System Design Overview

1. Executive Summary

This presents the system architecture for a virtual payment platform designed to process digital payments through virtual cards. The platform integrates authentication, card management, transaction processing, and wallet services.

2. System Architecture Overview

The platform consists of three main architectural layers:
Client Layer: Web browsers and mobile applications that provide user interfaces for interacting with the platform.
API Gateway: Central entry point that routes requests, handles authentication, and manages API rate limiting.
Service Layer: Contains microservices for authentication, card management, transactions, and wallet operations.

3. Client-Server Architecture

The system implements a standard client-server model where clients initiate requests to the API Gateway. The gateway forwards requests to the Application Server, which processes business logic and interacts with the database. This architecture provides clear separation between presentation and business logic layers. This beneficial as it provides scalability through horizontal scaling, security through centralized authentication, and maintainability through modular design.

4. Redis Caching Implementation

Redis serves as an in-memory data store that significantly reduces database load and improves response times. The caching strategy follows a cache-aside pattern where the application checks Redis before querying PostgreSQL.

Cache Hit Scenario: When data exists in Redis, it is returned immediately without database access, reducing latency from milliseconds to microseconds.
Cache Miss Scenario: When data is not in Redis, the system queries PostgreSQL, stores the result in Redis for future requests, and returns the data to the client.
Common cached data includes user session tokens, frequently accessed card details, and transaction history.

5. Service Layer Architecture

The service layer implements a microservices architecture with four primary services:
Authentication Service: Manages user registration and login, JWT token generation, and session management using Redis for token storage.
Card Service: Interfaces with third-party card provider APIs to create, manage, and deactivate virtual cards for users.
Transaction Service: Processes all financial transactions and maintains transaction history.
Wallet Service: Manages user wallet balances, integrates with payment gateways for deposits and withdrawals.

6. Technology Stack & Justification

Programming Language: Java
Java is selected as the primary programming language for the following reasons:

- Java has extensive libraries and frameworks like Spring Boot that are specifically designed for building secure, scalable financial applications.
- Built-in security APIs, strong typing, and established security practices make Java ideal for handling sensitive financial data.
- Ensure high-performance transaction processing.
- Spring Boot, Spring Security, and Spring Data provide robust frameworks for API development, authentication, and database integration.
- Widely used in fintech and banking sectors, ensuring availability of experienced developers and community support.

Database: PostgreSQL

PostgreSQL is chosen as the relational database management system because:

- Guarantees reliable financial transactions with full support for atomicity, consistency, isolation, and durability.
- Strong constraint enforcement and foreign key relationships ensure data consistency across tables.
- Native JSONB data type allows flexible storage of third-party API responses while maintaining relational structure.
- Advanced features like table partitioning, replication, and connection pooling support high transaction volumes.
- Open Source: No licensing costs with strong community support and extensive documentation.

Caching: Redis

Redis is integrated for high-performance caching and session management:

- In-memory operations provide sub-millisecond response times for frequently accessed data.
- Perfect for storing JWT tokens and user sessions with automatic expiration.
- Caching frequently accessed data reduces PostgreSQL queries by up to 80%.

7. Conclusion

The proposed architecture provides a scalable, and secure foundation for the virtual payment platform. The combination of Java with Spring Boot, PostgreSQL for data persistence, and Redis for caching delivers an great solution capable of handling high transaction volumes while maintaining data integrity and security.

The microservices architecture ensures modularity and allows independent scaling of services based on demand. Integration with third-party card providers and payment gateways is handled through well-defined service interfaces, promoting maintainability and future extensibility.