

## 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:

LISTMODELMIXIN - method used is list()

CREATEMODELMIXIN - method used is create()

RETRIEVEMODELMIXIN - method used is retrieve()

UPDATEMODELMIXIN - method used is update()

DESTROYMODELMIXIN - 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 APIVIEWS.PY and comment out the classes EMPLOYEE AND EMPLOYEE DETAIL.
2. Import the needed library.

```
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.

```
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.

127.0.0.1:8000/api/v1/employees/

Django REST framework api\_djangoadmin

### Employees

OPTIONS GET

GET /api/v1/employees/

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

```
[
  {
    "id": 1,
    "emp_id": "ENP001",
    "emp_name": "Rosilie Lim",
    "designation": "Software Developer"
  },
  {
    "id": 2,
    "emp_id": "ENP002",
    "emp_name": "Jane Doe",
    "designation": "Web Designer"
  },
  {
    "id": 3,
    "emp_id": "ENP003",
    "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 :

The screenshot shows a web browser on the left and a code editor on the right. The browser displays the 'Employee Detail' page for the endpoint `GET /api/v1/employees/2/`. The response is a JSON object: `{ "id": 2, "emp_id": "EMP002", "emp_name": "Jane Doe", "designation": "Web Designer" }`. The code editor shows the implementation in `views.py`. The `EmployeeDetail` class inherits from `RetrieveModelMixin` and `GenericAPIView`. It has a `get` method that calls `self.retrieve(request, pk)`. The `views.py` file is highlighted in the explorer.

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

The screenshot shows a code editor with the `views.py` file open. The `EmployeeDetail` class now inherits from `UpdateModelMixin` in addition to `RetrieveModelMixin`. A new `put` method is added, which calls `self.update(request, pk)`. The `put` method implementation is highlighted with a red box.

To test:

Employees / Employee Detail

# Employee Detail

OPTIONS

GET

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 data

HTML form

Emp id

EMP002

Emp name

Janet Doe

Designation

Web 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	<input type="text" value="EMP002"/>
Emp name	<input type="text" value="Janet Doe"/>
Designation	<input type="text" value="Web Designer"/>

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,

# 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

DELETE

OPTIONS

GET

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 data

HTML form

Emp id

Emp name

Designation

PUT