

## 11. Design Authentication System

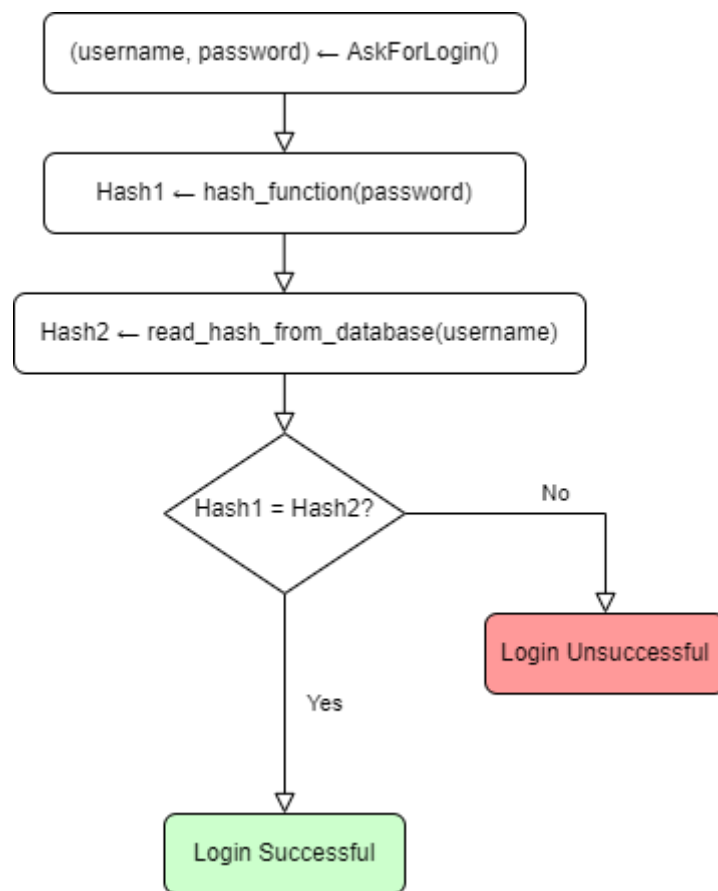
Designing a user authentication system is a key component of securing any application, ensuring that only authorized users can access and use the platform. Here's how you can design a solid authentication system with the right database structure to support it.

### Requirements of Any User Authentication Module

To provide effective protection against unauthorized access, the authentication module must support the following capabilities:

- **Register New Users:** Enable user registration with basic details.
- **Send Confirmation Emails:** Verify user identity with a confirmation email.
- **Password Recovery:** Allow users to recover a forgotten password securely.
- **Store Authentication Data Securely:** Protect sensitive authentication data.
- **Support Third-Party Authentication:** Allow users to log in via external providers like Google or Facebook.
- **Define Roles and Permissions:** Ensure users have appropriate access based on roles.

## Building a Database Model for User Authentication



### 1. Storing the Password Hash

For security, passwords should never be stored in plaintext. Instead, store the hashed version of the password using a secure hashing algorithm (e.g., SHA-256 or bcrypt). You should never store the actual password but only a hash of it.

The database model should include:

- **Username:** Unique user identifier.
- **Password Hash:** The hashed version of the user's password.
- **Salt:** A random string added to the password before hashing to prevent rainbow table attacks.

#### Table Example:

- ``user_authentication_data``

- `user\_id (PK)`
- `username (UNIQUE)`
- `password\_hash`
- `salt`
- `hash\_algorithm\_id (FK)`

## 2. Adding Salt

To further secure passwords, salt is added to each password before hashing. This ensures that even if two users have the same password, their hashes will be different.

## 3. Storing Email and Confirmation Token

When users register, they will need to confirm their email address to ensure it's valid. This can be done by sending a unique confirmation token to the user's email. The database will store this token along with a timestamp to track its expiration.

### Table Example:

- `user\_email\_verification`
- `user\_id (FK)`
- `email`
- `confirmation\_token`
- `token\_generated\_at`
- `token\_expiry\_at`
- `email\_verified` (boolean)

## 4. Password Recovery

If a user forgets their password, they should be able to recover it through a password reset process. This typically involves sending a password reset token to the user's email.

The database model for password recovery will include:

- Recovery Token: A one-time token to validate the password reset request.
- Token Expiry: Time to limit how long the reset token is valid.

### Table Example:

- `user\_password\_recovery`
- `user\_id (FK)`
- `recovery\_token`
- `token\_generated\_at`
- `token\_expiry\_at`

## 5. External Authentication via OAuth

Third-party authentication is increasingly popular for streamlining login processes. You can use OAuth to allow users to log in using their Google, Facebook, or other external accounts. For this, store a reference to the external provider and its authentication token.


### Table Example:

- `user\_external\_authentication`
- `user\_id (FK)`
- `external\_provider\_id`
- `external\_provider\_token`

- `external\_provider\_user\_id`

**Log in**

Don't have an account? [Sign up](#)

 Log in with Google

OR

Email

Password

**Log In**

[Forgot Password?](#)

## 6. User Identity vs. User Account

It's important to separate user identity from the user account. This allows flexibility for users to change their email, username, or other credentials without affecting their identity or account.

### Tables:

- `user\_identity`
- `user\_id (PK)`
- `first\_name`
- `last\_name`
- `email`
- `phone\_number`

- `user\_account`
- `user\_id (PK)`
- `username`
- `password\_hash`

## 7. Multi-Factor Authentication (MFA)

Multi-factor authentication adds an extra layer of security by requiring a second form of authentication, such as a one-time password (OTP) or a mobile app.

### Table Example:

- `user\_mfa`
- `user\_id (FK)`
- `mfa\_enabled (boolean)`
- `mfa\_type` (e.g., SMS, TOTP)
- `mfa\_secret\_key`

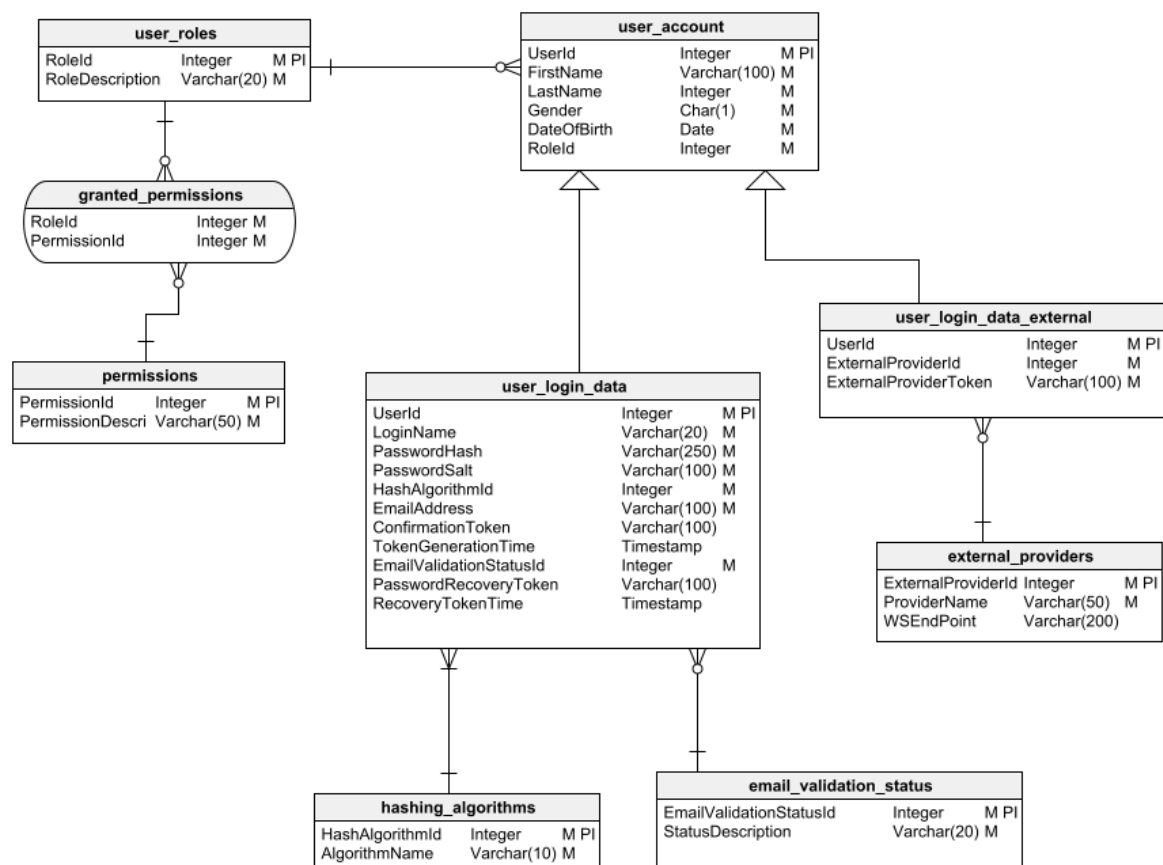
## 8. Roles and Permissions

Assigning roles and permissions allows you to define different levels of access within your system. For example, admin users might have full access, while regular users have limited access.

### Tables:

- `roles`
- `role\_id (PK)`
- `role\_name`
- `role\_description`
- `permissions`
- `permission\_id (PK)`

- `permission\_name`
- `permission\_description`
- `user\_roles`
- `user\_id (FK)`
- `role\_id (FK)`
- `role\_permissions`
- `role\_id (FK)`
- `permission\_id (FK)`



Database Schema

## Putting It All Together: Full Database Schema

```
CREATE TABLE user_authentication_data (  
  user_id INT PRIMARY KEY,  
  username VARCHAR(255) UNIQUE,  
  password_hash VARCHAR(255),  
  salt VARCHAR(255),  
  hash_algorithm_id INT  
);  
  
CREATE TABLE user_email_verification (  
  user_id INT FOREIGN KEY REFERENCES user_authentication_data(user_id),  
  email VARCHAR(255),  
  confirmation_token VARCHAR(255),  
  token_generated_at TIMESTAMP,  
  token_expiry_at TIMESTAMP,  
  email_verified BOOLEAN  
);  
  
CREATE TABLE user_password_recovery (  
  user_id INT FOREIGN KEY REFERENCES user_authentication_data(user_id),  
  recovery_token VARCHAR(255),  
  token_generated_at TIMESTAMP,  
  token_expiry_at TIMESTAMP  
);  
  
CREATE TABLE user_external_authentication (  
  user_id INT FOREIGN KEY REFERENCES user_authentication_data(user_id),  
  external_provider_id INT,  
  external_provider_token VARCHAR(255),  
  external_provider_user_id VARCHAR(255)  
);  
  
CREATE TABLE user_identity (
```



```

user_id INT PRIMARY KEY,
first_name VARCHAR(255),
last_name VARCHAR(255),
email VARCHAR(255),
phone_number VARCHAR(255)
);

CREATE TABLE user_account (
    user_id INT PRIMARY KEY,
    username VARCHAR(255),
    password_hash VARCHAR(255)
);

CREATE TABLE user_mfa (
    user_id INT FOREIGN KEY REFERENCES user_authentication_data(user_id),
    mfa_enabled BOOLEAN,
    mfa_type VARCHAR(255),
    mfa_secret_key VARCHAR(255)
);

CREATE TABLE roles (
    role_id INT PRIMARY KEY,
    role_name VARCHAR(255),
    role_description TEXT
);

CREATE TABLE permissions (
    permission_id INT PRIMARY KEY,
    permission_name VARCHAR(255),
    permission_description TEXT
);

CREATE TABLE user_roles (
    user_id INT FOREIGN KEY REFERENCES user_authentication_data(user_id),

```

```
    role_id INT FOREIGN KEY REFERENCES roles(role_id)
);

CREATE TABLE role_permissions (
    role_id INT FOREIGN KEY REFERENCES roles(role_id),
    permission_id INT FOREIGN KEY REFERENCES permissions(permission_id)
);
```

## Conclusion

This robust user authentication system model is designed with scalability, security, and flexibility in mind. It incorporates essential features like password hashing, email confirmation, password recovery, external authentication, MFA, and role-based access control. You can expand and modify it as per your specific application's needs, ensuring that it remains secure and user-friendly.