Cognecto Hackathon Requirements - Chatbot Project

Goal: Build a domain-specialized chatbot for Cognecto that can handle document ingestion (CSV/PDF/docs), maintain context and chat history, render charts/tables, and manage token usage efficiently.

1. Chat History
- Store each conversation (user + assistant + timestamps).
- Allow retrieval, export, and session management.
- Use DB (Postgres/Mongo) + cache (Redis). Retain summaries for long chats.

2. Question Context
- Include relevant prior messages and retrieved document chunks.
- Use Retrieval-Augmented Generation (RAG) for context assembly.
- Keep context token budget within limits (~3-5k).

3. Multiple Pages or Data
- Handle multi-page PDFs and long documents by chunking with metadata.
- Store page numbers, sections, filenames for accurate retrieval and citations.

4. Chart and Table
- Allow inline rendering of tables and charts (bar, line, pie).
- Model returns chart data in JSON; frontend renders using Chart.js/Recharts.

5. Ability to Read CSV Each Row
- Parse CSV, index rows, and allow per-row query and analysis.
- Build APIs for upload, query, and row retrieval. Stream large files in chunks.

6. Token Management
- Use windowing and summarization to keep prompts under token limits.
- Track token count with tiktoken-like library and trim content if needed.
- Optimize context retrieval and control costs.

7. UX - OpenAI Style
- Clean, chat-based UI with left-side session list, central chat pane, upload area.
- Include model selection, system prompt edit, and token usage indicator.
- Inline rendering for tables/charts and file citations.

8. Prompt for Question & New Session

- Provide input field with templates and "New Session" options.
- Support guided prompts and clear/reset session controls.

9. Ability to Load CSV, PDF and Document

- File ingestion pipeline: CSV, PDF, DOCX, XLSX, PPTX.
- Extract text/tables, chunk, embed, and store with metadata in vector DB.
- Support document search and question answering.

10. Take Front End API Key

- Prefer backend-managed API keys for security.
- If using user-supplied keys, never store plaintext and mask in logs.
- Provide clear user consent and deletion options.

API Example Endpoints:

- POST /session - create new chat session
- GET /session/{id} - get chat history
- POST /session/{id}/message - send message
- POST /upload - upload and parse document
- GET /doc/{doc_id}/search?q=... - semantic search
- POST /doc/{doc_id}/row-query - query CSV rows
- POST /chart - render chart from assistant output

Priorities for Hackathon:

1. MVP: chat history, context (RAG), CSV upload & per-row query, chart output, token guard.
2. Optional: PDF table extraction, auto-summarization, encrypted key handling.
3. Security first: protect API keys and user data.