Gramener

A Straive Company

Software Requirements Specification

(SRS Document)

for

Credit Card Management System

By

Krishna Sai Keerthan Nagandla	E430388
Kethu Sesha Sarath Reddy	E430373
Nayini Sai Nithin	E430387
Kethavath Sundar	E430375

Supervisor

Preety Singh

Table of Contents

1. Introduction

1.1 Purpose

To develop a credit card management system with secure, user-friendly interfaces.

1.2 Scope

Managing customer accounts, transactions, credit card details, payments, etc.

1.3 Definitions, Acronyms, and Abbreviations

Definitions

- **Credit Card Management System**: A web-based platform to manage user credit card details, transactions, and payments.
- **User**: Any individual interacting with the system, categorized as “Admin" or "Customer."

Acronyms and Abbreviations

1. **SRS (Software Requirements Specification)**
   A document outlining the functional and non-functional requirements of the system.
2. **API (Application Programming Interface)**
   Protocols and tools for enabling communication between frontend (React) and backend (Flask).
3. **MySQL (Structured Query Language)**
   A relational database used for storing and managing structured data like users, credit cards, and transactions.
4. **HTTPS (HyperText Transfer Protocol Secure)**
   A protocol ensuring secure communication over the web.
5. **JSON (JavaScript Object Notation)**
   A lightweight format for data exchange between system components.

1.4 Overview

This document outlines the functional and non-functional requirements for the Credit Card Management System and serves as a roadmap for its development. The primary goal is to build a secure, user-friendly web-based system leveraging Flask for the backend, MySQL for data management, and React for the frontend.

The SRS is structured as follows:

1. **Introduction**
   Provides the purpose, scope, and context of the project. It also includes relevant definitions, acronyms, and references to clarify terminology used throughout the document.
2. **System Overview**
   A high-level description of the system, highlighting its core components and their interactions, including the architecture and key modules.
3. **Functional Requirements**
   Details the key functionalities, such as user authentication, credit card management, transaction processing, and payment handling.
4. **Non-Functional Requirements**
   Specifies performance, security, usability, and scalability expectations, ensuring the system is robust and future-proof.
5. **System Architecture**
   Describes the architecture of the system, including frontend-backend interactions, database design, and the communication between components.
6. **User Interface Requirements**
   Explains the visual and functional expectations for key user-facing pages like the dashboard, transaction history, and payment pages, accompanied by wireframes or mockups where applicable.
7. **Data Requirements**
   Defines the database schema, relationships, and validation rules for managing system data.
8. **Assumptions and Dependencies**
   Lists assumptions regarding the system environment, tools, and technologies, as well as any external dependencies.
9. **Future Work and Updates**
   Outlines planned enhancements and features for future development phases.
10. **Appendices**
    Includes a glossary of terms, additional references, or resources for further understanding.

This structured approach ensures that stakeholders and developers have a clear and comprehensive understanding of the system's objectives and requirements, facilitating efficient and effective development.

2. System Overview

The Credit Card Management System is a web-based platform designed to streamline the management of user accounts, credit card information, transactions, and payments. It combines secure backend processing, a responsive frontend interface, and efficient database management to provide a seamless user experience.

2.1 System Objectives

- ➢ To provide users with a secure and user-friendly platform to manage credit card details and track transactions.
- ➢ To allow administrators to manage users, monitor transactions, and generate analytics.
- ➢ To ensure scalability, reliability, and data security throughout the system.

2.2 Key Features

1. **User Management**
    - Account creation, authentication, and authorization with role-based access control.
    - Profile management, including personal details and credit card assignments.
2. **Credit Card Management**
    - Add, update, or remove credit card information.
    - Set and manage credit limits, expiration dates, and other parameters.
3. **Transaction Management**
    - Record and display transaction histories.
    - Enable secure payment processing and dues management.
4. **Notifications and Alerts**
    - Provide real-time notifications for transactions.
    - Send alerts for payment due dates and unusual account activity.
5. **Analytics and Reporting**
    - Generate visual reports for account activity and transaction summaries.

2.3 System Architecture Overview

The system follows a three-tier architecture:

1. **Frontend (Presentation Layer)**
    - Built using **React**, the frontend provides a responsive and intuitive user interface.

    - It communicates with the backend via REST APIs for all user interactions.
2. **Backend (Application Layer)**
    - Developed using **Flask**, the backend serves as the core logic layer.
    - It includes API endpoints for managing users, credit cards, and transactions.
    - Implements security features like encryption, authentication, and validation.
3. **Database (Data Layer)**
    - Powered by **MySQL**, the database stores and organizes data related to users, credit cards, and transactions,.
    - It supports relational schema design to ensure data integrity and efficient queries.

2.4 System Workflow

1. **User Interaction**: Users interact with the system through the React frontend, logging in or signing up as required.
2. **Data Processing**: The frontend sends requests to the Flask backend through REST APIs. The backend processes these requests, applies business logic, and communicates with the database.
3. **Data Management**: MySQL manages the data, performing CRUD operations based on backend requests. The results are returned to the backend and then displayed on the frontend.

2.5 Target Audience

The system is designed for:

1. **Customers**: Individuals who need to manage their credit cards, view transactions, and make payments.
2. **Administrators**: System managers responsible for overseeing user activities and ensuring system compliance.

3. Functional Requirements

3.1 User Authentication and Authorization

➢ The system must allow users to register with their details, such as name, email, password, and contact information.

➢ Users must be able to log in using their registered email and password.
➢ Password reset functionality must be available via email. (future work)
➢ Role-based access control (RBAC):
    - **Admin** users can manage all aspects of the system, including user accounts, credit cards, and reports.
    - **Customer** users can manage their own credit card details and view their transactions.

3.2 User Management

➢ Admins must be able to:
    - View a list of all registered users.
    - Add, update, or deactivate user accounts.
➢ Customers must be able to:
    - View and update their own profile information.

3.3 Credit Card Management

➢ Customers must be able to:
    - Add credit card details, including card number, expiration date, CVV.
    - Edit or delete their own credit card details.
➢ Admins must be able to:
    - Assign or modify credit card details for any user.
    - Set system-wide rules, such as maximum credit limits.

3.4 Transactions and Payments

➢ The system must support recording transactions linked to a specific credit card.
➢ Customers must be able to view their transaction history and make payments toward their outstanding balance securely.
➢ Customers must be able to view their transaction history with filters for date, amount, and type (e.g., purchases or payments). (future work)
➢ Admins must be able to monitor all transactions.

3.5 Notifications and Alerts

➢ Customers must receive notifications for:
    ○ Successful or failed transactions.
    ○ Payment due dates and overdue balances.
➢ Admins must receive alerts for suspicious activities (e.g., unusual transaction volumes). (future work)

3.6 Reporting and Analytics

- ➢ Customers must be able to view summary reports, including:
  - Monthly transaction summaries.
  - Outstanding balances and payment history.
- ➢ Admins must be able to generate system-wide reports, including:
  - User activity reports.
  - Transaction trends and analytics.

3.7 Security Features

- ➢ The system must enforce strong password policies during registration and updates. (future work)
- ➢ Sensitive information (e.g., passwords and card details) must be securely encrypted and never stored in plaintext. (future work)

4. Non-Functional Requirements

4.1 Performance Requirements

- ❖ The system should handle up to “40” concurrent users(Docker account limit) without significant performance degradation.
- ❖ Response time for all API calls must not exceed 2 seconds under normal load.
- ❖ The system should process and display transaction data (e.g., filters, summaries) within 3 seconds for datasets of up to 10,000 records. (future work)
- ❖ Database queries should be optimized to handle 10,000 transactions per user efficiently.

4.2 Security Requirements

- ❖ The system must use HTTPS to ensure secure communication between users and the server.
- ❖ The system must log and alert administrators of failed login attempts after 5 consecutive attempts to prevent brute-force attacks. (future work)

4.3 Usability Requirements

- ❖ The user interface must be intuitive and responsive, providing a consistent experience across devices (tablet, mobile - future work).

- ❖ Users should be able to complete core tasks, such as viewing transaction history or making payments, in 3 clicks or less from the dashboard.
- ❖ Error messages must be clear, actionable, and displayed in real time for invalid inputs.

4.4 Scalability Requirements

- ❖ APIs must be modular and extensible to accommodate new features (e.g., integration with third-party payment gateways).
- ❖ Ability to handle increased users and transactions over time.(future work)

4.5 Compatibility Requirements

- ❖ The frontend must be compatible with the latest versions of modern browsers, including Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
- ❖ The system should work seamlessly on devices with varying screen sizes (responsive design).
- ❖ The system must support integration with third-party APIs and services for additional functionalities, such as fraud detection or payment processing.

5. System Architecture

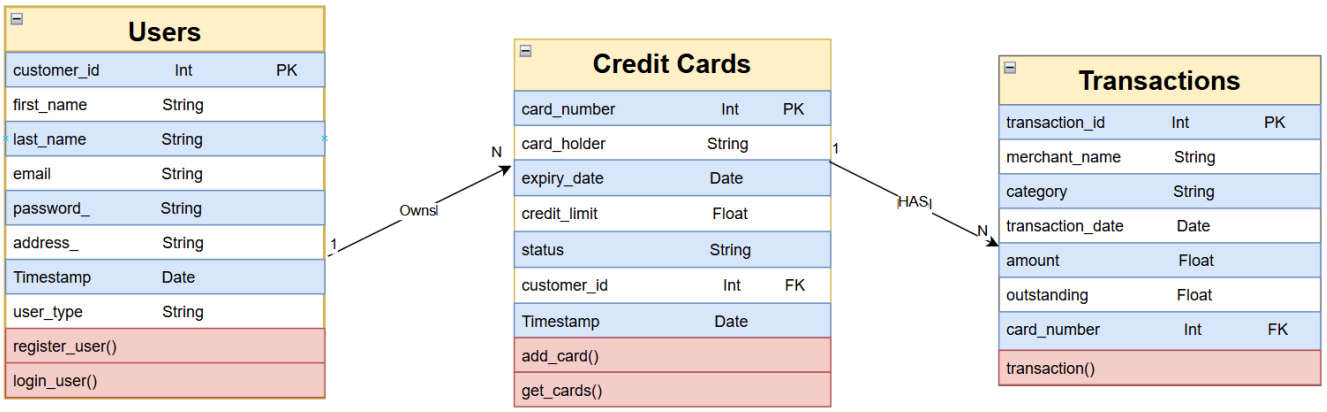
The Credit Card Management System is designed using a three-tier architecture: the Frontend, Backend, and Database, with communication between these layers facilitated via RESTful APIs.

5.1 Frontend

- ➢ The frontend is developed using React for creating a dynamic and responsive user interface.
- ➢ It handles user interactions such as logging in, managing credit cards, viewing transaction history, and making payments.
- ➢ React communicates with the backend through REST APIs, ensuring that the user interface is separated from business logic for flexibility.

5.2 Backend

5.2.1 Class Diagram



5.2.2 Backend Architecture Description

- The backend is powered by Flask, a lightweight Python framework used to develop RESTful API services.
- It is responsible for processing business logic, handling user requests, managing authentication, and interacting with the database.
- The backend exposes API endpoints for operations like user registration, login, credit card management, and transaction processing.

5.3 Database

- The **database** is based on **MySQL**, a relational database management system.
- It stores all data related to users, credit cards and transactions in a structured format.
- The database uses normalized tables to ensure data integrity and efficient querying.

5.4 Communication

- Communication between the **Frontend** and **Backend** is done via **RESTful APIs**, using **HTTP methods** (GET, POST, PUT, DELETE).
- The frontend makes requests to the backend to retrieve or update data, and the backend processes these requests and communicates with the database to fetch or modify the data accordingly.

6. User Interface Requirements

6.1 General Requirements

- ➢ The design adheres to a consistent theme with a clean, modern look and feel.
- ➢ Navigation is intuitive, with clearly labeled menus and easy-to-access features.
- ➢ Error messages, confirmations, and alerts displayed prominently with meaningful feedback.

6.2 User Roles and Interfaces

6.2.1 Customer Interface

The **Customer Interface** include the following:

- ➢ **Login and Registration Page**:
  - Fields for email and password.
  - Options for account creation.
- ➢ **Dashboard**:
  - Overview of credit card details, recent transactions, and outstanding balances.
  - Quick links to manage credit cards, view transaction history, and make payments.
- ➢ **Credit Card Management**:
  - Forms to add, update, or delete credit card information.
  - Validation for sensitive fields like card number and CVV.
- ➢ **Transaction History**:
  - A table displaying all transactions.
  - Summary of monthly expenses and payments.
- ➢ **Payment Section**:
  - Options to make secure payments for outstanding balances.
  - Integration with third-party payment gateways.(future work)

6.2.2 Admin Interface

The **Admin Interface** must include the following:

- ➢ **Login Page**:
  - Admin-specific login with enhanced security.
- ➢ **User Management**:
  - List of all registered users with options to deactivate accounts, reset passwords, or update user roles.
- ➢ **Credit Card Management**:

- Ability to assign or revoke credit cards for customers.
- Set or modify system-wide rules (e.g., maximum credit limits).

➢ **Reports and Analytics**:

- Dashboard showing system-wide metrics such as total users, active credit cards, and transaction trends. (future work)
- Downloadable reports in PDF or Excel format. (future work)

6.3 Feedback and Notifications

➢ Users must receive real-time feedback for their actions, such as:

- Confirmation messages for successful operations (e.g., "Payment processed successfully").
- Error messages for invalid inputs (e.g., "Invalid CVV format").

➢ Notifications for important updates like payment due dates or unusual transactions must be displayed as:

- Pop-up alerts or banners within the app.
- Email notifications, depending on user preferences. (future work)

7. Data Requirements

7.1 Data Storage Requirements

7.1.1 Users Table

➢ **Purpose**: Stores information about registered users, including their personal details and account credentials.

➢ **Key Attributes**:

- **customer_id**: Unique identifier for each customer (Primary Key).
- **first_name** and **last_name**: Customer's name, allowing a maximum length of 100 characters.
- **email**: Unique email address for login and communication, restricted to 150 characters.
- **password_**: Encrypted password for user authentication, with a length suitable for hash storage.
- **address_**: Optional text field for storing the customer’s address.
- **user_type**: Enumeration of the admin and user
- Timestamp indicating when the customer account was created.

7.1.2 Credit Cards Table

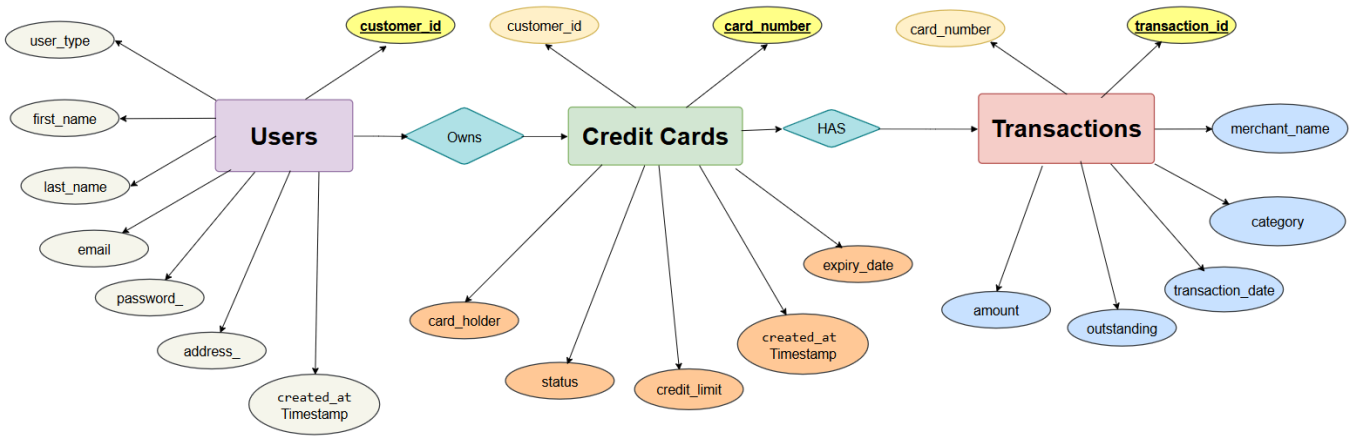
- ➢ **Purpose**: Tracks the credit cards associated with customers, including their details and statuses.
- ➢ **Key Attributes**:
  - **card_number**: Unique 20-character string representing the card number(Primary Key).
  - **customer_id**: Foreign Key linking the card to its respective customer in the customers table.
  - **card_holder**: Name of the cardholder, allowing a maximum of 100 characters.
  - **expiry_date**: Date field indicating when the card will expire.
  - **credit_limit**: Numeric value defining the credit limit for the card.
  - **status**: Enumeration of the card's current state, with possible values:
    - **Active**: The card is in use.
    - **Blocked**: Temporarily inaccessible due to security or other issues.
    - **Suspended**: Paused for administrative reasons.
    - **Closed**: Permanently deactivated.
  - Timestamp indicating when the credit card was created.

7.1.3 Transactions Table

- **Purpose**: Logs all transactions performed using the credit cards.
- **Key Attributes**:
  - **transaction_id**: Unique identifier for each transaction (Primary Key).
  - **card_number**: Foreign Key linking the transaction to the relevant credit card in the credit_cards table.
  - **merchant_name**: Name of the merchant where the transaction occurred, allowing up to 255 characters.
  - **category**: Text field for categorizing the transaction (e.g., groceries, utilities).
  - **transaction_date**: Timestamp recording when the transaction occurred, defaulting to the current time.
  - **amount**: Numeric value for the transaction amount.
  - **outstanding**: Numeric value representing the remaining balance after the transaction.

7.2 Relationships

7.2.1 Entity Relationship Diagram (ERD)



7.2.2 Description of Relationships

- ➢ **One-to-Many Relationship**:
  - A customer can own multiple credit cards.
  - A credit card can have multiple transactions.
- ➢ **Foreign Key Constraints**:
  - Ensure that every credit card belongs to a valid customer.
  - Ensure that every transaction is associated with a valid credit card.
- ➢ **On Delete Cascade**:
  - Deleting a customer automatically deletes their associated credit cards and transactions.
  - Deleting a credit card automatically deletes its associated transactions.

7.3 Security Requirements

- ➢ **Encryption**:
  - Store password values in the customers table using a hashing algorithm like **bcrypt** for enhanced security.
  - Encrypt sensitive data such as card_number before storage.(future work)
- ➢ **Access Control**: Restrict database access to authorized backend services and administrators.
- ➢ **Data Auditing**: Maintain logs for any data modifications, including updates or deletions in credit_cards and transactions. (future work)

7.4 Data Validation and Constraints

- ➢ **Unique Constraints**:
  - email in the customers table must be unique to prevent duplicate accounts.
  - card_number in the credit_cards table must be unique to avoid duplication of card details.
- ➢ **Required Fields**:
  - Mandatory fields include first_name, last_name, email, password_ in customers, and card_number, card_holder, expiry_date, credit_limit in credit_cards.
  - Transactions must include amount and outstanding values.
- ➢ **Data Types**: Use appropriate data types such as VARCHAR, TEXT, DECIMAL, and TIMESTAMP to optimize storage and retrieval.
- ➢ **Enumerations**: The status field in credit_cards ensures predefined values to maintain consistency.

8. Assumptions and Dependencies

8.1 Assumptions

1. **Internet Access**: Users have a stable internet connection for seamless system access.
2. **Supported Devices**: The frontend is optimized for modern browsers (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) and assumes users are accessing the system through compatible devices.
3. **User Competence**: Users are familiar with basic web application usage.
4. **Accurate Data**: Users provide valid information during registration and transactions.
5. **Third-Party Services**: Payment gateways and notification services remain functional.
6. **System Uptime**: The hosting environment for the backend and database is assumed to have high availability (99.9% uptime).
7. **Compliance**: The system adheres to relevant data protection and banking regulations.
8. **Maintenance**: Regular updates and maintenance are performed to ensure smooth operation.

8.2 Dependencies

1. **Technology Stack**:
    a. **Frontend**: The system is built using **React**, and its performance depends on React's stability and compatibility with modern web standards.
    b. **Backend**: The backend uses **Flask**, relying on its library ecosystem and Python runtime for efficient operation.
    c. **Database**: The **MySQL** database is critical for data storage and retrieval, and the system depends on MySQL's availability and scalability.
2. **Third-Party APIs**: Essential for payment processing, notifications, and external libraries.
3. **Hosting and Infrastructure**: Cloud services (e.g., AWS, Google Cloud) or Docker for application and database hosting.
4. **Security Protocols**: SSL/TLS encryption and adherence to PCI DSS standards.
5. **Regulatory Compliance**: Any updates in legal or financial regulations may require system changes.
6. **Team and Users**: Skilled developers for maintenance and proactive user adoption of system best practices.

9. Future Work and Updates

9.1 Planned Features

List features that will be implemented in future versions, such as:

- ➢ Customers must be able to view their transaction history with filters for date, amount, and type (e.g., purchases or payments).
- ➢ Password reset/recovery functionality must be available via email.
- ➢ Admins must receive alerts for suspicious activities.
- ➢ Integration with third-party payment gateways.
- ➢ Maintain logs for any data modifications, including updates or deletions in credit_cards and transactions.
- ➢ A trained chat bot for users to get 24/7 help.
- ➢ Upload the Aadhar document and the text will be extracted and fill the columns automatically.

9.2 System Enhancements

- ➢ Advanced Fraud Detection: Integration of AI and machine learning algorithms to analyze transaction patterns and detect fraudulent activities in real-time.
- ➢ Mobile Application Development: Development of a dedicated mobile application for Android and iOS platforms, providing users with enhanced accessibility and offline features.
- ➢ Multi-Currency Support: Enabling transactions and credit card management in multiple currencies to cater to international users and businesses.
- ➢ Enhanced Reporting and Analytics: Development of more robust reporting dashboards with real-time data visualization, transaction trends, and financial insights for both users and administrators.
- ➢ Voice Assistant Integration: Incorporating voice-based interaction features to allow users to manage credit cards or access information using virtual assistants like Alexa or Google Assistant.
- ➢ Enhanced Security Features: Implementation of biometric authentication methods (e.g., fingerprint, face recognition) for additional security during login and transaction processes.
- ➢ User Feedback Integration: Regular collection and analysis of user feedback to prioritize and implement new features or improve existing functionalities.

9.3 Long-Term Goals

- ➢ Global Expansion: Adapt the system to support international markets, including multi-language support, compliance with global financial regulations, and region-specific features.
- ➢ AI-Powered Financial Advisor: Integrate an AI-driven personal financial advisor to provide users with insights, budgeting recommendations, and personalized credit card usage tips.
- ➢ Partnership Ecosystem: Build partnerships with merchants, loyalty programs, and fintech companies to provide users with exclusive benefits and seamless financial experiences.
- ➢ Autonomous Operations: Move toward autonomous system maintenance and upgrades using AI to predict, plan, and execute improvements without manual intervention.