Project Report – Distributed Electronic Medical Records System (DME)

1. Project Overview

This project consists of designing and implementing a **centralized Electronic Medical Records (EMR/DME) system** for a network of clinics. The system enables healthcare professionals (doctors, nurses, pharmacists) to access and manage patient records, prescriptions, treatments, and notifications across multiple establishments. Due to the high number of users and large data volume, a **distributed 3-tier architecture** was adopted instead of a monolithic design.

2. Adopted Architecture

The system follows a **3-Tier Distributed Architecture**, ensuring separation of concerns, scalability, and maintainability:

Presentation Tier

- Implemented using **React**
- Provides user interfaces for medical staff
- Handles user interactions, form submissions, and data visualization
- Communicates with the backend via REST APIs
- Receives real-time updates through asynchronous notifications

Business Logic Tier

- Implemented using **Django**
- Contains the core application logic
- Organized into domain-oriented services:
  - **Patient Service**
  - **Prescription Service**
  - **Treatment Service**
  - **Notification Service**
- Each service is responsible for a specific business domain
- Publishes and consumes events asynchronously via Kafka

Data Tier

- Implemented using **PostgreSQL**
- Centralized relational database
- Stores all persistent data:
  - Patient records
  - Prescriptions
  - Treatments
  - Notifications metadata
- Ensures data consistency and transactional integrity

3. Asynchronous Communication and Middleware

To decouple services and support scalability, the system uses **Apache Kafka** as a **message-oriented middleware**.

- Kafka acts as an **event broker** between services
- Services publish domain events (e.g., `prescription.created`, `treatment.updated`)
- Other services consume these events asynchronously
- This approach:
  - Reduces direct dependencies between services
  - Improves fault tolerance
  - Allows independent scaling of components

Kafka is integrated within the **business logic tier** as an infrastructure component, without breaking the 3-tier architectural model.

4. Notification Handling

The **Notification Service** is implemented as a backend service that:

- Subscribes to Kafka topics
- Processes medical events
- Sends notifications to concerned users
  This service is logically part of the **business tier**, while communication remains asynchronous through Kafka.

5. Security Considerations

- Authentication and authorization are handled at the backend level
- Secure API access between frontend and backend
- Controlled access to patient data based on user roles

6. Technologies Used

- **Frontend:** React
- **Backend:** Django
- **Database:** PostgreSQL
- **Middleware:** Apache Kafka

7. Conclusion

The proposed solution successfully implements a **secure, scalable, and decoupled distributed system** using a 3-tier architecture. By combining Django, React, PostgreSQL, and Kafka, the system efficiently handles concurrent users, large data volumes, and asynchronous workflows while remaining maintainable and extensible.