## These slides are part of the downloadable resources of ***The Complete Guide to Django REST Framework and Vue JS*** Udemy course.

**Let’s keep in touch:**

►YouTube Channel: https://www.youtube.com/channel/UCxPWtz5N--X3IyYJ13Zr99A?sub_confirmation=1

►Twitter: https://www.twitter.com/pymike00

►GitHub: https://github.com/pymike00

# Django REST Framework

Level Three - Objectives

## DRF Level Three - Objectives

In this section we are going to discuss the most advanced aspects of the creation of a REST API with Python and Django REST Framework, completing the acquisition of the knowledge and skills that are necessary to create real-world use cases, as we'll discuss in great detail in the final section of the course.

## DRF Level Three - Objectives

- Knowledge of the main Authentication Methods in DRF
- Set Up a Registration and Authentication System via REST
- Familiarity with Django-REST-Auth
- Knowledge of Viewset and Router Classes
- Filtering via Django REST Framework
- Being able to write automated tests with Django and DRF

- Bonus: how to extend Django's User Model with a custom Profile Model

# Django REST Framework

## Level Three - Lessons

- UserProfiles API - Setup
- Authentication in DRF
- Django-REST-Auth Pt. 1
- Django-REST-Auth Pt. 2
- Viewset & Router Classes
- The Filtering System
- Automated Tests Pt. 1
- Automated Tests Pt. 2

# UserProfiles API

Project Setup

## DRF Level Three - UserProfiles API

In this lesson we are going to start developing a user-profile-based Django project, that we are then going to use and customize throughout the whole section.

We will see how to extend Django’s basic User model with a second model that we’ll create, allowing us to store further information about our users such as a user’s biography, city and avatar.

Each user will then be able to write status messages that will be bind to his profile, similarly to what happens in most Social Networks.

## DRF Level Three - UserProfiles API

This will also give us the chance to introduce, to those who do not already know them, Django’s **Signals**!

Signals allow certain *senders* to notify a set of *receivers* that some action has taken place elsewhere in the framework.

In our UserProfile API project we are going to use signals to automatically create and bind a Profile’s Instance to a User Object as soon as a new one is created.

**Let’s get started!**

# UserProfiles API - Reference Links

https://docs.djangoproject.com/en/2.1/topics/auth/customizing/#extending-the-existing-user-model

https://docs.djangoproject.com/en/2.1/topics/signals/

https://docs.djangoproject.com/en/2.1/ref/signals/#django.db.models.signals.post_save

## Authentication in DRF

## DRF Level Three - Authentication

We can define **Authentication** as the system that lets us associate a series of identification credentials to an incoming request, obtaining critical information such as the specific user who sent the request.

As we have seen in the previous lectures, we can then use a system of permissions to decide whether or not to accept the incoming request, for example based on the type of user who sent it (admin users, basic users and so on...)

## DRF Level Three - Authentication

It is important to underline the difference between authentication and permissions!

**Authentication is always run at the very start of our views**, before the authorization checks occur, and before any other code is allowed to be executed.

Authentication by itself won't allow or disallow an incoming request, it simply identifies the credentials that the request was made with.

## DRF Level Three - Authentication

Django REST Framework provides us **different authentication systems** *out of the box*. In this lesson we are going to talk about the most important ones, evaluating their pros and cons, their most appropriate use cases and implementation details.

We are also going to talk about a new authentication standard called **JWT**, that can be easily implemented in Django REST Framework with the help of some **third party packages**.

## Basic Authentication

## DRF - Basic Authentication

It’s the **most primitive** and the **least secure** authentication system provided by Django REST Framework. The request/response cycle looks like this:

- The client makes a HTTP request to the server
- The server responds with a HTTP 401 Unauthorized response containing the WWW-Authenticate header, explaining how to authenticate (WWW-Authenticate: Basic)
- The client sends its auth credentials in base 64 with the Authorization header. Authentication credentials are here unencrypted.
- The server evaluates the access credentials and responds with the 200 or 403 status code, therefore authorizing or denying the client’s request.

## DRF - Basic Authentication

Because of its low efficiency and security standards, Basic Authentication is generally only appropriate for testing.

As DRF's documentation suggests:

*If you use `BasicAuthentication` in production you must ensure that your API is only available over `https`. You should also ensure that your API clients will always re-request the username and password at login, and will never store those details to persistent storage.*

## Token Authentication

## DRF - Token Authentication

This is the ideal system for authenticating smartphone and desktop clients.
The request/response flow looks like this:

- The client sends it’s authentication credentials once
- The server checks the credentials, and if they are valid it creates an *exclusive signed token* made of a string of characters that then sends back to the client as response
- The client sends its token within the *Authorization Header* of every following request
- The server checks the received token and if valid, allows the request to proceed.

## DRF - Token Authentication

We will often use this authentication scheme during this section, using it with clients that we will write ourselves using the *Requests* module.

Token authentication is also often used with *Single Page Applications* written using frameworks like React and Vue.

Often times in such cases the authentication token gets saved either in a cookie or in the browser’s localStorage...

## DRF - Token Authentication

…but it is very important to underline that saving the authentication token in localStorage is very dangerous, as it makes it vulnerable to XSS attacks!

Using a httpOnly cookie on the other hand is much safer because this way the token can’t be accessed via JavaScript… but because of that you are now losing the flexibility that you would get by using localStorage instead!

## DRF - Token Authentication

Regarding Single Page Applications, it might be true that there is no authentication scheme that is “universally accepted” by the development community, as each project has different requirements. However it is always very important to opt for secure and reliable solutions whenever possible.

For circumstances like ours, Django REST Framework’s official documentation suggests to use **Session Authentication**.

## Session Authentication

## DRF - Session Authentication

This authentication scheme uses Django's default session backend for authentication.

Session authentication is the safest and most appropriate way for authenticating AJAX clients that are running in the same session context as your website, and uses a combination of **Sessions** and **Cookies**.

It is the authentication scheme that we will use in the Final Project.

**How does it work?**

## DRF - Session Authentication

The request/response cycle looks like this:

- Users send their authentication credentials, typically using a Login Form
- The server checks the data and if correct, it creates a corresponding *Session Object* that will be saved in the database, sending back to the client a *Session ID*
- The Session ID gets saved in a Cookie in the browser and will be part of every future request to the server, that will check it every time
- When the client logs out, the Session ID is destroyed by both the client and the server, and a new one will be created at the next login.

## DRF - Session Authentication

If successfully authenticated using Session Authentication, Django will provide us the corresponding User Object, accessible via **request.user**.

For non-authenticated requests, an *AnonymousUser* instance will be provided instead.

## DRF - Session Authentication

**Important:** once authenticated via session auth, the framework will require a valid CSRF token to be sent for any *unsafe* HTTP method request such as PUT, PATCH, POST, DELETE.

The CSRF token is an important Cross-Site Request Forgery vulnerability protection, and we will see how to properly include it in our requests in the Final Project.

## BONUS:
## JSON Web Tokens

## DRF - JSON Web Tokens

JWT or **JSON Web Token**, is a new standard which can be used for token-based authentication.

One of the main differences with other token-based standards is that because of their structure, JWT tokens don’t require database validation.

JSON Web tokens can be easily used in a DRF powered REST API using the *django-rest-framework-simplejwt* package, which can be installed via pip.

# Authentication in DRF - Reference Links

https://www.django-rest-framework.org/api-guide/authentication/

https://docs.djangoproject.com/en/2.1/ref/contrib/auth/#django-contrib-auth

https://docs.djangoproject.com/en/2.1/ref/contrib/auth/#anonymoususer-object

https://security.stackexchange.com/questions/988/is-basic-auth-secure-if-done-over-https

https://jwt.io/

## Django-REST-Auth

## DRF Level Three - Django-REST-Auth

In this lesson you will learn how to use the Django-REST-Auth package in order to expose registration and authentication endpoints for your REST APIs!

Thanks to these endpoints, clients such as Android and iOS apps will be able to easily and independently communicate with *all* the services provided by your web app's backend, via REST.

To make learning easier, this lecture will be split in two parts.

**Let's Code!**

## Django-REST-Auth - Reference Links

https://django-rest-auth.readthedocs.io/en/latest/

https://github.com/pennersr/django-allauth

## ViewSets & Routers

# DRF Level Three - ViewSets & Routers

**ViewSet** classes allow us to combine the logic for a set of related views in a single class: a ViewSet could for example allow us to get a list of elements from a queryset, but also allow us to get the details of a single instance of the same model.

ViewSet work at the highest abstraction level compared to all the API views that we have learned to use so far.

## DRF Level Three - ViewSets & Routers

ViewSets are in fact another kind of Class Based View, that does not provide any method handlers such as .get() or .post(), and instead provides action methods such as .list() and .create(), like some other classes we already know!

We will understand how and why by analyzing their source code.

ViewSets are typically used in combination with the **Router** class, allowing us to automatically get a url path configuration that is appropriate to the different kind of actions that the ViewSet provides, following convention standards.

## ViewSets & Routers - Reference Links

https://www.django-rest-framework.org/api-guide/viewsets/

https://www.django-rest-framework.org/api-guide/routers/

## Filtering in DRF

## DRF Level Three - Filtering

By default, the views that we have used so far to expose lists of elements have always returned whole querysets. Sometimes however, you might want to get specific results out of the same list endpoint, according to some predefined criteria.

In this lesson we will learn how to improve our REST API by using Django REST Framework’s filtering system and by personalizing the get_queryset() method in our views.

**Let’s get started!**

## Filtering in DRF - Reference Links

https://www.django-rest-framework.org/api-guide/filtering/

## Automated Testing