High-Level Design for Chess Website

By Shreyansh Goyal

How would you design a large-scale distributed chess website? Matching users, minimizing latency, stopping cheating, and game management are explained here.

System Requirements

Functional requirements:

We'll focus on the following requirements to design a chess game server. The system should have:

1. Support for two online players to play a game of chess.
2. A matching algorithm to match the peers
3. A game engine where: a. Match requests with an interval of min and max rating for opponents. b. Game info is stored. c. One side is assigned white and the other black.
4. A chat application.
5. Move validation.
6. Both players will play their moves one after the other. The white side plays the first move.
7. Players can't cancel or roll back moves.
8. Log of all moves made by either player.
9. A game termination state. The game can finish either in a checkmate from one side, forfeit or stalemate (a draw), or resignation. 10.Checks for cheating by players (making the same moves as an engine).

Non-Functional requirements:

1. Low Latency

2. Low Bandwidth

3. Fault-tolerant

4. The system should be consistent

Capacity Estimation:

1. How many users connect at a time? Let us assume 1M DAU(Daily Active Users) Active users/ min = 1M/(24*60) = 900 users/ min Let us assume that, on average, a game lasts for 5 mins. Then we can assume a load of 5000 connected users. In place of any event/ peak, we can consider this value equal to 20K connected users.

2. Capacity to be computed by matching engine If we assume each request to have a size of 1KB, we can have total memory to be 20K * 1KB = 20MB We are using a Balanced BST to find O(logN) matches. That's 20K*log(20K) = 300K instructions.

Design Process :

1. *Matching Engine* — a. Close-rated players should play each other. b. Matching engine service can have a cache of all challenge requests with max and min rating, maintained in a TreeSet or SortedSet. This data structure makes search requests fast. c. Store all the requests in a memory with an example for TTL(Time To Live) of 30 seconds. d. An SQL Database is used here to index time and rating ranges.

2. *Analysis Engine* — a. Batch Processing of games, run with engines like Stockfish. b. Look at the "Workflow Management Platform" for more details.

3. *Game Engine* — a. Bidirectional connection to accept the move and then make a note of it. For this Bi-directional connection, we use WebSockets.

Can we avoid the connection between client and server?

1. We can't do that because we use the game engine to validate moves.

2. The client code or requests may be compromised. Hence it cannot be used as a source of truth.

3. To validate the move, we store the state of the game and check if it's legal on the server.

Classes :

1. Game Engine
2. Matching Service
3. Chat Service

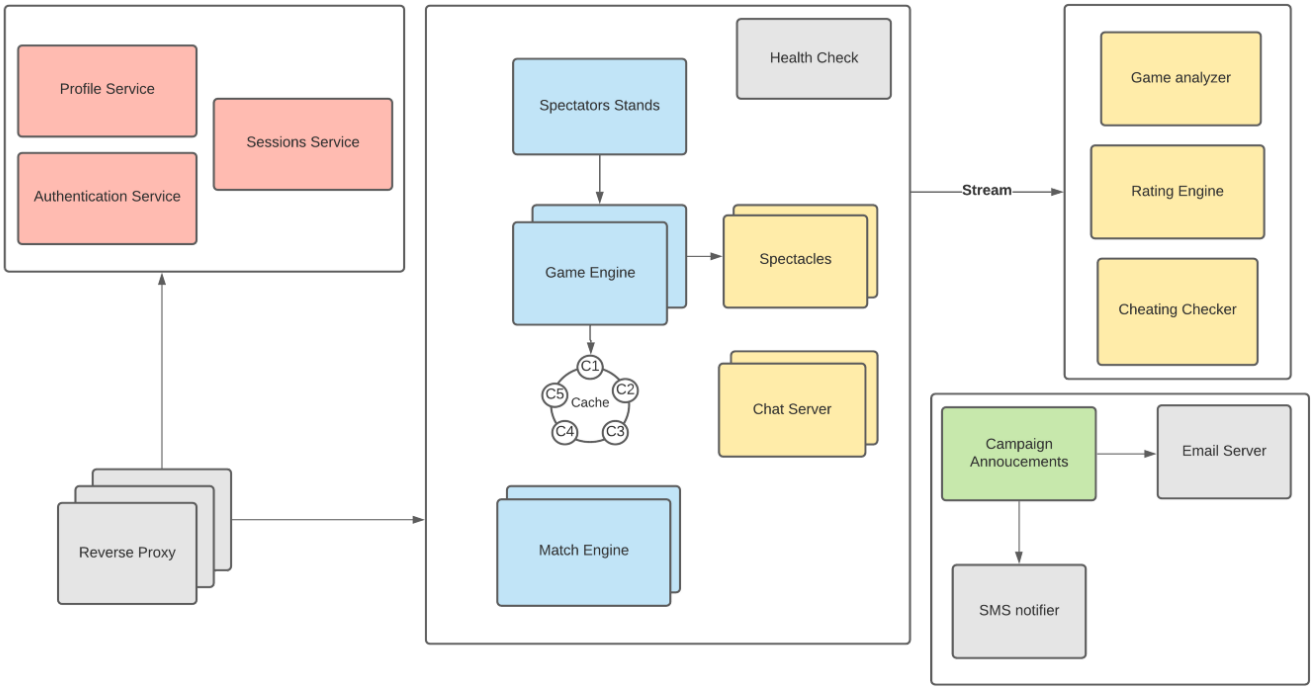
Scaling :

1. The system shards requests based on request IDs.
2. The system caches user and game information.

APIs:

1. makeMove(gameId, userId, Move)
2. getGame(gameId, userId)
3. createChallenge(userId, challenge)
4. sendMessage(gameId, userId, message)
5. getMessages(gameId, userId)

Architectural Design :



That’s it for now!