1. Using Django’s built-in form handling.

Ans. Django provides a powerful form handling system that allows developers to create, validate, and process forms easily. The forms module provides a way to define forms as Python classes and automatically generate HTML, handle validation, and process user input securely.

1. Implementing Django’s authentication system (sign up, login, logout, password management).

Ans. **1. Components of Django’s Authentication System**

**A. User Model (django.contrib.auth.models.User)**

Django includes a default **User model** with essential fields such as:

* username – Unique identifier for the user.
* email – User’s email address.
* password – Encrypted password stored securely.
* is\_active – Indicates whether the user account is active.
* is\_staff – Determines if the user has access to the admin panel.

Django also allows **custom user models** for flexibility in user management.

**2. User Authentication Process**

Django’s authentication framework follows a **request-response cycle** for user verification and session handling:

1. **User Registration (Sign Up)**
   * Django provides the UserCreationForm to create new users.
   * Passwords are automatically encrypted before storage.
   * Upon successful registration, users can be logged in immediately or redirected to a login page.
2. **User Login**
   * Users provide their credentials (username/email and password).
   * Django checks the credentials against the database.
   * If valid, a **session is created**, and the user is logged in.
3. **User Logout**
   * When a user logs out, Django deletes the session data, effectively logging the user out.
   * Users must log in again to access protected pages.
4. **Password Management**
   * Users can **change** their password while logged in.
   * Users who **forget their password** can request a reset link via email.
   * Django uses **password hashing** for security, ensuring that stored passwords are not readable in plain text.

**3. Django’s Built-in Authentication Views**

Django provides prebuilt views that simplify authentication handling:

1. LoginView – Handles user authentication.
2. LogoutView – Logs out the user and clears the session.
3. PasswordChangeView – Allows logged-in users to change their password.
4. PasswordResetView – Sends a password reset link to the user’s email.

These views can be customized to fit the application's UI and functionality.

**4. Securing User Authentication**

Django implements several security measures for authentication:

**A. Password Hashing**

* Django automatically hashes user passwords before storing them in the database.
* It uses strong algorithms like **PBKDF2, Argon2, and bcrypt** to protect passwords.

**B. CSRF Protection**

* Django includes **Cross-Site Request Forgery (CSRF) protection** to prevent malicious form submissions.
* All authentication forms should include {% csrf\_token %} for added security.

**C. Session Management**

* Django maintains **user sessions** to keep users logged in after authentication.
* Sessions can be stored in the database, cache, or signed cookies.

**D. Access Control**

* The @login\_required decorator ensures only authenticated users can access certain views.
* Django’s **permissions system** allows role-based access control for different user types.

**5. User Authentication Flow in Django**

1. **User visits login page → enters credentials → Django verifies**.
2. **If credentials are correct → user session is created → user is redirected to a dashboard/home page**.
3. **If incorrect → an error message is displayed**.
4. **When user logs out → session is cleared → user is redirected to login page**.
5. **If a user forgets their password → Django sends a password reset link to their email**.