CHUBB®

CAPSTONE PROJECT (Loan Management System)

Enterprise-Grade Secure Full-Stack Web Application

INDEX

(flight app with angular)

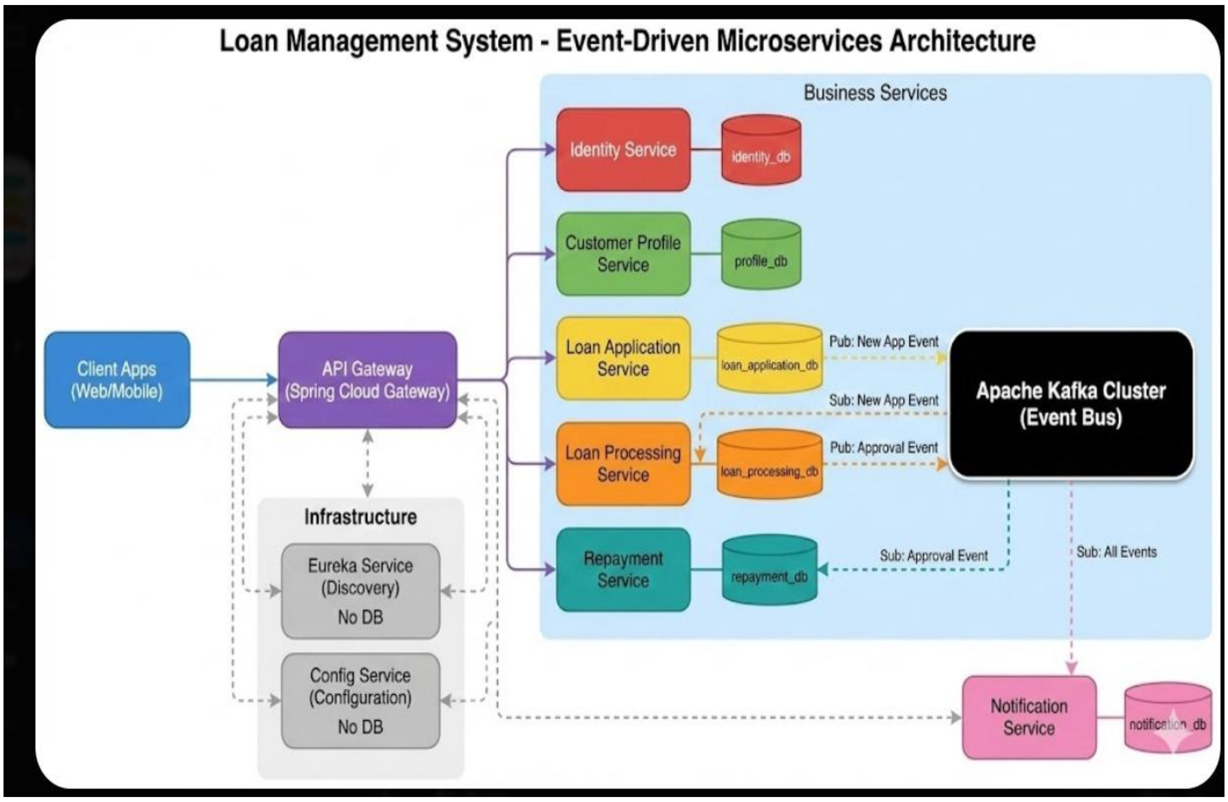
INDEX

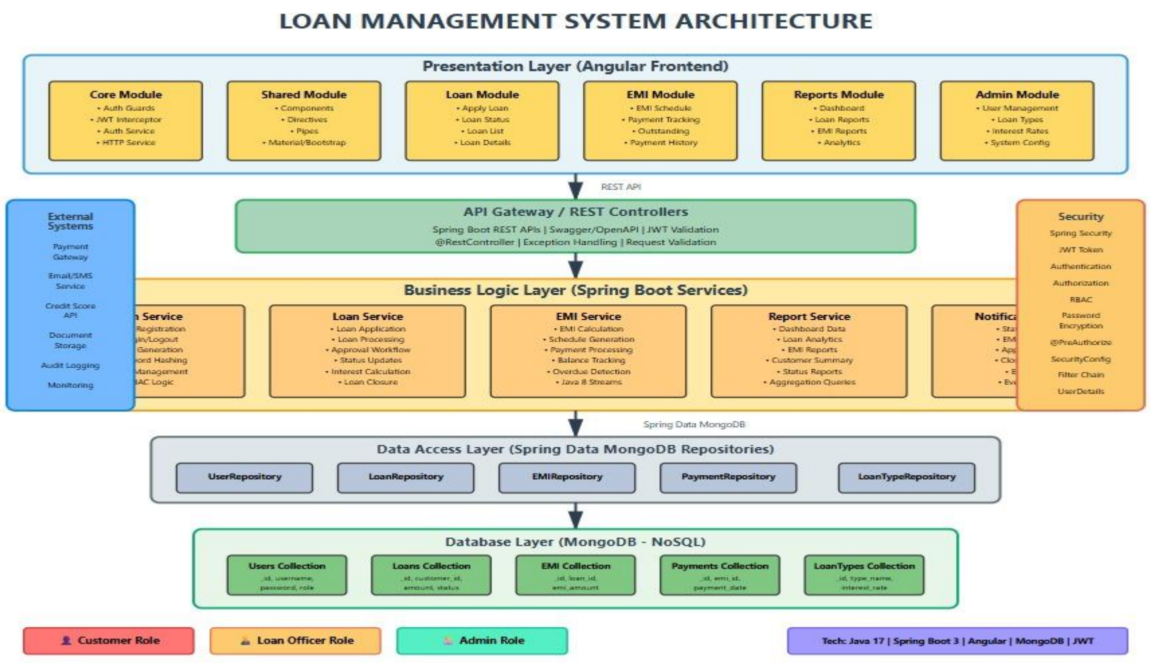
1) Purpose of the document

This document serves as the **authoritative technical and architectural reference** for the **Loan Management System (LMS)**.
Its primary objectives are to:

- Explain the **business rationale and system vision**
- Provide a **high-level system overview** for stakeholders
- Describe the **architectural style, core components, and design decisions**
- Detail the **microservice responsibilities, APIs, and data ownership**
- Define **security, error handling, validation, and non-functional requirements**
- Present a **robust testing and quality assurance strategy**

System Overview





2.1 Business Objective

The Loan Management System is designed to **digitize and automate the end-to-end loan lifecycle** for banks, NBFCs, and financial institutions.
The system replaces manual and semi-automated workflows with a **secure, scalable, and audit-ready platform.**

Key business goals include:

- Faster loan application processing
- Transparent approval workflows
- Accurate EMI computation and repayment tracking
- Secure handling of sensitive financial data
- Real-time dashboards and operational reports
- Cloud-ready, API-first architecture

2.2 High-Level Features

- Multi-role system: **Admin, Loan Officer, Customer**
- Online loan application and tracking

- Configurable loan types, interest rates, and tenures
- Automated EMI schedule generation
- Repayment tracking and loan closure
- Secure authentication using JWT
- Role-based authorization (RBAC)
- Reporting and dashboard analytics
- Production-ready REST APIs

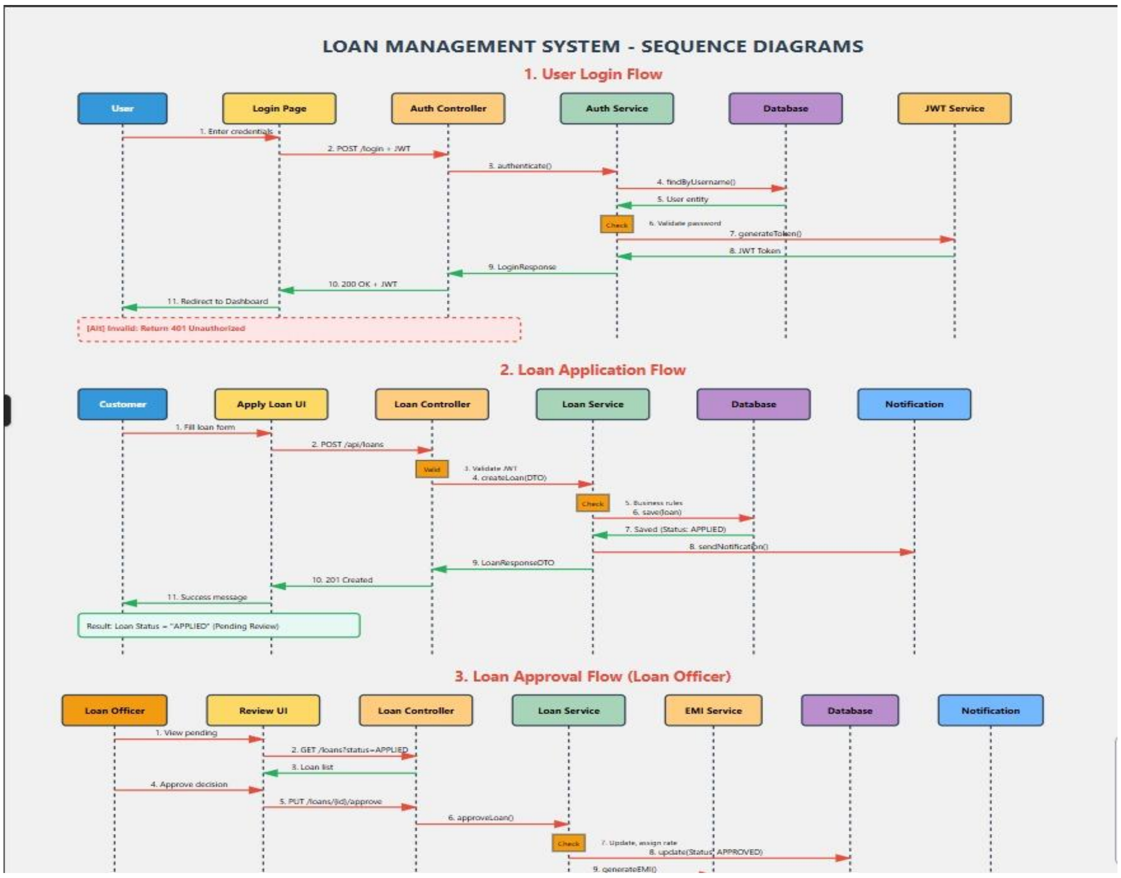
3. **Architecture Style**

3.1Architectural 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 / OpenAPI
- Maven

5.2 Frontend

- **Angular (latest)**
- **TypeScript**
- **Angular Material / Bootstrap**
- **Reactive Forms**
- **HTTP Interceptors**
- **Route Guards**

5.3 Database

- **PostgreSQL / MySQL**
- **Separate schema per 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**

6.2 Product Service (Loan Service)

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 demonstrates **enterprise-level system design, secure full-stack development, and real-world financial workflows**.
It is **production-ready**, **cloud-deployable**, and built using **industry best practices**, making it a strong representation of modern backend and frontend engineering capabilities.