Django REST Framework

Mr. Sushant M. Nair
B. Tech. Comp. Engg. (2025), KJ Somaiya College of Engineering, Mumbai

Introduction

- Till now, we have seen how Django can be used to make efficient web applications.
- One thing to note is how tightly coupled the Frontend and the Backend are.
- Firstly, let’s clarify what we mean by Frontend

Introduction

- Frontend part of an application refers to the visible aspect of the website - the HTML, CSS and JS parts in terms of Django
- Backend part of an application refers to the behind-the-stage aspect of the website - the Python part (views.py, urls.py, models.py, forms.py, settings.py, etc.).

Introduction

- Frontend controls the visual aspects of the webpage. It also determines which elements will be positioned at what places on the page, as well as how different elements interact with the user and with each other.
- Backend controls the data creation, storage and retrieval part. This is what makes websites personal - the data is specific to you. It also stores the general information that everyone gets to see.

- Now coming back to Django where we left off - the Frontend and Backend are tightly coupled. What do we mean by this? Take a look at the following picture:

views.py ...\firstapp × | reward.html | reward2.html | reward3.html | reward4.html

firstproject > firstapp > views.py > monthly_reward

```
11  def monthly_reward(request, month):
27          return render(request, "firstapp/reward4.html", {
28              "text": reward_text, "month_name": month
29          })
30
31
32      except:
33          # return HttpResponseNotFound("<h1>This month is not supported!</h1>")
34          return render(request, "404.html")
35
```

- It can easily be observed that the backend code in views.py is **directly referring to** the frontend code, i.e., the monthly_reward function of views.py is **directly referring to** the reward4.html
- This means that if changes are made to the backend, i.e., some logic changed, some new functionality was added, of anything else, we would have to make sure that the **reference stays intact.**
- Not doing this would result in inconsistencies - the frontend and backend simply won’t work together.

- It also means that it would be hard for the same backend to refer to multiple variations of frontend.
- For instance, consider the example we have been discussing - we have only one frontend which has been built for a Web Interface suited for a laptop.
- Now, what if I want frontend for Mobile Web Interface, or even a Mobile App?
- This necessitates multiple frontends, which can complicate the scenario where the backend is tightly coupled with frontend

- This is where Django REST Framework comes into picture. It decouples the frontend and backend.
- The way this is achieved is through **Application Programming Interface - API's.**
- The backend runs separately and produces data after processing.
- This data is made available on the system through API, in the form of variables.
- The frontend picks up these variables by accessing the API, whose values have been set by the backend, and based on those values renders the pages to be unique for each user.

- With the background in place, we can proceed for practice.
- First, let’s build context.
- Let’s just do things and then later understand the reason. In that way, understanding would be easier.
- The first project we will make is named “rest1” where we will simply be doing things.
- The second project we will make is named “rest2” where we will actually understand how things are working.

- Let’s start our project, then let’s build app.
- First, ensure that your Virtual Environment is activated.
- Then, run the following commands:

```
(djenv) PS C:\gh_repos\Django-Tutorial\Django> django-admin startproject rest1
(djenv) PS C:\gh_repos\Django-Tutorial\Django> cd rest1
(djenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> django-admin startapp restfulapp1
(djenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> pip install djangorestframework
Requirement already satisfied: djangorestframework in d:\college applications\django\djenv\lib\site-packages (3.15.1)
Requirement already satisfied: django>=3.0 in d:\college applications\django\djenv\lib\site-packages (from djangorestframework) (5.0.6)
Requirement already satisfied: asgiref<4,>=3.7.0 in d:\college applications\django\djenv\lib\site-packages (from django>=3.0->djangorestframework) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in d:\college applications\django\djenv\lib\site-packages (from django>=3.0->djangorestframework) (0.5.0)
Requirement already satisfied: tzdata in d:\college applications\django\djenv\lib\site-packages (from django>=3.0->djangorestframework) (2024.1)
```

- Now, add the framework and the app in the settings.py file

settings.py ×

Django > rest1 > rest1 > settings.py > ...

```python


# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "rest_framework",
    "restfulapp1"
]
```

- Next, go to models.py file of the restfulapp1 app and type the following:

settings.py models.py ×

Django > rest1 > restfulapp1 > models.py > BioData > __str__

```python
from django.db import models

# Create your models here.
class BioData(models.Model):
    first_name = models.CharField(max_length=100)
    middle_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    age = models.PositiveIntegerField()
    student = models.BooleanField(default=True)
    description = models.TextField()

    def __str__(self):
        return self.first_name
```

- Next, create serializers.py file in the restfulapp1 app directory and type the following:

settings.py | models.py | serializers.py

Django > rest1 > restfulapp1 > serializers.py > BioDataSerializer > Meta

```python
from rest_framework import serializers #type: ignore
from .models import BioData

class BioDataSerializer(serializers.ModelSerializer):
    class Meta:
        model = BioData
        fields = '__all__'
```

- Next, go to views.py file in the restfulapp1 app directory and type the following:

settings.py | models.py | serializers.py | views.py

Django > rest1 > restfulapp1 > views.py > ...

```python
from rest_framework import generics #type: ignore
from .models import BioData
from .serializers import BioDataSerializer

# Create your views here.
class BioDataListCreate(generics.ListCreateAPIView):
    queryset = BioData.objects.all()
    serializer_class = BioDataSerializer

```

- Keep in mind that the variable names “queryset” and “serializer_class” must be as they are. Don’t invent your own names for them (like “query” instead of “queryset”).

- Next, create urls.py file in the restfulapp1 app directory and type the following:

settings.py | models.py | serializers.py | views.py | urls.py

Django > rest1 > restfulapp1 > urls.py > ...

```python
from django.urls import path
from .views import BioDataListCreate

urlpatterns = [
    path('biodata/', BioDataListCreate.as_view(), name='bio-data-list-create')
]
```

- Keep in mind that the variable name “urlpatterns” must be as it is. Don’t invent your own names for it (like “url” instead of “urlpattern”).

- Now, go to urls.py file in rest1 folder type the following:

settings.py | models.py | serializers.py | views.py | urls.py ...\restfulapp1 | urls.py ...\rest1

Django > rest1 > rest1 > urls.py > ...

```
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path('students-and-alumni/', include('restfulapp1.urls')),
]
```

- As always, don’t change variable names. These names are specifically searched for by Django in whose absence either due to non-declaration or name-change, errors can be thrown.

- Now, enter the commands and run the server:

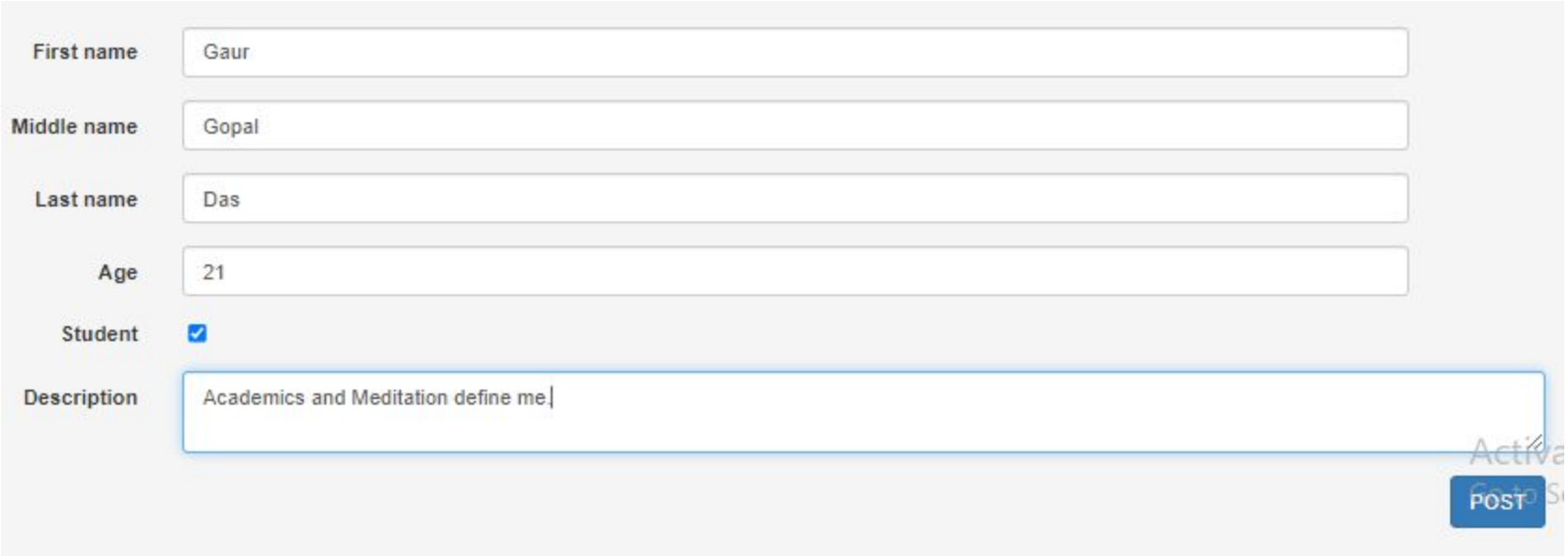
```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> py manage.py makemigrations
Migrations for 'restfulapp1':
  restfulapp1\migrations\0001_initial.py
    - Create model BioData
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, restfulapp1, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying restfulapp1.0001_initial... OK
  Applying sessions.0001_initial... OK
```

- Now, enter the commands and run the server:

```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 31, 2024 - 18:12:57
Django version 5.0.6, using settings 'rest1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

- Note: If you have multiple virtual environments in the same machine, conflict can occur which can result in “Module Not Found - rest_framework” error when running py manage.py makemigrations.
- To resolve this, you should create a new virtual environment, install django and djangorestframework, and run the commands.
- Notice that my virtual environment has changed? This is the reason.

- Enter your details:



- You should see this screen after pressing “Post”:

http://127.0.0.1:8000/students-and-alumni/biodata/

Django REST framework

Bio Data List Create

Bio Data List Create

OPTIONS GET

POST /students-and-alumni/biodata/

```
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "first_name": "Gaur",
    "middle_name": "Gopal",
    "last_name": "Das",
    "age": 21,
    "student": true,
    "description": "Academics and Meditation define me."
}
```

- Notice that this data is in JSON format.

```
{
    "id": 1,
    "first_name": "Gaur",
    "middle_name": "Gopal",
    "last_name": "Das",
    "age": 21,
    "student": true,
    "description": "Academics and Meditation define me."
}
```

- This is Java Script Object Notation.
- This data is generated by the backend and made available via the API
- Now, the frontend code just needs to access this data via the API.
- Once the data is available, the user-specific frontend page can be rendered.

Now let us get to Part 2...

In this part, we will understand exactly what each part does and why

- First, let us create a new project rest2 with a new app restfulapp2 by running the following commands:

```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest1> cd ..
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django> django-admin startproject rest2
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django> cd rest2
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> django-admin startapp restfulapp2
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2>
```

- Also, let us install Postman. It is a desktop application which lets us play around with all types of requests (yes, GET and POST aren't the only ones out there...)

https://www.postman.com/downloads/

Product Pricing Enterprise Resources and Support Public API Network

Contact Sales Sign In Sign Up for Free

Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

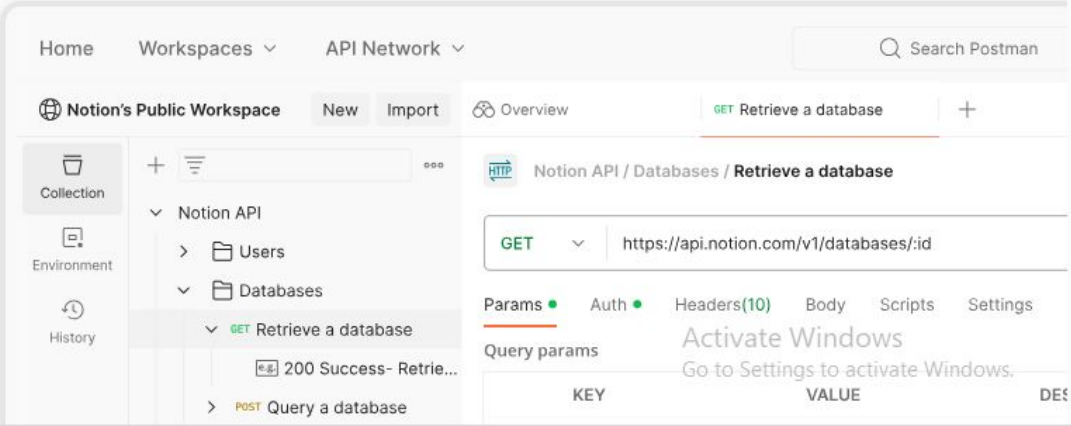
The Postman app

Download the app to get started with the Postman API Platform.

Windows 64-bit

By downloading and using Postman, I agree to the Privacy Policy and Terms.

Release Notes →



- FIRST THINGS FIRST!!!
- Go to settings.py file in rest2 folder

urls.py ...\rest1 | settings.py ...\rest2 | views

Django > rest2 > rest2 > settings.py > ...

```
33  INSTALLED_APPS = [
34      "django.contrib.admin",
35      "django.contrib.auth",
36      "django.contrib.contenttypes",
37      "django.contrib.sessions",
38      "django.contrib.messages",
39      "django.contrib.staticfiles",
40      "rest_framework",
41      "restfulapp2",
42  ]
```

- Add those last two lines there!!!
- Else you will get a heartbreaking error and wonder why on earth only I have to suffer like this!
- At least you'll develop your debugging skills like that ;-)

- Now, let us go to the views.py file of restfulapp2.
- In the code that comes with this document, uncomment the section labelled “Part A”.

views.py ...\restfulapp2 × | Untitled-1 | Untitled-2 | Untitled-3

Django > rest2 > restfulapp2 > views.py > index

```python
from rest_framework.decorators import api_view #type: ignore
from rest_framework.response import Response #type: ignore

# Create your views here.

# PART A

@api_view(['GET'])
def index(request):
    course_name = 'Python Frameworks for the Web'
    course_contents = ['Flask', 'Django', 'Tornado']
    provider = 'ABC University'
    certificate = True
    live = True
    mode = 'Online'
    courses = {
        'course_name' : course_name,
        'course_contents': course_contents,
        'provider': provider,
        'certificate': certificate,
        'live': live,
        'mode': mode
    }
    return Response(courses)
```

DECORATORS

- What is the @api_view thing?
- It is known as a Decorator
- Decorators are used for a wide range of purposes.
- They essentially enforce constraints; certain conditions only under which a function should work.
- So in the provided code, the @api_view(['GET']) decorator enforces the constraint that the function defined below it will work only on the issuance of a GET request.
- Note that it is a list; more request types can be put e.g., @api_view(['GET', 'POST']), etc.

REQUESTS

- We mentioned terms like GET and POST
- These are called Requests
- Requests are calls made by a client (like your computer) to the server during the time the client requests a page from the server.
- They specify the kind of action the client wants to perform on that page in the server.
- GET request means the client is trying to GET page information from the server - read only.

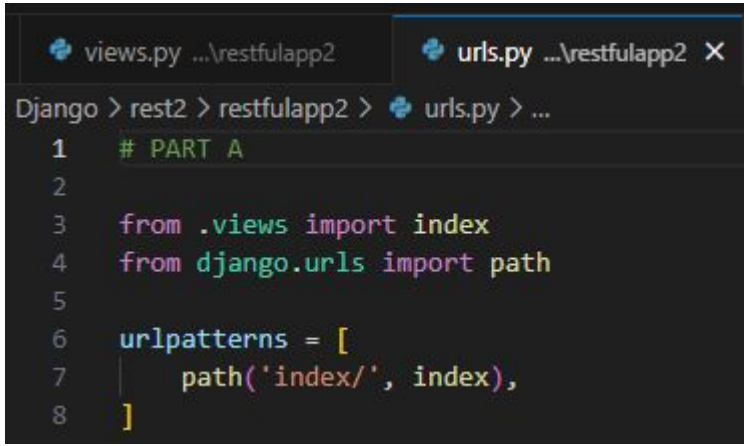
REQUESTS

- POST request means that the client is trying to POST or publish information to the page on the server - a bunch of units of content at a time. This usually happens for the first time.
- PUT request means that the client is trying to PUT information to the page on the server when the client wants to publish a single unit of content at a time. This usually happens for subsequent times.
- PATCH request means that the client is trying to modify one or more field of a particular piece of information, based on its ID.

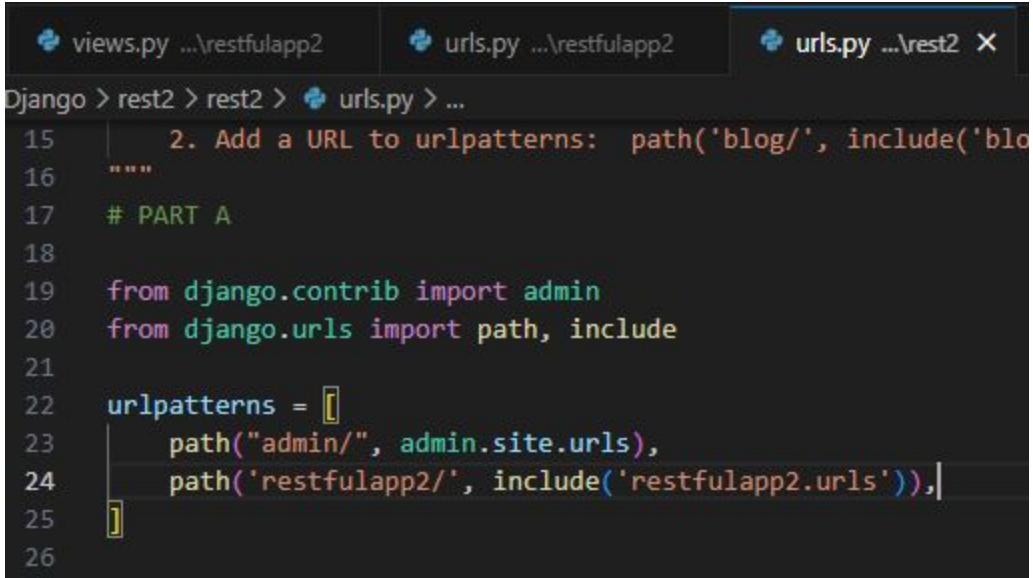
REQUESTS

- DELETE request means that the client is trying to remove a particular piece of information, based on its ID.
- Of course, not all requests are allowed. It depends on the list in @api_view decorator
- Developers need to have a good understanding of the proper situation when a particular request should be allowed, lest the system should suffer from integrity issues.

- Next, go to the restfulapp2 directory and create the urls.py file
- In the supplied code, find the section PART A
- Uncomment the lines of code of that section.



- Next, go to the rest2 directory and go to the urls.py file
- In the supplied code, find the section PART A
- Uncomment the lines of code of that section.



- Next, run the following commands:

```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 31, 2024 - 22:02:24
Django version 5.0.6, using settings 'rest2.settings'
Starting development server at http://127.0.0.1:8000/
```

- If everything goes well, this should be seen:

http://127.0.0.1:8000/restfulapp2/index/

Django REST framework

Index

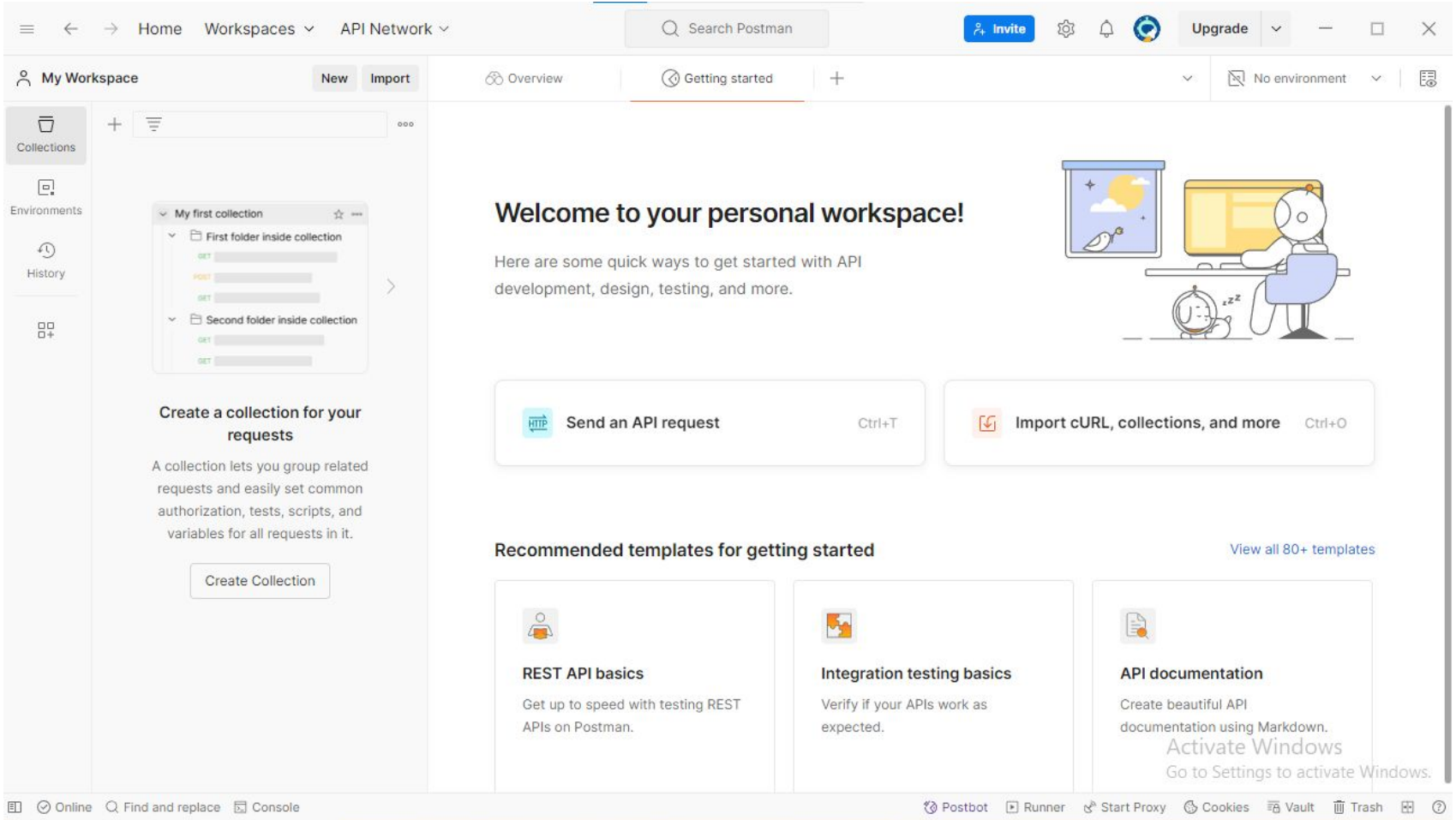
Index

OPTIONS GET

```
GET /restfulapp2/index/
```

```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "course_name": "Python Frameworks for the Web",
    "course_contents": [
        "Flask",
        "Django",
        "Tornado"
    ],
    "provider": "ABC University",
    "certificate": true,
    "live": true,
    "mode": "Online"
}
```

- Now here’s where Postman comes in:



- Click on the ‘+’ symbol and paste the URL of the page just run by the server.
- You’ll get to see the JSON response of the page.
- This is the data which the backend makes available for the frontend to consume.
- Assignment: Experiment with different request types. As shown in the picture in the next slide, the GET request works. But choose some other request from the dropdown menu and watch what happens.

GET http://127.0.0.1:8000/restfulapp2/index/

Params Authorization Headers (6) Body Scripts Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "course_name": "Python Frameworks for the Web",
    "course_contents": [
        "Flask",
        "Django",
        "Tornado"
    ],
    "provider": "ABC University",
    "certificate": true,
    "live": true,
    "mode": "Online"
}
```

Accepting Data from a POST Request

- Go to Postman and enter the URL http://127.0.0.1:8000/restfulapp2/index/
- Next, select POST out of the requests dropdown.
- Next, select the Body tab below the URL bar.
- Now, select the raw option.
- After that, in the same line on the right, select JSON from the dropdown.
- Add content as shown in the next slide.

POST http://127.0.0.1:8000/restfulapp2/index/ Send

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "name": "Govinda",
3     "age": 40
4 }
```

- Now, go to views.py file in the restfulapp2 folder.
- Uncomment PART C
- Notice how data coming from the client is captured

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | serializers.py ...\restfulapp2

Django > rest2 > restfulapp2 > views.py > index

```
60
61  # PART C
62
63
64  @api_view(['GET', 'POST', 'PUT', 'PATCH', 'DELETE'])
65  def index(request):
66      if request.method == 'GET':
67          request_method = 'GET'
68      elif request.method == 'POST':
69          data = request.data
70          print('------------FETCHING RECEIVED DATA------------')
71          print(data)
72          print('----------------------------------------------')
73          request_method = 'POST'
```

- On Sending the request in Postman, you can see the following:

```
System check identified no issues (0 silenced).
May 31, 2024 - 23:11:14
Django version 5.0.6, using settings 'rest2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

------------FETCHING RECEIVED DATA------------
{'name': 'Govinda', 'age': 40}
----------------------------------------------
[31/May/2024 23:13:13] "POST /restfulapp2/index/ HTTP/1.1" 200 193
```

- That was about how to get data from a POST request.
- Now let’s see how to get data from a GET request.
- Go to Postman and in the URL section add ?search=”<text>” at the end.

- Now, go to views.py file in the restfulapp2 folder and uncomment PART D
- Keep in mind the name of the variable - search.

views.py ...\restfulapp2 | models.py ...\restfulapp2 | serializers.py ...\restfulapp2

Django > rest2 > restfulapp2 > views.py > index

```
98   # PART D
99
100
101  @api_view(['GET', 'POST', 'PUT', 'PATCH', 'DELETE'])
102  def index(request):
103      if request.method == 'GET':
104          print('------------FETCHING RECEIVED DATA------------')
105          print(request.GET.get('search'))
106          print('----------------------------------------------')
107          request_method = 'GET'
```

- Now, hit send in Postman.

```
System check identified no issues (0 silenced).
May 31, 2024 - 23:19:02
Django version 5.0.6, using settings 'rest2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

------------FETCHING RECEIVED DATA------------
"Govinda"
----------------------------------------------
[31/May/2024 23:20:24] "GET /restfulapp2/index/?search=%22Govinda%22 HTTP/1.1" 200 192
```

Identifying Request Type

- Now, let us try something more. Go to views.py file of the restfulapp2 folder.
- Uncomment PART B
- Now we are dealing with a way to know which request is being sent.
- This is done using request.method

urls.py ...\rest1 | settings.py ...\rest2 | views.py ...\restfulapp2 | urls.py

Django > rest2 > restfulapp2 > views.py > ...

```python
# PART B

@api_view(['GET', 'POST', 'PUT', 'PATCH', 'DELETE'])
def index(request):
    if request.method == 'GET':
        request_method = 'GET'
    elif request.method == 'POST':
        request_method = 'POST'
    elif request.method == 'PUT':
        request_method = 'PUT'
    elif request.method == 'PATCH':
        request_method = 'PATCH'
    elif request.method == 'DELETE':
        request_method = 'DELETE'
    course_name = 'Python Frameworks for the Web'
    course_contents = ['Flask', 'Django', 'Tornado']
    provider = 'ABC University'
    certificate = True
    live = True
```

- On running the server, you will see this:
- Notice the last line
- Now lets run it on Postman for other requests

```
HTTP 200 OK
Allow: PUT, GET, POST, PATCH, OPTIONS, DELETE
Content-Type: application/json
Vary: Accept

{
    "course_name": "Python Frameworks for the Web",
    "course_contents": [
        "Flask",
        "Django",
        "Tornado"
    ],
    "provider": "ABC University",
    "certificate": true,
    "live": true,
    "mode": "Online",
    "request_method": "GET"
}
```

PUT http://127.0.0.1:8000/restfulapp2/index/ Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	••• Bulk Edit
	Key	Value	Description	

Body Cookies Headers (10) Test Results 200 OK 8 ms 536 B Save as example

Pretty Raw Preview Visualize JSON


```
{
    "course_name": "Python Frameworks for the Web",
    "course_contents": [
        "Flask",
        "Django",
        "Tornado"
    ],
    "provider": "ABC University",
    "certificate": true,
    "live": true,
    "mode": "Online",
    "request_method": "PUT"
}
```

Serializer

- Lets suppose we made a database call to a table named Person.
- The database query would be written in Python format as:

```
Person.objects.all()
```

- This returns a Query Set like:

```
<QuerySet [<Person: ABC>, <Person: PQR>, ..., <Person: XYZ>]>
```

Serializer

- However, that cannot be understood by a Frontend Framework.
- It needs to be converted to JSON format which can be understood by a Frontend Framework, something like the one below:

  [{“id”: 1, “name”: “ABC”}, {“id”: 2, “name”: “PQR”}, ..., {“id”: n, “name”: “XYZ”}]

- This is done by Serializer. It serializes the data.
- In addition it is not a good Security practice to expose QuerySet openly via an API.

- First, let us create a Database Table. For that, go to models.py file in restfulapp2 folder.
- Uncomment the PART A

models.py ...\restfulapp2 × admin.py serializers.py ...\res

Django > rest2 > restfulapp2 > models.py > Person

```python
from django.db import models

# Create your models here.

# PART A


class Person(models.Model):
    first_name = models.CharField(max_length=100)
    middle_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    age = models.PositiveIntegerField()
    student = models.BooleanField()
    # True => student; False => alumni
    description = models.TextField()

```

- Now in the same folder, go to serializers.py file. Note that this file needs to be separately created and does not come by default.
- Uncomment the PART A

serializers.py ...\restfulapp2 × | urls.py ...\restfulapp2 | urls.py ...\rest2 | Untitled-1 | Untitled-2

Django > rest2 > restfulapp2 > serializers.py > ...

```python
from rest_framework import serializers #type: ignore
from .models import Person

# PART A


class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        # The below statement is used to serialize particular columns of the table
        # fields = ['first_name', 'age']
        # The below statament is used to serialize all columns of the table
        fields = '__all__'
        # The below statement is used to serialize all columns except the ones mentioned
        # exclude = ['middle_name', 'last_name']

```

- Now go to views.py file in the restfulapp2 folder.
- Uncomment PART E

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py

Django > rest2 > restfulapp2 > views.py > ...

```python

# PART E


from .models import Person
from .serializers import PersonSerializer

@api_view(['GET', 'POST'])
def PersonAPI(request):
    if request.method == 'GET':
        objects = Person.objects.all()
        serializer = PersonSerializer(objects, many=True)
        return Response(serializer.data)
    else:
        data = request.data
        serializer = PersonSerializer(data = data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors)

```

- Let’s understand that:
- If GET request is issued by the client, i.e., client wants to GET data from the server, readonly, then the if statement becomes True.
- In that case, the objects is fetched from the Person Table. It is of the form

  <QuerySet [<Person: ABC>, <Person: PQR>, ..., <Person: XYZ>]>

- The serializer converts the QuerySet to JSON
- That is then returned as Response.

- If POST request is issued by the client, i.e., client wants to POST data to the server, write operation, then the if statement becomes False.
- In that case, the data is fetched.
- The serializer converts the JSON data to QuerySet (reverse process)
- If the serializer data is valid, the data is saved.
- That is then returned as Response.
- Otherwise, error is returned as Response.

- Now, go to urls.py file in the restfulapp2 folder.
- Uncomment PART B

urls.py ...\restfulapp2 ✕ urls.py ...\rest2

Django > rest2 > restfulapp2 > urls.py > ...

```
13  # PART B
14
15
16  from .views import PersonAPI
17
18  urlpatterns = [
19      path('person/', PersonAPI),
20  ]
21
```

- Now, go to urls.py file in the restfulapp2 folder.
- Uncomment PART B
- Run the makemigrations, migrate and runserver commands to create the Person table.

urls.py ...\restfulapp2 ✕ urls.py ...\rest2

Django > rest2 > restfulapp2 > urls.py > ...

```
13  # PART B
14
15
16  from .views import PersonAPI
17
18  urlpatterns = [
19      path('person/', PersonAPI),
20  ]
21
```

http://127.0.0.1:8000/restfulapp2/person/

Django REST framework

Person Api

Person Api

OPTIONS GET

```
GET /restfulapp2/person/
```

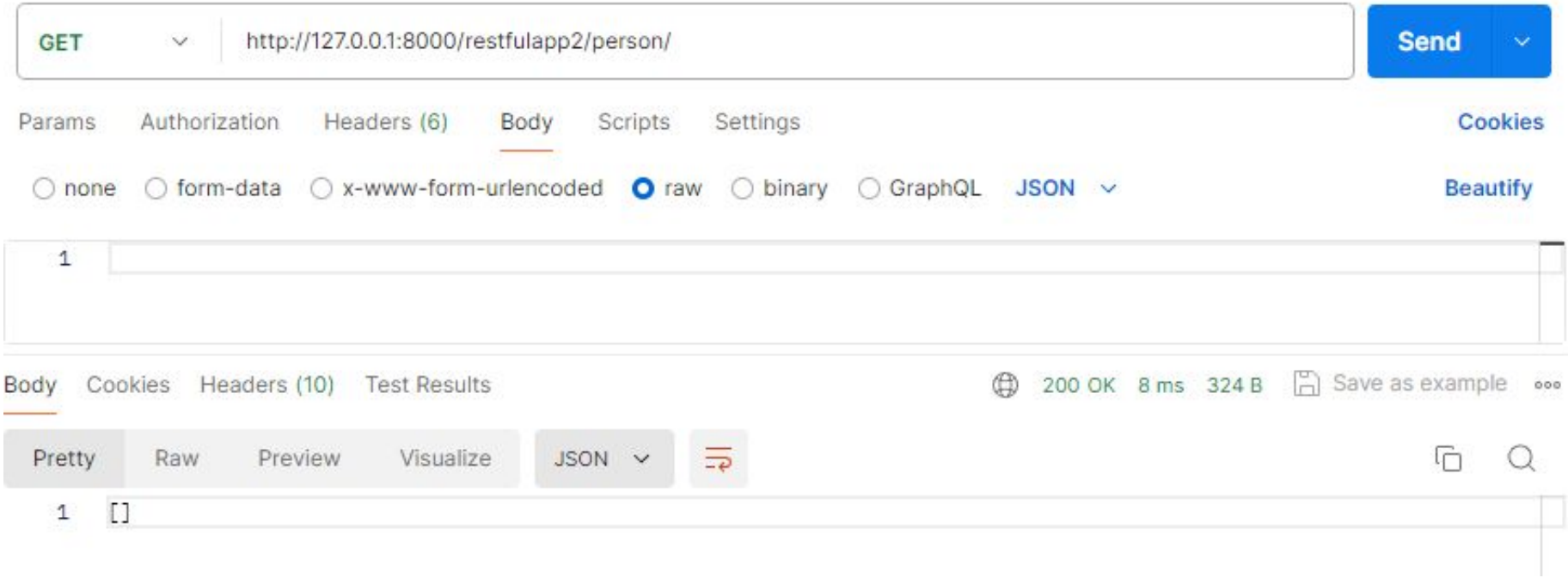
```
HTTP 200 OK
Allow: GET, OPTIONS, POST
Content-Type: application/json
Vary: Accept

[]
```

Media type: application/json

Content:

POST

- Now do as shown in the picture.
- No data comes as no data is in the Table. Lets add some...



- Here’s what happens if you don’t send data via POST…

```
1  {
2  ....
3  }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "first_name": [
3          "This field is required."
4      ],
5      "middle_name": [
6          "This field is required."
7      ],
8      "last_name": [
9          "This field is required."
10     ],
11     "age": [
12         "This field is required."
13     ],
14     "student": [
15         "This field is required."
16     ],
17     "description": [
18         "This field is required."
19     ]
20 }
```

- That’s better…

POST http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

```
1 {
2     "first_name": "Gaur",
3     "middle_name": "Gopal",
4     "last_name": "Das",
5     "age": 40,
6     "student": "False",
7     "description": "Academics and Meditation define me."
8 }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 1,
3     "first_name": "Gaur",
4     "middle_name": "Gopal",
5     "last_name": "Das",
6     "age": 40,
7     "student": false,
8     "description": "Academics and Meditation define me."
9 }
```

- That’s better…

http://127.0.0.1:8000/restfulapp2/person/

Django REST framework

Person Api

Person Api

GET /restfulapp2/person/

```
HTTP 200 OK
Allow: GET, OPTIONS, POST
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    }
]
```

- Now consider what happens when you do this…

models.py ...\restfulapp2 | serializers.py ...\restfulapp2 | urls.py ...\restfulapp2 | urls.py ...\rest2

Django > rest2 > restfulapp2 > serializers.py > PersonSerializer

```python
# PART A

from rest_framework import serializers #type: ignore
from .models import Person

class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        # The below statement is used to serialize particular columns of the table
        # fields = ['first_name', 'age']
        # The below statament is used to serialize all columns of the table
        # fields = '__all__'
        # The below statement is used to serialize all columns except the ones mentioned
        exclude = ['middle_name', 'last_name']
```

- Some fields are no longer required...

POST http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2  ....
3  }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "first_name": [
3          "This field is required."
4      ],
5      "age": [
6          "This field is required."
7      ],
8      "student": [
9          "This field is required."
10     ],
11     "description": [
12         "This field is required."
13     ]
14 }
```

- It is recommended that you create some more records…

Pretty Raw Preview Visualize JSON


```
        "first_name": "Govardhan",
        "middle_name": "DEF",
        "last_name": "GHI",
        "age": 21,
        "student": true,
        "description": "Academics and Power define me."
    },
    {
        "id": 6,
        "first_name": "Gopal",
        "middle_name": "DEF",
        "last_name": "GHI",
        "age": 21,
        "student": true,
        "description": "Academics and Care define me."
    },
    {
        "id": 7,
        "first_name": "Vishnu",
        "middle_name": "DEF",
        "last_name": "GHI",
        "age": 21,
        "student": true,
        "description": "Academics and Influence define me."
    },
    {
        "id": 8,
        "first_name": "Vamana",
```

- Till now we have seen how to GET and POST data.
- Now let's see how to modify data using PATCH.
- Go to views.py file of the restfulapp2 folder
- Uncomment PART F

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | serializers.py ...\restfulapp2 | urls.p

Django > rest2 > restfulapp2 > views.py > PersonAPI

```
161 def PersonAPI(request):
180     else:
181         data = request.data
182         object = Person.objects.get(id = data['id'])
183         serializer = PersonSerializer(object, data = data, partial = True)
184         if serializer.is_valid():
185             serializer.save()
186             return Response(serializer.data)
187         return Response(serializer.errors)
188
```

- STEPS:
- 1: Obtain the request data
- 2: Fetch the QuerySet based on ID of the data
- 3: Serialize it in a manner that updating one or more of the fields in the data is allowed
- 4: If valid, update the particular field and return

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | serializers.py ...\restfulapp2 | urls.p

Django > rest2 > restfulapp2 > views.py > PersonAPI

```
161 def PersonAPI(request):
180     else:
181         data = request.data
182         object = Person.objects.get(id = data['id'])
183         serializer = PersonSerializer(object, data = data, partial = True)
184         if serializer.is_valid():
185             serializer.save()
186             return Response(serializer.data)
187         return Response(serializer.errors)
188
```

- RESULT: Before Operation and After

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

```
1 {
```

ody Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
47    {
48        "id": 6,
49        "first_name": "Gopal",
50        "middle_name": "DEF",
51        "last_name": "GHI",
52        "age": 21,
53        "student": true,
54        "description": "Academics and Care define me."
55    },
```

PATCH http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary Grap

```
1 {
2     "id": 6,
3     "last_name": "Krishnan"
4 }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 6,
3     "first_name": "Gopal",
4     "middle_name": "DEF",
5     "last_name": "Krishnan",
6     "age": 21,
7     "student": true,
8     "description": "Academics and Care define me."
9 }
```

- Now let's look at how to delete a Person's Details
- Go to views.py file in the restfulapp2 folder
- Uncomment PART G

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | serializers.py ...\restfulapp2

Django > rest2 > restfulapp2 > views.py > PersonAPI

```
197  def PersonAPI(request):
224      else:
225          data = request.data
226          object = Person.objects.get(id = data['id'])
227          object.delete()
228          return Response({'message': 'Person Details Removed'})
229
```

- Example…
- This guy is only concerned about Academics and nothing else. So lets remove him… ;-)

#otherthingsmatterinlife

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

```
1 {
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
11    {
12        "id": 2,
13        "first_name": "ABC",
14        "middle_name": "DEF",
15        "last_name": "GHI",
16        "age": 21,
17        "student": true,
18        "description": "Academics define me."
19    },
```

- Result:

DELETE http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Set

none form-data x-www-form-urlencoded raw

```
{
    "id": 2
}
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Person Details Removed"
}
```

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

```
{
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    },
    {
        "id": 3,
        "first_name": "Govinda",
        "middle_name": "DEF",
        "last_name": "GHI",
        "age": 21,
        "student": true,
        "description": "Academics and Peace define me."
    },
```

Validation in Django REST Serializer

- What if a user adds invalid data? Special character in name, underage, etc.?
- Go to serializers.py file in restfulapp2 folder
- Uncomment PART B

serializers.py ...\restfulapp2 | urls.py ...\restfulapp2 | urls.py ...\rest2 | Untitled-1 | Untitled-2 | Untitled-3 | Untitled-4

Django > rest2 > restfulapp2 > serializers.py > PersonSerializer > validate_last_name

```python
class PersonSerializer(serializers.ModelSerializer):

    def validate_age(self, age):
        if age < 18:
            raise serializers.ValidationError('Underage')
        return age

    # notice how some characters which carry special meaning, like doublequotes, are mentioned

    def validate_first_name(self, first_name):
        special_characters = "~`!@#$%^&*()_+-={[}]|:;\"<>,.?/"
        if any(character in special_characters for character in first_name):
            raise serializers.ValidationError(f'First Name {first_name} has special characters which are not allowed.')
```

- Notice the PREFIX validate_
- This is a standard Django REST syntax

validate_<field_name>

- This type of function is specially meant for validation purpose - the system automatically runs the validation check for a variable if that variable's validation function is present

serializers.py ...\restfulapp2 × | urls.py ...\restfulapp2 | urls.py ...\rest2 | Untitled-1 | Untitled-2 | Untitled-3 | Untitled-4

Django > rest2 > restfulapp2 > serializers.py > PersonSerializer > validate_last_name

```
20  class PersonSerializer(serializers.ModelSerializer):
30      def validate_age(self, age):
31          if age < 18:
32              raise serializers.ValidationError('Underage')
33          return age
34
35      # notice how some characters which carry special meaning, like doublequotes, are mentioned
36
37      def validate_first_name(self, first_name):
38          special_characters = "~`!@#$%^&*()_+-={[}]|:;\"<>,.?/"
39          if any(character in special_characters for character in first_name):
40              raise serializers.ValidationError(f'First Name {first_name} has special characters which are not allowed.')
```

- Example

POST http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2      "first_name": "ABCDEF@",
3      "middle_name": "GHIJKL_",
4      "last_name": "MNOPQR+",
5      "age": 12,
6      "student": true,
7      "description": "Academics and Meditation define me."
8  }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "first_name": [
3          "First Name ABCDEF@ has special characters which are not allowed."
4      ],
5      "middle_name": [
6          "First Name GHIJKL_ has special characters which are not allowed."
7      ],
8      "last_name": [
9          "First Name MNOPQR+ has special characters which are not allowed."
10     ],
11     "age": [
12         "Underage"
13     ]
14 }
```

Foreign Key and Depth

- First, go to models.py file of restfulapp2 folder
- Uncomment PART B

models.py ...\restfulapp2 × admin.py serializers.py ...\restfulapp2 urls.py ...\restfulapp2

Django > rest2 > restfulapp2 > models.py > ...

```python
# PART B

class Courses(models.Model):
    course_name = models.CharField(max_length=100)
    course_code = models.CharField(max_length=8)
    certificate = models.BooleanField()
    course_duration = models.TimeField()
    course_domain = models.CharField(max_length=100)

    def __str__(self):
        return self.course_name

class Person(models.Model):
    first_name = models.CharField(max_length=100)
    middle_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    age = models.PositiveIntegerField()
    student = models.BooleanField()
    description = models.TextField()
    course = models.ForeignKey(Courses, on_delete=models.CASCADE,
                               related_name='courses', null=True, blank=True)
```

Foreign Key and Depth

- Now, go to admin.py file of restfulapp2 folder
- Uncomment PART A

admin.py × serializers.py ...\restfulapp2

Django > rest2 > restfulapp2 > admin.py

```
1   from django.contrib import admin
3   # Register your models here.
4
5   # PART A
6
7
8   from .models import Courses
9
10  admin.site.register(Courses)
11
```

Foreign Key and Depth

- Now, run the makemigrations, migrate and runserver commands.

```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py makemigrations
Migrations for 'restfulapp2':
    - Create model Courses
    - Add field course to person
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, restfulapp2, sessions
Running migrations:
  Applying restfulapp2.0004_courses_person_course... OK
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 01, 2024 - 17:48:39
Django version 5.0.6, using settings 'rest2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

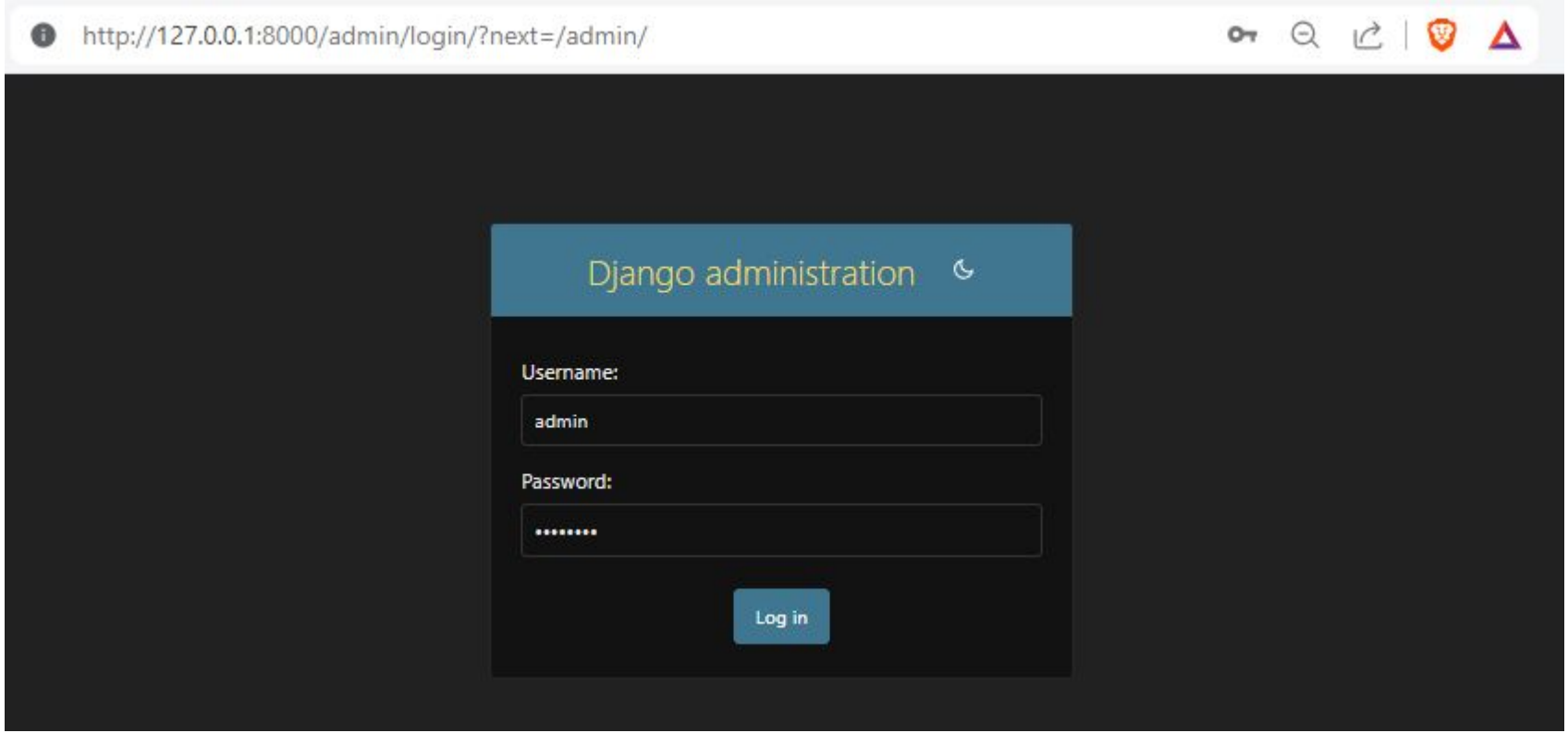
Foreign Key and Depth

- Now, stop the server (CTRL+C) and run the createsuperuser command.

```
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py createsuperuser
Username (leave blank to use 'hp'): admin
Email address: admin@gmail.com
Password:
Password (again):
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(djvirtualenv) PS C:\gh_repos\Django-Tutorial\Django\rest2> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 01, 2024 - 01:45:20
Django version 5.0.6, using settings 'rest2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

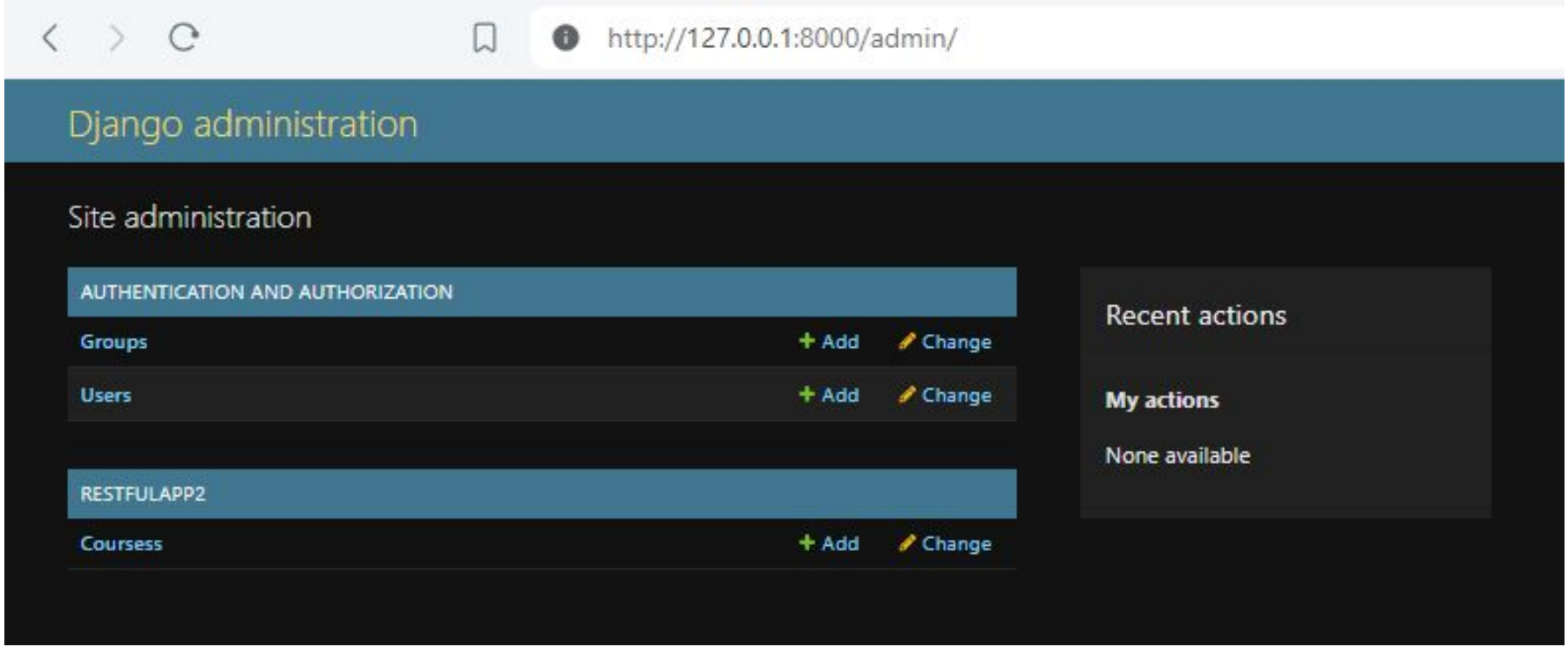
Foreign Key and Depth

- Now, login as administrator



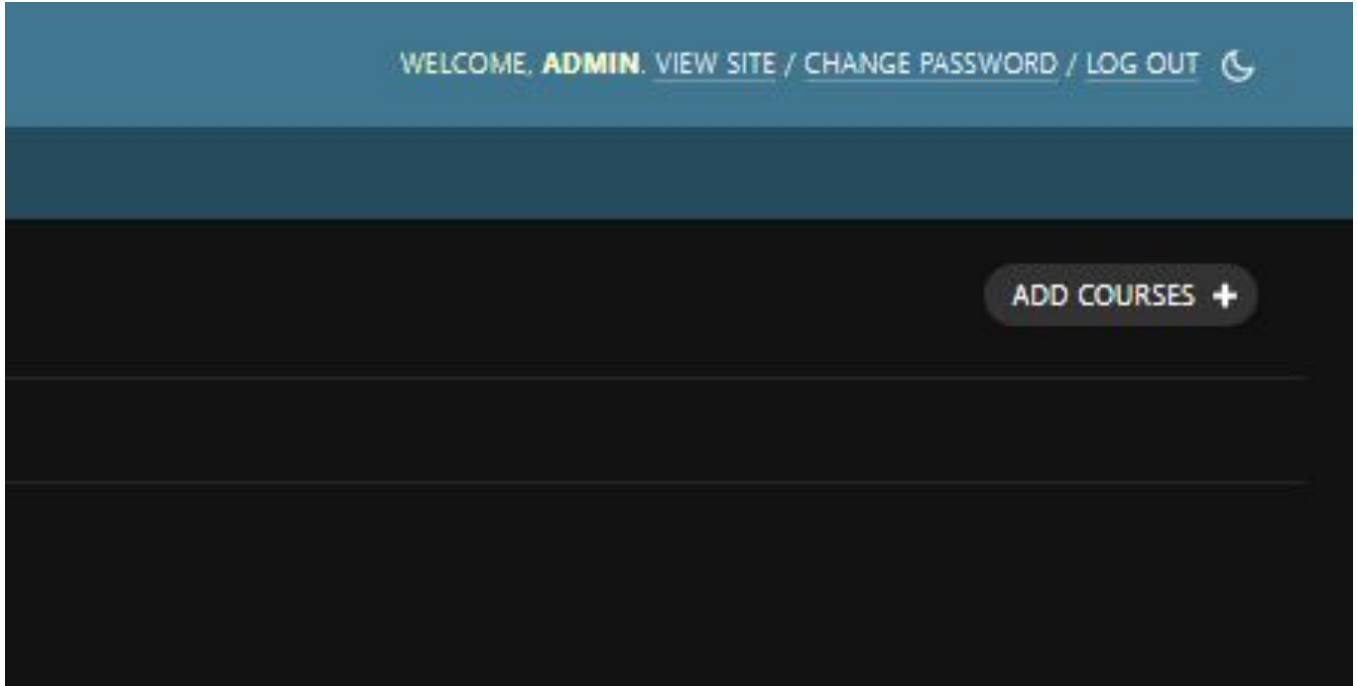
Foreign Key and Depth

- The registered model (Courses) is visible



Foreign Key and Depth

- Click on Courses. No data is present. Let’s create some. Click on the ADD COURSES + button on the right.



Foreign Key and Depth

Add courses

Course name: Python for the Web

Course code: ABCD1234

☑ Certificate

Course duration: 12:22:46 Now | 🕒

Note: You are 5.5 hours ahead of server time.

Course domain: Computer Engineering

SAVE | Save and add another | Save and continue editing

Select courses to change

Action: --------- Go 0 of 5 selected

	COURSES
☐	Applied Machine Learning
☐	Applied Natural Language Processing
☐	Applied Deep Learning
☐	JavaScript for the Web
☐	Python for the Web

5 coursess

Foreign Key and Depth

- Now let us register Person Table so that it also is visible in Django Administration.
- Go to admin.py file in restfulapp2 folder
- Uncomment PART B

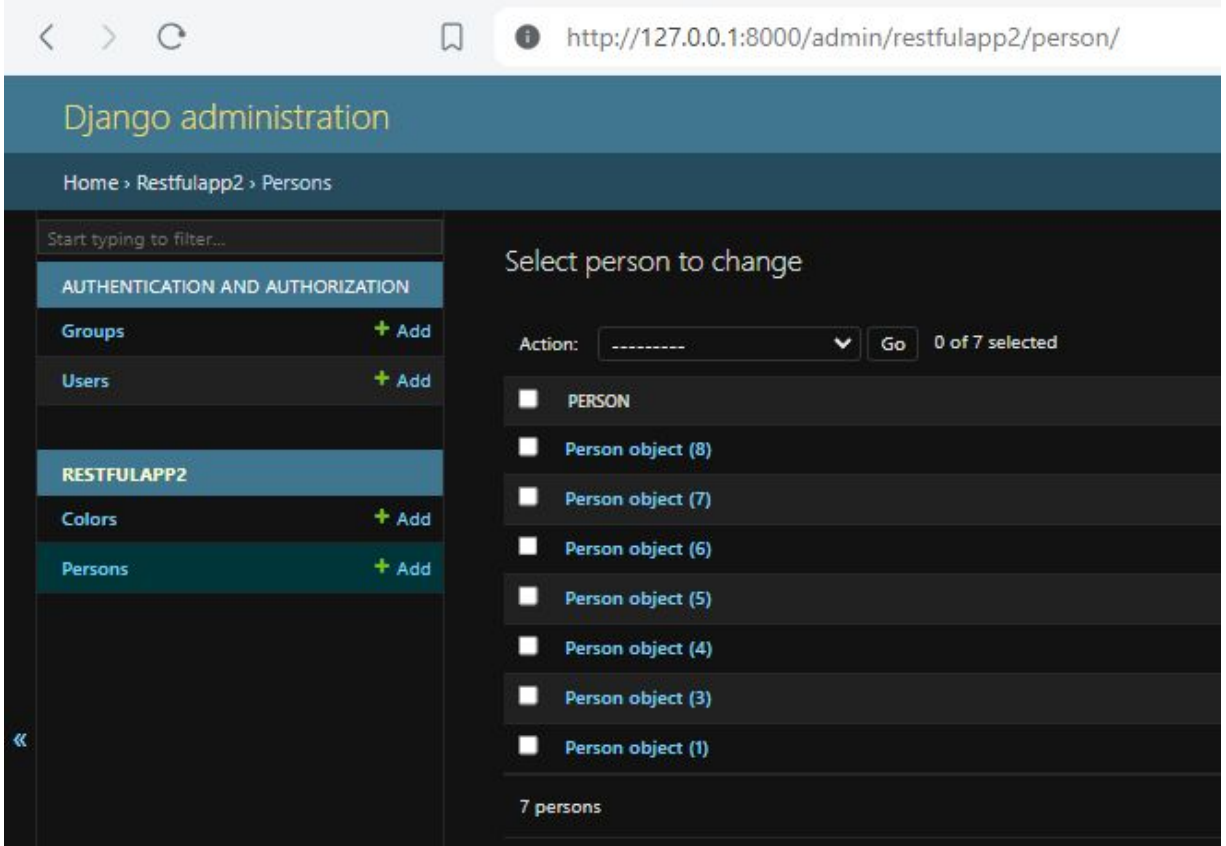
admin.py × serializers.py ...\restfulapp2

Django > rest2 > restfulapp2 > admin.py

```
12
13  # PART B
14
15  from .models import Color
16  from .models import Person
17
18  admin.site.register(Color)
19  admin.site.register(Person)
```

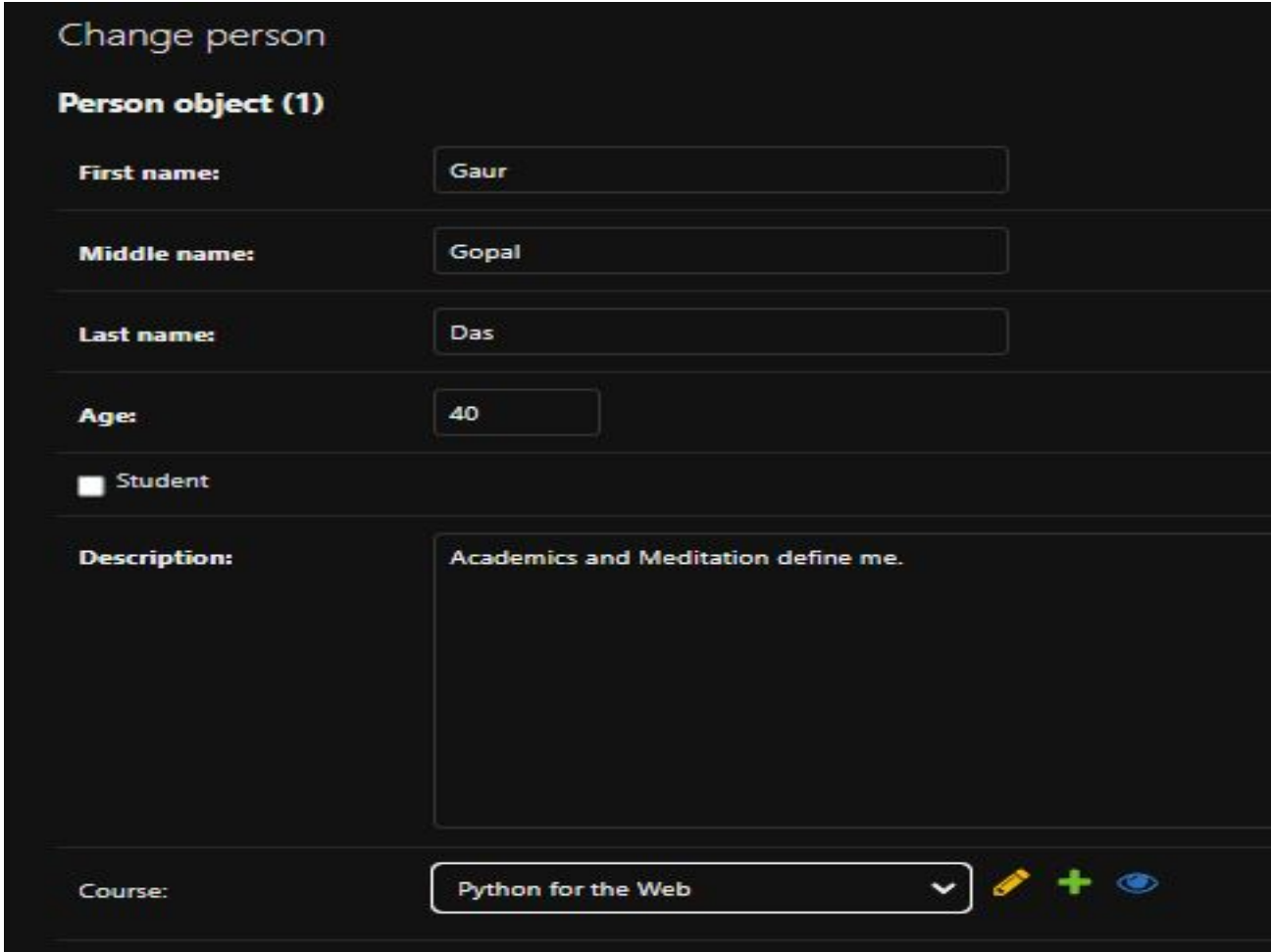
Foreign Key and Depth

- Reload, and click on “Person” which should now appear.



Foreign Key and Depth

- Set course for some people. For showing purposes, I have set course for 5 students out of 7. Click on Person object (n) in order to update the values.



Foreign Key and Depth

- Now go to Postman

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQ

```
1 {
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
1  [
2      {
3          "id": 1,
4          "first_name": "Gaur",
5          "middle_name": "Gopal",
6          "last_name": "Das",
7          "age": 40,
8          "student": false,
9          "description": "Academics and Meditation define me.",
10         "course": 1
11     },
```

Foreign Key and Depth

- It can be observed that the “course” attribute is not giving course name but its ID.
- This is not very understandable.
- Here’s where we can use Depth variable.
- But first, let’s remove those records where there is no information about course, i.e., those students who have not enrolled in any course.

Filtration

- There exist some persons who don’t have a course associated. Let’s filter them out.

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary G

```
1 {
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
32    {
33        "id": 5,
34        "first_name": "Govardhan",
35        "middle_name": "DEF",
36        "last_name": "GHI",
37        "age": 21,
38        "student": true,
39        "description": "Academics and Power define me.",
40        "course": null
41    },
```

Filtration

- Go to views.py file in restfulapp2 folder.
- Uncomment PART H

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py | ser

Django > rest2 > restfulapp2 > views.py > PersonAPI

```
232  # PART H
233
234  from .models import Person
235  from .serializers import PersonSerializer
236
237  @api_view(['GET', 'POST', 'PUT', 'PATCH', 'DELETE'])
238  def PersonAPI(request):
239      if request.method == 'GET':
240          objects = Person.objects.filter(course__isnull = False)
241          serializer = PersonSerializer(objects, many=True)
242          return Response(serializer.data)
```

Filtration

- Now only those records are displayed which have an enrolled course.

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

```
1 {
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
1  [
2      {
3          "id": 1,
4          "first_name": "Gaur",
5          "middle_name": "Gopal",
6          "last_name": "Das",
7          "age": 40,
8          "student": false,
9          "description": "Academics and Meditation define me.",
10         "course": 1
11     },
```

Depth

- Let's get back to depth
- Go to serializers.py file in restfulapp2 folder.
- Uncomment PART C

serializers.py ...\restfulapp2 × | urls.py ...\restfulapp2 | urls.py ...\rest2 | Untitled-1 | Untitled-2

Django > rest2 > restfulapp2 > serializers.py > PersonSerializer > Meta

```python
# PART C

class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        # The below statement is used to serialize particular columns of the table
        # fields = ['first_name', 'age']
        # The below statament is used to serialize all columns of the table
        fields = '__all__'
        # The below statement is used to serialize all columns except the ones mentioned
        # exclude = ['middle_name', 'last_name']
        depth = 1
```

Depth

- Result
- Course name is now visible.
- However, there are multiple other values being displayed
- Can we get only course name?


```
[
    {
        "id": 1,
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me.",
        "course": {
            "id": 1,
            "course_name": "Python for the Web",
            "course_code": "ABCD1234",
            "certificate": true,
            "course_duration": "12:22:46",
            "course_domain": "Computer Engineering"
        }
    },
```

- We can create a Serializer of Course table to resolve this issue.
- Go to serializers.py file in restfulapp2 folder.
- Uncomment PART D

serializers.py ...\restfulapp2 × urls.py ...\restfulapp2 urls.py ...\rest2

Django > rest2 > restfulapp2 > serializers.py > CoursesSerializer > Meta

```python
# PART D

from .models import Courses

class CoursesSerializer(serializers.ModelSerializer):
    class Meta:
        model = Courses
        # To get only course_name and not any other column
        fields = ['course_name']

class PersonSerializer(serializers.ModelSerializer):
    course = CoursesSerializer()
    class Meta:
        model = Person
```

- Much better...

```
1   [
2       {
3           "id": 1,
4           "course": {
5               "course_name": "Python for the Web"
6           },
7           "first_name": "Gaur",
8           "middle_name": "Gopal",
9           "last_name": "Das",
10          "age": 40,
11          "student": false,
12          "description": "Academics and Meditation define me."
13      },
```

- Beware!

```python
class PersonSerializer(serializer
    course = CoursesSerializer()
    class Meta:
        model = Person
```

```
1  [
2      {
3          "id": 1,
4          "course": {
5              "course_name": "Python for the Web"
6          },
7          "first_name": "Gaur",
8          "middle_name": "Gopal",
9          "last_name": "Das",
10         "age": 40,
11         "student": false,
12         "description": "Academics and Meditation define me."
13     },
```

- Observe the variable name “course” in both pictures.
- If in the serializers.py code you put something else instead of “course” like, “c_name”, it will give an error.
- This is because in models.py file, the Person Model is defined with course attribute, and not c_name!
- Please keep this fact in mind. Be careful about names.

Serializer Method Field

- Let us suppose that we need to add one more field in the Person Table - one whose value is common to all Persons.
- One way of doing that is by modifying the Person Model in models.py file, by adding that field (let us take that field as nationality).
- However as you know, after the creation of a Table and the addition of data records, if we want to modify the Table by adding more columns, we have to specify null=True and blank=True.
- Else, integrity issues arise and the system throws error.

Serializer Method Field

- Let us suppose that we specify null=True and blank=True for the new nationality field. What next?
- This would mean that all existing records in Person Table will have nationality=Null. So if I have to set it to, say, Indian, I would have to manually update the Table.
- Plus what if I'm uncertain that this new field would be required in the future? It shouldn't be that I spend time doing something only to find that it was not required.
- Here's where Django REST Framework's Serializer Method Field comes to the rescue.
- It allows us to simply add the column and remove it at will, without touching models.py file!

Serializer Method Field

- Go to serializers.py file in restfulapp2 folder.
- Uncomment PART E

serializers.py ...\restfulapp2 | urls.py ...\restfulapp2 | urls.py ...\rest2 | Untitled-1 | Untitled-2

Django > rest2 > restfulapp2 > serializers.py > PersonSerializer > get_country

```python
class PersonSerializer(serializers.ModelSerializer):
    course = CoursesSerializer()
    nationality = serializers.SerializerMethodField()

    class Meta:
        model = Person
        # The below statement is used to serialize particular columns of the table
        # fields = ['first_name', 'age']
        # The below statament is used to serialize all columns of the table
        fields = '__all__'
        # The below statement is used to serialize all columns except the ones mentioned
        # exclude = ['middle_name', 'last_name']

    def get_country(self):
        return "India"
```

Serializer Method Field

- Notice the lines 148 and 159-160.
- Again, keep in mind about the names.
- Also, notice that we necessarily include the prefix get_ in the format **def get_<field name>(self, object):**

```
146  class PersonSerializer(serializers.ModelSerializer):
147      course = CoursesSerializer()
148      nationality = serializers.SerializerMethodField()
149
150      class Meta:
151          model = Person
152          # The below statement is used to serialize part
153          # fields = ['first_name', 'age']
154          # The below statament is used to serialize all
155          fields = '__all__'
156          # The below statement is used to serialize all
157          # exclude = ['middle_name', 'last_name']
158
159      def get_nationality(self, object):
160          return "India"
```

Serializer Method Field

- Now let us check out the result. Great!

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "course": {
            "course_name": "Python for the Web"
        },
        "nationality": "India",
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    },
```

Serializer Method Field

- Now consider that you want to display additional information about the course a Person is enrolled in.
- This can also be achieved using Serializer Method Field.
- Go to serializers.py file in restfulapp2 folder.
- Uncomment PART F
- Notice the lines 198 and 209-211
- A careful understanding of this part is essential...
- Again be careful about names!
- Notice that part **id = object.course.id**
- Couldn't that variable course be courses? No!
- This is because the Person Model is defined with variable name as **course** and NOT **courses**!

Serializer Method Field

- Whereas, Courses is name of a Model (Courses Model)
- While, we are dealing with Person Serializer.
- However, the data comes from Courses table.
- That's why **Courses.objects.get()**

```
196 class PersonSerializer(serializers.ModelSerializer):
197     course = CoursesSerializer()
198     course_details = serializers.SerializerMethodField()
199 
200     class Meta:
201         model = Person
202         # The below statement is used to serialize particular columns of the table
203         # fields = ['first_name', 'age']
204         # The below statament is used to serialize all columns of the table
205         fields = '__all__'
206         # The below statement is used to serialize all columns except the ones mentioned
207         # exclude = ['middle_name', 'last_name']
208 
209     def get_course_details(self, object):
210         course_object = Courses.objects.get(id = object.course.id)
211         return {'course_domain': course_object.course_domain, 'certificate': course_object.certificate}
```

Serializer Method Field

- Sweet!

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "course": {
            "course_name": "Python for the Web"
        },
        "course_details": {
            "course_domain": "Computer Engineering",
            "certificate": true
        },
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    },
```

Serializer Class

- Till now we have seen Model Serializer
- This is an in-built version of Serializer
- But suppose our requirements are not met by Model Serializer
- You guessed it! Django REST Framework allows us to make our own custom Serializer to suit our needs.
- This method is especially useful when all we need to do is just **validate** the data.
- Take for example, Login.
- We need to verify if the proper person is trying to access an account, i.e., we need to **validate** the data entered by that person, whether the password is correct or not, etc.

Serializer Class

- Go to serializers.py file in restfulapp2 folder.
- Uncomment PART G

serializers.py ...\restfulapp2 × urls.py ...\restfulapp2 urls.

Django > rest2 > restfulapp2 > serializers.py > LoginSerializer

```
# PART G

from .models import Courses

class LoginSerializer(serializers.Serializer):
    email = serializers.EmailField()
    pword = serializers.CharField()
```

Serializer Class

- Go to views.py file in restfulapp2 folder.
- Uncomment PART I

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py

Django > rest2 > restfulapp2 > views.py > login

```
# PART I

from .models import Person
from .serializers import PersonSerializer, LoginSerializer

@api_view(['POST'])
def login(request):
    data = request.data
    serializer = LoginSerializer(data = data)
    if serializer.is_valid():
        data = serializer.validated_data
        return Response({'message': 'Logged In'})
    return Response(serializer.errors)
```

Serializer Class

- Go to urls.py file in restfulapp2 folder.
- Uncomment PART C

urls.py ...\restfulapp2 ✕ urls.py ...\rest2

Django > rest2 > restfulapp2 > urls.py > ...

```
23  # PART C
24
25  from .views import PersonAPI, login
26
27  urlpatterns = [
28      path('person/', PersonAPI),
29      path('login/', login)
30  ]
```

Serializer Class

- Run it on Postman

POST http://127.0.0.1:8000/restfulapp2/login/

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "email": [
3         "This field is required."
4     ],
5     "pword": [
6         "This field is required."
7     ]
8 }
```

Serializer Class

- Now let's add some data. Of course, we have not yet implemented backend verification. This is just to ensure that the email and password is in the right format.

POST http://127.0.0.1:8000/restfulapp2/login/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

```
1 {
2     "email": "gopal.krishnan@gmail.com",
3     "pword": "12345678"
4 }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Logged In"
3 }
```

API View

- It is a class which lets the user customize the logic.
- Consider the current way in which different types of requests are handled in views.py file

```
if request.method == 'GET':
    objects = Person.objects.filter(course__isnull = False)
    serializer = PersonSerializer(objects, many=True)
    return Response(serializer.data)
elif request.method == 'POST':
    data = request.data
    serializer = PersonSerializer(data = data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors)
elif request.method == 'PUT':
```

- There are a lot of if-elif-else statements. We can make this more professional using API View.

API View

- Go to views.py file in restfulapp2 folder.
- Uncomment PART J
- Notice the differences:

1. A new import (327)
2. PersonAPI goes from being a function to being a class (338)
3. if request.method == ‘GET’: becomes def get(self, request):

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py | serializ

Django > rest2 > restfulapp2 > views.py > ...

```
322
323  # PART J
324
325  from .models import Person
326  from .serializers import PersonSerializer, LoginSerializer
327  from rest_framework.views import APIView #type: ignore
328
329  @api_view(['POST'])
330  def login(request):
331      data = request.data
332      serializer = LoginSerializer(data = data)
333      if serializer.is_valid():
334          data = serializer.validated_data
335          return Response({'message': 'Logged In'})
336      return Response(serializer.errors)
337
338  class PersonAPI(APIView):
339      def get(self, request):
340          objects = Person.objects.filter(course__isnull = False)
341          serializer = PersonSerializer(objects, many=True)
342          return Response(serializer.data)
343      def post(self, request):
344          data = request.data
345          serializer = PersonSerializer(data = data)
346          if serializer.is_valid():
347              serializer.save()
348              return Response(serializer.data)
349          return Response(serializer.errors)
```

API View

- Now go to urls.py file in restfulapp2 folder
- Uncomment PART D

urls.py ...\restfulapp2 ✕ urls.py ...\rest2 Untitled

Django > rest2 > restfulapp2 > urls.py > ...

```
34  # PART D
35
36  from .views import PersonAPI, login
37
38  urlpatterns = [
39      path('person/', PersonAPI.as_view()),
40      path('login/', login)
41  ]
```

API View

- Run it on Postman
- It works similar to the way the previous method i.e., API View Decorator worked. But using API View Class has its own advantages.

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "course": {
            "course_name": "Python for the Web"
        },
        "course_details": {
            "course_domain": "Computer Engineering",
            "certificate": true
        },
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    },
```

Model View Set

- Django REST Framework intends to make you focus more on writing your application and less on the inner details.

- With a Model View Set, the nearly 50 lines of code we wrote for the PersonAPI gets reduced to just 3!

- The Model View Set is capable of handling all the CRUD Operations which we just manually wrote.

Model View Set

- Go to views.py file in restfulapp2 folder.
- Uncomment PART K

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py

Django > rest2 > restfulapp2 > views.py > PersonViewSet

```
# PART K

from rest_framework import viewsets # type: ignore
from .serializers import PersonSerializer
from .models import Person

class PersonViewSet(viewsets.ModelViewSet):
    serializer = PersonSerializer
    queryset = Person.objects.all()
```

Model View Set

- CAUTION!!! Names again! DO NOT use “serializer” otherwise the following error appears:

AssertionError at /restfulapp2/person/

'PersonViewSet' should either include a `serializer_class` attribute, or override the `ge

Request Method:	GET
Request URL:	http://127.0.0.1:8000/restfulapp2/person/
Django Version:	5.0.6
Exception Type:	AssertionError
Exception Value:	'PersonViewSet' should either include a `serializer_class` attribute, method.
Exception Location:	C:\gh_repos\Django-Tutorial\Django\djvirtualenv\Lib\site-packages\rest_fra
Raised during:	restfulapp2.views.PersonViewSet

- Use “serializer_class” instead.
- Notice that the error message includes the solution?

views.py ...\restfulapp2 × | models.py ...\restfulapp2 | admin.py

Django > rest2 > restfulapp2 > views.py > PersonViewSet

```
# PART K

from rest_framework import viewsets # type: ignore
from .serializers import PersonSerializer
from .models import Person

class PersonViewSet(viewsets.ModelViewSet):
    serializer_class = PersonSerializer
    queryset = Person.objects.all()
```

Model View Set

- Let’s run and see…

AttributeError at /restfulapp2/person/

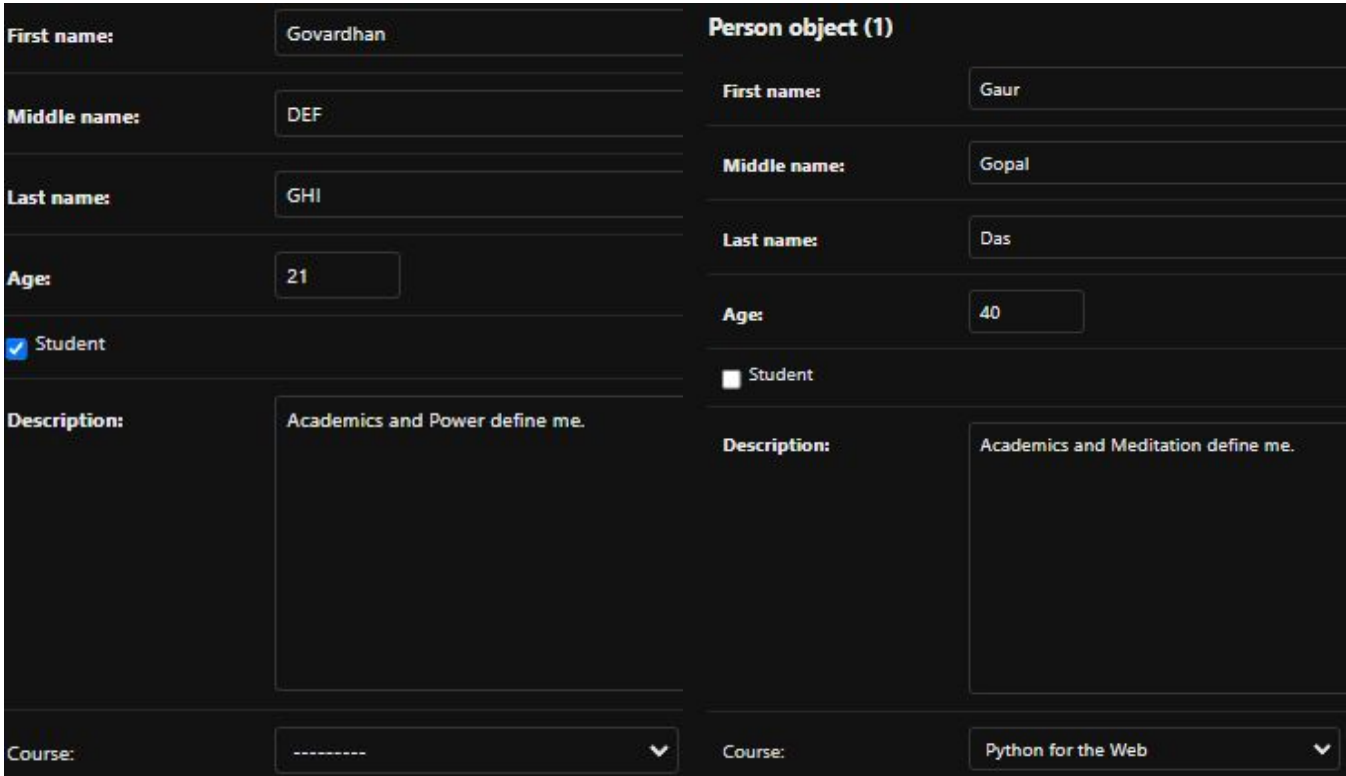
'NoneType' object has no attribute 'id'

Request Method:	GET
Request URL:	http://127.0.0.1:8000/restfulapp2/person/
Django Version:	5.0.6
Exception Type:	AttributeError
Exception Value:	'NoneType' object has no attribute 'id'
Exception Location:	C:\gh_repos\Django-Tutorial\Django\rest2\re
Raised during:	restfulapp2.views.PersonViewSet

- Oops…
- Why is this happening?

Model View Set

- Let’s go to Django Admin and find out.
- Notice that certain persons have a course while others don’t. This is what is causing the exception.



Model View Set

- To see how it is solved, go to serializers.py file in restfulapp2 folder
- Uncomment PART H
- If the course for a Person exists, then if becomes true and it executes. Otherwise, None is returned.
- Also, observe the arrows
- Now, the naming should be understood.

serializers.py ...\restfulapp2 × | urls.py ...\restfulapp2 1 | urls.py ...\rest2

Django > rest2 > restfulapp2 > serializers.py > ...

```
305
306  class PersonSerializer(serializers.ModelSerializer):
307      course = CoursesSerializer()
308      course_details = serializers.SerializerMethodField()
309
310      class Meta:
311          model = Person
312          # The below statement is used to serialize particular co
313          # fields = ['first_name', 'age']
314          # The below statement is used to serialize all columns o
315          fields = '__all__'
316          # The below statement is used to serialize all columns e
317          # exclude = ['middle_name', 'last_name']
318
319      def get_course_details(self, object):
320          if object.course:
321              return {
322                  'course_code': object.course.course_code,
323                  'course_domain': object.course.course_domain,
324              }
325          return None
```

Model View Set

- Now run http://127.0.0.1:8000/restfulapp2/ on Postman.

- As we saw in the urls.py file, the URL is actually automatically generated.

Model View Set

- On clicking that URL ...
- Notice, how elegantly the case is handled where a Person doesn't have a Course.
- Now, you can experiment by using GET, POST, PUT, PATCH and DELETE requests to carry out the CRUD operations.
- And keep in mind, all this was done using just 3 lines of code in the views.py file!

GET http://127.0.0.1:8000/restfulapp2/person/

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "course": {
            "course_name": "Python for the Web"
        },
        "course_details": {
            "course_code": "ABCD1234",
            "course_domain": "Computer Engineering"
        },
        "first_name": "Gaur",
        "middle_name": "Gopal",
        "last_name": "Das",
        "age": 40,
        "student": false,
        "description": "Academics and Meditation define me."
    },
    {
        "id": 3,
        "course": null,
        "course_details": null,
        "first_name": "Govinda",
        "middle_name": "DEF",
        "last_name": "GHI",
        "age": 21,
        "student": true,
        "description": "Academics and Pleasure define me."
    },
```

References

- My sincere gratitude towards Mrs. Nirmala Baloorkar, Assistant Professor, Dept. of Comp. Engg. at K J Somaiya College of Engineering, Vidyavihar, as I got the idea to make a tutorial like this based on her tutorial which I received.
- Also, I am grateful to Mr. Abhijeet of Scaler for the wonderful explanation of REST framework: Getting Started with Django REST Framework | FREE Full Course | SCALER - YouTube
- Also make sure that you go through the official documentation to get in-depth understanding of everything in it: 1 - Serialization - Django REST framework (django-rest-framework.org)

Release Notes

- This is Version 2.0 of Django Tutorial.
- This version saw the addition of REST Framework tutorial to complement the existing Normal Django tutorial.
- The REST Framework tutorial has been implemented in the rest1 and rest2 directories.
- You are encouraged to share this tutorial with anyone, but please exercise kindness by attributing the source whenever sharing this material/using this material in your project.
- You can attribute by pasting the link to the GitHub Repository in the References section of your work, or any equivalent section, wherever appropriate.

Future Scope

- This Version 2.0 of Django Tutorial is an incomplete representation of the Tutorial for Django REST Framework.
- The remaining part of Django REST Framework will be completed and released as part of Version 3.0
- Version 4.0 would contain a similar guide as to creating a Fully Functional Webapp using Normal Django.
- Version 5.0 would contain a similar guide as to creating a Fully Functional Webapp using Django REST Framework.