## Player acces sq HTTP CHANNEL **CLIENT SERVER** Ping phase to the central server. It checks whether the IP, known in advance ACK() by all users, is valid and responsive. The client requests a list of games in the startup phase from the server. {lobbies: [{lobbyld, LobbyInformation,}, ...]} getLobbies() = The server responds with the games and their primary information (who Lobbies[] started the game, the required number of players, how many (and who) are present, the flight level...). It could also return an empty list. The client opens a WebSocket and immediately sends a message containing the request to connect to a specific game. OPEN WEBSOCKET = {lobbyld, username, requestId} The server first checks if the lobby exists. If the lobby does NOT exist (or exists but has already started, meaning it is not in the waiting lobby pool), a new lobby is created (which may also have a different ID). A success response is returned with the default information of the new lobby. The player is considered the initiator of the lobby. {requestId: uuid, success: true, LobbyInformation}

## Get a component sq

the lobby information.

messages.

If the lobby exists and is waiting, the server checks that the username

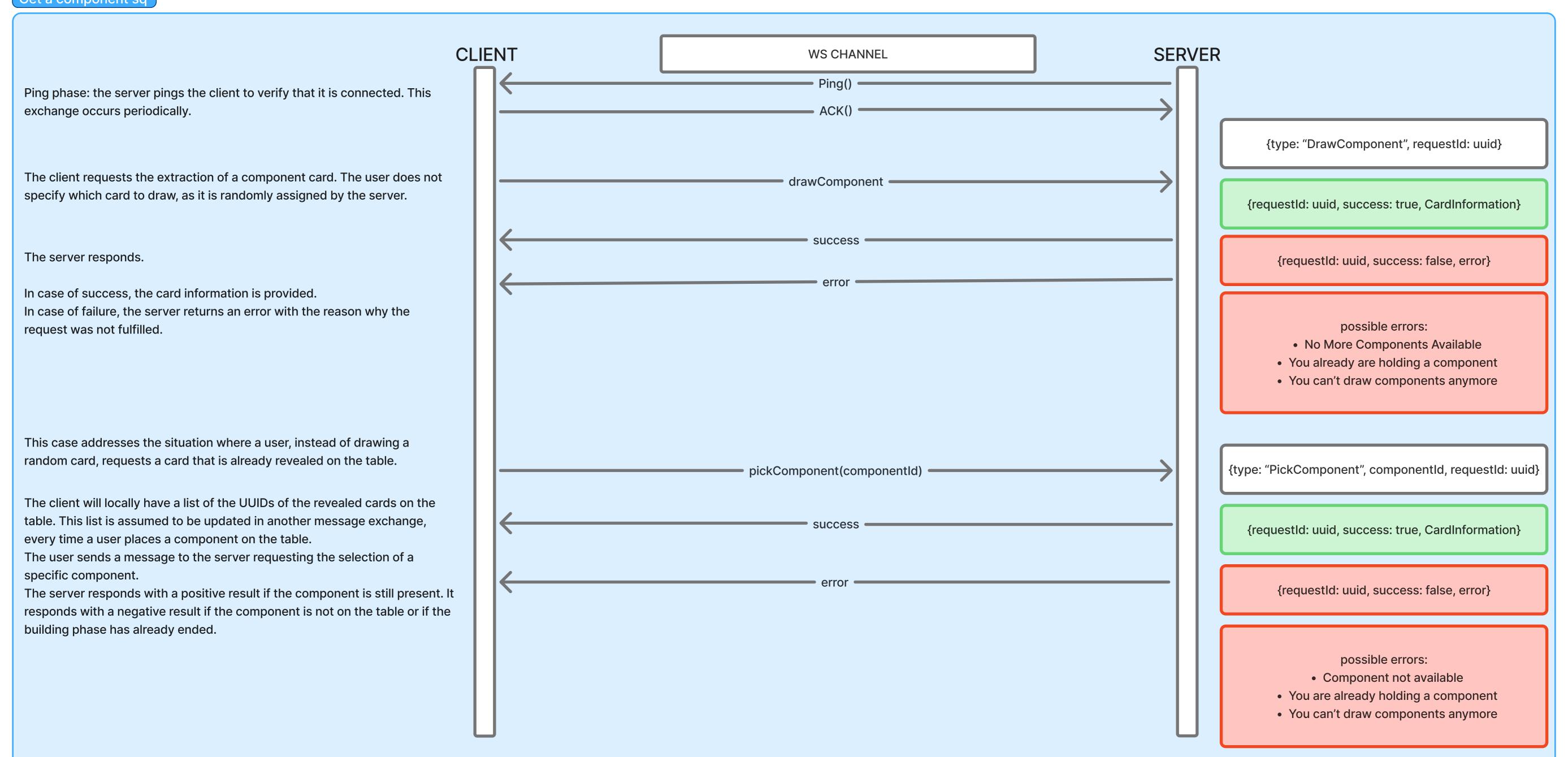
If the username is a duplicate, the WebSocket is closed, and the user is

redirected to the game list. Otherwise, a success message is returned with

This WebSocket will receive messages with lobby updates (e.g., game level

changes, another player quitting, or game start) as well as game-related

entered by the user has not already been used in the lobby.



WebSocket ACK() \_\_\_\_\_

{requestId: uuid, success: false, error}

possible errors:

Username already in use

## Attach a component sq

