## ****Design Architecture for Banking Microservices Application****

### ****Overview****

The banking application is built using a microservices architecture. Each service is responsible for a specific business capability and communicates with other services via REST APIs or message queues. The **authentication-service** acts as the centralized service for authentication and authorization, and all other services must consume it to validate user requests.

### ****Architecture Diagram****

Below is a textual representation of the architecture. You can visualize it using a diagramming tool.

1. **Client Layer**:

* Web Application (React js/ Next js)
* Mobile Application (Kotlin/Flutter)

1. **API Gateway:**

* Acts as the entry point for all client requests.
* Routes requests to the appropriate microservice.
* Handles cross-cutting concerns like logging, rate limiting, and SSL termination.

1. **Microservices Layer:**

* **Authentication Service (Spring Boot):**
  + Centralized service for user authentication and authorization.
  + Issues JWT tokens for secure communication.
* **History Service (Spring Boot):**
* Tracks and stores transaction history (Deposit, Withdraw/Transfer, Bill Payments, Cable Subscription, Flight Booking, Hotel Booking, GiftCard Purchase and so on)**.**
* **Bank Collection Service (Golang):**
  + Handles/ Storing User bank-related operations like account creation, updates, etc when user deposited FOR future used for the user.
* **Deposit Service (Spring Boot):**
  + Manages deposit transactions.
* **Withdrawal Service (Spring Boot):**
  + Manages withdrawal transactions.
* **Notification Service (Spring Boot):**
  + Sends notifications (email, SMS, push) to users using RabbitMQ message broker.
* **Blacklist Service (Golang):**
  + Manages blacklisted users or transactions.
* **Revenue Service (Spring Boot):**
  + Tracks revenue and financial metrics.
* **Beneficiaries Service (Golang):**
  + Manages beneficiary-related operations.
* **Maintenance Service (Spring Boot):**
  + Handles debt recovery and related operations.
  + Handle platform fee Crons Job Schedule.
* **Wallet Service (Spring Boot):**
  + Manages all users digital wallet operations balance.

1. **Data Layer:**
   1. Each microservice has its own database “Pesco\_db” using (MySQL, PostgreSQL, MongoDB).
   2. Shared caching layer (Redis) for performance optimization.
   3. Message queue (RabbitMQ) for asynchronous communication.
2. **Infrastructure Layer:**
   1. Docker for containerization.
   2. Kubernetes for orchestration.
   3. CI/CD pipeline (Jenkins/GitLab CI) for automated deployments.
   4. Monitoring and logging (Prometheus, Grafana, ELK Stack).