Faraaz Mohsin
Next Inc.
Pre-Assessment for JavaScript Full-Stack Developer Position

Part 2: Follow-Up Questions

Explain the architecture of your WebSocket server and RESTful API server. How have you structured your code to handle different endpoints and requests efficiently?

WebSocket server:
This WebSocket is an event-driven architecture with different aspects such as handling new connections, incoming messages, client disconnections, and errors. It is capable of handling multiple connections and reacting to various messages. The architecture breaks down into two groups, the server-side and client-side. The server-side is handling, http server, WebSocket server, client management, and message broadcasting. The client-side is handling the websocket client, user interface in the browser, and messages. For the WebSocket server, http endpoints were not implemented, instead structured the code to use WebSocket events and message handling. The first event *"clients.add"* is the main entry point for a new WebSocket connection. When a new client connects, it adds a client to a group of connected clients. The next event is *"broadcast"*, which broadcasts the message to all the other connected clients and logs the received messages. There is also an event created for removing the clients from the group of connected clients, which is the *"clients.delete"*. Lastly, there is also an implementation of error handling for the WebSocket connection. Since the code is structured around WebSocket events. This event-driven approach is efficient because it only executes the code when specific events occur.

Restful API:
The architecture of this restapi is simple and has efficient handling of endpoints and requests. We're setting up the server using Express.js and then followed by a middleware to parse incoming requests. Under the setup we include the endpoints (POST, GET, DELETE) defined by Express's routing methods. Each of these routes follow an operation, such as creating, reading, and deleting resources.

Discuss the design decisions, libraries, and frameworks used in your implementation. Explain how your servers handle different types of requests and how they could be extended or modified for additional functionality in the future.

WebSocket server:
The main design aspect of this WebSocket is simplicity. The idea was to keep it straightforward and easy to understand. While keeping it simple, it's also scalable. Grouping the connected clients allows for efficient addition and removal of clients. Another design decision of this WebSocket was to broadcast messages to all clients except the sender. When creating this WebSocket there was no usage of external libraries and only utilized the "ws" library, which is popular and an efficient WebSocket implementation. There are 4 types of requests handled in the WebSocket, such as connections to connect clients, messages to broadcast for clients, disconnecting the clients, and handling errors for the server. A modification that can be made to the connection event in the WebSocket could be user authentication. When a client connects, they can send a "register" message along with a username. The server will store the username and check if it's already been taken.

Restful API:
The main design aspect of this WebSocket is that it follows CRUD operations. Another design decision made for this restapi server is that error handling is implemented in routes and globally. Some of the libraries and frameworks used to create this restapi are Express.js, body-parser, and morgan. Express.js is simple to use and a popular choice. Body-parser was implemented to handle parsing of requests and morgan is used for logging purposes. The different types of requests handled in the restapi are POST(create), GET(read), and DELETE(Delete). Each request is handled by a specific route. In the future we can add more routes to extend the functionality such as PUT for updates, and GET by ID for retrieving specific resources.