# **Virtual Card platform**

The advancement of financial technology has transformed how individuals and organizations manage money, leading to the rise of secure and efficient digital payment solutions. This project focuses on the design and documentation of a Virtual Card Platform, a system that enables users to create and manage virtual cards, perform financial transactions, and maintain secure digital wallets. The platform also integrates Know Your Customer (KYC) verification to ensure compliance, trust, and regulatory adherence in all user activities. It emphasizes modular design principles, security, and data integrity, providing a scalable and maintainable solution for modern financial ecosystems.

This documentation presents the complete system architecture of the Virtual Card Platform, detailing its major components, database entities, and API endpoints. It further explores the interactions between subsystems such as user management, wallet services, and transaction processing. In addition to describing the core functionalities, the report examines potential technical difficulties within the architecture—such as security challenges, scalability issues, and data consistency—and proposes appropriate engineering solutions to address them. The result is a well-structured, secure, and efficient digital payment framework that aligns with real-world financial system standards.

## Functional Requirements

Functional requirements define what the **Virtual Card Platform** must do to achieve its objectives. They describe the **core features and behaviors** that the system must support to meet user and business needs. Below is an explanation of each major requirement:

### User Registration and Authentication

The platform must allow users to create an account using secure credentials such as email, password, and verification methods. Authentication ensures that only verified users can access the system. Login and logout sessions are handled using secure tokens (JWT), and password management (reset or update) must follow strong security practices.

### KYC (Know Your Customer) Verification

Before users can perform any financial operation, they must complete KYC verification. This process requires uploading valid identification documents (e.g., national ID, passport, or driver’s license). The system stores these documents securely and marks the user as “verified” once the admin or automated system approves them. This ensures compliance with financial regulations and prevents fraud.

### Wallet Management

Each user has a digital wallet linked to their profile. The wallet supports balance updates, top-ups (funding), withdrawals, and transfers between users. All transactions are atomic — meaning they either complete successfully or not at all — to maintain financial consistency. The wallet must also maintain real-time balance checks and transaction histories.

### Virtual Card Management

Users can create, activate, freeze, or delete virtual cards. Each card has a unique number, expiry date, and CVV generated securely. The cards are linked to users’ wallets and can be used for online payments. Card operations include viewing balance, retrieving transaction statements, and integrating with third-party payment providers for authorization and processing.

### Transaction Management

The system must record every wallet or card transaction, including deposits, withdrawals, and transfers. Each transaction has a timestamp, amount, source, destination, and status. This data supports auditing, analytics, and dispute resolution. The transaction module must ensure that operations are processed securely and logged for regulatory compliance.

### Admin Management

Administrators oversee user activities and system operations. Admins can view KYC documents, approve or reject verifications, monitor transaction logs, and manage flagged accounts. The admin interface should provide dashboards and reports to ensure transparency and accountability within the system.

## Non-Functional Requirements

1. **Security**

The system must ensure data confidentiality, integrity, and user privacy. All user credentials and financial information must be encrypted during storage and transmission. Secure authentication mechanisms such as **JWT tokens** and **role-based access control** must be implemented to prevent unauthorized access.

1. **Performance**

The platform should deliver quick and efficient responses, ensuring smooth user interactions. Core operations like login, wallet updates, and card creation must execute within **2–3 seconds** on average, even under heavy usage.

1. **Scalability**

The system must be able to handle increased user activity and transaction volumes without performance degradation. It should support **horizontal scaling** through modular architecture and load balancing to accommodate future growth.

1. **Reliability and Availability**

The platform must maintain continuous service availability, targeting an uptime of **99.9%**. Backup mechanisms and failover systems should be in place to ensure data recovery and business continuity during system failures.

1. **Usability**

The user interface should be intuitive and easy to navigate for all user types. Visual feedback, clear messages, and responsive design must ensure that users can perform actions like KYC verification, wallet top-up, or virtual card management with minimal effort.

## Resource Estimation

This section provides an overview of the expected resource requirements for the **Virtual Card Platform** based on projected usage, system interactions, and data storage needs. The goal is to anticipate how the system will scale over time and identify potential bottlenecks before deployment.

### ****User Estimation****

The platform is expected to serve a growing number of users as digital financial services gain adoption.  
We can classify user growth in three stages:

| **Stage** | **Expected Users** | **Description** |
| --- | --- | --- |
| **Initial Launch** | 1,000 – 5,000 users | Pilot phase, limited region or school-wide deployment |
| **Growth Phase** | 10,000 – 50,000 users | Expanded adoption with verified KYC onboarding |
| **Mature Stage** | 100,000+ users | Full public release with active transactions and card issuance |

**Implication:**  
At scale, the system should support thousands of **concurrent sessions**, requiring efficient authentication management, caching, and asynchronous background processing.

### ****Query Per Second (QPS) Estimation****

**QPS** measures how many read/write operations the system handles per second — across user logins, wallet checks, and transactions.

| **System Area** | **Typical Queries** | **Frequency Estimate** |
| --- | --- | --- |
| Authentication | Login, token refresh | Low frequency, spikes during business hours |
| Wallet | Balance checks, top-ups, transfers | Moderate, consistent throughout day |
| Transactions | Debit/credit logs, card spending | High frequency, often concurrent |
| KYC | Uploads, admin reviews | Moderate, asynchronous |
| Card API | Creation, freeze/unfreeze | Low, triggered by user action |

**Estimated QPS Range:**

* Initial phase: **5–20 QPS** (light load)
* Growth phase: **100–500 QPS** (moderate load)
* Mature phase: **>1,000 QPS** (requires load balancing)

**Solution:**  
Use **horizontal scaling** with load-balanced API servers and **database read replicas** to distribute the query load effectively.

### ****Potential Bottlenecks****

| **Bottleneck Area** | **Description** | **Mitigation Strategy** |
| --- | --- | --- |
| **Database Write Load** | High transaction volume may saturate single instance | Implement sharding or read/write separation |
| **KYC File Uploads** | Large files can slow down requests | Use background upload services and CDN caching |
| **Authentication Server** | Spikes during login hours | Use token-based sessions and distributed cache (Redis) |
| **Transaction Consistency** | Race conditions during balance updates | Use database transactions and locking |
| **External APIs** | Delays from third-party card/KYC services | Implement retries and asynchronous webhooks |

### . Summary

| **Aspect** | **Estimate** | **Scalability Plan** |
| --- | --- | --- |
| **Users** | Up to 100,000+ | Horizontal scaling & load balancing |
| **QPS** | 100–1,000+ | Database replication & caching |
| **Storage** | Tens of TBs (mostly KYC docs) | Cloud storage with encryption |
| **Compute** | 2–8 vCPUs per node | Auto-scaling cluster |
| **Network** | Moderate (API + file upload) | CDN + rate limiting |
| **Main Bottlenecks** | DB writes, KYC uploads | Queueing & sharding solutions |

## Core Entities and API Design

The Virtual Card Platform is composed of several **key entities** that represent users, identity verification, wallets, virtual cards, and financial transactions. Each entity interacts through well-defined **RESTful APIs** that enable modular and secure communication between the frontend, backend, and third-party services.

### ****Core Entities****

Below are the main entities and their relationships:

| **Entity** | **Description** | **Key Attributes** |
| --- | --- | --- |
| **User** | Represents a registered account holder who can create and manage virtual cards. | user\_id, name, email, phone, password\_hash, role, status, created\_at |
| **KYC Profile** | Stores identity verification details for each user. | kyc\_id, user\_id, id\_type, id\_number, photo\_id\_url, proof\_of\_address\_url, status, verified\_by, verified\_at |
| **Wallet** | Represents a user’s balance and financial activity. | wallet\_id, user\_id, balance, currency, last\_updated |
| **Virtual Card** | A digital payment card linked to a user’s wallet. | card\_id, user\_id, wallet\_id, card\_number, cvv, expiry\_date, status, limit, created\_at |
| **Transaction** | Records every debit/credit or card payment operation. | transaction\_id, wallet\_id, card\_id, amount, type, status, description, created\_at |
| **Admin** | Authorized personnel managing KYC approvals and system settings. | admin\_id, name, email, role, permissions, last\_login |
| **Notification** | System messages or alerts sent to users. | notification\_id, user\_id, type, message, read\_status, created\_at |

**Entity Relationships:**

* A **User** has one **KYC Profile** and one **Wallet**.
* A **Wallet** can have multiple **Virtual Cards**.
* Each **Virtual Card** can generate multiple **Transactions**.
* **Admins** oversee **KYC verification** and system auditing.

### ****REST API Endpoint Design****

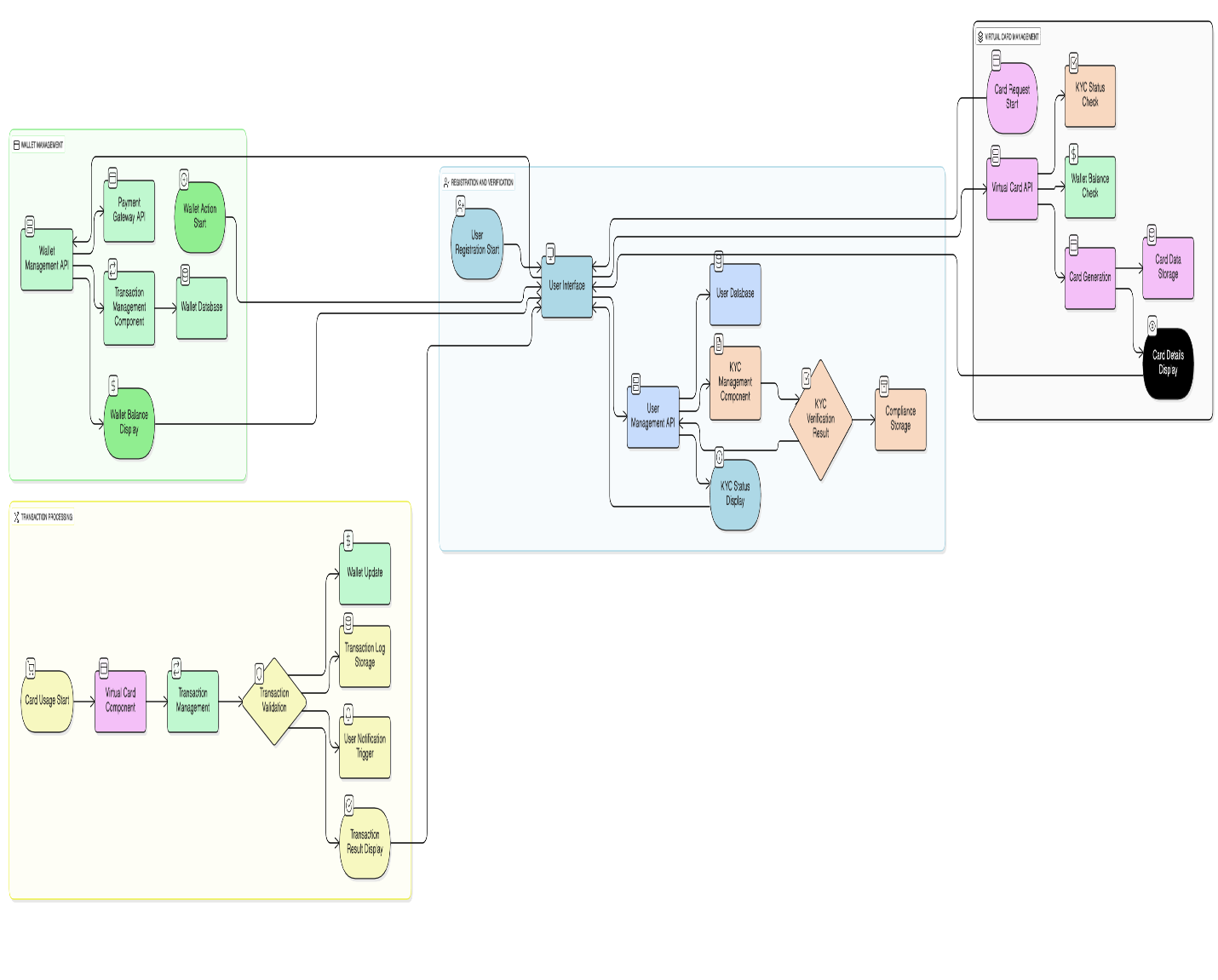
The API layer exposes secure REST endpoints grouped by entity and function. All APIs are versioned (e.g., /api/v1/...) and protected using token-based authentication (JWT).

| **Module** | **Purpose** | **Example Endpoints** |
| --- | --- | --- |
| **Authentication & Users** | Handle user registration, login, and profile management. | /api/v1/register, /api/v1/login, /api/v1/profile, /api/v1/change-password |
| **KYC Management** | Manage user identity verification and document submission. | /api/v1/kyc/submit, /api/v1/kyc/status, /api/v1/admin/kyc/review, /api/v1/admin/kyc/verify/{kyc\_id} |
| **Wallet Management** | Manage digital wallet balances, top-ups, and transfers. | /api/v1/wallet, /api/v1/wallet/topup, /api/v1/wallet/transfer, /api/v1/wallet/history |
| **Virtual Card Management** | Handle creation and control of user virtual cards. | /api/v1/cards, /api/v1/cards/create, /api/v1/cards/{id}/freeze, /api/v1/cards/{id}/delete |
| **Transactions** | Record and track all payment and transfer activities. | /api/v1/transactions, /api/v1/transactions/{id}, /api/v1/transactions/initiate |
| **Admin Panel** | Administrative controls for user, KYC, and transaction oversight. | /api/v1/admin/dashboard, /api/v1/admin/users, /api/v1/admin/transactions, /api/v1/admin/settings |
| **Notifications & Logs** | Manage system alerts and activity tracking. | /api/v1/notifications, /api/v1/notifications/read/{id}, /api/v1/logs/activity |

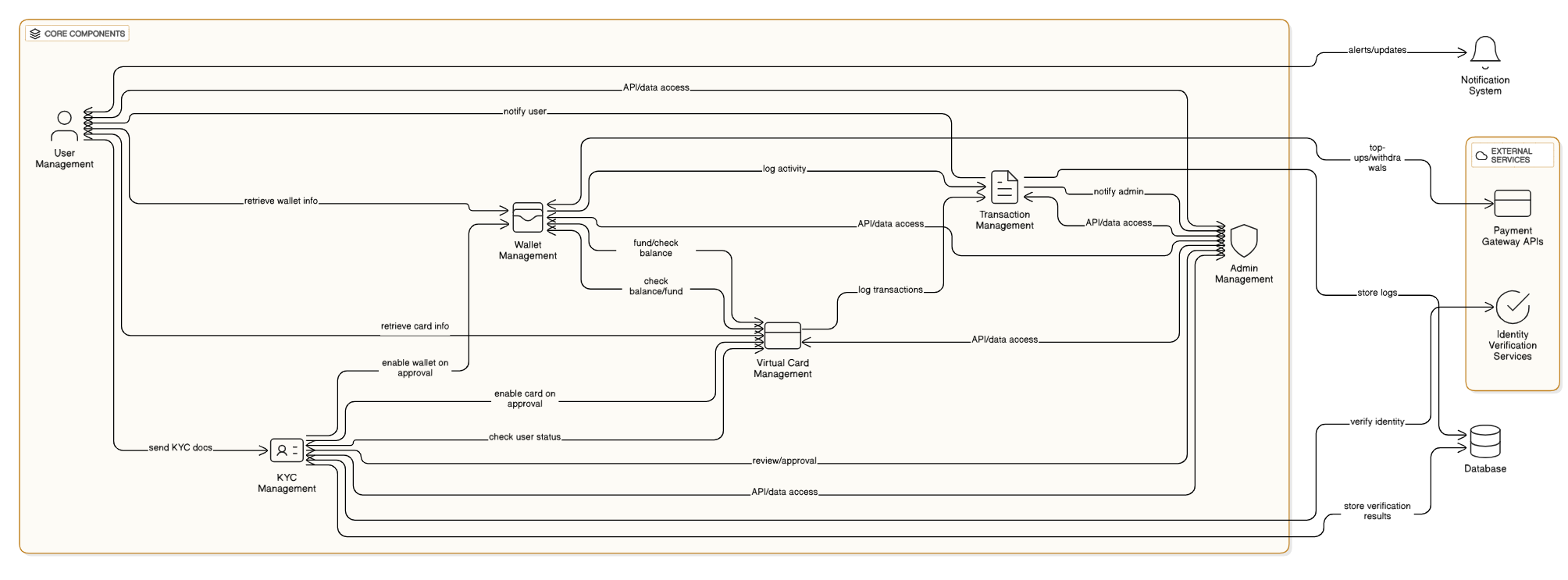
## High-level design

## The High-Level Design (HLD) section outlines the overall architecture and structure of the Virtual Card System. It provides a conceptual blueprint that illustrates how the system’s core components interact to deliver a secure, scalable, and efficient digital card service.

1. **Data Flow for the virtual card platform**

****

1. **Main components interaction**

****

## Detailed Design: Problems, Proposed Solutions, and Trade-Offs

This section provides a deep analysis of the design decisions made in the Virtual Card Platform. It identifies the **key implementation challenges**, presents **proposed solutions**, and discusses the **trade-offs** that arise between performance, cost, scalability, and security.

1. **KYC Management Complexity**

**Problem:**  
KYC verification involves collecting and validating user identity documents such as ID cards and proof of address. Manual verification can delay onboarding and increase administrative workload, while automated systems introduce integration and accuracy challenges.

**Proposed Solution:**

* Implement a **hybrid KYC workflow**: automatic verification (OCR and ID matching) combined with manual review for flagged cases.
* Store documents securely in **encrypted cloud storage** (e.g., AWS S3 or Azure Blob).
* Use a **message queue (RabbitMQ/Kafka)** to handle KYC verification asynchronously, so users don’t experience delays.

**Trade-Offs:**

* Automation reduces verification time but increases integration cost and complexity.
* Manual review improves accuracy but limits scalability.
* A hybrid model balances cost, speed, and compliance.

1. **Security and Data Protection**

**Problem:**  
User credentials, card details, and KYC documents are highly sensitive. A data breach could result in financial loss and regulatory penalties.

**Proposed Solution:**

* Encrypt all sensitive data at rest (AES-256) and in transit (TLS 1.3).
* Use **tokenization** for card numbers and **hashed passwords** (bcrypt).
* Apply **role-based access control (RBAC)** and maintain **audit trails** for every admin action.
* Periodically run **vulnerability scans** and enforce **multi-factor authentication (MFA)**.

**Trade-Offs:**

* Heavy encryption adds processing overhead but ensures compliance (e.g., PCI DSS).
* Strict access control may slow admin operations but reduces insider threats.
* Security measures increase cost but are non-negotiable for financial systems.

1. **Transaction Consistency and Concurrency**

**Problem:**  
Multiple users may perform transactions simultaneously, risking double spending or inconsistent wallet balances.

**Proposed Solution:**

* Use **ACID-compliant transactions** in the database to ensure atomic updates.
* Implement **transaction queues** or **locking mechanisms** to serialize balance updates.
* Introduce **idempotency keys** to prevent duplicate payments from retries.

**Trade-Offs:**

* Synchronous locking ensures accuracy but can increase latency.
* Asynchronous updates improve throughput but complicate reconciliation.
* Balancing strong consistency with performance requires careful transaction design.

1. **Scalability and Performance Bottlenecks**

**Problem:**  
As user numbers grow, the system must handle higher query rates and large transaction volumes without degradation.

**Proposed Solution:**

* Deploy **load balancers** and **horizontal scaling** for API servers.
* Use **database replication** (read/write separation) and **caching** (Redis) for frequent queries.
* Optimize queries and apply **pagination** in endpoints returning large datasets.
* Use **content delivery networks (CDN)** for faster access to KYC files and static assets.

**Trade-Offs:**

* Scaling horizontally adds infrastructure cost but improves availability.
* Caching boosts speed but risks stale data if not properly invalidated.
* CDN integration adds complexity but improves user experience globally.

1. **Integration with External Payment and Card APIs**

**Problem:**  
The platform relies on third-party APIs for virtual card creation and payment processing. These services can be slow or temporarily unavailable.

**Proposed Solution:**

* Implement **retry logic with exponential backoff** for failed requests.
* Use **webhooks and asynchronous callbacks** to handle delayed responses.
* Log all external API interactions for debugging and audit purposes.

**Trade-Offs:**

* Async communication improves resilience but increases system complexity.
* Dependency on third-party services introduces vendor lock-in risk.
* Building internal fallback mechanisms improves reliability but adds development effort.

1. **Data Storage and Growth Management**

**Problem:**  
KYC documents, transaction records, and logs will grow rapidly over time, risking database bloat and performance degradation.

**Proposed Solution:**

* Use **object storage** for large files and **relational databases** for structured data.
* Implement **data archiving** and **log rotation** policies.
* Employ **partitioning** or **sharding** in the database as data volume increases.

**Trade-Offs:**

* Archiving improves performance but makes old data less accessible.
* Sharding enhances scalability but complicates queries and backups.
* Object storage reduces cost per GB but adds latency compared to local storage.

1. **Monitoring and Maintenance**

**Problem:**  
Without proper monitoring, performance issues or system failures might go undetected, affecting reliability and user trust.

**Proposed Solution:**

* Use **centralized logging and monitoring tools** (e.g., Prometheus, Grafana, ELK Stack).
* Set up **alerting systems** for downtime, error spikes, and transaction failures.
* Schedule regular **backups** and define a **disaster recovery (DR)** plan.

**Trade-Offs:**

* Monitoring tools add operational cost but prevent major outages.
* Frequent backups consume storage but ensure fast recovery.
* Real-time analytics improve visibility but increase system load.

1. **User Experience and Accessibility**

**Problem:**  
Complex interfaces or slow verification processes can discourage user adoption.

**Proposed Solution:**

* Design an intuitive frontend with clear onboarding steps.
* Provide real-time KYC and transaction status updates.
* Implement responsive design and multilingual support.

**Trade-Offs:**

* Enhanced UX requires more frontend development effort.
* Real-time updates increase API calls but improve transparency.
* Accessibility features expand reach but add design complexity.

1. **Summary of Trade-Offs**

| **Area** | **Key Trade-Off** |
| --- | --- |
| KYC Workflow | Speed vs Accuracy |
| Security | Performance vs Compliance |
| Transactions | Consistency vs Latency |
| Scalability | Cost vs Availability |
| Integrations | Simplicity vs Resilience |
| Storage | Cost vs Accessibility |
| Monitoring | Operational Overhead vs Reliability |
| User Experience | Development Effort vs Adoption |

**Conclusion**

The detailed design of the Virtual Card Platform emphasizes **security, modularity, and scalability**. Every design decision was made by weighing **trade-offs** between cost, complexity, and performance. By combining hybrid KYC processing, secure data practices, scalable infrastructure, and resilient integrations, the system ensures both **regulatory compliance** and **high user satisfaction**.