Phase 3: Software Architecture Design – SwiftPay

Overview

The system architecture of SwiftPay outlines how different components of the mobile application will interact to deliver financial services such as registration, login, balance inquiry, money transfer, and more. The system is designed to be scalable, secure, and maintainable.

Architecture Type: Layered (Three-Tier) Architecture

SwiftPay follows a three-tier architecture, which includes:

1. Presentation Layer (Frontend – Mobile App)

• Purpose: This is what the user sees and interacts with.

• Technology: Developed using mobile app frameworks (e.g., Flutter, React Native).

• Functions:

• User Registration & Login

• View Balance & Transaction History

• Perform Transfers or Payments

• Display Notifications

• Access Financial Education Content

2. Application Layer (Backend – Business Logic)

• Purpose: This layer processes all user requests and handles business logic.

• Technology: Typically built using Node.js, Django, or Laravel.

• Functions:

• Authenticate users

• Process and validate transactions

• Handle multi-currency logic

• Communicate with the database

• Manage user sessions

• Trigger push notifications

3. Data Layer (Database – Storage)

• Purpose: Stores all persistent data securely.

• Technology: Could use MySQL, PostgreSQL, or MongoDB.

• Data Stored:

• User profiles and credentials (encrypted)

• Transaction records

• Currency exchange rates

• Notification logs

• Saved educational content bookmarks

Security Considerations

• All data transfers use HTTPS encryption.

• User data and passwords are encrypted in the database.

• Implements multi-factor authentication (MFA) for sensitive operations.

• Logs and monitors suspicious activities for fraud detection.

System Interactions (Example Flow)

1. User logs in via the frontend.

2. Request goes to the backend, which authenticates the user.

3. Backend queries the database for user data.

4. Response is sent back to the frontend and displayed to the user.

Scalability & Performance

• The system can be deployed on cloud platforms (e.g., AWS, Heroku).

• Load balancers and caching (e.g., Redis) can be used for high performance.

• Microservices architecture can be adopted in the future for better modularity.