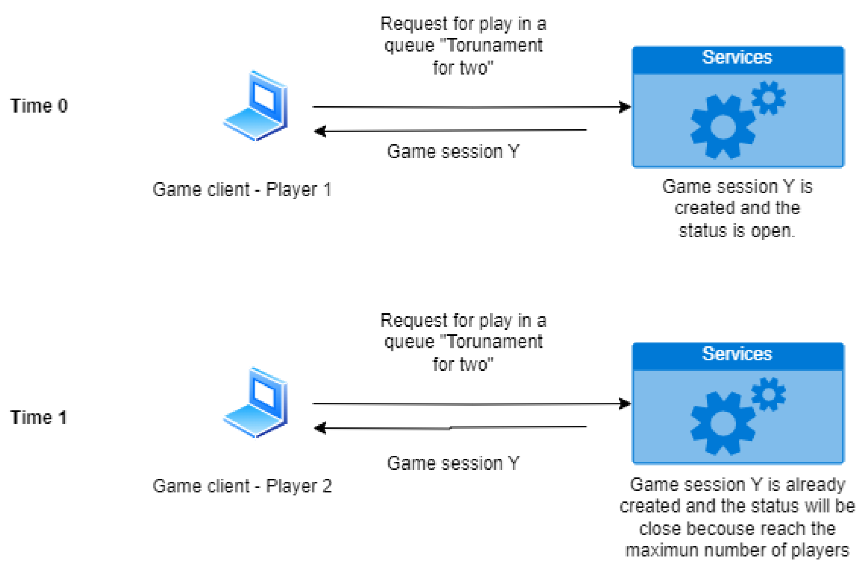
Problem statement

We have already created a video game called "Survivor," which is a battle royale shooter. The game is based on tournaments, and the players need to select a queue to battle each other. You need to create a backend system to manage the players and queues that will be part of the game.

The normal flow for the player to start a tournament is to select the queue and play on it; an example of a queue could be "tournament for two", and in this tournament, there should be two players battling each other. Following the same example, when a player requests to play in a queue the first time, it will create a game session, and the game session will be in status opened. After that, if a second player tries to join in the same queue, they will be joined in the game session mentioned, but in this case, the game session will be closed because it will reach the maximum number of players for this queue. When a player is assigned to a game session, this relationship cannot be removed.


Time 0
Request for play in a queue "Torunament for two"
Services
Game session Y
Game client - Player 1
Game session Y is created and the status is open.
Time 1
Request for play in a queue "Torunament for two"
Services
Game session Y
Game client - Player 2
Game session Y is already created and the status will be close becouse reach the maximun number of players

So that we have the following entities:

Players: It will store the name and ID of the player.

Queues: It will store the name and ID of the queue and the maximum capacity of the players. We cannot create more than 10 queues. This number should be configurable.

Game session: It will store the game session ID, queue ID, list of players (which could be in a separate table), and status (opened or closed). When a player attempts to join a queue, it must first determine whether there is an open session related to this queue. If any session is opened, it can be joined, and if it reaches the maximum capacity, the session will be closed. If no open game sessions are found, a new game session will be created and joined. Remember, the system cannot remove the relationship between game sessions and players.

Note: Forget any concurrency issues.

Exercice:

Create a simple API REST with the following methods and verbs:

- players:
    - get all: Return a list of players.
    - get by id: Return a concrete player.
    - create: Return the data of the player and the ID generated.
    - update by id (only name)
    - delete by id
- queues:
    - get all: Returns a list of queues.
    - get by id: Return to the concrete queue.
    - create: Create a queue, please take into account that you can create 10 queues (by a file configuration).
    - update by id (only name)
    - delete by id
- game session:
    - get all: Returns a list of game sessions.
    - create or update (it needs to specify queue ID and player ID): The logic is explained above. Return the ID of the game session and the related data.
    - get by status: Return the list of queues filtered by status.

Specifications:

- The language will be Golang, and you can use Chi/Gin/Fiber as an HTTP web framework.
- Persistence is up to you, you can use both in memory or any SQL database.
- Create a README.md. Indicate how to set up the project.
- Create a unit and/or integration test (at least one of them).
- Use swagger for documentation.
- Create a Dockerfile.
- There are some specifications not mentioned that you need to figure out.
- Send the code in a compressed file.
- Keep it simple 🙂

Recommendations:

- Try to use a Restful naming convention.
- Try to use the HTTP codes.
- Try to respect the Golang style convention.