Let’s create Django-Rest project

Pre-requirements:

1. Create venv
2. Activate it
3. Install Django
4. Install djangorestframework
5. Start new Django project called blog
6. Add rest framework

Add to INSTALLED\_APPS = (

...

'rest\_framework',

)

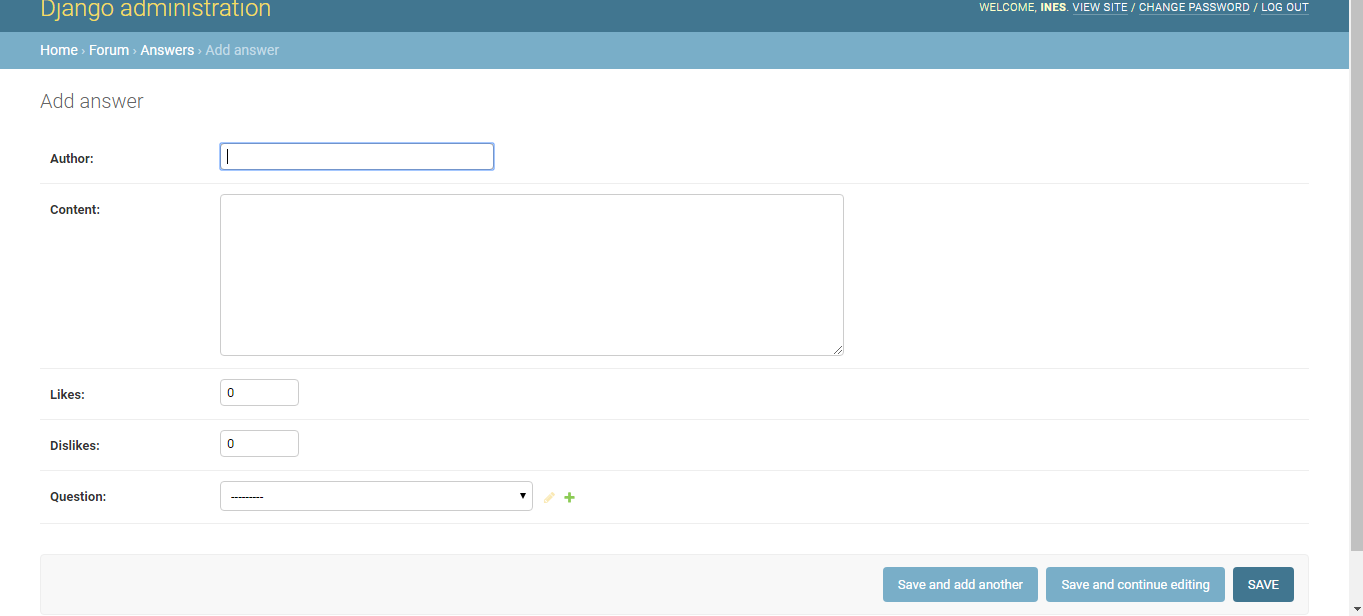
1. Start app called forum
2. Add the new app to installed apps
3. Check if you can start the server

**CRUD with REST**

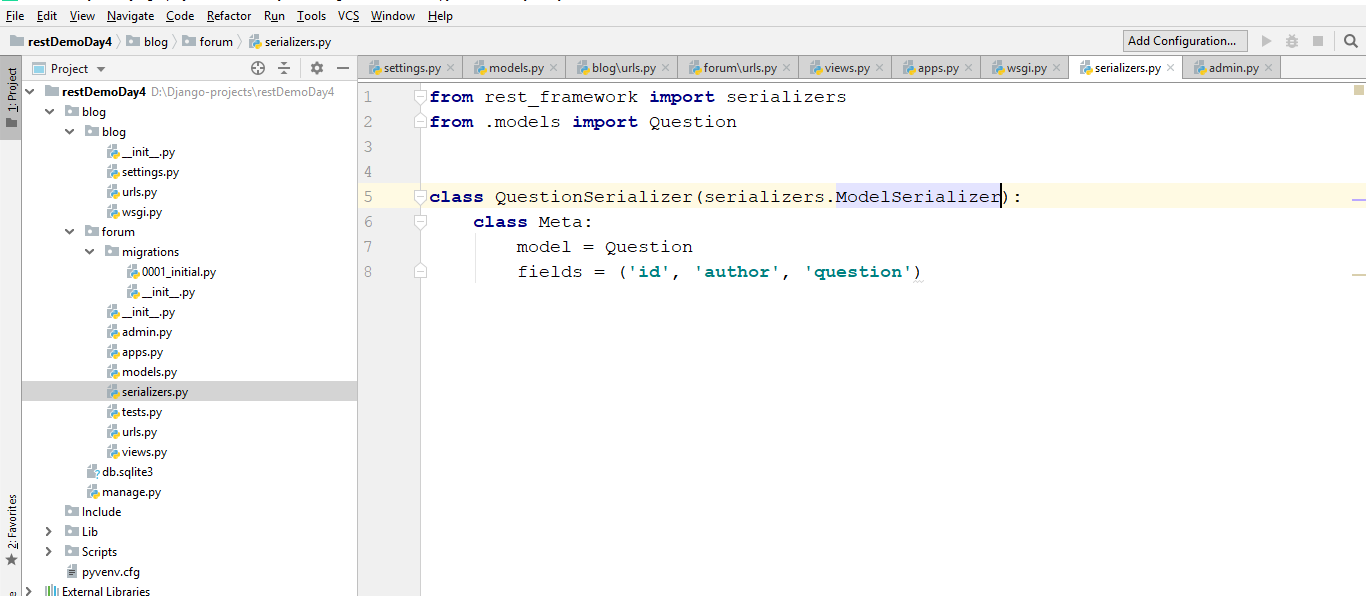
We are going to use class-based views, because it is going to be very easy to implement the CRUD with them and we are going to follow the best practices.

1. Create two models – Question and Answer, the question should have author and question fields, the answer should have author content, like, dislike fields
2. Decide on your own how to make a relationship between question and answer model in order to have one question with multiple answers
3. Overwrite toSTring methods of both of the classes and check in admin panel if the relationship is correct

You should see something like this for answer



1. After that create in our forum app a .py file called serializers
2. Go there and create a QuestionSerializer class

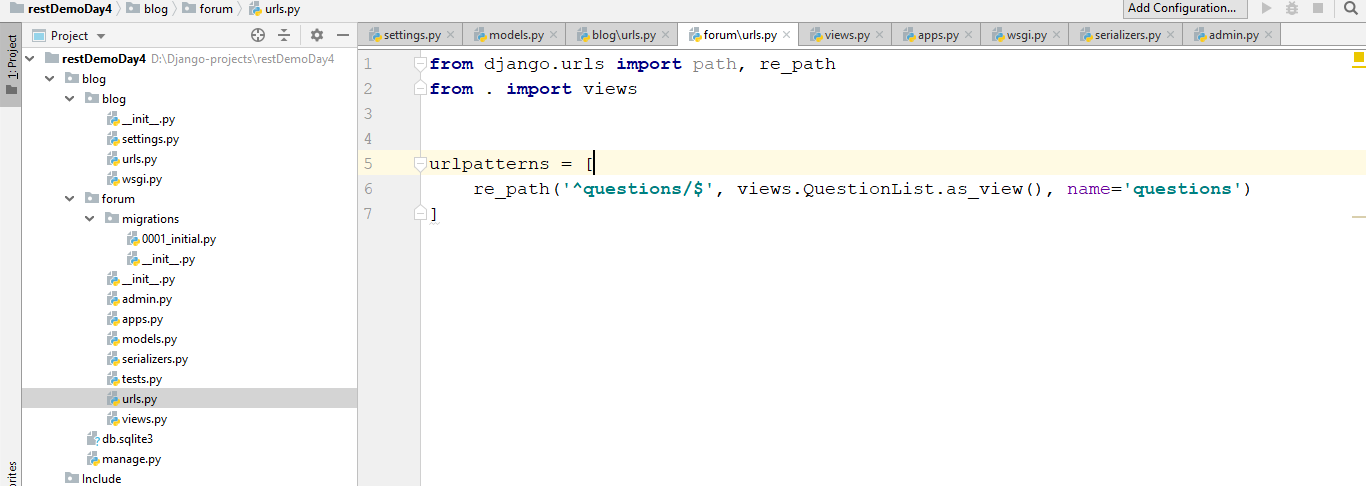


Note that we import serializers from rest\_framework

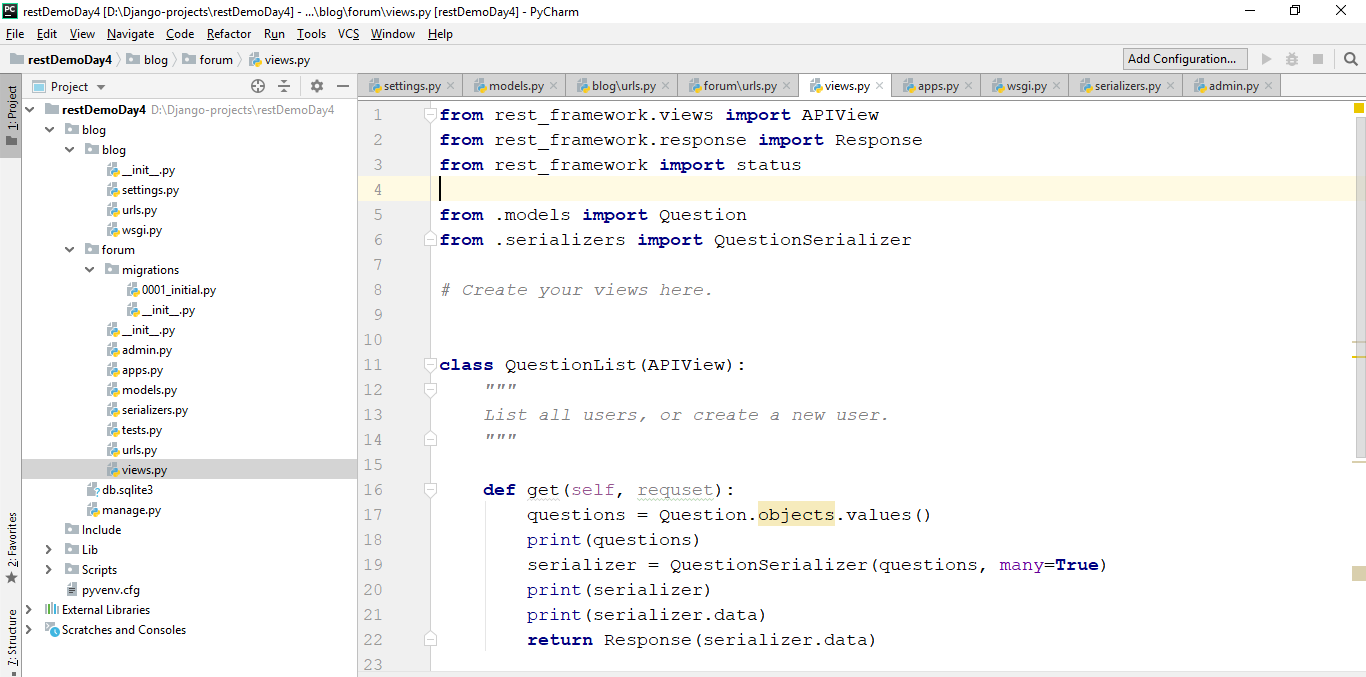
Be sure that you inherits from serializers.MODELSERIALIZER

In the meta class set model to our model and set the tuple for the fields for the fields we want in our case – all of them

1. Try fields to be equal to ‘\_\_all\_\_’ . Is there any difference?
2. Create urls.py file and set a path



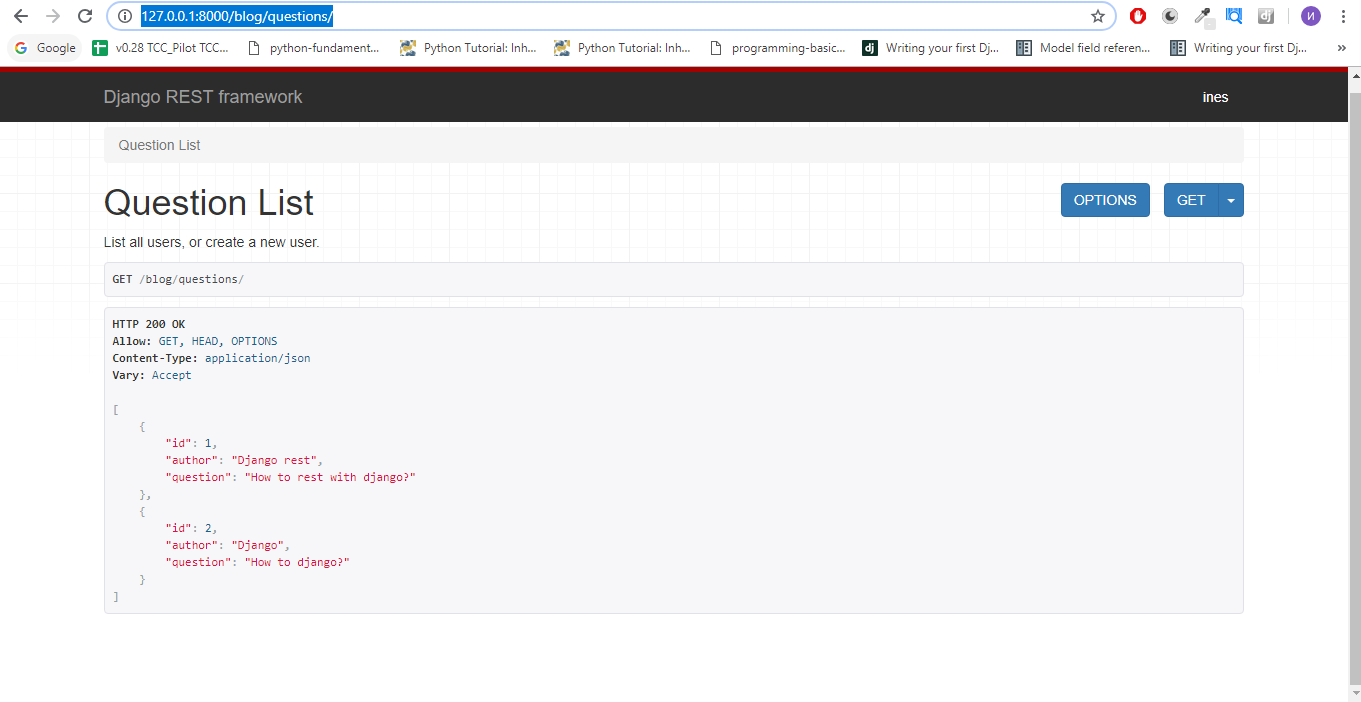
1. Create class in views.py called ‘QuestionList’



A couple of things to note here

* We have APIView which is very useful when we create REST applications – it gives us nice looking templates to see our results
* Note that we do not use functions here but classes. In this classes we have get, post, put and delete functions (def) and we do not need to check explicitly for request method
* Experiment with print method to see how the data changes after that remove the print methods
* Note that the Response is from rest\_framework.response also APIView and status

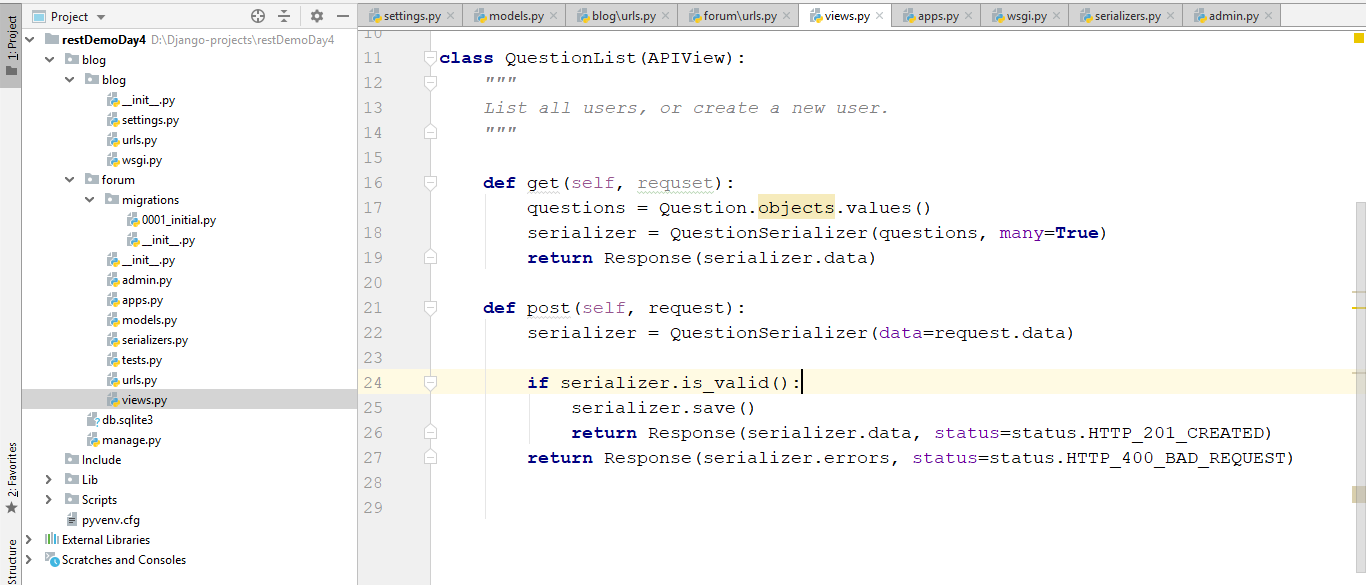
1. Go to <http://127.0.0.1:8000/blog/questions/> and check the results, but first create some questions through admin panel. If everything went well you should see the following result:



So far so good, but we have just the R from CRUD

1. Let’s do the C and make our application create a question. It is easier than you think – we need t write in our Question List class in views.py a function called post. It will accept self and request again

We need to say which is our serializer class which we will use in our case QuestionSerioalizer and say which data to use – we will get it from request.data. We check if it is valid – we save it and return the object, else we return the error. Here is how it looks like in our code:



It sounds complicated but it is just a few lines of code

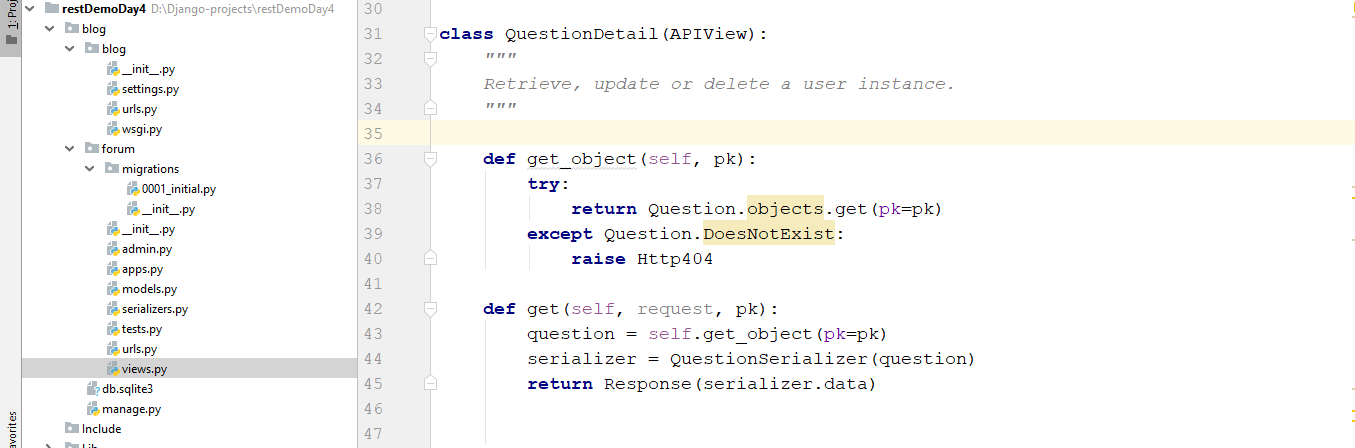
11.Next step is to build a read for a simple question, edit and delete

12. Let’s create the Read part – create new class QuestionDetail(APIView)

In it we need to define get\_object function and make the rules what to return

**def** get\_object(self, pk):  
 **try**:  
 **return** Question.objects.get(pk=pk)  
 **except** Question.DoesNotExist:  
 **raise** Http404

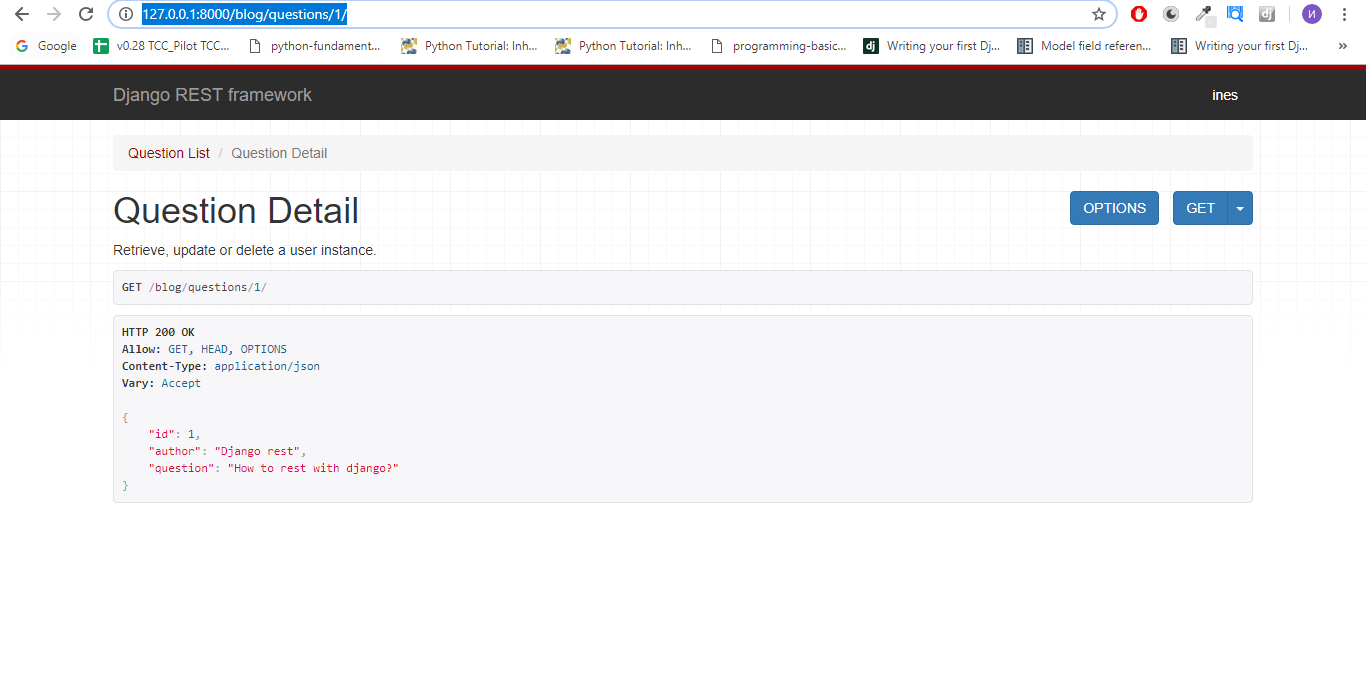
After that we should create get function which for the question calls self.get\_object, serialize it and return it



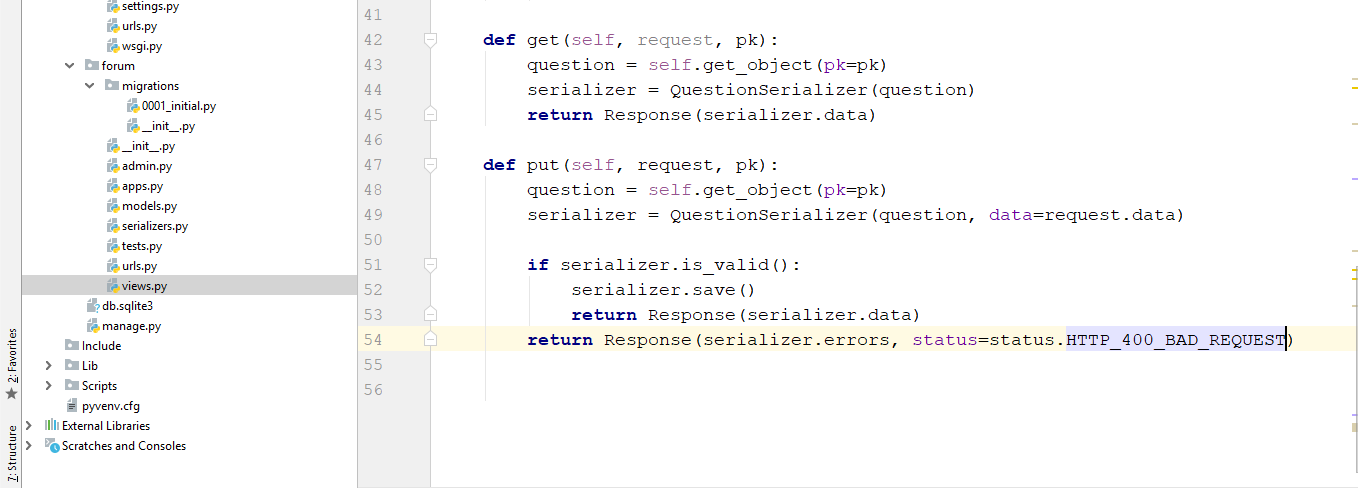
We need to write a url for this:

re\_path(**'^questions/(?P<pk>\d+)/$'**, views.QuestionDetail.as\_view(), name=**'detail'**)

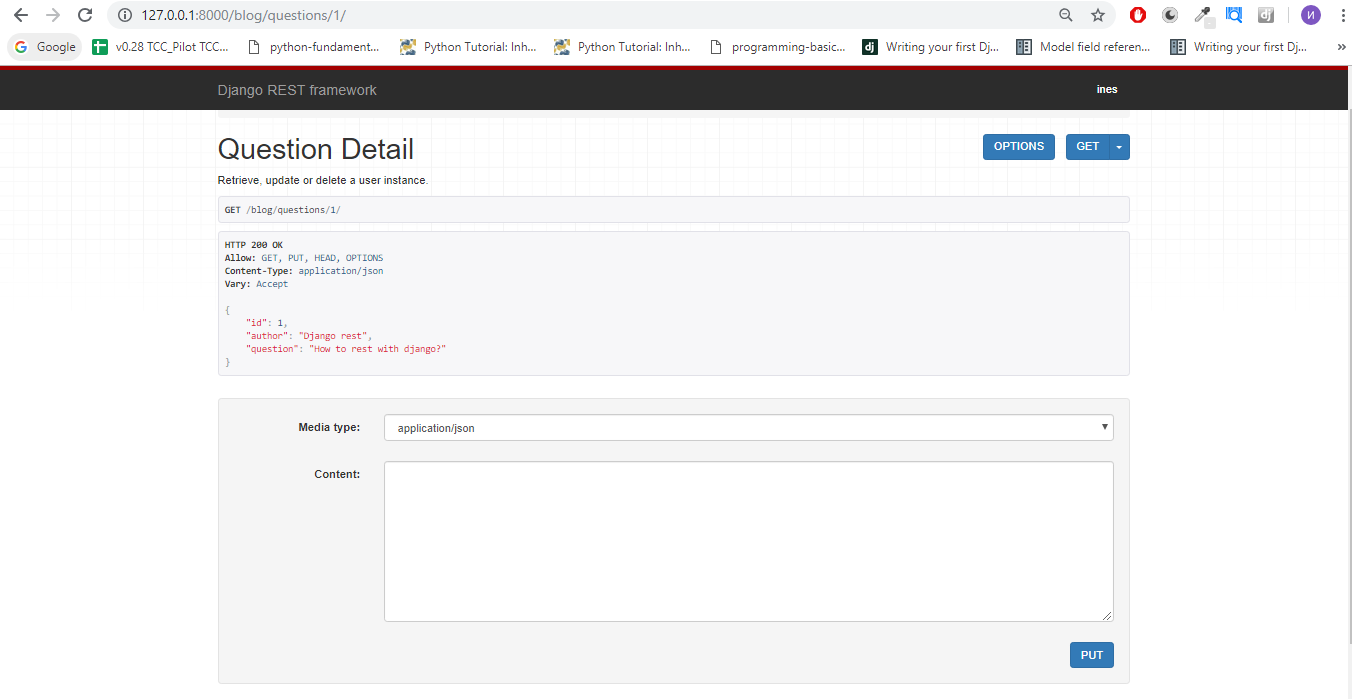
Let’s check the result – go to <http://127.0.0.1:8000/blog/questions/1/>



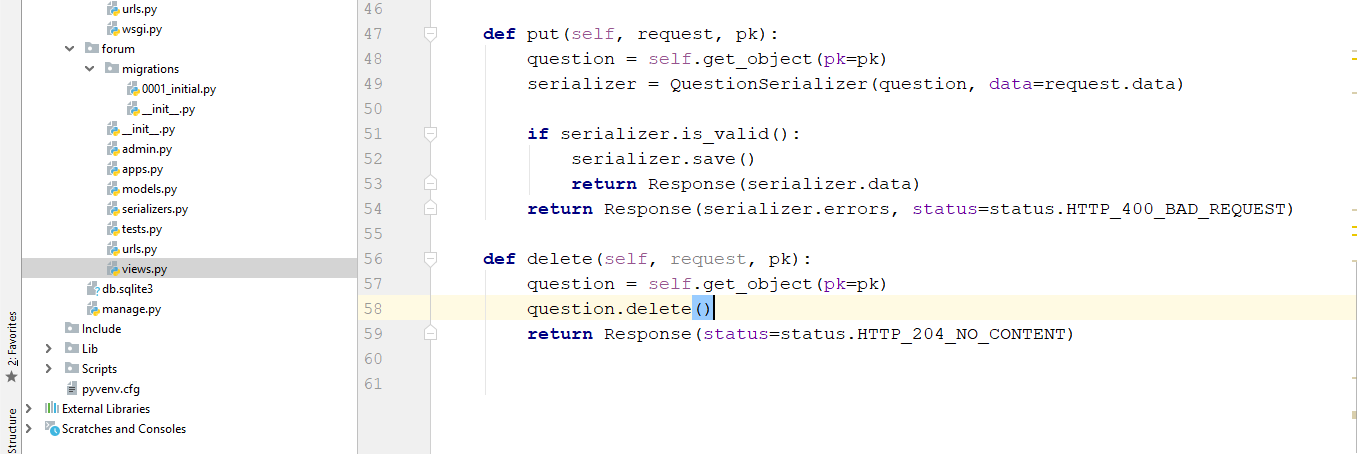
13. We should create UPDATE from CRUD, so in the same class we create def put which takes self, request and pk, again we will use our get\_object. If serializer is valid we save it if not return the serializer’s errors



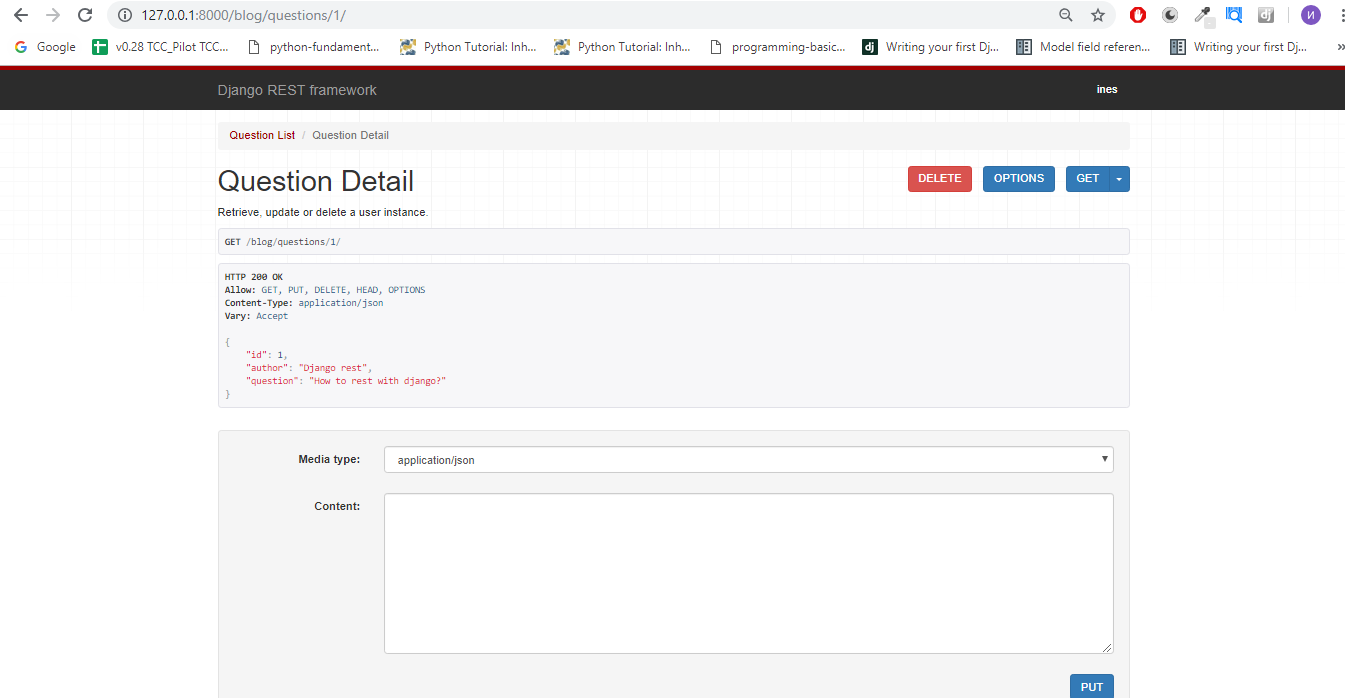
Check the result on the same url, you should have the following:



14. For the delete method is like child’s play – we create def delete, takes self,request and pk, we get question instance through get\_object function, we say question.delete(), and return Response for no content:



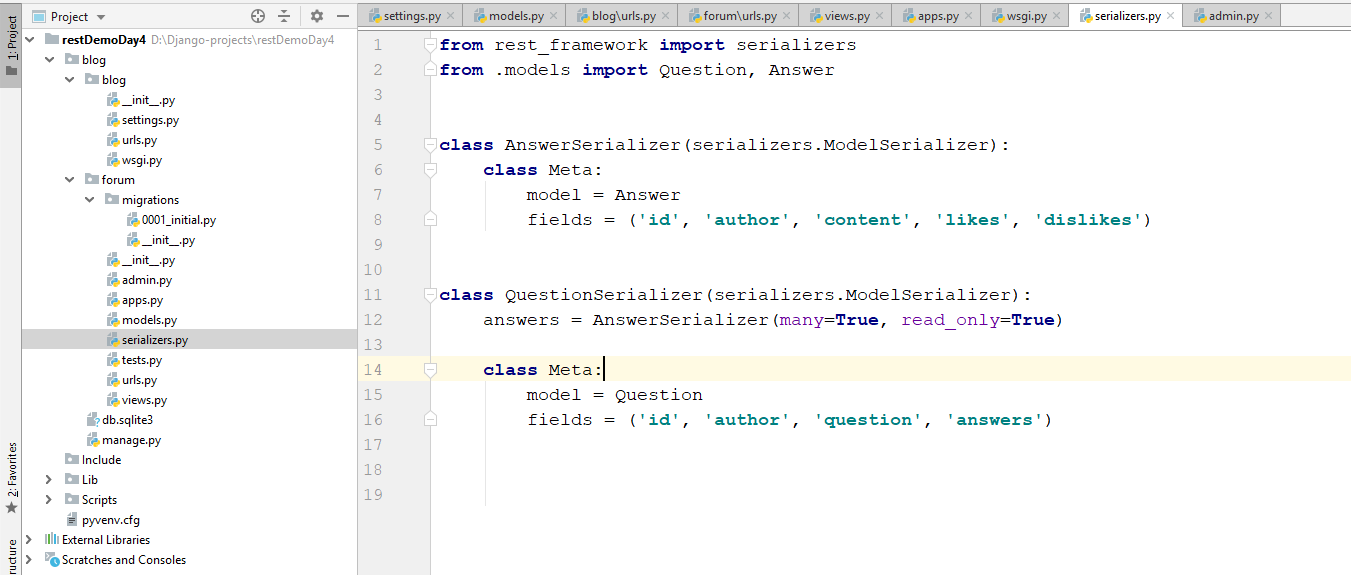
Let’s see the changes:



Did you notice the red delete button? ☺

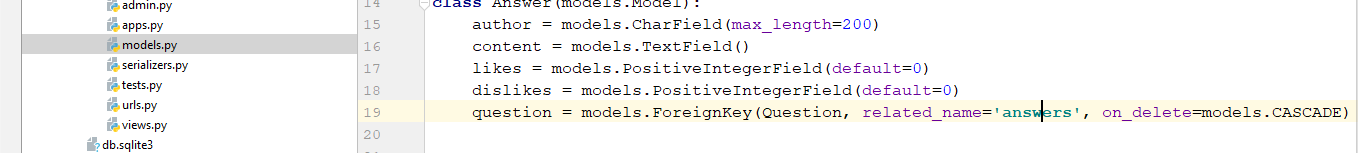
**NESTED SERIALIZERS**

But what about answers? We can not see them inside our question instance? We have to use so called nested serializers. First let change some things in our serializers.py

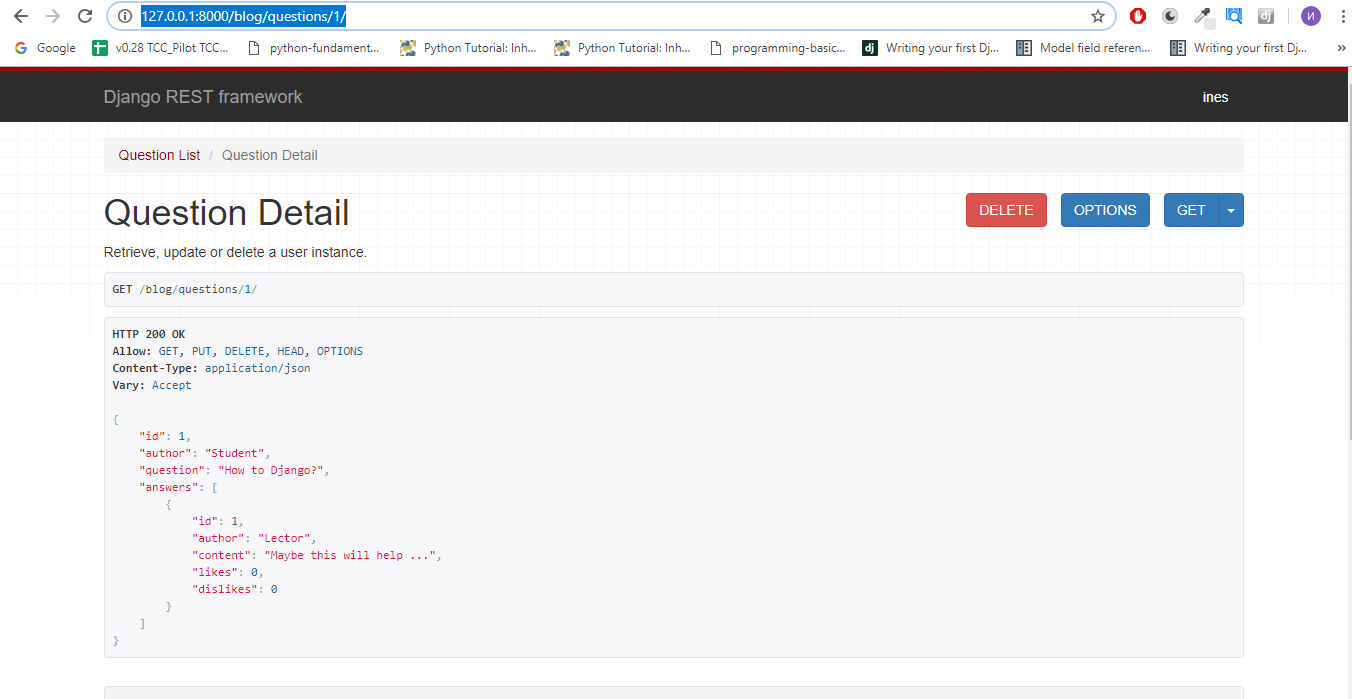


Note how we have to add new serializer which is for Answers. After that we add a field before our Meta class called answers = we take and serialize the answers of the question and after that serialize the questions two – nested serializers.

But that won’t work. Why? Because we haven’t associate this field. To do that go to models.py and add another param to the foreign key called related\_name=’answers’:



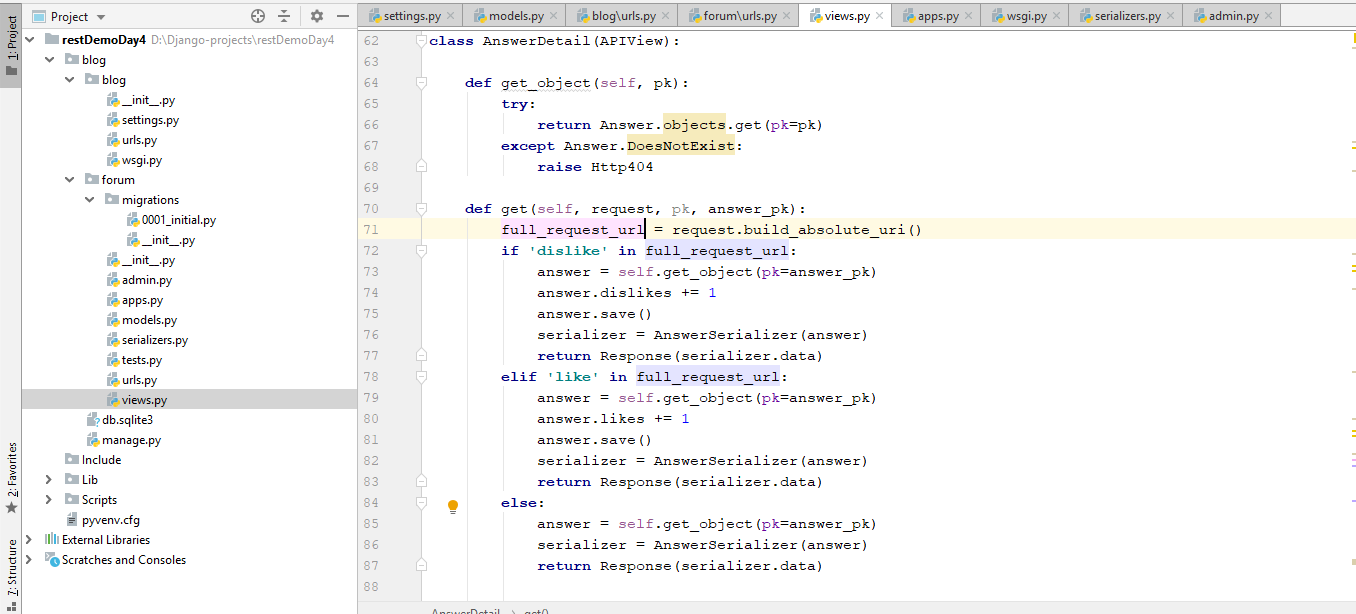
Go to <http://127.0.0.1:8000/blog/questions/1/> and check he results – now we have to be able to see this –

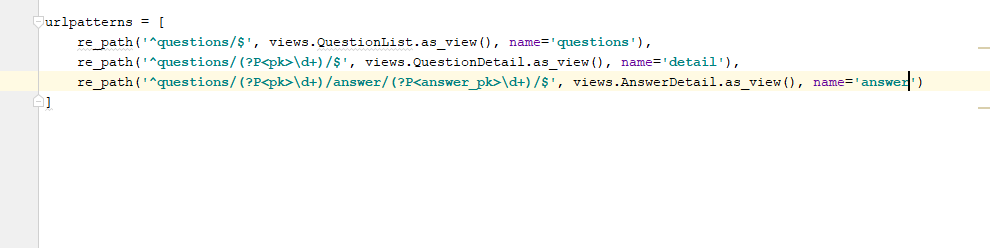


**OPTIONAL – LIKE/DISLIKE**

Do you remember the like and the dislikes fields we have added? Ok, let’s implement like/dislike and get logic logic:

Let’s do it like the previous – create class AnswerDetail which inherits APIView. Implement get\_object. Think of solution in one function – in get function – we will have 3 conditions if like/dislike or just get with simple if and elif statements. You can play with the urls and try to build it as string and check if like/dislike is there…. Below are pictures to help you:





Do to:

<http://127.0.0.1:8000/blog/questions/1/answer/1/?dislike>

<http://127.0.0.1:8000/blog/questions/1/answer/1/?like>

<http://127.0.0.1:8000/blog/questions/1/answer/1/>

And see how data is changing…

Can you think of better implementation of this part? Ok, build it ☺

What functionality is missing here? Try to implement it.