System Architecture

Software Architecture is a high-level structure of the software system that includes the set of rules, patterns, and guidelines that dictate the organization, interactions, and the component relationships

Assignment

Design a system architecture for the following scenario.

1) Overall site architecture and interaction for a financial system that offers virtual cards to users for payments processing with the following modules.

   i. **User Registration:** Allow users to register by providing necessary information.

      - **Login:** Authenticate users based on their credentials.
      - **Forgot password:** Provide a secure process for users to recover their passwords.
      - **Session Management:** Efficiently manage user sessions to ensure security.
      - **Access Control:** Define roles and permissions for different user types.

   ii. user management

   iii. card provision (3th party)

   iv. kyc(know your customer) modules

   v. rules engine(fraud detection)

   vi. payment processing (3rd party)

   vii. notifications

   viii. cashing(ratis)

what stack will you choose for such a system (mobile app) and hence draw a request-response architecture for the server and client.

Document

Architecture documentation document(pdf)

PowerPoint presentation of system architecture

A github with all architecture files

carldrake969@gmail.com

functional requirements

These are things the system must do to meet the needs of the users. It is a description of what feature the system should have, accomplish and how the system should behave.

Functional requirements of our financial system offering virtual cards to users include:

1. User authentication (login feature)
2. card creation
3. payment processing
4. notifications
5. user management(example profile management)
6. card creation (user request for virtual cards from the financial institution)
7. cashing
8. Fraud detection

non-functional requirements

These are things that the system should perform rather that what it should do. It focuses on the quality of the system

Non-functional requirements of our financial system offering virtual cards to users may include:

1. reliability: The system should be available when needed by the user
2. efficiency: the system should be able to response to user request in a short period of time
3. Scalability: The system should be able to serve millions of users.
4. Security : all user data must be encrypted and the user must be lock from account after 5 failed login attempts.(authentication,authorization and data protection standards)
5. Maintainability: how easy it is to modify and test the code

Architectural views of the system

This provides the visual diagrams and models, detailing structure of the system from different views

1. Conceptual View(context diagram): this shows the system as a whole and its interactions with external entities (users, third-party services amd other systems)
2. Logical view(component view): breaks the system down into major logical components(example API Gateway, User Service, Notification shows the relationships and dependencies between these components. Defines the overall architecture (monolithic, microservices), shows how data flows through the system (example user registration)
3. physical view (deployment view): links the logical components to the physical infrastructure(servers,cloud services)

Technical decisions

1. Architecture: it defines how components (like DB,UI and Servers) interact, how data flows, and how the system is organized to meet users needs.
    - For building a financial system offering virtual cards to customer we will consider using the micro-service architecture.
    - **Micro-service architecture:** this architect breaks the system into small, independent services, each handling one task(example user authentication, payment processing,user management etc). Micro-service architecture is mostly used for large, scalable systems needing frequent updates.
2. Technology Stack:

✔ Programming language and framework: java/spring boot

- **java**: java is a powerful and widely used object-oriented programming language with a strong error handling, built-in security mechanisms and platform independent.
- **spring/spring boot**: spring is a framework designed to build java applications. It enables us to manually configure our java projects

    spring boot is built on top of spring and it is designed to simplify and speed up development. It comes with auto-configuration, starter dependencies and an embedded server(eg Tomcat). It removes manual setup required in spring, making it perfect for micro-services, REST API etc

- ✔ Database:PostgreSQL: it is an open-source relational database management system(RDMS). It follows the SQL but also includes modern features.
    - ◆ PostgreSQL ensures data integrity by supporting atomicity,consistency,isolation and durability in transactions
    - ◆ Postresql supports json,arrays,xml,and custom data types

security measures:

- Trade-offs and risks:

Architectural Diagrams

Low-Level Design(LLD) for Authentication System Design

Low-level design majorly focuses on component and module of the system. It focuses on the actual implementation details, algorithms, and data structures.

High-Level Design(HLD) for Authentication System Design

High-level design provides a indepth overview of the overall system architecture, which describes the interaction between major components. It mainly focus on the system's structure, major modules, and the flow of data.

User Management Microservice:

This microservice handles tasks related to user registration, profile management, and user data storage. It includes functionalities such as creating new user accounts, updating user information, and handling account deletion requests.

API Endpoints:

- /register: Create a new user account.
- /update/:userId: Update user information.
- /delete/:userId: Delete a user account.

Authentication Microservice:

Responsible for verifying user credentials during the login process, implementing multi-factor authentication (MFA), and generating authentication tokens. This microservice ensures the security of the authentication process.

API Endpoints:

- /login: Authenticate user credentials.
- /logout: End user session and revoke authentication tokens

Authorization Microservice:

Manages access control and permissions based on user roles. This microservice ensures that authenticated users have the appropriate permissions to access specific resources or perform certain actions.

API Used for our Authentication System

APIs (Application Programming Interfaces) serve as the communication channels between different microservices and external components. The APIs define the rules and protocols for how different software components should interact. In the context of an authentication system, various APIs are used for seamless communication between microservices:

Token-Based APIs:

For secure communication and data exchange, token-based APIs, such as JSON Web Tokens (JWT), are often employed. JWTs can be used to carry authentication information securely between microservices without the need to repeatedly verify credentials.

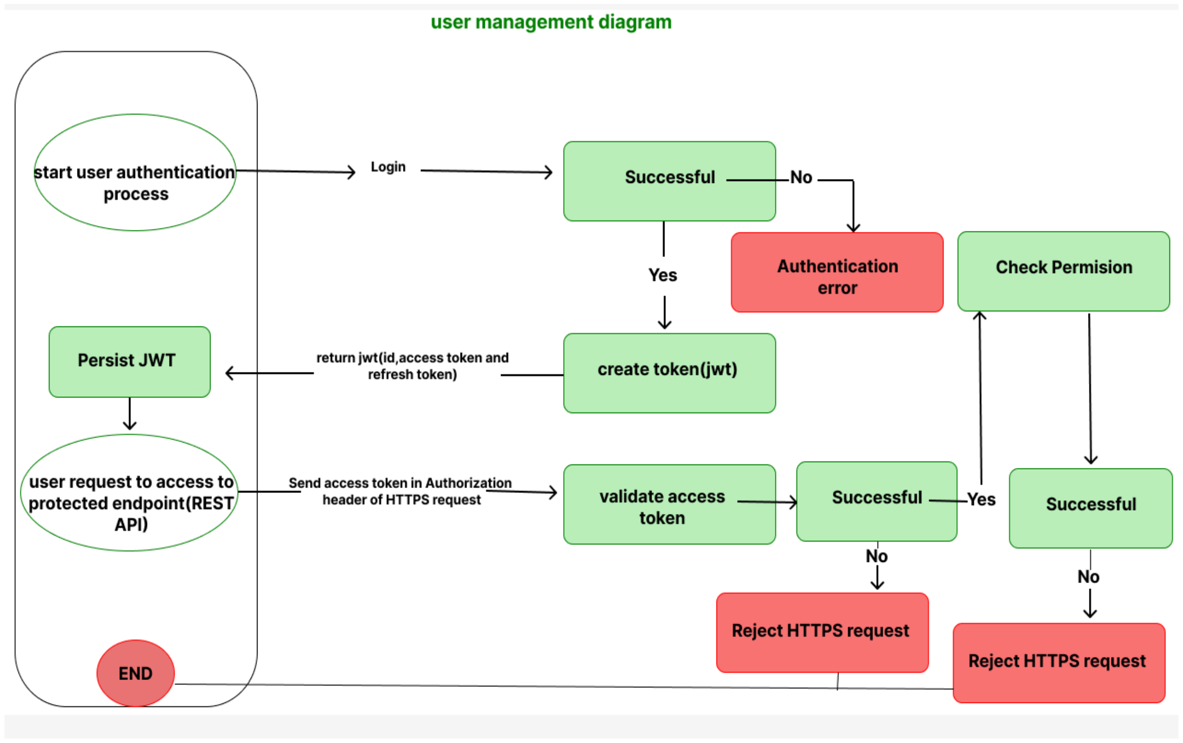
user management

<u>elements</u>

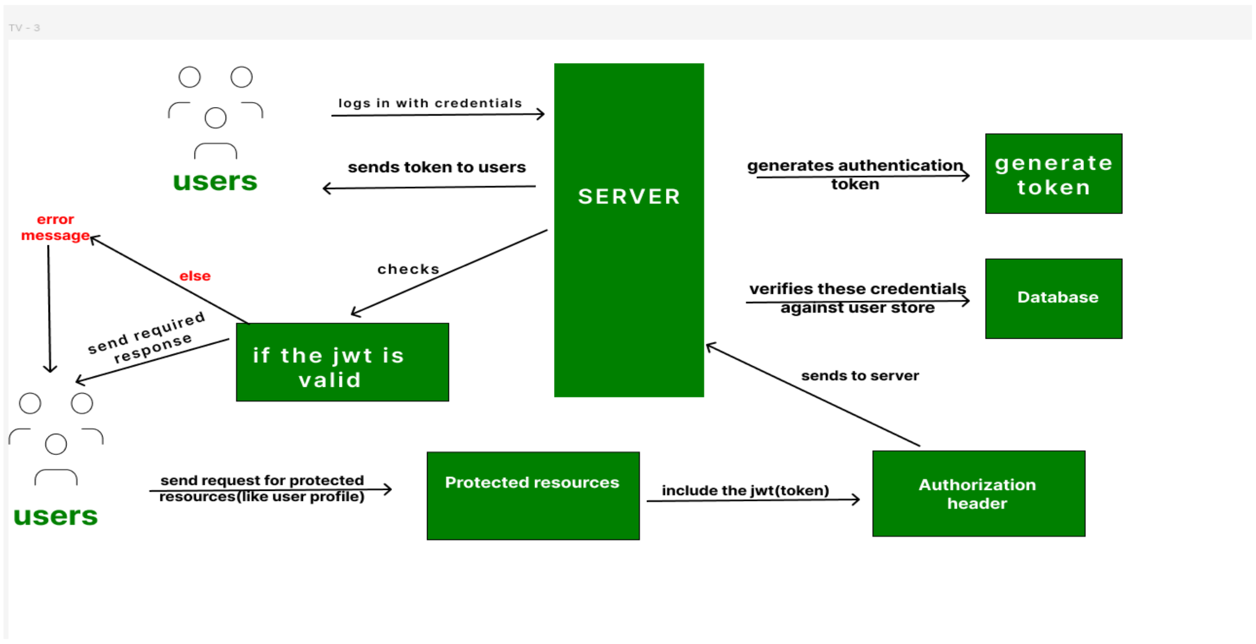
- **Authentication:**
  The process of verifying a user's identity, often through passwords.
- **Authorization:**
  Determines what actions a user is allowed to perform within the system after their identity has been verified.
- **Access Control:**
  Manages and defines which resources a user can access, often by assigning specific permissions.
- **User profile:**
  Contains detailed information about each user, such as name, email,

and contact details.

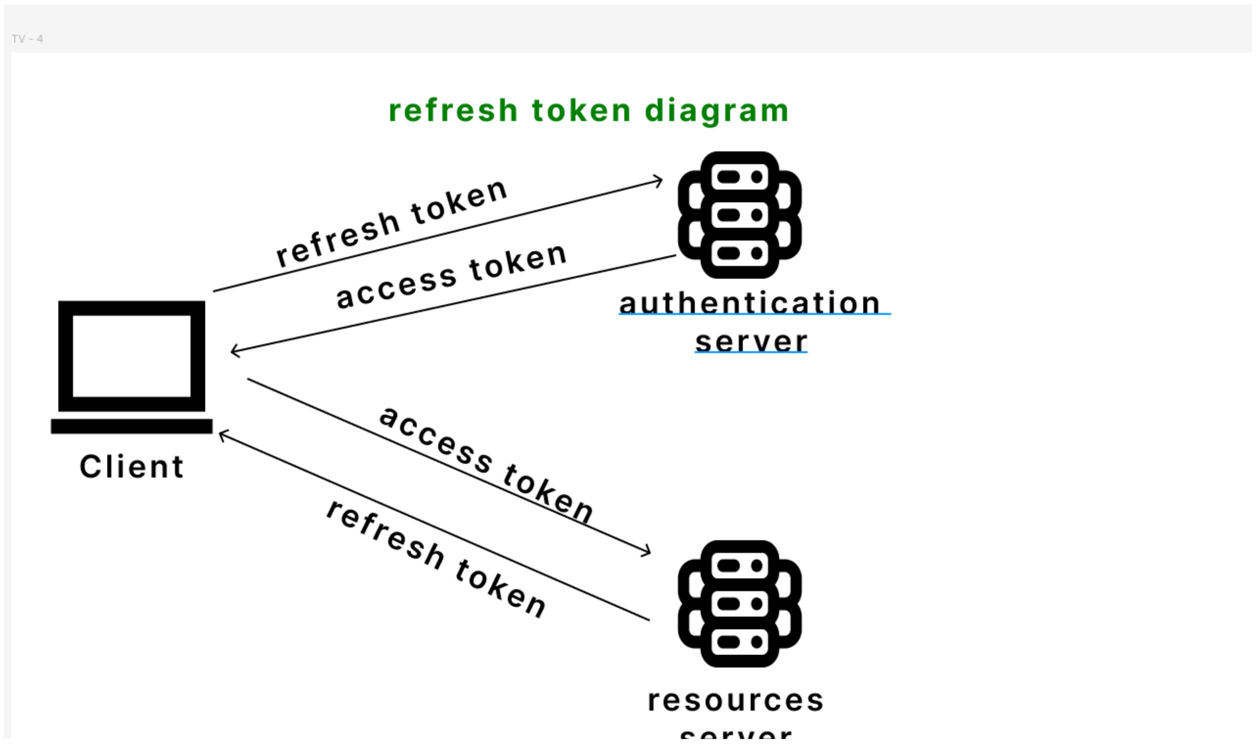
- **Auditing and monitoring**
  Tracks user activity to detect suspicious behavior and ensure compliance, often by creating an audit trail.



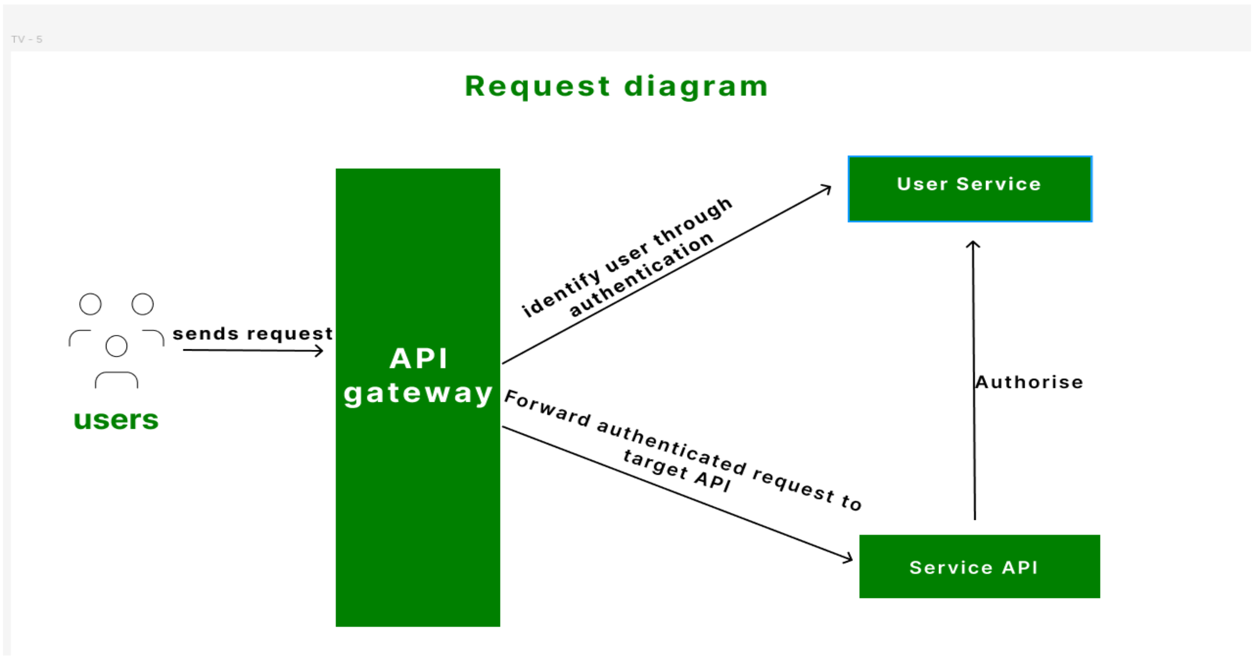
Authentication



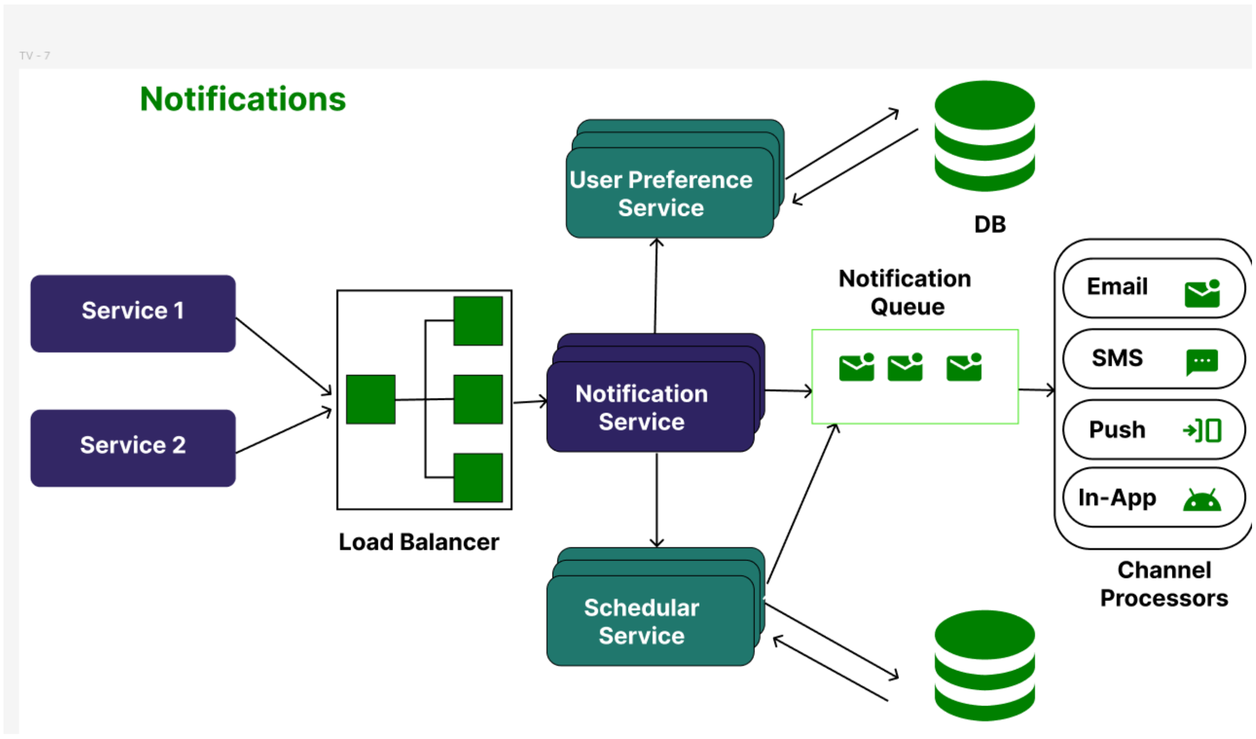
Refresh token



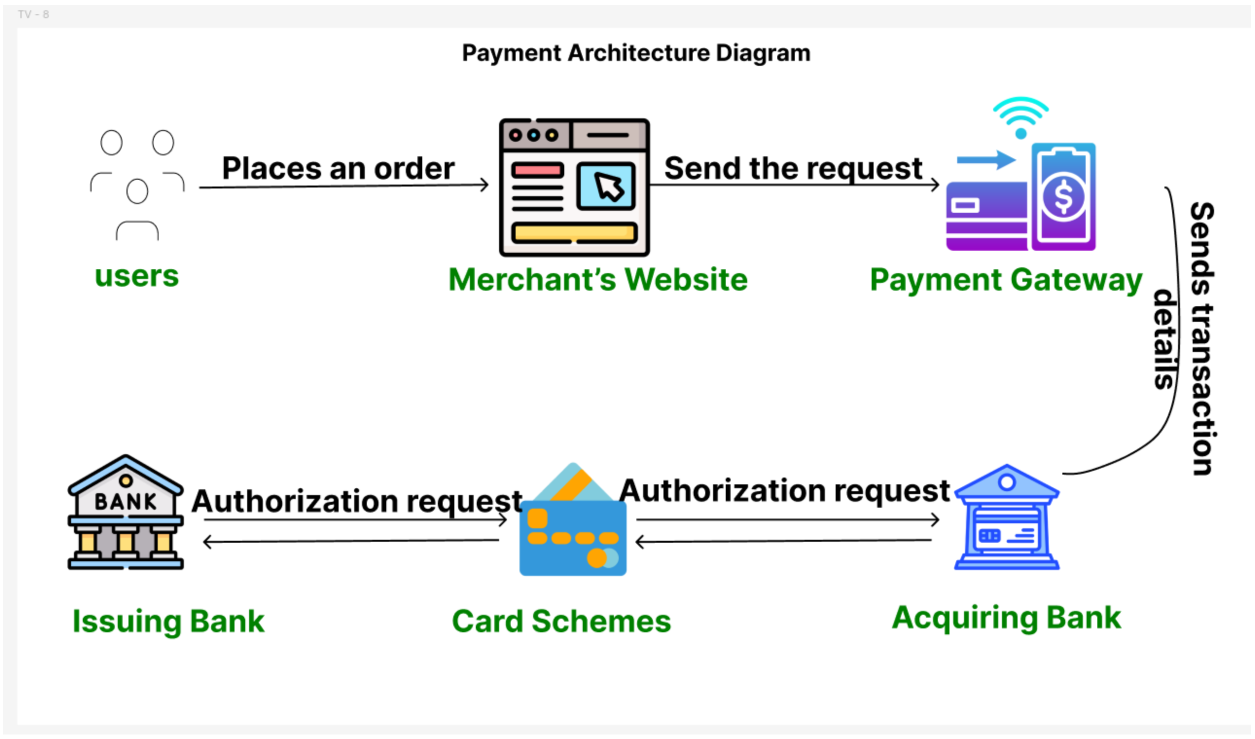
User-Request Diagram



Notifications



Payment Processing

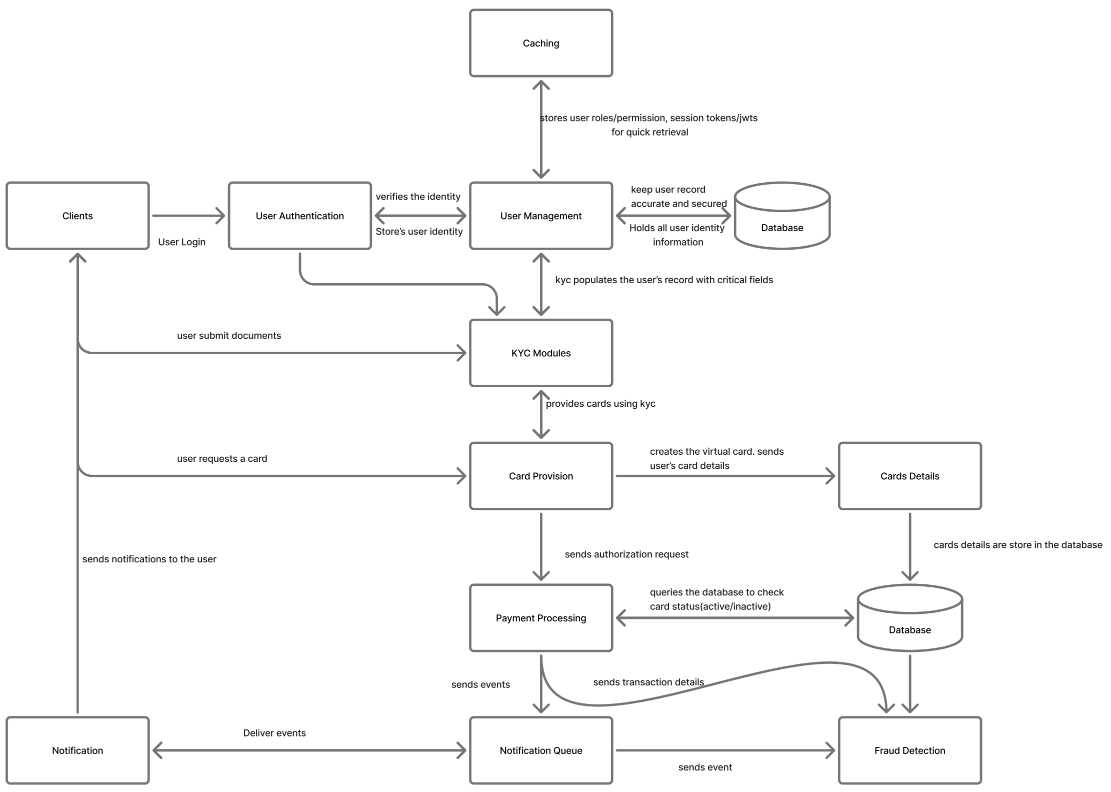


Virtual Card Creation

- **User/Client Request:** The user requests a new card via the User Portal or API. The request hits the **API Gateway**.
- **API Gateway:** Authenticates the user and routes the request to the **Card Management Service (CMS)**.
- **CMS:** Validates the request (limits, KYC status) via the **User & KYC Service**.
- **CMS:** Generates a unique **Virtual Card Number (VCN)** and other card details (e.g., CVV, Expiry).
- **CMS to Tokenization Service:** The CMS sends the **VCN** to the highly secured **Tokenization Service**.
- **Tokenization Service:** Stores the VCN securely (e.g., in a Hardware Security Module/HSM or encrypted vault) and returns a non-sensitive **Token** back to the CMS.
- **CMS:** Stores the card metadata and the **Token** (but *not* the VCN/PAN) in its database.
- **CMS to Ledger Service (Asynchronous):** CMS sends an event (e.g., CardCreatedEvent) via the **Message Broker** to notify the **Ledger Service** to create an initial account record tied to the new card.
- **Ledger Service:** Updates its internal records, ready to track transactions.
- **Response:** CMS returns the successful creation confirmation and the Card Token (for display) to the user via the API Gateway.
- **User & KYC Service**

  Manages user accounts, identity verification (Know Your Customer/KYC), and profile data.

System architecture design of a financial system offering virtual cards to customers


Caching
stores user roles/permission, session tokens/jwts for quick retrieval
Clients
User Login
User Authentication
verifies the identity
Store's user identity
User Management
keep user record accurate and secured
Holds all user identity information
Database
kyc populates the user's record with critical fields
user submit documents
KYC Modules
provides cards using kyc
user requests a card
Card Provision
creates the virtual card. sends user's card details
Cards Details
cards details are store in the database
sends notifications to the user
sends authorization request
Payment Processing
queries the database to check card status(active/inactive)
Database
sends events
sends transaction details
Notification
Deliver events
Notification Queue
sends event
Fraud Detection