Django rest\_framework

Restful APIs

* d

Rest framework

* Django rest is a framework for Django that allows me to create restful apis so easy.
* In order for me to use Django I need to perform following steps:
  + Install it using pip:
    - Pip install djangorestframework
  + Add it to my Installed apps inside setting file
    - ‘rest\_framework‘
  + Go to views folder where I need to create my methods that handles for me the rest endpoints.
  + Import api\_view from decorators and response from class from rest\_framework.
    - From rest\_framework.decorators import api\_view
    - From rest\_framework.response import Response.
  + Then define my method and add api\_view decorator to it.
  + And inside response return Response( resource ) class.
  + Example:

@api\_view()

def all\_items(request):

return Response( ‘ Resource‘ ) # output is Resource string

* + To define a method that take parameter:

@api\_view()

def all\_items(request, id):

return Response( id ) # output is id value

* Then I create url file then create a path for these resources.
* Add path file to the app urls.
* Run server and hit on these endpoints.

Serializer

* Serializers convert a model object into dictionary key pair value.
* Serializer classes are external representation of my data but not all data I want to expose to api, this is where I need serializer.
* Serializers are external representation of my data and inside of I declare only the fields that I want to expose to the api and keep fields in my model I don’t to expose out of it.
  + The models I have are internal representation of data.
  + Serializer is external representation where my methods will interact with.
* Serializer has a class called JsonRenderer it has a method called render that takes a dictionary object and convert it to a json object.
* How to create serializer?
  + Create a serializer.py file and inside it:
    - Import serializers object from rest
      * From rest\_framework import serializers.
    - Then create a class that inherit from class Serializer inside serializers object.
      * Class ModelSerializer(serializers.Serialize):
    - This class take properties provided and convert them into dictionary.
      * Inside the class I define fields just like I am defining model fields: serializers.CharField(max\_length=255)
  + Inside views.py file:
    - I need to import two things, the serializer I created and the Model I want to server.
      * From serializers(my file .py) import ModelSerializer
      * From . import models import Model
    - Inside method I query my product with specific id.
    - Then I call ModelSerializer constructor and give me my query object.
    - Then in the Response I return my serializer.data to return data to user.
      * Return Response(serializer.data)
      * And because sometimes user request an object that does not exist, I should return 404.
        + To do that I wrap my response with try and except and except should return Model.DoesNotExist
      * Or I use get\_object\_or\_404 from Django.shortcuts:
        + From Django.shortcuts import get\_object\_or\_404 Then I invoke my object based on it:
        + It takes two parameters, model and the filter value.

get\_object\_or\_404(Model , pk=1)

* + - I can pass queryset to ModelSerializer() constructor that I created, and it take many property because it inherit from serializers.Serialze class.
    - Meaning I pass query set and property , the property tell the serializer to iterate over the queryset and generate json object list from it.
    - Example:
      * ProductSerializer(querySet, many=True)
* if I want to get consistent status code, I can import status object from rest framework
  + from rest\_framework import status.
    - It contains list of status code that I can repond with.

Create Serializer custom fields.

* When I want to change the field name, I will get error says that my serializer can’t find the field name in source Model.
  + To solve this inside the custom field type I use resource= ‘field\_name’
  + And the name will be updated.
* I can create a method field, for example I have a unit price and I want it with tax.
* In this case scenario I don’t have the field inside the model and I will get error:
* I can use the serializer class SerializerMethodField(‘method\_name’)
* And then I define a method that takes two parameters self and object have the price field
* Example:

Custom\_price = serializers.SerializerMethodField(‘ custom\_price’)

Def custom\_price(self, product):

Return product.unit\_price \* 1.6

**Serializing Relationships**

**There are 4 ways to serialize relations.**

1. **Get related field with id**
   1. **In order to get the id of related object I need to import the related model, then use following code and provide a queryset object like below to get the id.**
   2. **Serializers.PrimarykeyRelatedField( queryset=RelatedModel.object.all())**
2. **Get string values**
   1. **Convert each object of related field to string and return it with result.**
   2. **In order to achieve that I need 3 things:**
      1. **\_\_str\_\_ magic method to return the object string.**
      2. **Call method StringRelatedField()**
      3. **Because of lazy loading the result won’t get the related objects data by default, it will result my app to query n times which will kill the app performance.**
         1. **To solve this lazy loading issue, I need to use select\_related(‘collection’) method inside my method.**
3. **Get nested object.**
   1. **this way I need to create a serializer for my related model and add fields I need to return.**
   2. **Then I create a field and assign to it a Constructor of my related model I created in A step.**
4. **Get it as hyperlink**

**Model Serializer**

* **Model serializer is another way to create my serializer class.**
* **To create model serializer I have to first inherit from serializers .ModelSerializer instead of serializers.Serializer.**
* **Then create meta class.**
  + **Inside meta class I provide two fields:**
    - **Model = ModelName**
    - **Fields = [‘ field\_name’ , ‘etc…’]**
  + **I can override fields and use custom fields that I created by just creating my own methods.**

## Deserializing Objects

* **Deserialize data is when I receive data from client, I need to deserialize it.**
* **Serializing data is when sending data to client.**
* **To make my method support multi http methods I need to provide array of string that contains the http methods I want my method to serve.**
* **To use post method I need to define action inside api\_view() decorator. It takes list of string to represent http method.**
  + **@api\_view[‘POST’ , ‘GET ’]**
* **The second step that I need to do is to make my method support multiple methods by using if statement like below:**
  + **Request.method == ‘ GET‘**
    - **It will serialize data and send it to URL.**
  + **Request.method ==’POST’**
    - **Will deserialize data coming from body request.**
  + **Then my ModelSerialize receive data from request object request.data.**
  + **CustomModel = (data= request.data)**
    - **By passing request.data to data property I am telling serializer to convert fom json to real data.**
  + **And to access the data coming from request.data object I need to use serializer validated\_data.**
    - **Serializer.validated\_data**
  + **If I try to access the data directly I will get an error and it will raise an exception.**
    - **to solve this problem I need to call is\_valid() method.**

**Saving data to database.**

**Creating data:**

* **In order to save data to database all I need is to call serializer.save()**
* **Full example:**

**@api\_view([ “POST”])**

**Def products(request):**

**Serializer = ProductSerializer(data = request.data)**

**Serializer.is\_valid(raise\_exception=True):**

**Serializer.save()**

**Return Response(serializer.data, status= status.HTTP\_201\_CREATED)**

**Update one object.**

**@api\_view([“PUT”])**

**Def products (request, id):**

**Product = Product.objects.get(pk=id)**

**Serializer = ProductSerializer(product ,data = request.data)**

**Serializer.is\_valid(raise\_exception=True):**

**Serializer.save()**

**Return Response(serializer.data)**