Client server with simple game play homework:

<u>**Server requirements:**</u>

- must be written in java
- needs to listen on port 8080
- every request must be http post
- the request type should be JSON

<u>Basic idea:</u>
A player must signup using the “signup” command for an account in the system and here is an example:
Request => {"playerName":"player1","password":"pass","command":"signup","uniqueId":"add59117-ee9f-4379-8d41-ae29b6626ce0"}
Response <= {"status":"OK","uniqueId":"add59117-ee9f-4379-8d41-ae29b6626ce0"}

Note: If already there exists a player with the given playerName an error must be returned.

After the player is signed up he can login using the “login” command like this:

Request => {"playerName":"player1","password":"pass","command":"login","uniqueId":"36cb9450-04b9-4656-95aa-1727be804d8b"}
Response <= {"token":"656461fe-767d-4e29-9720-f273f8235dd7","status":"OK","uniqueId":"36cb9450-04b9-4656-95aa-1727be804d8b"}

Note: If the pass is wrong or there is no such player an error must be returned.
Note: If the login is successful a valid token must be returned and for each operation that requires credentials it must be used: In this example the token is "656461fe-767d-4e29-9720-f273f8235dd7".

Now the player adds some money to his account using the “deposit” command:
Request => {"amount":20.00,"token":"656461fe-767d-4e29-9720-f273f8235dd7","command":"deposit","uniqueId":"08fe6af5-4e7f-4997-84c3-e97c4b91e649"}
Response <= {"status":"OK","uniqueId":"08fe6af5-4e7f-4997-84c3-e97c4b91e649"}

Note: The same way a withdraw must be supported using the “withdraw” command. If the player withdraw amount exceeds his current balance an error must be returned.

Now a simple bet is made using the “bet” command:
Request =>
{"amount":2.50,"gameType":"dice","additionalData":"[{\"u\":7,\"v\":1.50},{\"u\":6,\"v\":1.00}]","command":"bet","uniqueId":"15674256-7100-4734-9f75-01978610855f"}

Note: Explanation of the bet request:

- gameType points to a specific game to witch this bet is made.
- AdditionalData describes specifics of the bet. In this case for possible winning number 7 the bet is 1.50 and for the possible winning number 6 the bet is 1.00.

The response contains wining information for this player:
Response <= {"winAmount":1.73,"outcome":"7","status":"OK","uniqueId":"bdd97674-0977-4fb0-ad08-5760eeca595f"}

Note: The application must use mysql database that keeps track of the following:
- player information (player name and pass)
- all his payments (deposits and withdraws)
- his current balance
- all bets and wins

Note: The server must be designed to handle **concurrent** player operations. This means that if the player made bet and deposit operations at the same time his balance should continue to be consistent.

Dice game definition:

- A player makes bet for a given number in the range [2 .. 12]
- For each number an amount must be placed
- Random number generator generates a number between [2 .. 12] simulating throw of two dices.
- A winning number is returned and if the player has a bet for that number his winning is the betAmount * (1.00 + coefficient). The database must keep configuration for the coefficient for each number. In the example above the coefficient for the number 7 is 0.15 so the winning is 1.50 * (1.00 + 0.15) = 1.50 * 1.15 = 1.73.

Note: We expect proper test cases with a java http client.