Chat App System Architecture

QuickNode Interview Written Assignment

System Architecture Outline

Next are the main components of the architecture to satisfies the main requirements:

- A Front-End Client: a phone application (potentially, a web application later) to allows users to interact with each other - send content (messages, files, etc.), see their chat history and online/offline presence.
- A Back-End Client: a main service that handles the following services in coordination to satifies an increase in requests, database capacity, geography expansion and resilience:
    - Real-Time Chat Service: since we need real time communication, this service provides WebSockets connections: a full-duplex channel would better serve the requirement of "*messages should be delivered asap*" since the service can received them and route them without any barriers or lag (ideally);
    - Activity Service: this registers the last time the user was active on the chat so it will serce thre purposes: show the online/offline presence, last-time connection and let know the Notification Service if a new message should be directly serve throught the WebSocket or a push notification should be sent the the User;
    - Notification Service: based on the prior service, this one is the one in charge to either send a message to a user throw the websocket channel or send a push notification. This service gets the messages from a queue system;
    - General Service: through HTTP requests (a web service) it handles user's authentication/authorization, streaming of chat history and others (not in scope, e.x.: profile information);

Considerations

Next are some considerations, relations within components and tradeoffs/cost:

- Chat History: the history could be stored in the phone or in the cloud; but one would require sufficiente storage capacity and the other backups, security and requests load capacity. A good approach would be to have a ***hybrid storage*** solution where the recent chat is stored in the phone and more older chat is pushed to the cloud for future request. But, rest assured that, the best approach would be determined by how much we want to stored globally, concurrent users accessing cloud storage and resources to serve that.
- Real–time Chat: there are other protocols to serve this goal such as long polling or XMPP; Long-polling request the server for messages and keeps waiting until one exists

or a timeout is reached, although the server provides the message to the requester in real time it could be in detrinement of the resources since web service connections have to be kept alive;

- Load Balancing: the back-end client could be deployed across multiple servers to handle large number of concurrent users and geographies, this way we can offer high availability by distributing incoming traffic;
- Message Routing: the Real-time Chat Service and the Notification Service act in coordination to handle the messages routing, so if both users are connected (the Activity Service lets the system know if a user is in the App or not) the messages are routed through the WebSockets directed by the Notification Service; but if one or both are not connected the Notificatioon service pulls messages from the Messages Queue and sends a push notification to the users;

System Architecture Diagram