Q2: Data Model & Data Storage

The data model is designed to support a wide variety of form and question types by employing a hybrid database approach and flexible data structures.

- **PostgreSQL:**
  - **user**: Stores user credentials and profile information.
  - **permission**: Manages user access rights to forms.
  - **response**: Stores form responses. The response_data field is of type JSONB to accommodate varying response structures.
  - **respondent_details**: Stores details about form respondents, with a flexible respondent_data (JSONB) field.

- **MongoDB:**
  - **documents**: Represents a form, including its current CRDT state.
  - **crdt_operations**: Stores individual CRDT operations for real-time collaboration.
  - **last_processed_version**: Tracks processed CRDT operations' version for consistency.
  - **document_versions**: Stores snapshots of form data at specific points in time.
  - **document_distributions**: Tracks how forms are distributed.

Document Schema

- **Flexible Response Data:** The core of supporting diverse form and question types lies in the *response* entity. The *response_ata* field, stored as JSONB in PostgreSQL, is crucial. This JSON structure allows each response to contain a variable set of key-value pairs, where the keys are question identifiers, and the values are the respondent's answers. This schema-less approach handles multiple choice, short answer, rating scales, and other question types without requiring database schema changes for each new form.

- **Flexible Form Types:** The *document_versions* entity stores snapshots of the form at different stages. This is important because it allows distributing and collecting responses to a specific version of the form, ensuring consistency even if the form is edited later.

- **Respondent Details:** The *respondent_details* entity stores information about the users who responded to the form.

Efficient Analysis of Form Responses

- **PostgreSQL for Analysis:** PostgreSQL is used to store and analyze form responses because it provides robust querying capabilities and supports JSONB.

- **JSONB Querying:** PostgreSQL's JSONB functions allow us to extract and aggregate data from the *response.response_data* field. For example, we can extract the values for specific questions and calculate averages, counts, or other statistics.

- **Indexing:** Proper indexing, especially on the *response.document_id* in the table and potentially within the *response.response_data* JSONB column, is essential for efficient analysis, even with a large number of responses.

Data Consistency

Data consistency is achieved through a combination of strategies:

- **ACID Properties (PostgreSQL):**
  - PostgreSQL, used for storing user data, permissions, and responses, guarantees ACID (Atomicity, Consistency, Isolation, Durability) properties.
  - This ensures that transactions are processed reliably. For example, when a user's permissions are updated, the update is either fully applied or not at all, preventing inconsistencies.
  - ACID is crucial for maintaining the integrity of sensitive data and business logic.

- **CRDTs for Real-time Collaboration:**
  - For real-time collaborative editing of documents, the system leverages CRDTs (Collaborative Replicated Data Types).
  - CRDTs are designed to allow multiple users to edit the same document concurrently without conflicts.
  - Each CRDT operation is designed to converge, meaning that if all operations are applied to the document state, all users will eventually see the same consistent result.
  - CRDT operations are stored in the *crdt_operations* entity in MongoDB. The *crdt_operations.versionVector* attribute is used to track causal dependencies between operations, ensuring that they can be applied in a consistent order.

- **Version Vectors for Causal Ordering:**
  - To ensure that CRDT operations are applied in the correct order, the system uses version vectors.
  - Each operation is tagged with a version vector that tracks the history of operations seen by the client that generated the operation.
  - When a server or client receives an operation, it checks the version vector to determine if it has already seen the operations that causally precede it.
  - This ensures that operations are applied in causal order, preventing inconsistencies such as applying an edit that deletes a section of text before the section was actually created.

- **Idempotency:**
  - Both the server and the Kafka consumer check for idempotency of CRDT operations.
  - This means that if the same operation is received multiple times (for example: due to network retries), it is only applied once.
  - This prevents duplicate applications of the same operation, which could lead to data corruption.

- **Eventual Consistency (MongoDB):** MongoDB provides eventual consistency.

- **Snapshot:**
  - The system employs snapshotting to ensure that the document state can be recovered in case of server failures.
  - Kafka consumer periodically creates snapshots of the document and stores them in MongoDB and Redis.

- **Recovery:**
  - If a server fails, a new server can retrieve the latest snapshot and apply any subsequent CRDT operations from the *crdt_operations* to reconstruct the current document state.
  - The *last_processed_version* entity in PostgreSQL plays a key role in this recovery process by tracking the last processed operation, ensuring that no operations are missed.