



## Django Class Notes

---

Clarusway



### Flight Project 2 - User App

---

Nice to have VSCode Extentions:

- Djaneiro - Django Snippets
- SQLite Viewer

Needs

- Python
- pip
- virtualenv
- .gitignore file
- .env file
- Postman

### Summary

- Create Users App
- Authentication
  - Setting the authentication scheme
  - Install dj-rest-auth

### Create Users App

- Create users app to handle login, logout, and register.

```
python manage.py startapp users
```

- Go to settings/base.py and add under INSTALLED\_APPS:

```
# My apps:  
'users',
```

- Create a superuser:

```
python manage.py createsuperuser
```

- Enter a username and password. Password will not be visible on the screen.
- After that, run the server again and go to <http://localhost:8000/admin> in your browser. Login with username and password you created.

## Authentication

### Setting the authentication scheme

The default authentication schemes may be set globally, using the `DEFAULT_AUTHENTICATION_CLASSES` setting. We will use token auth.

```
REST_FRAMEWORK = {  
    'DEFAULT_AUTHENTICATION_CLASSES': [  
        'rest_framework.authentication.TokenAuthentication',  
    ]  
}
```

## Install dj-rest-auth

In this project, we will use dj-rest-auth. Follow [documentation](#) and install.

- Install package:

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

- Update requirements.txt:

```
pip freeze > requirements.txt
```

- Add `dj_rest_auth` app to `INSTALLED_APPS` in your django settings:

```
'rest_framework.authtoken',  
'dj_rest_auth',
```

- Migrate your database, this will apply db tables originated from auth token:

```
python manage.py migrate
```

- Include `users.urls` to main url pattern. Create `users/urls.py` and add `dj_rest_auth`:

```
urlpatterns = [  
    path('dj-rest-auth/', include('dj_rest_auth.urls'))  
]
```

- Check the endpoints;
  - Login (See the token created by package)
  - Logout
- For registration, `dj-rest-out` can be used. We will write from scratch for this project.
- A simple register serializer can be:

```
from rest_framework import serializers  
from django.contrib.auth.models import User  
  
class RegisterSerializer(serializers.ModelSerializer):  
  
    class Meta:  
        model = User  
        fields = (  
            'username',  
            'email',  
            'password',  
            'first_name',  
        )
```

- This one has some issues. No password confirmation, email is not a required field, password can be seen when typing, no email validation etc.
- Use already registration serializer we did in previous session:
- Serializer:

```
from rest_framework import serializers
from django.contrib.auth.models import User
from rest_framework.validators import UniqueValidator
from django.contrib.auth.password_validation import validate_password
```

```
class RegisterSerializer(serializers.ModelSerializer):
```

```
    email = serializers.EmailField(
        required = True,
        validators = [
            UniqueValidator(queryset=User.objects.all())
        ],
    )
```

```
    password = serializers.CharField(
        write_only = True, # GET methods can not return the password
        required = True,
        validators = [
            validate_password
        ],
        style = {
            'input_type': 'password',
        }
    )
```

```
    password2 = serializers.CharField(
        write_only = True,
        required = True,
        style = {
            'input_type': 'password',
        }
    )
```

```
class Meta:
    model = User
    fields = (
        'username',
        'email',
        'password',
        'password2',
    )
```

```
def create(self, validated_data):
    password = validated_data.pop('password')
    validated_data.pop('password2')
    user = User.objects.create(**validated_data)
    user.set_password(password)
    user.save()
    return user
```

```
def validate(self, data):
```

```

    if data.get('password') != data.get('password2'):
        data = {
            "password": "Password fields does not match!!!"
        }
        raise serializers.ValidationError(data)
    return data

```

- Create view:

```

from rest_framework.generics import CreateAPIView
from django.contrib.auth.models import User
from .serializers import RegisterSerializer

class RegisterView(CreateAPIView):
    queryset = User.objects.all()
    serializer_class = RegisterSerializer

```

- Set the url for register

```

from users.views import RegisterView

path('register/', RegisterView.as_view()),

```

- Test register endpoint.
- [Generating Tokens](#)
- Create signals.py and write logic to get token after user creation:

```

from django.contrib.auth.models import User
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token

@receiver(post_save, sender=User)
def create_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)

```

- To make this signal working go to apps.py and write ready method

```

def ready(self):
    import users.signals

```

We need to send the token to frontend in views

```
from rest_framework.response import Response
from rest_framework import status
from rest_framework.auth_token.models import Token

def create(self, request, *args, **kwargs):
    serializer = self.get_serializer(data=request.data)
    serializer.is_valid(raise_exception=True)

    user = serializer.save()
    data = serializer.data
    if Token.objects.filter(user=user).exists():
        token = Token.objects.get(user=user)
        data['token'] = token.key
    else:
        data['token'] = 'No token created for this user!!!'

    headers = self.get_success_headers(serializer.data)
    return Response(data, status=status.HTTP_201_CREATED, headers=headers)
```

- User can login or make requests after we add token to data and send it to the frontend.
- When we login we only send token.key to frontend. They may require more info like username. So we need to create nested serializer to send additional data.
- Check [dj-rest-auth](#) and see how to customize

```
REST_AUTH_SERIALIZERS = {
    # 'TOKEN_SERIALIZER': 'path.to.custom.TokenSerializer',
    'TOKEN_SERIALIZER': 'users.serializers.CustomTokenSerializer',
}
```

If not specified TOKEN\_SERIALIZER - response for successful authentication in `dj_rest_auth.views.LoginView`, default value `dj_rest_auth.serializers.TokenSerializer` and this serializer only returns token.key. Lets overwrite it

- Check TokenSerializer class, only field is key
- Go to serializers.py and create a nested serializer

```
from dj_rest_auth.serializers import TokenSerializer

class UserSerializer(serializers.ModelSerializer):

    class Meta:
```

```
model = User
fields = (
    'id',
    'username',
    'email',
)

class CustomTokenSerializer(TokenSerializer):

    user = UserSerializer(read_only=True)

    class Meta(TokenSerializer.Meta):
        fields = (
            'key',
            'user',
        )
```

- Test login.

😊 Happy Coding! 📁

