GameServer implementation Brief:

The GameServer is divided into four project namely dbservice, networkservice, eventdispatcher and gameserver. “networkservice” has dependency on “eventdispatcher” and “gameserver” has dependency on all other services.

dbservice provide an interface “DatabaseService” to read/write/update the model data into db. It has two implementation classes; each of Sql and NoSql databases. For this project, we can keep all identity information in MySQL database and for all analytics and metrics information can use Elastic search. I have partially implemented the MySqL implementation and used spring-hibernate framework and created Model, DAO classes and configured beans in application-context file. As MySql manages provides concurrency and synchronization we need not to take care of thread management and synchronization part.

networkservice provides two interfaces; “ClientConnection” and “ConnectionManager” for communicating with particular client and handling client connections respectively. For querying gameserver for games, players and other information the server can expose REST end-points. For playing game as we need bi-directional communication between client and server, websockets would be a good option for communication protocol. For chat, we can either use websockets or xmpp as communication protocol. I have partially implemented the server endpoints and callbacks. The server exposes an endpoint “/gameServer” and creates events on connection-open, connection-close and message-receive and submit it to the eventdispatcher. Eventdispatcher in turn dispatches the events to all registered listeners. It uses the Executor thread-pool framework for dispatching events. For chatting it exposed different endpoints one for each table; “/gameserver/{table-id}”. I have not put any validation on chat messages and clients w.r.t table and just broadcast all chats to all clients connected on the endpoint.

The gameserver implements the game logic as specified in requirements. Following classes are implemented to provided separate functionality.

ClientSession and SessionManager: ClinetSession represents a client session and SessionManager manages all client sessions and events. The SessionManager listens for ClientConnected, ClientDisconnected, MessageReceived Events and a session object is created/destroyed every time a client connects/disconnects. Every incoming message is submitted to TaskScheduler to process by the respective ClientSession; ClientSession parses the message and based on message type it does the actual processing.

PlayerManager: PlayerManager maintains all online players and provide functionaly to signup, login, logout of players. it also handles the ClientDisconnected event and unload the player information on client disconnection.

Table and TableManager: Table represents a virtual table for players to play a game. It manages the list of players who have joined the table and criterion for starting game. TableManager managers all existing table. Every time a player Join/Leaver the tabke it sends messages to other player on that table. It listen for ClientDisconnected event and update the table and send message to other players.

GameManager: this class keep map of all the games and provide control for start/end game and scheduling all game events. I have created a GameModel class to keep game information which can be updated to keep more information like game rules and game state.

There are classes to define various message-types which are not implemented and just defined to represent message type.

TODO:

1. Caching Interface. Currently I have kept created the data structures in server main memory; however, we need to keep all game and players information in some shared in-memory cache so that we can scale the solution to run multiple servers and maintain consistency b/w the server. We can use Redis for this not sure if there are other options.
2. For memory optimization, we can create Object-Poll of various class objects so that it eliminated the overhead of creating-deleting objects same way like ThreadPoll.
3. Rest Interface to provide stateless information to clients without actual connecting and join some table/game.