LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

Module Code & Title

CS6P05 Final Year Project MAD

Food Share - Android App

Artifact – Login Authentication

Student Details

Name: Sita Ram Thing

London Met Id: 22015892

College Id: NP01MA4S220003

Islington College, Kathmandu

24 April 2024

Contents

List of Figures

1. Authentication

What is Login Authentication?

By Vishal Sharma

Login authentication is the most common scenario where we're asked to authenticate ourselves. Let's look at some aspects and challenges of implementing a seamless authentication mechanism and learn how businesses can deliver a seamless user experience through a CIAM.

Figure 1: About the authentications.

What is the Difference Between Authentication and Login?

Authentication and login are often used interchangeably, but they serve distinct purposes in the realm of digital security.

Authentication

Authentication is the broader process of verifying a user's identity before granting access to a system or platform. It involves confirming that the user is who they claim to be. This verification car occur through various methods, such as passwords, biometrics, security tokens, or multifactor authentication (MFA).

Login

Login, on the other hand, is the specific act of gaining access to a system or application using verified credentials. It is a subset of authentication, representing the moment when a user enters their credentials (username and password, for example) to access their account.

In essence, authentication is the overall process of confirming identity, while login is the specific action taken to enter a system using authenticated credentials.

Figure 2: Difference between authenticating

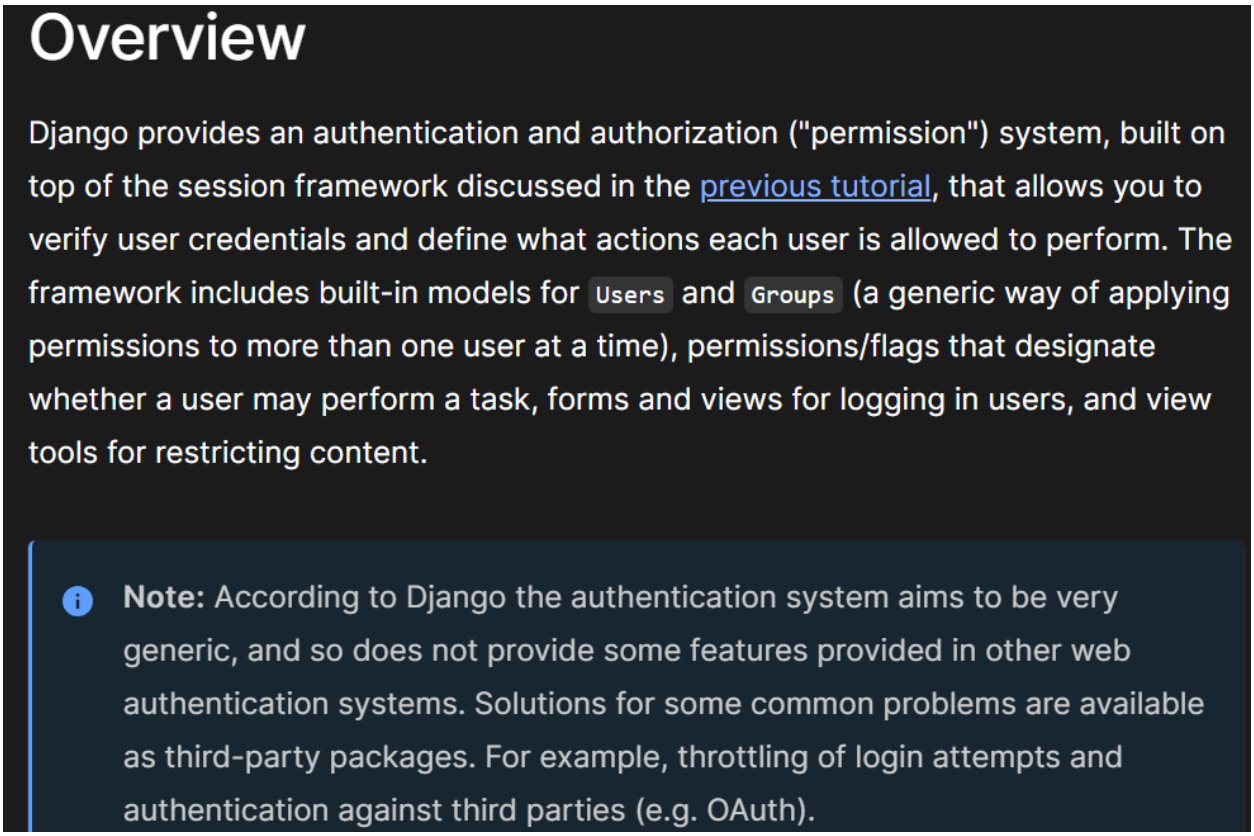

Figure 3: Overview of authentication.

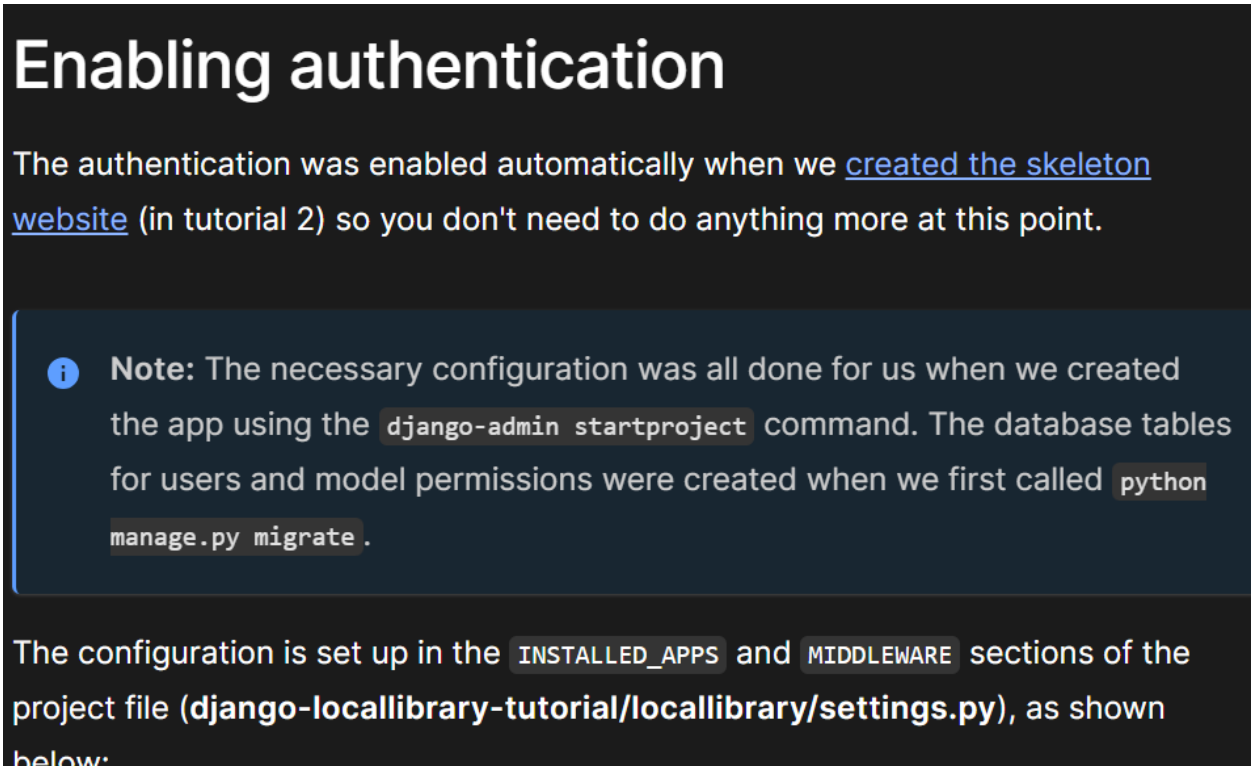

Figure 4: Enabling authentication.

PYTHON

```python
INSTALLED_APPS = [
    # …
    'django.contrib.auth',  # Core authentication framework and its default
models.
    'django.contrib.contenttypes',  # Django content type system (allows
permissions to be associated with models).
    # …

MIDDLEWARE = [
    # …
    'django.contrib.sessions.middleware.SessionMiddleware',  # Manages sessions
across requests
    # …
    'django.contrib.auth.middleware.AuthenticationMiddleware',  # Associates
users with requests using sessions.
    # …
```

Figure 5: Implementation of the Django framework

PYTHON

```python
from django.contrib.auth.models import User

# Create user and save to the database
user = User.objects.create_user('myusername', 'myemail@crazymail.com',
'mypassword')

# Update fields and then save again
user.first_name = 'Tyrone'
user.last_name = 'Citizen'
user.save()
```

Figure 6: create a new user for the authentication check

Note however that it is highly recommended to set up a *custom user model* when starting a project, as you'll be able to easily customize it in the future if the need arises. If using a custom user model the code to create the same user would look like this:

PYTHON

```python
# Get current user model from settings
from django.contrib.auth import get_user_model
User = get_user_model()

# Create user from model and save to the database
user = User.objects.create_user('myusername', 'myemail@crazymail.com',
'mypassword')

# Update fields and then save again
user.first_name = 'Tyrone'
user.last_name = 'Citizen'
user.save()
```

Figure 7: create the register of the new user.

2. login authentication Implementation in the system

```python
# login authentication
class LoginUser(APIView):
    authentication_classes = [TokenAuthentication]
    permission_classes = [AllowAny]

    def post(self, request):
        try:
            email = request.data.get('email')
            password = request.data.get('password')

            if email and password:
                auth_user = authenticate(request, username=email, password=password)
                if auth_user:
                    user = Users.objects.filter(email=email).first()
                    if auth_user.is_active and not user.is_delete:  # Fix the condition
                        refresh = RefreshToken.for_user(auth_user)
                        access_token = str(refresh.access_token)

                        serializer = UserSerializer(user)
                        profile = user.photo_url.url if user.photo_url else None

                        response_auth = {
                            'id': serializer.data['id'],
                            'username': serializer.data['username'],
                            'email': serializer.data['email'],
                            'profile': profile,
                            'role': serializer.data['role'],
                            'access_token': access_token,
                        }
                        return Response({'message': 'Login successful', 'is_success': True, 'status': 200, "auth": response_auth})
                    else:
                        return Response({'message': 'Your account is not activate', 'is_success': False, 'status': 401})
                else:
                    return Response({'message': 'The account does not have authentication permission.', 'is_success': False, 'status': 401})
            else:
                return Response({'message': 'Please provide email and password', 'is_success': False, 'status': 400})
        except Exception as e:
            return Response({"message": "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
```

(variable) photo_url: ImageFieldFile

Figure 8: Implementation of the authentication function

3 JWT access token

`pip install djangorestframework-simplejwt`

The djangorestframework-simplejwt package provides a simple way to implement JWT authentication in Django REST framework applications. It includes views and serializers for generating and refreshing JWT tokens, as well as a built-in token authentication backend for validating tokens.

Next we can then include it in the installed apps and add it to the settings:

```
INSTALLED_APPS = [


    'rest_framework_simplejwt.token_blacklist',
]
```

Figure 9: Django rest framework simple jwt implement

```python
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=180),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=50),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',

    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,
    'JWK_URL': None,
    'LEEWAY': 0,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'AUTH_HEADER_NAME': 'HTTP_AUTHORIZATION',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'USER_AUTHENTICATION_RULE': 'rest_framework_simplejwt.authenticatior

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken'
    'TOKEN_TYPE_CLAIM': 'token_type',
    'TOKEN_USER_CLASS': 'rest_framework_simplejwt.models.TokenUser',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

Figure 10: JWT token implementation.

```python
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(days=7),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=7),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,
    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
}
```

Figure 11: JTW implemented in the projects