React, PostgreSQL, and Cloud Services Integration Challenge (Thought Exercise)

Design a system architecture for a scalable, real-time chat application using React for the frontend, PostgreSQL for database management, and AWS or Azure for cloud services. Your design should include:

- User authentication and data security measures.
- Real-time message exchange and storage strategies.
- Scalability solutions for growing numbers of users and messages.

Integration of a feature that allows users to send cryptocurrency tips to each other, leveraging blockchain technology.

Provide a detailed discussion on how you would approach user authentication, data security, real-time communication, and the integration of blockchain technology for the tipping feature. Include considerations for choosing specific AWS or Azure services to support these functionalities.

Proposed solution:

Technologies to use

- Amazon Cognito for user authentication and authorization
- ReactJS for the frontend and AWS Amplify to host frontend
- AWS RDS PostgreSQL or Aruroa Serverless PostgreSQL to host our database
- DynamoDB for caching messages and user details
- EC2 for hosting our backend
- Cloud watch for keeping track of logs
- AWS Elastic load balancing to distribute traffic between multiple instances of EC2
- AWS Autoscale to increase or decrease EC2 instances depending on the load
- AWS Budgets to keep track of our spending and set spending limits
- S3 to host user profile images

Backend:
We can build our backend in any technology we want. For this example we are choosing NodeJS. Our backend will be responsible for creating a socket connection with our frontend for real time communication, storing and retrieving messages from our database and cache. We can host our backend on an EC2 instance, and we can use AWS Autoscale to spin up more EC2 instances if our instances get overloaded. We will also use a load balancer to distribute the load between our EC2 instances.

User security:
Instead of rolling our own auth, we can leverage AWS Cognito service to handle user authentication and authorization. This will help us make sure that only registered users are

accessing our messaging feature, and help our backend to send messages to the appropriate users.

Database(s):
To store our messages we will use AWS Aurora Serverless (PostgreSQL edition). The advantage of Aurora Serverless over traditional RDS is its ability to scale up and down depending on load. This will save on cost as well as making sure our database is responsive. But it is not a good idea to query our database repeatedly, because even in the case of Aurora it can still end up costing us a lot and slow things down for our end user. To decrease the load on our database and speed things up for our end users we use DynamoDB.
DynamoDB being a NoSQL database provides us with faster response times. We can use it as a caching layer, storing messages for the past week. This way when a user is trying to retrieve messages from the past week it will query from DynamoDB instead of PostgreSQL, making the application feel more responsive.

Frontend:
We will code up our frontend chat application in React. We can use CSS frameworks like TailwindCSS or MaterialUI to speed up the design process. And host the application on Amplify.

Search:
Users would want to search each other and groups on the platform. We can either search by email or crypto wallet. But in order to have a more flexible search, we would either need to leverage full text searching functions provided by PostgreSQL, or use Elasticsearch by AWS.

Media assets:
Users would want to set their profile picture, and send each other pictures, audio and videos. For that we would need to store that data somewhere, and we can use AWS S3 service to store and host the media. To optimize on cost, we can use S3's lifecycle rules which would push content from S3 Standard to Standard IA to Glacier based on how old the media content is.

Crypto/Blockchain integration:
We can integrate MetaMask to allow users to send each other crypto currency of their choice. If we wish to roll out our own crypto currency, we can look into using Blockchain by AWS. And integrate it with our system.

Budgeting:
It is important to manage our costs on AWS. AWS provides us with the budget service where we can add alerts if our bill exceeds a certain threshold, and even stop our services if we have utilized all of our monthly budget. Budgeting is important to avoid unexpected bills.