SYSTEM ARCHITECTURE FOR VITUAL PAYMENT APP

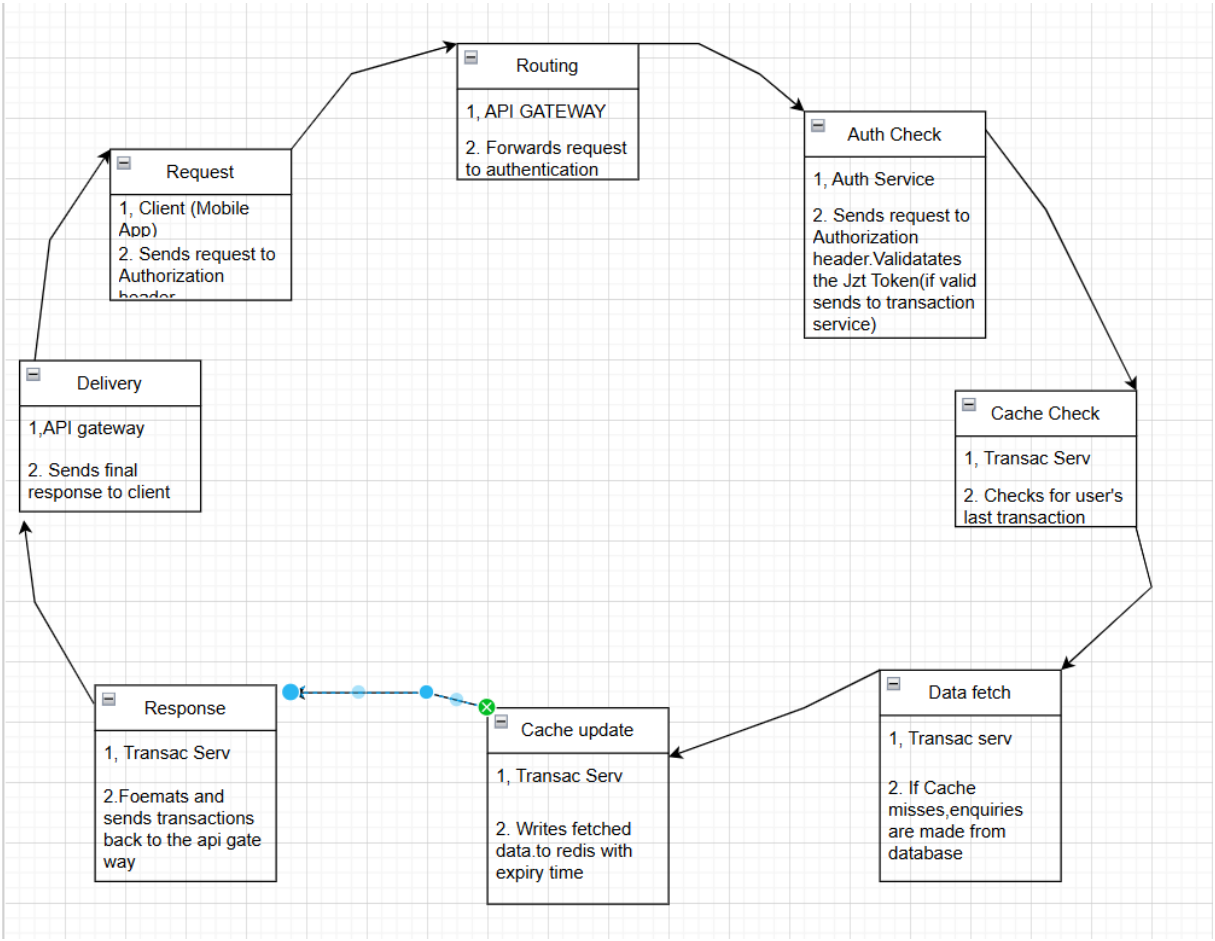
TASK 1: Using a digital design software such as; figma or draw.io, draw a client sever architecture model for the request/response cycle.

STEP	COMPONENT	ACTION
Request	Client (Mobile App)	Sends a request with an Authorization Header
Routing	API Gateway	Forwards the request to the Authentication Service.
Authe check	Auth Serv	validates the JWT token. If valid, forwards to the Transaction Service.
Cache Check	Transac serv	Checks for user's latest transact
Data fetch(miss)	Transac serv	If cache miss, queries the Primary Database (e.g., PostgreSQL,MYSQL).
Cache update	Transac serv	Writes the fetched data back to Redis (set \text{key}) with an expiry time.
Response	Transac serv	Formats and sends the transaction data back to the API Gateway
delivery	Api gateway	Sends final response to the client

Components:

1. 1 Client (Mobile App/Web): Initiates the request (e.g., "View Card Details," "Check Wallet Balance").
2. API Gateway/Load Balancer: First point of entry. Routes the request to the appropriate service.
3. Authentication Service: Validates the User's Token (JWT). If valid, passes the request. If invalid, returns a 401 (Unauthorized) response.
4. Backend Micro service (e.g., Card Service/Wallet Service):
5. Cache Check (Redis): Checks Redis for cached data (e.g., recent transactions, card metadata).
6. DB/Third-Party Call: If not in cache, the service fetches data from the primary Database (PostgreSQL/MongoDB) or calls the relevant Third-Party Service (Virtual Card Provider or Wallet Provider).
7. Cache Update (Redis): Stores or updates the fetched data in Redis (Cache-Aside Pattern).
8. Response Formation: The service formats the data.

9. Response Path: The formatted response travels back through the API Gateway to the Client.



1.1 client sever architecture model for the request/response cycle

TASK 2: How Redis Functions in Data Caching (and Architecture Diagram)?

(Redis is an in memory key value database used as a distributed cache and message broker,with optional durability. Its primary advantage for caching is its speed, as it stores data in RAM. A cache is a high-speed temporary storage area that holds copies of frequently accessed data to make future requests faster**).**

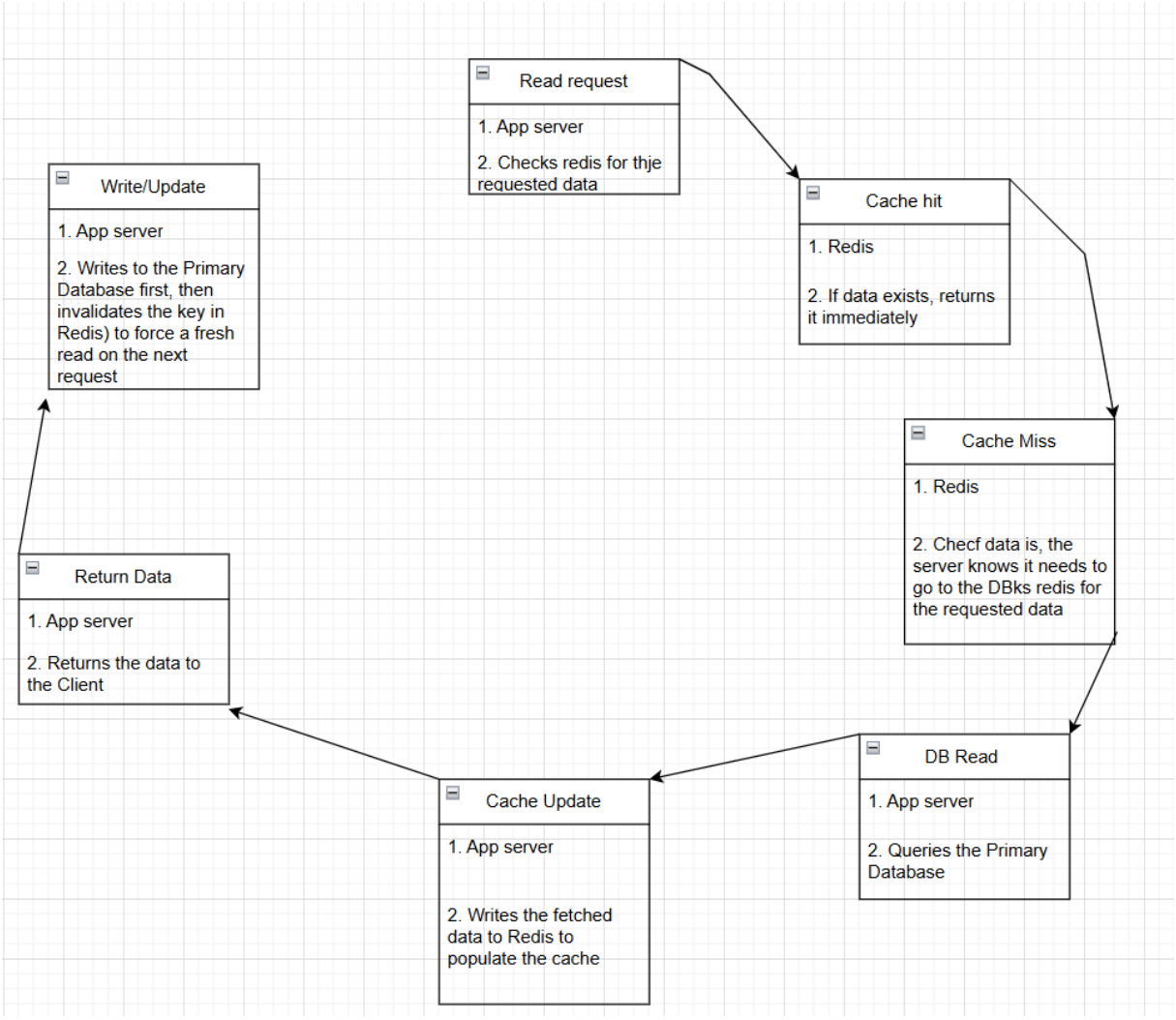
Function in Data Caching:

- Speed: As an in-memory store, it offers sub-millisecond latency for read/write operations, significantly faster than traditional disk-based databases.
- Use Cases:
- Transaction Caching: Storing the 10 most recent transactions for a user.
- Session Management: Storing user session tokens for fast authentication validation.

- Rate Limiting: Tracking the number of requests a user makes within a time window.
- Data Structures: Its rich data structures (Strings, Hashes, Lists, Sets, Sorted Sets) allow for flexible caching strategies (e.g., using a Sorted Set to store transactions ordered by timestamp).
- Eviction Policies: Redis can automatically remove keys based on rules (e.g., Least Recently Used - LRU) when memory limits are reached, ensuring it only holds the most relevant data.

Architecture Diagram (Cache-Aside Pattern)

- Focus: Demonstrates the Cache-Aside Pattern, the most common and robust caching mechanism.
- Components: Client, Application Server (Backend Service), Redis Cache, and Primary Database.



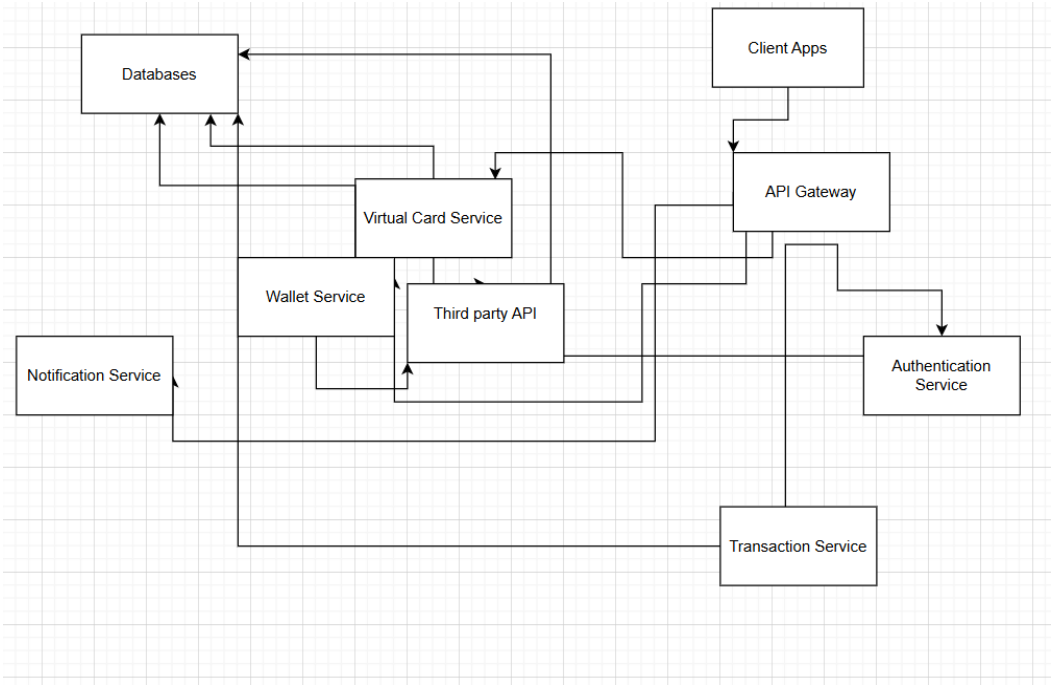
2.1 Redis Functions Architecture Diagram

TASK 3: Overall System Architecture:

Module	Purpose	Technology Interaction
Client Apps	Mobile (iOS/Android) \& Web Frontend	Calls the API Gateway
API Gateway	Entry point, Load Balancing, Security/Throttling.	Routes traffic to Microservices
Authentication Service	User registration, Login, JWT generation/validation.	Database (User Mgmt), Redis (Session Cache).
Transaction Service	Core payment logic, recording all transactions	Database (Transaction Records), Redis (Transaction Cache/Ledger checks)
Virtual Card Service	Handles card creation, activation, and status updates	hird-Party API (Virtual Card Provider), Database (Card Metadata)
Wallet Service	Manages user balances and fund transfers	Third-Party API (Wallet Provider), Database (Local balance/audit log)
Notification Service	Sends SMS/Email/Push notifications for transactions	External {Email} Providers
Databases	Primary data storage (User \& Transaction history)	PostgreSQL (Transactional integrity)

Architecture Flow:

i. Client interacts only with the API Gateway.
ii. The API Gateway routes to the relevant Microservice.
iii. Microservices use Redis for fast lookups (transactions, sessions) and the Primary Database for persistent storage.
iv. The Virtual Card and Wallet Services communicate with External Third-Party APIs for their core functions.



3.1 System Architecture

TASK 4: Programming language and DBMS justification

i. **Go (Golang)**:

- **Concurrency**: Go's native support for concurrency with goroutines and channels is highly efficient for handling a high volume of simultaneous requests.
- **Performance**: As a compiled language, it offers excellent performance, crucial for low-latency transaction processing.
- **Simplicity**: Go's simple syntax and tooling lead to faster development cycles and easier maintenance.
- **Microservices**: It is a perfect fit for a microservices architecture.

ii. **Java**:

- **Mature Ecosystem**: Java has a long-standing reputation in the enterprise and financial sectors with robust libraries and frameworks like Spring Boot.
- **Security**: The Java Virtual Machine (JVM) provides a secure environment, and the ecosystem is well-equipped with security APIs.
- **Scalability**: Java scales both vertically and horizontally, suitable for high-transaction volumes.

For this specific project, **Go is the recommended choice** due to its superior performance in concurrent operations, which aligns perfectly with a high-throughput, transaction-based system leveraging Redis.

Database Management System (PostgreSQL)

For the primary system of record, a relational database is the most suitable choice because financial data requires strong consistency and transactional integrity (ACID compliance(atomicity, consistency, isolation, and durability)).

i. **PostgreSQL**:

- **ACID Compliance**: Ensures that transactions are processed reliably, critical for financial data where data integrity is non-negotiable.
- **Reliability and Security**: It is known for its robustness, data integrity, and strong security features, with a long and trusted history in the industry.
- **Extensibility**: PostgreSQL is highly extensible and supports various data types, including native JSON support for flexibility.
- **Scalability**: Supports both vertical and horizontal scaling with techniques like sharding to handle growing data loads.
- **Open-source**: As an open-source solution, it avoids vendor lock-in and has a strong community