# Loan Management System (LMS) - Project Documentation

---

## Team: Loan Management System 1

## Prepared By:

- Golagana Vandana
- Aguru Sandeep
- Potu Purna Sai
- Athikamsetti Janaki Ram

---

## Abstract

The Loan Management System (LMS) is a full-stack web application designed to manage the complete loan lifecycle with role-based access for Admins and Customers. Customers can register, apply for loans, track EMIs, view payment history, and download receipts or NOCs in PDF format using jsPDF, with email services to deliver these documents directly to their inbox. Admins can review applications, approve or reject loans, monitor repayments, and view analytics through interactive dashboards. The application is built with React for the frontend and Spring Boot with MySQL for the backend, secured using JWT authentication and BCrypt encryption. Key features include automated EMI calculation, email notifications, CIBIL score simulation, and a responsive UI for an efficient and user-friendly loan management experience.

# 1. Introduction

## 1.1 Purpose

The primary purpose of the Loan Management System (LMS) is to digitalize and automate the complete loan lifecycle for both customers and administrators. Traditionally, loan processing involves manual verification, paper-based documentation, and lengthy approval timelines, which lead to inefficiency and human errors.

This system provides a secure, user-friendly, and transparent platform where:

- Customers can seamlessly register, apply for loans, check their eligibility (via simulated CIBIL score), and monitor their loan progress in real time.
- Administrators can efficiently manage customer information, evaluate loan requests, configure loan types, and track repayment activities.

By leveraging Spring Boot, React, and MySQL, the LMS ensures scalability, automation, and reduced operational overhead. It not only streamlines the loan approval and repayment process but also enhances customer trust through features like status timelines, EMI reminders, OTP-secured payments, and automated NOC issuance.

## 1.2 Scope

The scope of the project is divided into two key roles: Customers and Administrators.

Customers:

- Account Management: Register and log in securely with JWT authentication.
- Loan Application: Submit loan requests by providing Aadhaar, PAN, income, employment type, and purpose of the loan.
- Eligibility Check: Get a system-generated CIBIL score to assess loan eligibility.
- Loan Tracking: View application status via a visual timeline with automated email updates.
- EMI Management: View repayment schedule, receive reminders before due dates, and make EMI payments through OTP verification.
- Document Access: Download EMI receipts and loan closure certificates (NOCs) directly from the system.
- Withdrawal Option: Cancel/withdraw a loan application before admin approval.

Administrators:

- Loan Processing: Approve, reject, or close loan applications with comments.
- Loan Configuration: Manage loan types, interest rates, penalty policies, and max loan limits.
- User Management: Monitor customer profiles, active loans, and repayment history.
- Penalty Management: Apply late fees or penalties automatically for delayed EMI payments.
- Analytics & Reporting: Access an interactive dashboard with visual analytics on loans, repayments, and defaults.
- Export & Notifications: Generate reports (PDF/Excel) using JasperReports and send reminders/receipts via email.

---

## 2. System Overview

### 2.1 Features

- Authentication: JWT-based login for Admin and Customer roles
- Role-Based Dashboards: Graphs, analytics, and user-specific views
- Loan Lifecycle Management: Application, review, approval/rejection/closure, and
- EMI generation
- EMI Management: Automated schedules, due tracking, and payment restrictions
- Loan Type Configuration: Admin panel for interest, penalty, and policy setup
- Status Tracking: Timeline view with badges and comments for customers
- Chat Module: Admin-customer real-time communication system
- Profile Management: Secure editing of profile and password
- Email Notifications: Automated PDF receipts and NOC sent via email
- Dashboard Analytics: Visual analytics using Recharts
- PDF Generation: EMI and NOC documents generated using jsPDF and
- html2canvas
- Admin Utilities: User deletion, loan filtering, and super admin key validation

### 2.2 User Roles

- **Customer:** Loan application and tracking
- **Admin:** Loan approval, monitoring, and user management

## 3. Project Structure

```
LMS1/
├── backend/ # Spring Boot (Java 21)
│ ├── src/main/java/com/loanmanagement/
│ │ ├── controller/
│ │ ├── service/
│ │ ├── model/
│ │ ├── repository/
│ │ ├── config/
│ │ └── dto/
│ ├── src/main/resources/
│ │ ├── application.properties
│ │ └── application-secret.properties
│ └── pom.xml
├── frontend/ # React (Node.js 18+)
│ ├── src/
│ │ ├── components/
│ │ ├── services/
│ │ ├── styles/
│ │ ├── assets/
│ │ └── utils/
│ ├── public/
│ └── package.json
│
└── README.md
```

## 4. Backend — Deep Dive

- Framework: Spring Boot (Java 21) is used to build a RESTful API.
- Authentication: Implemented with JWT (JSON Web Tokens) for secure, stateless authentication. Passwords are secured using BCrypt hashing.
- Communication: Email (JavaMailSender), Chat system
- PDF Support: EMI & NOC PDFs via jsPDF (frontend) + email from backend
- Core Controllers:
  - AuthController: Login,register,update password
  - AdminLoanController:Approve/reject/close loans
  - AdminLoanTypeController: Manages loan types and configurations.
  - CustomerLoanController: Loan application,EMI views,timeline
  - AdminDashboardController: Analytics and charts
  - AdminUserManagementController: Admin user listing and deletion
  - ChatController: Messaging endpoints
- Services: Contains the core business logic, separating it from the controllers for maintainability.
- Repositories: JPA repositories are used for data access, providing a clean interface to interact with the database.
- DTOs (Data Transfer Objects): Employed to structure API responses and requests, ensuring data is transferred efficiently and securely.

---

## 5. Frontend — Deep Dive

- Framework: React is utilized to create a dynamic single-page application.
- Routing: React Router is used for navigating between different pages and managing the application's URLs.
- UI Libraries:
  - react-icons: Provides a wide range of icons. o react-toastify: Used for showing clean, customizable toast notifications.
  - recharts: Renders interactive charts for the dashboard analytics.

  html2canvas: Enables the download of dashboard charts as images.

- State Management: React's built-in hooks, such as useState and useEffect, along with a Context API for authentication, handle state management.
- Key Components:

  AdminDashboardMain, CustomerDashboardMain, ApplyLoanForm, CustomerLoanList, AdminLoanList, LoanTypeConfig, UserManagementPage EmiPaymentsPage, EmiTable, AdminChat, CustomerChat

---

## 6. Database Design

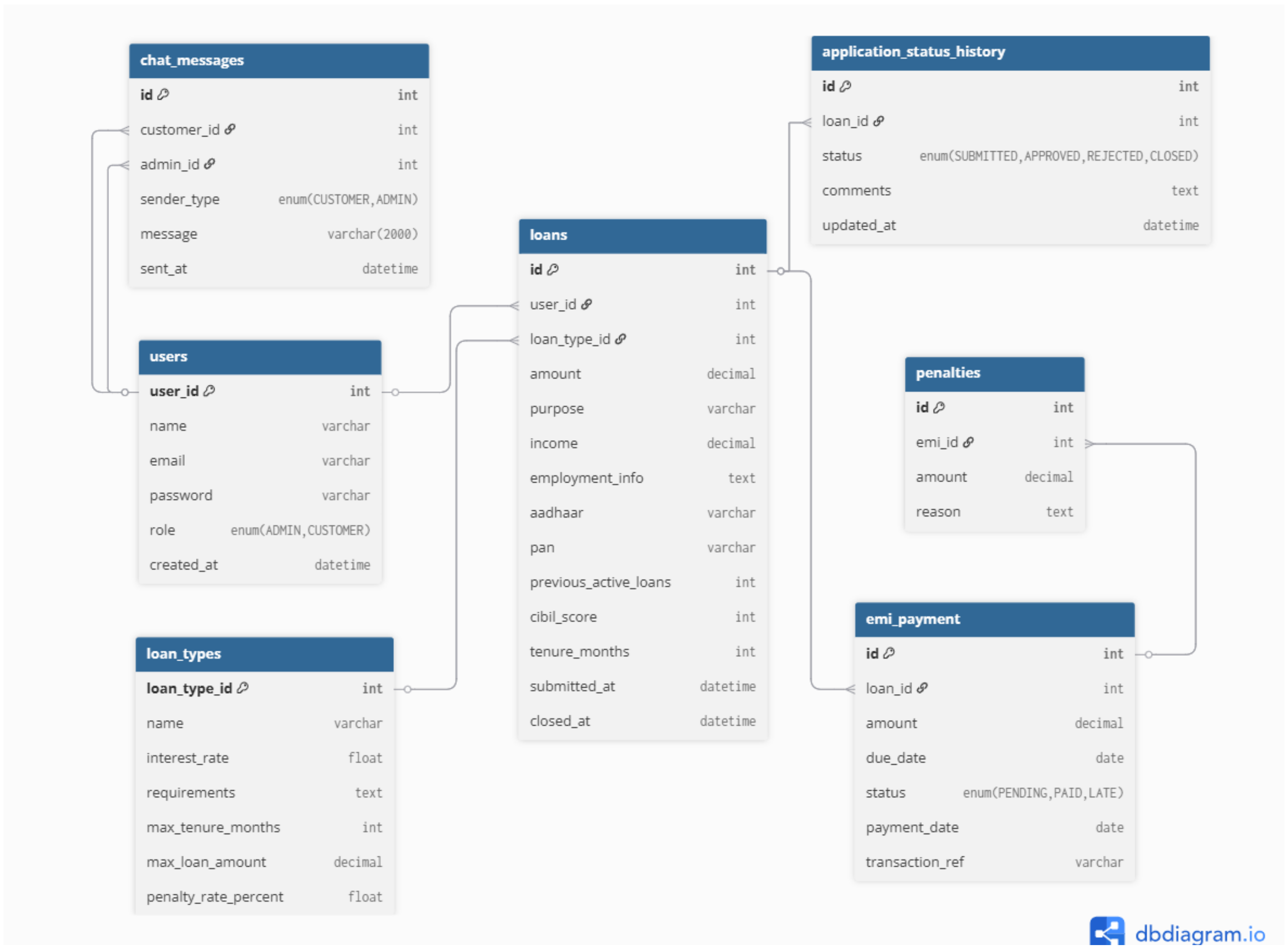
### 6.1 ERD Entities

- User
- Loan
- Loan_Type
- EMI_Payment
- Penalty
- Chat_messages
- Application_Status_History

### 6.2 Key Tables Overview

| **Table Name** | **Key Fields** |
|---|---|
| users | id, username, name, email, password, role, contact info, address details |
| loan_types | id, name, interestRate, penaltyRatePercent, maxLoanAmount, maxTenureYears, requirements, maxLoansPerCustomerPerLoanType |
| loans | id, referenceId, customer_id, loan_type_id, amount, tenure, interestRate, income, cibilScore, purpose, status, timestamps |
| emi_payment | id, loan_id, emiNumber, dueDate, amount, status, paymentDate |
| application_status_history | id, loan_id, status, comment, timestamp |
| chat_messages | id, customer_id, admin_id, senderType, message, sentAt |
| penalties | id, loan_id, amount, date, reason |

### 6.2 Database Schema



| chat_messages | |
|---|---|
| id | int |
| customer_id | int |
| admin_id | int |
| sender_type | enum(CUSTOMER,ADMIN) |
| message | varchar(2000) |
| sent_at | datetime |

| users | |
|---|---|
| user_id | int |
| name | varchar |
| email | varchar |
| password | varchar |
| role | enum(ADMIN,CUSTOMER) |
| created_at | datetime |

| loan_types | |
|---|---|
| loan_type_id | int |
| name | varchar |
| interest_rate | float |
| requirements | text |
| max_tenure_months | int |
| max_loan_amount | decimal |
| penalty_rate_percent | float |

| loans | |
|---|---|
| id | int |
| user_id | int |
| loan_type_id | int |
| amount | decimal |
| purpose | varchar |
| income | decimal |
| employment_info | text |
| aadhaar | varchar |
| pan | varchar |
| previous_active_loans | int |
| cibil_score | int |
| tenure_months | int |
| submitted_at | datetime |
| closed_at | datetime |

| application_status_history | |
|---|---|
| id | int |
| loan_id | int |
| status | enum(SUBMITTED,APPROVED,REJECTED,CLOSED) |
| comments | text |
| updated_at | datetime |

| penalties | |
|---|---|
| id | int |
| emi_id | int |
| amount | decimal |
| reason | text |

| emi_payment | |
|---|---|
| id | int |
| loan_id | int |
| amount | decimal |
| due_date | date |
| status | enum(PENDING,PAID,LATE) |
| payment_date | date |
| transaction_ref | varchar |

## 7. API Reference

### AuthController

POST /api/auth/register — Register a new user

POST /api/auth/login — Login and receive JWT

GET /api/auth/me — Get current user details (JWT required)

POST /api/auth/update-password — Update password

---

### AdminController

GET /api/admin/me — Get admin's own profile

GET /api/admin/user/{userId} — Get any user profile by ID

PUT /api/admin/user/{userId} — Update any user by ID

PUT /api/admin/update — Update admin's own profile

**AdminUserManagementController**

GET /api/admin/user-management/validate-super-key?key=... — Validate super admin key

GET /api/admin/user-management — Get all users with active loan count

DELETE /api/admin/user-management/{userId} — Delete a user

---

**AdminLoanController**

GET /api/admin/loans — Get all loans

GET /api/admin/loans/{id} — Get loan details by ID

DELETE /api/admin/loans/{id} — Delete a loan

PUT /api/admin/loans/{id} — Update loan status

---

**AdminLoanTypeController**

GET /api/admin/loan-types — Get all loan types

GET /api/admin/loan-types/{id} — Get loan type by ID

POST /api/admin/loan-types — Create a new loan type

PUT /api/admin/loan-types/{id} — Update a loan type

DELETE /api/admin/loan-types/{id} — Delete a loan type

PUT /api/admin/loan-types/loan-type-config/{id} — Update loan type config fields

PUT /api/admin/loan-types/interest-rate-config/{id} — Update interest/penalty rates

---

**AdminDashboardController**

GET /api/admin/dashboard/summary — Get dashboard summary

GET /api/admin/dashboard/users — Get user breakdown stats

GET /api/admin/dashboard/loans — Get loan breakdown stats

---

**CustomerController**

GET /api/customer/me — Get current logged-in customer profile

PUT /api/customer/update — Update customer profile

**CustomerLoanController**

POST /api/customer/loans — Apply for a loan

GET /api/customer/loans — Get all loans for the customer

GET /api/customer/loans/{id} — Get a specific loan by ID

GET /api/customer/loans/active-loan-counts — Get active loan counts (summary)

GET /api/customer/loans/active-loan-counts-detailed — Get active loan counts (detailed)

GET /api/customer/loans/{loanId}/status-history — Get status history for a loan

GET /api/customer/loans/emi/{loanId} — Get loan with EMI schedule

POST /api/customer/loans/emi/pay/{emiId} — Pay an EMI

---

**ChatController**

POST /api/chat/customer/send — Customer sends a message

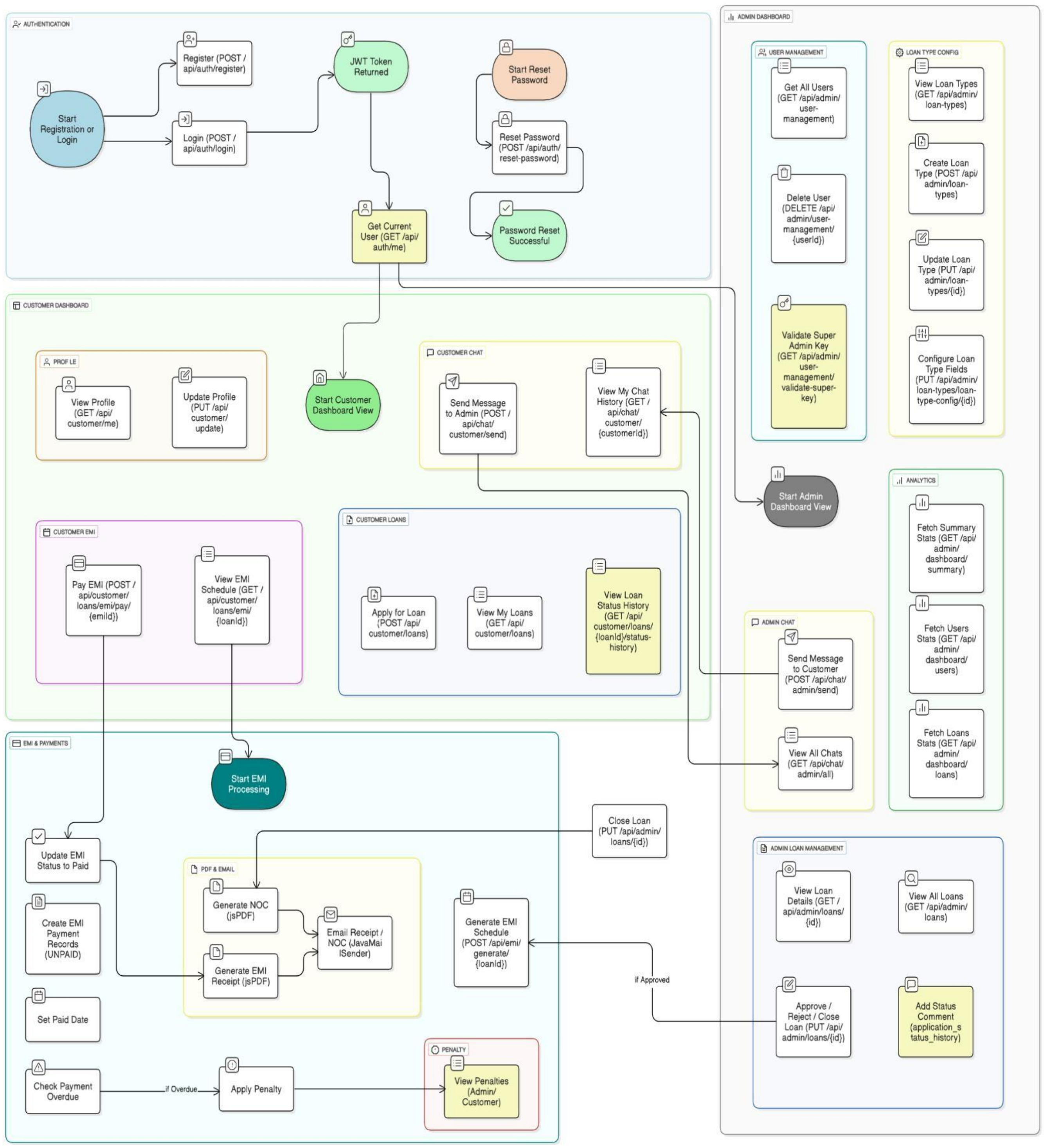
POST /api/chat/admin/send — Admin sends a message to a customer

GET /api/chat/customer/{customerId} — Get all messages for a customer

GET /api/chat/admin/all — Admin gets all chats

GET /api/chat/me — Get current user info (for chat)

## 8. Data Flow Diagrams


AUTHENTICATION
Start Registration or Login
Register (POST / api/auth/register)
Login (POST / api/auth/login)
JWT Token Returned
Get Current User (GET /api/ auth/me)
Start Reset Password
Reset Password (POST /api/auth/ reset-password)
Password Reset Successful
CUSTOMER DASHBOARD
PROFILE
View Profile (GET /api/ customer/me)
Update Profile (PUT /api/ customer/ update)
Start Customer Dashboard View
CUSTOMER CHAT
Send Message to Admin (POST / api/chat/ customer/send)
View My Chat History (GET / api/chat/ customer/ {customerId})
CUSTOMER EMI
Pay EMI (POST / api/customer/ loans/emi/pay/ {emiId})
View EMI Schedule (GET / api/customer/ loans/emi/ {loanId})
CUSTOMER LOANS
Apply for Loan (POST /api/ customer/loans)
View My Loans (GET /api/ customer/loans)
View Loan Status History (GET /api/ customer/loans/ {loanId}/status-history)
EMI & PAYMENTS
Start EMI Processing
Update EMI Status to Paid
Create EMI Payment Records (UNPAID)
Set Paid Date
Check Payment Overdue
if Overdue
Apply Penalty
PDF & EMAIL
Generate NOC (jsPDF)
Generate EMI Receipt (jsPDF)
Email Receipt / NOC (JavaMailSender)
Close Loan (PUT /api/admin/ loans/{id})
Generate EMI Schedule (POST /api/emi/ generate/ {loanId})
PENALTY
View Penalties (Admin/ Customer)
ADMIN DASHBOARD
USER MANAGEMENT
Get All Users (GET /api/admin/ user-management)
Delete User (DELETE /api/ admin/user-management/ {userId})
Validate Super Admin Key (GET /api/admin/ user-management/ validate-super-key)
LOAN TYPE CONFIG
View Loan Types (GET /api/admin/ loan-types)
Create Loan Type (POST /api/ admin/loan-types)
Update Loan Type (PUT /api/ admin/loan-types/{id})
Configure Loan Type Fields (PUT /api/admin/ loan-types/loan-type-config/{id})
Start Admin Dashboard View
ANALYTICS
Fetch Summary Stats (GET /api/ admin/ dashboard/ summary)
Fetch Users Stats (GET /api/ admin/ dashboard/ users)
Fetch Loans Stats (GET /api/ admin/ dashboard/ loans)
ADMIN CHAT
Send Message to Customer (POST /api/chat/ admin/send)
View All Chats (GET /api/chat/ admin/all)
ADMIN LOAN MANAGEMENT
View Loan Details (GET / api/admin/loans/ {id})
View All Loans (GET /api/admin/ loans)
Approve / Reject / Close Loan (PUT /api/ admin/loans/{id})
Add Status Comment (application_status_history)
if Approved

## 9.Testing

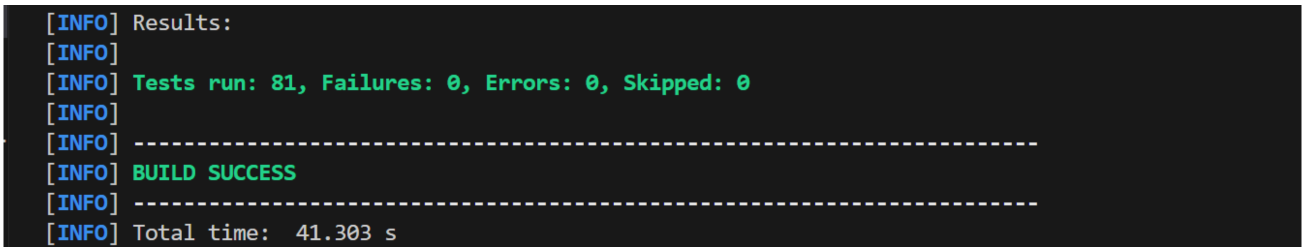
The LMS project was thoroughly tested to ensure reliability and correctness of both frontend and backend components. Testing covered unit tests, integration tests, and API validations.

### 9.1 Backend

#### 9.1.1 Unit Testing (JUnit + Mockito)

- **Tools Used:** JUnit, Mockito
- **Scope Covered:**
    - Authentication (Login, Register, Password Update)
    - Loan Management (Apply, Update, Delete, EMI Payment)
    - Admin Features (User Management, Loan Type Configurations)
    - Customer Features (Profile Update, Loan History, EMI Tracking)

**Test Results:**



```
[INFO] Results:
[INFO]
[INFO] Tests run: 81, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  41.303 s
```

- **Total Tests:** 81
- **Passed:** 81
- **Failed:** 0
- **Errors:** 0
- **Skipped:** 0

This indicates **100% success** across all test cases.

#### 9.1.2 Integration Testing

- Performed using **Postman Collections**.
- Verified end-to-end flow:
    - User Registration → Login → Loan Application → EMI Payment → Loan Closure

## 10.How to run locally

## Backend (Spring Boot)

**Prerequisites:**

- Java 21
- Maven 3.8+
- MySQL 8.x

**Steps:**

**Step 1: Configure the database and secrets**

- Open application.properties and application-secret.properties.
- Update them with your MySQL credentials and any required secrets.

**Step 2: Create the database**

- CREATE DATABASE lms_db;

**Step 3: Build and run the backend**

- cd backend
- mvn clean install
- mvn spring-boot:run

**Backend URL:** [http://localhost:8081](http://localhost:8081)

---

## Frontend(React)

**Prerequisites:**

- Node.js 18+
- Npm

**Steps:**

**Step 1: Install dependencies**

- cd frontend
- npm install

**Step 2: Start the frontend development server**

- npm start

**Frontend URL:** http://localhost:3000

---

## 11.Security Notes and Improvements

**Current Implementations**

- BCrypt for password hashing
- JWT stored in localStorage
- Spring Security for endpoint access control
- Unique email enforcement

**Suggested Enhancements**

- Rate-limit sensitive endpoints (OTP, login)
- Enable CSRF protection for mutating requests
- Store audit logs of critical actions

---

## 12.Recommendations and Next Steps

- Send EMI due reminders
- Integrate push notifications (Firebase/WebSockets)
- Build a Progressive Web App (PWA) or mobile app
- Maintain audit logs for compliance and tracking

## 13. Conclusion

SmartLend successfully integrates customer and admin needs into a single, efficient platform for managing loans. It reduces manual errors, improves transparency, and enhances user experience with features like EMI reminders, loan tracking, and a user-friendly dashboard.