Django supports multiple types of authentication mechanisms to help secure applications. Here are the main types:

1. **Session Authentication**:
   * This is the default authentication method in Django, where user credentials are verified and a session ID is stored on the server. The session is stored in a cookie, which is checked on each request to confirm if the user is authenticated.
2. **Token Authentication**:
   * Token-based authentication is used for APIs, particularly when using Django REST Framework (DRF). When a user logs in, they receive a token, which they include in the headers of future requests to authenticate themselves without needing to log in again.
3. **JSON Web Token (JWT) Authentication**:
   * JWT is another token-based authentication method, often used in modern web applications. JWTs are stateless and typically used for API authentication. The token contains user information and is signed to verify authenticity. JWT authentication is commonly implemented with packages like djangorestframework-simplejwt.
4. **OAuth2 Authentication**:
   * OAuth2 allows users to log in using third-party providers (e.g., Google, Facebook). Django packages like django-allauth or django-oauth-toolkit provide built-in support for OAuth2, making it easier to integrate social logins.
5. **Basic Authentication**:
   * Basic Authentication is a simple, base64-encoded username and password scheme, often used in development or with DRF for testing APIs. It’s not secure over HTTP, so it’s typically limited to HTTPS or local testing.
6. **Custom Authentication**:
   * Django allows developers to implement custom authentication backends if specific needs go beyond what’s provided. You can define custom logic by creating a new authentication backend class and configuring it in Django’s settings.
7. **API Key Authentication**:
   * Often used for service-to-service communication, API Key Authentication involves assigning unique keys to each client. The client includes the API key in requests, and Django verifies this key to authenticate the request.

Each of these authentication types can be configured in Django based on the needs of your application, especially if you're working with Django REST Framework for APIs. Let me know if you’d like examples or guidance on implementing any of these.

***OAuth***

OAuth (Open Authorization) is an open standard for access delegation, commonly used to allow users to grant applications access to their information on other websites without sharing their passwords. It enables secure, token-based access to resources and is widely adopted by platforms like Google, Facebook, GitHub, and many others. Here’s a deep dive into how OAuth works, its components, and its flow:

### ***1. How OAuth Works***

OAuth allows a user (resource owner) to grant limited access to their resources on a server (resource server) to a third-party application (client). This access is granted without the client ever knowing the user’s credentials. Instead, the authorization is based on tokens, which serve as secure identifiers for the authenticated session.

OAuth uses access tokens (and sometimes refresh tokens) to manage permissions and validate user identities without the need for repeated logins or direct credential exchanges.

### ***2. Key Components in OAuth***

To understand how OAuth works, it helps to know its key components:

* **Resource Owner (User)**: The user who owns the data and grants access to the client application.
* **Client (Application)**: The third-party application that requests permission to access the user’s data.
* **Authorization Server**: The server that authenticates the user and issues access tokens after the user consents to grant access.
* **Resource Server**: The server hosting the user’s resources (API or service) that the client wants to access. The resource server verifies the token to grant access.

***3. OAuth Flow (Authorization Code Grant)***

The Authorization Code Grant is the most commonly used OAuth flow for web applications. Here’s how it works, step-by-step:

1. **User Interaction with the 3rd-Party App**:
   * The user opens a social media app that has a feature to link with Google Photos. This app asks the user if they want to grant access to their Google Photos.
2. **Request to Google Authorization Server**:
   * When the user agrees, the social media app redirects the user to Google's authorization server. This is the start of the OAuth process.
3. **Google Authorization Page Displayed to User**:
   * The user sees a Google Authorization Page that specifies the permissions the social media app is requesting. Here, the app asks to access the user’s photos.
4. **User’s Consent**:
   * The user is given two options: “Yes” (Approve) or “No” (Deny). If the user approves, the flow continues.
5. **Authorization Code Issued**:
   * After approval, Google’s authorization server provides an **authorization code** to the social media app. This code is a one-time use code that allows the app to request an access token.
6. **Social Media App Sends Authorization Code**:
   * The social media app sends the authorization code back to Google’s authorization server to exchange it for an access token.
7. **Access Token Issued**:
   * Google’s authorization server verifies the authorization code and issues an **access token** to the social media app. This token allows the app to access the user’s Google Photos.
8. **Access to Resource Server (Photos)**:
   * The social media app uses the access token to send a request to Google’s resource server (where the user’s photos are stored).
9. **Resource Server Verifies Token**:
   * Google’s resource server verifies the access token to ensure it is valid and has the correct permissions.
10. **Resource Server Responds with Requested Photos**:
    * Once verified, the resource server provides access to the user’s photos, allowing the social media app to display or use them as requested.
11. **Introspection (Token Verification by Resource Server)**:
    * During the process, if the resource server needs to further verify the token details, it can do an **introspection** by consulting the authorization server.
12. **Access Granted to App**:
    * Finally, the social media app now has access to the photos and can use them as allowed by the permissions granted.

### **Summary**

This flow allows a third-party app to access a user’s Google Photos securely without needing the user’s password. Instead, it uses an authorization code and access token managed by Google’s authorization server.

### ***4. Types of OAuth Grants***

OAuth offers several grant types to suit different use cases:

* **Authorization Code Grant**: Best suited for web applications; involves redirecting the user and exchanging an authorization code for an access token.
* **Implicit Grant**: Suitable for single-page applications (SPAs) where a token is issued immediately without an intermediate code exchange.
* **Password Grant**: Deprecated due to security concerns. The client directly receives the user’s credentials and exchanges them for a token.
* **Client Credentials Grant**: Ideal for server-to-server communication without user interaction, where a client directly requests an access token with its credentials.
* **Refresh Token Grant**: Refresh tokens allow an application to obtain a new access token after the old one expires, without requiring the user to log in again.

### **5. Access Tokens and Refresh Tokens**

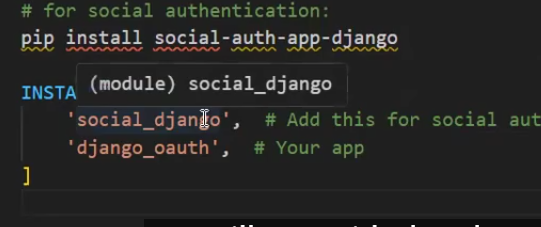
* **Access Tokens**: These tokens grant temporary access to specific resources on the resource server. They have a short lifespan and are included in requests to the resource server.
* **Refresh Tokens**: Refresh tokens are used to obtain new access tokens once they expire, avoiding the need to re-authenticate the user. They have a longer lifespan and are generally stored securely by the client.

### ***6. OAuth Scopes***

Scopes specify what resources the client is requesting. For example, when logging in with Google, scopes like profile, email, or calendar determine what data the third-party application can access. Users can review these scopes before granting permission.

***7. Benefits of OAuth***

* **Enhanced Security**: Users don’t need to share their credentials with third-party applications.
* **Granular Permissions**: OAuth provides control over which resources are accessible and for how long.
* **Improved User Experience**: Users can log in with one set of credentials across multiple applications.
* **Token-Based Access**: Tokens allow stateless, decentralized access, which is ideal for modern distributed systems.



**Project**

**Step 1:**

C:\Users\Admin\Desktop\DjangoPractice\Oauth and Social>pip install django-allauth

C:\Users\Admin\Desktop\DjangoPractice\Oauth and Social>django-admin startproject oauth\_project

C:\Users\Admin\Desktop\DjangoPractice\Oauth and Social>cd oauth\_project

C:\Users\Admin\Desktop\DjangoPractice\Oauth and Social>django-admin startapp django\_oauthapp

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'django\_oauthapp',

'social\_django',

]

@55-'DIRS': [os.path.join(BASE\_DIR, "templates")],

@line 68 & 69: 'social\_django.context\_processors.backends',

'social\_django.context\_processors.login\_redirect',

@line 74: AUTHENTICATION\_BACKENDS = AUTHENTICATION\_BACKENDS = (

'social\_core.backends.google.GoogleOAuth2', # Example for Google OAuth2

'django.contrib.auth.backends.ModelBackend', # Default backend

'social\_core.backends.google.GoogleOAuth2',

'social\_core.backends.github.GithubOAuth2',

)

@line 136 LOGIN\_URL = 'login'

LOGOUT\_URL = 'logout'

LOGIN\_REDIRECT\_URL = 'home'

LOGOUT\_REDIRECT\_URL = 'home'

SITE\_ID = 1

**Step 2: Follow these steps for generating secret key n password and add them in the setting.py at the end(line 145)**

**SOCIAL\_AUTH\_GOOGLE\_OAUTH2\_KEY = '919450085876-9aag8uv4nn4ud9dfn72877d2h2bbrlcc.apps.googleusercontent.com'**

**SOCIAL\_AUTH\_GOOGLE\_OAUTH2\_SECRET = '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'**

**'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'**

**# SOCIAL\_AUTH\_GITHUB\_OAUTH2\_KEY =**

**# SOCIAL\_AUTH\_GITHUB\_OAUTH2\_SECRET =**

1. Go to Google Developer Console:

https://console.developers.google.com/project

2. Create Project

3. Provide Project name and click on Create

4. Click on Select Project in Notification popup

5. Click on API and Services

6. Click on Enable API and Services

7. Search for “Gmail API” in the search box

8. Click on Gmail API

9. Click on Enable

10. You will see the option to CREATE CREDENTIALS

11. Click on User data and NEXT

12. Provide App Name = which you have created in the (django app).

13. Provide email id

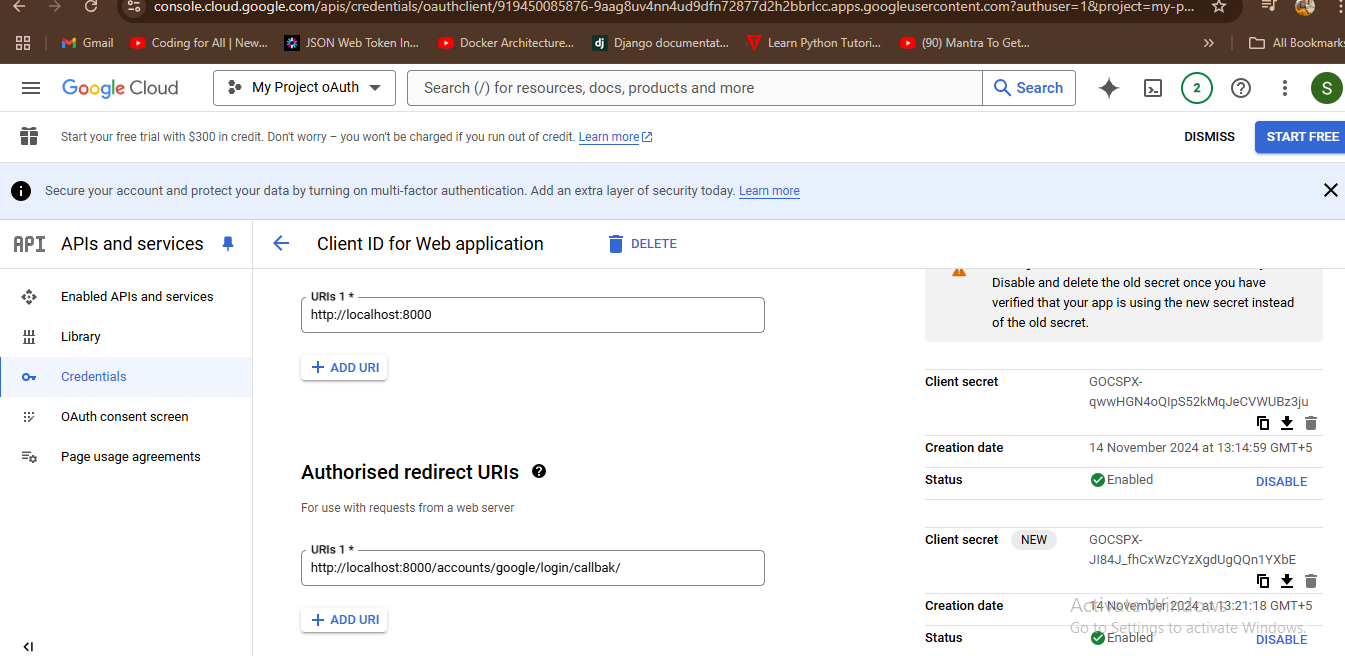
14. Provide email id for (OAuth Consent Screen)

15. Select Web Application for Application Type

16. Authorized JavaScript origins: http://localhost:8000

17. Authrozed URL Redirect: http://localhost:8000/accounts/google/login/callbak/

18. Submit



**Step 3: Define views.py**

**from django.shortcuts import render, redirect**

**from django.contrib.auth import authenticate, login as auth\_login, logout as auth\_logout**

**from django.contrib.auth.models import User**

**from django.contrib.auth.decorators import login\_required**

**from django.contrib import messages**

**# Signup view**

**def signup(request):**

**if request.method == "POST":**

**username = request.POST.get('username')**

**email = request.POST.get('email')**

**password = request.POST.get('password')**

**try:**

**user = User.objects.create\_user(username=username, email=email, password=password)**

**messages.success(request, 'Account created successfully!')**

**return redirect('login')**

**except:**

**messages.error(request, 'Username already exists. Please choose a different one.')**

**return render(request, 'signup.html')**

**return render(request, 'signup.html')**

**# Login view**

**def login(request):**

**if request.method == "POST":**

**username = request.POST.get('username')**

**password = request.POST.get('password')**

**user = authenticate(request, username=username, password=password)**

**if user is not None:**

**auth\_login(request, user)**

**return redirect('home')**

**else:**

**messages.error(request, 'Invalid username or password')**

**return render(request, 'login.html')**

**# Logout view**

**def logout(request):**

**auth\_logout(request)**

**return redirect('home')**

**# Home view**

**@login\_required**

**def home(request):**

**return render(request, 'home.html')**

**# Profile view**

**@login\_required**

**def profile(request):**

**return render(request, 'profile.html', {"user": request.user})**

**Step 4: Define app-urls**

**from . import views**

**urlpatterns = [**

**path('signup/', views.signup, name='signup'), # URL for signup**

**path('login/', views.login, name='login'), # URL for login**

**path('logout/', views.logout, name='logout'), # URL for logout**

**path('home/', views.home, name='home'), # URL for home**

**path('profile/', views.profile, name='profile'),**

**path('oauth/', include('social\_django.urls',namespace='social')),**

**]**

**Step 5 : Project urls**

**from . import views**

**urlpatterns = [**

**path('signup/', views.signup, name='signup'), # URL for signup**

**path('login/', views.login, name='login'), # URL for login**

**path('logout/', views.logout, name='logout'), # URL for logout**

**path('home/', views.home, name='home'), # URL for home**

**path('profile/', views.profile, name='profile'),**

**path('oauth/', include('social\_django.urls',namespace='social')),**

**]**

**Step 6: Define html in templates folder**

**home.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Home</title>**

**</head>**

**<body>**

**<h1>Welcome to Your Home Page</h1>**

**<p>Welcome, {{ user.username }}!</p>**

**<a href="{% url 'profile' %}">Go to Profile</a><br>**

**<a href="{% url 'logout' %}">Logout</a>**

**</body>**

**</html>**

**Signup.html**

**<!-- templates/signup.html -->**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Sign Up</title>**

**<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">**

**</head>**

**<body>**

**<div class="container">**

**<h1 class="mt-5">Sign Up</h1>**

**<form method="post" class="mt-4">**

**{% csrf\_token %}**

**<div class="form-group">**

**<input type="text" class="form-control" name="username" placeholder="Username" required>**

**</div>**

**<div class="form-group">**

**<input type="email" class="form-control" name="email" placeholder="Email" required>**

**</div>**

**<div class="form-group">**

**<input type="password" class="form-control" name="password" placeholder="Password" required>**

**</div>**

**<button type="submit" class="btn btn-primary">Sign Up</button>**

**</form>**

**<p class="mt-3">Already have an account? <a href="{% url 'login' %}">Log in</a></p>**

**</div>**

**</body>**

**</html>**

**Login.html**

**<!-- templates/login.html -->**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Login</title>**

**<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">**

**</head>**

**<body>**

**<div class="container">**

**<h1 class="mt-5">Login</h1>**

**<form method="post" class="mt-4">**

**{% csrf\_token %}**

**<div class="form-group">**

**<input type="text" class="form-control" name="username" placeholder="Username" required>**

**</div>**

**<div class="form-group">**

**<input type="password" class="form-control" name="password" placeholder="Password" required>**

**</div>**

**<a href="{% url 'home' %}"><button type="submit" class="btn btn-primary">Login</button></a>**

**</form>**

**<p class="mt-3">Don't have an account? <a href="{% url 'signup' %}">Sign up</a></p>**

**<h2 class="mt-4">Or log in with:</h2>**

**<a href="{% url 'social:begin' 'google-oauth2' %}" class="btn btn-danger">Google</a>**

**<a href="{% url 'social:begin' 'github' %}" class="btn btn-dark">GitHub</a>**

**</div>**

**</body>**

**</html>**

**Logout.html**

**<!-- logout.html -->**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Logout</title>**

**</head>**

**<body>**

**<h1>You have been logged out successfully!</h1>**

**<p>Thank you for visiting. See you next time!</p>**

**<a href="{% url 'login' %}">Login Again</a> |**

**<a href="{% url 'signup' %}">Sign Up</a>**

**</body>**

**</html>**

**Profile.html**

**<!-- profile.html -->**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>User Profile</title>**

**</head>**

**<body>**

**<h1>Profile of {{ user.username }}</h1>**

**<p>Email: {{ user.email }}</p>**

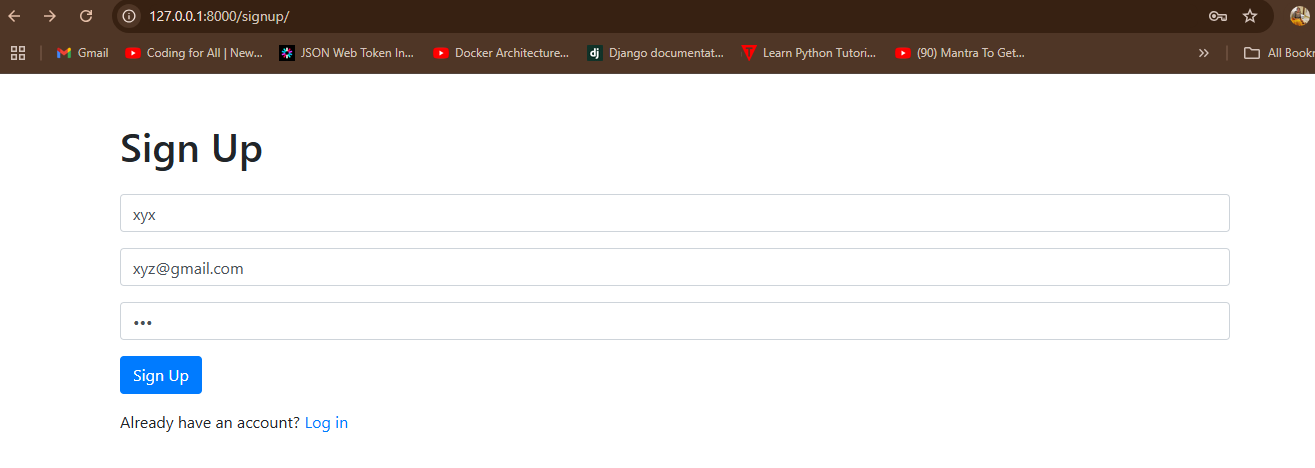
**<a href="{% url 'home' %}">Back to Home</a><br>**

**<a href="{% url 'logout' %}">Logout</a>**

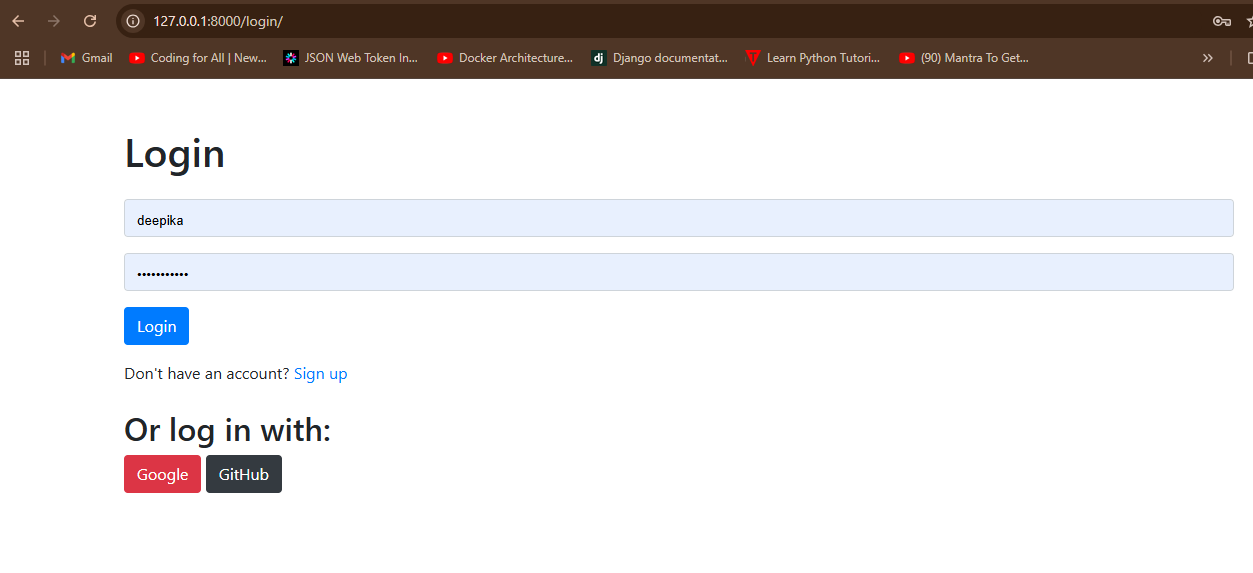
**</body>**

**</html>**

**Output:**

****

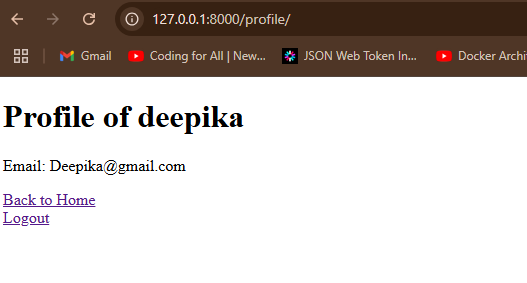
**Login//**

****

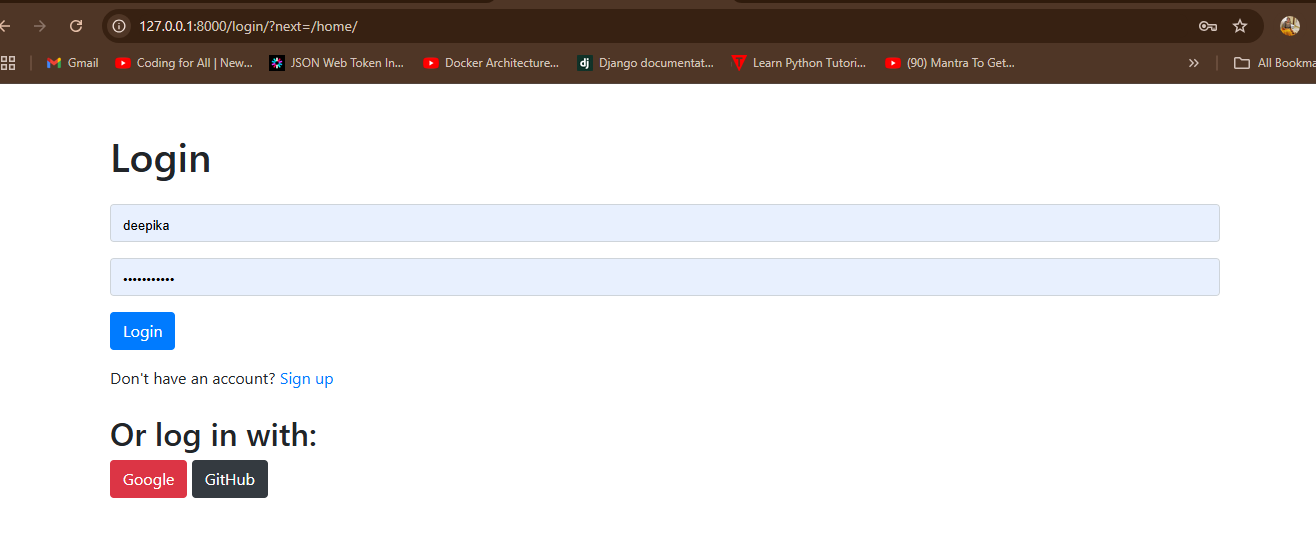
**Home url//**

****

**profile//**

****

**logout// redirecting to home**

****