Compliance Report for "Key Account Manager (KAM) Lead Management System"

Introduction

Udaan, a B2B e-commerce platform, requires a Lead Management System for Key Account Managers (KAMs) who manage relationships with large restaurant accounts. This system will help track and manage leads, interactions, and account performance.

Requirements

Core Requirements

1. Lead Management -- Implemented
   - Add new restaurant leads
   - Store basic restaurant information
   - Track lead status
2. Contact Management -- Implemented
   - Multiple Points of Contact (POCs) per restaurant
   - Store contact details (name, role, contact information)

    - Support multiple POCs with different roles
3. Interaction Tracking -- Implemented
    - Record all calls made to leads
    - Track orders placed
    - Store interaction dates and details
4. Call Planning -- Implemented
    - Set call frequency for each lead
    - Display leads requiring calls today
    - Track last call made
5. Performance Tracking -- Implemented
    - Track well-performing accounts
    - Monitor ordering patterns and frequency
    - Identify underperforming accounts

Technical Requirements

1. Data Models -- Implemented
    - Database schema/data structures design
    - Entity relationship management
    - Efficient querying capabilities
2. API Design -- Implemented
    - RESTful APIs for all operations
    - Error handling
    - Authentication and authorization
3. Business Logic -- Implemented
    - Logic for determining today's calls
    - Account performance metrics calculation
    - Lead status transition handling

Implementation Guide

System Design Questions

1. "Can you describe a basic way to structure this application? Just focus on the main components."

    Answer :

Component	Role
Database	Stores data (e.g., leads, interactions, metrics)
Models	SQLAlchemy models representing database tables
Schemas	Pydantic models for input validation and response serialization
CRUD Operations	Centralized functions for interacting with the database
Routers	FastAPI endpoints organized by domain
Authentication	Secure access with JWTs and OAuth2
Frontend Pages	Streamlit pages for user interaction

API Client	Handles communication between Streamlit and FastAPI

2. "What type of database would work well here? SQL or NoSQL?"
   Answer : For this project, a SQL database (e.g., PostgreSQL or MySQL) is the most suitable choice due to its structured data model, strong relational capabilities, and need for consistent querying.

   SQL Database (Relational):
   - **Strengths:**
     - **Structured Data**: The data for the KAM Lead Management System (e.g., leads, interactions, POCs, etc.) is highly structured, making relational databases a good fit.
     - **Complex Queries**: SQL databases support complex joins and queries, which will be necessary for tasks like finding underperforming accounts or summarizing interactions.
     - **ACID Compliance**: Ensures data consistency, particularly useful for critical tasks like tracking account performance or updating lead statuses.
     - **Relationships**: The system has clear entity relationships (e.g., leads, POCs, and interactions), which are naturally modeled with SQL.
   - **Conclusion**: SQL is suitable because the system involves structured, relational data with strong consistency requirements and complex querying needs. (ER Diagram can be found below)

3. "What are some simple ways we could handle more users as the application grows?"

   Answer: To handle more users as the application grows, some simple strategies that can be employed include:

1. **Horizontal Scaling**:
   - Add more instances of the application server to distribute the load.
   - Use a load balancer to evenly distribute traffic across instances.
2. **Database Optimization**:
   - Use read replicas for the database to handle read-heavy workloads.
   - Archive old or less frequently used data to reduce database size.
3. **Caching**:
   - Use in-memory caching (e.g., Redis, Memcached) to store frequently accessed data like performance metrics or leads, reducing database queries.
   - Implement HTTP caching for API responses.
4. **API Rate Limiting**:

    - Limit the number of API requests per user or IP address to prevent overloading the system.

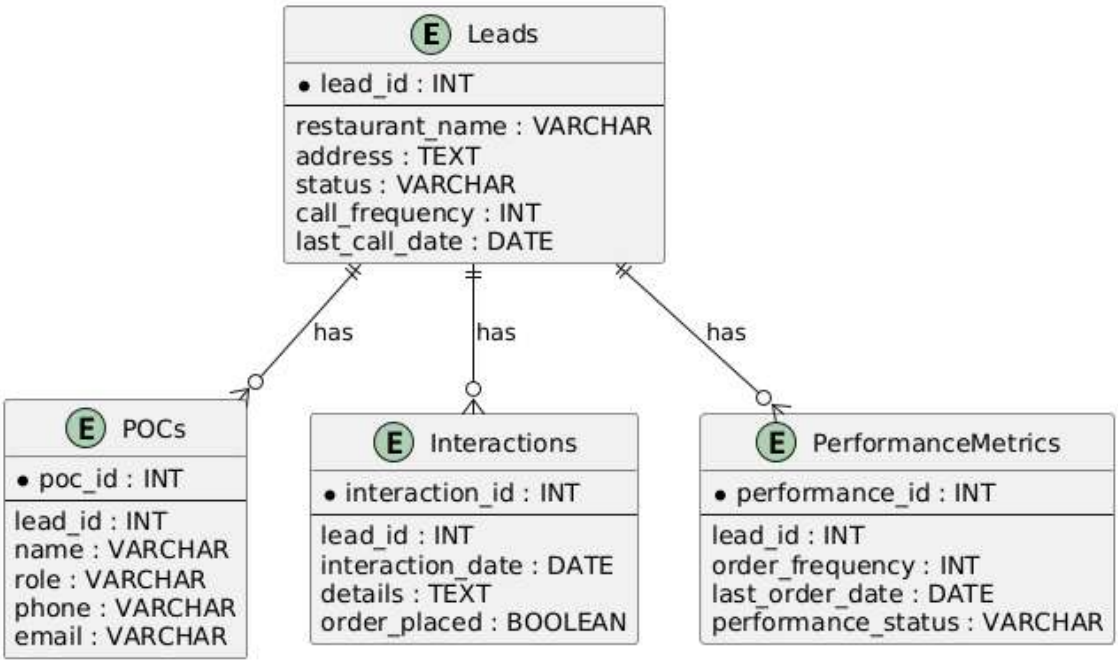
5. **Monitor and Analyze**:
    - Use monitoring tools to track system performance and identify bottlenecks (e.g., high CPU usage, slow database queries).
    - Perform load testing to understand system capacity and plan for scaling.
6. **Session Management**:
    - Use distributed session storage (e.g., Redis) to allow seamless scaling of application servers.
    - Implement token-based authentication to avoid server-side session overhead.
7. **Cloud Scaling Features**:
    - Leverage cloud provider features like auto-scaling, managed databases, and serverless functions for flexible scaling.
    - Use multi-region deployments to improve latency for users in different locations.
8. **Partitioning and Sharding**:
    - Partition data by user or region to reduce database contention.
    - Use sharding to distribute data across multiple databases.

By implementing these strategies, the application can handle more users efficiently while maintaining performance and reliability.

Data Modeling Questions

1. "Could you sketch out some basic tables/collections we'd need?"

Answer: The ER Diagram is as follows



2. "How would you connect these different pieces of data together?"

   Answer: The above ER Diagram shows the relationship/ connection between these difference pieces of data

3. "What fields should we consider indexing to make things faster?"

   Answer:

   1. Any field frequently used in WHERE clauses or filters should be indexed:

   - status in Leads for querying leads by their current status (e.g., "Active").
   - interaction_date in Interactions for filtering interactions by date.
   - performance_status in PerformanceMetrics for identifying underperforming or well-performing accounts.

   2. Index fields commonly used in ORDER BY clauses:

   - last_call_date in Leads for sorting leads by the most recent call.
   - last_order_date in PerformanceMetrics for ranking accounts by recent activity.

   3. Fields Used in Joins:

   - lead_id in PerformanceMetrics for joining with the Leads table.

4. Foreign Keys:

    - Index foreign key fields (lead_id in POCs, Interactions, and PerformanceMetrics) to speed up joins between related tables.

5. Fields Used in Aggregations:

    - Index fields used in aggregate functions like COUNT, SUM, or AVG: order_placed in Interactions for counting the number of orders.

Implementation Details

Write code that implements above requirements.

Edge Cases

1. How would you handle change in KAM?

    Answer: To handle a change in KAM (Key Account Manager) in the Lead Management System:

- **Reassign Leads**: Update the database to assign the affected leads to the new KAM.
- **Notify Stakeholders**: Notify clients and internal teams about the change to ensure seamless communication.
- **Data Access Control**: Adjust user permissions to ensure the outgoing KAM no longer has access to the leads.
- **Transfer Performance Metrics**: Link the performance metrics and history to the new KAM for continuity.
- **Audit Trail**: Maintain logs of the KAM change for transparency and record-keeping.

2. How would you handle timezone differences for call scheduling?

    Answer: To handle timezone differences for call scheduling:

- **Standardize to UTC**: Store and manage all timestamps in UTC format to ensure consistency.
- **Local Time Conversion**: Convert UTC to the user's local time based on their timezone settings when displaying or scheduling calls.
- **Timezone Detection**: Automatically detect the user's timezone or allow them to set it manually.
- **Conflict Resolution**: Use timezone-aware libraries to prevent overlaps or conflicts in scheduling across regions.

- **Clear Communication**: Display both local time and UTC time in call reminders to avoid confusion.

In the provided code, this is handled by UTC format

Submission Guidelines

Code Requirements

- Working condition code : Working code is given in zip file
- Clear dependency specification :
  Following are the dependency specifications
  - **Python**: 3.9 or above
  - **Docker**: 20.10 or above
  - **Docker Compose**: 1.29 or above
  - **FastAPI**: Version specified in fastapi/requirements.txt
  - **Streamlit**: Version specified in streamlit/requirements.txt
  - **PostgreSQL:** Version specified in docker compose
  - **jq library on Linux:** Latest Version for JSON parsing needed for testing using test script
- Sample data inclusion : As provided in database/createdatabase.sql file

Documentation Requirements

README.md file containing: (README.md file with following 7 + 2 sections is inside zip file)

1. Project overview
2. System requirements
3. Installation instructions
4. Running instructions
5. Test execution guide
6. API documentation
7. Sample usage examples

Additional Details

8. Project Structure
9. Interesting Techniques

Video Demonstration

5-10 minute video showing: (xx minutes video showing the following demonstrations)

1. Code setup process
2. Application running
3. Major features demonstration
4. Sample inputs/outputs

5. MP4 format

Additional Points:
1. Sample test script
2. Possible Enhancements

Submission Format (zip file containing 4 items mentioned below)

ZIP file containing:

1. Source code directory
2. README.md
3. requirements.txt or package.json
4. Demonstration video (demo.mp4)

Evaluation Criteria

1. Modularity: Docker Compose organizes and builds the configuration for multi-container applications, which is aligned with the principles of the Builder design pattern
    - Separation of concerns
    - Design pattern usage
    - Component reusability
    - Well-defined interfaces
2. Extensibility
    - New feature addition ease -- yes
    - Component configurability -- yes
    - Interface/abstract class usage -- yes
3. Completeness
    - All required features -- yes
    - Error handling -- yes
    - Edge case coverage -- yes
    - Comprehensive testing -- yes
4. Code Readability
    - Clear naming conventions -- yes
    - Proper documentation -- yes
    - Consistent coding style -- yes
    - Appropriate comments -- yes

Bonus Features

1. Unit Testing: Created a shell script to test all interfaces and functions (provided as test_endpoints.sh) to ensure test coverage and demonstrated in video.
    - Comprehensive test coverage
    - Scenario and edge case testing
    - Integration tests
    - Test documentation
2. REST API Implementation
    - RESTful endpoint design
    - API documentation
    - HTTP status codes
    - Request/Response validation
    - Authentication/Authorization

3. Functional User Interface
    - Clean, intuitive design
    - Responsive layout
    - Interactive features
    - Error handling
    - Cross-browser compatibility
4. Deployment -- Docker
    - Online accessibility
    - Deployment documentation
    - Environment configuration
    - CI/CD pipeline

Project Timeline

- Submission Deadline: January 5th, 2025 -- Deadline met
- Early submissions encouraged - Submitted before dead line
- Rolling basis evaluation –

Submission Process

1. Complete implementation -- Done
2. Record demonstration video -- Done
3. Create ZIP file with all components -- Done
4. Verify all requirements – Coding requirements , Technical requirements & documentation requirements are verified and all are meeting
5. Submit via email – Submitted by replying to the mail received

Important Notes

1. Incomplete submissions not considered - Completed in all respect and submitted
2. Original work required - Done
3. Third-party libraries allowed with justification : Following are the 3rd party libraries used and their justification is as follows
    a. Docker and Docker compose: For a deployable environment which can execute on any device
    b. jq library on Linux: Latest Version for JSON parsing needed for testing using test script
4. Plagiarism results in disqualification – All code and content submitted is original and free from plagiarism. Strict adherence to academic and professional integrity is maintained, ensuring that all work is independently created