Generic Example

Assuming we have an app that will allow for image uploading, and will have Permisions and Authentications

PS: *The code snippets in this summary are meant to illustrate each step of creating a Django REST API, but they are not necessarily related to each other. If you want to follow along, please use them as examples for each individual step, but do not assume that they form a coherent and integrated example.*

1. Create and activate your virtual env

```
virtualenv venv
source ./venv/bin/activate
```

2. Install django and rest_framework then start a project

```
pip install django
pip install djangorestframework
django-admin startproject <project_name> .
```

3. Put rest framework in the installed apps in **settings.py**

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
# 3rd party
'rest_framework', # new
]
```

4. Create a new app

```
python3 manage.py startapp <app_name>
```

7. Put your new app in the settings.py installed apps

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
# 3rd party
'rest_framework',
# Local
'appname.app.AppnameConfig', #new
]
```

5. Configure the urls to include urls form your new app

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path # new
from django.conf.urls.static import static

urlpatterns = [
path('admin/', admin.site.urls),
path('api/', include('your_app.urls')), # new
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
# the static line will include all media files you upload as
# individual endpoints so they can be accessed by a
# frontend framework
```

6. Install pillow and django-cleanup to work with images

```
pip install pillow
pip install django-cleanup
```

7. Put django-cleanup in your intsalled apps in settings.py

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
```

```
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
# 3rd party
'rest_framework',
'django_cleanup.apps.CleanupConfig', #new
# Local
'appname.apps.AppnameConfig',
]
```

8. Configure your project to upload files to media directory

settings.py

```
import os
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

you need to create the media directory in the root of your project and inside it you can create directories for different image categories

9. Start Creating your models

```
 #  your_app/models.py
from django.db import models

class Todo(models.Model):
  title = models.CharField(max_length=200)
  body = models.TextField()
  image = ImageField(upload_to="
  def __str__(self):
      return self.title
```

10. Make migrations and migrate (remember to make migrations for a specific app to make it easier in debugging)

```
python3 manage.py makemigrations <app_name>
python3 manage.py migrate
```

11. Register your admin

```
from django.contrib import admin
from .models import <your Models>
admin.site.register(your_model)
```

12. Create SuperUser to login to your admin

```
python manage.py createsuperuser
```

then follow the propmt to create username and password for the admin

13. Make your app serializers

```
from rest_framework import serializers
from .models import <your Models>

class YourModelSerializer(serializers.ModelSerializer):
class Meta:
        model = <ModelName>
        fields = ('id', 'title', 'body',) # available fields or '__all__'
```

14. Make your Views

```
from rest_framework import generics
from .models import <your Models>
from .serializers import < your serializers>

# endpoint that will return list of objects
class ListTodo(generics.ListAPIView):
    queryset = your_model.objects.all()
    serializer_class = <your Serializer>

# endpoint that will return one object
class DetailTodo(generics.RetrieveAPIView):
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
```

take note that those views were read-only to allow for crud operations you can use **ListCreateAPIView** for making crud on a list view and **RetrieveUpdateDestroyAPIView**

from performing crud in a single object view. example:

```
class ListTodo(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

class DetailTodo(generics.RetrieveUpdateDestroyAPIView):
    queryset = Post.objects.all()
   serializer_class = PostSerializer
```

15. Make your app's urls

```
from django.urls import path
from .views import ListTodo, DetailTodo

urlpatterns = [
path('<int:pk>/', DetailTodo.as_view()),
path('', ListTodo.as_view()),
]
```

16. Deal with Cross Origin Resource Sharing (cors)

```
pip install django-cors-headers
```

in your settings.py

- add corsheaders to the INSTALLED_APPS
- add CorsMiddleware above CommonMiddleWare in MIDDLEWARE
- create a CORS_ORIGIN_WHITELIST

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
# 3rd party
'rest_framework',
'corsheaders', # new
# Local
'todos',
```

```
]

MIDDLEWARE = [
'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'corsheaders.middleware.CorsMiddleware', # new
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# new
CORS_ORIGIN_WHITELIST = (
'http://localhost:3000',
'http://localhost:8000',
)
# we added both react port and django port in the whitelist
```

17. You can create tests

```
from django.test import TestCase
from .models import Todo

class TodoModelTest(TestCase):
    # class method to create the object you will test
    @classmethod
    def setUpTestData(cls):
        Todo.objects.create(title='first todo', body='a body here')

    # fisrt test case to test the title
    def test_title_content(self):
        todo = Todo.objects.get(id=1)
        expected_object_name = f'{todo.title}'
        self.assertEqual(expected_object_name, 'first todo')

    # second test case to test the todo body
    def test_body_content(self):
        todo = Todo.objects.get(id=1)
        expected_object_name = f'{todo.body}'
        self.assertEqual(expected_object_name, 'a body here'
```

Permissions

We add permissions in a website where we can have multiple users and we want to add permissions specific to each user

in order to use django built in permissions you must import django built in User model and link any model you want to add permissions on to that user model using a foreign key

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

class Post(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=50)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def  __str__(self):
        return  self.title
```

to make testing our permissions easier we need to configure django to allow login and logout from the browsable API

Within the project-level urls.py file, add a new URL route that includes rest_framework.urls.

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
path('admin/', admin.site.urls),
path('api/v1/', include('posts.urls')),
path('api-auth/', include('rest_framework.urls')), # new
]
```

we can have project-level permissions, views-level permissions and custom permissions

Project Level Permissions

Django REST Framework ships with a number of built-in project-level permissions settings we can use, including:

- **AllowAny** - any user, authenticated or not, has full access
- **IsAuthenticated** - only authenticated, registered users have access
- **IsAdminUser** - only admins/superusers have access
- **IsAuthenticatedOrReadOnly** - unauthorized users can view any page, but only authenticated users have write, edit, or delete privileges

You can configure this inside the settings.py

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated', # new
    ]
}
```

Views Level Permissions

You need to import permissions and add permission_classes to your view with a specific value

```
from rest_framework import generics, permissions
from .models import Post
from .serializers import PostSerializer

class PostList(generics.ListCreateAPIView):
    permission_classes = (permissions.IsAuthenticated,)
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Custom permissions

You might want to add custom permissions other than the ones provided by django Internally, Django REST Framework relies on a BasePermission class from which all other

per-
mission classes inherit. That means the built-in permissions settings like AllowAny, IsAuthenticated,
and others extend it. Here is the actual source code

```
class BasePermission(object):
    """
    A base class from which all permission classes should inherit.
    """
    def has_permission(self, request, view):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True

    def has_object_permission(self, request, view, obj):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True
```

To create our own custom permission, we will override the has_object_permission method.

you will create permissions.py

```
from rest_framework import permissions

class IsAuthorOrReadOnly(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        # Read-only permissions are allowed for any request
        if request.method in permissions.SAFE_METHODS:
        return True
        # Write permissions are only allowed to the author of a post
        return obj.author == request.user
```

Now we can use it inside our view

```
class PostDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = (IsAuthorOrReadOnly,)
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Authentication

this is the implementation of django restframework built-in TokenAthentication

1. The first step is to configure our new authentication setting in the settings.py

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication', # new
    ],
}
```

2. Add the authtoken app which generates the tokens on the server. It comes included with Django REST Framework but must be added to our INSTALLED_APPS

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken', # new
    # Local
    'posts',
]
```

3. Since we have made changes to our INSTALLED_APPS we need to sync our database. Stop the server with Control+c. Then run the following command.

```
python3 manage.py migrate
```

4. install the package dj-rest-auth and add it to the installed apps to make your login and logout endpoints

```
pip install dj-rest-auth
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken',
    'dj_rest_auth', # new
    # Local
    'posts',
]
```

5. Update our config/urls.py file with the dj_rest_auth package.

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')), # new
]
```

now you can go to

- http://127.0.0.1:8000/api/v1/dj-rest-auth/login/.
- http://127.0.0.1:8000/api/v1/dj-rest-auth/logout/.
- http://127.0.0.1:8000/api/v1/dj-rest-auth/password/reset
- http://127.0.0.1:8000/api/v1/dj-rest-auth/password/reset/confirm

6. To make our registration endpoint we need to install a third-party package called **django-allauth**

```
pip install django-allauth
```

7. Make some configurations in the settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites', # new
    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken',
    'allauth', # new
    'allauth.account', # new
    'allauth.socialaccount', # new
    'dj_rest_auth',
    'dj_rest_auth.registration', # new
    # Local
    'posts',
]

EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend' # new
SITE_ID = 1 # new
```

8. we updated the apps so we need to migrate

```
python3 manage.py migrate
```

9. Add the new URL route for the registration

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
```

```
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')),
    path('api/v1/dj-rest-auth/registration/', # new
    include('dj_rest_auth.registration.urls')),
]
```

now there is a user registration endpoint at
http://127.0.0.1:8000/api/v1/dj-rest-auth/registration/

JWT Authentication

PS. *This is another alternative to the previous Authentication method*

1. Install the **djangorestframework-simplejwt**

```
pip install djangorestframework-simplejwt
```

2. Make the following configuration in the settings.py

```
INSTALLED_APPS = [
...........
# Register your app
'corsheaders',
'rest_framework',
'rest_framework_simplejwt.token_blacklist'   #new
]

REST_FRAMEWORK = {
     'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
      ],
}
.....
SIMPLE_JWT = {
     'ACCESS_TOKEN_LIFETIME': timedelta(minutes=10),
     'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
     'ROTATE_REFRESH_TOKENS': True,
```

```
    'BLACKLIST_AFTER_ROTATION': True
}
```

3. Create the URL for access token and refresh token in main/urls.py file.

```
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt import views as jwt_views

urlpatterns = [
    .....
    path('token/',
         jwt_views.TokenObtainPairView.as_view(),
         name ='token_obtain_pair'),
    path('token/refresh/',
         jwt_views.TokenRefreshView.as_view(),
         name ='token_refresh')
]
```

4. migrate your database

```
python3 manage.py migrate
```

Now we must create a logout functionality

5. To create logout API, we will write the LogoutView in our views.py file in app

```
class LogoutView(APIView):
    permission_classes = (IsAuthenticated,)

    def post(self, request):
        try:
            refresh_token = request.data["refresh_token"]
            token = RefreshToken(refresh_token)
            token.blacklist()
            return Response(status=status.HTTP_205_RESET_CONTENT)
         except Exception as e:
            return Response(status=status.HTTP_400_BAD_REQUEST)
```

6. create the URL for LogoutView in our urls.py file in app directory.

```
urlpatterns = [
.....
    path('logout/', views.LogoutView.as_view(), name ='logout')
]
```

By using that URL http://localhost:8000/logout/, we can invalid the refresh token and add token into blacklist. By using Postman we can hit this URL with POST request and pass the access token in authorization and refresh token in body.