Backend Interview Questions

Problem Statement

Grey needs a single backend service that simultaneously:

1. **Records every financial event under strict double-entry accounting.**
    - The system must ensure that for every debit, there is a corresponding credit, and vice versa, following the rules of double-entry accounting.
    - **Background on Double-Entry Accounting (DEA):**
        - The core equation is:

          ```
          Assets = Liabilities + Owner's Equity
          ```

        - **Assets:** What the bank owns (e.g., cash, buildings, equipment).
        - **Liabilities:** What the bank owes (e.g., users' account balances, salaries).
        - **Owner's Equity:** Contributions the owners made to start or keep the company running (e.g., capital, stock).
    - You are free to define services, methods, data flow, and architecture as you see fit. Consider scalability, reliability, and handling of edge cases and failures.
2. **Lets users attach any type of file—images, videos, documents, audio—to those events and retrieve them within seconds, anywhere in the world.**
    - Supported file types:
        - **Images** (e.g., `.jpg`, `.png`, `.heic`)
        - **Videos** (e.g., `.mp4`, `.mov`)
        - **Documents** (e.g., `.pdf`, `.docx`, `.xlsx`)
        - **Audio files** (e.g., `.mp3`, `.wav`)

- Uploaded files should:
  - Be available for **viewing or downloading within seconds** of upload.
  - Be **secure**, with **access control and file validation**.
  - Be **scalable**, supporting **millions of uploads** and **global access**.
  - Optionally support **preview generation** (e.g., image thumbnails, PDF previews).

Scale Assumptions

- The platform must support ~10 million financial transactions and ~50 million file uploads per month.
- Integrity, security, and low latency are non-negotiable.

Your Task

Describe how you would build, deploy, and operate this system end-to-end. Cover:

- Overall architecture and component services
- Data models and storage choices (ledger tables, metadata for files)
- API contracts or interface definitions for posting transactions and uploading/retrieving files
- How you ensure strict debit-equals-credit enforcement, atomicity, and consistency in the ledger
- File ingestion flow (upload URLs or endpoints), validation, storage (hot vs. cold tiers), and global delivery (e.g., CDN design)
- Security controls (authentication, authorization, encryption at rest/in transit, audit logs)
- Scalability strategy (sharding or partitioning, load-balancing, queueing for heavy workloads)
- Reliability and failure handling (idempotency, retries, dead-letter queues, backup/restore, disaster recovery)
- Observability and monitoring (key metrics, alerting strategy)

- Handling of edge cases (e.g., partial failures during a transaction, invalid file formats, corrupted uploads)

Feel free to illustrate your approach with diagrams, pseudo-code, or sequence flows where appropriate. Be explicit about trade-offs and alternative approaches you considered.