Bilkent University

Department of Computer Engineering

CS 319 Course Project

Project Short-Name: Monopoly Bilkent

Design Report Iteration 1

- Mehmet Yaylacı
- Vural Doğan Akoğlu
- Mert Laleci
- Barış Tiftik
- İlhan Koç

Supervisor: Eray Tüzün

Contents

Introduction **3**

Purpose of the system **3**

Design goals **3**

- Performance Criteria 3
- Dependability Criteria 4
- Cost Criteria 4
- Effectiveness 4
- Maintenance Criteria 5
- Changeability 5

High-level software architecture **6**

- Subsystem decomposition 6
- Hardware/software mapping 7
- Persistent data management 7
- Access control and security 8
- Boundary conditions **8**
  - Initialization 8
  - Termination 8
  - Failure 9

Low-level design **9**

- Object Design Trade-Offs **9**
  - Efficiency vs. Portability 9
  - Functionality vs. Usability 9
  - User Experience vs. Development Time 10

Final Object Design **10**

- Server Side 10
  - Models Component 17
- Client Side 20
  - UI Component 20

1. Introduction

1.1. Purpose of the system

The purpose is mainly to convert the Monopoly board game into an online pc game by adding some extra features to its board version. The Board version of the game limits the gameplay. However, the online version of the game intends to be more flexible and players can play the game in quarantine times with no physical contacts to each other. Also, most people already play monopoly (board version) in a classical way but after several times it can be boring. Therefore, we add some additional changes. The online version offers a bargain opportunity between two players and 4 different modes to finish the game quickly. That's why players need to think and implement different strategies so that their opponents don't finish the game fast and cannot become winners. The other section of this report shows some information about how we can design our system.

1.2. Design goals

1.2.1. Performance Criteria

When it comes to games played by many people and based on the turn system such as our game monopoly (2-6 players), the importance of the performance of the fame increases as the players can get bored until their turn. The player's turn needs to be 1-1.5 minutes, but in their turns players can roll dice, their token can be moved according to dice and they can have multiple actions such as buying property, picking cards and applying appropriate action on the cards, building houses/hotels, making offers to the opponent player. Therefore, all of these actions must be done quickly and their response time needs to be minimized. In addition, the menu side of the game and their response time are also important such as sign-up, sign-in, opening game settings and modes, creation of the lobby, joining lobby and we state approximately response time of these actions in the analysis report part 3.4.2.

1.2.2. Dependability Criteria

In Monopoly Bilkent edition, dependability is important and related with usability, availability and functionality. In the goal of usability, game screen design needs to be simple and efficient such as in the menuScreen there is always an information box (?) which shows players what they can do on this screen and which buttons are used for. In the game whenever player's turn start their gameScreen displays the

properties he/she owned, if there is any chance and community chest card he/she had, the amount of money in own bank account and also boardScreen displays which property owned by which player and all bank accounts. Such design features reduce the need of memorization, and help players to set up different strategies to bargain with the opponent players. In the goal of availability, many players have access to the game all time in the game to play or before the game because of sign-up/sign-in or creating lobby/ joining lobby. Lastly, the goal of functionality is quite important because we want the players to set their own rules and play according to them, so that they can be effective in the design of the game and use the game as they wish.

1.2.3. Cost Criteria

Although this is a term project, cost criteria is important for us because of the time restrictions. Therefore, as the developers, we will increase our time by making well planned, project focused meetings.

1.2.4. Effectiveness

In the analysis part of this project, we think about dozens of additional features but there is limited time to develop this game. So, during the design and implementation period, we evaluated the most effective options among others. And while making this decision, we have taken into consideration what we believe are easier, actually applicable and more important.

1.2.5. Maintenance Criteria

Our project(Monopoly Bilkent Edition) is based on OOP, that's why it has a hierarchical structure(subsystems, interfaces, abstract classes, subclasses). Therefore, without any maintenance problems. the wanted or additional features can be implemented to this project easily.

1.2.6. Changeability

Since our game consists of some basic classes that differ in subclasses (for ex: a place in the board such as property, estate etc.), our software will be adaptable to the changes easily.

2. High-level software architecture

2.1. Subsystem decomposition

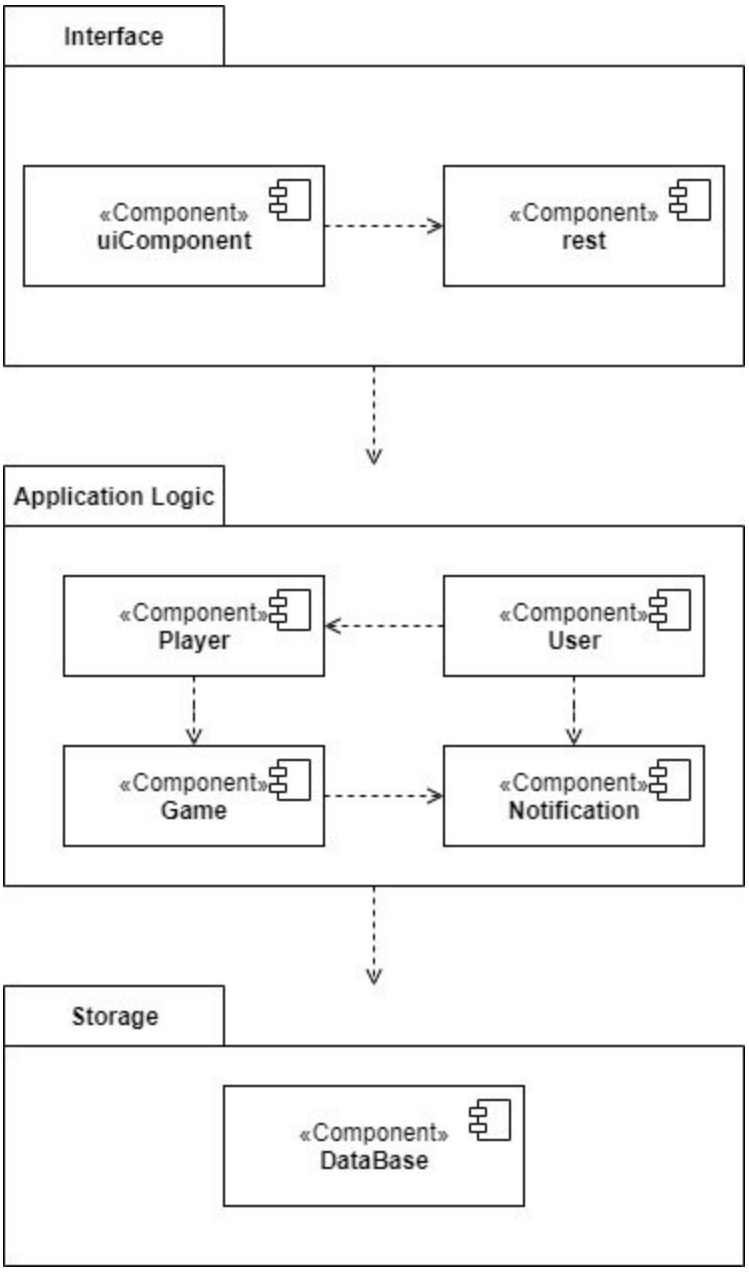

Figure 1: Subsystem Decomposition Diagram

In this project, the closed three-tier architectural style is used. The interface package consists of two components that are “uiComponent” and “rest”. The “uiComponent” represents the view of the user in the client and “rest” component shows the response of the server to the user’s actions in the client. “Rest” component will be explained in this report with more details on “Server Side”.

The application logic package includes classes “user”, “player” and “game”. The information about the user comes from the inputs of the user, so the user can directly want a request through notification component (such as login, change avatar etc.) or the user can enter a game and become a player and send requests through game actions.

The storage package includes one database that holds information about users such as username, password, and their avatars.

2.2. Hardware/software mapping

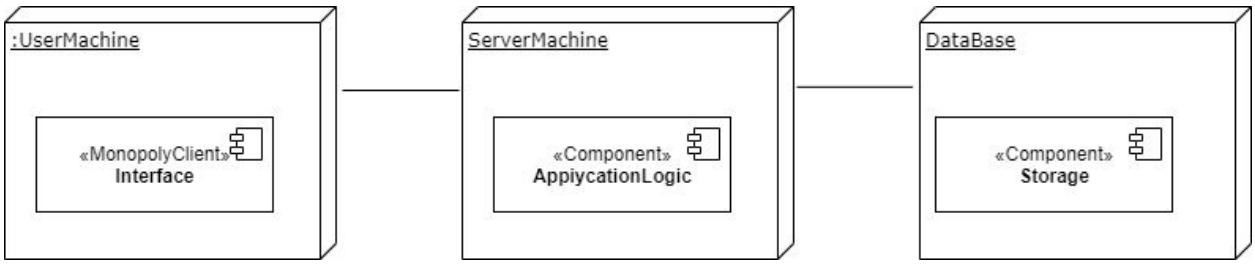

Figure 2: Hardware/Software Mapping

The user machine represents each user's hardware with our game client in it. In the client a user can interact with the server and the database. Since our game will be available only for Windows operating systems, the diagram does not show any operating system. After the interaction, the system will send a response through the server machine, therefore, every user in the game can see the response at the same time.

2.3. Persistent data management

Our project, which is the Monopoly game, requires a local storage which includes the avatars of the players, the images of the board such as the properties, chance and community card decks. In addition, all necessary game data such as detailed information of the players and lobbies which players connect to play games will be stored in the database of the game.

2.4. Access control and security

Since our game uses Restful API, most of the requests will go through our server. For the lobby part, because of the unique lobby code, most of the time relevant users will be able to join easily.

Actors/Object	Lobby	Game Engine
LobbyCreator	view() changeMode() startGame() assignNewHost() dismissPlayer() quit()	play() quit()
Other Users in Lobby	view() applyForOwner() quit()	play() quit()

Figure 3 : Lobby and in-game permissions

2.5. Boundary conditions

2.5.1. Initialization

The initialization of the Monopoly game, players need to register for the game system by using the sign-up scene and the system checks the username that is chosen by the player that already exists in the system. After registration is completed players can enter the system by using a sign-in scene. At this stage, the system checks whether the username and password used by the player exist or match. Players who complete this part can set up their own lobbies as well as log into other lobbies where they have the lobby code. If the player wanted to set up a lobby, the system assigns that player unique lobby code that is not used in another lobby. Other players can enter the same lobby with this unique code. When the players in the lobby are ready, the creator of the lobby starts the game, while the loading screen is shown to the players, the system performs functions such as creating board, property, chance and community cards and assigning and arranging bank accounts of all players in the background.

2.5.2. Termination

In the lobby view, the creator of the lobby will be the host of the lobby and have access to more functions such as dismissing a player in the lobby. When the creator leaves the lobby, the permissions of the creator pass to

another player in the lobby. The creator can also assign a player as the new host. All players will have the same permissions when the game starts. If one of them leaves the game, he/she cannot rejoin the game.

2.5.3. Failure

In the login screen, if the user's password and username does not match, an error message will be shown to the user.

In the registration screen, if the user chooses an invalid name that is already taken, a warning message that informs the user to change the name will be shown in the screen.

In the lobby, if a connection failure occurs for one of the players, the system will wait for the player if the number of players is not enough to start the game, otherwise the system continues to run.

If the creator of the lobby has the connection problem, then the system will directly assign a new host to the lobby. If there is no player in the lobby, the lobby will be closed after the 10 seconds of waiting time.

In the game the system will wait for the players who have the connection problem for 10 seconds then will give the turn to the next player (the system will skip the player with connection problem)

3. Low-level design

3.1. Object Design Trade-Offs

3.1.1. Efficiency vs. Portability

The monopoly game is just for the desktop platform. We focus only on PC support for the Monopoly Bilkent. Having a PC platform will let us develop a more efficient game in terms of performance. However, this will result in a little bit of a problem in terms of portability because the desktop is outdated by mobile devices in portability.

3.1.2. Functionality vs. Usability

Monopoly is already a simple and straightforward game. The game pops up some decision choices to users whether to buy a property or not. The user rolls dice to move and buy some property, pay rent, etc. In order to make this game more interactable, we added some new features. These can be read in the

requirements report. We believe that these extra features will increase the functionality of the game. However, these additions may take some extra time when users play. This decreases the usability of the game since the game becomes more detailed.

3.1.3. User Experience vs. Development Time

User experience and development time are two important components of a project that have direct proportion to each other. Therefore, because of the limited development time, our version of monopoly may have a lack of quality of the user experience in terms of interface and some user friendly functions.

3.2. Final Object Design

3.2.1. Server Side

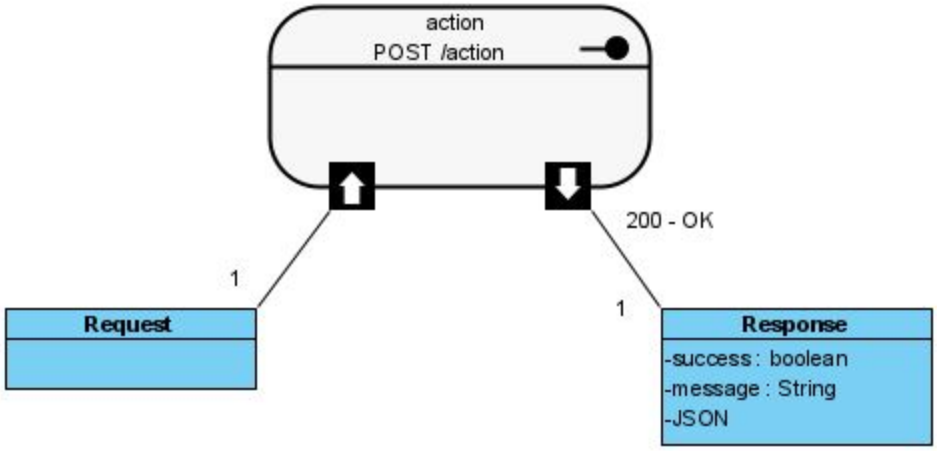

Figure 4 : General Action

Server side of the system is related to the Restful API in order to interact with clients that's why we handle the Restful API with the structure above.

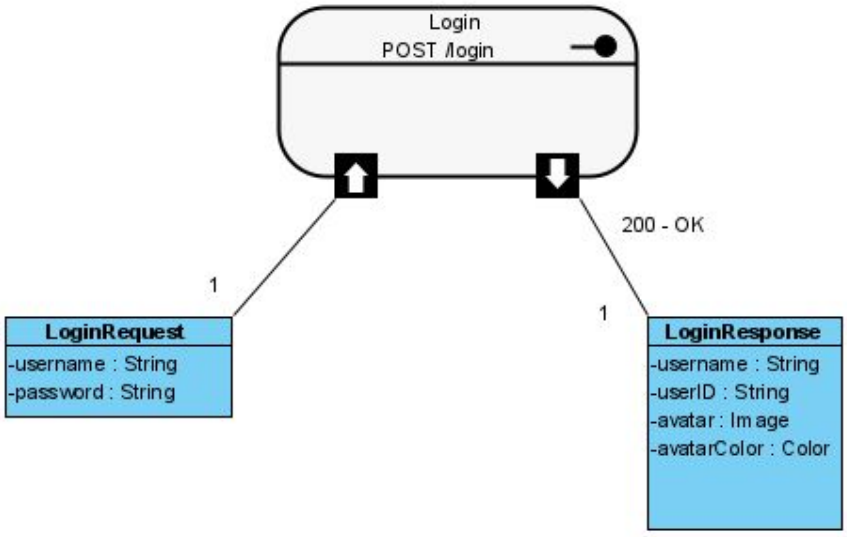

Figure 5 : Login Action

Login: On the server-side, this endpoint checks, whether the username and the password entered into the system match and exist in the database. If it is valid, the user can enter the system and do other actions below. If it is not, return a failure message and reset the process.

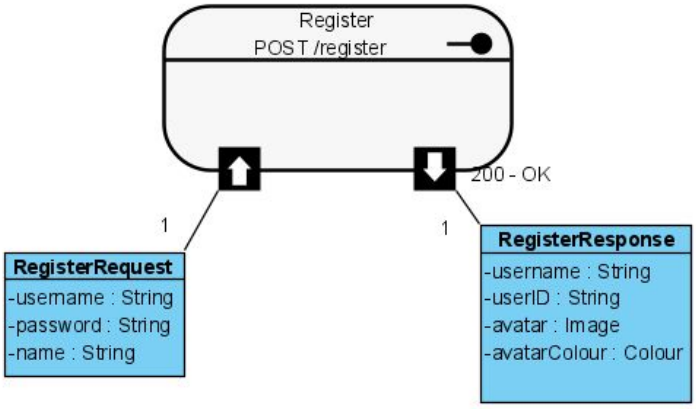

Figure 6 : Register Action

Register: On the server-side, this endpoint adds a new user to the database and to add the user successfully username has to be unique. If it is unique, the server returns a response with a User object, otherwise returns an error message with a flag.

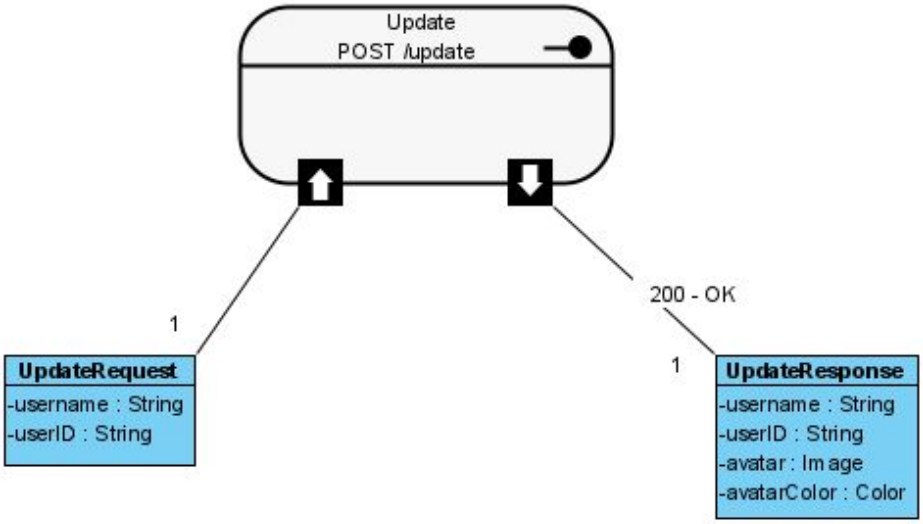

Figure 7 : Update Action

Update: On the server-side, this endpoint updates details about the user and there is a problem with it, returns a failure message or it is not, returns a success message.

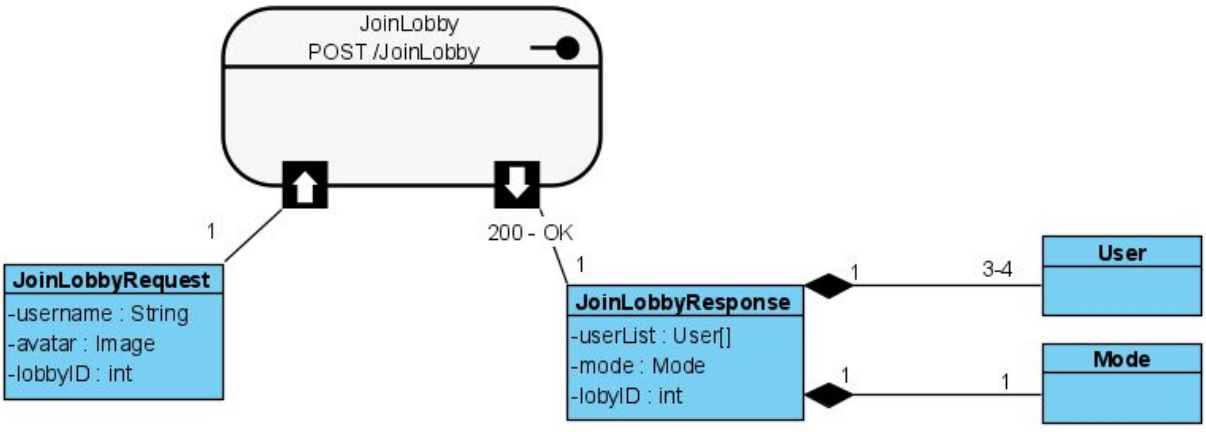

Figure 8 : Join Lobby Action

JoinLobby: On the server-side, this endpoint checks whether lobby code exists and the room is full or not. If the code is wrong or the lobby is full, the server returns a flag with an error message, if not it returns the lobby information to the user.

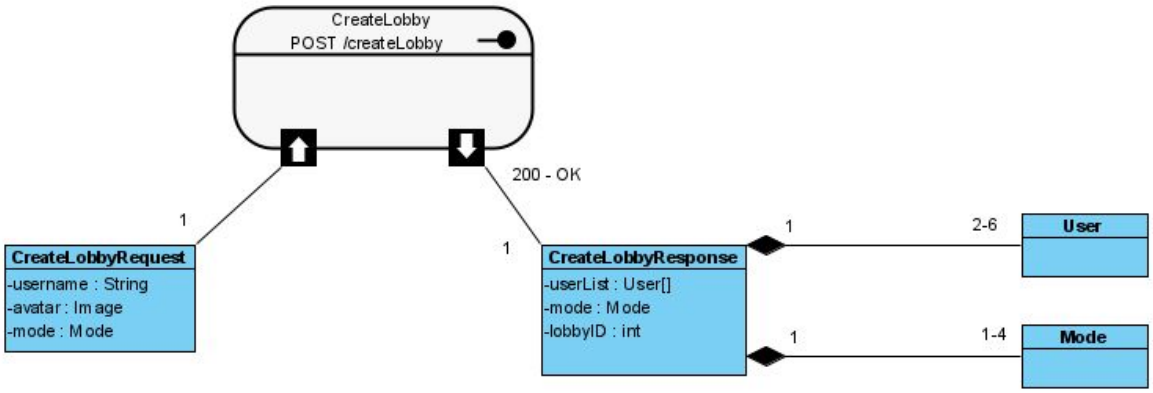

Figure 9 : Create Lobby Action

Create Lobby: On the server-side, this endpoint allows the user to create a lobby with the modes he/she wants, if there is no problem with request-side.

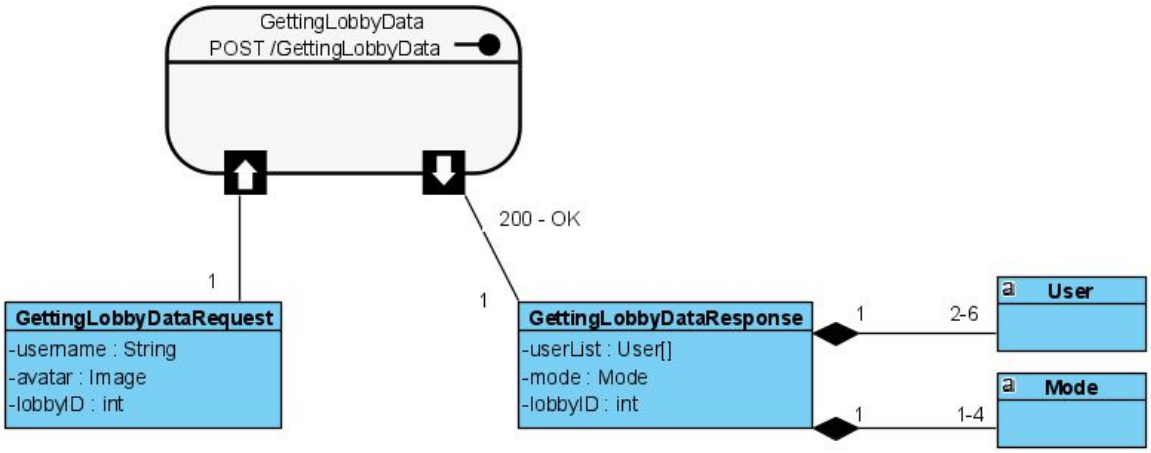

Figure 10 :Getting Lobby Data Action

Getting Lobby Data: On the server-side, this endpoint sends the game data to the user who wants to reach the data.

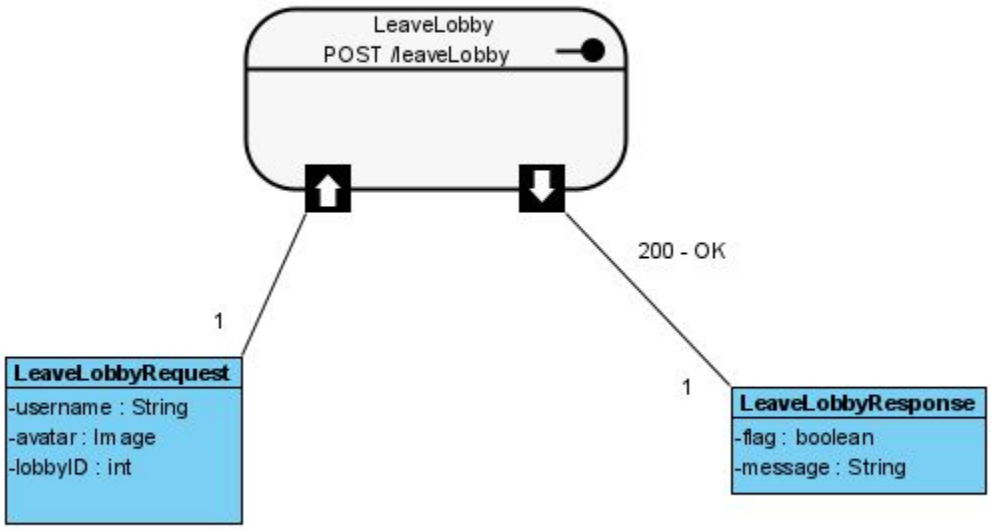

Figure 11 : Leave Lobby Action

Leave Lobby: On the server-side, this endpoint lets the user to successfully leave the lobby.

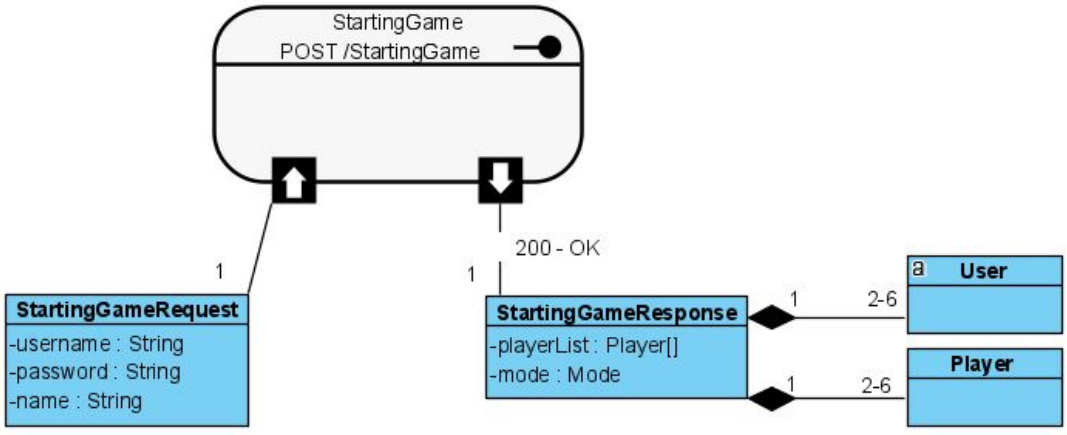

Figure 12 :Starting Game Action

Starting Game: On the server-side, this endpoint checks if the game is ready to start. For this, the game's only requirement is the number of players, if there are at least two users, the game will successfully start. Otherwise, the game will not be started and an error message will be sent.

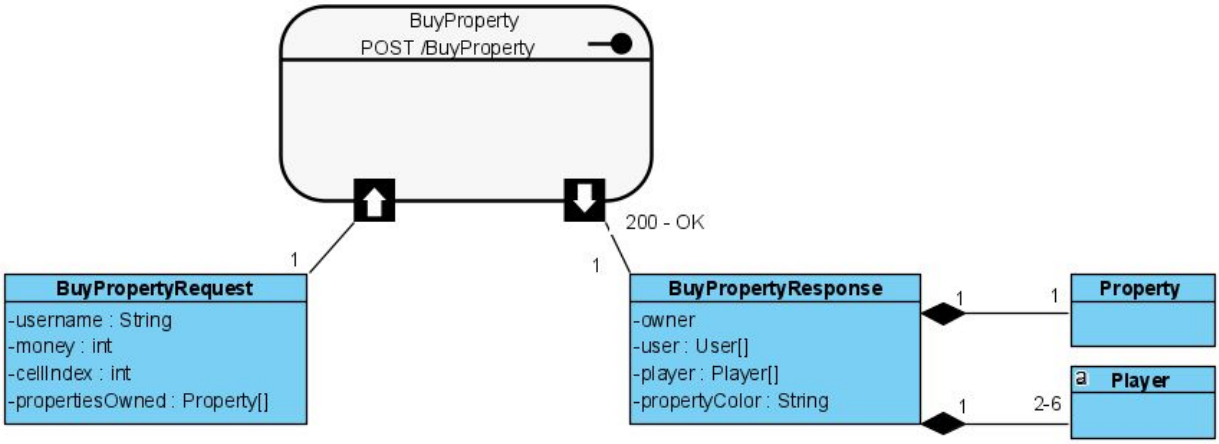

Figure 13 : Buy Property Action

Buy Property: On the server-side, this endpoint is related to buying property. Checks date of the user who wants to buy a property. If there is no problem, it returns a success message.

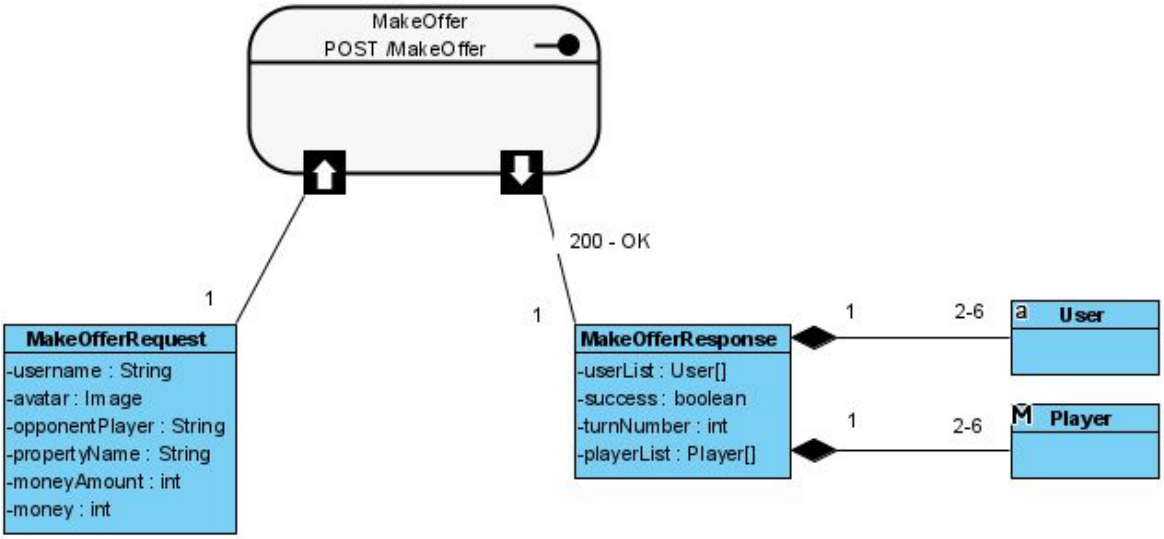

Figure 14 : Make an Offer Action

Make Offer: On the server-side, this endpoint is related with making an offer to opponent players. Checks data of the user who wants to make an offer to an opponent player. If there is no problem, it returns a success message.

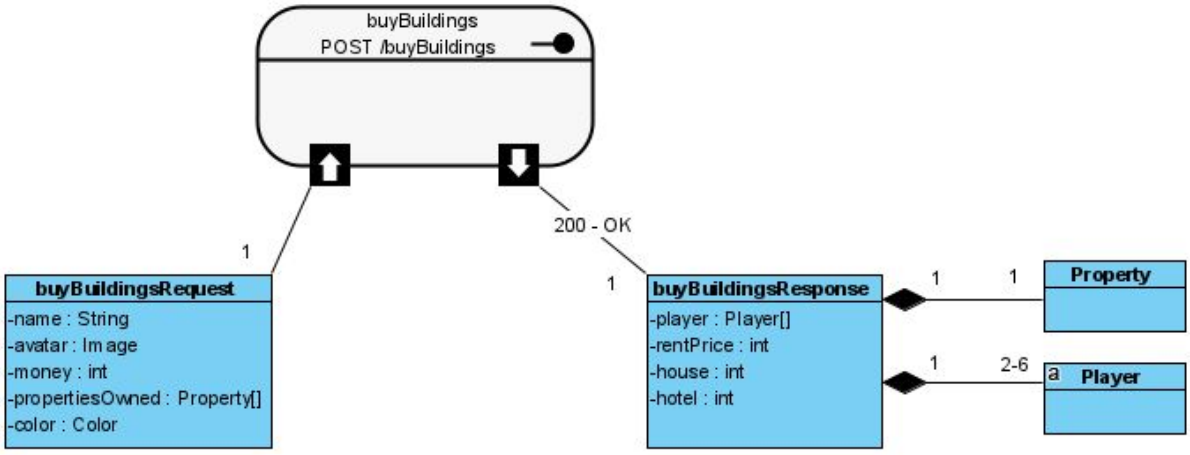

Figure 15 : Buy Buildings Action

Buy Buildings: On the server-side, this endpoint is related with buying homes and hotels. Checks data of the user who wants to buy a home/hotel and if there is no problem, it returns a success message.

3.2.1.1. Models Component

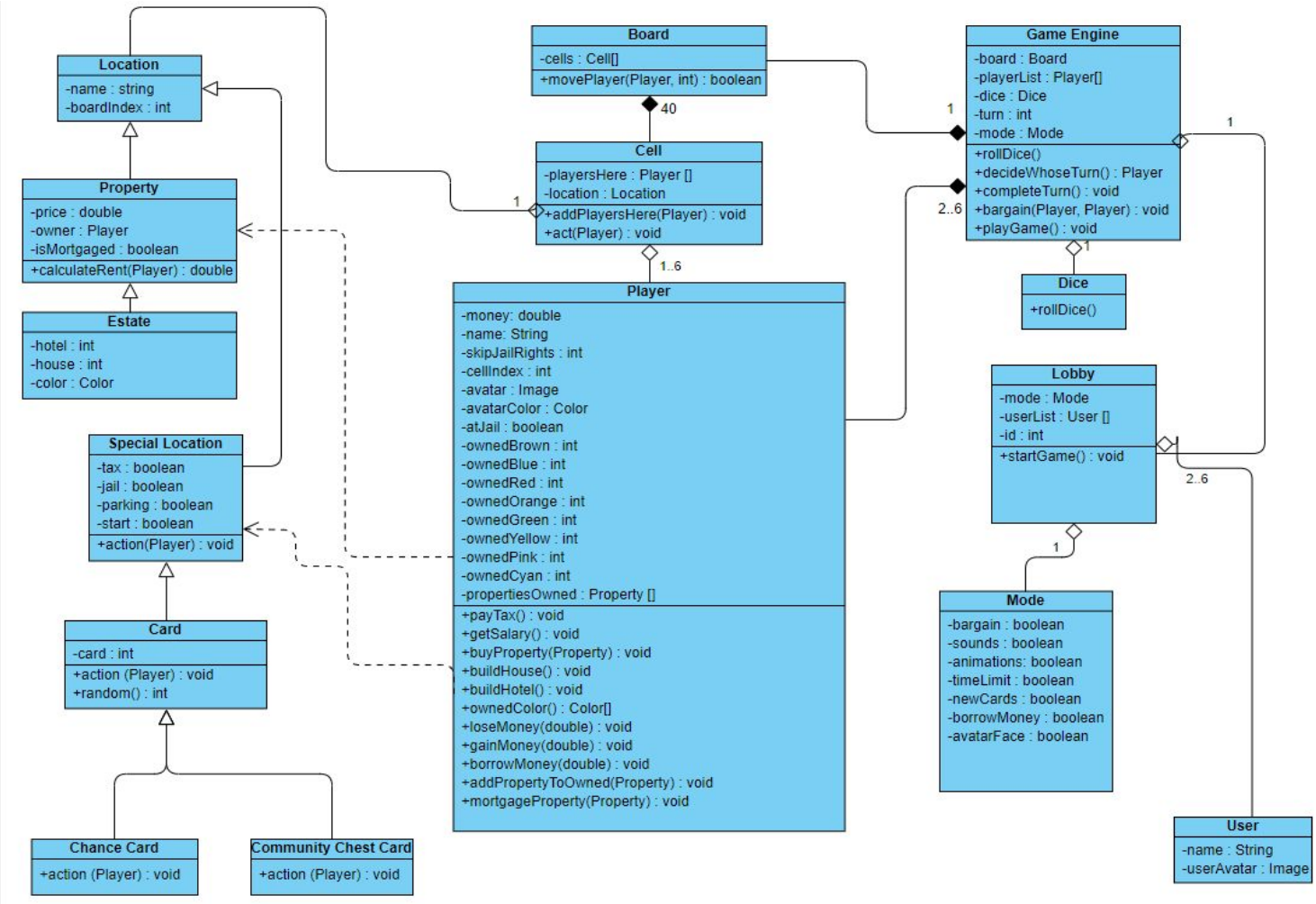

Figure 16 : Models Class Diagram of the server side

Player: Represents a player of classical Monopoly.
-money savings of the Player
-name name of the Player
-skipJailRights free pass credits for jail
-cellIndex current location of the Player
-avatar an avatar of a player
-avatarColor color of the avatar
-atJail flag for the player if he is at jail
-ownedColors (general explanation) counters of the colors that Player has
-propertiesOwned a collection of what Player has as properties
+payTax() Player pays the tax if he visits a tax location
+getSalary() Player gets the salary if he visits the start location
+buyProperty(Property) Player buys a property calls addLocationToOwned

+buildHouse(), +buildHotel() Player builds houses or a hotel to an estate
+ownedColor() Returns the colors of the completed color collections e.g. when three yellow estates is owned yellow will be added to return list
+loseMoney(double) is called when Player loses money
+gainMoney(double) is called when Player gains money
+borrowMoney(double) Player take loans
+addPropertyToOwned(Property) is called in the buy property method
+mortgageProperty(Property) Player mortgages a property

Cell: A data structure class that holds locations and etc.
-location Cell holds a location
-playersHere current players who are actually here
+addPlayersHere(Player) add a player here

Board: A game board that hold cells
-cells 40 cells
+movePlayer(Player, int) moves the player in board

Location: A class which is the parent class of the other location types in the game.
-name Holds the name of the location
-boardindex Starting from the “start” location this holds the value of the location relative to the board.

Property: A class that holds buyable locations.
-price Price of the property
-owner Which player owns the property.
-isMortgaged Current mortgage situation
+calculateRent(Player) this will calculate the rent for the Player to pay. This method will call Player’s loseMoney()

Special Location: A class that holds the other locations that are not buyable.
-tax True if Player needs to pay tax.
-jail True if this is jail.
-parking Also called “mayfest”. True if Player is on parking.
-start True if Player earns money.
+action(Player) Changes the attributes of a given Player accordingly.

Estate: A class that holds buyable locations.
-hotel Shows if there is a hotel or not.
-color Color of the estate.
-house Number of houses

Card: A class that holds cards.
-card Index of the exact card

+random() Returns a random int value in the range of card number
+action(Player) Changes the attributes of a given Player accordingly.

Chance Card: A class that holds cards.
+action(Player) Changes the attributes of a given Player accordingly.

Community Chest Card: A class that holds cards.
+action(Player) Changes the attributes of a given Player accordingly.

Game Engine: A class that has the loop that makes the game continue.
-board Holds a Board object
-playerList [] Holds the players
-dice Holds a Dice object
-turn Number of the turns played
-mode Holds a Mode object
+rollDice() this method will roll dice.
+decideWhoseTurn() returns the Player who has the turn.
+completeTurn() this method will complete the actions after the turn has been played.
+bargain(Player, Player) this method will do the actions of a bargain and change the attributes accordingly.
+playGame() this method is the main loop.

Dice: This class is the dice class.
+rollDice() this method will roll dice that will return the summation of two dices that have been thrown randomly

Lobby: Concerned with the actions with the creation of online lobbies.
-mode Holds a mode object.
-userLİst [] This holds the Users in a game lobby
+startGame() This function will start the game.

Mode: A class that keeps track of the game rules that have been chosen
-bargain Rule of the game
-sounds Change if sounds are present
-animations Change if animations are present
-timeLimit Change if there is a time limit for the game
-newCards Change if new cards are present
-borrowMoney Change if players can borrow money from the bank
-avatarFace Change if avatars with faces can be selected

User: This is a player, but before starting a game.
-name The name of the User
-userAvatar Photo chosen by a User to be displayed as an avatar

3.2.2. Client Side

3.2.2.1. UI Component

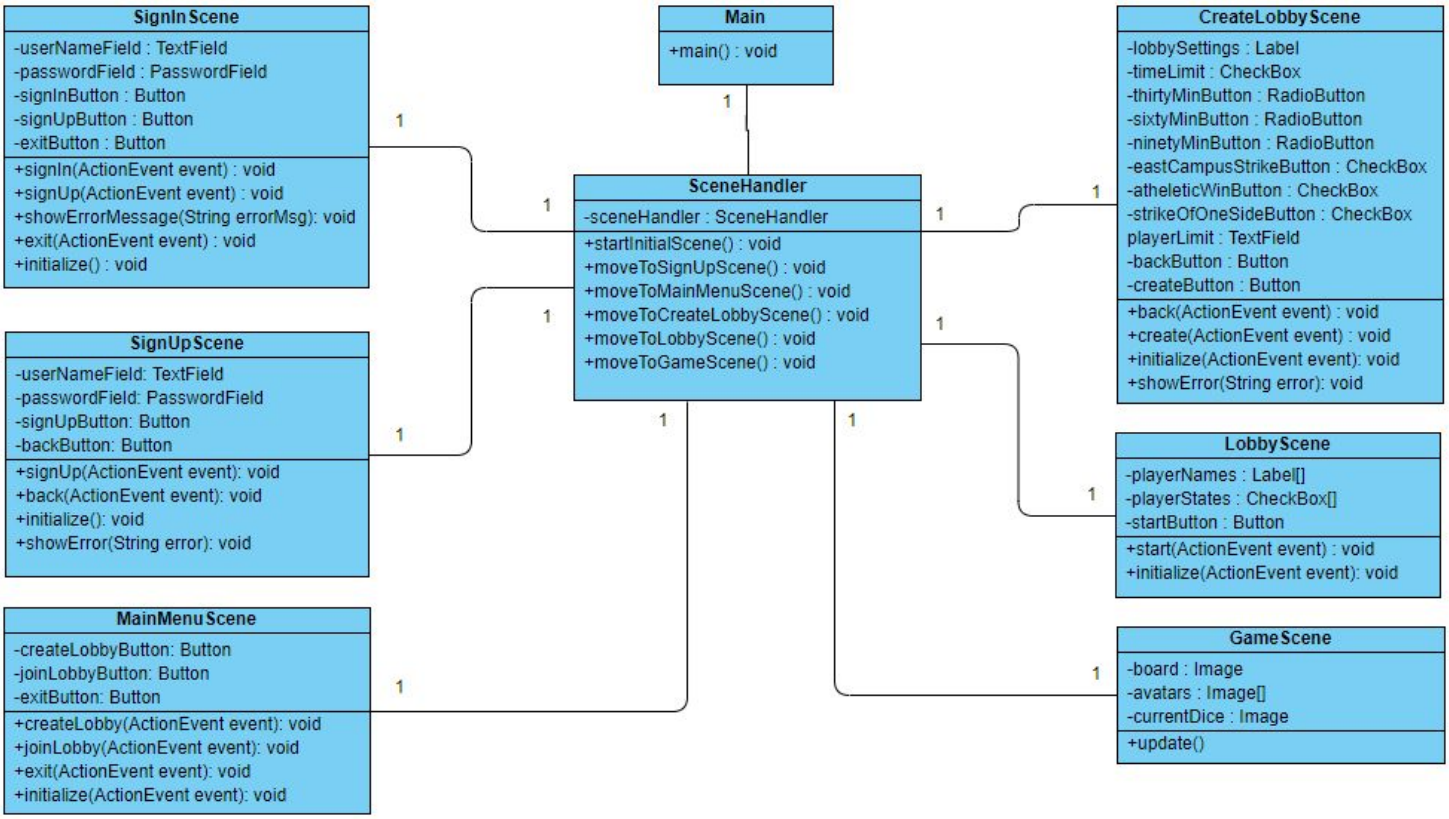

Figure 17 : UI Component Client Side

SignInScene:
userNameField: Where user enters their username to sign in
passwordField: Where user enters their password to sign in
signInButton: Button to sign in
signUpButton: Button to sign up instead of sign in
exitButton: Button to exit the game

SignUpScene:
userNameField: Where user enters their username to sign up
passwordField: Where user enters their password to sign up
signUpButton: Button to sign up
backButton: Button to go to previous scene
MainMenuScene:
createLobbyButton: Button to create new lobby
joinLobbyButton: Button to join an existing lobby
exitButton: Button to exit the game

CreateLobbyScene:
lobbySettings: Label including “Lobby Settings”
timeLimit: Checkbox in order to enable following radio buttons
thirtyMinButton: 30 min time limit choice
sixtyMinButton: 60 min time limit choice
ninetyMinButton: 90 min time limit choice
eastCampusStrikeButton: a checkbox to activate the mod that decides the winner when he owns all east properties
athleticWinButton: a checkbox to activate the mod that decides the winner when he owns all he owns all sport related properties
strikeOfOneSideButton: a checkbox to activate the mod that decides the winner when he owns all properties in a corridor
playerLimit: a text field where user enters the maximum players of the game
backButton: a button to go to previous scene
createButton: a button to create the lobby

LobbyScene:
playerNames: Label array to show “Player x” etc.
playerStates: CheckBox to state players as ready
startButton: a button to initialize the game

GameScene:
board: an image to initialize the board
avatars: image array to hold avatar of the players
currentDice: current dice image to show the roll results “::: .” or “:: :.” like this 6 1 or 4 3

4. References

1- Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java