Muhammad Soliman

Game Service

m.solimanz@hotmail.com

Assumptions:

This document interprets what I understand from the requirements are coming up in the last page of this document.

However the requirements were not clear enough, like in parts awards and game statistics there were no use cases describing the access patterns or scenarios in which we are going to use or develop as this affects on design decisions or structure may be even the technology stack I might choose.

For example, I added game statistics to the same table of users however if we want to get top 5 players according to those statistics (wins/losses/deaths/kills) we should separate them out to a different table and calculate a rank then sort and partition using this rank. At the end we can get the top N players as easy and highly available and performant we can.

What I meant from the first words I gave here is that design might be hugely changed according to the requirements. I am very open for more discussion to know more about my thoughts of high availability and scalability.

As I understand this service should handle high workloads so I designed the service to be backed by “MongoDB” however there are other solutions like Cassandra or HBase. Still the requirements here are the key to choose which one. For example if we need highly available solution without caring about consistency we can choose Cassandra meanwhile if we care about transactions and strong consistency we can choose HBase.

This service/code is just a simple demo which is not completed, even its requirements are not clear, I hope evaluation should cover all aspects of **coding, designing, and especially documentation.**

Accomplishments

Personal Info and **Full Profile Info** should be returned separately, that is why you might find I put personal info, game stats, awards and all matches the user played before in “**Users**” collection. Then I exposed 2 services one to get all profile information and the other to get only personal information (the default)

<u>**For users**</u>

- To bring all users (personal info/full profile)
- To bring users by id (personal info/full profile)
- To bring the current match that he/she is playing now
- To list all matches that user were playing before.
- To bring only stats for users
-

Game Service

m.solimanz@hotmail.com

For matches:

- Ability to create anew match
- Ability to add players into the a running match.
- To fetch all active/running matches that are currently running now along with their players

- To bring all matches along with players info.

Calculated Fields (e.g. playersCount):

Some times it helps determining for example how many players into the match and whether it is the maximum number. Instead of get the array of players and calculating the length of it, I preferred to keep an atomic counter outside to get it quickly.

For Game Stats:

- If we need to bring Top N players
- Either we can separate to different table and keep a rank as calculated field calculated based on wins/kills/deaths/losses
- Put the **rank** in the same table as users and do the same (indexing should be there) - implement ranking functionality based on different stats criteria.

Indexing:

Compound Indexing in NoSQL: As of my understandings, it supports queries that involve 2 or more fields in the compound index as well as queries that involve **only the prefix** of the compound index. (**Start of the Hash Key**) that is why I did 2 indexes to support logins either by userName or email (e.g. getUserByUserNameAndPassword), also it supports querying only by userName (prefix) or email (start of compound key) to expose for example service named "getUserByUserName" or "getUserByEmail"

- The same concept in other NoSQL DBs like HBase or DynamoDB.

More about DB optimizations in **my blog** http://msoliman.me/2017/07/02/5-essential-key-concepts-databases-optimizations/

Test Suites:

Can be found under each collection/domain

Configuration:
Multiple Environments Support, e.g. Development, Staging, and Production

Documentation:

Game Service

m.solimanz@hotmail.com

Most of code is documented to help understand some the way of the methodology used there.

What is missing:

Passwords should be hashed pre-saving into the "Users" collection using any cryptography library (e.g. bcrypt) and all subsequent queries should be executed using the same hashing functionality)

Game Collections: to have multiple games, categories and tags of those games, and interests of each user. Based on their interests, recommendation libraries can be used to analyze and recommend for them games they might like as Ads.

- **Likes** array would be in "Users" collections at this time and of type ["Game"]

Game Service

m.solimanz@hotmail.com

Initial Requirements:

We'd like to see how you would design a service that allows a developer to create, update, and modify user records. User records must include at least the following information:

- Personal Details: Personal details would consist of information that would be tied directly to a user. This should take into account information such as a player's username and email.
- Game Statistics: This would be non-personal information about a player, based on their gameplay. Examples would include kills, deaths, wins, losses, etc.
- Achievements: Awards that a player earns after meeting a pre-determined criteria created by the game developer. e.g. win 5 matches
- Matches: Gameplay sessions that occur between 2 or more players. Examples of a Match could be anything from two players competing a full game of chess, or 16 players competing in a team-based shooter.

Your response should include:
- A document detailing your interpretation of the prompt and assumptions that you have made
- (Pseudo) Code detailing the API you would expose and data models you would use. Any technology(s) can be used here.
- Partial implementations (CLI, REST API, test suites, etc). A major part of our assessment is a review of your actual written code, so it is important to show us a good portion of your application implemented.
- Any other information work you'd like to share.

Previous submissions have additionally included (all optional):
- Database Schemas
- Recommended Tech Stack
- Possible deployment architecture