Designing Authentication System | System Design

Keeping your important digital information safe is like building a strong foundation, and the key to that is a good security system. This article will help you understand how to make a robust security system step by step. We'll start with figuring out what you want to protect and what you want to achieve. Then, we'll talk about the detailed design aspects, like how the system works at both the small and big levels, the structure of the database, using smaller specialized services, and making sure the system can handle more load without slowing down.

Important Topics for the Authentication System Design

- Requirements Gathering for Authentication System Design
- Capacity Estimation for Authentication System Design
- Use Case Diagram for Authentication System Design
- Low-Level Design(LLD) for Authentication System Design
- High-Level Design(HLD) for Authentication System Design
- Database Design for Authentication System Design
- Microservices used for Authentication System Design
- API Used for Authentication System Design

- API Code Implementation for Authentication System
- Scalability for Authentication System Design

1. Requirements Gathering for Authentication System Design

Functional Requirements for Authentication System Design

- **User Registration:** Allow users to register by providing necessary information.
- **Login:** Authenticate users based on their credentials.
- **Multi-Factor Authentication (MFA):** Implement a robust MFA system.
- **Password Recovery:** 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.
- **Audit Trail:** Maintain detailed logs of authentication events for auditing.

Non-Functional Requirements for Authentication System Design

- **Security:** Prioritize data security through encryption, secure storage, and secure communication.
- **Scalability:** Design the system to handle a growing number of users and transactions.
- **Performance:** Ensure low latency and quick response times.
- **Reliability:** Minimize system downtime and ensure high availability.
- **Usability:** Develop an intuitive user interface for a seamless experience.

2. Capacity Estimation for Authentication System Design

You can estimate the system capacity by analyzing certain data like traffic, number of user coming on site. Here is the simplified calculation given:

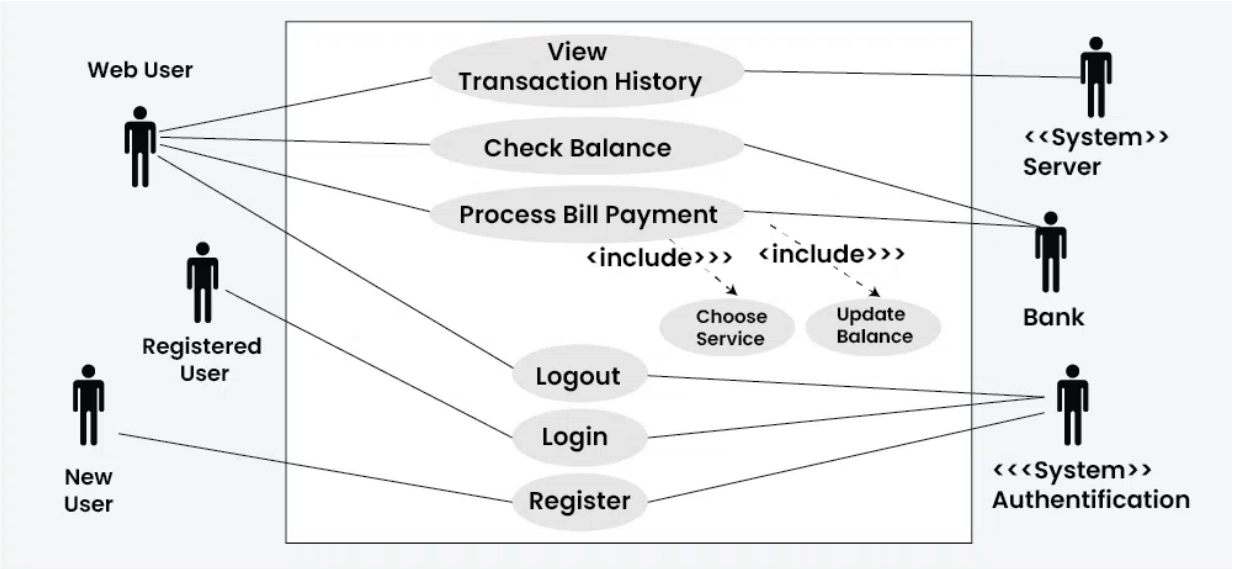
2.1. Traffic Estimation

> *Assumption - Traffic is 100,000 vistors per month*
> *Each authentication request is assumed to take 1 second for simplification.*
> *Traffic per second = 100000/30*24*60*60= 0.038*
> *Authentication Requests per Second = Traffic per Second*
> ***Authentication Requests per Second** = **0.038***

2.2. Storage Estimation

> *Assumption - Each authentication request is assumed to take approx 2kb/file size*
> *Monthly Storage = Monthly Visitors × Average Authentication request/User Data Size*
> *Monthly Storage=100,000×2 KB*
> ***Monthly Storage** = 200,000KB or **195.31 MB**(approx)*

3. Use Case Diagram for Authentication System Design



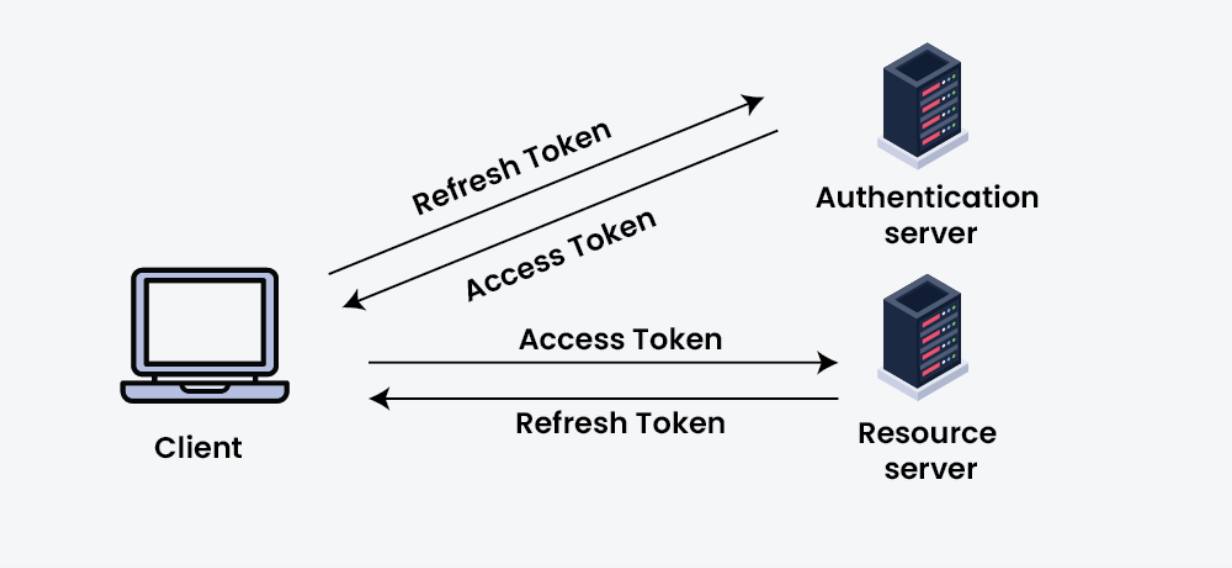
Use Case Diagram | Design Authentication System

The web user initiates the interaction by logging in or registering. After successful authentication, the user can perform actions like viewing transaction history, checking balance, or processing bill payments. The user may choose to log out when the interaction is complete.

- The system server oversees the entire interaction, coordinating the authentication process and managing user sessions. It facilitates the user's selection of services and handles the logout process.
- The bank actor interacts with the system to update the user's balance based on transactions or activities initiated by the web user.
- The registered user initiates the login process, providing valid credentials for authentication. Upon successful authentication, the user gains access to various services offered by the system.
- The new user initiates the registration process, providing necessary information to create a new account.After successful registration, the user can proceed to log in and access the system.
- The system authentication component manages the authentication process for both registered and new users. It verifies user credentials during login and facilitates the registration process for new users.

4. 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. Key components in the low-level design of an authentication system are described below:



Let's understand the main components of Low Level Design:

1. Authentication Server:

- **Handling Refresh Token**
  - Client sends a request for a refresh token.
  - Authentication server validates the client's identity.
  - If valid, a new refresh token is generated and sent to the client.
- **Handling Access Token**
  - Client sends authentication credentials for access token.
  - Authentication server verifies the credentials.
  - If valid, an access token is generated and sent to the client.

2. Client:

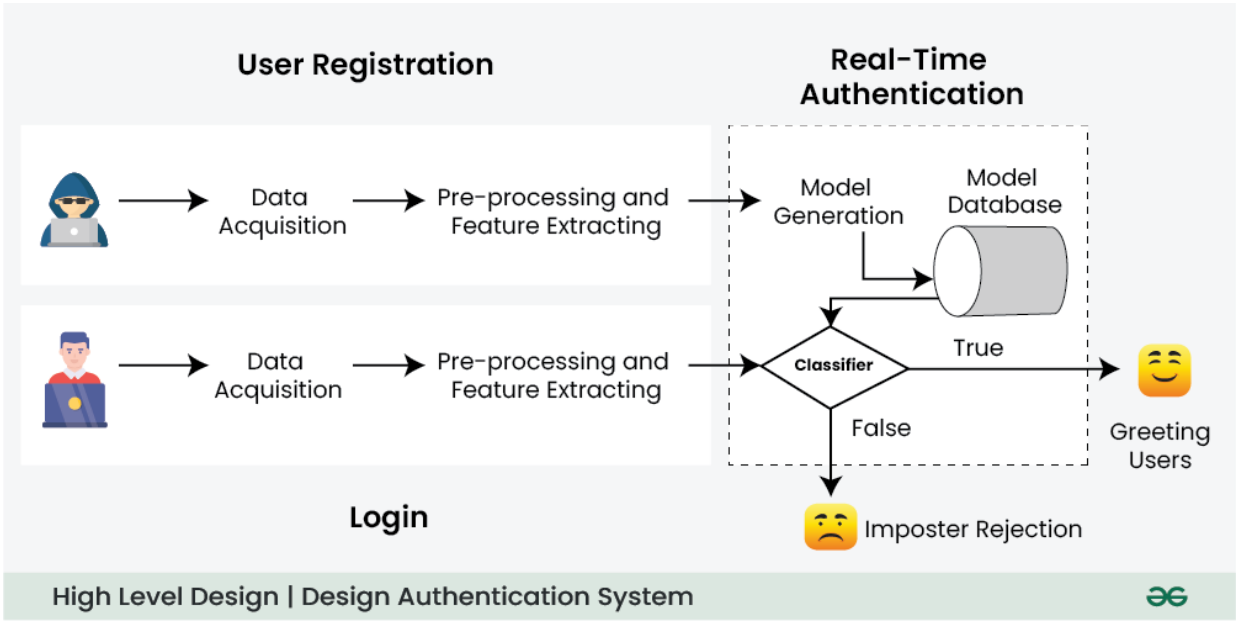
- **Connecting to Authentication Server**
  - Client establishes a connection to the authentication server.
  - It utilizes the refresh token or authentication credentials to interact.
- **Connecting to Resource Server**
  - Client establishes a connection to the resource server.
  - It uses the access token to request protected resources.
- **Using Refresh Token**
  - Client sends a request for a new access token using the refresh token.
  - Authentication server validates the refresh token.
  - If valid, a new access token is issued to the client.
- **Using Access Token**
  - Client includes the access token in requests to the resource server.
  - Resource server validates the access token.
  - If valid, it grants access to the requested protected resource.

3. Resource Server:

- Resource server validates the access token.
- If valid, it provides access to the requested resource.
- Client sends a request with an access token to access a protected resource.

5. 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. Key components in the high-level design of an authentication system are described as follow:



Let's understand High Level Design of the Authentication System :

1. User Registration Section:

- **Data Acquisition**
  - User provides registration information.
  - System acquires and validates user data.
- **Pre-processing and Feature Extracting**
  - Raw user data undergoes pre-processing for normalization and cleaning.
  - Features are extracted for use in the registration process.

2. Login Section:

- **Data Acquisition**
  - User provides login credentials.
  - System acquires and validates user login data.
- **Pre-processing and Feature Extracting**
  - Raw login data undergoes pre-processing for normalization.
  - Features are extracted for authentication.

3. Real-Time Authentication Section:

What is a model?
A "model" refers to a representation or set of parameters that characterize a specific user's behavior or characteristics. These models are generated based on the registered user data and are stored in a database for real-time use during the authentication process.

- **Model Generation**
  - Models are generated based on registered user data.
  - Model parameters are stored for real-time use.
- **Model Database**
  - The database stores generated user models.
- **Classifier (if True)**
  - Classifier determines if the user is legitimate.
  - If classified as true, user is authenticated.
  - System welcomes and grants access to the authenticated user.
- **Classifier (if False)**
  - If classified as false (imposter), the system initiates imposter rejection.
  - Appropriate actions are taken to prevent unauthorized access.

6. Database Design for Authentication System Design

Database design for authentication system:

6.1. User Table

The User Table stores user data with the following fields:

- **`user_id` (PK):** Unique identifier for each user.
- **`username`:** User's username for authentication.
- **`email`:** User's email address for communication.

- **`password`:** Encrypted password for user authentication.
- **`created_at`:** Timestamp indicating when the user account was created.

6.2. Credentials Table

The Credentials Table stores login credentials, including hashed passwords, with the following fields:

- **`credential_id` (PK):** Unique identifier for each credential.
- **`user_id` (FK)**: Foreign key referencing the User Table.
- **`password_hash`:** Hashed password for user authentication.
- **`last_login`:** Timestamp indicating the user's last login.

6.3. Password Table

Password table are used to store passwords set by user. It includes field like

- **password_id (PK):** Unique identifier for each password entry.
- **user_id (FK):** Foreign key referencing the User Table.
- **password_hash:** Hashed password for user authentication.

6.4. PasswordResetRequests Table

is used to store information related to password reset requests initiated by users. It include field like

- **request_id (PK):** Unique identifier for each password reset request.
- **user_id (FK):** Foreign key referencing the User Table.
- **token_value:** Value of the token for the password reset.
- **expiration_time:** Timestamp indicating when the password reset token expires.

6.5. Session Table

The Session Table tracks user sessions with the following fields:

- **`session_id` (PK):** Unique identifier for each session.
- **`user_id` (FK):** Foreign key referencing the User Table.
- **`login_time`:** Timestamp indicating the session login time.
- **`last_activity`:** Timestamp indicating the session's last activity.

6.6. Token Table

The Token Table stores information about user tokens with the following fields:

- **`token_id` (PK):** Unique identifier for each token.
- **`user_id` (FK)**: Foreign key referencing the User Table.
- **`token_value`:** Value of the token for authentication.
- **`expiration_time`:** Timestamp indicating when the token expires.

7. Microservices used for Authentication System Design

7.1. 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.

7.2. 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.
- `/mfa/:userId`: Handle multi-factor authentication.

7.3. 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 Endpoints:

- `/grant/:userId/:permission`: Grant specific permissions to a user.
- `/revoke/:userId/:permission`: Revoke permissions from a user.
- `/check/:userId/:resource`: Check user's access to a specific resource.

7.4. Session Management Microservice:

Handles the creation, maintenance, and termination of user sessions. This microservice ensures secure session handling and can implement features like session timeouts and token revocation.

API Endpoints:

- `/create/:userId`: Create a new user session.

- `/expire/:sessionId`: Expire a user session.
- `/validate/:sessionId`: Validate an active user session.

8. API Used for Authentication System Design

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:

8.1. RESTful APIs:

RESTful APIs are commonly used for communication between microservices due to their simplicity and statelessness. Each microservice exposes a set of RESTful endpoints, allowing other services to make HTTP requests to perform specific actions.

8.2. 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.

8.3. OpenID Connect and OAuth 2.0:

OpenID Connect and OAuth 2.0 are widely adopted authentication and authorization protocols. They define a set of rules for secure and standardized user authentication, allowing third-party applications to access user data without exposing sensitive credentials.

8.4. GraphQL:

GraphQL is an alternative to RESTful APIs that allows clients to request only the specific data they need. In the context of an authentication system, GraphQL can be used to efficiently query user information and manage authentication-related operations.

9. API Code Implementation for Authentication System

9.1. User Registration API (POST):

Endpoint: `/api/user/register`

Description: Allows users to securely create their accounts.

Request

```
{
  &amp;quot;username&amp;quot;: &amp;quot;example_user&amp;quot;,
  &amp;quot;email&amp;quot;: &amp;quot;user@example.com&amp;quot;,
  &amp;quot;password&amp;quot;: &amp;quot;securepassword123&amp;quot;
}
```

Response

9.2. Authentication API (POST):

Endpoint: `/api/user/authenticate`

Description: Initiates user authentication.

Request

```
{
  &amp;quot;username&amp;quot;: &amp;quot;example_user&amp;quot;,
  &amp;quot;password&amp;quot;: &amp;quot;securepassword123&amp;quot;
}
```

Response

9.3. Access Protected Resource API (GET):

Endpoint: `/api/resource/access`

Description: Allows access to a protected resource.

Request

```
GET /api/resource/access
Host: your-authentication-api.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Response

9.4. Update Password API (PUT):

Endpoint: `/api/user/update-password`

Description: Updates the user's password.

Request

```
{
  &amp;quot;user_id&amp;quot;: &amp;quot;12345&amp;quot;,
  &amp;quot;current_password&amp;quot;: &amp;quot;oldpassword&amp;quot;,
  &amp;quot;new_password&amp;quot;:
&amp;quot;newsecurepassword456&amp;quot;
}
```

Response

10. Scalability for Authentication System Design

Consideration for scalability is crucial to ensure the system can handle increased load. Key strategies for scalability in an authentication system include:

10.1. Load Balancing

Implement load balancing mechanisms to distribute incoming authentication requests evenly across multiple servers. This ensures optimal resource utilization and prevents any single point of failure.

10.2. Horizontal Scaling

Design the system to scale horizontally, allowing the addition of more servers or instances to accommodate growing user traffic.

10.3. Caching

Utilize caching mechanisms for frequently accessed data, such as user credentials or session information, to reduce the load on the database and improve response times.

10.4. Elasticity

Implement auto-scaling features to dynamically adjust resources based on demand. This ensures efficient resource utilization during peak periods and cost-effectiveness during low traffic times.

11. Conclusion

Designing an authentication system is a critical aspect of any secure application. Balancing usability and security, understanding the importance of various components, and staying vigilant against emerging threats are key to building a resilient authentication system.