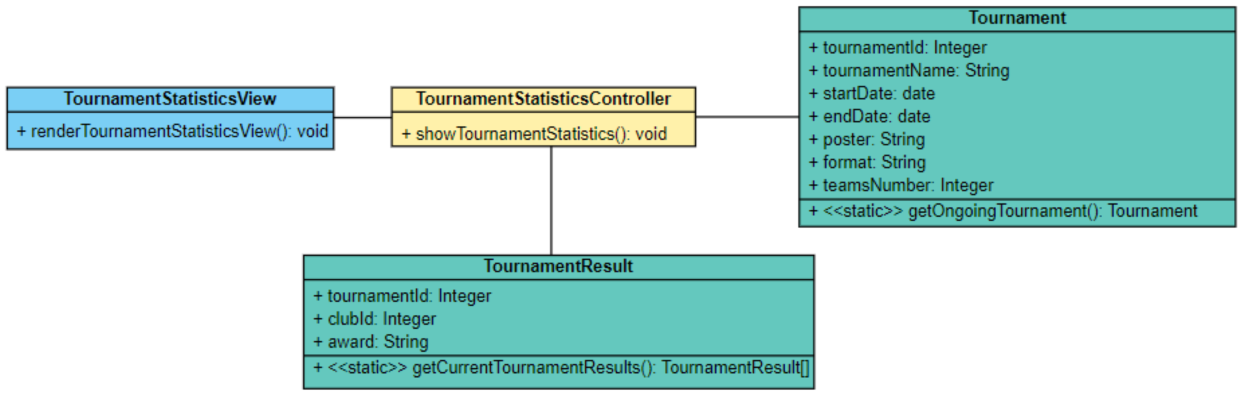
Match

- matchId: Integer variable storing id of the match.
- tournamentId: Integer variable storing id of the tournament.
- date: Date variable storing beginning date of the tournament.
- refereeId: Integer variable storing id of the referee.
- guessClubId: Integer variable storing id of the guess club.
- homeClubId: Integer variable storing id of the home club.

- stadiumId: Integer variable storing id of the stadium club.
- round: Integer variable storing round of the match.
- extratime: Boolean variable stores a true value if there is a extratime and false otherwise.
- result: String variable storing result of the match.
- status: String variable storing status of the match.
- getMatches(): Get all matches of the given tournament..

4.13.3 Class diagram

Class diagram



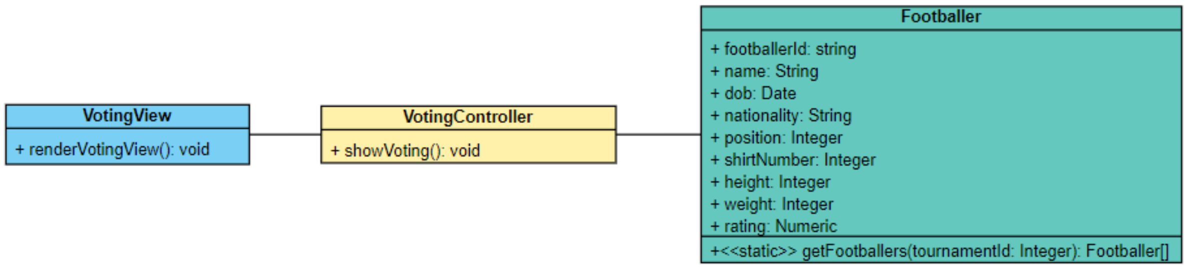
TournamentStatisticsView
- renderTournamentStatisticsView(): Render view displaying tournament statistics.

TournamentResult
- tournamentId: Integer variable storing id of TournamentResult object.
- clubId: Integer variable storing id of the club.
- award: String variable storing name of the award.
- getCurrentTournamentResults(): Get all TournamentResult objects of the currently ongoing Tournament.

Tournament
- tournamentId: Integer variable storing id of the tournament.
- tournamentName: String variable storing name of the tournament.
- startDate: Date variable storing beginning date of the tournament.
- endDate: Date variable storing ending date of the tournament.
- poster: String variable storing path of poster.
- format: String variable storing format of the tournament.
- teamsNumber: Integer variable storing number of teams participating in the tournament.
- getOngoingTournament(): Get a Tournament object that is on going.

4.13.4 Class diagram

Class diagram



VotingView
- renderVotingView(): Render a view displaying the player evaluation results in this tournament.

VotingController
- showVoting(): Get data from Footballer model and call renderVotingView().

Footballer
- footballerId: Integer variable storing id of the footballer.
- name: String variable storing name of the footballer.
- dob: Date variable storing date of birth of the footballer.
- nationality: String variable storing nationality of the footballer..
- position: String variable storing position of the footballer..
- shirtNumber: Integer variable storing shirt number of the footballer..
- height: Integer variable storing height of the footballer.
- weight: Integer variable storing weight of the footballer.
- rating: Integer variable storing rating of the footballer.
- getFootballers(): Get all footballers participating in this tournament.

5. Deployment

6. Implementation View