**Create a Serializer:**

Adding a serializer to our view.

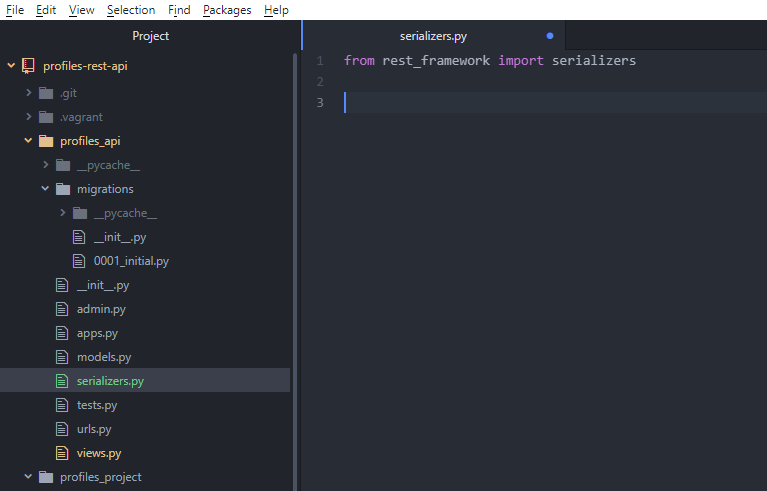
Serializer – Is a feature of DRF that allows us to convert data inputs into Python objects and vice versa.

(Similar to Django forms which has various fields that we want to accept as inputs)

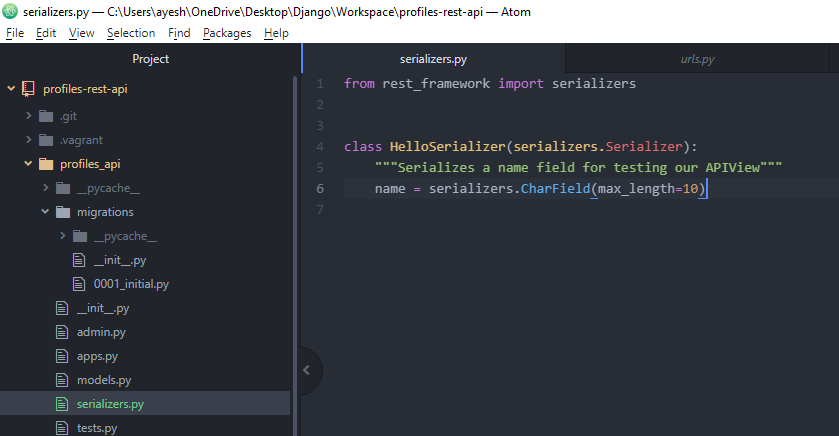
So if we include post or update functionality to our HelloApi View, we would need a serializer to receive the content that we post to the API.

1. In Atom editor, create a new file name serializer.py inside profiles\_api app

2. Perform Imports



3. Create Serializer class



Class

We will create a serializer that accepts name as input and then we will add this to our API view using which we will test the post functionality.

> Define the serializer

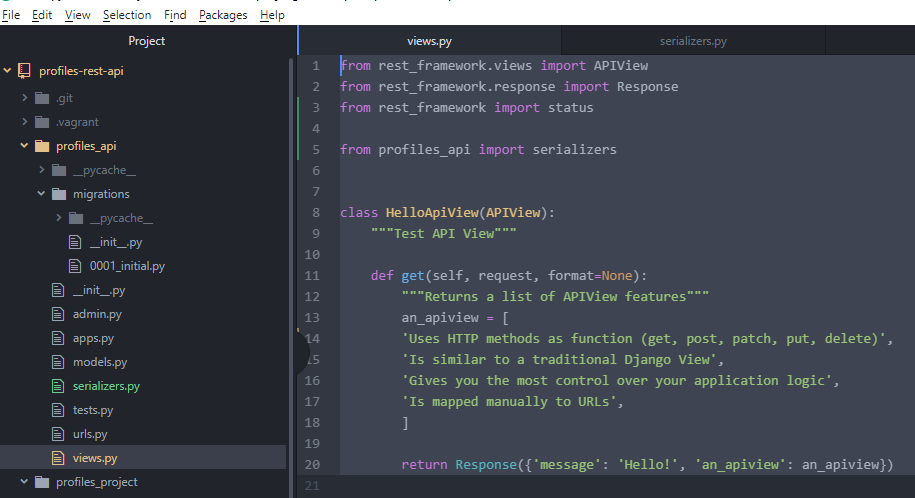
> Specify the fields which we want in our serializer input

Passed to the request that will be validated by serializer.

Serializer takes care of the validation rules. (For e.g. if we want input field of certain type, serializer makes sure that the input passed to the api is of correct type as we require for that field)

**Adding POST method to the API View:**

1. In Atom editor, go to views.py and perform some inputs.

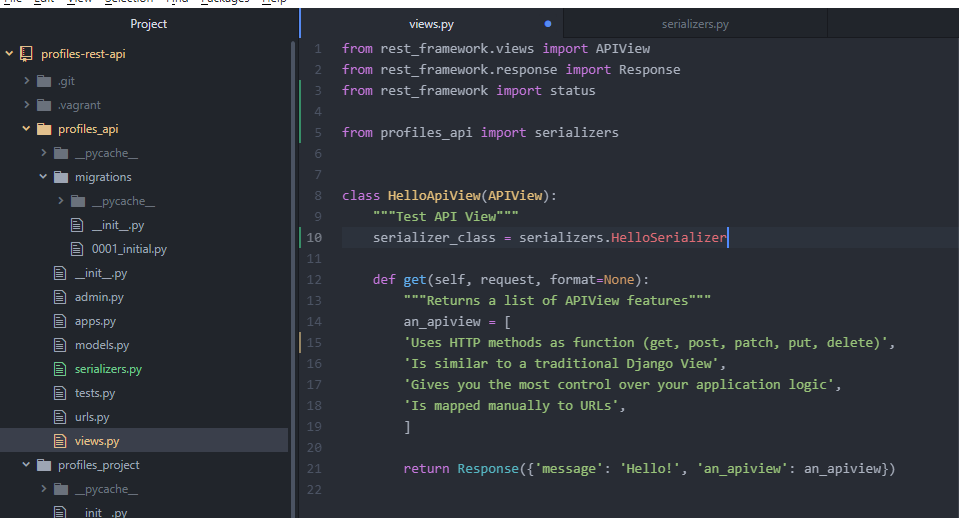


Is a list of handy HTTP status codes which we can use when returning responses from our API.

serializers module (serializers.py) created under profile\_api app.

(We will use it to tell what data to expect when making PUT, POST, PATCH request to the API.)

2. Making changes to the API view class



This configures our API View to have the serializer class that we created earlier. (serializers.py)

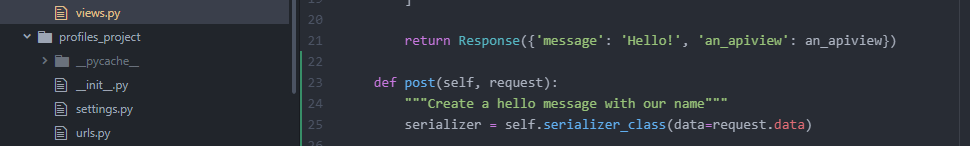
This means that whenever we are sending a post put or patch request, expect an input with name.

3. Add the POST function

Create a new function.

Retrieve the serializer and pass in the data that was sent in the request.

self.serializer\_class function comes default with the APIView that retrieves the configured serializer class for our view. (Standard way of retrieving the serializer class when working with serializer view)



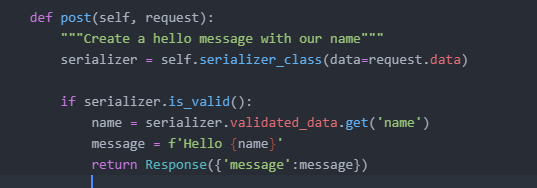
Assign the data. When we make a post request to our API view, the data gets passed in as request.data

(part of the request object that’s passed to our post request)

We assign this data to our serializer class and then we create a new variable for our serializer class called serializer.

4. Validate the serializer

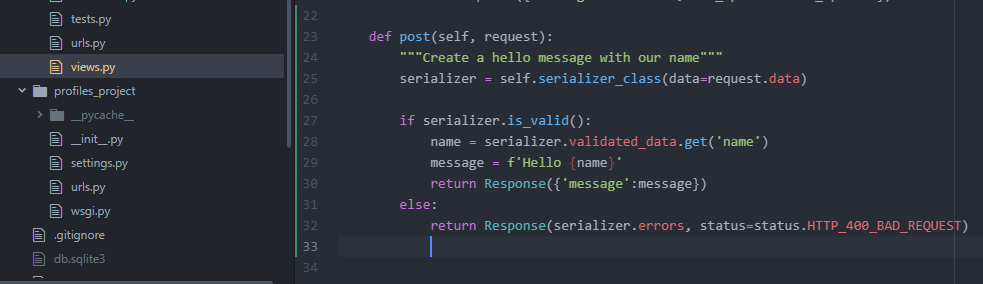
DRF serializers provide the functionality to validate the input i.e. assuring that the input is valid as per the specification of our serializer field.



If the serializer is valid (in this case 10 character length), then we will perform the logic. Here in this case we will retrieve the name field (defined in serializers.py) from the validated data.

Return this message from the API containing the name passed in the post request.

5. Else



Else return errors that were generated by the serializer.

serializer.errors – gives a dictionary of all the errors based on the validation rules that were applied to the serializer.

By default the response returns HTTP 200 OK request. We will change it to 400 Bad request since it is an error.

We return standard error response code for this type of error in an API.