

# Credit Card Management Portal

## Zeta - Software Development Foundation Program

Abhishekh (Team Lead), Akuthota Sravani, Aravind P, Pothuri Tejaswi, Ashritha Sadu

# Contents

<b>1</b>	<b>Project Overview</b>	<b>4</b>
1.1	Project Title .....	4
1.2	Project Goal .....	4
1.3	Technologies Used.....	4
1.4	Key Features .....	5
1.5	Buy Now Pay Later (BNPL) Feature .....	5
1.6	Target Audience.....	6
1.7	Repository.....	6
<b>2</b>	<b>System Architecture</b>	<b>6</b>
2.1	Backend (Spring Boot) .....	6
2.2	Frontend (Vue.js).....	7
2.3	Database .....	7
2.4	Containerization.....	7
<b>3</b>	<b>Schema Design</b>	<b>8</b>
3.1	User Table .....	8
3.2	Card Table .....	8
3.3	Card Applications Table .....	9
3.4	Transactions Table.....	9
3.5	User Profiles Table .....	9
3.6	BNPL Installments Table .....	10
3.7	Relationships .....	10
<b>4</b>	<b>Setup and Configuration</b>	<b>11</b>
4.1	Backend Setup .....	11
4.2	Frontend Setup.....	11
4.3	Database Setup .....	11
4.4	Docker Setup .....	13
<b>5</b>	<b>API Documentation</b>	<b>14</b>
5.1	Endpoints.....	14
5.2	Example Request/Response .....	14
<b>6</b>	<b>Test Plan</b>	<b>15</b>
6.1	Unit Tests.....	15
6.2	Integration Tests .....	16
6.3	End-to-End Tests .....	16
6.4	Test Cases .....	16
<b>7</b>	<b>Implementation Details</b>	<b>17</b>
7.1	Module 1: View Credit Cards and Activate/Block .....	17
7.2	Module 2: Apply for Credit Card and View Requests .....	17
7.3	Module 3: Manage Card Limit .....	17
7.4	Module 4: Simulate and View Transactions .....	17
7.5	Module 5: Customer Profile Management.....	18
7.6	BNPL Feature Implementation .....	18

<b>8</b>	<b>User Interface Design</b>	<b>18</b>
----------	------------------------------	-----------

8.1	Design Principles .....	18
8.2	Key Screens .....	18
<b>9</b>	<b>Security Considerations</b>	<b>19</b>
9.1	Data Protection .....	19
9.2	API Security .....	19
9.3	Future Security Enhancements .....	19
<b>10</b>	<b>Performance Optimization</b>	<b>19</b>
10.1	Backend Optimization .....	19
10.2	Frontend Optimization.....	19
10.3	Database Optimization.....	19
<b>11</b>	<b>Deployment</b>	<b>20</b>
11.1	Deployment Environment.....	20
11.2	Deployment Steps.....	20
<b>12</b>	<b>Troubleshooting</b>	<b>20</b>
12.1	Common Issues .....	20
<b>13</b>	<b>Future Enhancements</b>	<b>20</b>
<b>14</b>	<b>Team and Roles</b>	<b>21</b>
14.1	Team Members and Module Ownership.....	21
14.2	Additional Contributions .....	21
<b>15</b>	<b>Appendix</b>	<b>21</b>
15.1	Additional Resources.....	21
15.2	System Diagram .....	21

# 1 Project Overview

## 1.1 Project Title

Credit Card Management Portal

## 1.2 Project Goal

The Credit Card Management Portal is a sophisticated web-based application designed for a financial institution to enable customers to manage their credit cards and Buy Now Pay Later (BNPL) plans with ease. The portal provides a secure, intuitive, and responsive platform that supports a wide range of functionalities, including viewing and managing card details, applying for new credit cards, activating or blocking cards, adjusting credit limits, simulating transactions (including BNPL options), and generating detailed transaction reports. The primary goal is to enhance customer satisfaction by offering a seamless and efficient credit management experience while ensuring robust security, scalability, and compliance with banking standards. By integrating modern financing options like BNPL, the portal aligns with contemporary consumer needs, fostering financial flexibility and convenience.

## 1.3 Technologies Used

The application leverages a modern technology stack to ensure performance, maintainability, and scalability:

- **Backend:**

- Spring Boot (Java 17): Provides a robust framework for building RESTful APIs, enabling rapid development and easy integration.
- REST APIs: Facilitate communication between frontend and backend, ensuring modularity and scalability.
- Maven: Manages dependencies and streamlines the build process.

- **Frontend:**

- Vue.js 3 (Composition API): Offers a reactive and component-based framework for building a dynamic user interface.
- Tailwind CSS: Enables rapid styling with utility-first classes, ensuring a responsive and visually appealing design.
- Vite: A fast build tool that optimizes frontend development and deployment.
- JavaScript: Enhances interactivity and client-side logic.

- **Database:**

- MySQL 8.x: A reliable and widely-used relational database, chosen for its performance, scalability, and compatibility with Spring Boot via JPA.

- **Testing:**
  - JUnit: Ensures backend logic reliability through unit and integration tests.
  - Vitest: Tests frontend components for correctness and performance.
  - Cypress: Validates end-to-end user workflows, simulating real-world usage.
- **Containerization:**
  - Docker: Containerizes application components for consistent deployment.
  - Docker Compose: Orchestrates multi-container setups, simplifying development and production environments.
- **Version Control:**
  - Git: Manages source code versioning and collaboration.

## 1.4 Key Features

The portal offers a comprehensive set of features tailored to customer needs:

- **Card Management:** View detailed card information (number, status, limit, expiry) and perform actions like activation or blocking.
- **Card Applications:** Submit applications for new credit cards with customizable options (VISA, Mastercard, Amex) and track application status.
- **Limit Adjustments:** Modify credit limits within bank-defined constraints, providing financial flexibility.
- **Transaction Simulation:** Simulate transactions, including BNPL options, with customizable installment plans (THREE, SIX, NINE months).
- **Transaction Reports:** Access detailed transaction histories with filtering capabilities for regular and BNPL transactions.
- **Profile Management:** Maintain and update customer profiles with validated personal and financial information.
- **BNPL Management:** Create, manage, and pay BNPL installments, with automated late fee calculations for overdue payments.

- **Transaction Creation:** Users can initiate BNPL transactions by selecting an installment plan during checkout.
- **Installment Management:** Create, update, pay, or delete installments, with clear visibility of due dates and amounts.
- **Late Fee Tracking:** Automatically calculates and tracks late fees for overdue installments, ensuring transparency.
- **Eligibility Checks:** Validates user eligibility based on credit history and card limits, minimizing risk.

This feature not only improves customer retention by offering payment flexibility but also aligns with industry trends, positioning the bank as a forward-thinking financial institution.

## 1.5 Target Audience

The portal serves a diverse user base:

- **Customers:** Individuals seeking to manage their credit cards and BNPL plans efficiently.
- **Bank Staff:** Support teams responsible for reviewing card applications and assisting customers.
- **Administrators:** IT personnel tasked with system maintenance, monitoring, and updates.

## 1.6 Repository

The source code is hosted at: <https://github.com/Abhishekhzeta/credit-card-management-portal>

The repository includes detailed documentation, setup instructions, and contribution guidelines to facilitate collaboration and maintenance.

# 2 System Architecture

## 2.1 Backend (Spring Boot)

The backend is built using Spring Boot, a Java-based framework that simplifies the development of production-ready applications. It follows a layered architecture to ensure modularity and maintainability:

- **REST API Endpoints:** Expose endpoints for card management, transaction processing, BNPL operations, and user profile updates, adhering to RESTful principles.
- **Data Models (Entities):** JPA entities (e.g., User, Card, Transaction, BNPLInstallment) map to database tables, ensuring seamless data persistence.
- **Service Layer:** Encapsulates business logic, such as validating card applications, processing transactions, and calculating BNPL installments.

- **Repository Layer:** Spring Data JPA repositories provide CRUD operations and custom queries for efficient database interactions.
- **Configuration:** Application properties configure database connections (MySQL), CORS policies, and environment-specific settings.

The backend is designed for scalability, with features like connection pooling and asynchronous processing to handle high transaction volumes.

## 2.2 Frontend (Vue.js)

The frontend, developed with Vue.js 3 (Composition API), delivers a dynamic and responsive user experience. Key components include:

- **Reusable Components:** Modular components (e.g., `CardList.vue`, `ApplyCard.vue`, `TransactionHistory.vue`) promote code reuse and maintainability.
- **Routing:** Vue Router manages navigation between views, ensuring a single-page application (SPA) experience.
- **State Management:** Pinia centralizes application state, simplifying data sharing across components.
- **API Integration:** Axios handles HTTP requests to the backend, with error handling and retry mechanisms for reliability.
- **Styling:** Tailwind CSS provides utility-first styling, enabling rapid development of responsive and accessible UI elements.

The frontend prioritizes usability, with intuitive layouts and real-time feedback to enhance user engagement.

## 2.3 Database

MySQL 8.x serves as the relational database, chosen for its robust performance, wide adoption, and compatibility with Spring Boots JPA implementation. The schema is optimized for efficient querying and data integrity, with tables for users, cards, applications, transactions, profiles, and BNPL installments. MySQLs support for transactions, indexing, and foreign key constraints ensures data consistency and scalability. The schema design is detailed in Section 3.

## 2.4 Containerization

Docker and Docker Compose streamline deployment by containerizing the frontend, backend, and database. This approach ensures environment consistency, simplifies scaling, and facilitates CI/CD integration. Docker Compose orchestrates multi-container setups, defining services, networks, and dependencies.

## 3 Schema Design

The MySQL database schema is meticulously designed to support all application functionalities, including BNPL. Each table is optimized for performance, with appropriate data types, constraints, and indexes to ensure efficient querying and data integrity. Below are the key tables, their fields, and their roles in the system.

### 3.1 User Table

Stores customer information for authentication and eligibility checks.

Field	Type	Description
id	LONG	Primary key, auto-incremented for unique identification.
name	VARCHAR(255)	Customers full name, required for profile display.
email	VARCHAR(255)	Unique email for login and communication, validated for format.
phoneNumber	VARCHAR(20)	Optional contact number for notifications and support.
address	TEXT	Residential address for credit verification and billing.
isEligibleForBNPL	BOOLEAN	Flag indicating BNPL eligibility, determined by credit history (default: FALSE).

### 3.2 Card Table

Manages credit card details and their association with users.

Field	Type	Description
id	LONG	Primary key, uniquely identifies each card.
cardNumber	VARCHAR(16)	Unique 16-digit card number, masked in UI for security.
cardType	VARCHAR(50)	Card brand (e.g., VISA, Mastercard, Amex), used for application processing.
status	VARCHAR(20)	Card state (ACTIVE, BLOCKED, INACTIVE), controls usability.
creditLimit	DOUBLE	Maximum allowed credit, set by bank policies.
availableLimit	DOUBLE	Remaining spendable credit, updated after transactions.
expiryDate	DATE	Card expiration date, validated during transactions.
userId	LONG	Foreign key linking to User, establishing ownership.

### 3.3 Card Applications Table

Tracks applications for new credit cards, including approval status.

Field	Type	Description
id	LONG	Primary key for application tracking.
userId	LONG	Foreign key to User, identifying the applicant.
cardType	VARCHAR(50)	Requested card type (e.g., VISA, Mastercard).
requestedLimit	DOUBLE	Desired credit limit, subject to bank approval.
applicationDate	DATE	Submission date, used for processing timelines.
status	VARCHAR(20)	Application state (PENDING, APPROVED, REJECTED).
reviewedBy	VARCHAR(255)	Optional field for reviewers name or ID.
reviewDate	DATE	Optional date of review completion.

### 3.4 Transactions Table

Records all card transactions, including BNPL transactions.

Field	Type	Description
id	LONG	Primary key for transaction records.
cardId	LONG	Foreign key to Card, linking transaction to a card.
merchantName	VARCHAR(255)	Name of the merchant or service provider.
amount	DOUBLE	Transaction amount, validated against available limit.
transactionDate	DATE	Date of transaction, used for reporting.
category	VARCHAR(50)	Transaction type (e.g., Food, Travel), for filtering.
isBNPL	BOOLEAN	Indicates if the transaction uses BNPL (default: FALSE).

### 3.5 User Profiles Table

Stores detailed customer profiles for credit assessment and personalization.

Field	Type	Description
id	LONG	Primary key for profile records.
userId	LONG	Foreign key to User, ensuring one profile per user.

fullName	VARCHAR(255)	Complete name for official records.
email	VARCHAR(255)	Contact email, synchronized with User table.
phone	VARCHAR(20)	Contact number for communication.
address	TEXT	Detailed address for verification.
annualIncome	DOUBLE	Income data for credit eligibility checks.
createdAt	TIMESTAMP	Profile creation timestamp for audit trails.
updatedAt	TIMESTAMP	Last update timestamp for tracking changes.

### 3.6 BNPL Installments Table

Manages BNPL installment plans and payment statuses.

Field	Type	Description
id	LONG	Primary key for installment records.
transactionId	LONG	Foreign key to Transaction, linking to a BNPL transaction.
installmentNumber	INTEGER	Sequence number (e.g., 1, 2, 3) for tracking.
amount	DOUBLE	Installment amount, calculated from transaction total.
dueDate	DATE	Payment due date, used for late fee calculations.
isPaid	BOOLEAN	Payment status (default: FALSE).
lateFee	DOUBLE	Fee for overdue payments (default: 0.0).

### 3.7 Relationships

The schema enforces data integrity through relationships:

- **User to Card:** One-to-Many, as a user can own multiple cards.
- **User to CardApplication:** One-to-Many, allowing multiple applications per user.
- **Card to Transaction:** One-to-Many, as a card can have multiple transactions.
- **User to UserProfile:** One-to-One, ensuring a single profile per user.
- **Transaction to BNPL Installment:** One-to-Many, as a BNPL transaction can have multiple installments.

Indexes on foreign keys (e.g., userId, cardId) and frequent query fields (e.g., email, cardNumber) optimize performance.

## 4 Setup and Configuration

### 4.1 Backend Setup

The backend requires a Java environment and Maven for dependency management. **Prerequisites:**

- Java JDK 17 or later
- Maven 3.8.x or later

**Steps:**

1. Clone the repository: `git clone https://github.com/Abhishekhzeta/credit-card-management-portal`
2. Navigate to the backend directory: `cd backend`
3. Install dependencies: `mvn install`
4. Configure MySQL connection in `application.yml` (e.g., URL, username, password)
5. Run the application: `./mvnw spring-boot:run`

### 4.2 Frontend Setup

The frontend requires Node.js for development and build processes. **Prerequisites:**

- Node.js v18.x or later
- npm

**Steps:**

1. Navigate to the frontend directory: `cd frontend`
2. Install dependencies: `npm install`
3. Update API base URL in `.env` to point to the backend (e.g., `http://localhost:8080`)
4. Run the development server: `npm run dev`

### 4.3 Database Setup

MySQL 8.x is used for data storage, with a schema tailored for the application. **Prerequisites:**

- MySQL 8.x or later

**Steps:**

1. Create the database: `CREATE DATABASE credit_card_portal;`

**2. Run the schema scripts located in backend/src/main/resources/db/migration:**

```
1 CREATE TABLE users (  
2     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
3     name VARCHAR(255) NOT NULL,  
4     email VARCHAR(255) UNIQUE NOT NULL,  
5     phone_number VARCHAR(20),  
6     address TEXT,  
7     is_eligible_for_bnpl BOOLEAN DEFAULT FALSE  
8 );  
9  
10 CREATE TABLE cards (  
11     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
12     card_number VARCHAR(16) UNIQUE NOT NULL,  
13     card_type VARCHAR(50) NOT NULL,  
14     status VARCHAR(20) NOT NULL,  
15     credit_limit DOUBLE NOT NULL,  
16     available_limit DOUBLE NOT NULL,  
17     expiry_date DATE NOT NULL,  
18     user_id BIGINT,  
19     FOREIGN KEY (user_id) REFERENCES users(id)  
20 );  
21  
22 CREATE TABLE card_applications (  
23     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
24     user_id BIGINT,  
25     card_type VARCHAR(50) NOT NULL,  
26     requested_limit DOUBLE NOT NULL,  
27     application_date DATE NOT NULL,  
28     status VARCHAR(20) NOT NULL,  
29     reviewed_by VARCHAR(255),  
30     review_date DATE,  
31     FOREIGN KEY (user_id) REFERENCES users(id)  
32 );  
33  
34 CREATE TABLE transactions (  
35     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
36     card_id BIGINT,  
37     merchant_name VARCHAR(255) NOT NULL,  
38     amount DOUBLE NOT NULL,  
39     transaction_date DATE NOT NULL,  
40     category VARCHAR(50),  
41     is_bnpl BOOLEAN DEFAULT FALSE,  
42     FOREIGN KEY (card_id) REFERENCES cards(id)  
43 );  
44  
45 CREATE TABLE user_profiles (  
46     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
47     user_id BIGINT,  
48     full_name VARCHAR(255) NOT NULL,  
49     email VARCHAR(255) NOT NULL,  
50     phone VARCHAR(20),
```

```
51     address TEXT,  
52     annual_income DOUBLE,  
53     created_at TIMESTAMP NOT NULL,  
54     updated_at TIMESTAMP,  
55     FOREIGN KEY (user_id) REFERENCES users(id)  
56 );  
57  
58 CREATE TABLE bnpl_installments (  
59     id BIGINT PRIMARY KEY AUTO_INCREMENT,  
60     transaction_id BIGINT,  
61     installment_number INTEGER NOT NULL,  
62     amount DOUBLE NOT NULL,  
63     due_date DATE NOT NULL,  
64     is_paid BOOLEAN DEFAULT FALSE,  
65     late_fee DOUBLE DEFAULT 0.0,  
66     FOREIGN KEY (transaction_id) REFERENCES transactions(id)  
67 );
```

## 4.4 Docker Setup

Docker ensures consistent deployment across environments.

1. Build Docker images: `docker-compose -f docker-compose.yml`
2. `up -d -build`
3. Access the application at `http://localhost:5173`

### Docker Compose Configuration:

```
1 version: '3.8'  
2 services:  
3     backend:  
4         build: ./backend  
5         ports:  
6             - "8080:8080"  
7         depends_on:  
8             - db  
9     frontend:  
10        build: ./frontend  
11        ports:  
12            - "5173:5173"  
13    db:  
14        image: mysql:8  
15        environment:  
16            MYSQL_DATABASE: credit_card_portal  
17            MYSQL_USER: user  
18            MYSQL_PASSWORD: password  
19            MYSQL_ROOT_PASSWORD: rootpassword
```

## 5 API Documentation

### 5.1 Endpoints

The backend exposes RESTful APIs for all functionalities, with clear request/response structures.

Endpoint	Method	Description
/api/cards/{userId}	GET	Retrieves all cards for a user, including status and limits.
/api/cards/{cardId}/status	PUT	Updates card status (e.g., ACTIVE, BLOCKED).
/api/cards/apply	POST	Submits a new card application with type and limit.
/api/cards/applications/{userId}	GET	Lists all card applications for a user.
/api/cards/{cardId}/limit	PUT	Adjusts the credit limit for a card.
/transactions	POST	Simulates a standard transaction.
/transactions/{cardId}	GET	Retrieves transaction history for a card.
/transactions/bnpl	POST	Creates a BNPL transaction with an installment plan.
/bnpl/installments	POST	Adds a new BNPL installment. Lists all BNPL installments.
/bnpl/installments/{id}	GET	Retrieves details for a specific installment.
/bnpl/installments/pending/{transactionId}	GET	pending installments for a transaction.
/bnpl/installments/overdue	GET	Lists overdue installments for a card.
/bnpl/installments/{id}	POST	
/pay	POST	Processes payment for an installment.
bnpl/installments/{id}	PUT	
bnpl/installmenst/{id}	DELETE	Updates installment details.
bnpl/installments/{id}	GET	Deletes an installment.
/api/latefees/card/{cardId}	GET	Calculates total late fees for a cards installments.
/api/profile/{userId}	GET	
/api/profile/{userId}	PUT	
/api/profile	POST	Fetches user profile details. Updates user profile information. Registers a
/api/profile/login	POST	

/api/profile/password	PUT	new user. Authenticates a user. Updates user password.
-----------------------	-----	--

## 5.2 Example Request/Response

**POST /api/transactions/bnpl?installmentPlan=THREE:**

```
1 Request:
2 {
  "cardId":1,
  "amount": 1200.0,
  "category": "Furniture",
  "merchantName":"HomeDepot"
}

Response:
{
  "id":2,
  "cardId":1,
  "amount": 1200.0,"category":
  "Furniture",
  "merchantName":"HomeDepot",
  "status": "Pending",
  "isBNPL": true
}
```

**POST /api/bnpl/installments:**

```
1 Request:
2 {
3     "transactionId": 2,
4     "installmentNumber": 1,
5     "amount": 400.0,
6     "dueDate": "2025-06-25",
7     "isPaid": false
8 }
9
10 Response:
11 {
12     "id": 1,
13     "transactionId": 2,
14     "installmentNumber": 1,
15     "amount": 400.0,
16     "dueDate": "2025-06-25",
17     "isPaid": false,
18     "lateFee": 0.0
19 }
```

## 6 Test Plan

### 6.1 Unit Tests

Unit tests validate individual components for correctness and reliability.

- **Backend:** JUnit and Mockito test services and repositories. Examples:
  - `cardService.getCardsByUserId`: Ensures correct card retrieval.
  - `transactionService.createBNPLTransaction`: Validates BNPL transaction logic
  - `installmentService.calculateLateFee`: Verifies late fee calculations.

**Frontend:** Vitest tests components

- Examples:
  - `CardList.vue`: Checks card rendering and status display.
  - `ApplyCard.vue`: Validates form submission and error handling.
  - `TransactionHistory.vue`: Ensures transaction filtering works.

## 6.2 Integration Tests

Integration tests verify interactions between components:

- MockMvc tests API endpoints for correct responses and error handling.
- Frontend-backend integration tests validate data flow for card management and BNPL operations.
- Integration testing ensures that:

The **end-to-end flow** between components (controller → service → repository → DB) works as expected.

**REST APIs** behave correctly under real application context.

The **application context** loads properly (Spring Boot startup, DB connection, etc.)

## 6.3 End-to-End Tests

Cypress simulates user workflows to ensure system reliability:

- Applying for a card and verifying status updates.
- Creating and paying BNPL installments.
- Updating and validating user profiles.

## 6.4 Test Cases

Module	Test Case	Input	Expected Result
Module 1	Fetch Cards	userId=123	List of users cards with details.
Module 1	Update Status	cardId=1, status=BLOCKED	Card status updated to BLOCKED.
Module 2	Submit Application	cardType=VISA, requestLimit=5000	Application saved, status=PENDING.
Module 3	Update Limit	cardId=1, newLimit=6000	Credit limit updated successfully.
Module 4	Create BNPL Transaction	cardId=1, amount=1200, plan=THREE	BNPL transaction created, status=Pending.

Module 4	Fetch Transactions	cardId=1	List of transactions, including BNPL.
Module 5	Update Profile	userId=123, email=new@example.com	Profile updated with new email.
BNPL	Pay Installment	installmentId=1, amount=400	Installment marked as paid.
BNPL	Fetch Overdue Installments	cardId=1	List of overdue installments with late fees.

## 7 Implementation Details

### 7.1 Module 1: View Credit Cards and Activate/Block

Developed by Abhishekh, this module enables users to view their credit cards and manage their status. The backend uses Spring Data JPA to query the `cards` table by `userId`, with caching to optimize performance. The frontend leverages `CardList.vue` to display cards in a responsive grid, with buttons for activating or blocking cards. This module ensures users have immediate control over their cards, enhancing security and usability.

### 7.2 Module 2: Apply for Credit Card and View Requests

Akuthota Sravani implemented this module, which streamlines the card application process. The backend validates application data (e.g., card type, requested limit) and persists it to the `card_applications` table. The frontends `ApplyCard.vue` provides a user-friendly form with real-time validation, while `RequestStatus.vue` displays application statuses. This module simplifies the application process, reducing customer effort and improving engagement.

### 7.3 Module 3: Manage Card Limit

Aravind P developed this module, allowing users to adjust credit limits within bank-defined constraints. The backend enforces limit validation, updating the `cards` table, while the frontends `ManageLimit.vue` offers an intuitive interface for limit adjustments. This feature empowers users to tailor their credit needs, enhancing financial flexibility.

### 7.4 Module 4: Simulate and View Transactions

Pothuri Tejaswi implemented this module, enabling transaction simulation and history viewing. The backend processes transactions, updating the `transactions` table and available limits, while the frontends `NewTransaction.vue` and `TransactionHistory.vue` provide transaction creation and filtering capabilities. The BNPL integration, added by Abhishekh, allows users to select installment plans, making this module a cornerstone of the portals functionality.

## 7.5 Module 5: Customer Profile Management

Ashritha Sadu developed this module, enabling users to maintain their profiles. The backend validates and updates the `user_profiles` table, ensuring data consistency. The frontend `UserProfile.vue` offers an editable form with validation, improving user trust through accurate profile management.

## 7.6 BNPL Feature Implementation

Abhishekhs BNPL feature adds significant value by enabling installment-based payments. Key components include:

- **Transaction Integration:** BNPL transactions are created via the `transactions` table with `isBNPL=TRUE`.
- **Installment Management:** The `bnpl_installments` table tracks installment details, with APIs for CRUD operations.
- **Late Fee Logic:** Automatically calculates fees for overdue installments based on due dates, stored in the `lateFee` field.

This feature enhances customer retention by offering payment flexibility, with robust backend logic ensuring reliability.

# 8 User Interface Design

## 8.1 Design Principles

The frontend adheres to modern design principles:

- **Responsiveness:** Tailwind CSS ensures compatibility across devices (desktop, tablet, mobile).
- **Accessibility:** WCAG 2.1 compliance supports screen readers and keyboard navigation.
- **Usability:** Clear layouts, consistent navigation, and real-time feedback enhance user experience.

## 8.2 Key Screens

- **Dashboard:** Summarizes card balances, limits, and recent transactions, providing a quick overview.
- **Card List:** Displays cards in a grid with status, limit, and action buttons (activate/block).
- **Apply Card:** A form for submitting card applications, with validation and confirmation.
- **Transaction History:** A filterable table for viewing regular and BNPL transactions.
- **Profile:** An editable form for updating personal and financial details.

## 9 Security Considerations

### 9.1 Data Protection

- Card numbers are masked (e.g., XXXX-XXXX-XXXX-1234) in the UI to prevent exposure.
- Passwords are hashed using bcrypt for secure storage.
- HTTPS encrypts all API communications, protecting data in transit.

### 9.2 API Security

- Input validation prevents SQL injection and XSS attacks.
- Rate limiting mitigates brute-force attempts.
- CORS restricts access to trusted origins.

### 9.3 Future Security Enhancements

- Implement OAuth2 for authentication.
- Add JWT for session management.
- Conduct regular penetration testing.

## 10 Performance Optimization

### 10.1 Backend Optimization

- JPA query optimization reduces database load.
- Caching (e.g., Redis) for frequent queries like card details.
- Asynchronous processing for transaction and BNPL operations.

### 10.2 Frontend Optimization

- Lazy loading of Vue.js components minimizes initial load time.
- Vite optimizes assets, reducing bundle sizes.
- Minified CSS/JavaScript improves page load speed.

### 10.3 Database Optimization

- Indexes on `userId`, `cardId`, and `email` speed up queries.

- Connection pooling optimizes MySQL connections.
- Query caching reduces redundant database hits.

## 11 Deployment

### 11.1 Deployment Environment

The application supports local, staging, and production environments, with Docker ensuring consistency.

### 11.2 Deployment Steps

1. Build Docker images: `docker-compose build`
2. Run containers: `docker-compose up -d`
3. Verify application at `http://localhost:5173`

## 12 Troubleshooting

### 12.1 Common Issues

Issue	Solution
Database connection error	Verify MySQL credentials and ensure the database is running.
API returns 404	Check endpoint URLs and backend server status.
Frontend not loading	Ensure Vite server is running ( <code>npm run dev</code> ).
CORS errors	Update CORS settings in <code>application.yml</code> .
Docker container fails	Inspect logs ( <code>docker logs &lt;container_id&gt;</code> ).
BNPL transaction failure	Confirm user eligibility and sufficient card limit.

## 13 Future Enhancements

- **BNPL Enhancements:** Support dynamic installment durations and interest rates.
- **Authentication:** Integrate OAuth2 for secure user login.
- **Notifications:** Implement email/SMS alerts for application status and overdue payments.
- **Admin Dashboard:** Build a dashboard for bank staff to manage applications and transactions.

- **Mobile App:** Develop a React Native app for iOS and Android.
- **Analytics:** Add spending insights and transaction trends.

## 14 Team and Roles

### 14.1 Team Members and Module Ownership

Member Name	Module Responsibility	Backend Component	Frontend Component
Abhishekh (Team Lead)	Module 1: View Credit Cards + Activate/Block	GET /cards/{userId}, PUT /cards/{cardId}/status	CardList.vue
Akuthota Sravani	Module 2: Apply for Credit Card + View Requests	POST /cards/apply, GET /cards/applications/{userId}	ApplyCard.vue, RequestStatus.vue
Aravind P	Module 3: Manage Card Limit	PUT /cards/{cardId}/limit	ManageLimit.vue
Pothuri Tejaswi	Module 4: Simulate & View Transactions	POST /transactions, GET /transactions/{cardId}	NewTransaction.vue, TransactionHistory.vue
Ashritha Sadu	Module 5: Customer Profile Management	GET/PUT /api/profile/{userId}	UserProfile.vue

### 14.2 Additional Contributions

Abhishekhs BNPL feature implementation added a competitive edge, enabling flexible payment options and robust installment management.

## 15 Appendix

### 15.1 Additional Resources

- **Repository:** <https://github.com/Abhishekhzeta/credit-card-management-port>
- **API Collection:** Postman collections (BNPL\_API\_Collection.json, CapstoneApi.postman\_collection.js)

### 15.2 System Diagram

The architecture includes:

- Frontend (Vue.js) interacting with Backend (Spring Boot) via REST APIs.
- Backend communicating with MySQL database via JPA.

- Docker containers for frontend, backend, and MySQL.
- CLASS DIAGRAM:

