CHUBB®

CAPSTONE PROJECT (Loan Management System)

Enterprise-Grade Secure Full-Stack Web Application

INDEX

(flight app with angular)

INDEX

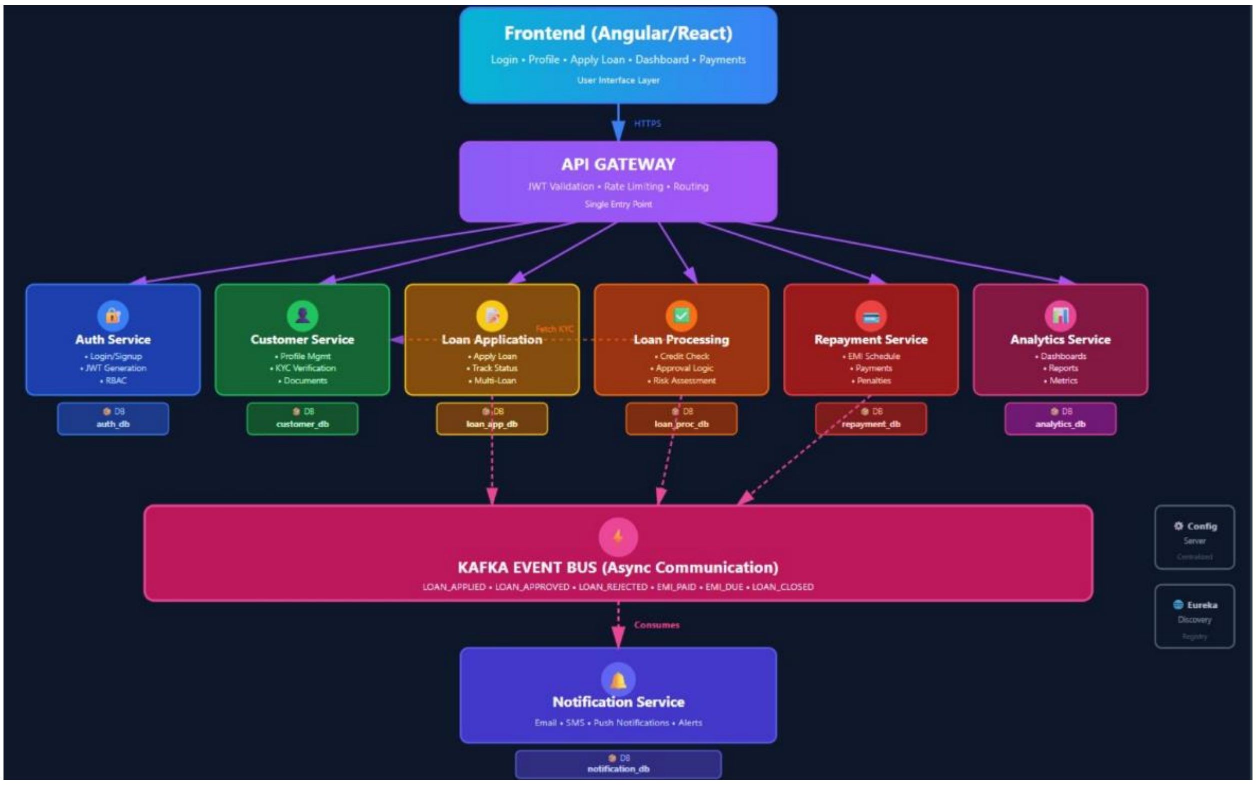
1) Purpose of the document

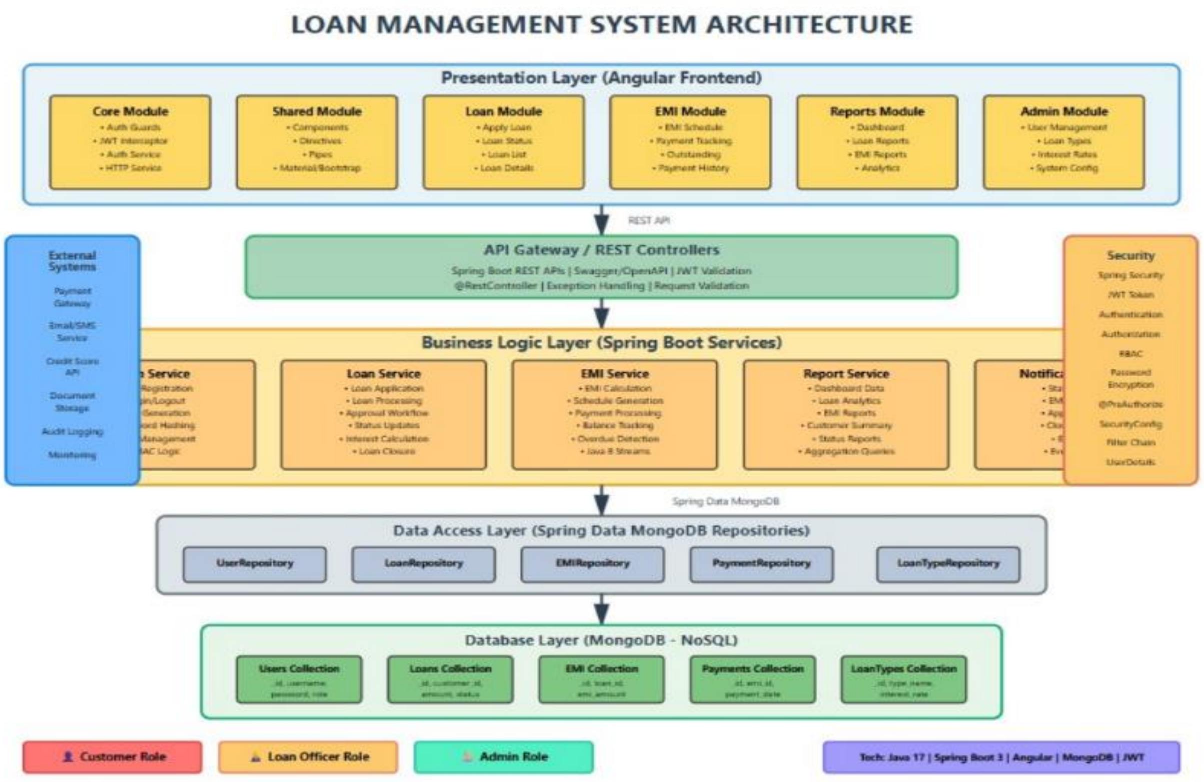
This document provides a comprehensive technical and architectural description of the Loan Management System (LMS). It is intended to serve as a reference for understanding how the system is designed, implemented, and operated within an enterprise environment.

The key goals of this document are to:

- Present the **business motivation** behind the system and outline the overall solution vision
- Offer a **high-level overview** of the system for academic evaluators and technical stakeholders
- Explain the **architectural approach**, major components, and important design choices
- Clearly define the **roles and responsibilities of each microservice**, including API exposure and data ownership
- Describe the **security model**, validation mechanisms, error handling strategy, and non-functional requirements
- Outline the **testing approach and quality assurance practices** used to ensure system reliability and correctness

System Overview




LOAN MANAGEMENT SYSTEM ARCHITECTURE
Presentation Layer (Angular Frontend)
Core Module
• Auth Guards
• JWT Interceptor
• Auth Service
• HTTP Service
Shared Module
• Components
• Directives
• Pipes
• Material/Bootstrap
Loan Module
• Apply Loan
• Loan Status
• Loan List
• Loan Details
EMI Module
• EMI Schedule
• Payment Tracking
• Outstanding
• Payment History
Reports Module
• Dashboard
• Loan Reports
• EMI Reports
• Analytics
Admin Module
• User Management
• Loan Types
• Interest Rates
• System Config
REST API
API Gateway / REST Controllers
Spring Boot REST APIs | Swagger/OpenAPI | JWT Validation
@RestController | Exception Handling | Request Validation
External Systems
Payment Gateway
Email/SMS Service
Credit Score API
Document Storage
Audit Logging
Monitoring
Business Logic Layer (Spring Boot Services)
Service
Registration
in/Logout
Generation
ord Hashing
Management
AC Logic
Loan Service
• Loan Application
• Loan Processing
• Approval Workflow
• Status Updates
• Interest Calculation
• Loan Closure
EMI Service
• EMI Calculation
• Schedule Generation
• Payment Processing
• Balance Tracking
• Overdue Detection
• Java 8 Streams
Report Service
• Dashboard Data
• Loan Analytics
• EMI Reports
• Customer Summary
• Status Reports
• Aggregation Queries
Notifica
Security
Spring Security
JWT Token
Authentication
Authorization
RBAC
Password Encryption
@PreAuthorize
SecurityConfig
Filter Chain
UserDetails
Spring Data MongoDB
Data Access Layer (Spring Data MongoDB Repositories)
UserRepository
LoanRepository
EMIRepository
PaymentRepository
LoanTypeRepository
Database Layer (MongoDB - NoSQL)
Users Collection
_id, username, password, role
Loans Collection
_id, customer_id, amount, status
EMI Collection
_id, loan_id, emi_amount
Payments Collection
_id, emi_id, payment_date
LoanTypes Collection
_id, type_name, interest_rate
Customer Role
Loan Officer Role
Admin Role
Tech: Java 17 | Spring Boot 3 | Angular | MongoDB | JWT

2.1 Business Objective

The Loan Management System aims to modernize and streamline the complete loan processing lifecycle for banks, NBFCs, and other financial institutions. By eliminating manual and partially automated processes, the system delivers a **secure, scalable, and compliance-ready digital solution** capable of handling real-world financial operations.

The primary business objectives of the system are:

- Reduce loan processing time through automated workflows
- Ensure clear and traceable loan approval mechanisms
- Provide precise EMI calculations with end-to-end repayment monitoring
- Protect sensitive customer and financial data using strong security controls
- Enable real-time insights through dashboards and operational analytics
- Support cloud deployment with a flexible, API-driven system design

2.2 High-Level Features

- Support for multiple user roles, including **Administrator, Loan Officer, and Customer**
- Digital loan application submission with real-time status tracking
- Flexible configuration of loan products, including interest rates and repayment tenures
- Automatic generation of EMI schedules based on loan parameters
- End-to-end monitoring of repayments with support for loan closure
- Secure user authentication implemented using **JWT tokens**
- Fine-grained access control through **Role-Based Authorization (RBAC)**
- Analytical dashboards and reporting for operational insights
- Well-structured, production-ready **RESTful APIs** for system integration

3. Architecture Style

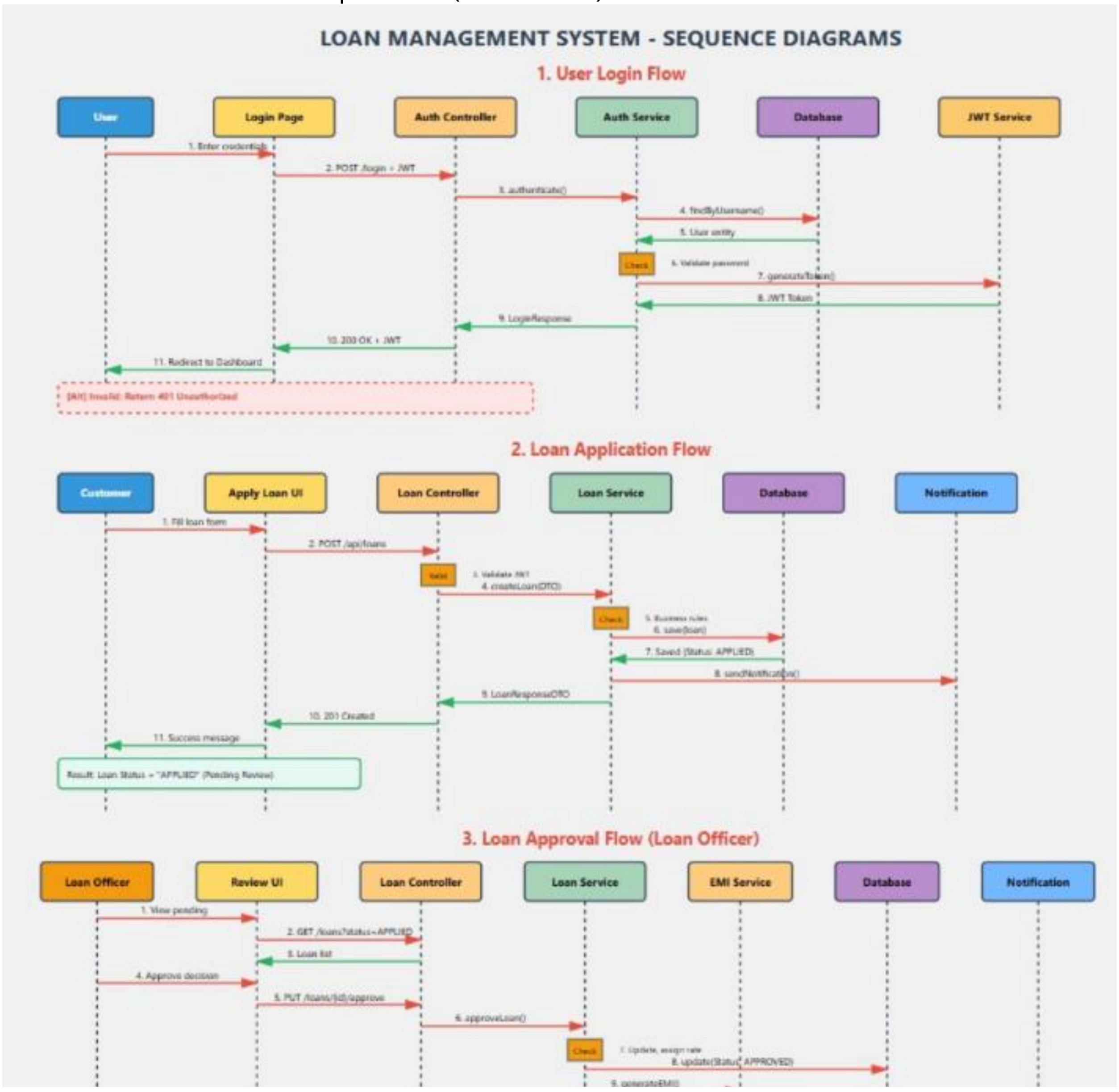
3.1 Architectural Pattern

The system follows a **Microservices-Oriented Architecture** combined with **Layered Design Principles**.

Key characteristics:

- Loose coupling between services
- Independent data ownership
- REST-based inter-service communication
- Stateless backend services

- Frontend-backend separation (SPA + APIs)



4. Core System Components

4.1 Frontend Layer

- Single Page Application (SPA)
- Handles UI rendering, form validation, and role-based navigation
- Communicates with backend via secured REST APIs

4.2 Backend Layer

- Stateless REST services
- Implements business logic, validation, security, and workflows
- Exposes versioned APIs

4.3 Persistence Layer

- Independent databases per service
- Ensures data isolation and scalability

4.4 Security Layer

- · JWT authentication
- · Role-based authorization
- · Encrypted password storage
- · Secure API endpoints

5. Technology Stack

5.1 Backend

- Java 17+
- Spring Boot 3.x
- Spring Web (REST APIs)
- Spring Data JPA
- Hibernate ORM
- Spring Security with JWT
- Swagger / OpenAI
- Maven

5.2 **Frontend**

- Angular
- TypeScript
- Bootstrap
- Reactive Forms
- HTTP Interceptors
- Route Guards

5.3 **Database**

- PostgreSQL / MySQL
- Separate DB of each microservice

5.4 **DevOps & Tools**

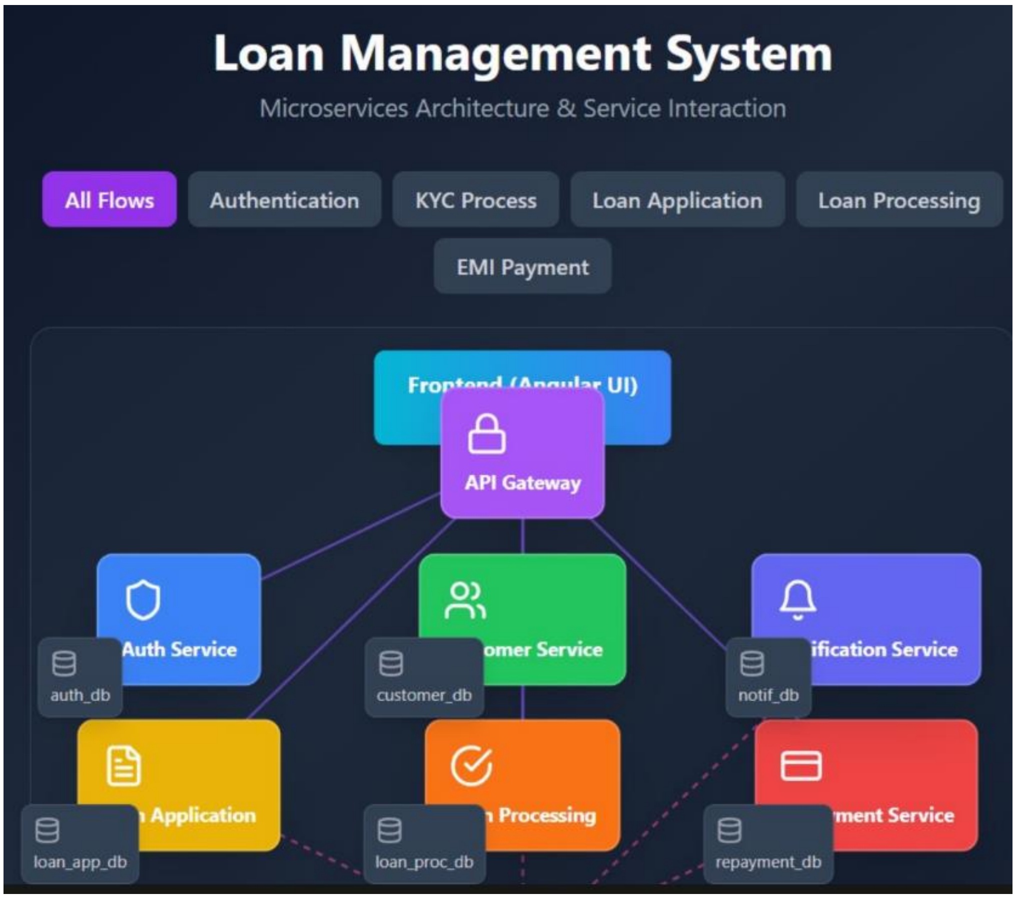
- Git & GitHub

- Postman
- Environment-based configuration (dev/test/prod)

5.5 **Testing**

- JUnit 5

- Mockito
- Postman collections



6. Microservice Design

6.1 User Service

Responsibilities

- User registration and authentication
- Role and permission management
- JWT generation and validation
- Password hashing and security policies

APIs

- POST /auth/register
- POST /auth/login
- GET /users
- GET /users/{id}
- PUT /users/{id}
- DELETE /users/{id}

Database

- User table

- Role table
- User-Role mapping table

<u>6.2 Product Service (Loan Service)</u>

Responsibilities

- **Loan type management**
- **Interest rate and tenure rules**
- **Loan application submission**
- **Loan status lifecycle management**

APIs

- POST /loans/apply
- GET /loans/{id}
- GET /loans/customer/{customerId}
- PUT /loans/{id}/approve
- PUT /loans/{id}/reject

Database

- Loan table
- Loan type table
- Loan status history table

6.3 Order Service (EMI & Repayment Service)

Responsibilities

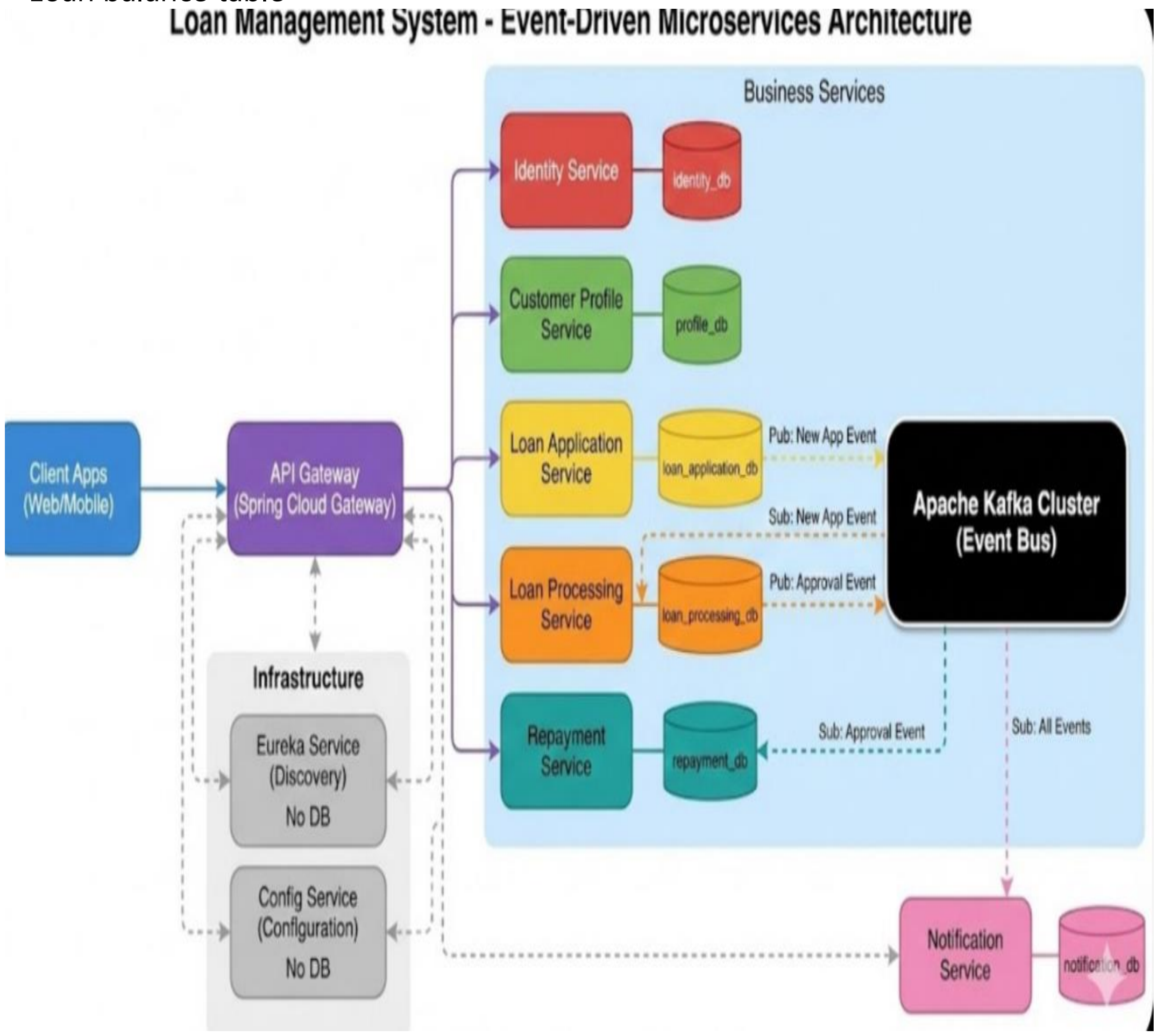
- EMI calculation
- EMI schedule generation
- Repayment tracking
- Outstanding balance computation
- Automatic loan closure

APIs

- · GET /emis/loan/{loanId}
- · POST /emis/pay
- · GET /repayments/customer/{customerId}

Database

- • EMI schedule table
- · Repayment table
- · Loan balance table



7. Data Design (Low-Level Design)

7.1 User Document

- userId
- name
- email
- password (hashed)
- role
- status
- createdAt

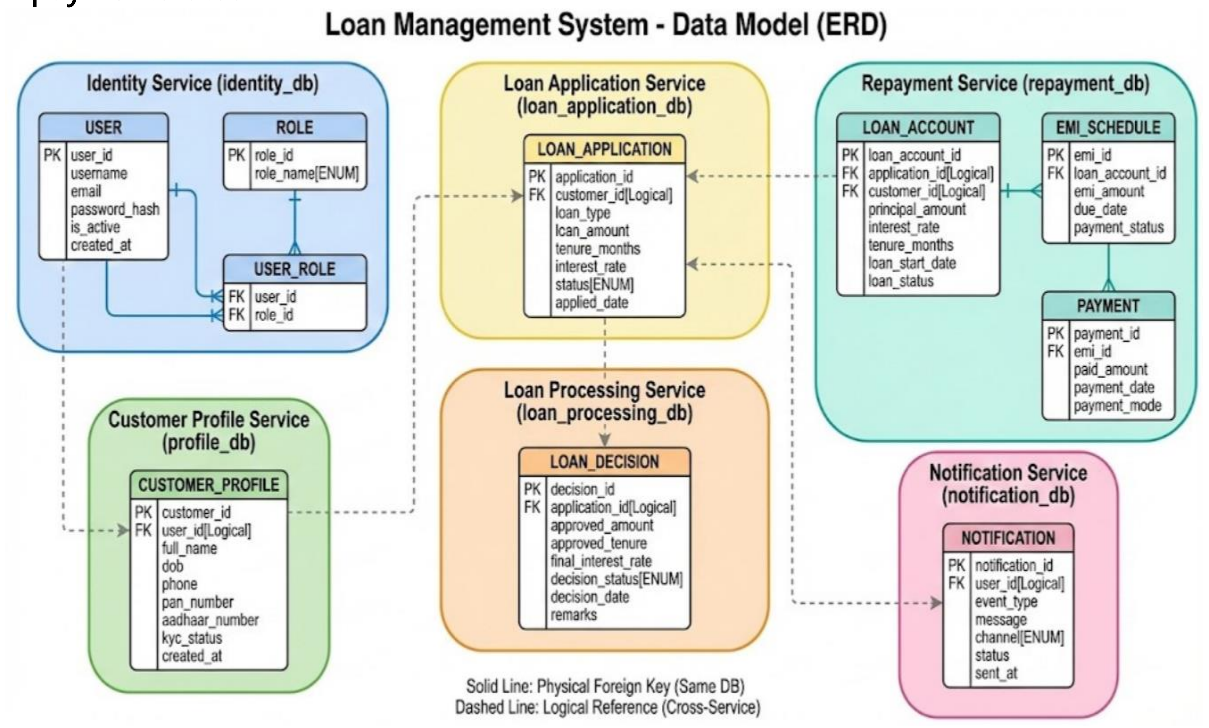
7.2 Product (Loan) Document

- · loanId
- · customerId
- · loanType
- · principalAmount
- · interestRate

- · tenure
- · status
- · approvalRemarks

7.3 Order (EMI) Document

- · emiId
- · loanId
- · dueDate
- · emiAmount
- · paidAmount
- · outstandingBalance
- · paymentStatus



8. API Design & Validation

- RESTful conventions followed
- DTOs used for request/response
- Input validation using @Valid
- Custom validators for:
- Loan amount limits
- Tenure constraints
- EMI payment rules

9. Error Handling Strategy

9.1 Global Exception Handling

- Centralized exception management using @ControllerAdvice
- Custom exceptions:
- ResourceNotFoundException
- ValidationException
- UnauthorizedAccessException
- Standardized error response format:
- timestamp
- status
- errorCode
- message
- Pat

10. Security Design

- JWT-based authentication
- Stateless session management
- Role-based access control
- Password hashing using BCrypt
- Secured endpoints with method-level authorization
- HTTP interceptors for token propagation

11. Non-Functional Requirements

- Scalability: Horizontally scalable services
- Security: Encrypted credentials and secure APIs
- Performance: Optimized queries and pagination
- Maintainability: Clean layered architecture
- Availability: Fault-tolerant stateless services
- Auditability: Complete loan lifecycle traceability

12. Testing Strategy

12.1 Unit Testing

- Service-layer tests using JUnit and Mockito
- Mocked repositories and external dependencies

12.2 API Testing

- Postman collections for all endpoints
- Authentication and authorization test cases

12.3 Validation & Security Testing

- Input validation tests

- Unauthorized access tests
- Token expiry handling

Conclusion

The Loan Management System showcases a robust enterprise-grade architecture by combining secure full-stack development with practical financial domain workflows. The application is designed to meet real-world operational requirements and follows modern software engineering principles.

With its cloud-ready deployment model, scalable microservices architecture, and adherence to industry best practices, the system effectively demonstrates strong backend and frontend engineering skills. This project serves as a comprehensive example of contemporary application design suitable for both academic evaluation and professional environments.