Personal Loan Application System

Zeta Pioneers: *Ekta Goel (Team Lead), Aom Vivek Wankhede, Maram Krishna Koushik Reddy, Aanchal Maheshwari, Premansu Chakraborty*

GitHub Link : *https://github.com/ektagoel-12/Personal-Loan-Application-System*

PROJECT OVERVIEW

Project Goal

To provide an end-to-end solution for managing personal loan workflows: from user registration and application, to admin review/decision, EMI estimation/repayment schedule, and support ticketing — all packaged as a modular, maintainable, containerized full-stack application.

Technologies Used

- **Backend**
  - Java
  - Spring Boot
  - Spring Security / JWT (likely)
  - Spring Data JPA
- **Frontend**
  - Vue.js (or Vue)
  - JavaScript
  - Axios
- **Database**
  - PostgreSQL
- **Testing**
  - JUnit
- **Build / Packaging**
  - Maven

SYSTEM ARCHITECTURE
Backend (Spring Boot)

REST API Endpoints

Authentication APIs (/auth)

- GET /auth → Check server health.
- POST /auth/register → Register a new user.
- POST /auth/login → Authenticate user and return tokens.

- POST /auth/refresh → Refresh JWT access token using refresh token.

User Management APIs (/users)

- GET /users → Fetch all users.
- GET /users/{id} → Fetch user by ID.
- GET /users/paid/{id}→ Get total amount paid by userId
- PATCH /users/{id} → Update user details (income, credit score, etc.).
- DELETE /users/{id} → Delete user by ID.

Admin APIs (/admin)

- GET /admin → Get admin dashboard data (stats overview).
- GET /admin/loans/{id} → View details of a loan application by ID.
- PUT /admin/loans/{id}/status → Approve/Reject a loan with remarks.

Loan Application APIs (/api/loans)

- GET /api/loans → Fetch all loan applications.
- GET /api/loans/{id} → Fetch loan application by ID.
- GET /api/loans/user/{userId} → Fetch all loans applied by a specific user.
- POST /api/loans → Create a new loan application.
- PUT /api/loans/{id} → Update an existing loan application *(Not implemented)*.
- DELETE /api/loans/{id} → Delete loan application *(Not implemented)*.

Repayment & EMI APIs (/api)

- POST /api/emi/preview → Preview EMI calculation before loan approval.
- GET /api/loans/{loanId}/schedule → Get repayment schedule of a loan.
- POST /api/loans/{loanId}/generate-schedule → Generate repayment schedule (only for approved loans).
- POST /api/loans/{loanId}/schedule/{scheduleId}/pay → Mark an EMI as paid.

Support Ticket (User APIs) (/ticket/user)

- POST /ticket/user → Create a new support ticket.
- GET /ticket/user/email/{userEmail} → Fetch tickets created by a specific user.
- GET /ticket/user/ticketId/{id} → Fetch a ticket by its ID.

Support Ticket (Admin APIs) (/ticket/admin)

- GET /ticket/admin → Fetch all support tickets.
- GET /ticket/admin/ticketId/{id} → Fetch a specific ticket by ID.
- GET /ticket/admin/status/{status} → Fetch tickets by status (OPEN, PENDING, CLOSED).
- PATCH /ticket/admin/{ticketId}/response → Add/Update a response to a ticket.
- PATCH /ticket/admin/{ticketId}/{status} → Update status of a ticket.

Data Models (Entities)

- User
    - id
    - name
    - email
    - password
    - roles
    - creditScore
    - income
    - aadhar
    - createdAt
    - updatedAt
- LoanApplication
    - id
    - user
    - amount
    - tenureMonths
    - income
    - creditScore
    - status (PENDING, APPROVED, REJECTED)
    - applicationDate
    - loanType
    - reviewedBy
    - reviewedAt
    - reviewRemarks
- RepaymentSchedule
    - id
    - loan
    - month
    - principalAmount
    - interestAmount
    - balanceRemaining
    - isPaid
- SupportTicket
    - id
    - user
    - loan
    - subject
    - Description
    - type
    - status (OPEN, RESOLVED, CLOSED)
    - createdAt

- response

Services

1. AdminLoanServiceImpl

- **Purpose:** Handles all admin-side loan management operations.
- **Responsibilities:**
  - Retrieve pending loan applications for review.
  - Approve or reject loan applications based on admin decision.
  - Update loan status (APPROVED, REJECTED) along with remarks, reviewedBy, and reviewedAt.
  - Provide admin dashboard summary (e.g., total loans, pending loans, approved loans).

2. AuthServiceImpl

- **Purpose:** Manages user authentication and authorization.
- **Responsibilities:**
  - Register new users by validating details and saving hashed passwords.
  - Authenticate users using email & password.
  - Issue JWT access and refresh tokens upon successful login.
  - Validate tokens for secure access.
  - Refresh expired access tokens using refresh tokens.

3. JwtServiceImpl

- **Purpose:** Handles JSON Web Token (JWT) generation and validation.
- **Responsibilities:**
  - Generate secure access tokens after successful login.
  - Generate refresh tokens for session renewal.
  - Verify and decode tokens to extract user claims.
  - Ensure token expiration and security compliance.

4. LoanApplicationServiceImpl

- **Purpose:** Manages loan application lifecycle for users.
- **Responsibilities:**
  - Allow users to submit new loan applications.
  - Validate business rules:
    - The threshold Amount is decided by the user's income.
    - Credit score must be between 300–900.
    - Restricts the user to only 5 number of active loans.
  - Retrieve loans by user or loan ID.
  - Set initial loan status to PENDING.

5. RepaymentScheduleServiceImpl

- **Purpose:** Manages EMI calculation and repayment schedules.
- **Responsibilities:**

- Calculate EMI based on principal, interest rate, and tenure.
- Generate repayment schedule once loan is approved.
- Split EMI into **principal** and **interest** components month-wise.
- Track remaining loan balance after each EMI.
- Mark EMI entries as isPaid = true when payments are made.

6. SupportTicketAdminServiceImpl

- **Purpose:** Enables admin management of support tickets.
- **Responsibilities:**
  - Retrieve all support tickets or filter by status (OPEN, RESOLVED, CLOSED).
  - View details of a specific ticket.
  - Add or update admin responses to tickets.
  - Change the ticket status (e.g., from OPEN → IN_PROGRESS → RESOLVED).

7. SupportTicketUserServiceImpl

- **Purpose:** Handles user-side ticket creation and tracking.
- **Responsibilities:**
  - Allow users to create new support tickets (linked to loans if applicable).
  - Retrieve tickets submitted by a specific user (via email or ID).
  - Fetch details of a ticket by ticket ID.
  - Ensure tickets are initialized with status = OPEN and empty response.

8. UserServiceImpl

- **Purpose:** Manages user accounts and profile updates.
- **Responsibilities:**
  - Retrieve all registered users.
  - Get user details by ID.
  - Update user details (e.g., income, credit score, contact info,role) with RBAC.
  - Delete users from the system.
  - Get the total amount paid by the user.

Repositories

- UserRepository
- LoanApplicationRepository
- RepaymentScheduleRepository
- SupportTicketRepository

Frontend (Vue.js)

Components

1. LandingPage.vue

- Entry point of the application.
- Provides a welcome screen and overview of features.
- Contains navigation links for **Login**, **Register**, and other key modules.
- Acts as a public-facing page before authentication.

2. HomeDashboard.vue

- Main dashboard for **logged-in users**.
- Displays an overview of:
    - Active loans and their statuses.
    - Total Amount paid
    - Borrowing capacity
    - Total amount to be paid
    - EMI summary (upcoming and overdue payments).
    - Recent support tickets.
- Provides quick links to apply for new loans, check EMI schedule, or raise tickets.

3. User LoginForm.vue

- Implements the **login functionality** for users and admins.
- Captures email and password input.
- Calls /auth/login API to authenticate users.
- Stores JWT token in Vuex/localStorage for secured requests.
- Redirects users to the correct dashboard based on role (USER → HomeDashboard, ADMIN → AdminDashboard).

4. UserRegistrationForm.vue

- Registration form for **new users**.
- Collects details like name, email, password, income, credit score, etc.
- Calls /auth/register API to create an account.
- Provides form validation and error handling (duplicate email, weak password).
- Redirects to the login page after successful registration.

5. UserManagement.vue

- Available only for **Admins**.
- Displays a list of all registered users.
- Provides options to:
    - View user details.
    - Update user roles.
    - View loan histories of each users
- Helps admins manage customer onboarding and maintain system integrity.

6. LoanForm.vue

- Allows users to **apply for a new loan**.
- Input fields: loan amount, tenure, income, credit score, purpose.
- Performs client-side validation (credit score in valid range).
- Submit loan request via /api/loans API.
- Shows confirmation once the loan is created with status = UNDER_REVIEW. (Only to the user, in the backend it will be PENDING)

7. LoanComponent.vue

- Displays a list of all loans applied by the user.
- Shows loan details such as **amount, tenure, status, and purpose**.
- Provides filters (by status/date).
- Allows navigation to **LoanDetail.vue** for more information.

8. LoanDetail.vue

- Shows full details of a specific loan application.
- Includes loan status (UNDER_REVIEW, APPROVED, REJECTED).
- Displays admin remarks (if any).
- Status labels are decided based on the user role.

9. EmiCalculator.vue

- Provides a form to preview EMI values before applying for a loan.
- Input: principal amount, tenure, interest rate.
- Uses EMI formula to calculate:
    - Monthly EMI amount.
    - Total interest payable.
    - Total repayment.
- Dynamically updates results on user input.

10. RepaymentSchedule.vue

- Displays detailed EMI schedule for **approved loans**.
- Shows month-wise breakdown:
    - EMI amount
    - Principal component
    - Interest component
    - Balance remaining
    - Payment status (Paid / Pending)
- Allows marking EMI as paid (via /api/loans/{loanId}/schedule/{scheduleId}/pay).

11. RaiseTicketView.vue

- Allows users to **raise support tickets**.
- Input fields: subject, description, ticket type, linked loan (optional).
- Calls /ticket/user API to create a new ticket.
- Initializes ticket with status = OPEN.

12. UserTicketsView.vue

- Displays list of all tickets raised by the logged-in user.
- Shows ticket details: subject, status, created date, response (if any).
- Allows filtering by status (OPEN, RESOLVED, CLOSED).
- Links to **TicketDetailsView.vue** for deeper view.

13. TicketDetailsView.vue

- Detailed view of a single support ticket.
- Displays: subject, description, type, status, createdAt, updatedAt.
- Shows admin's response if available.
- Updates dynamically when ticket status changes.

14. AdminTicketView.vue

- Ticket management interface for **Admins**.
- Displays all tickets in the system (with filters by status).
- Allows admin to:
    - View ticket details.
    - Add or update responses.
    - Change status (OPEN → IN_PROGRESS → RESOLVED/CLOSED).
- Ensures proper tracking of customer support requests.

15. AdminDashboard.vue

- Central dashboard for **Admins**.
- Displays system-wide metrics:
    - Pending loan applications.
    - Approved vs. Rejected loans.
    - Active support tickets.
    - Overall user statistics.
- Provides quick actions for reviewing loans and resolving tickets.

16. BarChart.vue & PieChart.vue

- **BarChart.vue**: Visualizes loan/EMI/ticket statistics using bar graphs.
- **PieChart.vue**: Visualizes distribution data (e.g., approved vs rejected loans, ticket statuses).
- Integrated within **AdminDashboard.vue** for quick insights.

Routing

Authentication:

- /login → UserLoginForm
- /register → UserRegistrationForm

User Section:

- /dashboard → HomeDashboard
- /loans → LoanComponent

- /loans/:id → LoanDetail
- /loan/apply → LoanForm
- /emi-calculator → EmiCalculator
- /repayment-schedule/:loanId → RepaymentSchedule
- /tickets → UserTicketsView
- /tickets/:id → TicketDetailsView
- /tickets/raise → RaiseTicketView

Admin Section:

- /admin/dashboard → AdminDashboard
- /admin/users → UserManagement
- /admin/tickets → AdminTicketView

State Management

Uses Vuex for:

- Managing authentication state (JWT tokens).
- Storing user details and roles.
- Managing loan data across components.
- Handling support tickets (user and admin).

API Integration

- **POST /auth/register**
  - **Frontend Component:** UserRegistrationForm.vue
  - **Purpose:** Registers a new user by sending name, email, password, income, credit score, etc.
  - **Response:** Returns newly created user details.
- **POST /auth/login**
  - **Frontend Component:** UserLoginForm.vue
  - **Purpose:** Authenticates user and returns JWT access + refresh token.
  - **Response:** Token + user details (used for routing based on role).
- **POST /auth/refresh**
  - **Frontend Component:** UserLoginForm.vue (token renewal logic)
  - **Purpose:** Refreshes access token using refresh token.
  - **Response:** New access token.
- **GET /users/{id}**
  - **Frontend Component:** UserLoginForm/UserRegistrationForm
  - **Purpose:** getCurrentUser.
  - **Response:** Current user object
- **POST /users**
  - **Frontend Component:** UserRegistrationForm
  - **Purpose:** getNewUser.

    - **Response:** new user object
- **PUT /users/{id}**
    - **Frontend Component:** UserManagementComponent
    - **Purpose:** updates the current user.
    - **Response:** updated user object
- **DELETE /users/{id}**
    - **Frontend Component:**
    - **Purpose:** deletes the current user by Id
    - **Response:** void
- **DELETE /users/paid/{id}**
    - **Frontend Component:** - HomeDashboard
    - **Purpose:** total amount paid by userId
    - **Response:** Get the double
- **POST /api/loans**
    - **Frontend Component:** LoanForm.vue
    - **Purpose:** Submits a new loan application.
    - **Response:** Loan ID + initial status (NEW or UNDER_REVIEW).
- **GET /api/loans/user/{userId}**
    - **Frontend Component:** LoanComponent.vue, ApplicationStatus.vue
    - **Purpose:** Fetches all loan applications for the logged-in user.
    - **Response:** List of loans with details and statuses.

- **GET /api/loans/{loanId}**
    - **Frontend Component:** LoanDetail.vue
    - **Purpose:** Fetches detailed information about a specific loan.
    - **Response:** Loan details including status, admin remarks, etc.

- **GET /admin/loans/pending**
    - **Frontend Component:** AdminDashboard.vue
    - **Purpose:** Retrieves all pending loan applications.
    - **Response:** List of pending loans for admin review.

- **PUT /admin/loans/{loanId}/status**
    - **Frontend Component:** AdminDashboard.vue
    - **Purpose:** Updates loan status to APPROVED or REJECTED with remarks.
    - **Response:** Updated loan details.

- **POST /api/emi/preview**
    - **Frontend Component:** EmiCalculator.vue
    - **Purpose:** Previews EMI calculation based on amount, rate, and tenure.
    - **Response:** EMI amount, total interest, total repayment.

- **GET /api/loans/{loanId}/schedule**

    - **Frontend Component:** RepaymentSchedule.vue
    - **Purpose:** Fetches repayment schedule for an approved loan.
    - **Response:** Month-wise EMI breakdown (principal, interest, balance).

- **POST /api/loans/{loanId}/generate-schedule**
    - **Frontend Component:** RepaymentSchedule.vue
    - **Purpose:** Generates repayment schedule once loan is approved.
    - **Response:** List of repayment schedule entries.

- **POST /api/loans/{loanId}/schedule/{scheduleId}/pay**
    - **Frontend Component:** RepaymentSchedule.vue
    - **Purpose:** Marks a specific EMI as paid.
    - **Response:** Confirmation message.

- **POST /ticket/user**
    - **Frontend Component:** RaiseTicketView.vue
    - **Purpose:** Creates a new support ticket.
    - **Response:** Ticket ID + status (OPEN).

- **GET /ticket/user/{userEmail}**
    - **Frontend Component:** UserTicketsView.vue
    - **Purpose:** Fetches all tickets raised by a user.
    - **Response:** List of tickets with status and responses.

- **GET /ticket/user/ticketId/{id}**
    - **Frontend Component:** TicketDetailsView.vue
    - **Purpose:** Fetches detailed information of a specific ticket.
    - **Response:** Ticket details including subject, description, status, response.

- **GET /ticket/admin**
    - **Frontend Component:** AdminTicketView.vue
    - **Purpose:** Retrieves all tickets in the system.
    - **Response:** List of tickets with filtering options.

- **GET /ticket/admin/status/{status}**
    - **Frontend Component:** AdminTicketView.vue
    - **Purpose:** Retrieves tickets filtered by status (OPEN, RESOLVED, CLOSED).
    - **Response:** List of filtered tickets.

- **PATCH /ticket/admin/{ticketId}/response**
    - **Frontend Component:** AdminTicketView.vue
    - **Purpose:** Adds or updates admin's response to a ticket.
    - **Response:** Updated ticket details with response.

- **PATCH /ticket/admin/{ticketId}/{status}**
    - **Frontend Component:** AdminTicketView.vue
    - **Purpose:** Updates ticket status (e.g., OPEN → RESOLVED).
    - **Response:** Updated ticket details with new status.

Database

1. **Users Table (users) -** Stores registered users and admins.

- id (PK) → Unique user ID
- name → Full name
- email → Unique email (used for login)
- password → Encrypted password
- role → Enum: USER, ADMIN
- created_at → Registration timestamp
- updated_at → Last update timestamp

Relationship:

- One user → Many loan applications (loans.user_id)
- One user → Many tickets (tickets.user_id)

2. **Loan Applications Table (loans) -** Stores loan details for each user.

- id (PK) → Unique loan ID
- user_id (FK → users.id) → Loan applicant
- loan_type → Enum: PERSONAL, HOME, EDUCATION, etc.
- amount → Requested loan amount
- income → Applicant's monthly income
- status → Enum: PENDING, APPROVED, REJECTED
- review_remarks → Admin remarks (optional)
- reviewed_by (FK → users.id, ADMIN)
- reviewed_at → Timestamp of review
- created_at → Loan application date

Relationship:

- One loan → One repayment schedule (repayment_schedule.loan_id)
- One loan → Many EMI records (emi.loan_id)

3. **Repayment Schedule Table (repayment_schedule) -** Stores high-level schedule details for each loan.

- id (PK) → Unique repayment schedule entry ID
- loan_id (FK → loan_applications.id) → Associated loan
- month → EMI month number (1…N)
- principal_amount → Monthly principal component
- interest_amount → Monthly interest component
- balance_remaining → Remaining principal after EMI payment
- is_paid → Boolean, marks EMI as paid (default false)

4. **Tickets Table (tickets) -** Stores support tickets raised by users.

- ticket_id (PK) → Unique ticket ID
- user_id (FK → users.id) → Ticket raised by user

- loan_id (FK → loan_applications.id, optional) → Related loan (if any)
- subject → Ticket subject
- description → Detailed issue description
- type → Enum: GENERAL_QUERY, LOAN_RELATED, REPAYMENT etc.
- status → Enum: OPEN, IN_PROGRESS, RESOLVED, CLOSED (default OPEN)
- create_at → Ticket creation timestamp
- updated_at → Last updated timestamp
- response → Admin's response message

Test Plan

Unit Tests :

- AuthService / JwtService
    - Register: validation, password hashing, duplicate email.
    - Login: valid credentials produce token; invalid credentials throw exception.
    - Refresh token logic (verify refresh token claims).
- UserService
    - CRUD operations, patch/update behavior, validation rules.
- LoanApplicationService / LoanService
    - Business rules:
        - income > ₹25,000 validation.
        - creditScore is in [300,900].
        - Duplicate loan within 24 hours prevented.
    - addLoanApplication() behavior: persists proper loan and sets rate from purpose.
- RepaymentScheduleService / EmiCalculator
    - EMI formula correctness (compare known inputs/outputs).
    - generateSchedule() produces month-by-month principal/interest/balance and isPaid=false.
- AdminLoanService
    - updateStatus: sets reviewedBy, reviewedAt, reviewRemarks and correct status transitions.
- SupportTicketService
    - Create ticket validation, update response, update status transitions.

Tools & annotations

- JUnit 5 (org.junit.jupiter.*)
- Mockito (@Mock, @InjectMocks, when, verify.

Integration Tests :

- Validate REST controllers + service + repository working together.
- Validate security filters / JWT integration where applicable.
- Validate persistence behavior (DB writes/reads).

- Validate important flows:
    - Register → Login → get JWT → call secured APIs.
    - Apply for loan (POST /api/loans) → verify loan saved → admin approves (PUT /admin/loans/{id}/status) → generate schedule (POST /api/loans/{id}/generate-schedule).
    - Preview EMI (POST /api/emi/preview).
    - Create ticket and admin response flows.

Tools & annotations

- @SpringBootTest (boots entire context) for controller slice + mocked services.
- @DataJpaTest for repository-level tests.

Test Cases

Unit Tests

Testing individual methods/services in isolation using JUnit and Mockito.

1. User Registration Service
    - Scenario: Register a new user with valid data.
    - Input Data:
        - Name: "Aanchal"
        - Email: "aanchal@example.com"
        - Password: "password123"
        - Role: "USER"
    - Expected Result:
        - User saved successfully in DB.
        - Response: User registered successfully.
        - Password is encrypted before saving.

2. User Registration – Duplicate Email
    - Scenario: Try to register with an already existing email.
    - Input Data:
        - Email: "aanchal@example.com" (already in DB)
    - Expected Result:
        - Service throws UserAlreadyExistsException.
        - No duplicate entry created in DB.

3. Loan Application Creation
    - Scenario: User submits a new loan application.
    - Input Data:
        - User ID: 101

    - Amount: 200000
    - Tenure: 24 months
    - Income: 50000
    - Credit Score: 720
- Expected Result:
    - Loan application saved with status = NEW.
    - Application date is auto-generated.

4. Loan Status Update (Admin)
- Scenario: Admin reviews loan and approves it.
- Input Data:
    - Loan ID: 301
    - New Status: "APPROVED"
    - Remarks: "Verified and approved."
- Expected Result:
    - Loan status updated in DB.
    - ReviewedBy field updated with Admin ID.
    - Response message: "Loan approved successfully."

5. Repayment Schedule Calculation
- Scenario: Generate repayment schedule for an approved loan.
- Input Data:
    - Loan ID: 301
    - Amount: 100000
    - Interest: 10%
    - Tenure: 12 months
- Expected Result:
    - 12 repayment rows generated.
    - Each row contains principal, interest, remaining balance.
    - Total principal matches loan amount.

6. Ticket Creation (User)

- Scenario: User raises a support ticket for repayment issue.
- Input Data:
    - User ID: 101
    - Subject: "Issue in repayment"
    - Description: "Payment not reflecting"
    - Type: "REPAYMENT"
- Expected Result:
    - Ticket saved with status = OPEN.

- CreatedAt auto-populated.
- Response = empty string initially.

Integration Tests

1. User Login API - Endpoint: POST /auth/login

Input Data:
{ "email": "aanchal@example.com", "password": "password123" }

Expected Result:

- Response contains JWT token.
- Status = 200 OK.

2. User Login : Invalid Credentials - Endpoint: POST /auth/login

Input Data:
{ "email": "aanchal@example.com", "password": "wrongpass" }

Expected Result:

- Response = "Invalid credentials".
- Status = 401 Unauthorized.

3. Apply for Loan API - Endpoint: POST /api/loans

Input Data:
{ "amount": 250000, "tenureMonths": 36, "income": 60000, "creditScore": 750 }

Expected Result:

- Loan saved in DB with status = NEW.
- Response contains loan ID.
- Status = 201 Created.

4. Get Loan Details API - Endpoint: GET /api/loans/{id}

Input Data:
Loan ID: 301

Expected Result:

- Returns loan details (amount, tenure, status, user).

- Status = 200 OK.

5. Generate Repayment Schedule API - Endpoint: POST /api/loans/{loanId}/generate-schedule

Input Data:

Loan ID: 301

Expected Result:

- Response contains EMI breakdown for each month.
- DB contains rows in repayment_schedule.
- Status = 201 Created.

2.6 Ticket Management API - Endpoint: POST /api/tickets

Input Data:
{ "userId": 101, "subject": "Payment not updating", "description": "EMI paid but not reflected", "type": "REPAYMENT" }

Expected Result:

- Ticket saved in DB with status = OPEN.
- Response includes ticket ID.
- Status = 201 Created.

SETUP AND CONFIGURATION
Backend Setup

Prerequisites

- Java 17+ (Project is built with Spring Boot, requires Java 17 or above)
- Maven 3.6+ (or use the included Maven Wrapper mvnw)
- IDE: IntelliJ IDEA / Eclipse / VS Code (optional, for development)
- MySQL 8.0+ (or compatible version)
- MySQL Workbench / CLI (optional for management)

Steps to Run the Backend

1. Clone the repository:
   git clone https://github.com/ektagoel-12/Personal-Loan-Application-System.git
   cd Personal-Loan-Application-System/backend

2. Verify dependencies:
   mvn clean install
3. Run the backend:
   mvn spring-boot:run

   Backend will start on: http://localhost:8080

Frontend Setup

Prerequisites

- Node.js v16+
- npm (Node Package Manager)
- Vue CLI (if not installed globally, use npx)

Steps to Run the Frontend

1. Navigate to frontend folder:
   cd Personal-Loan-Application-System/frontend
2. Install dependencies:
   npm install
3. Run the Vue development server:
   npm run serve

   Frontend will start on: http://localhost:5173

Database Setup

Steps to Configure Database

1. Create a new database in MySQL:
   CREATE DATABASE loan_application_db;
2. Update **application.properties** (or application.yml) in backend:
   spring.datasource.url=jdbc:mysql://localhost:3306/loan_application_db
   spring.datasource.username=root
   spring.datasource.password=your_password
   spring.jpa.hibernate.ddl-auto=update
   spring.jpa.show-sql=true
   spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
3. Run the backend again (mvn spring-boot:run).
   Hibernate will auto-generate tables (users, loan_applications, repayment_schedule, tickets) based on entities.
4. Verifying the Setup

- Backend: Open http://localhost:8080/actuator/health → should return {"status":"UP"}

- Frontend: Open http://localhost:5173 → Vue app should load.
- Database: Check tables in loan_application_db → should be auto-created.

API DOCUMENTATION

AuthController (/auth)

Method	Endpoint	Request Body	Response	Description
GET	/auth	–	"Server is up and running"	Health check
POST	/auth/register	RegisterRequest (name, email, income, creditScore, password, aadhar)	AuthResponse	Register a new user
POST	/auth/login	LoginRequest (email, password)	AuthResponse	Login and get tokens
POST	/auth/refresh	RefreshTokenRequest	TokenResponse	Refresh JWT access token

UserController (/users)

Method	Endpoint	Request Body	Response	Description
GET	/users	–	List<User>	Get all users
GET	/users/{id}	–	User / Error	Get user by ID
GET	/users/paid/{id}	-	200 OK	Total Paid
PATCH	/users/{id}	UpdateUserRequest	User / Error	Update user details
DELETE	/users/{id}	–	200 OK	Delete user by ID

AdminController (/admin)

Method	Endpoint	Request Body	Response	Description
GET	/admin	–	AdminDto	Get admin dashboard stats
GET	/admin/loans/{id}	–	AdminLoansList	Get loan by ID
PUT	/admin/loans/{id}/status	UpdateLoanStatus	AdminLoansList	Approve/Reject loan

LoanApplicationController (/api/loans)

Method	Endpoint	Request Body	Response	Description
GET	/api/loans	–	List<LoanApplicationResponse>	Get all loan applications
GET	/api/loans/{id}	–	LoanApplicationResponse	Get loan by ID
GET	/api/loans/user/{userId}	–	List<LoanApplicationResponse>	Get all loans for a user
POST	/api/loans	LoanApplicationResponse	LoanApplicationResponse	Create new loan
PUT	/api/loans/{id}	LoanApplication	Not implemented	Update loan
DELETE	/api/loans/{id}	–	Not implemented	Delete loan

RepaymentScheduleController (/api)

Method	Endpoint	Request Body	Response	Description
POST	/api/emi/preview	EmiPreviewRequest	EmiPreviewResponse	Preview EMI before loan approval
GET	/api/loans/{loanId}/schedule	–	List<RepaymentScheduleDTO>	Get repayment schedule
POST	/api/loans/{loanId}/generate-schedule	–	List<RepaymentSchedule>	Generate schedule (only for APPROVED loans)
POST	/api/loans/{loanId}/schedule/{scheduleId}/pay	–	"EMI with id {scheduleId} has been paid."	Mark EMI as paid

SupportTicketUserController (/ticket/user)

Method	Endpoint	Request Body	Response	Description
POST	/ticket/user	SupportTicketRequestDto	SupportTicketResponseDto	Create support ticket
GET	/ticket/user/email/{userEmail}	–	List<SupportTicketResponseDto>	Get tickets by user email
GET	/ticket/user/ticketId/{id}	–	SupportTicket	Get ticket by ID

SupportTicketAdminController (/ticket/admin)

Method	Endpoint	Request Body	Response	Description
GET	/ticket/admin	–	List<SupportTicketResponseDto>	Get all support tickets
GET	/ticket/admin/ticketId/{id}	–	SupportTicket	Get ticket by ID
GET	/ticket/admin/status/{status}	–	List<SupportTicketResponseDto>	Get tickets by status (OPEN, PENDING, CLOSED)
PATCH	/ticket/admin/{ticketId}/response	SupportTicketResponseBodyDto	SupportTicketResponseDto	Add/Update response for ticket
PATCH	/ticket/admin/{ticketId}/{status}	–	SupportTicketResponseDto	Update ticket status

Example Request & Responses:

1. User Management & Authentication

Endpoint: /api/auth/register

Request Body:

```
{
 "name": "Dilip Malani",
 "email": "dilip.malani@example.com",
 "password": "password123",
 "role": "USER"
}
```

Response Body:

```
{
 "id": 1,
```

```
 "name": "Dilip Malani",
 "email": "dilip.malani@example.com",
 "role": "USER"
}
```


Endpoint: /api/auth/login
Request Body:


```
{
 "email": "dilip.malani@example.com",
 "password": "password123"
}
```


Response Body:


```
{
 "token": "eyJhbGciOiJIUzI1NiIsInR...",
 "user": {
  "id": 1,
  "name": "Dilip Malani",
  "email": "dilip.malani@example.com",
  "role": "USER"
 }
}
```


2. Loan Application

Endpoint: /api/loans/apply
Request Body:


```
{
 "userId": 1,
 "amount": 500000,
 "income": 60000,
 "creditScore": 750,
 "tenureMonths": 24,
 "purpose": "Personal Loan"
}
```


Response Body:


```
{
 "id": 101,
 "userId": 1,
 "amount": 500000,
 "income": 60000,
 "creditScore": 750,
```

```
  "tenureMonths": 24,
  "status": "UNDER_REVIEW",
  "applicationDate": "2025-09-30",
  "purpose": "Personal Loan"
}
```


Endpoint: /api/loans/user/{userId}

Response Body:


```
[
  {
    "id": 101,
    "amount": 500000,
    "status": "APPROVED"
  },
  {
    "id": 102,
    "amount": 200000,
    "status": "REJECTED"
  }
]
```


3. Admin Loan Review

Endpoint: /api/admin/loans/pending

Response Body:


```
[
  {
    "id": 101,
    "userId": 1,
    "amount": 500000,
    "income": 60000,
    "creditScore": 750,
    "status": "UNDER_REVIEW"
  }
]
```


Endpoint: /api/admin/loans/{loanId}/status

Request Body:


```
{
  "status": "APPROVED",
  "reviewRemarks": "Approved based on eligibility"
}
```


Response Body:

```
{
  "id": 101,
  "status": "APPROVED",
  "reviewedBy": "admin1",
  "reviewRemarks": "Approved based on eligibility",
  "reviewedAt": "2025-09-30T12:34:56"
}
```


4. EMI Calculator

Endpoint: /api/emi/preview

Request Body:


```
{
  "amount": 500000,
  "tenureMonths": 24,
  "interestRate": 12.0
}
```


Response Body:


```
{
  "emi": 23537.95,
  "totalInterest": 56510.80,
  "totalPayment": 556510.80
}
```


5. Support Ticket System

Endpoint: /api/support

Request Body:


```
{
  "userId": 1,
  "loanId": 101,
  "subject": "Loan Status Inquiry",
  "description": "When will my loan get approved?",
  "type": "GENERAL"
}
```


Response Body:


```
{
  "id": 501,
```

```
  "userId": 1,
  "loanId": 101,
  "subject": "Loan Status Inquiry",
  "description": "When will my loan get approved?",
  "status": "OPEN",
  "createdAt": "2025-09-30T14:20:00"
}
```


DEPLOYMENT

Deployment Environment

- **Type:** Local Development Deployment
- **Operating System:** macOS
- **Backend:** Spring Boot (running on http://localhost:8080)
- **Frontend:** Vue.js (running on http://localhost:5173)
- **Database: PostgreSQL** (running locally on localhost:5432)
- **Access Scope:** Accessible only on the local machine (no staging/production environment).

Deployment Steps

- **Open Frontend:** http://localhost:5173
- **Frontend** interacts with Backend APIs at http://localhost:8080.
- Data is persisted in local **PostgreSQL DB (loan_application_db).**

FUTURE ENHANCEMENTS

The current implementation of the **Personal Loan Application System** provides core functionality for loan applications, approvals, EMI calculations, and support ticket handling. To make the application more scalable, user-friendly, and closer to real-world enterprise-grade systems, the following future improvements can be considered:

1. Enhanced Security

- Implement **multi-factor authentication (MFA)** for secure logins.
- Add **role-based permission levels** (e.g., Super Admin, Loan Officer, Customer Service Agent).
- Integrate **OAuth2 / Single Sign-On (SSO)** for enterprise environments.

2. Loan Processing Enhancements

- Introduce **automated loan scoring engine** using AI/ML to evaluate creditworthiness.
- Support for **multiple loan types** (Car Loan, Home Loan, Education Loan, Business Loan).
- Add **document upload & verification system** for KYC and loan proof submission.
- Implement **auto-notifications** (SMS/Email) for loan approval/rejection updates.

3. EMI & Payment System Improvements

- Integrate with **real payment gateways** (Razorpay, Stripe, PayPal, UPI) for EMI payments.
- Add **payment reminders & notifications** via email/SMS before due dates.
- Support **prepayment/foreclosure options** with recalculated schedules.
- Add **penalty calculation** for late EMI payments.

4. Customer Support Enhancements

- Enable **live chat support** integrated into the frontend.
- Add **ticket categorization & prioritization** (High, Medium, Low).
- Introduce **SLA tracking** for tickets to monitor resolution times.
- Add **chatbot/AI assistant** to handle FAQs and basic support queries automatically.

5. Admin & Analytics Features

- Build **advanced dashboards** with real-time analytics (loan defaults, repayment trends).
- Add **export functionality** (CSV/PDF) for reports.
- Support **drill-down visualizations** in bar/pie charts.
- Add **system health monitoring** (API uptime, DB status).

TEAM AND ROLES

1. Aom Wankhede – Module 1: User Management & Authentication

- Implemented **User entity and repository** with role-based access (USER, ADMIN).
- Built authentication APIs (/auth/register, /auth/login, /auth/refresh) with JWT-based security.
- Developed frontend forms (UserLoginForm.vue, UserRegistrationForm.vue) for user onboarding.
- Implemented role-based routing (User → Dashboard, Admin → AdminDashboard).
- Wrote unit tests for authentication and registration logic.

2. Krishna Koushik – Module 2: Loan Application Submission & Status Tracking

- Designed backend APIs for loan application submission and retrieval (/api/loans).

- Implemented **business rules** for income validation, credit score checks, and duplicate submission prevention.
- Developed frontend forms (LoanForm.vue, ApplicationStatus.vue, LoanComponent.vue) for applying and tracking loan applications.
- Integrated APIs to display loan statuses dynamically.

3. Aanchal Maheshwari – Module 3: Admin Dashboard & Loan Processing Engine

- Implemented **Admin APIs** for reviewing loan applications and updating statuses (/admin/loans).
- Added entity updates to include review remarks, reviewedBy, and reviewedAt fields.
- Built **Admin Dashboard frontend** (AdminDashboard.vue) to list pending/approved/rejected loans with action buttons.
- Wrote test cases for loan approval/rejection logic.

4. Ekta Goel – Module 4: EMI Calculator

- Implemented the **EMI calculation logic** using the standard EMI formula.
- Developed backend service to generate **repayment schedules** (principal, interest, balance).
- Built frontend components (EmiCalclator.vue, RepaymentSchedule.vue) to display EMI previews and repayment schedules.
- Wrote unit tests for EMI calculation service.

5. Premansu Chakraborty – Module 5: Customer Support Ticketing System

- Designed backend APIs for ticket creation, user ticket retrieval, and admin ticket management (/ticket/user, /ticket/admin).
- Implemented entities and services for support ticket lifecycle management.
- Developed frontend components (RaiseTicketView.vue, UserTicketsView.vue, TicketDetailsView.vue, AdminTicketView.vue) for ticket handling.
- Added admin features to update ticket responses and statuses.

APPENDIX

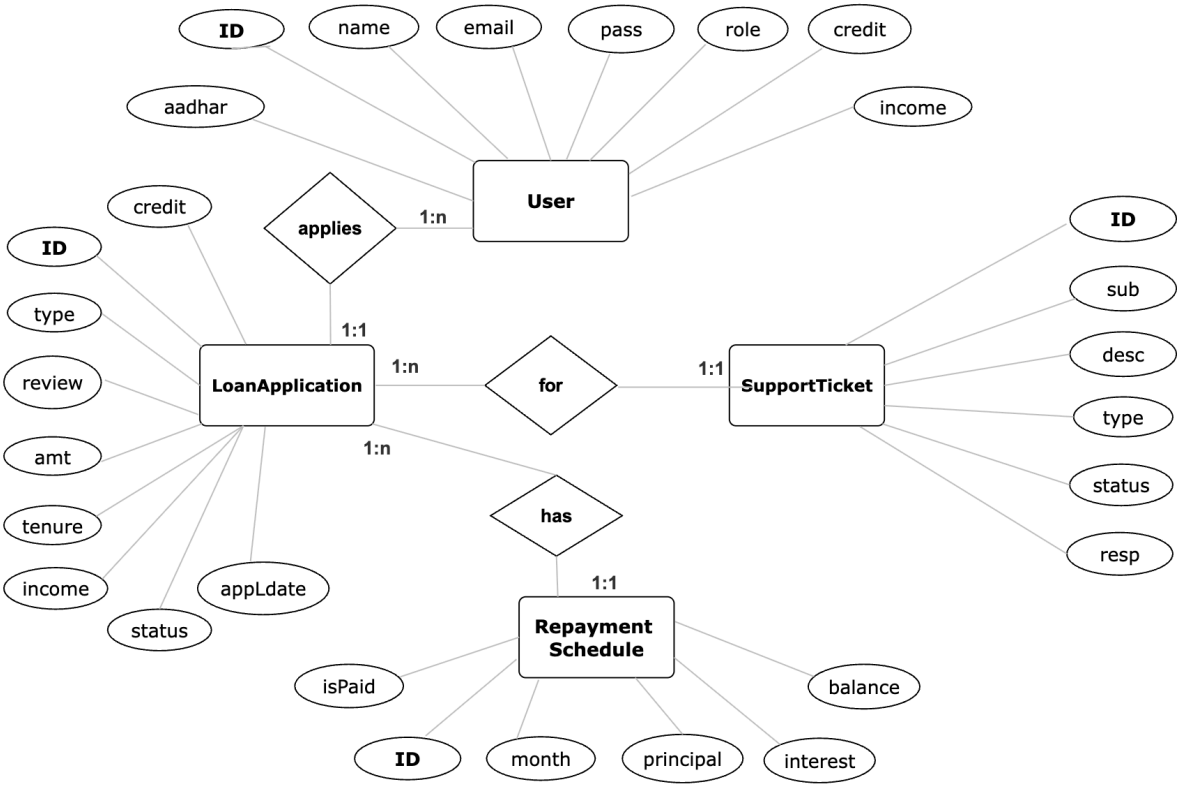

Fig.1:ER Diagram

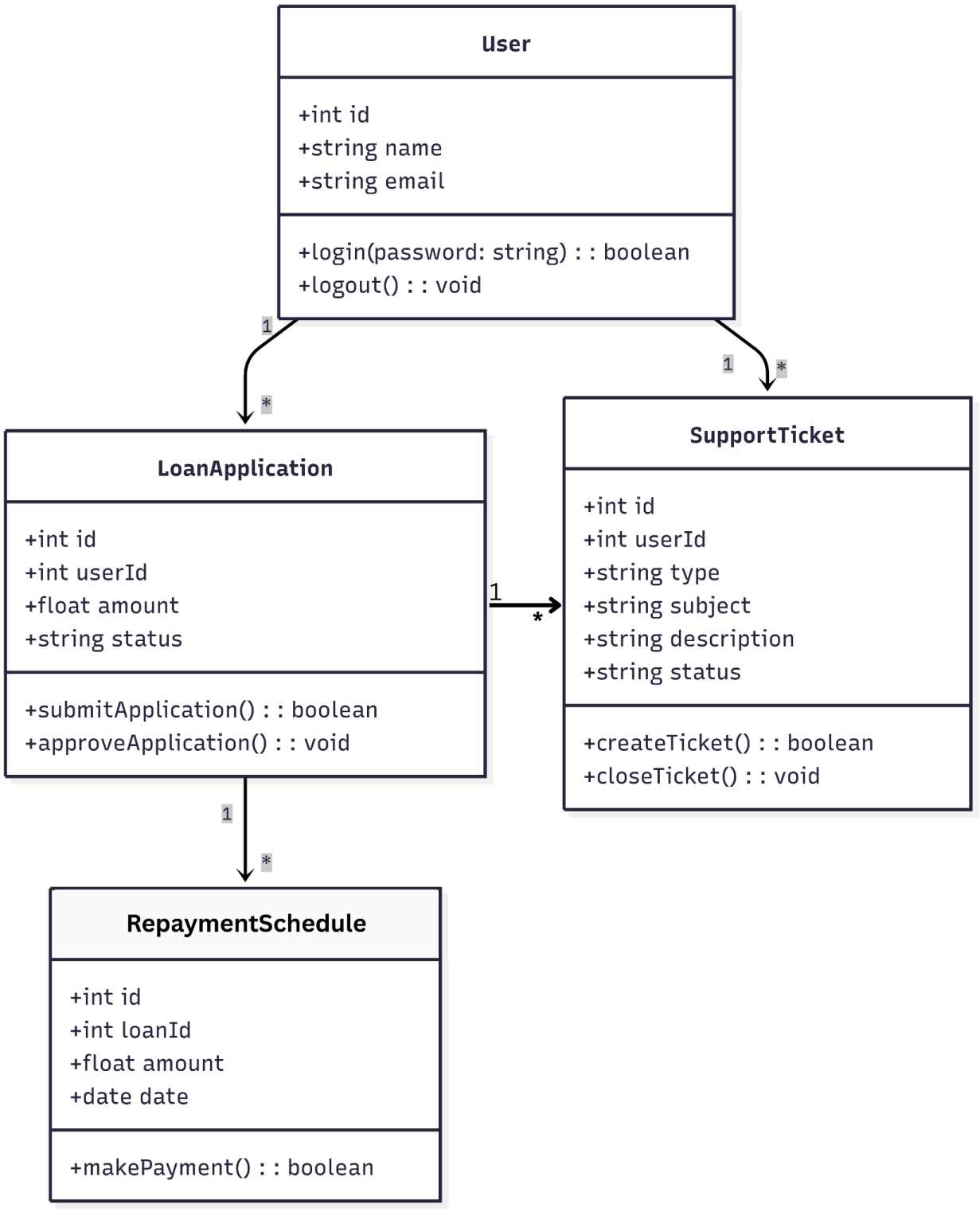

Fig.2:Class Diagram