# Introduction

Clients perform actions and receive events. Actions can update the Lobby Model which typically causes the server to emit one or more events. Actions do not have a return value as such. Failed or non-permitted actions will cause a ‘ActionRejected’ event to get emitted.

The goal is to keep the local client Lobby Model synchronized with the remote server Lobby Model.

# Implementation Details

## Lobby Model

### Layout

players{

(namestring): {

name: (namestring),

game: string

}

},

games{

(namestring): {

gamename: (namestring),

password: string,

ownername: namestring,

players: [namestring, …],

invited: [namestring, …],

maxplayers: int

}

}

### Methods

ICollection<Player> Players

ICollection<Game> Games

AddPlayer(string name): Player

RemovePlayer(string name)

CreateGame(string gName, string pName, string password, int maxplayers): Game

GetGame(string gName): Game

GetGameByPlayer(string pName): Game

PlayerHasGame(string pName): bool

HasPlayer(string pName): bool

HasGame(string name): bool

## Password Database

Table Users:

* playername
* email
* salt
* hash
* iterations (of hashing algorithm)
* status {pending, verfied}

Table Temporary

* username
* hash
* datetime

Table Sessions

* username
* hash
* expire

## Server Operations

Server actions and the possible resulting client actions.

All actions other than Login will perform a ‘is-logged-in’ check. A failed check will result in the client receiving: AuthError {}.

For each operation the server endpoints are listed under ‘Server’, the possible responses are listed under one of: ‘Client’, ‘3rd Party’, ‘Global’.

Client responses get sent to the requesting client only. 3rd Party responses get sent to a single client that is not the requesting party. Global responses get sent to all connected clients.

### Login

Effects:

* Logs a player in to the server.
* Assigns the player a session hash they can use for authentication.
* The session becomes invalid after a preset amount of time.

Reasons for rejection:

* Player already logged in.
* Player name is invalid.
* Player name is unknown.
* Password is incorrect.

Server:

Login{playername: string, password: string}

LoginSession{hash: string}

Client:

LoginRejected{reason: string}

LoginAccepted{hash: string}

Global:

PlayerLogin{playername: string}

### Logout

Effects:

* Logs a player out of the server.
* Unassigns their session hash.

Reasons for rejection:

* The player is in a game (call leave game / cancel game first).

Server:

Logout {}

Client:

LogoutRejected{reason: string}

LogoutAccepted{}

Global:

LeaveLobby{playername: string}

### Join Game

Effects:

* Player joins a game if they are not already in a game.
* Adds the player to the game. (TODO TEST)
* Adds the game to the player. (TODO TEST)

Reasons for rejection:

* The player is already in a game (leave game / cancel game first).
* The password does not match or was not provided & not invited. (TODO test invited)
* The game is full.
* The game no longer exists.

Server:

JoinGame {gamename: string, password?: string}

Client:

JoinRejected{reason: string}

JoinAccepted{gamename: string}

Global:

PlayerJoined{gamename: string, playername: string}

### Create Game

Effects:

* Creates a new game and adds a player to it.
* Adds the player to the game.
* Adds the game to the player.

Reasons for rejection:

* The player is already in a game (call leave game / cancel game first).
* A game with the given name already exists.
* The name fails a name check (spaces, letters, number only).

Server:

CreateGame {gamename: string, maxplayers: int, password?: string}

Client:

CreateRejected{reason: string}

CreateAccepted{gamename: string}

Global:

~~NewGame{gamename: string}~~

~~PlayerJoined{gamename: string, playername: string}~~

NewGame{game: game}

### Leave Game

Effects:

* Removes a player from the game.
* Removes the game from the player.
* If the player is the owner, then the game terminates, and all players are removed.

Reasons for rejection:

* The player is not in a game.

Server:

LeaveGame {}

Client:

LeaveRejected{reason: string}

LeaveAccepted{}

Global:

RemoveGame{gamename: string}

PlayerLeave{gamename: string, playername: string}

### Remove Player

Effects:

* Removes a player from the game.
* Owner removes a player from a game.
* Removes the game from the player.

Reasons for rejection:

* The player is not in the game.
* The owner cannot get kicked.

Server:

KickPlayer {targetplayer: string}

Client:

KickRejected{reason: string}

KickAccepted{}

3rd Party:

KickedFromGame{} # to the client kicked from the game.

Global:

KickPlayer{gamename: string, playername: string}

LeaveLobby{playername: string}

### Start Game

Effects:

* Removes all players from the lobby.
* Players session-hashes remain valid.
* Game get’s removed from the lobby.

Reasons for rejection:

* The player is not in a game.

Server:

StartGame {}

Client:

StartGame{ip: string, port: int}

Global:

RemoveGame{gamename: string}

### Invite Player

Effects:

* Add a player to the game invite list (does not require password).

Reasons for rejection:

* The invited player is not in the lobby.
* The inviting player is not in a game.

Server:

Invite {playername: string}

Client:

InviteAccepted {playername: string}

InviteRejected {playername: string, reason: string}

3rd Party:

Invite{gamename: string}

### Register Player Effects

* Add the player and password hash to the player database.
* Send a confirmation email.

Reasons for rejection:

* The name is already in use
* The email is already in use
* The name is invalid

Server:

RegisterPlayer{name: string, password: string, email: string}

Client:

RegisterAccepted {}

RegisterRejected {reason: string}

### Recover Password

Effects

* Email a temporary password to the user.

Reasons for rejection:

* Unknown email.

Server:

ResetPassword(playername: string)

Client:

ResetAccepted {}

ResetRejected {reason: string}

### Delete Player

Effects:

* Remove a player from the database.
* Player must be logged out.

Reasons for rejection:

* Unknown player.
* Password not matching
* Player is logged in.

Server:

DeletePlayer(playername: string, password: string)

Client:

DeleteAccepted {}

DeleteRejected {reason: string}

### Request Player List

Effects:

* Retrieve a list of all current players.

Reasons for rejection:

Server:

RequestPlayers()

Client:

PlayerList {players: object[]}

### Request Game List

Effects:

* Retrieve a list of all current games.

Reasons for rejection:

Server:

RequestGames()

Client:

GameList {games: object[]}