Loan Management System: Project Documentation

---

1. Project Overview

Name: Loan Management System\ **Technology Stack:** Spring Boot 3, Spring Security, JWT, Hibernate (JPA), H2 Database, REST API, Microservices (optional), Multithreading, Collections, Logging, Monitoring, Exception Handling, Design Patterns, SOLID principles.

Objective: To manage users, loan applications, repayment schedules, credit scoring, and document verification through a secure, scalable, and maintainable backend system.

---

2. Module Breakdown

A. Authentication Module

- Spring Security with JWT
- Login & Registration APIs
- Role-based access (CUSTOMER, ADMIN, AGENT)
- Password hashing with BCrypt

B. User Management

- User registration
- Role management (via user_roles)
- Status tracking (ACTIVE, BLOCKED, INACTIVE)

C. Loan Application Module

- Apply for a loan based on type
- Approve/Reject logic (Admin/Agent)
- Calculate EMI, interest
- Maintain application status

D. Repayment Schedule Module

- Generate monthly EMI schedules
- Track payment status (PENDING, PAID, MISSED)
- Apply penalties for late payments

E. Document Management

- Upload & link documents to loan applications
- Types: AADHAR, PAN, INCOME_PROOF
- File path or cloud URI support

F. Credit Score Integration (Optional)

- Fetch and store score from 3rd party
- Store score, bureau name, and report

G. Notifications (Optional)

- In-app, SMS, or Email notifications
- Track status (SENT/FAILED)

H. Audit Logs

- Track every action (CREATE_LOAN, APPROVE_LOAN)
- Store user ID, action, entity, timestamp

I. Monitoring & Logs

- Spring Boot Actuator
- Custom log patterns and log file
- Endpoint: `/actuator`, `/actuator/health`, etc.

---

3. UML Overview

Entities (UML Class Diagram):

1. **User**
2. id: Long
3. username, password, fullName, email, phone, status, createdAt
4. Roles: ManyToMany with Role
5. **Role**
6. id: Long
7. roleName: String (ADMIN, CUSTOMER, AGENT)
8. **LoanType**
9. id: Long
10. name, interestRate, tenureMonths, minAmount, maxAmount
11. **LoanApplication**

12. id: Long

13. userId (FK), loanTypeId (FK), amount, tenureMonths, status, appliedOn, approvedOn, remarks

14. **LoanDocument**

15. id: Long

16. loanAppId (FK), documentType, filePath, uploadedOn

17. **RepaymentSchedule**

18. id: Long

19. loanAppId (FK), emiDate, emiAmount, status, paidOn, penaltyAmount

20. **CreditScore** (Optional)

21. id: Long

22. userId (FK), bureauName, score, reportPath, fetchedOn

23. **Notification** (Optional)

24. id: Long

25. userId (FK), message, type, status, sentOn

26. **AuditLog**

27. id: Long

28. userId (FK), action, entity, entityId, timestamp

---

4. Design Patterns Used

- **Builder Pattern**: For entity object creation
- **Strategy Pattern**: For EMI calculation or approval logic
- **Factory Pattern**: For document handling service selection
- **Singleton**: For shared config beans (e.g., JWT utils)

---

5. SOLID Principles Application

- **S**: Services follow single responsibility (e.g., LoanService, UserService)
- **O**: EMI calculators or notifiers are open for extension via interfaces
- **L**: Document upload services adhere to interface inheritance
- **I**: Smaller, focused interfaces (e.g., NotificationSender, CreditScoreFetcher)
- **D**: Beans injected via `@Autowired` or constructors

---

6. Concurrency/Multithreading

- Async task execution using Spring's `@Async`
- Pool config in `application.properties`
- Thread naming pattern: `loan-async-*`

---

7. Collections API Usage

- Maps for in-memory lookup (e.g., document types)
- Lists for storing schedules, documents
- Sets for role assignment

---

8. application.properties Highlights

```
spring.datasource.url=jdbc:h2:mem:loanappdb
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
spring.security.user.name=admin
spring.security.user.password=admin123
logging.level.com.loanmanagement=DEBUG
spring.task.execution.pool.core-size=5
```

---

9. Security (Spring Security + JWT)

- JWT token generation on login
- JWT filter before authentication
- Role-based endpoint authorization via `@PreAuthorize`
- Stateless session handling

---

10. Endpoints (Sample)

- `POST /api/auth/login`
- `POST /api/users/register`
- `GET /api/loans/types`
- `POST /api/loans/apply`
- `GET /api/repayments/schedule/{loanId}`

Let me know if you want:

- UML class diagram as PNG
- Swagger API documentation
- ERD or SQL DDL
- Code generation starting from models