Software Requirements Specification (SRS)

For Virtual Payment Platform

1. Introduction

1.1 Purpose

The purpose of this document is to specify the functional and non-functional requirements for the development of the **Virtual Payment Platform**.
This platform is designed to provide secure digital financial services including **virtual card issuance**, **wallet management**, and **user authentication**.
The document will serve as a reference for developers, testers, project managers, and stakeholders throughout the system's lifecycle.

1.2 Scope

The Virtual Payment Platform aims to enable users to perform secure online transactions using virtual cards linked to a digital wallet.
The system provides functionalities for:

- User registration and login
- Wallet creation and management
- Virtual card generation via third-party APIs
- Transaction processing and caching using Redis
- User feedback and transaction history tracking

1.3 Objectives

- Develop a secure and scalable virtual payment system.
- Enable users to generate and manage virtual cards.
- Provide real-time transaction processing.
- Integrate Redis for caching frequently accessed data.
- Ensure data privacy and security through encryption and authentication.

1.4 Definitions, Acronyms, and Abbreviations

Term	Definition
API	Application Programming Interface
DBMS	Database Management System
JWT	JSON Web Token
UI	User Interface
UX	User Experience
SRS	Software Requirements Specification
Redis	Remote Dictionary Server (for caching)

Frikang Favour

2. Overall Description

2.1 Product Perspective

The system operates as a **client-server architecture**, where the client (web or mobile app) communicates with the backend via RESTful APIs.
The backend handles business logic, authentication, and transactions, while the database stores user and transaction data. Redis will serve as a caching layer for faster data retrieval.

2.2 Product Functions

- User authentication (sign up, sign in)
- Wallet balance management
- Virtual card creation and linking
- Transaction processing and tracking
- Redis caching for speed optimization
- User feedback module
- Admin dashboard for system management

2.3 User Characteristics

User Type	Description
End Users	Individuals using virtual cards and wallets for transactions
Administrators	Manage user data, monitor transactions, and handle reports
Third-Party API	External payment or card service providers integrated into the system

2.4 Operating Environment

- **Frontend:** React.js / Flutter
- **Backend:** Node.js (Express.js Framework)
- **Database:** MySQL
- **Cache:** Redis
- **API Gateway:** RESTful endpoints
- **Deployment:** Cloud-based (AWS / Azure)

2.5 Design Constraints

- Must comply with data privacy laws (e.g., GDPR).
- All passwords must be hashed (bcrypt).
- Tokens must be secured with JWT.
- APIs must use HTTPS protocol.

2.6 Assumptions and Dependencies

- Users must have an active internet connection.
- System relies on third-party payment and virtual card APIs.
- Redis server must be properly configured for caching.
- Node.js runtime and MySQL database are functional.

3. System Features

3.1 Authentication and User Management

Description:
Allows new users to register, log in, and manage their profiles.

Functionalities:

- Register new users with full details.
- Secure login using email and password.
- Token-based authentication (JWT).

Inputs: first name, last name, email, phone, address, password.
Outputs: Success message, authentication token.

3.2 Wallet Management

Description:
Enables users to maintain a virtual wallet that stores money for transactions.

Functionalities:

- View wallet balance.
- Deposit and withdraw funds.
- View transaction history.
- Sync wallet balance with virtual card.

Constraints:
Each wallet must be uniquely associated with a user ID.

3.3 Virtual Card Service

Description:
Connects with a third-party API (e.g., Paystack, Flutterwave) to issue and manage virtual cards.

Functionalities:

- Request new virtual card.
- Link card to user’s wallet.
- Fetch card details and transaction logs.

Inputs: user_id, card_type, amount.
Outputs: card_number, expiry_date, card_balance.

3.4 Transaction System

Description:
Handles all transaction processing between wallets and cards.

Functionalities:

- Record deposits and withdrawals.
- Verify transaction amounts and limits.
- Cache recent transactions in Redis for faster response times.

Process:
Transaction → API Gateway → Business Logic → MySQL (Persistent) + Redis (Cache).

3.5 Feedback System

Description:
Allows users to submit anonymous feedback about their experience.

Functionalities:

- Feedback form with anonymity toggle.
- Storage in feedback database.
- Analytics for administrators.

3.6 Administrative Dashboard

Description:
Provides tools for administrators to monitor and manage system activities.

Functionalities:

- View users and wallets.
- Review transactions and feedback.

Frikang Favour

- Manage API connections.

4. External Interface Requirements

4.1 User Interfaces

- Simple, responsive web and mobile UI.
- Login and registration forms.
- Wallet and card management dashboards.

4.2 Hardware Interfaces

- Device with internet access (smartphone or PC).

4.3 Software Interfaces

- **Frontend:** communicates with backend through REST APIs.
- **Third-party APIs:** handle card creation and payment operations.

4.4 Communication Interfaces

- All communication over HTTPS.
- JSON used for data exchange.

5. Non-Functional Requirements

5.1 Performance Requirements

- API response time should be < 2 seconds for cached data.
- Server must handle at least 1000 concurrent users.

5.2 Security Requirements

- User passwords hashed using bcrypt.
- JWT tokens used for all sessions.
- Role-based access control for admin features.

5.3 Reliability and Availability

- System uptime target: 99.5%.
- Redis ensures redundancy in case of database delay.

5.4 Maintainability

- Modular code structure using Node.js routes and services.
- Easy to add new modules or APIs.

5.5 Scalability

- Built on microservice architecture.
- Supports horizontal scaling across servers.

5.6 Usability

- Clean interface with clear navigation.
- Error messages and input validation for user-friendly experience.

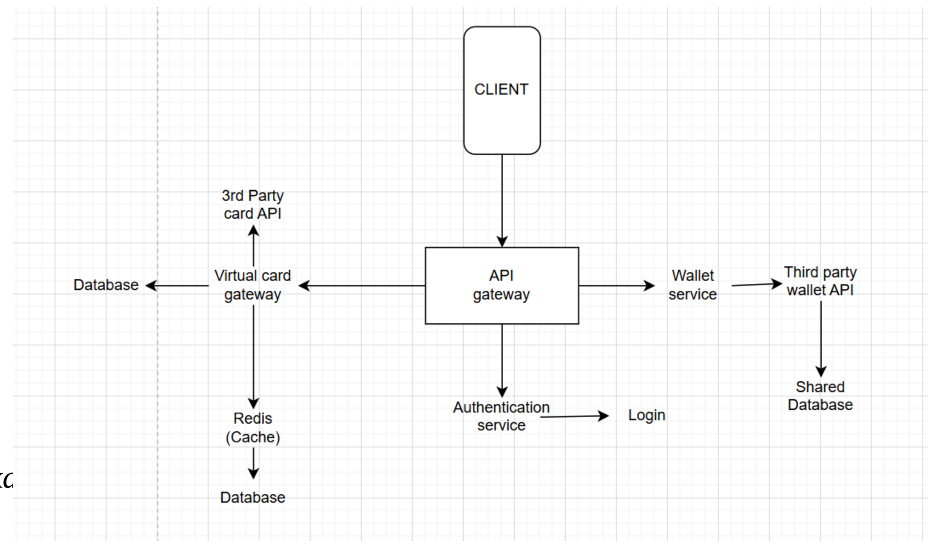
6. Database Design

Table	Key Fields	Description
users	id, name, email, password	Stores user info
wallets	wallet_id, user_id, balance	Tracks wallet balance
cards	card_id, user_id, card_number, expiry	Stores virtual card details
transactions	trans_id, wallet_id, amount, type	Records user transactions
feedback	feed_id, user_id, message	Stores feedback messages

7. System Architecture

The system uses a 3-tier client/server structure:

```
Client (Web/Mobile)
       ↓
API Gateway (Node.js)
       ↓
Database (MySQL) + Cache (Redis)
```

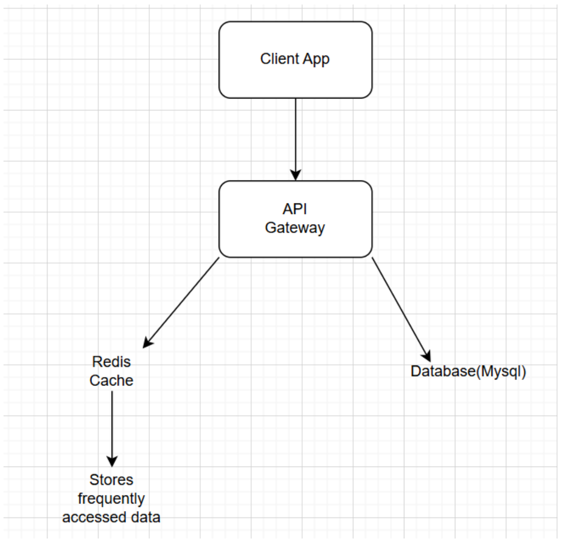


Frikc

8. Redis Caching Mechanism

Redis acts as a high-speed data cache for transactions and wallet balances. When a transaction request is made:

1. The system checks Redis first.
2. If found, it returns cached data instantly.
3. If not, it fetches from MySQL and stores in Redis for next use.



9. System Design Considerations

- Developed using **Node.js** for non-blocking event-driven operations.
- **MySQL** selected for structured data storage and ACID compliance.
- **Redis** used for caching to boost performance.
- **JWT** ensures secure user sessions.

10. Testing Plan

Test Type	Description	Tool
Unit Test	Verify individual modules (Auth, Wallet)	Jest / Mocha
Integration Test	Ensure modules interact correctly	Postman
System Test	Full workflow simulation	Manual testing
Performance Test	Measure speed & load capacity	Apache JMeter

11. Conclusion

The Virtual Payment Platform successfully implements a secure, modular, and scalable architecture for virtual financial services.
With authentication, wallet, and virtual card modules completed and tested, the project establishes a strong foundation for future integration with payment gateways and mobile apps.

Frikang Favour