Q5: Service Level System Design

1. Service Architecture Overview

Designed as a microservices architecture with the following core services:

- **Authentication Service**
  - Handles user authentication and JWT token management
  - Verifies permissions and authorization for all operations

- **User Service**
  - Manages user authentication, authorization, and profile data

- **Document Service**
  - Manages form creation, editing, retrieval and versioning
  - Processes and applies CRDT operations
  - Serialize and Deserialize Document Snapshot data if needed

- **Real-time Collaboration Service:**
  - Provides the user interface for interacting with the form system
  - Manages real-time collaboration via WebSockets
  - Broadcasts CRDT operations to connected clients
  - Ensures operation causality and idempotency

- **Distribution Service:**
  - Creates and manages form distributions
  - Generates and manages distribution links
  - Handles respondent authentication for form access

- **Response Service:**
  - Processes form submissions from respondents
  - Stores and manages response data
  - Provides response analytics

- **Kafka Consumer Service:**
  - Processes CRDT operations from Kafka
  - Creates periodic snapshots of documents
  - Manages caching of document versions

2. Client Application Interactions

2.1.0 Document Editor (Client)

- **Authentication Flow:**
  1. Client sends credentials (username/password) to the API Gateway
  2. API Gateway routes to Authentication Service which interacts with User Service
  3. Authentication Service validates credentials and issues JWT
  4. Client stores JWT for subsequent requests

- **Document Creation Flow:**
  1. Client sends document creation request with JWT to API Gateway
  2. API Gateway routes to Document Service
  3. Document Service verifies the user's authorization (using the User Service)
  4. Document Service creates a new document in MongoDB and returns document Id
  5. Client stores document ID to load the document for editing

- **Document Loading Flow:**
  1. Client sends existing document retrieval request with JWT to API Gateway
  2. API Gateway routes to Document Service
  3. Document Service verifies the user's authorization (using the User Service)
  4. Document Service retrieves the latest document data from MongoDB (potentially via the Document Service and Redis)
  5. Document Service sends the deserialized document data to the Client

- **Real-time Collaborative Editing Flow:**
  1. Client establishes WebSocket connection via API Gateway to Real-Time Collaboration Service
  2. Client sends CRDT operations to Collaboration Service when:
     - User pauses typing (200ms)
     - Regular interval passes (2 seconds)
     - Character threshold reached (20 characters)
  3. Real-time Collaboration Service processes the CRDT operation, updates the CRDT data in MongoDB while updating Server In-Memory document, and broadcasts the operation to other connected clients
  4. Client receives CRDT operations from other users via WebSocket
  5. Client applies operations to local document state

- **Document Versioning Flow:**
  - Client sends a request for a specific document version to the API Gateway
  - API Gateway routes the request to the Document Versioning Service
  - Document Versioning Service retrieves the snapshot from Redis or MongoDB and sends it to the Client via the API Gateway

- **Form Distribution Flow:**
  - Client sends a request for a specific document version to the API Gateway
  - API Gateway routes the request to the Document Versioning Service
  - Document Service retrieves the snapshot from Redis or MongoDB and sends it to the Client via the API Gateway

2.1.1 Document Respondent (Client)

- **Form Access Flow:**
  - Respondent accesses form via distribution link
  - API Gateway routes to Distribution Service
  - Distribution Service validates link and retrieves form version
  - Document Service provides form structure
  - Respondent receives form data to render

- **Form Submission Flow:**
  - Respondent completes form and submits responses to the API Gateway Service
  - API Gateway Service routes the request to the Response Service
  - Response Service validates and stores the response data in PostgreSQL
  - Respondent receives confirmation of submission

2.2 Service-to-Service Communication

- **Synchronous Communication (REST/HTTP):**
  - **Client to API Gateway**: RESTful API calls for Authentication Service, User Service, Document Service, Distribution Service, Response Service
  - **API Gateway to Services**: HTTP routing based on request path and method
  - **Inter-Service Synchronous Calls**: REST APIs for direct service-to-service communication when immediate response is required

- **Asynchronous Communication (Event-Driven):**
  - **Document Service to Kafka**: Publishing CRDT operations as events
  - **Kafka to Kafka Consumer Service**: Consuming CRDT operation events
  - **Real-Time Collaboration Service to Clients**: Real-time updates via WebSockets

- **Database Access Patterns:**
  - **Authentication Service → PostgreSQL**: User and permission data
  - **Document Service → MongoDB**: Document storage and CRDT operations
  - **Document Service → Redis**: Snapshot caching and retrieval
  - **Response Service → PostgreSQL**: Form response storage
  - **Kafka Consumer Service → MongoDB**: Storage of document snapshots and version vectors

3. Service Interactions by Flow

- **CRDT Operation Processing Flow:**

```
Client → API Gateway
  → Real-Time Collaboration Service
  → Document Service (apply operation, acquire lock)
  → MongoDB (store operation)
  → Kafka (publish operation)
  → Document Service (release lock)
  → Collaboration Service (broadcast to other clients)
  → Acknowledge Client
  → Client (other connected users)
```

- **Kafka Consumer Processing Flow:**

```
Kafka → Kafka Consumer Service (receive operation)
  → In-memory processing (CRDT merge operation)
  → MongoDB (store version vector)
  → MongoDB (create snapshot periodically)
  → Redis (cache snapshot)
```

- **Document Version Retrieval Flow:**

```
Client → API Gateway
  → Document Service
  → Redis (check for snapshot)
  → MongoDB (if not in Redis)
  → Redis (cache snapshot)
  → Document Service
  → Client
```

- **Server Recovery Flow:**

```
New Server → MongoDB (retrieve last processed version vector)
  → MongoDB (retrieve pending operations)
  → MongoDB (retrieve latest snapshot)
  → In-memory processing (apply operations)
  → Server now consistent
```

4. Communication Protocols

- **HTTP/HTTPS:** For standard request/response communication
- **WebSocket:** For real-time, bidirectional communication between the Client and the Server
- **Kafka:** For asynchronous, publish-subscribe messaging for CRDT operations

5. Data Transfer Formats

- **JSON:** Used for transferring data between services in HTTP/HTTPS requests and WebSocket messages

6. Service-Level Fault Tolerance

- **Authentication Service Failure:**
    - Multiple instances behind load balancer
    - JWT verification can be performed by any instance

- **Document Service Failure:**
    - Stateless design allows requests to be routed to other instances
    - New instance can recover state from MongoDB and version vectors

- **Real-Time Collaboration Service Failure:**
    - WebSocket connections re-establish to new instance
    - Clients maintain local document state during reconnection

- **Kafka Failure:**
    - Document Service falls back to direct MongoDB operation retrieval
    - System continues functioning with reduced performance
    - Automatic recovery when Kafka becomes available

- **Redis Cache Failure:**
    - System falls back to MongoDB for snapshots
    - Performance impact but no data loss
    - Automatic cache rebuilding when Redis recovers

7. Service Deployment Architecture

Each service is containerized and deployed as multiple instances for high availability.

- **Authentication Service:**
    - Stateless
    - Horizontally scalable

- **Document Service**:
    - Stateless with shared state in MongoDB
    - Horizontally scalable

- **Real-Time Collaboration Service**:
    - Sticky sessions for WebSockets
    - Horizontally scalable

- **Document Distribution Service**:
    - Stateless
    - Horizontally scalable

- **Response Service**:
    - Stateless
    - Horizontally scalable

- **Kafka Consumer Service**:
    - Stateful
    - Scaled based on Kafka partition count