D'accord, voici une explication détaillée pour créer un backend personnalisé dans Django pour l'authentification en utilisant l'email comme nom d'utilisateur.

### 1. Créer le backend personnalisé

Dans le répertoire de ton application Django, crée un fichier `backends.py` et ajoute le code suivant :

#### `backends.py`

```python

from django.contrib.auth.backends import BaseBackend

from django.contrib.auth.models import User

class EmailAuthBackend(BaseBackend):

def authenticate(self, request, username=None, password=None, \*\*kwargs):

try:

user = User.objects.get(email=username)

if user.check\_password(password):

return user

except User.DoesNotExist:

return None

def get\_user(self, user\_id):

try:

return User.objects.get(pk=user\_id)

except User.DoesNotExist:

return None

```

#### Explications ligne par ligne :

1. \*\*Imports nécessaires :\*\*

```python

from django.contrib.auth.backends import BaseBackend

from django.contrib.auth.models import User

```

- `BaseBackend` : Classe de base pour créer des backends d'authentification personnalisés.

- `User` : Modèle utilisateur par défaut de Django.

2. \*\*Déclaration de la classe `EmailAuthBackend` :\*\*

```python

class EmailAuthBackend(BaseBackend):

```

- Hérite de `BaseBackend`.

3. \*\*Méthode `authenticate` :\*\*

```python

def authenticate(self, request, username=None, password=None, \*\*kwargs):

```

- Cette méthode prend en paramètres la requête, le nom d'utilisateur (qui sera l'email dans ce cas) et le mot de passe.

4. \*\*Recherche de l'utilisateur par email :\*\*

```python

try:

user = User.objects.get(email=username)

```

- Recherche l'utilisateur dans la base de données en utilisant l'email (stocké dans `username`).

5. \*\*Vérification du mot de passe :\*\*

```python

if user.check\_password(password):

return user

```

- Vérifie si le mot de passe fourni correspond au mot de passe de l'utilisateur.

- Si oui, retourne l'utilisateur ; sinon, retourne `None`.

6. \*\*Gestion de l'utilisateur non trouvé :\*\*

```python

except User.DoesNotExist:

return None

```

7. \*\*Méthode `get\_user` :\*\*

```python

def get\_user(self, user\_id):

try:

return User.objects.get(pk=user\_id)

except User.DoesNotExist:

return None

```

- Cette méthode récupère l'utilisateur par son ID (utilisé par Django pour maintenir l'utilisateur connecté).

### 2. Configurer Django pour utiliser le backend personnalisé

Dans le fichier `settings.py`, ajoute le chemin de ton backend personnalisé à la liste `AUTHENTICATION\_BACKENDS`.

#### `settings.py`

```python

AUTHENTICATION\_BACKENDS = [

'path.to.your.app.backends.EmailAuthBackend', # Remplace par le chemin réel

'django.contrib.auth.backends.ModelBackend', # Garder le backend par défaut pour les fonctionnalités Django

]

```

- \*\*`path.to.your.app.backends.EmailAuthBackend`\*\* : Remplace `path.to.your.app` par le chemin de ton application. Par exemple, si ton application s'appelle `accounts`, cela deviendra `'accounts.backends.EmailAuthBackend'`.

### 3. Modifier la vue de connexion

#### `views.py`

```python

from django.shortcuts import render, redirect

from django.contrib import messages

from django.contrib.auth import authenticate, login

def login\_view(request):

if request.method == 'POST':

email = request.POST['email']

password = request.POST['password']

user = authenticate(request, username=email, password=password) # Utilise 'username' pour l'email

if user is not None:

login(request, user)

messages.success(request, 'You are now logged in')

return redirect('index')

else:

messages.error(request, 'Invalid credentials! Please try again.')

return redirect('login')

else:

return render(request, 'login.html')

```

- \*\*`authenticate(request, username=email, password=password)`\*\* : Note que `username=email` passe l'email en tant que nom d'utilisateur.

### 4. Mettre à jour les URL

#### `urls.py`

```python

from django.urls import path

from .views import login\_view

urlpatterns = [

path('login/', login\_view, name='login'),

# autres URLs

]

```

### Conclusion

Maintenant, ton projet Django est configuré pour authentifier les utilisateurs en utilisant leur email au lieu du nom d'utilisateur. Le backend personnalisé `EmailAuthBackend` gère l'authentification et est configuré dans `settings.py`. Tu as également mis à jour ta vue de connexion pour utiliser l'email comme identifiant.