DESIGN REPORT - 1

Group - 26
Alexander Björnsson, Ernir Elí Ellertsson, Ívar Orri Tómasson
Nökkvi Benediktsson, Olgeir Otri Engilbertsson

12 DECEMBER 2025
T-113-VLN1
Dr. Gylfi Þór Guðmundsson

Contents

Introduction

The premise of this design report is to get the basic wireframe of our program mapped, such as the requirements, use cases, state diagram and class diagram. The program itself is an e-Sports tournament maker managing a tournament and recording all the events of the tournament such as results, winners etc. Besides that, it also manages other parts of the event such as players, team captains and organizers which can make modifications to the database if needed. The program will be written in Python and the deadline for the beta version of the is December the 12th

Requirements

- We noted down every requirement for our system for it to run, classified by severity from A to C. They are also classified by Functional and Nonfunctional requirements.
- A requirements is a list of all the requirements needed for the system to run at its core and features that must come forward in the final product such as organizers being able to create tournaments with the needed information to make it
- B requirements on other hand is a list of features that are not necessarily needed for the program but nice additions to have for more data and sophisticated results
- C requirements are mostly made up of our own ideas that are seemingly hard to implement therefore come as the least important requirement as its not necessarily to be implemented, note if any C requirements are to be implemented then all or most B requirements must already be fulfilled.
- Updated changes: We've updated a few B requirements and managed to implement all of them except for one. An example of an implemented B requirement is the club system. all the functionality for the club where you can add, remove and view all data on the club and what teams are included in said tournament amongst other data has been implemented. The "status" column in the requirements has been updated to display which of the requirements are implemented, A and B requirements are fully implemented and fully functional whereas all except one C requirement has been implemented.

A requirement:

Functional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info	Status
Players:					
A1	As a user I want to be able to store players personal data	Organizer, Captain	A	Name, date of birth, home address, phone number, email, link, handle & team	Implemented
A2	As a Captain I want to be able to modify a players data	Captain	A	modify phone number, email, address, links & handle	Implemented
A3	As an organizer I want to be able to modify a players data	Organizer	A	modify phone number, email, address, links & handle	Implemented
Teams:					
A4	As a user I want the system to store all team data	System	A	Unique name, One captain, Link to web page, ASCII logo	Implemented
A5	As an organizer I want a Team to only to be created once	Organizer	A	Same team can only be created once	Implemented

Organizer:					
A6	As an organizer I want to be able to create a	Organizer	A	Name, Captain Web Link Ascii Players in team	Implemented
A8	As an organizer I want to be able to create a tournament	Organizer	A	ID, Name, Venue, Start date, End date, Contract, Contact email, contact_number, num_of_servers, teams_in_tournament	Implemented
Tournament Data:					Work in progress
A9	As a user I want the system to store tournament information	System	A	Start date, End date, Unique name, Venue, Contract, Contact person (email + phone)	Implemented
A10	As a organizer I want the system to create a game schedule automatically after I create the tournament	System	A	Knockout style: generated after all data is registered	Implemented
Games					

And Results:					
A11	As a Organizer I want to input match results	Organizer	A		Implemented
A12	As a user I want the system to update schedule after results	System	A	Completed games show results; future matches update	Implemented
Information retrieval:					
A13	As a user I want to view game schedule	Users	A	Completed games show results; future games show team + time	Implemented
A14	As a user I want to see a list of all the teams	Users	A	Name Club Tournaments played Wins	Implemented
A15	As a user I want to view the tournaments a team has participated in	Users	A	Tournaments played	Implemented
A16	As a user I want to see a teams roster	Users	A	Team name, captain, club, web link, Ascii, tournaments played in, tournaments won,	Implemented

				runner up times, teams players	
A17	Lock schedule after creation	System	A	Teams and players cannot be charged	Implemented
A18	As an organizer or captain I want to be able to see private data of the players	Organizer, Captain	A	Handle, Team name, Date of birth, Address, Phone number, Email, Link, Tournaments played, Tournaments won,	Implemented

Nonfunctional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info	Status
A19	For all inputs, user shall know what to input	User	A	Users should know what the input possibilities are, every time.	Implemented

B Requirements

Functional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info	Status
B1	Support double elimination	Organizer	B	Loser's bracket needed	Not implemented
B2	Allow teams to play multiple games at a tournament	Organizer	B		Implemented
B3	Support clubs: As a user I want to be able to make a club	Organizer	B	Club has a name, colors, hometown, country	Implemented
B4	As a user I want the system to award points after matches and tournaments	System	B	Points for players, teams, clubs	Implemented
B5	As a user I want to see players statistics	Users	B	Wins, earnings, clubs played for, tournaments played	Implemented
B6	As a user I want to see team's statistics	Users	B	Tournaments won, total wins, total earnings	Implemented

B7	As a user I want to see Club statistics	Users	B	Sort teams by earnings, wins, etc.	Not implemented (X)
B8	As a user I want to see all the clubs in the system Clubs	Users	B	Name Country Tournaments played Wins	Implemented
B9	As an organizer I want to be able to add team to club	Organizer	B	Team name	Implemented
B10	As an organizer I want to be able to remove team from club	Organizer	B	Team name	Implemented
B11	As a user I want to be able to remove a player from a team	Organizer	B	Player handle	Implemented
B12	As a User I want to be able to add players to a team	Organizer	B	Player handle	Implemented

Nonfunctional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info
B9	Every UI shall be a formatted table	System	B	

C Requirements

Functional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info	Status
C1	New team created for each tournament	Organizer	C	Complicates statistics	Not implemented ekki
C2	As an organizer I want the system to Process results files automatically	Organizer	C	Imports instead of manual entry	Implemented
C3	Players can be in more than one team	System	C		Not implemented
C4	Login requires player handle & password	System	C		Not implemented

Nonfunctional requirements:

Number	Name (and possibly a short description)	User group(s)	Priority (A/B/C)	Additional info
C5	Club statistic page should have its colors.	System	C	The formatting on the club page should consist of the clubs’ colors. ekki

Use Cases

Our Use cases consist of every requirement and their corresponding priority, each use case is listed with a source, precondition, base flow, alternative flow post condition, actors and those who wrote the use case as authors.

- Precondition tells us what must be true so the user can do the actions needed to reach their goal.
- Base flow shows a route the user takes to reach their goals including what they need to do on each step.
- Alternative flow shows what could go wrong on the route such as an invalid input or a false statement that raises an exception.
- Post condition is what happens after the user has reached their goals and what turns true.
- Actors are the type of users going through the use case and getting to their goal.

Use Cases 1 --- “Store player personal data”

Name:	As a captain or organizer I want to be able to create a new player
Number:	UC – 01
Priority:	A
Source	A1
Precondition:	The team the player belongs to already exists.
Base flow:	1. User selects “Create new player”. 2. System asks for Name. • User inputs a non-empty, valid name. • System validates the name successfully. 3. System asks for Date of birth in the format day-month-year. • User inputs a valid date of birth in the correct format. • System validates that the date exists and that the player’s age is within the allowed range. 4. System asks for Home address. • User inputs a non-empty home address. • System validates the address successfully. 5. System asks for **Phone number** as +354 XXXXXXX. • User inputs the local part of the phone number. • System prefixes it with 354, validates the full number, and the validation succeeds. 6. System asks for Email. • User inputs a valid email address. • System validates the email successfully. 7. System asks for Link and displays the prefix https://. • User inputs the rest of the link (e.g. example.com). • System prefixes with https://, validates the link, and the validation succeeds.

	8. System asks for Handle. • User inputs a valid handle with allowed characters that does not already exist. • System validates the handle successfully. 9. System calls the logic layer to create the player with name, dob, address, number, email, link, handle. 10. System confirms that the player has been created.
Alternative flow:	**A1: User selects "Create new player"** • System asks for one of the required fields (Name / Date of birth / Address / Phone number / Email / Link / Handle). • User presses Enter without typing anything. • System raises EmptyInput and prints a message that the value is required (e.g. "Player needs to have a name"). • System asks the user to enter the same field again. **A2: User selects "Create new player"** • System asks for Date of birth in format day-month-year. • User enters a value in the wrong format, a date that does not exist, or a date that makes the player too young or too old. • System raises one of: TooYoungError, TooOldError, InvalidAgeException, or DateDoesNotExistError and prints the corresponding error message. • System asks the user to enter the date of birth again. **A3: User selects "Create new player"** • System asks for Phone number (+354 prefix). • User enters a value that is not a valid number. • System raises invalidNumberException and prints "Number is invalid, try again". • System asks the user to enter the phone number again.

	A4: User selects "Create new player" • System asks for Email. • User enters a value that is not in a valid email format. • System raises InvalidEmailException and prints "Email is invalid, try again". • System asks the user to enter the email again. **A5: User selects "Create new player"** • System asks for **Link** and shows the prefix https://. • User enters the rest of the link in an invalid format (for example, missing a dot). • System raises InvaldlinkException and prints an error message (e.g. "Link has to contain a dot"). • System asks the user to enter the link again. **A6: User selects "Create new player"** • System asks for Handle. • User enters a handle with invalid characters or a handle that already exists. • System raises InvalidCharacterHandle or HandleExistsException and prints the corresponding error message. • System asks the user to enter a new handle.
Post condition	The player is added to the system with all their info. If something fails, nothing is saved.
Actors:	Organizer, Captain
Authors:	Nökkvi Benediktsson

Use Case 2 --- “Modify player info”

Use case name	As a captain or organizer I want to be able to change certain info on the player
Number	UC – 02
Priority	A
Source	A2, A3
Precondition	User is Captain or organizer A player is available to modify
Base flow	1. User selects “Edit Player”. 2. User types in the handle of the player he wants to modify the data of. 3. User does changes to each of the following data: Phone, Email, Address and link. 4. System validates the changes which in turn makes it True, delivering the string “"Modifications to player info have been made." In the UI layer
Alternative flow	A1: User tries to change the phone number but inputs an invalid phone number System raises the exception class that prints out an error message A2: User tries to change the Email but doesn’t input a Email with the valid form System raises the exception class that prints out an error message that explains what went wrong
Post condition	Changes are saved and updated to the database.
Actors	Captains / Organizers
Authors	Alexander Björnsson

Use case 3 --- “Create New Team”

Use case name	As an organizer I want to be able to create a new team
Number	UC – 03
Priority	A
Source	A4 A5 A6
Precondition	User is an Organizer Data layer is available Team name to be registered does not yet exist in the system User is in “team menu”
Base flow	1. User Selects “Create new team” 2. System asks for Team name and validates that it’s not empty nor already in use 3. System asks for Web link and validates that it’s not empty nor that it’s Invalid. 4. System asks for Ascii logo and validates that it’s not empty nor that it’s Invalid 5. System asks for the user to input how many players will be on the team. And does the following Validations: • Raise exception if the input is a digit. • If the input is not empty. • If the amount is less than five. • If the amount is less than two 6. System asks for user handle to add to the team 7. System validates if the handle of the player exists, if the player is already in the team, and if the player is already on the team. If any of these validations raises an exception then it prints out an error message for said exception

	8. If no exceptions are raised the system appends the player to an empty lists and does that for every player that has been validated 9. System asks for the user to choose a Captain by handle 10. System tries to validate if the handle provided is in the list where previous players were added. If not it raises one of the following exceptions 11. System sends data down to the Logic layer which sends it down to the database and updates team file
Alternative flow	A1: User selects "Create new team" • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System raises exception when user does not input anything and leaves it empty, with a corresponding error message A2: User selects "Create new team • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System validates the format of Web link, Ascii logo, and Amount of players • System raises an exception with a corresponding error message when any of these are in an invalid format (e.g. Web link not a valid URL, Ascii logo invalid, Amount of players not a number) A3: User selects "Create new team • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System validates that the Amount of players is within the allowed range • System raises an exception with a corresponding error message when the amount of players is below the minimum required (e.g. less than 2 and not more than 5, according to the rules)

	A4. User selects “Create new team • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System validates if the entered Player handle or Captain handle exists in the system • System raises an exception with a corresponding error message when the handle does not exist A5. User selects “Create new team • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System checks if the Team name is already in use and if the Player handle is already in this team or already assigned to another team • System raises an exception with a corresponding error message when any of these values are already in use A6. User Selects “Create new team” • System asks for (Team name / Web link / Ascii logo / Amount of players / Player Handle / Captain handle) • System checks that the chosen Captain handle is in the list of players added to the team • System raises an exception with a corresponding error message when the captain handle is not found in the player list
Post condition	1. A new team record is registered 2. Unique team name 3. All data is added to database
Actor	Organizer
Author	Alexander Björnsson

Use Case 4 --- “Organizers can create tournaments”

Use case name	As an organizer of a tournament, I want to be able to make one
Number	UC – 04
Priority	A
Source	A8 A9 A17
Precondition	User picks Organizer and then “Tournament Menu” There are teams available to add to the tournament
Base flow	1. User selects “Create new tournament”**.** 2. System asks for Tournament name and user enters a valid, non-empty name. 3. System asks for Tournament id and user enters a valid, non-empty id that does not already exist. 4. System asks for Start date and End date in the format day-month-year. • User enters valid dates that exist, • Start date is not in the past, • End date is after start date, • Tournament length is between 2–7 days, • Format is correct. 5. System asks for Venue and user enters a valid, non-empty venue. 6. System asks for Contract and user enters a valid, non-empty contract reference. 7. System asks for Contact person’s email and user enters a valid email address. 8. System asks for Contact person’s phone number (+354) and user enters a valid number; system prefixes with 354 and validates it. 9. System asks for Number of servers and user enters a valid digit between 2–4 or 4-7 depending on how many teams there are.

	10. System asks for Number of teams and user enters a valid digit between 16 and 64 (inclusive). 11. System asks the user to add each team to the tournament, one by one, until the chosen number of teams has been added. • For each team, user enters a team identifier that exists and is not already in the tournament list. 12. System sends all tournament information to the logic layer: • id, name, venue, start_date, end_date, contract, contact_email, contact_number, num_of_servers, teams_in_tournament. 13. System creates the tournament and confirms success.
Alternative flow	**A1: User selects "Create new tournament"** • System asks for one of the required fields (Name / Tournament id / Start date / End date / Venue / Contract / Contact person's email / Contact person's phone number / Number of servers / Number of teams / Team to tournament). • User leaves the input empty and presses Enter. • System raises EmptyInput and prints a corresponding error message (e.g. "Tournament needs a name", "Tournament needs a contact persons email"). • System asks the user to enter the same field again **A2: User selects "Create new tournament"** • System asks the user input a "Tournament Id" • User inputs an Id that is already in use • System raises IdAlreadyExists and prints a corresponding error message (e.g. "Id already exists") **A3: User selects "Create new tournament"** • System asks for Start date and End date in the format day-month-year.

	• User enters dates that are invalid in one of the following ways: – Start date is in the past. – End date is before start date. – Tournament duration is not between 2–7 days. – Dates are not in the correct format (missing "-" or wrong order). – One of the dates does not exist. • System raises one of: InvalidStartDateInPast, InvalidStartDateBefore, InvalidAmountOfDays, InvalidFormat, or DateDoesNotExistError and prints the corresponding error message. • System asks the user to enter start date and end date again. **A4: User selects "Create new tournament"** • System asks for Contact person's email. • User enters an email that is not in a valid format. • System raises InvalidContactEmail and prints "Contact persons email is invalid, try again". • System asks the user to enter the contact person's email again. **A5: User selects "Create new tournament"** • System asks for Contact person's phone number (+354) • User enters a value that is not a valid phone number • System raises InvalidContactNumber and prints "Contact persons phone number is invalid, try again". • System asks the user to enter the contact person's phone number again. **A6: User selects "Create new tournament"** • System asks for Number of servers.

- User enters a value that is not a digit or not in the allowed range.
- System raises ValueError when the input is not a digit, or InvalidServers when the number is not between 2–4 with less than 32 teams or number is not between 4-7 with tournament with 32 or more teams, and prints the corresponding error message.
- System asks the user to enter the number of servers again.

A7: User selects “Create new tournament”

- System asks for Number of teams.
- User enters a value that is not a digit, or a number less than 16 or greater than 64.
- System raises ValueError when the input is not a digit, or WrongNumOfTeams when the number is outside the allowed range and prints the corresponding error message.
- System asks the user to enter the number of teams again.

A8: User selects “Create new tournament”

- System asks the user to add teams to the tournament one by one.
- For a given team, user enters a team identifier that is invalid in one of the following ways:
  – Team is already in the tournament list.
  – Team does not exist in the system.
- System raises TeamAlreadyInTournament or TeamDoesNotExist and prints the corresponding error message.
- System asks the user to enter a different team for that position in the tournament.

Post condition	Tournament data is sent down through logic layer and then added to the database
Actors	Organizer
Authors	Alexander Björnsson

Use Case 5 --- “Organizer handles match results”

Use case name	As an organizer I want to be able to handle the match results in the tournament
Number	UC – 05
Priority	A
Source	A10, A11, A12, B4
Precondition	User picks organizer role User picks tournament menu A tournament that the user can pick exists
Base flow	1. User selects **“Input match scores”**. 2. System asks: Enter Tournament ID for inputin match results (q/Q to quit):. 3. User enters the ID of an existing tournament. 4. System validates that the tournament ID exists. 5. System retrieves the tournament. 6. System retrieves all **active matches** for that tournament. 7. For each active match: • System prints the match number and teams (team A vs team B). • System asks for **team A’s score**. • User inputs team A’s score. • System asks for **team B’s score**. • User inputs team B’s score. • System updates the game results with the given scores. 8. When all active matches have been updated, the system finishes the process and control returns to the tournament flow (e.g. tournament menu).

Alternative flow	**A1: User selects “Input match scores”** • System asks for Tournament ID for inputting match results (q/Q to quit). • User enters q or Q. • System returns to the tournament menu without asking for or updating any match scores. **A2: User selects “Input match scores** • System asks for Tournament ID for inputting match results (q/Q to quit). • User enters a tournament ID that does not exist in the system. • System raises TournamentNotExistError and prints “Tournament does not exist”. • System restarts the Input match scores process and asks the user again for a Tournament ID. **A3: User selects “Input match scores** • System has already validated a tournament ID and retrieved all active matches. • System prints a match and asks for **team A’s score** (Q/q to quit). • User enters q or Q instead of a score. • System returns to the tournament menu without updating further match scores. **A4: User selects “Input match scores** • System prints a match and the user has already entered **team A’s score**. • System asks for **team B’s score** (Q/q to quit). • User enters q or Q instead of a score.

	• System returns to the tournament menu without updating further match scores.
Post condition	A full schedule exists, and all completed matches have correct result stored. The backed or schedule is updated after each match result
Actors	Organizer, System (automatically generating and updating schedule)
Authors	Nökkvi Benediktsson

Use case 6 ---" View Game Schedule"

Use case name	As a user I want to be able to view the game schedule
number	UC – 06
Priority Source	A A13
Precondition	1. Tournament and match data exist in the 2. system stored match result where applicable 3. User picks one of the 3 available roles 4. User picks tournament menu
Base flow	1. User selects **"See tournament info"**. 2. System asks: Enter Tournament ID for information (q/Q to quit): 3. User enters the ID of an existing tournament. 4. System validates that the tournament ID exists. 5. System retrieves the tournament with that ID. 6. System prints a formatted **Tournament info** section showing: • ID • Name • Venue • Start date • End date • Contract • Contact email • Contact phone number • State • Number of servers 7. System prints a formatted **Matches in tournament** section.

	8. System retrieves all matches and prints each match whose tournament_id matches the entered ID. 9. System asks the user: Press Q/q to quit. 10. User enters q or Q. 11. System returns to the tournament menu.
Alternative flow	**A1: User selects "See tournament info"** • System asks for Tournament ID for information (q/Q to quit). • User enters q or Q. • System returns to the tournament menu without showing any tournament information **A2: User selects "See tournament info"** • System asks for Tournament ID for information (q/Q to quit). • User enters a tournament ID that does not exist in the system. • System raises TournamentNotExistError and prints "Tournament does not exist". • System restarts the **See tournament info** process and asks the user again for a Tournament ID for information. **A3: User selects "See tournament info"** • System has already displayed the tournament info and the matches in the tournament. • System asks the user: Press Q/q to quit. • User enters something other than q or Q. • System keeps asking the user to Press Q/q to quit until the user enters q or Q. • When the user enters q or Q, the system returns to the tournament menu. **A4: User selects "See tournament info"** • System has validated the tournament ID and retrieved the tournament.

	• System retrieves all matches. • None of the matches have a tournament_id equal to the entered ID. • System prints the **Matches in tournament** header but no matches under it. • System then asks the user to Press Q/q to quit and returns to the tournament menu when the user does so.
Postcondition	User sees a correct and up to date schedule Completed games show results Upcoming games show participating teams and date & time
Actors	All users
Authors	Alexander Björnsson

Use Case 7 --- "View a list of all the teams"

Use Case Name	As a user I want to be able to see a list of all the teams
Number	UC – 07
Priority	A
Source	A14, B6
Precondition	Teams and tournaments are registered in the system User selects one of the three roles available User selects "Team menu"
Basic Flow	1. User selects **"See all teams"**. 2. System retrieves all teams from the logic layer. 3. System prints a header row with columns: Name, Club, Tournaments Played, Wins. 4. System prints each team on its own line with name, club, tournaments played, and wins. 5. System asks the user: Press q/q to quit. 6. User enters q or Q. 7. System returns to the team menu.
Alternative Flow	1. **A1: User selects "See all teams"** • System prints all teams in a table format. • System asks the user: Press q/q to quit. • User enters something other than q or Q. • System keeps asking the user to Press q/q to quit until the user enters q or Q. • When the user finally enters q or Q, the system returns to the team menu
Post condition	User sees the list of tournaments for a chosen team
Actors	All users
Authors	Alexander Björnsson

Use Case 8 --- “Anyone can view team info”

Use case name	As a user I want to be able to search and view a team’s information
Number	UC – 8
priority	A
source	A15, A16, A17, B6
Precondition	Teams and players are registered in the system
Basic flow	1. User selects **“See team info”**. 2. System asks: Enter team name for information (q/Q to quit): 3. User enters the name of an existing team. 4. System checks if the team exists and validates the team name successfully. 5. System retrieves the players in the team. 6. System retrieves the team information (name, captain, club, web link, ASCII logo, tournaments played, tournaments won, runner-up count). 7. System prints a formatted Team inf**o** section with all team details. 8. System prints a formatted Players in team section and lists all players in the team. 9. System returns to the team menu.
Alternative flow	A1: User selects “See team info” • System asks: Enter team name for information (q/Q to quit): • User inputs q/Q • Returns to the team menu A2: User selects “See team info” • System asks: Enter team name for information (q/Q to quit): • User inputs a name which system tries to validate

	• System raises TeamDoesNotExist which displays a corresponding error message
Postcondition	User has retrieved and viewed team information
Actor	All users
Author	Alexander Björnsson

Use case 9 --- “View player private data”

Use case name	As a captain or organizer I want to be able to view a players personal data
number	UC – 9
priority	A
Source	A18, B5
Precondition	User picked Organizer or Captain role Players exist in the system with stored personal data User has already picked “player settings” from Organizer/Captain menu
Basic flow	1. User selects “View player info 2. System asks for a handle of the player that the user wants to see the info of 3. User inputs said handle 4. System validates if the handle provided is a player in the system 5. System displays the player’s private data (e.g.name, address, phone, etc.) 6. Actor returns to the previous menu
Alternative flow	A1: User selects “View player info” • User inputs a players handle • System validates the handle but raises an exception that the handle of the player is not in the database • PlayerNotExist is raised and a corresponding error message is displayed
postcondition	User sees all the info of the player and returns to the previous menu
Actor	Captain and Organizer
Author	Alexander Björnsson

Use case 10 --- “View player list”

Use case name	As a user I want to be able to view all the players
Number	UC – 10
Priority	A
Source	A13, A18, B5
Precondition	Players exist in the system
Basic flow	1. User selects “View Players” from the menu 2. The system displays a list of all players and only handles and team. 3. User scrolls or searches through the list. 4. Users select a player to view more information. 5. User returns to the main menu.
Alternative flow	A1: User selects “Player menu” • System shows Player menu and q/Q to quit • User inputs q/Q which returns back to the menu
Post condition	User has seen a list of players.
Actors	All users
Authors	Nökkvi Benediktsson

Use case 11 --- “Support clubs (teams belong to clubs)”

Use case name	As a user I want to be able to create a club
Number	UC – 11
Priority	B
Source	B3
Precondition	1. User has the **Organizer** role 2. User is currently in the **Club Menu**. 3. Teams to be assigned to the club must already exist in the system (but not belong to another club).
Basic flow	1. User selects "Create a club" 2. 2. System displays initialization message and available colors. 3. System requests a **Club Name**. 4. User inputs a unique name. 5. System requests a **Color** from the displayed list. 6. User inputs a valid color. 7. System requests a **Hometown**. 8. User inputs a hometown. 9. System requests a **Country**. 10. User inputs a country. 11. System requests the **Number of teams** (1-5). 12. User inputs a valid integer. 13. **Loop:** System asks for a specific Team Name (repeats for the number of teams defined in step 12). 14. User inputs a valid, available Team Name. 15. System validates all data and saves the new Club. 16. User returns to the Club Menu.
Alternative flow	**A1: General Back/Quit**

	• At any input prompt, User inputs the "q" or "Q" key. • BackButton exception is raised. • System returns User to the Club Menu. **A2: Invalid Name** • User inputs an empty name or an existing club name. • ClubNameExistsError or EmptyInput is raised. • System displays error ("Name already exists" or "Club needs a name") and repeats Step 3. **A3: Invalid Color** • User inputs a color not in the list. • ColorNotAvailable is raised. • System displays "Color not available" and repeats Step 5. **A4: Invalid Number of Teams** • User inputs non-digit or number outside 1-5 range. • ValueError or InvalidNumOfTeams is raised. • System displays error and repeats Step 11. **A5: Invalid Team Assignment** • User inputs a team that doesn't exist, is already in the club, or belongs to another club. • TeamDoesNotExistError, TeamAlreadyInClubError, or TeamNotAvailableError is raised. • System displays specific error message and repeats Step 13 for that specific team entry.
Post condition	A new Club is created in the system with the defined name, color, location, and associated teams.
Actors	Organizer
Authors	Ernir Elí Ellertsson

Use case 12 --- “View clubs “

Use case name	As a user I want to see all clubs
Number	UC – 12
Priority	B
Source	B7
Precondition	1. User is currently in the **Club Menu**. 2. Clubs exist in the system.
Basic flow	1. User selects "See all clubs" (Option 2) from the Club Menu. 2. System retrieves the list of all clubs and their data (club.name, club.country, club.tournaments, club.wins). 3. System displays a formatted table with the headings: **Name**, **Country**, **Tournaments Played**, and **Wins**. 4. System lists all existing clubs, displaying the relevant data under the corresponding headings. 5. System prompts the user to "Press q/Q to quit." 6. User inputs 'q' or 'Q'. 7. System returns the user to the previous menu (Team Menu, as per your code).
Alternative flow	1. **A1: No Clubs Exist** • System retrieves the list of clubs, but the list is empty. • System displays a message indicating that "No clubs are currently in the system. • System continues to Step 5 (prompts user to quit)
postcondition	User sees all the clubs in a list with listed info
Actors	User, Captain and Organizer
Author	Alexander Björnsson

User case 13 --- “See club info”

Use case name	Player statistics
Number	UC – 13
Priority	B
Source	B8
Precondition	1. User is currently in the **Club Menu**. 2. The Club to be viewed must exist in the system.
Basic flow	1. User selects "See club info" (Option 3). 2. **System prompts** the user to enter a Club Name (or 'q' to quit). 3. User inputs a valid Club Name. 4. System validates that the Club exists. 5. System retrieves club details and the list of teams assigned to that club. 6. **System displays** the Club's specific data: • Name, Color, Hometown, Country • Tournaments played • Total tournament second places (Wins/Runner-ups) 7. **System displays** the list of names of all Teams in the club. 8. System prompts user to press 'q' to stop viewing. 9. User inputs 'q'. 10. System returns the user to **Step 2** (Prompt to enter a club name).
Alternative flow	**A1: User quits from Search Prompt** • At Step 2, User inputs "q" or "Q". • System returns User to the **Club Menu**. **A2: Club Does Not Exist** • User inputs a name that is not in the database. • ClubDoesNotExist exception is raised. System displays error message "Club does not exist".

	• System recurses and returns to **Step 2** (Prompts for name again).
Post condition	User has viewed the detailed statistics and team roster for a specific club and is ready to search for another club or return to the menu.
Actors	Users, Organizer, Captain
Authors	Alexander Björnsson

Use case 14 ---" Add Team to Club"

Use case name	As an organizer or captain I want to be able to add a team to a club
Number	UC – 14
Priority	B
Source	B9
Precondition	1. User has the **Organizer** role. 2. User is currently in the **Club Menu**. 3. Both the Club and the Team must already exist in the system.
Basic Flow	1. User selects "Add team to a club" (Option 4). 2. **System prompts** for a Club Name. 3. User inputs a valid Club Name. 4. System validates the club exists and checks if it has reached its maximum capacity (5 teams). 5. **System prompts** for the Team Name to be added. 6. User inputs a valid Team Name that is not currently assigned to any club. 7. System validates the team and links it to the selected club. 8. System displays "Team added to club". 9. System returns the user to the **Team Menu**.
Alternative flows	**A1: User Quits** • At the club name prompt, User inputs "q" or "Q". • System returns User to the **Club Menu**. **A2: Club Does Not Exist** • User inputs a name not in the database. • ClubDoesNotExist is raised; system displays error and repeats Step 2 **A3: Club is Full**

	• System detects the club already has the maximum number of teams. • System displays "Club is full" and restarts the use case at Step 1. **A4: Team Error/Back** • User inputs a name for a team that doesn't exist (TeamDoesNotExistError) or is already assigned (TeamNotAvailableError). • System displays specific error and repeats Step 5. • If BackButton exception occurs, system returns User to the **Team Menu**.
Post condition	Team’s club status is changed to whatever club the team was added to and user goes back to team menu
Actor	Organizer
Author	Alexander Björnsson

Use case 15 --- “Remove team from club”

Use case name	As an organizer I want to be able to remove a team from a club
Number	UC – 15
Priority	B
Source	B10
Precondition	1. User has the **Organizer** role. 2. User is currently in the **Club Menu**. 3. The Club must contain at least two teams (as per code logic).
Basic flow	1. User selects "Remove team from a club" (Option 5). 2. **System prompts** for the Club Name. 3. User inputs a valid Club Name. 4. System validates that the club exists and contains 2 or more teams. 5. **System prompts** for the Team Name to be removed. 6. User inputs a valid Team Name that belongs to that specific club. 7. System removes the team from the club record. 8. System displays "Player has been removed from the club". 9. System returns the user to the **Team Menu**
Alternative flow	**A1: User Quits** • At either prompt (Step 2 or 5), User inputs "q" or "Q". • System returns User to the **Club Menu**. **A2: Club Does Not Exist** • User inputs a name not in the database. • System displays "Club does not exist" and repeats Step 2. **A3: Club Has Too Few Teams** • System detects the club has fewer than 2 teams.

	• System displays "The club does not have enough teams to remove one". • System redirects user to the **Add Team to Club** function. **A4: Team Validation Errors** • User inputs a team that doesn't exist (TeamDoesNotExistError). • System displays error and repeats Step 5. **A5: Team Not in Selected Club** • User inputs a valid team that is not part of the specified club. • System displays "Team is not in the Club". • System restarts the removal process at Step 1
Post condition	The specified team is dissociated from the club, and the system state is updated.
Actors	Organizer
Authors	Alexander Björnsson

Use case 16 --- “Add player to a team”

Use case name	As a user I want to add a player to a team
Number	UC – 16
Priority	B
Source	B12
Precondition	1. User has the **Captain** or **Organizer** role. 2. User is in the **Team Menu**. 3. The Team and the Player must already exist in the system.
Basic flow	1. User selects "Add player to a team". 2. **System prompts** for the Team Name. 3. User inputs a valid Team Name. 4. System validates that the team exists. 5. **System prompts** for the Player’s handle or ID. 6. User inputs a valid Player handle. 7. System validates the player is not already assigned to a team (or is available). 8. System links the Player to the Team. 9. System displays a success message. 10. System returns the user to the **Team Menu**.
Alternative flows	**A1: User Quits** • At the team name prompt, User inputs "q" or "Q". • System returns User to the **Team Menu**. **A2: Team Does Not Exist** • User inputs a team name that is not in the system. • TeamExistsError (or similar) is raised. • System displays "Team does not exist" and repeats Step 2. **A3: Player Validation Errors**

	• User inputs a player handle that doesn't exist or is already on a team. • System displays corresponding error and repeats Step 5.
Post conditions	The player is successfully added to the team roster, and the system records are updated.
Actor	Organizer
Author	Alexander Björnsson

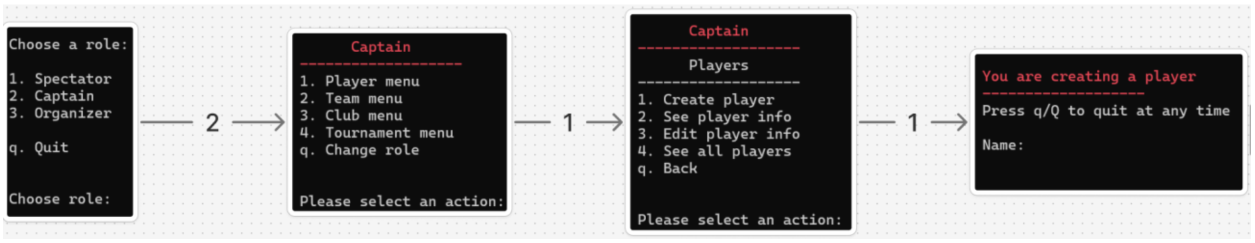
Use case 17 --- “Remove a player from team”

Use case name	As a user I want to be able to add players to a team
Number	UC – 17
Priority	B
Source	B11
Precondition	1. User has the **Captain** or **Organizer** role. 2. User is currently in the **Team Menu**. 3. The Team must exist and have at least 4 players.
Basic flow	1. User selects "Remove player from a team". 2. **System prompts** for the Team Name. 3. User inputs a valid Team Name. 4. System validates the team exists and checks if it has 4 or more players. 5. **System prompts** for the Player's handle to be removed. 6. User inputs a valid Player handle. 7. System validates that the player exists, is not the team captain, and is currently a member of that team. 8. System removes the player from the team roster. 9. System displays "Player has been removed from the team". 10. System returns the user to the **Team Menu**.
Alternative flow	**A1: User Quits** • At Step 2 or 5, User inputs "q" or "Q". • System returns User to the **Team Menu**. **A2: Team Does Not Exist** • User inputs a name not in the database. • TeamExistsError is raised; system displays error and repeats Step 2. **A3: Team Size Constraint**

	<ul><li>System detects the team has fewer than 4 players.</li><li>System displays "There are too few players in the team to remove from it".</li><li>System restarts the function from Step 1.</li></ul>**A4: Cannot Remove Captain**<ul><li>User selects the handle of the team captain.</li><li>CantRemoveCaptainError is raised.</li><li>System displays "Captain can not be removed" and repeats Step 5.</li></ul>**A5: Player Not in Team**<ul><li>User inputs a valid handle, but the player is not registered to the selected team.</li><li>System displays "player is not in the team" and repeats Step 5</li></ul>
Post condition	The player is dissociated from the team, and the team size is reduced by one.
Actors	Organizer
Authors	Ernir Elí Ellertsson

Happy Paths for Use Cases

UC – 1, UC – 3, UC – 4



```
Choose a role:

1. Spectator
2. Captain
3. Organizer

q. Quit

Choose role:
```

```
       Captain
-------------------
1. Player menu
2. Team menu
3. Club menu
4. Tournament menu
q. Change role

Please select an action:
```

```
       Captain
-------------------
       Players
-------------------
1. Create player
2. See player info
3. Edit player info
4. See all players
q. Back

Please select an action:
```

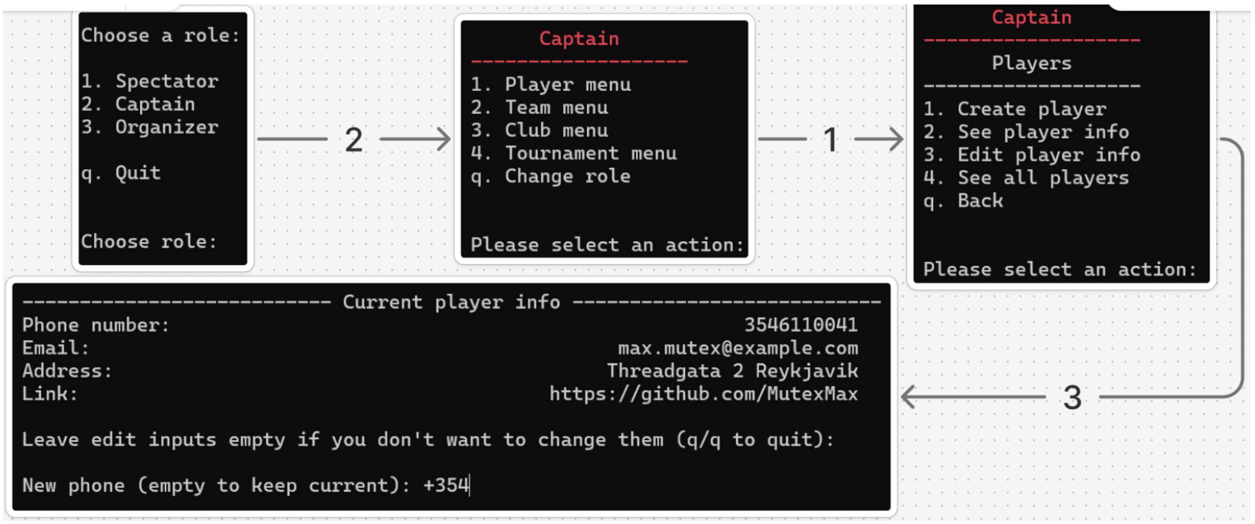
```
You are creating a player
-------------------
Press q/Q to quit at any time

Name:
```

Happy path UC – 1, store player personal data

Happy path UC – 3, create new team is a similar flow, unless you go through team menu

Happy path UC – 4, create tournament is a similar flow, unless you go through tournament menu

UC – 2



```
Choose a role:

1. Spectator
2. Captain
3. Organizer

q. Quit

Choose role:
```

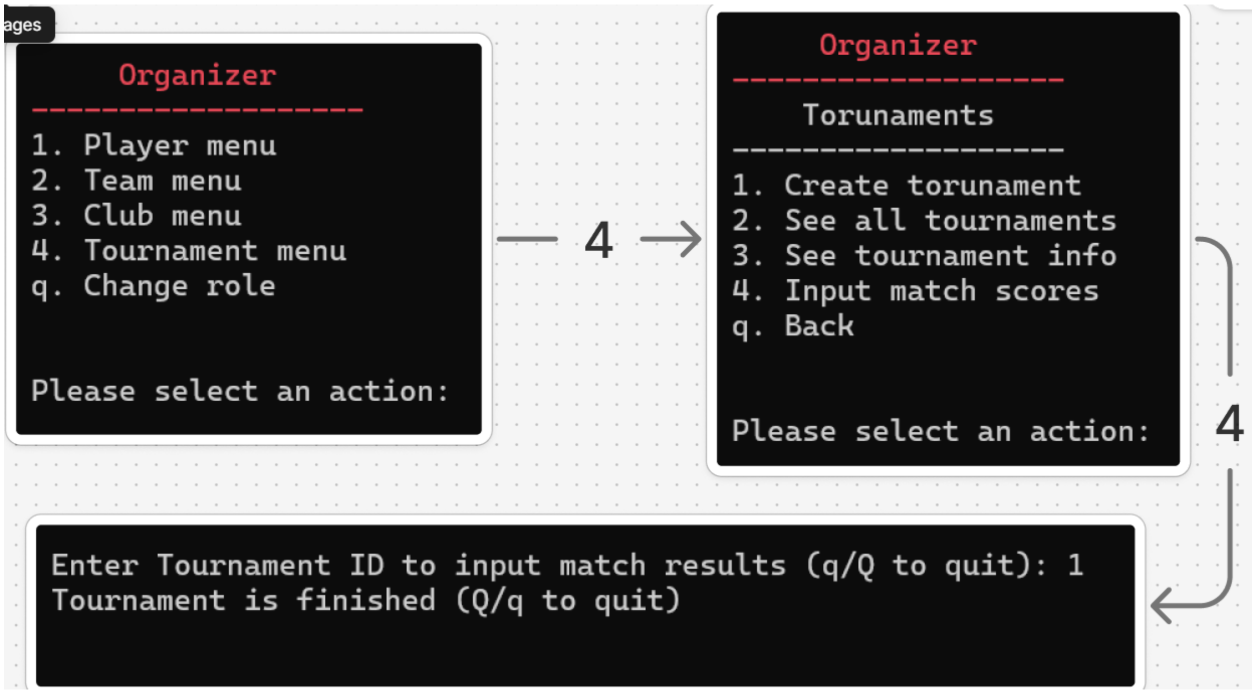
```
       Captain
-------------------
1. Player menu
2. Team menu
3. Club menu
4. Tournament menu
q. Change role

Please select an action:
```

```
       Captain
-------------------
       Players
-------------------
1. Create player
2. See player info
3. Edit player info
4. See all players
q. Back

Please select an action:
```

```
--------------------------- Current player info ----------------------------
Phone number:                                                    3546110041
Email:                                                 max.mutex@example.com
Address:                                              Threadgata 2 Reykjavik
Link:                                             https://github.com/MutexMax

Leave edit inputs empty if you don't want to change them (q/q to quit):

New phone (empty to keep current): +354
```

Happy path UC – 2, modify player info

UC - 5



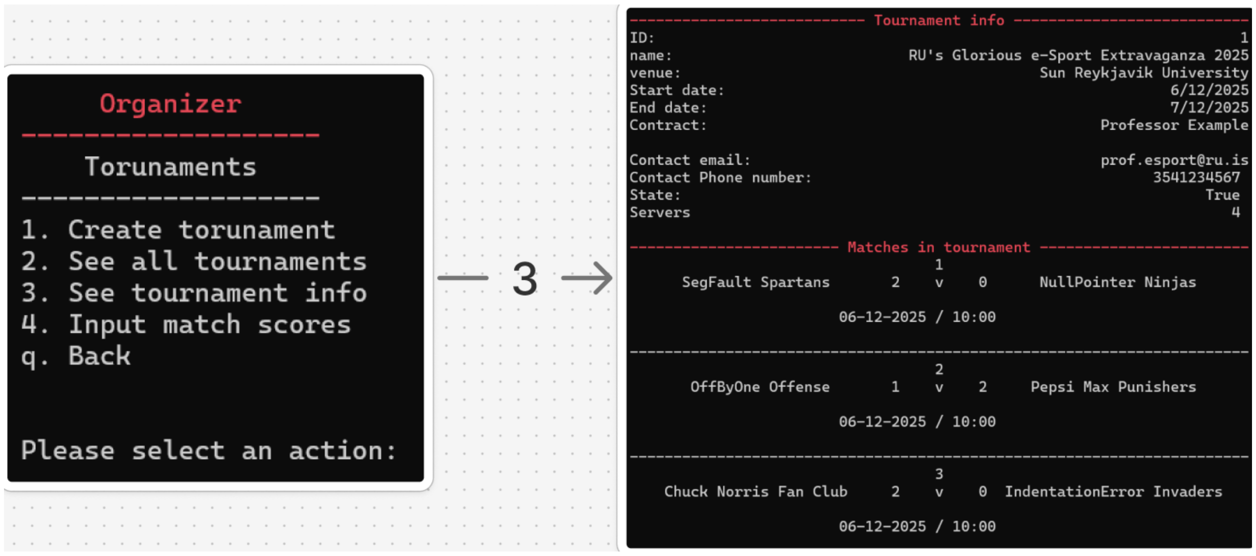
```
      Organizer
--------------------
1. Player menu
2. Team menu
3. Club menu
4. Tournament menu
q. Change role


Please select an action:
```

```
      Organizer
--------------------
    Torunaments
--------------------
1. Create torunament
2. See all tournaments
3. See tournament info
4. Input match scores
q. Back


Please select an action:
```

```
Enter Tournament ID to input match results (q/Q to quit): 1
Tournament is finished (Q/q to quit)
```

Happy path UC – 5, organizer handles match results

UC – 6

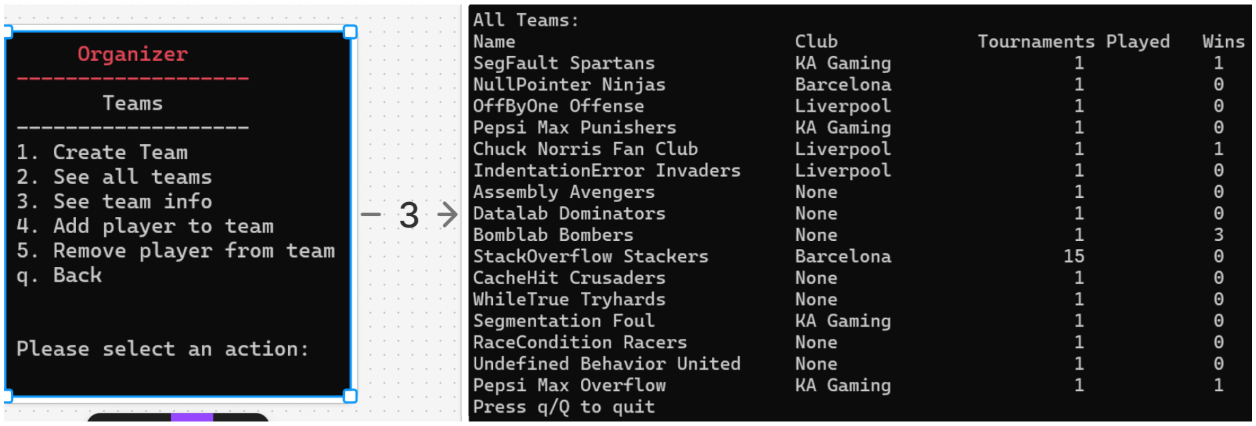


```
    Organizer
--------------------
    Torunaments
--------------------
1. Create torunament
2. See all tournaments
3. See tournament info
4. Input match scores
q. Back


Please select an action:
```

```
--------------------------- Tournament info ---------------------------
ID:                                                                   1
name:                              RU's Glorious e-Sport Extravaganza 2025
venue:                                          Sun Reykjavik University
Start date:                                                    6/12/2025
End date:                                                      7/12/2025
Contract:                                              Professor Example

Contact email:                                         prof.esport@ru.is
Contact Phone number:                                         3541234567
State:                                                              True
Servers                                                                4

------------------------ Matches in tournament ------------------------
                                    1
      SegFault Spartans       2     v     0      NullPointer Ninjas

                        06-12-2025 / 10:00

-----------------------------------------------------------------------
                                    2
       OffByOne Offense       1     v     2     Pepsi Max Punishers

                        06-12-2025 / 10:00

-----------------------------------------------------------------------
                                    3
    Chuck Norris Fan Club     2     v     0  IndentationError Invaders

                        06-12-2025 / 10:00
```

Happy path, UC – 6, Anyone can view game schedule

UC – 7, UC – 10, UC - 12



```
     Organizer
--------------------
       Teams
--------------------
1. Create Team
2. See all teams
3. See team info
4. Add player to team
5. Remove player from team
q. Back


Please select an action:
```

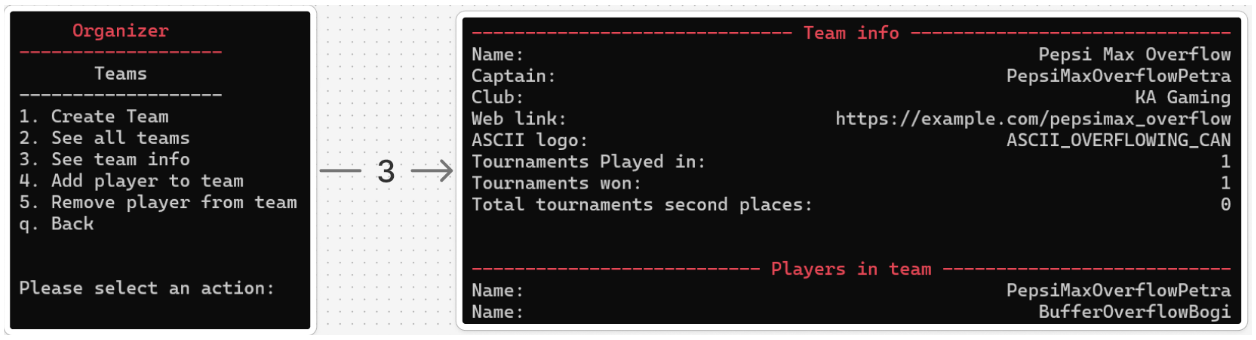
```
All Teams:
Name                          Club                Tournaments Played   Wins
SegFault Spartans             KA Gaming                          1        1
NullPointer Ninjas            Barcelona                          1        0
OffByOne Offense              Liverpool                          1        0
Pepsi Max Punishers           KA Gaming                          1        0
Chuck Norris Fan Club         Liverpool                          1        1
IndentationError Invaders     Liverpool                          1        0
Assembly Avengers             None                               1        0
Datalab Dominators            None                               1        0
Bomblab Bombers               None                               1        3
StackOverflow Stackers        Barcelona                         15        0
CacheHit Crusaders            None                               1        0
WhileTrue Tryhards            None                               1        0
Segmentation Foul             KA Gaming                          1        0
RaceCondition Racers          None                               1        0
Undefined Behavior United     None                               1        0
Pepsi Max Overflow            KA Gaming                          1        1
Press q/Q to quit
```

Happy path, UC – 7, anyone can view a list of all teams

Happy path, UC – 8, Anyone can view player list is similar flow, go through player menu

Happy path, UC – 12, Anyone can view clubs is a similar flow, go though club menu

UC – 8, UC – 9, UC – 13



```
     Organizer
-------------------
       Teams
-------------------
1. Create Team
2. See all teams
3. See team info
4. Add player to team
5. Remove player from team
q. Back


Please select an action:
```

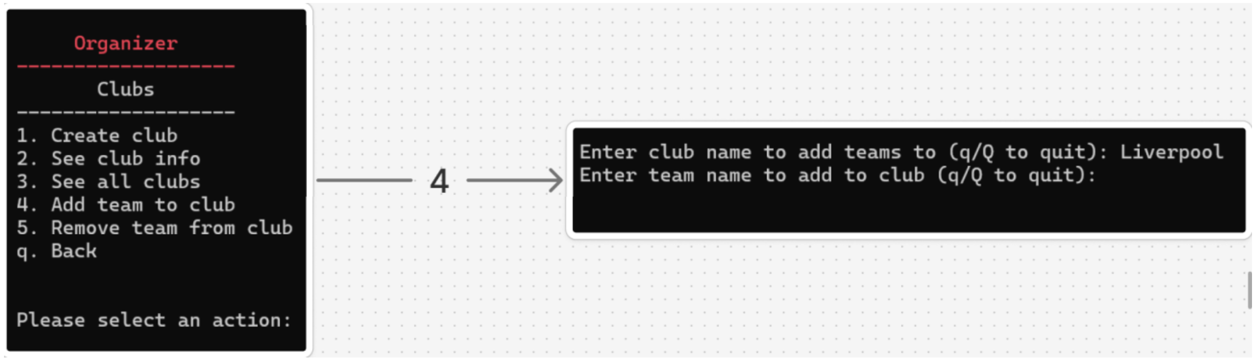
```
------------------------------ Team info ------------------------------
Name:                                                   Pepsi Max Overflow
Captain:                                              PepsiMaxOverflowPetra
Club:                                                            KA Gaming
Web link:                            https://example.com/pepsimax_overflow
ASCII logo:                                          ASCII_OVERFLOWING_CAN
Tournaments Played in:                                                   1
Tournaments won:                                                         1
Total tournaments second places:                                         0


--------------------------- Players in team ---------------------------
Name:                                                 PepsiMaxOverflowPetra
Name:                                                  BufferOverflowBogi
```

Happy path, UC – 8, Anyone can view team info

Happy path, UC – 9, View player private data, is a similar flow, just go through player menu

Happy path, UC – 13, See club info, is similar flow, just go through club info

UC – 14, UC – 15, UC – 16, UC - 17



```
     Organizer
--------------------
       Clubs
--------------------
1. Create club
2. See club info
3. See all clubs
4. Add team to club
5. Remove team from club
q. Back


Please select an action:
```

```
Enter club name to add teams to (q/Q to quit): Liverpool
Enter team name to add to club (q/Q to quit):
```

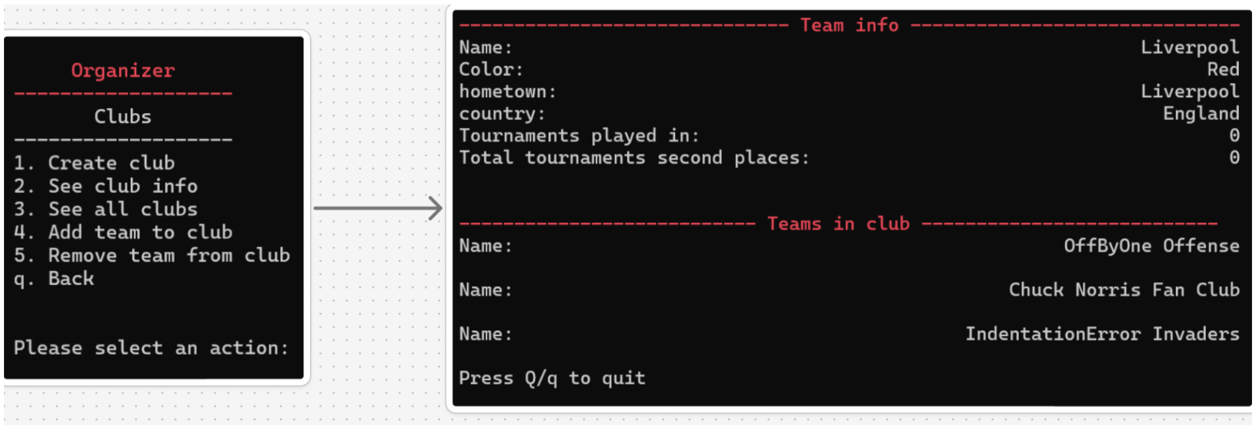
Happy path, UC – 14, Add team to club

Happy path, UC – 15, Remove team from club has a similar flow, just choose remove club

Happy path, UC – 16, Add player to team has a similar flow, just go through player menu

Happy path, UC – 17, Remove player from team, just go through player menu and remove

UC – 11



```
     Organizer
-------------------
       Clubs
-------------------
1. Create club
2. See club info
3. See all clubs
4. Add team to club
5. Remove team from club
q. Back


Please select an action:
```

```
------------------------------ Team info ------------------------------
Name:                                                         Liverpool
Color:                                                              Red
hometown:                                                     Liverpool
country:                                                        England
Tournaments played in:                                                0
Total tournaments second places:                                      0


--------------------------- Teams in club ---------------------------
Name:                                                   OffByOne Offense

Name:                                              Chuck Norris Fan Club

Name:                                          IndentationError Invaders

Press Q/q to quit
```

Happy path, UC – 11, Support club, player belongs to a club

State Diagram

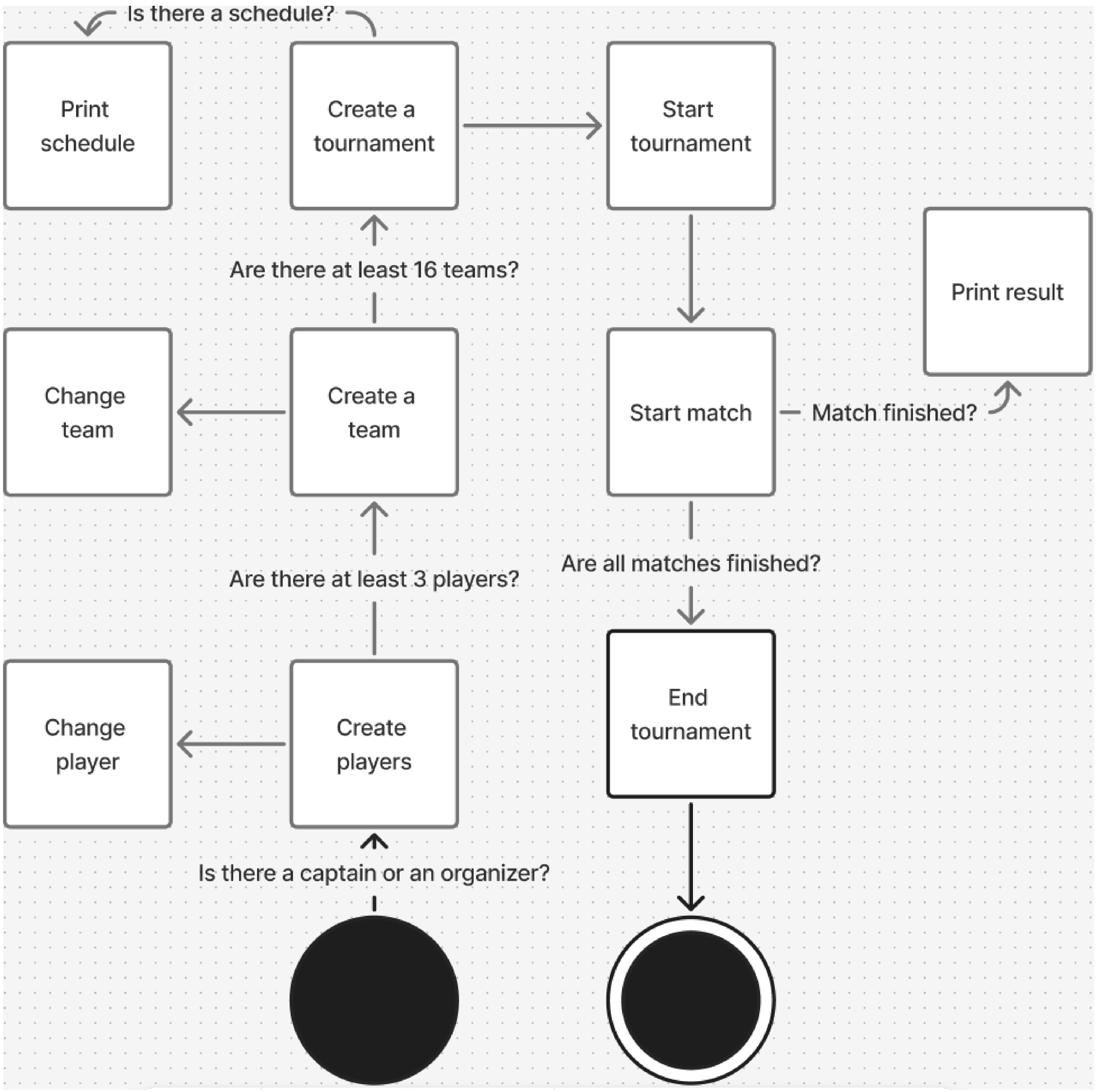
Our state diagram illustrates the flow of actions and decisions within a process. It visually represents the conditions that must be met for transitions between states to occur.

It begins in the initial state, represented by a solid black circle and progresses through various states based on specific conditions.

The transition is represented by an arrow. Each arrow may include a question. for that transition to happen, the requirements need to be fulfilled by returning true from the question on the arrow.

The squares represent states in the process. These are the states a user or system can take depending on their role and current context.

The last state is a final state and is shown as a black circle inside another circle. The final state indicates the end of the process, where no further transitions are possible.


Is there a schedule?
Print schedule
Create a tournament
Start tournament
Are there at least 16 teams?
Print result
Change team
Create a team
Start match
Match finished?
Are there at least 3 players?
Are all matches finished?
Change player
Create players
End tournament
Is there a captain or an organizer?

User Case Analysis

A user group analysis like the one below is basically a quick, structured snapshot of who you're designing for and the context they work in. By writing down things like who they are (age, education, skills), what they're trying to do (goals), what tools they use (laptops/software), where and when they use the system (environment + frequency), and any constraints (time pressure, accessibility, mixed skill levels), you reduce guesswork.

Why we use it:

- Make better design decisions (features, layout, wording, workflow) that fit real situations
- Prioritize requirements based on what matters most to that group
- Spot risks early (e.g., low time per task, inconsistent usage, different skill levels)
- Communicate clearly with your team/stakeholders using one shared understanding
- Plan testing (who to recruit, what tasks to test, what success looks like)

1) Organizers / Admins

Main section	Details
NAME of group	Organizers / Admins
WHO – Background	Age: ~18–45. Gender: all genders. Education: high school or higher, often university students or staff running tournaments. Abilities/Disabilities: not known, design for mixed abilities. Computer skills: good–very good. Number: very few – usually 1 main admin
WHY – Main goals	Set up and manage tournaments, teams and players. Generate schedules/brackets. Enter and update match results. Optionally manage clubs, points and statistics.
WHAT – Equipment	Mostly laptops or desktop PCs, sometimes an external monitor. Stable internet connection. Mouse and keyboard.
WHERE – Environment	At home when planning. At the tournament venue / LAN room while the event is running. Sometimes at school or university.
WHEN – Use of system	How often: a lot in the days before a tournament and very frequently during the event. For how long each time: sessions of about 30 minutes to several hours when a tournament is running.
HOW – Skills	Comfortable with forms, tables and filters. Understands tournament formats (knock-out, double elimination). Can handle simple data validation and corrections.
HOW MUCH – Importance	Critical – without organizers the system cannot create or update tournaments.

2) Team Captains

Main section	**Details**
NAME of group	Team Captains
WHO – Background	Age: ~18–35. Gender: all genders. Education: mostly secondary school or university students. Abilities/Disabilities: not known, support mixed abilities. Computer skills: good; used to games, Discord, Steam, etc. Number: one per team, from a handful to dozens depending on tournament size.
WHY – Main goals	Add and maintain player information for their team. Keep contact information up to date. Check schedules and tournaments for their team.
WHAT – Equipment	Personal gaming PCs or laptops. Sometimes phones or tablets for quick checks. Home or venue Wi-Fi.
WHERE – Environment	At home. At the tournament venue between matches. Sometimes at school or university.
WHEN – Use of system	How often: a few times when setting up the team and players, then before and during tournaments. For how long each time: short sessions, usually 5–20 minutes.
HOW – Skills	Comfortable with menus, forms and tables. Can understand simple error messages (missing info, duplicates). Beginner–moderate experience with "admin" tasks.
HOW MUCH – Importance	Very important – they keep their team's data correct and are the main contact with organizers.

3) Players

Main section	Details
NAME of group	Players (non-captain team members)
WHO – Background	Age: ~18–35. Gender: all genders. Education: mostly students, some older players. Abilities/Disabilities: not known, design for a range (e.g. colour-blind friendly). Computer skills: very good with computers and games, but may dislike complex admin UIs. Number: many – several players per team, can be tens or hundreds overall.
WHY – Main goals	See when and against whom they play. Check match results. View their own and their team's tournament history and basic statistics.
WHAT – Equipment	Gaming PCs or laptops. Very often smartphones for quick checks.
WHERE – Environment	At home before/after matches. At the venue. On the move using a phone.
WHEN – Use of system	How often: mainly around tournament days. For how long each time: very short visits (about 1–5 minutes) to check next match, results or history.
HOW – Skills	Good computer skills. Expect fast access with few clicks. Prefer simple, clear tables for times, teams and results.
HOW MUCH – Importance	Important but secondary – they don't manage data, but their experience strongly affects how good the system feels.

4) General Users / Spectators

Main section	**Details**
NAME of group	General Users / Spectators (friends, family, fans, journalists, teachers, etc.)
WHO – Background	Age: wide range, ~15–50+. Gender: all genders. Education: mixed – from pupils to staff/parents. Abilities/Disabilities: mixed and mostly unknown, so UI must be very clear and accessible. Computer skills: beginner to moderate. Number: potentially the largest group – anyone interested in the tournament.
WHY – Main goals	Check game schedules. See who won. Look up teams and players and which tournaments they have played in.
WHAT – Equipment	Mostly smartphones. Some laptops/desktops at school or work. Venue Wi-Fi or mobile data.
WHERE – Environment	At home. At the venue in the audience. On the go.
WHEN – Use of system	How often: mostly on match days or right after matches. For how long each time: very short sessions (under 5 minutes).
HOW – Skills	Need simple navigation and wording. Should not have to log in or understand tournament jargon. Basic click/scroll skills only.
HOW MUCH – Importance	Important for usability and visibility – they don’t run tournaments but are key for making information public and understandable.

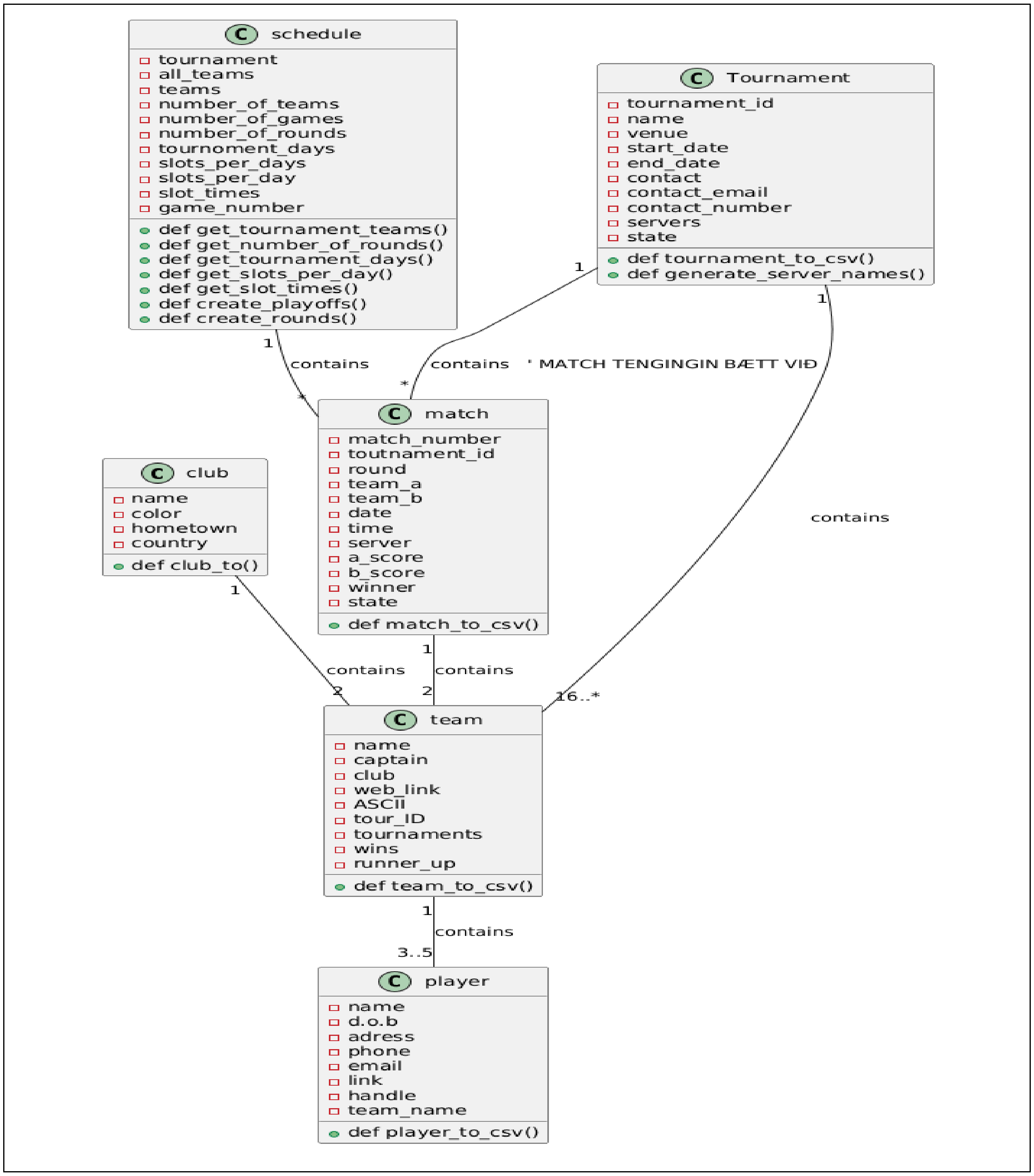
Class diagrams

A model diagram like the one below is used to map the structure of the system before (and during) implementation. It shows the main objects/entities, what data each one stores (attributes), what it can do (methods), and—most importantly—how the objects relate to each other (e.g., "one tournament has many matches"). This helps ensure the data design is consistent, makes it easier to spot missing information or wrong assumptions early, and acts as a shared blueprint for coding, database/storage, and testing.

More specifics about our model diagram:

- **Tournament** stores tournament info (dates, venue, contacts, servers, state) and contains many matches and many teams (16+).
- **Schedule** holds scheduling settings (days, slots, rounds, etc.) and generates rounds/playoffs, producing a list of matches.
- **Match** represents one game in the tournament and links to exactly two teams (team_a and team_b), plus time/date/server and results.
- **Team** stores team details and is connected to a **Club** and a **Tournament**.
- **Player** belongs to exactly one team, and each team contains 3–5 players.
- **Club** groups teams under the same organization (name, hometown, country, colour)

Model diagram



3-layer diagram

For a 3-layer (three-tier) diagram, the point is to show how your system is separated into clear responsibilities so it's easier to build, change, and debug.

- UI layer (Frontend):
  What the user interacts with (screens, forms, buttons). It collects input and shows output, but it shouldn't contain "real logic" like scheduling rules.
- Logic layer (The brain)
  This is the brain of the system. It contains the rules and workflows (e.g., create rounds, assign matches, validate data).
- Data layer (Database):
  Where information is stored and retrieved (tables, CSV files, etc.). This layer should not care about UI it just handles saving and loading.

APIs in the diagram

An API is basically the contract between layers:

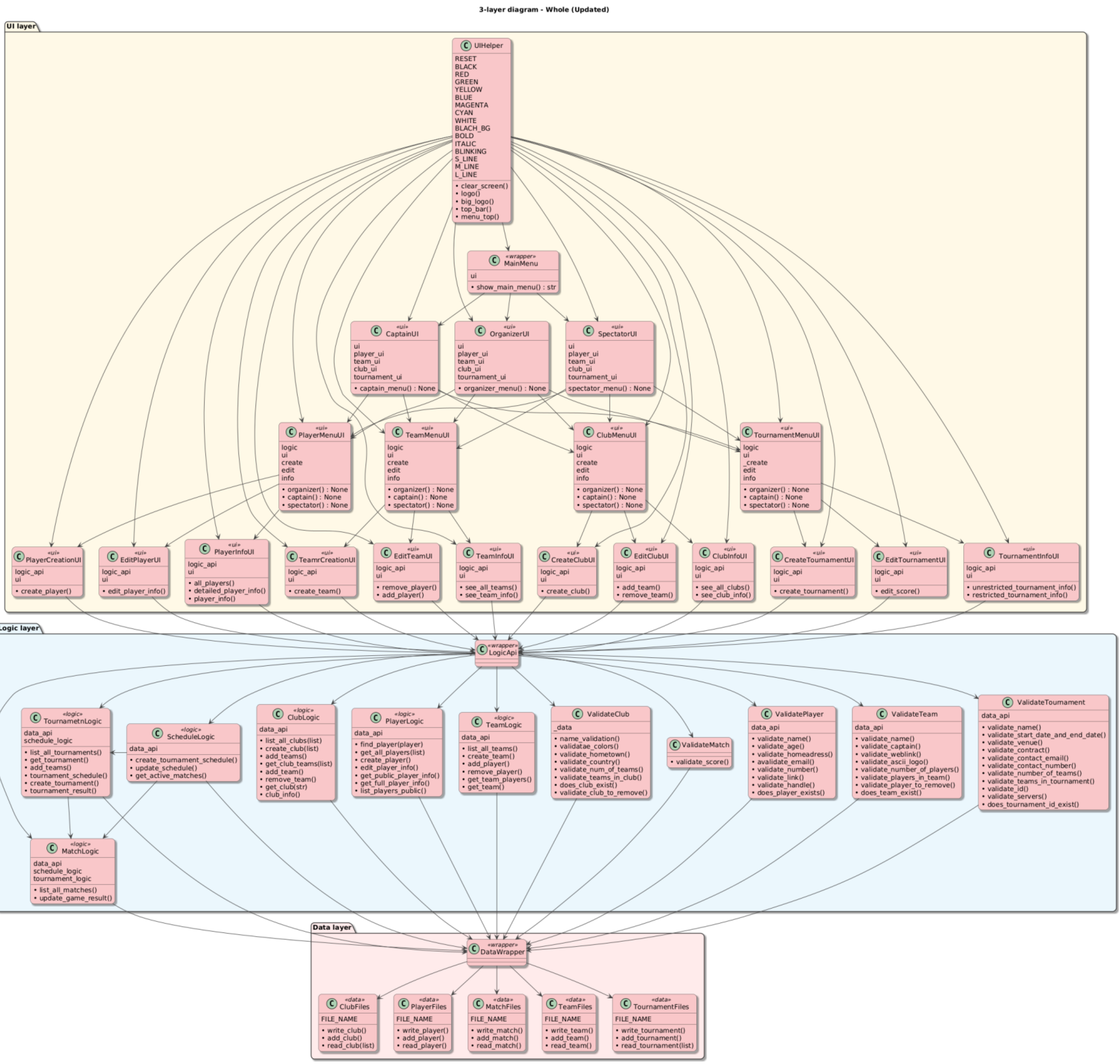
- The UI doesn't directly speak the database.
- Instead it calls the logic layer through a clear set of functions/endpoints (the API).
- The logic layer then uses the data API methods to talk to the data layer.

Why do we do this

This structure keeps the system modular:

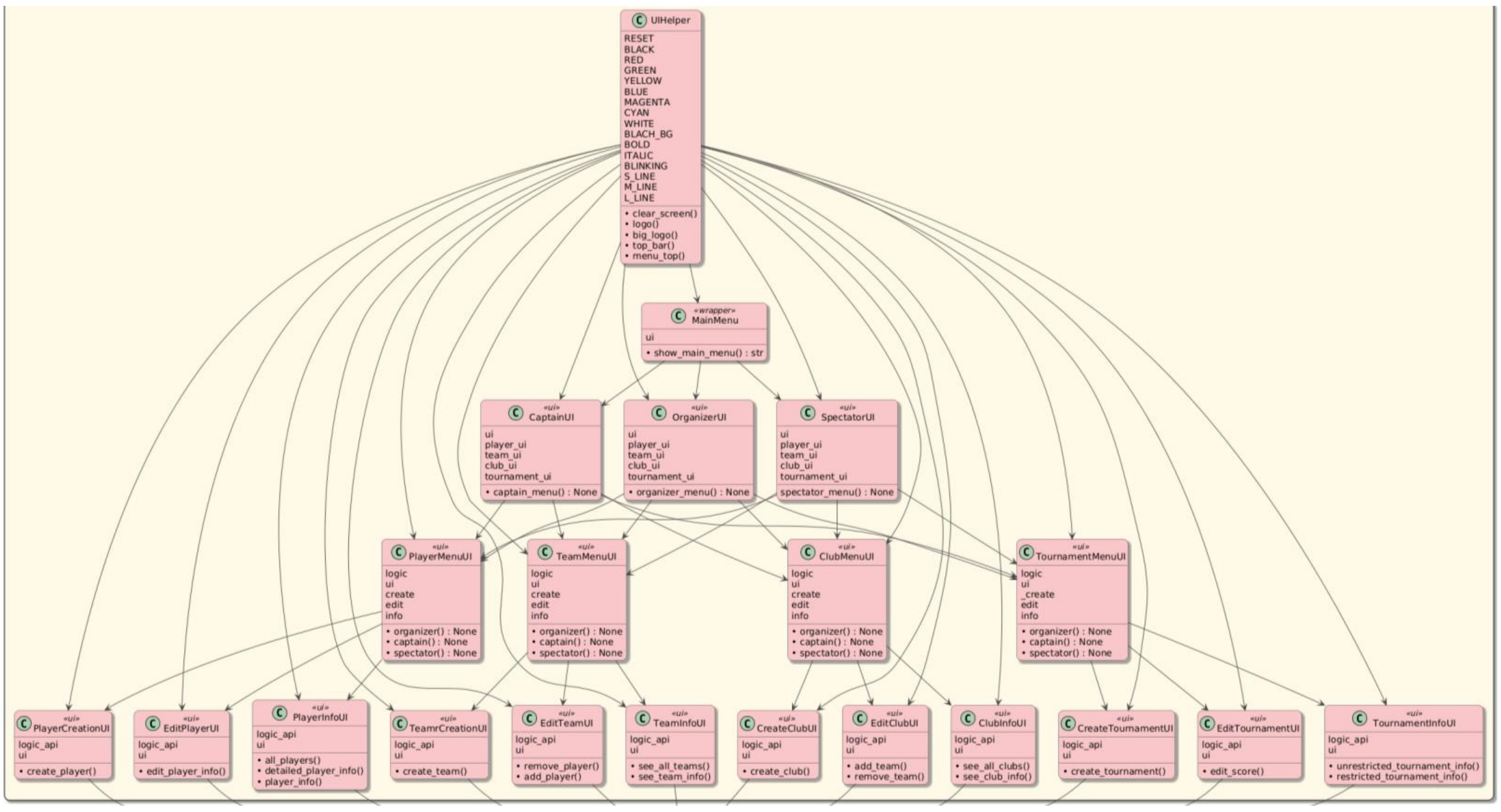
- You can change the UI without breaking storage
- You can swap CSV and database without rewriting the whole app
- Logic rules stay in one place, which reduces bugs and duplication

Whole

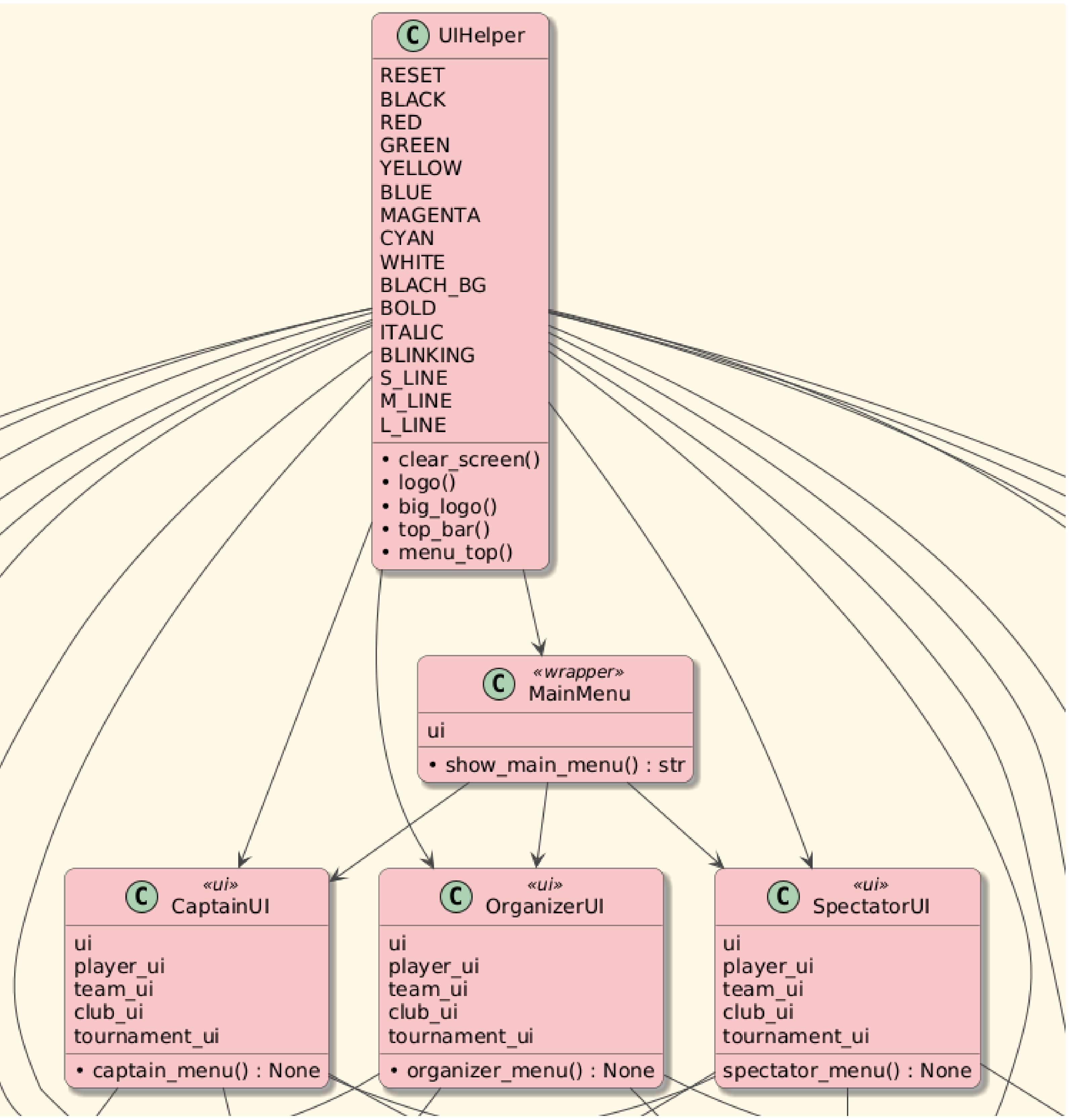

3-layer diagram - Whole (Updated)
UI layer
UIHelper
RESET
BLACK
RED
GREEN
YELLOW
BLUE
MAGENTA
CYAN
WHITE
BLACH_BG
BOLD
ITALIC
BLINKING
S_LINE
M_LINE
L_LINE
• clear_screen()
• logo()
• big_logo()
• top_bar()
• menu_top()
«wrapper» MainMenu
ui
• show_main_menu() : str
«ui» CaptainUI
ui
player_ui
team_ui
club_ui
tournament_ui
• captain_menu() : None
«ui» OrganizerUI
ui
player_ui
team_ui
club_ui
tournament_ui
• organizer_menu() : None
«ui» SpectatorUI
ui
player_ui
team_ui
club_ui
tournament_ui
spectator_menu() : None
«ui» PlayerMenuUI
logic
ui
create
edit
info
• organizer() : None
• captain() : None
• spectator() : None
«ui» TeamMenuUI
logic
ui
create
edit
info
• organizer() : None
• captain() : None
• spectator() : None
«ui» ClubMenuUI
logic
ui
create
edit
info
• organizer() : None
• captain() : None
• spectator() : None
«ui» TournamentMenuUI
logic
ui
_create
edit
info
• organizer() : None
• captain() : None
• spectator() : None
«ui» PlayerCreationUI
logic_api
ui
• create_player()
«ui» EditPlayerUI
logic_api
ui
• edit_player_info()
«ui» PlayerInfoUI
logic_api
ui
• all_players()
• detailed_player_info()
• player_info()
«ui» TeamrCreationUI
logic_api
ui
• create_team()
«ui» EditTeamUI
logic_api
ui
• remove_player()
• add_player()
«ui» TeamInfoUI
logic_api
ui
• see_all_teams()
• see_team_info()
«ui» CreateClubUI
logic_api
ui
• create_club()
«ui» EditClubUI
logic_api
ui
• add_team()
• remove_team()
«ui» ClubInfoUI
logic_api
ui
• see_all_clubs()
• see_club_info()
«ui» CreateTournamentUI
logic_api
ui
• create_tournament()
«ui» EditTournamentUI
logic_api
ui
• edit_score()
«ui» TournamentInfoUI
logic_api
ui
• unrestricted_tournament_info()
• restricted_tournament_info()
Logic layer
«wrapper» LogicApi
«logic» TournametnLogic
data_api
schedule_logic
• list_all_tournaments()
• get_tournament()
• add_teams()
• tournament_schedule()
• create_tournament()
• tournament_result()
«logic» ScheduleLogic
data_api
• create_tournament_schedule()
• update_schedule()
• get_active_matches()
«logic» ClubLogic
data_api
• list_all_clubs(list)
• create_club(list)
• add_teams()
• get_club_teams(list)
• add_team()
• remove_team()
• get_club(str)
• club_info()
«logic» PlayerLogic
data_api
• find_player(player)
• get_all_players(list)
• create_player()
• edit_player_info()
• get_public_player_info()
• get_full_player_info()
• list_players_public()
«logic» TeamLogic
data_api
• list_all_teams()
• create_team()
• add_player()
• remove_player()
• get_team_players()
• get_team()
ValidateClub
_data
• name_validation()
• validatae_colors()
• validate_hometown()
• validate_country()
• validate_num_of_teams()
• validate_teams_in_club()
• does_club_exist()
• validate_club_to_remove()
ValidateMatch
• validate_score()
ValidatePlayer
data_api
• validate_name()
• validate_age()
• validate_homeadress()
• avalidate_email()
• validate_number()
• validate_link()
• validate_handle()
• does_player_exists()
ValidateTeam
data_api
• validate_name()
• validate_captain()
• validate_weblink()
• validate_ascii_logo()
• validate_number_of_players()
• validate_players_in_team()
• validate_player_to_remove()
• does_team_exist()
ValidateTournament
data_api
• validate_name()
• validate_start_date_and_end_date()
• validate_venue()
• validate_contract()
• validate_contact_email()
• validate_contact_number()
• validate_number_of_teams()
• validate_teams_in_tournament()
• validate_id()
• validate_servers()
• does_tournament_id_exist()
«logic» MatchLogic
data_api
schedule_logic
tournament_logic
• list_all_matches()
• update_game_result()
Data layer
«wrapper» DataWrapper
«data» ClubFiles
FILE_NAME
• write_club()
• add_club()
• read_club(list)
«data» PlayerFiles
FILE_NAME
• write_player()
• add_player()
• read_player()
«data» MatchFiles
FILE_NAME
• write_match()
• add_match()
• read_match()
«data» TeamFiles
FILE_NAME
• write_team()
• add_team()
• read_team()
«data» TournamentFiles
FILE_NAME
• write_tournament()
• add_tournament()
• read_tournament(list)

UI layer

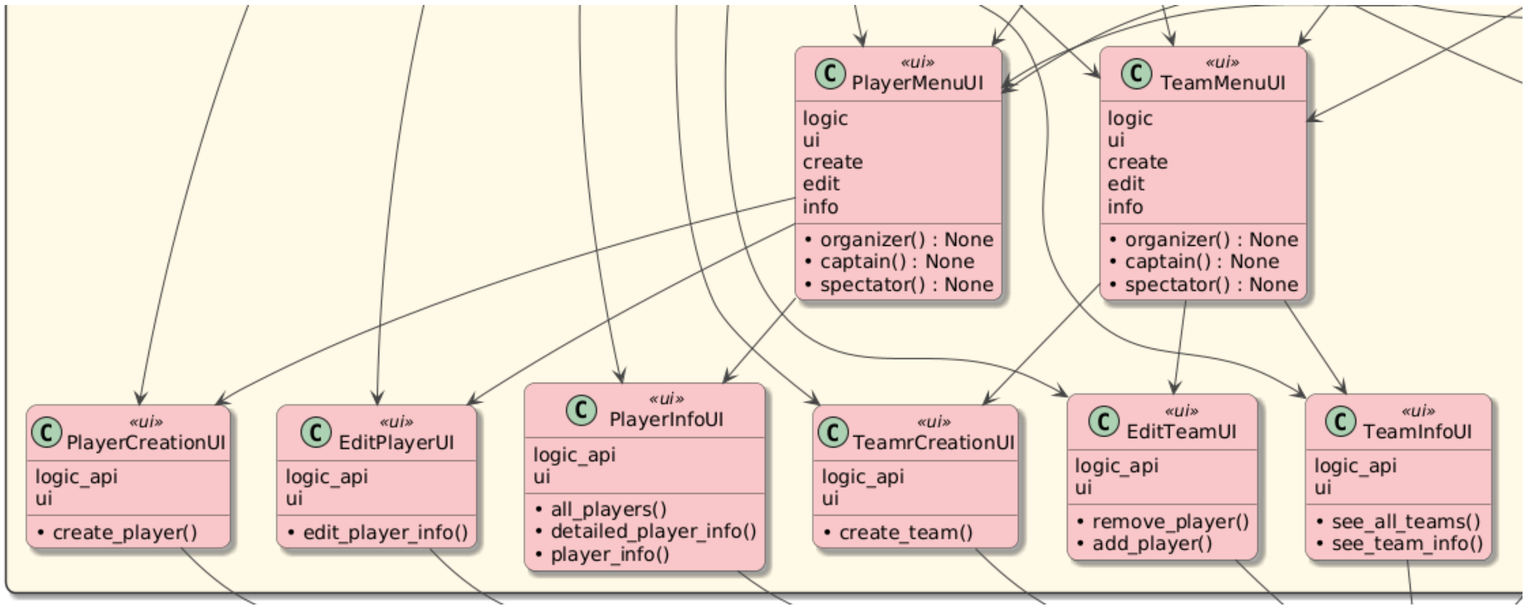
Whole UI layer



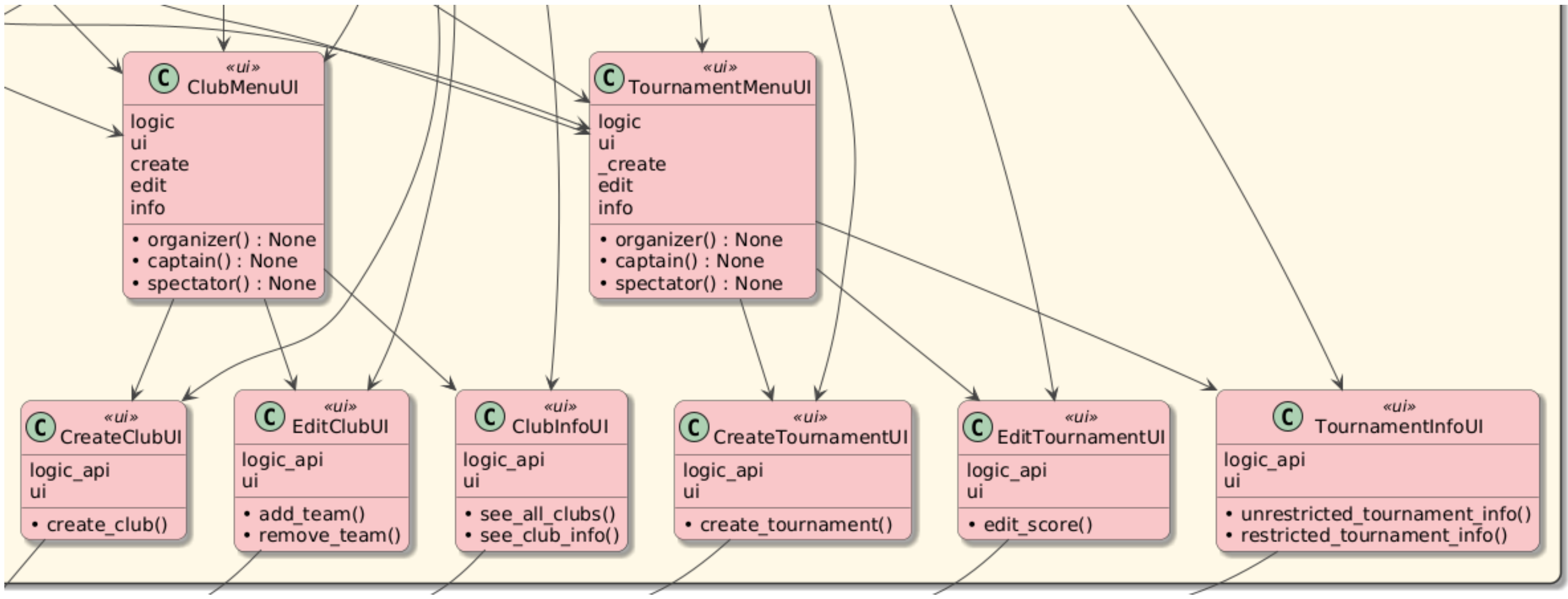
UI top


UIHelper
RESET
BLACK
RED
GREEN
YELLOW
BLUE
MAGENTA
CYAN
WHITE
BLACH_BG
BOLD
ITALIC
BLINKING
S_LINE
M_LINE
L_LINE
• clear_screen()
• logo()
• big_logo()
• top_bar()
• menu_top()
«wrapper»
MainMenu
ui
• show_main_menu() : str
«ui»
CaptainUI
ui
player_ui
team_ui
club_ui
tournament_ui
• captain_menu() : None
«ui»
OrganizerUI
ui
player_ui
team_ui
club_ui
tournament_ui
• organizer_menu() : None
«ui»
SpectatorUI
ui
player_ui
team_ui
club_ui
tournament_ui
spectator_menu() : None

UI bottom left

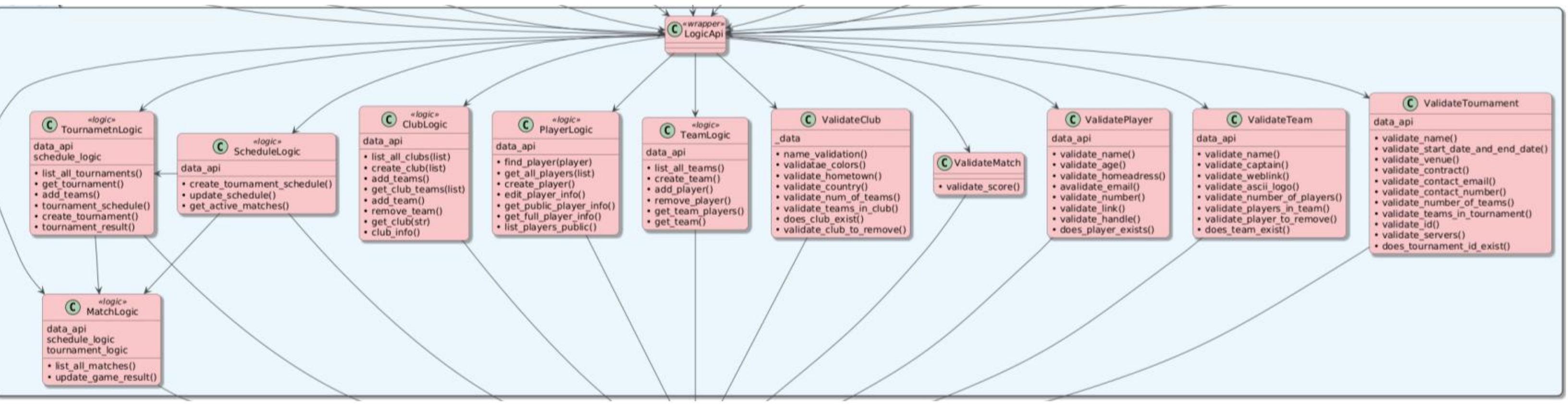


UI bottom right

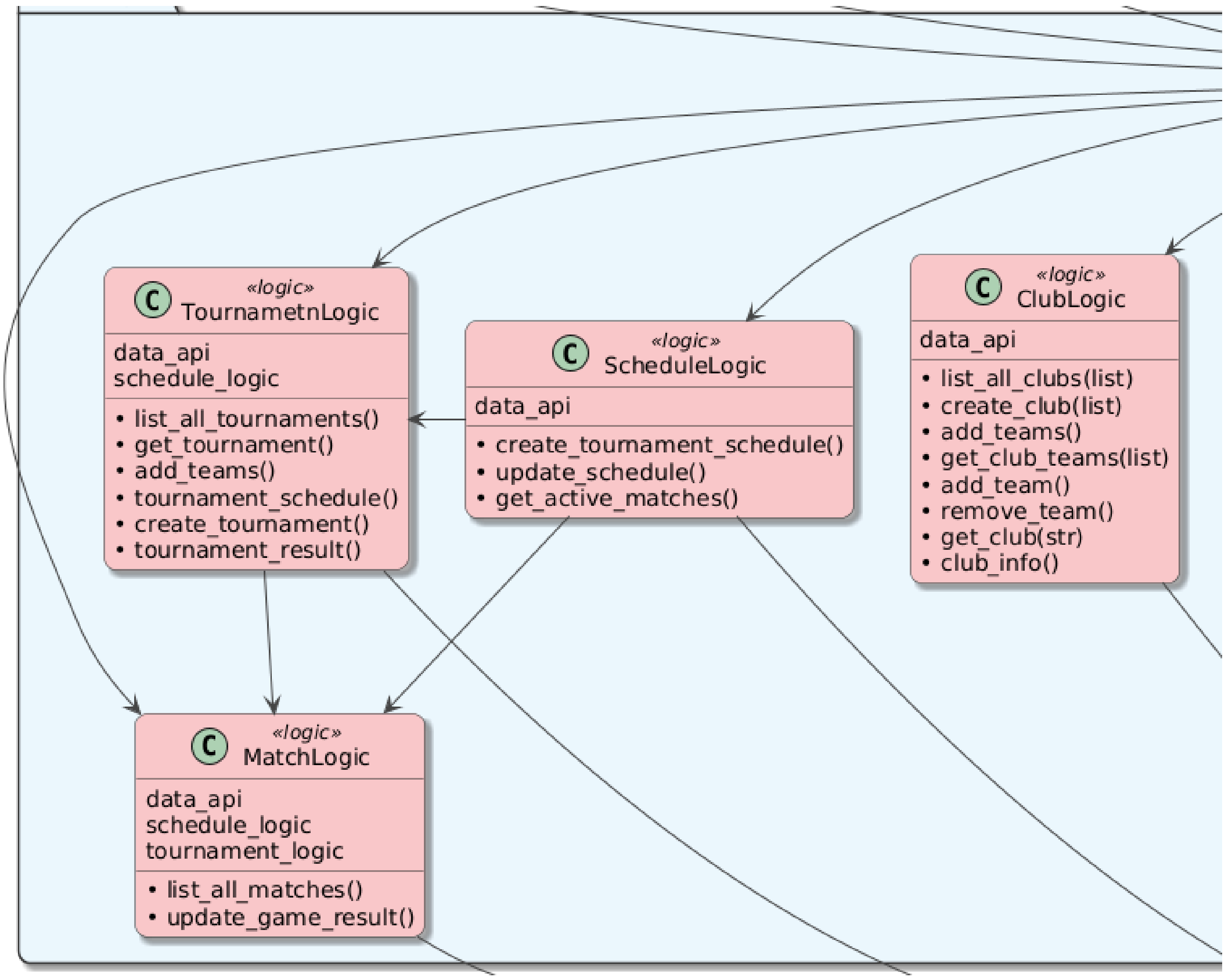


Logic Layer

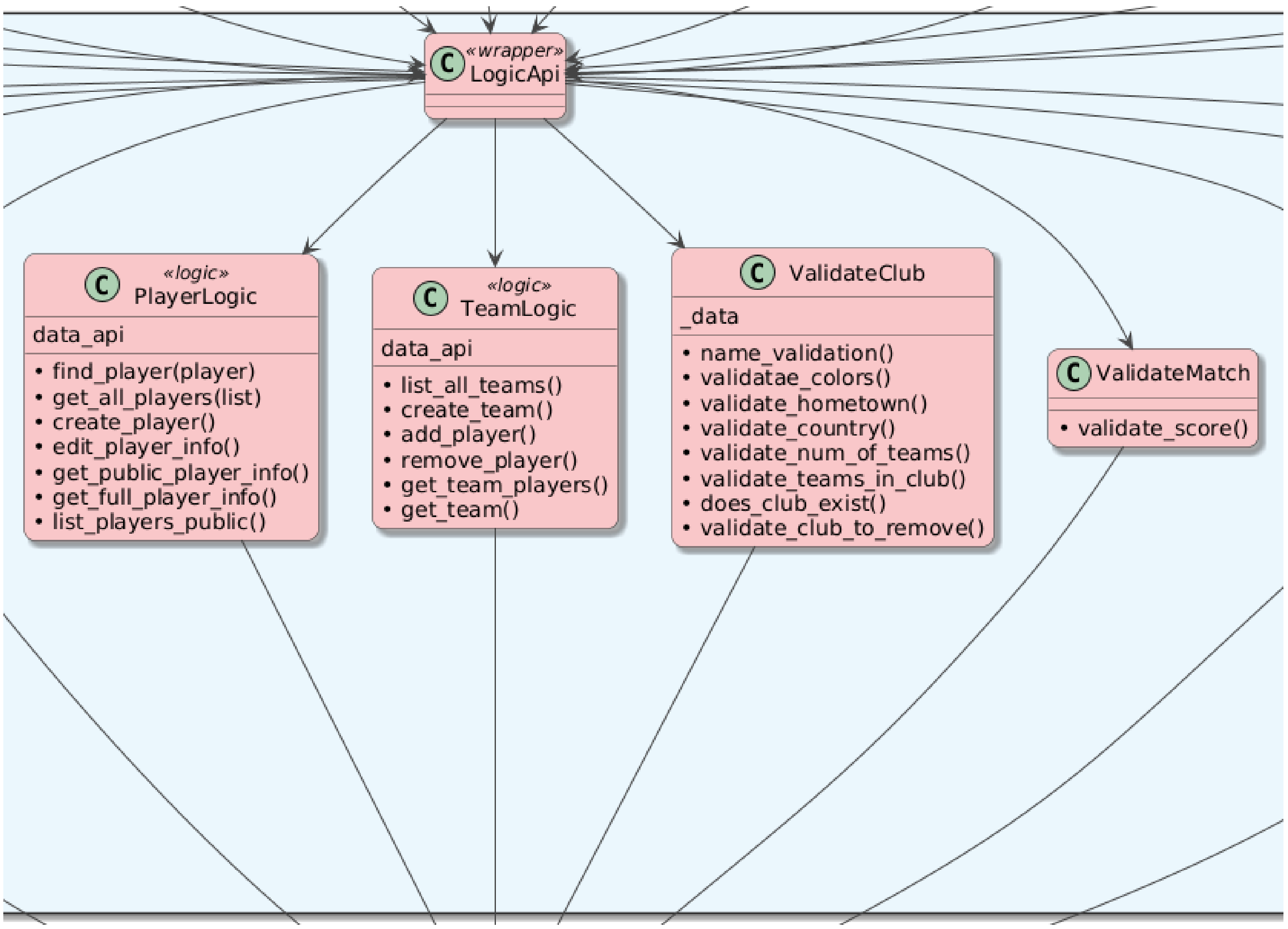
Whole



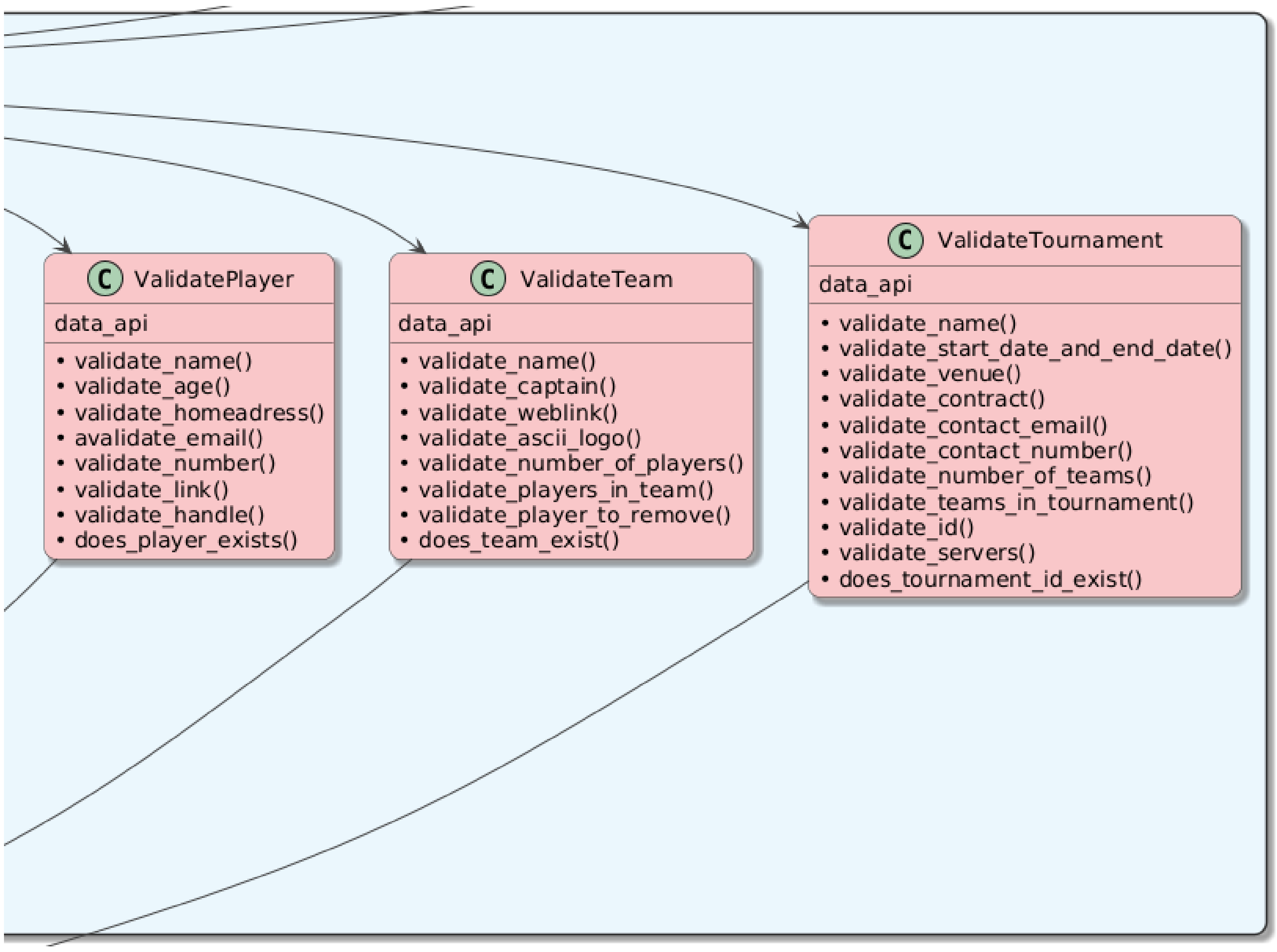
Left



Middle

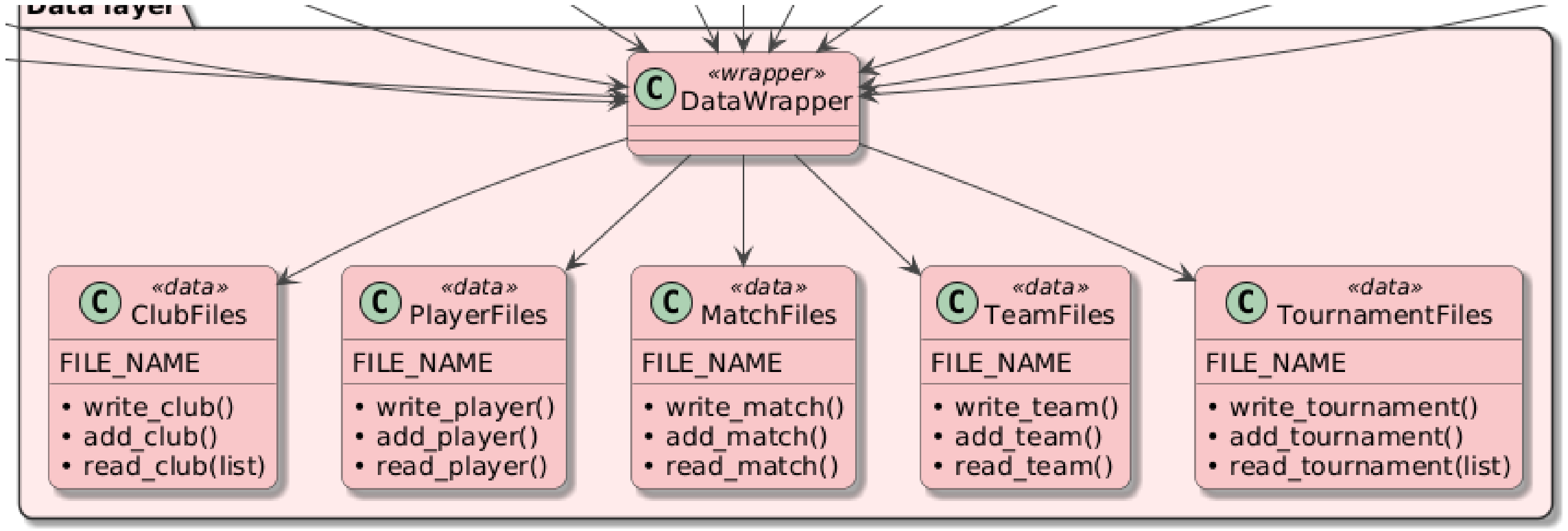

«wrapper»
LogicApi
«logic»
PlayerLogic
data_api
• find_player(player)
• get_all_players(list)
• create_player()
• edit_player_info()
• get_public_player_info()
• get_full_player_info()
• list_players_public()
«logic»
TeamLogic
data_api
• list_all_teams()
• create_team()
• add_player()
• remove_player()
• get_team_players()
• get_team()
ValidateClub
_data
• name_validation()
• validatae_colors()
• validate_hometown()
• validate_country()
• validate_num_of_teams()
• validate_teams_in_club()
• does_club_exist()
• validate_club_to_remove()
ValidateMatch
• validate_score()

Right


ValidatePlayer
data_api
• validate_name()
• validate_age()
• validate_homeadress()
• avalidate_email()
• validate_number()
• validate_link()
• validate_handle()
• does_player_exists()
ValidateTeam
data_api
• validate_name()
• validate_captain()
• validate_weblink()
• validate_ascii_logo()
• validate_number_of_players()
• validate_players_in_team()
• validate_player_to_remove()
• does_team_exist()
ValidateTournament
data_api
• validate_name()
• validate_start_date_and_end_date()
• validate_venue()
• validate_contract()
• validate_contact_email()
• validate_contact_number()
• validate_number_of_teams()
• validate_teams_in_tournament()
• validate_id()
• validate_servers()
• does_tournament_id_exist()

Data Layer

Whole



Conclusion

This design report mapped out the core structure of our e-Sports tournament maker before full development. By defining the purpose of the system and its scope early (tournament creation, match scheduling, result tracking, and management of teams/players/organizers), we created a clear foundation for a Python implementation and a realistic path toward the beta deadline on December 12th.

The requirements list gave us a prioritized checklist of what the system must include, separating functional and non-functional needs and ranking them from A–C by importance. This helped us focus first on essential features required for the system to operate, then expand into quality-of-life improvements. The updated requirement status also shows real progress: A and B requirements are implemented and functional except for one B requirement, with one C requirement implemented.

Our use cases and happy paths turned these requirements into concrete user goals and step-by-step interactions. By including preconditions, base flows, alternative flows, postconditions, and actors, we ensured that the system is designed around realistic user behavior and that common failures (like invalid input) are handled in a structured way.

Finally, the diagrams (state/class/model and 3-layer architecture). The model diagram defines the main entities and relationships in the tournament system, while the layered architecture separates UI, logic layer, and data layer through clear APIs/wrappers making the system easier to maintain, test, and extend.

Overall, the combination of requirements, use cases (with happy paths), and architecture/design diagrams provides a complete blueprint that supports both implementation and future improvement of the tournament maker.