Board Games Distributed Systems Semester Project 2020

Frameworks, libraries and Tools

- Programming languages:
  - Python
- Frameworks/libraries:
  - Flask
  - Flask-socketIO
  - Flask-marshmallow
  - Flask-jwt-extended
  - Flask-sqlAlchemy
  - Flask-migrate
  - kazoo
  - Zookeeper
- Tools
  - PostgreSQL
  - Docker

Frameworks/libraries - 1

- Flask:
  - Micro – web framework
  - supports extensions
- Flask-socketIO:
  - SocketIO support for flask application using low latency bi-directional communications between the clients and the server.
- Flask-jwt-extended:
  - In the register process, the client provides a username and a password. The server stores the hash/password to the database and creates a token, which is also stored to the database. During the login process, the server sends an access token to the client and client store it as a browser cookie

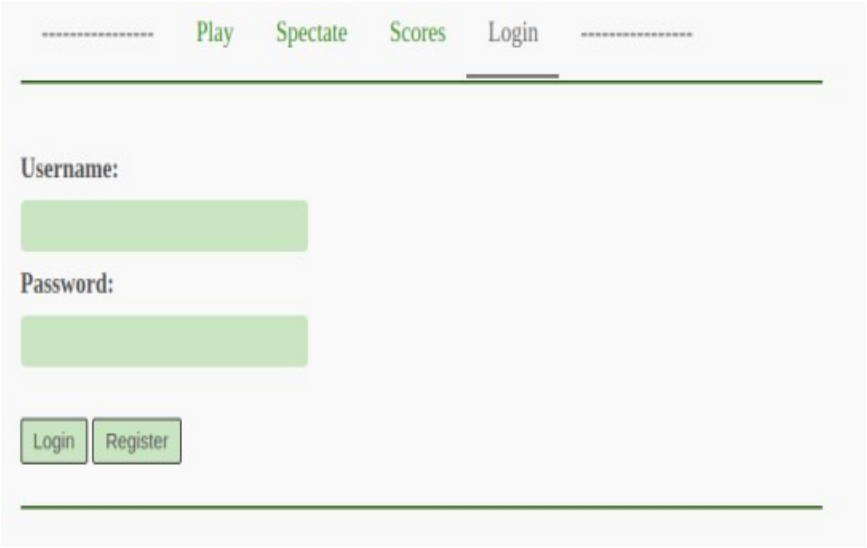
Frameworks/libraries - 2

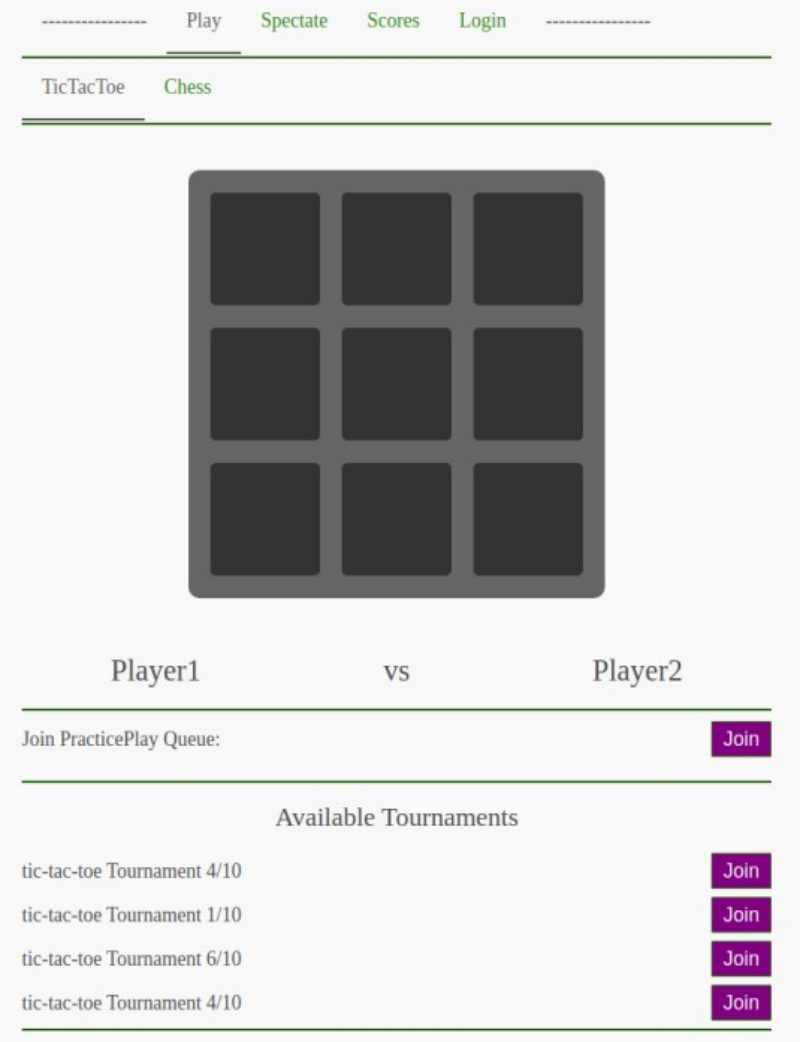
- Flask-sqlAlchemy:
  - Gives the ability to establish the communication between the database and the server
- Flask-marshmallow:
  - An ORM framework-agnostic library for converting complex datatypes, such as objects, to and from native Python datatypes.
- Flask-migrate:
  - Creating and maintaining the database schema defined in a python script
- Zookeeper
  - Service discovery
  - Play master crash recovery
  - Zookeeper Client: kazoo framework

Services

User-Interface-UI

- Jinja Templates:
  - template inheritance
  - Inject python variables values in to the html page
- Javascript:
  - socketIO : real-time bidirectional event-based communication
  - XMLHttpRequest: clients request data from server





User-Interface-1

- The bridge between the clients and the application micro services
- Message forwarding, injecting the user's token in the request header
- Receives game updates from PlayMaster and broadcast the game updates to clients connected to the specific socketio namespace

Authentication

- Communicate with User-Interface-Service and is responsible for the register and login process
- Stores User info in postgreSQL database and communicates with it through framework flask-sqlAlchemy
- On User register, it creates an access token in which is stores user specific information such as username and user role

Authenticate	Method	Path	Input
UserLoginResource	GET	/login	username, password
UserResource	GET	/users/ <int:user_id>	-
UserListResource	GET	/users/	-
UserListResource	POST	/users/	username, password, role

Play-Master

- Manages the hosting and execution of actives games
- Receives and processes the player input through the user interface service
- When a game state is updated, it informs the user interface about the occurred change in order to notify the clients interested in that game
- After a game is finished, it informs the game master about the result of the game
- Receives request from users that are interesting in a game and return the game state of the specific game

Play-Master

PlayMaster	Method	Path	Input
ChessGameRegisterResource	POST	/games/chess/register	play_id player1 Player2
ChessGameResource	GET	/games/chess/<int:play_id>	-
ChessGamePlayMoveResource	POST	/games/chess/<int:play_id>/ play	player_id square_fro m square_to
GameStateResource	GET	/games/<int:play_id>	-
PlayerGameStateResource	GET	/games/player/<int:player_id>	-

Game-Master

- Communicates with Database, User-Interface-Service and Play-Master-Service and is responsible for the matchmaking process and the creation of tournaments and the monitoring of active games
- After pairing two players, a notifications is sent to the play master in order to host the actual game. If the request is successful, the game master updates game status to active
- Informs the play-master-service about the creation and hosting of game when a pair of players are matched
- Generally keeps the status of games and tournaments and saves them in the database when it’s confirmed by the play master that a game or tournament has ended

Game-Master

GameMaster	METHOD	PATH	INPUT
PlayResource	GET	/plays/<int:play_id>	-
TournamentResousrce	GET	/tournaments/<int:tournament_id>	-
UserPlayResource	GET	/userplays/<int:user_id>/ <int:play_id>	-
UserTotalScoreResource	GET	/usertotalscore/<string:play_mode>	username

Game-Master

GameMaster	METHOD	PATH	INPUT
UserScoreListResource	GET	/userscorelist/ <string:play_mode>	username
ActivePlaysResource	GET	/plays/active	-
PracticePlayJoinResource	POST	/practiceplay/join	gametype
OpenTournamentsRegisterResource	POST	/opentournaments/register	player_id tournament_id

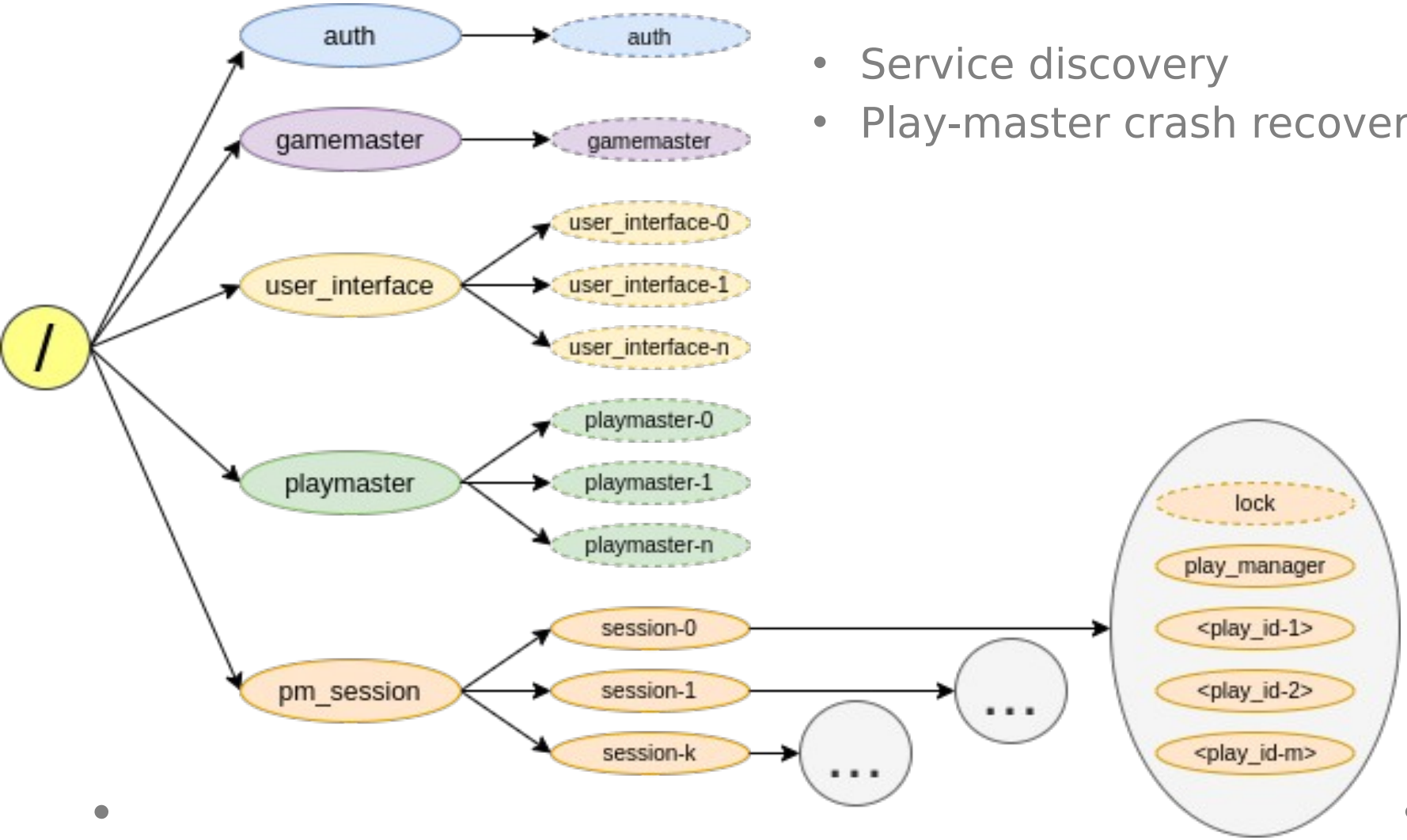
PostgreSQL-Schema

- User - Table
  - Primary key: ID
  - username, password, role, token
- Play - Table
  - Primary key: ID
  - gametype
- Tournament - Table
  - Primary key: ID
  - gametype, participants_number, creator_id
- User Play - Table
  - Primary key: userID, playID
  - tournamentID, tournament_rank, result
- User-Total-Score - Table
  - Practice(wins, loses, draws), Tournament(wins, loses, draws)

- The User table is used only by the authentication service
- The other four tables are used only by the game-master service

Zookeeper

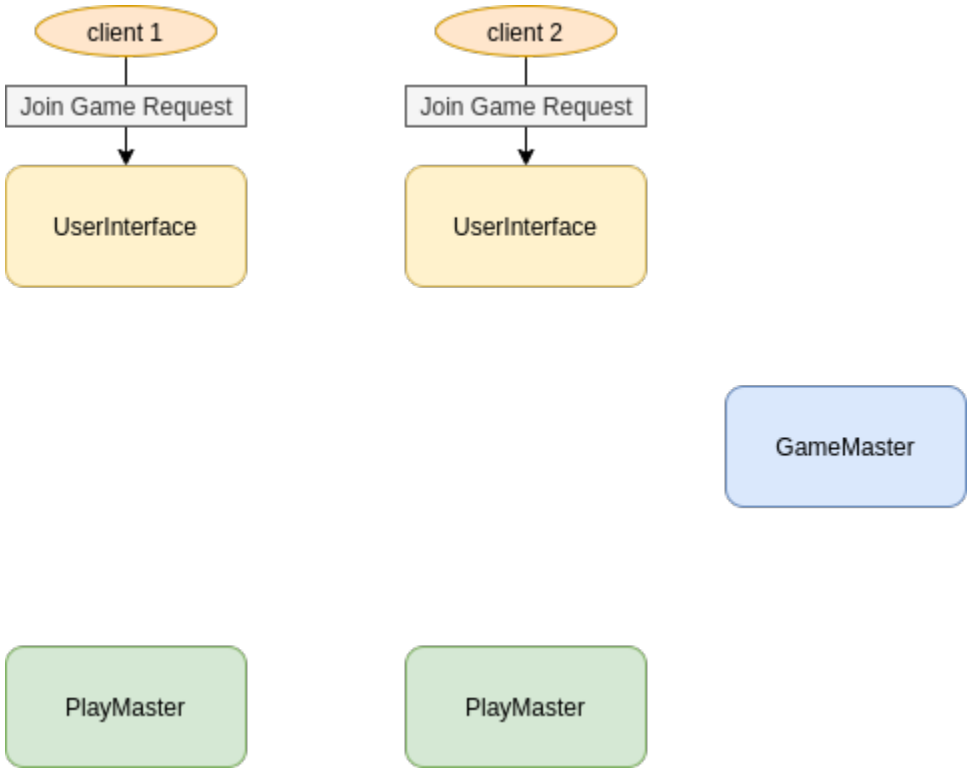
- Service discovery
- Play-master crash recovery



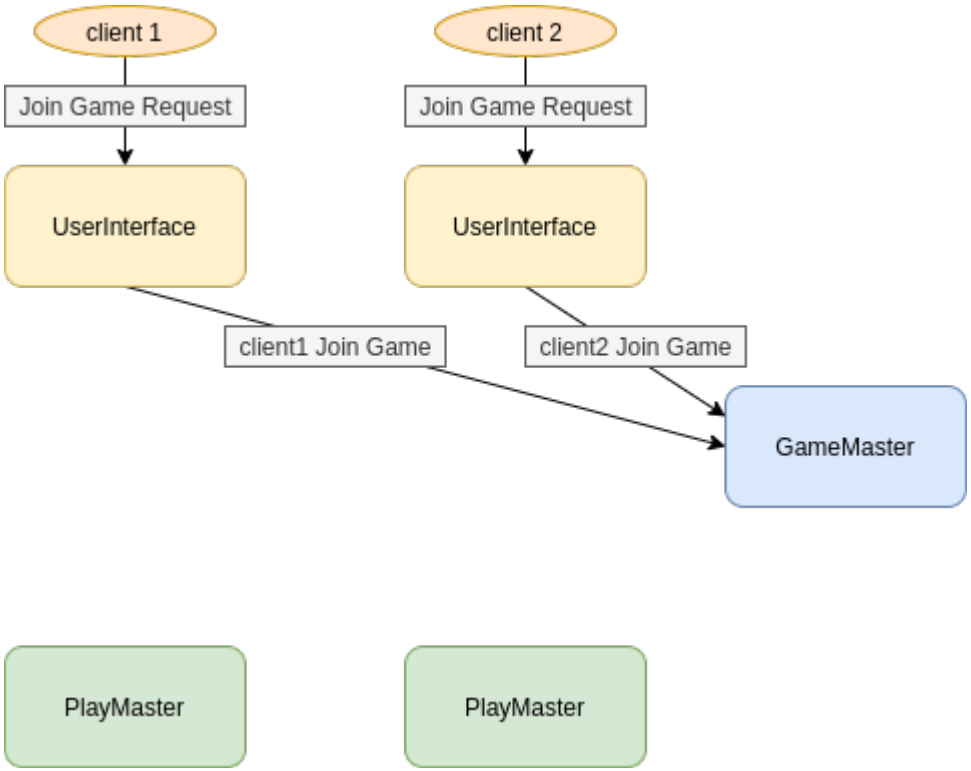
Docker

- Docker images
  - Authentication service
  - User Interface service
  - Game master service
  - Play master service
  - Zookeeper server
  - PostgreSQL database

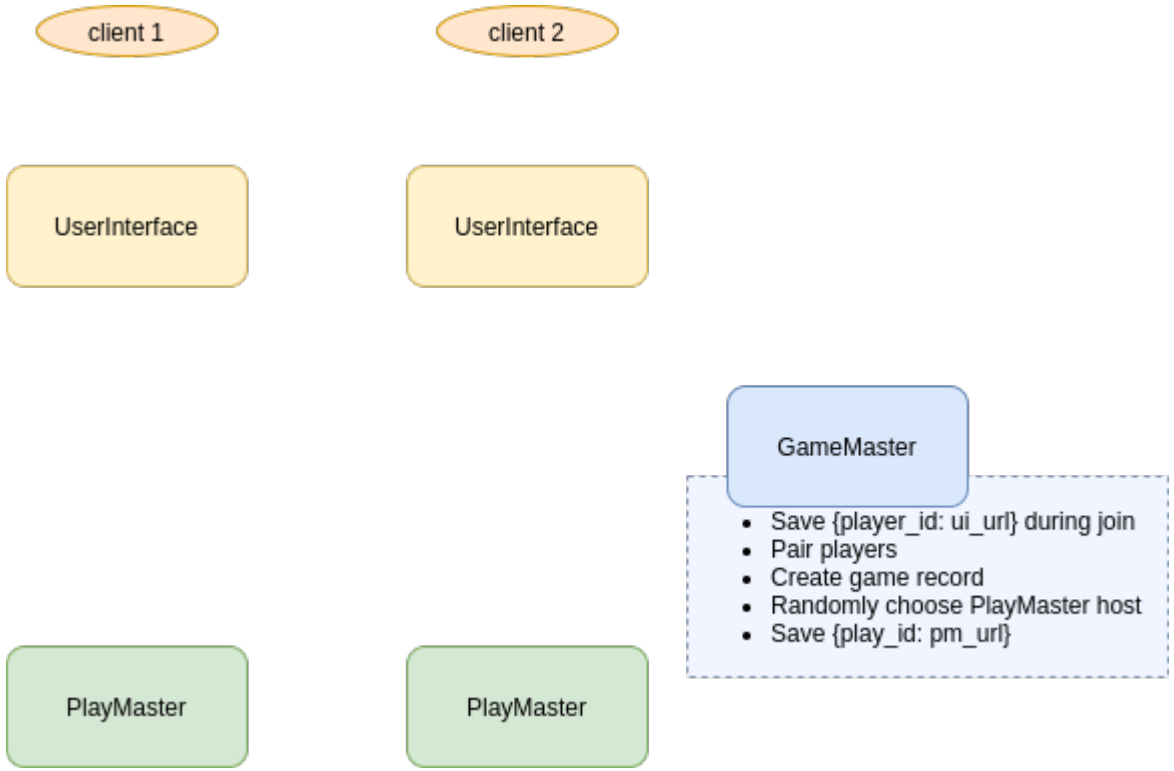
Use Case Scenario



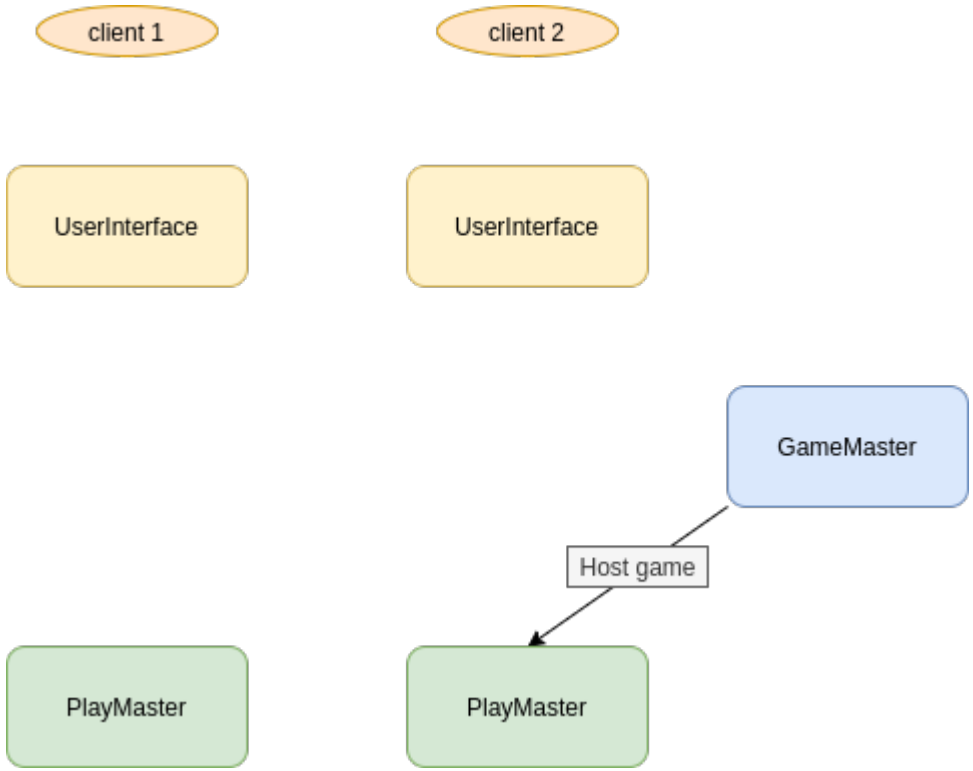
Use Case Scenario



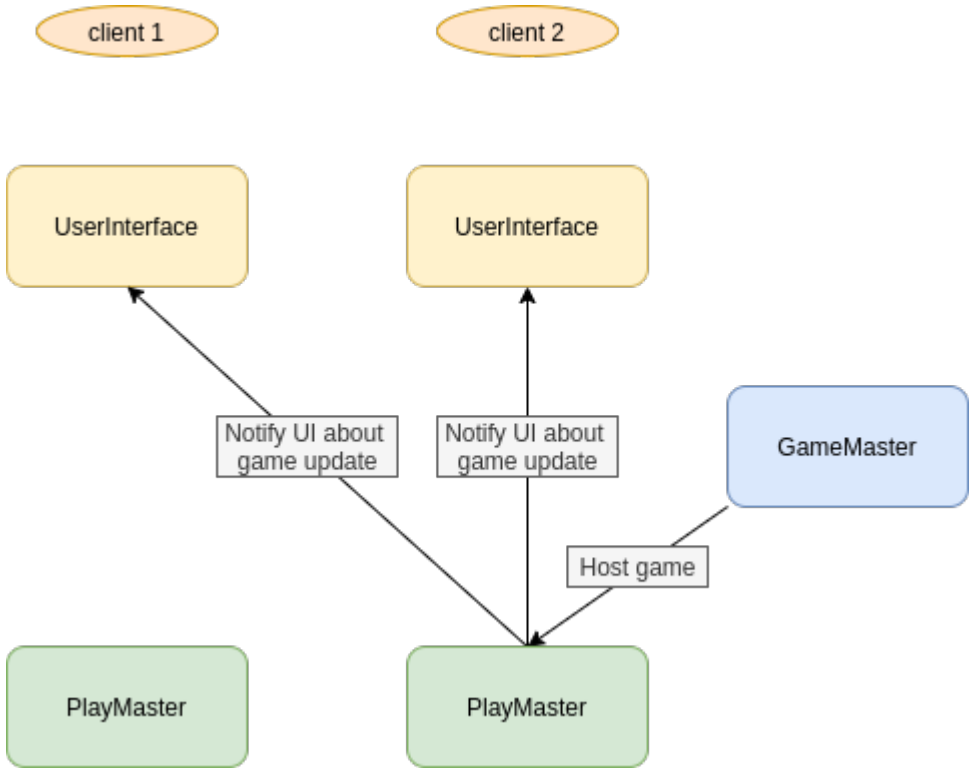
Use Case Scenario



Use Case Scenario



Use Case Scenario



Use Case Scenario