System Architecture Documentation

Project Title:

Architecture Design for a Financial System Offering Virtual Cards

Course:

Software Architecture assigment

Prepared by:

Brice Parfait  
Date: 20th October 2025

Introduction

This document presents the architecture design for a financial system that allows users to obtain and manage virtual cards for payment processing.  
The system integrates multiple functional modules, external services, and databases to ensure secure, efficient, and fault-tolerant operations. It was designed using Figma prototyping, based on a three-tier architecture that includes:

A **3-tier architecture** is a client–server architecture that divides an application into three logical layers or tiers. Each layer handles a specific responsibility, making the system modular, secure, and scalable.

* Frontend Layer (Client App)
* Backend Layer (Server and APIs)
* Database Layer (Storage and Third-party services)

System Overview

The system enables users to:

* Create an account and log in securely.
* Complete KYC verification.
* Request, view, and manage virtual payment cards.
* Perform transactions through a third-party payment gateway.
* Receive notifications and updates.
* Manage their account and perform cashing operations.

Architecture Diagram

Overall System Architecture

This diagram shows the 3-tier structure:

* Frontend (Mobile/Web App) → communicates via HTTPS requests to the backend.
* Backend (Server/API) → processes business logic, authentication, and connects with third parties.
* Database (MySQL + Cloud Storage) → securely stores user data, KYC records, and transaction logs.

Flow:

1. User interacts with frontend → request sent to backend API.
2. Backend validates, processes, and interacts with third-party APIs if needed.
3. Responses returned to frontend → data stored/retrieved from database.

Request–Response Flow Diagram

This diagram describes how the client, backend, and database exchange data during operations like login, card creation, payment, and notifications.

Example Flow – Login Process:

1. User enters credentials in the mobile app.
2. Frontend sends a POST request → /api/auth/login.
3. Backend verifies credentials → generates JWT token.
4. Response (token + user data) sent to frontend.
5. User session established → dashboard loaded.

Example Flow – Payment Processing:

1. User initiates a payment from the app.
2. Backend calls Payment Gateway API (e.g., Flutterwave).
3. Gateway processes and returns transaction status.
4. Backend stores result → updates wallet + sends notification

Module Interaction Diagram

This diagram focuses on how modules within the backend communicate and depend on one another.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Description | Input | Output |
| User Authentication | Handles login, signup, JWT tokens | Credentials | Session token |
| User Management | Manages user profile and roles | Token, user ID | Updated user data |
| KYC Module | Verifies identity with external database | KYC form | Verified/Rejected status |
| Card Provision | Connects to third-party API for card issuance | User token, KYC | Virtual card details |
| Payment Processing | Handles payments with third-party gateway | Card info, amount | Transaction receipt |
| Role Engine (Fault Detection) | Monitors transactions for fraud or unauthorized access | Logs | Alerts |
| Notification Service | Sends emails/SMS/push alerts | Event | Notification message |
| Cashing Module | Handles withdrawals to user bank account | Request | Transfer confirmation |

Flow Summary:

1. User authentication grants secure access.
2. User management controls access rights.
3. KYC validates user identity before enabling card services.
4. Payment and card modules interact with third parties for transactions.
5. Notifications and fault detection modules enhance reliability and transparency.

Technology Stack

|  |  |  |
| --- | --- | --- |
| Layer | Technology | Role |
| Frontend (Mobile App) | Flutter / React Native | User Interface |
| Backend | Node.js / Express or Django REST | API + Logic |
| Database | MySQL / MongoDB | Data Storage |

Conclusion:

This system ensures :

* Scalability: Each layer can be scaled independently.
* Security: the database is isolated from direct user access. Reucing risks of data leaks.
* Maintainability: changes in one layer doesn’t affect the other layer.

https://www.figma.com/design/Lfi6FEk1LCujDXZKp3N5P4/3-tier?node-id=0-1&t=dNaheXf2SUEF1ws8-1