

## Topic: 9. CRUD Operations using Mixins

Speaker: Personal / Notebook: API Development using Django Framework



Mixins are a way to allow the reusability of methods. In object-oriented programming, mixin is a class with methods from other classes.

With mixins, we can reuse certain CRUD operations. These are the following:

`LISTMODEL MIXIN` - method used is `list()`

`CREATMODEL MIXIN` - method used is `create()`

`RETRIEVEMODEL MIXIN` - method used is `retrieve()`

`UPDATEMODEL MIXIN` - method used is `update()`

`DESTROYMODEL MIXIN` - method used is `destroy()`

These are used with `GenericAPIView`. See the documentation [here](#).

1. To view all the records of Employees using Mixins, we update the `API/VIEWS.PY` and comment out the classes `EMPLOYEE` AND `EMPLOYEE DETAIL`.

2. Import the needed library.

```
File Edit Selection ... DjangoREST_APIProject  
EXPLORER  
  DJANGOREST_APIPROJECT  
    api  
      > __pycache__  
      > migrations  
      __init__.py  
      admin.py  
      apps.py  
      models.py  
      serializers.py  
      tests.py  
      urls.py  
      views.py  
    django_rest_main  
      > __pycache__  
urls.py django_rest_main  
urls.py api  
views.py api X  
admin.py employ  
api > views.py > Employees > post  
1  # from django.shortcuts import render  
2  # from django.http import JsonResponse  
3  from students.models import Student  
4  from employees.models import Employee  
5  
6  from .serializers import StudentSerializer, EmployeeSerializer  
7  from rest_framework.response import Response  
8  from rest_framework import status  
9  from rest_framework.decorators import api_view  
10 from rest_framework.views import APIView  
11 from django.http import Http404  
12  
13 from rest_framework import mixins, generics  
14  
15
```

3. Create the new classes and test the URL.

The screenshot shows the VS Code interface with the Django REST API project. The Explorer on the left shows the project structure, with 'views.py' highlighted under the 'api' directory. The main editor displays the code in 'views.py', which is highlighted with a red box. The code defines two API view classes: 'Employees' and 'EmployeeDetail'.

```
112 # uses Mixins for CRUD
113 class Employees(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):
114     queryset = Employee.objects.all()
115     serializer_class = EmployeeSerializer
116
117     def get(self, request):
118         return self.list(request)
119
120     def post(self, request):
121         return self.create(request)
122
123 class EmployeeDetail(generics.GenericAPIView):
124     pass
125
```

4. Testing the URL: <http://127.0.0.1:8000/api/v1/employees/>

This will result to having a form where we can input employee details and when submitted, this will be added to our Employee model.

The screenshot shows the Django REST framework API browser. The address bar shows the URL [127.0.0.1:8000/api/v1/employees/](http://127.0.0.1:8000/api/v1/employees/). The response is a JSON array of three employee objects, highlighted with a red box. Below the response, there is a form to add a new employee, also highlighted with a red box. The form has fields for 'Emp id', 'Emp name', and 'Designation', and a 'POST' button.

Employees

GET /api/v1/employees/

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

```
[
  {
    "id": 1,
    "emp_id": "EMP001",
    "emp_name": "Rosilie Lim",
    "designation": "Software Developer"
  },
  {
    "id": 2,
    "emp_id": "EMP002",
    "emp_name": "Jane Doe",
    "designation": "Web Designer"
  },
  {
    "id": 3,
    "emp_id": "EMP003",
    "emp_name": "Russell Lim",
    "designation": "Security"
  }
]
```

Raw data HTML form

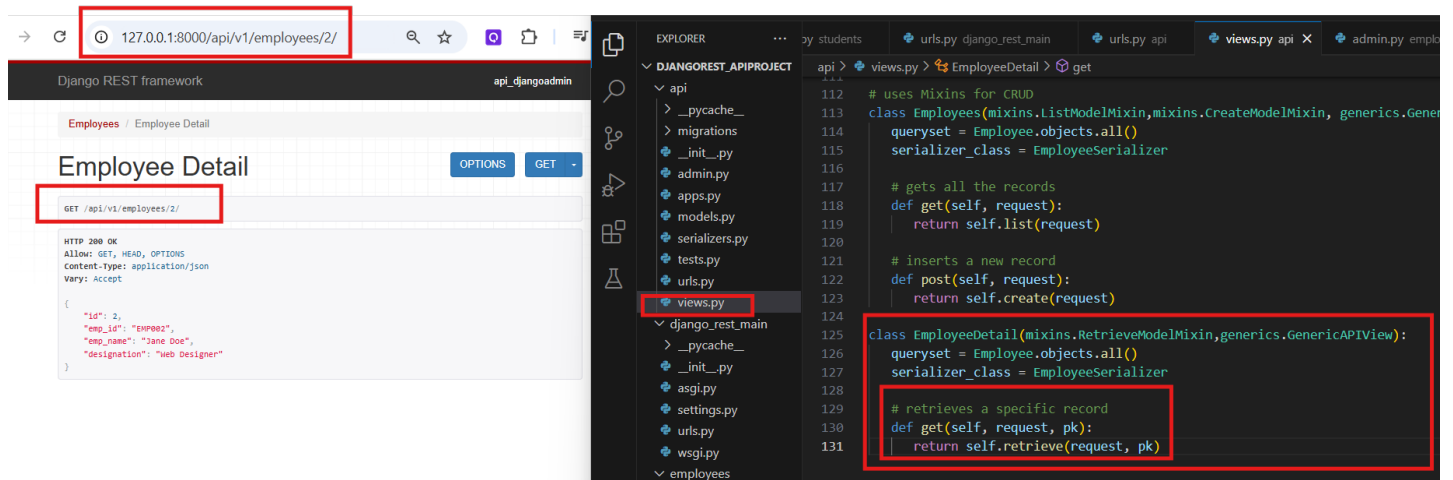
Emp id

Emp name

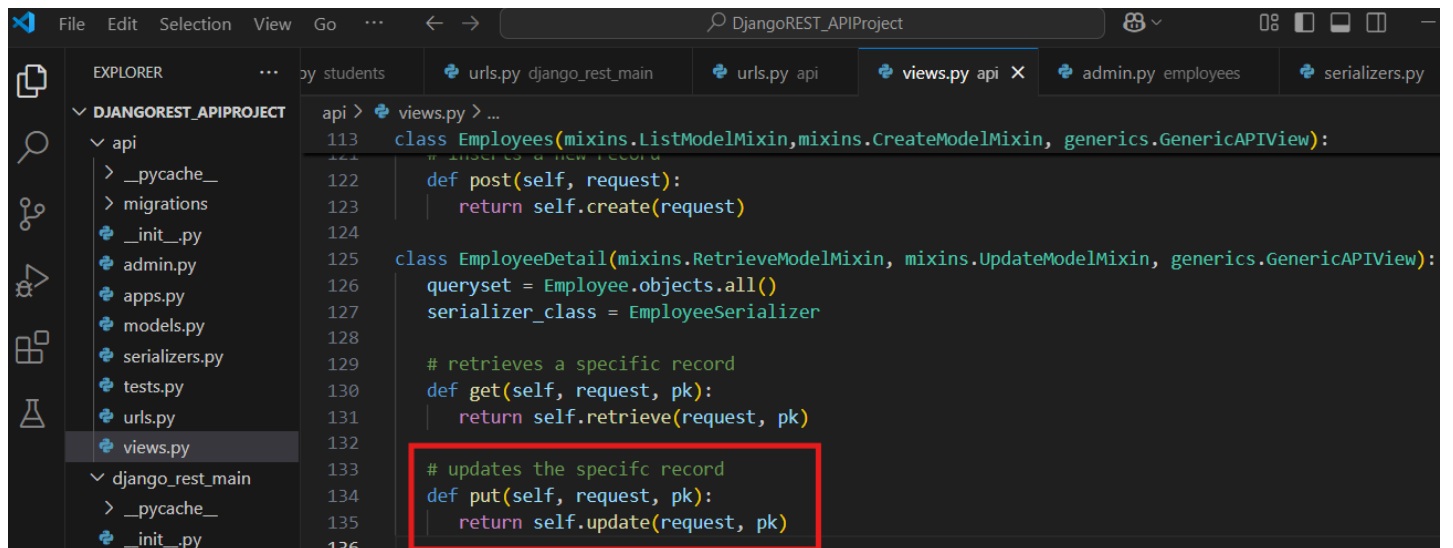
Designation

POST

5. For single - record Read/Update/Delete operations, we update our APIAPPS.PY as :



6. To update a specific record, we use the UPDATEMODEL MIXIN.



To test:

Django REST framework

api\_djangoadmin

Employees / Employee Detail

Employee Detail

OPTIONSGET

GET /api/v1/employees/2/

HTTP 200 OK

Allow: GET, PUT, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "id": 2,
  "emp_id": "EMP002",
  "emp_name": "Jane Doe",
  "designation": "Web Designer"
}
```

Raw dataHTML form

Emp idEMP002

Emp nameJanet Doe

DesignationWeb Designer

PUT

This will update the record to:

Employees / Employee Detail

# Employee Detail

OPTIONS

GET ▾

PUT /api/v1/employees/2/

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

{
  "id": 2,
  "emp_id": "EMP002",
  "emp_name": "Janet Doe",
  "designation": "Web Designer"
}
```

Raw data

HTML form

Emp id

EMP002

Emp name

Janet Doe

Designation

Web Designer

PUT

7. To delete a specific record.

```
111
112 # uses Mixins for CRUD
113 class Employees(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):
114     queryset = Employee.objects.all()
115     serializer_class = EmployeeSerializer
116
117     # gets all the records
118     def get(self, request):
119         return self.list(request)
120
121     # inserts a new record
122     def post(self, request):
123         return self.create(request)
124
125 class EmployeeDetail(mixins.RetrieveModelMixin, mixins.UpdateModelMixin, mixins.DestroyModelMixin, generics.GenericAPIView):
126     queryset = Employee.objects.all()
127     serializer_class = EmployeeSerializer
128
129     # retrieves a specific record
130     def get(self, request, pk):
131         return self.retrieve(request, pk)
132
133     # updates the specific record
134     def put(self, request, pk):
135         return self.update(request, pk)
136
137     # deletes/destroys a specific record
138     def delete(self, request, pk):
139         return self.destroy(request, pk)
```

8. To test and delete record 2,

→ 127.0.0.1:8000/api/v1/employees/2/



Django REST framework

api\_djangoadmin

Employees / Employee Detail

## Employee Detail

DELETE

OPTIONS

GET



GET /api/v1/employees/2/

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

```
{
  "id": 2,
  "emp_id": "EMP002",
  "emp_name": "Janet Doe",
  "designation": "Web Designer"
}
```

Raw data

HTML form

Emp id

EMP002

Emp name

Janet Doe

Designation

Web Designer

PUT

Employees / Employee Detail

# Employee Detail

DELETEOPTIONSGET

GET /api/v1/employees/2/

HTTP 404 Not Found  
Allow: GET, PUT, DELETE, HEAD, OPTIONS  
Content-Type: application/json  
Vary: Accept  

```
{  "detail": "No Employee matches the given query."}
```

Raw dataHTML form

Emp id

Emp name

Designation

PUT