18/05/2022
Wednesday

REST-API Framework

→ Application Progeamming Inteface - API

→ Tool used in Django is django REST API

→ Representational State Transfer [REST]

→ Jason Famat [In django objects are passed as data but in REST framework data are passed as dictionary]
eg: {'roll': 1, 'name': 'ammu', 'age': 22}

→ Ticket Booking by 3 users in diffeent source like website, phone and laptop but they are booking in same application inteface ie, API

→ In normal django project creation REST framework done.

Steps :-

1. Create a new project named [Sample] Django file
2. django-admin startproject sample
3. cd sample
4. python manage.py startapp app1
5. python manage.py runserver

6. Settings.py → INSTALLED APPS → 'app1',

7. Copy project → urls.py into app → urls.py

8. In project → urls.py → path, include

```
path('', include('app1.urls')),
```

9. views.py →

```
def display(request):
    d={'roll':1, 'name':'ammu'}
    return JsonResponse(d)
```

10. urls.py → from . import views

```
path('', views.display),
```

11. python manage.py runserver

o/p : {"roll": 1, "name":"ammu"}

12. Create table in models.py

```
class student(models.Model):
    roll = models.IntegerField()
    name = models.CharField(max_length=100)
    age = models.IntegerField()
```

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

13. Register the table in admin.py
```
from .models import student
admin.site.register(student)
```

14. python manage.py makemigrations
15. python manage.py migrate
16. python manage.py createsuperuser siva siva
17. Open Admin Dashboard and add the student details into the table student directly in admin page itself.
18. pip install djangorestframework
19. Settings.py → INSTALLED APPS → 'app1', 'rest_framework'

→ Serializers : To convert the object data into dictionary data from database.

20. In app folder create serializers.py file.
21. Serializers.py → from rest_framework import serializers

Siva
1234
} sat batch

```
class sample(serializers.Serializer):
    roll = serializers.IntegerField()
    name = serializers.CharField(max_length=50)
    age = serializers.IntegerField()
```

22. views.py :

```
from .serializers import sample
from .models import student

def display(request):
    if request.method == 'GET':
        d = student.objects.all()
        s = sample(d, many=True)
        return JsonResponse(s.data, safe=False)
```

23. python manage.py runserver

O/P :

```
[{'roll': 1, 'name': 'ammu', 'age': 22},
 {'roll': 2, 'name': 'siva', 'age': 23}]
```

19/5/22 Thursday

To inset data into database without html page

→ Using ModelSerializer and meta class

→ Using postman app - To check the API is applied or not.

1. Serializers.py → from .models import student1
class modelserializer(serializers.ModelSerializer):
    class Meta:
        model = student1
        fields = '__all__'

2. views.py → from .models import student
from .serializers import modelserializer
from rest_framework.parsers import JSONParser
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def display(request):
    if request.method == 'POST' 'GET':
        d = student.objects.all()
        s = modelserializer(d, many = True)
        return JsonResponse(s.data, safe = False)

obj → Dict

String → Dict

```
elif request.method == 'POST':
    d = JSONParser().parse(request)
    s = modelserializer(data=d)
    if s.is_valid():
        s.save()
        return JsonResponse(s.data)
    else:
        return JsonResponse(s.errors)
```

3. python manage.py runserver

4. Copy and paste the url into postman app and add new items into table.

* Update and delete items in table/database

1. views.py

~~Continue with the GET and POST code.~~

```
@csrf_exempt
def update(request, d):
    try:                                  (error handle)
        demo = student.objects.get(pk=d)
    except student.DoesNotExist:
        return HttpResponse('Invalid')

    if request.method == 'GET':
        s = modelserializer(demo)
        return JsonResponse(s.data)

    elif request.method == 'PUT':
        d = JSONParser().parse(request)
        s = modelserializer(demo, data=d)
        if s.is_valid():
            s.save()
            return JsonResponse(s.data)
        else:
            return JsonResponse(s.errors)
```

set primary key(pk)

Search pk to update, delete

Update code

Delete code

```
elif request.method == 'DELETE':
    demo.delete()
    return HttpResponse('Deleted')
```

2. urls.py –

```
path('update/<int:id>', views.update)
```

To search id so we need to act as variable id.

3. python manage.py runserver

While running : http://127.0.0.8000/update/1

Here '1' is the id = 1, In above while updating first search the primary key (pk) then update it and while deleting also first search the pk (id) and then delete it.

4. Open Postman app and do balanceprocess.

20/5/22
Friday

Default Template design & ~~Insert Data~~ Using ~~Postman~~ HTTP Methods

→ Using 'api_views' decorators

→ In REST no html page used bcz default template will be displayed in api_view()

1. views.py

```
from .models import student
from .serializers import modelserializer
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response
@api_view(['GET', 'POST'])
def display(request):
    if request.method == 'GET':
        demo = student.objects.all()
        s = modelserializer(demo, many = True)
        return Response(s.data)
    elif request.method == 'POST':
        s = modelserializer(data = request.data)
        if s.is_valid():
            s.save()
```

```
            return Response(s.data, status =
                                status.HTTP_201_CREATED)
        else:
            return Response(s.errors, status =
                                status.HTTP_400_BAD_REQUEST)
```

→ Update, Search and Delete

```
@api_view(['GET', 'PUT', 'DELETE'])
def update(request, d):
    try:
        demo= student.objects.get(pk=d)
    except student.DoesNotExist:
        return Response(status=status.
                            HTTP_404_NOT_FOUND)
    if request.method=='GET':
        s=model.serializer(demo)
        return Response(s.data)
    elif request.method=='PUT':
        s=modelserializer(demo, data=request.
                                        data)
        if s.is_valid():
            s.save()
```

```
        return Response(s.data)
    else:
        return Response(s.errors, status=
                status.HTTP_400_BAD_REQUEST)
elif request.method=='DELETE':
    demo.delete()
    return Response(status=status.
                HTTP_204_NO_CONTENT)
```

23/05/22
Monday

→ Class Based APIView

1. views.py

```
from rest_framework.views import APIView

class display(APIView):
    def get(self, request):
        demo = student.objects.all()
        s = modelserializer(demo, many=True)
        return Response(s.data)
    def post(self, request):
        s = model.serializer(data=request.data)
```

```
        if s.is_valid():
            s.save()
            return Response(s.data, status=
                status.HTTP_201_CREATED)

        else:
            return Response(s.errors, status=
                status.HTTP_400_BAD_REQUEST)


class update(APIView):
    def get_object(self, d):
        try:
            return student.objects.get(pk=d)
        except student.DoesNotExist:
            return Response(status=status.HTTP_
                404_NOT_FOUND)

    def get(self, request, d):
        demo= self.get_object(d)
        s= modelserializer(demo)
        return Response(s.data)
```

```
def put(self, request, id):
    demo = self.get_object(id)
    s = modelserializer(demo, data =
                        request.data)
    if s.is_valid():
        s.save()
        return Response(s.data)
    return Response(s.errors, status = status.
                    HTTP_400_BAD_REQUEST)

def delete(self, request, id):
    demo = self.get_object(id)
    demo.delete()
    return Response(status = status.
                    HTTP_204_NO_CONTENT)
```

2. urls.py

```
path('', views.display.as_view()),
path('update/<int:id>', views.update.
                        as_view()),
```

24/05/22
Tuesday

Generic Views And ~~Mixins~~ Mixins

→ Django generic views, were developed as a shortcut for common usage patterns.

→ The generic views provided by rest framework allow you to quickly build API views that map closely to your database models.

1. Views.py

```
from rest_framework import generics, mixin

class genericapiview(generics.GenericAPIView,
                     mixins.ListModelMixin,
                     mixins.CreateModelMixin):
    serializer_class = model serializer
    queryset = student.objects.all()
```

```
    def get(self, request):
        return self.list(request)

    def post(self, request):
        return self.create(request)


class update(generics.GenericAPIView,
             mixins.UpdateModelMixin,
             mixins.RetrieveModelMixin,
             mixins.DestroyModelMixin):
    serializer_class = model_serializer
    queryset = student.objects.all()
    lookup_field = 'id'
    def get(self, request, id=None):
        return self.retrieve(request)
    def put(self, request, id=None):
        return self.update(request, id)
    def delete(self, request, id):
        return self.destroy(request, id)
```

-> Authentication

1. views.py

```
from rest_framework.authentication import
        SessionAuthentication, BasicAuthentication
from rest_framework.permissions import
        IsAuthenticated

class genericapiview(generics.GenericAPIView,
    mixins.ListModelMixin, mixins.CreateModelMixin):
    serializer_class = modelserializer
    queryset = student.objects.all()
    lookup_field = 'id'
    authentication_classes = [SessionAuthentication,
                              BasicAuthentication]
    permission_classes = [IsAuthenticated]
```

```
def get(self, request):
    return self.list(request)
def post(self, request):
    return self.create(request)
```

25/05/22