

## 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()

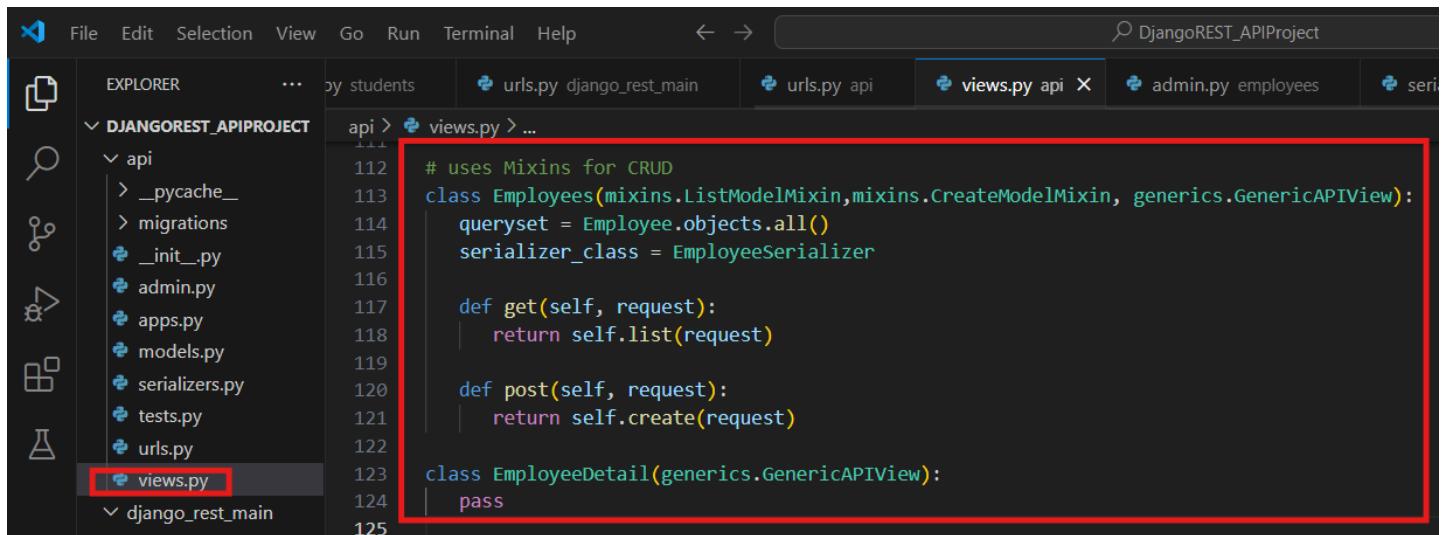
These are used with GenericAPIView. See the documentation [here](#).

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

2. Import the needed library.

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows the 'EXPLORER' view with the 'DJANGOREST\_APIPROJECT' folder expanded, showing subfolders like 'api', 'admin.py', 'apps.py', 'models.py', 'serializers.py', 'tests.py', and 'urls.py'. The 'views.py' file is currently selected and highlighted with a red box. The main code editor shows the following Python code for the 'Employees' view: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.



```

# uses Mixins for CRUD
class Employees(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer

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

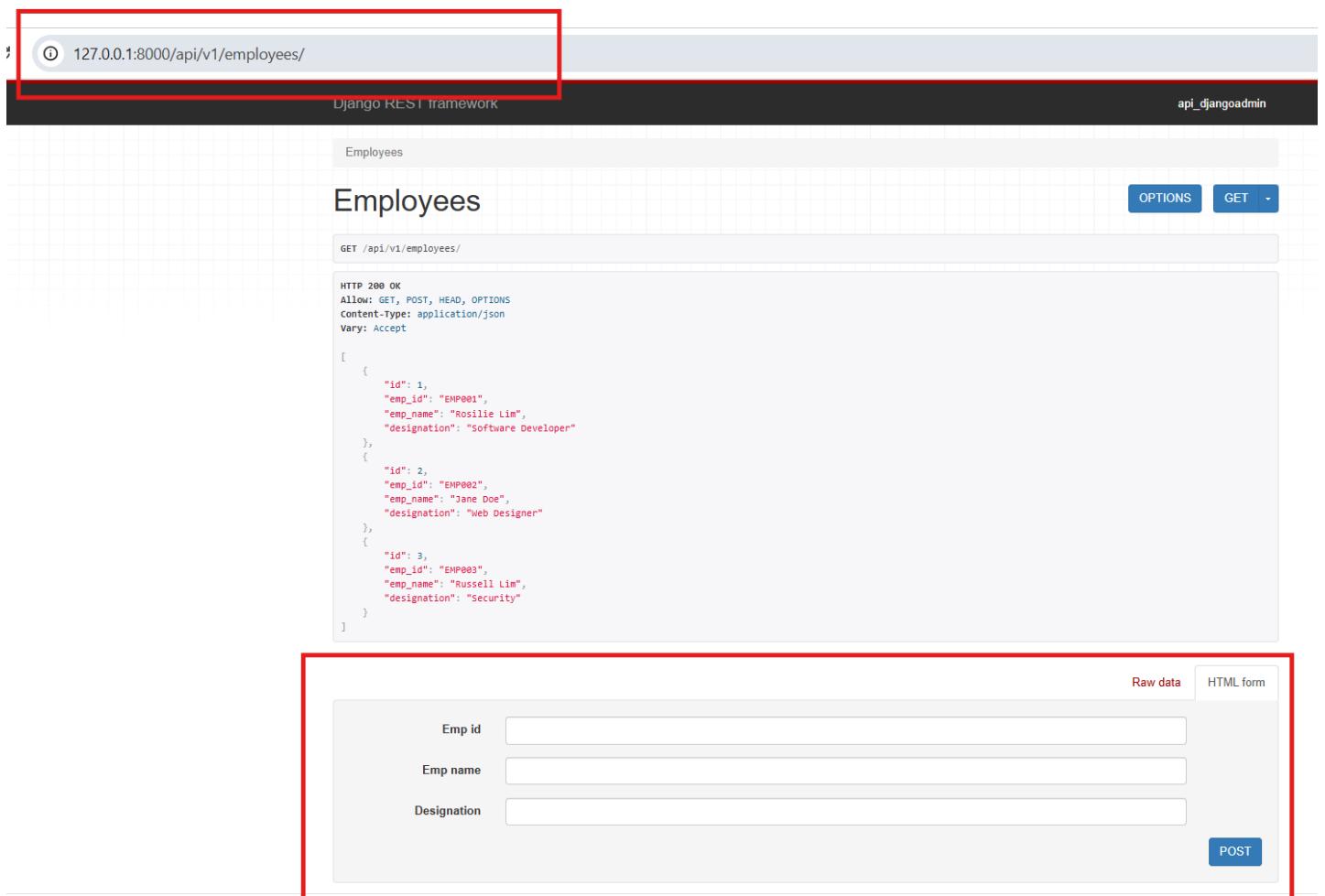
    def post(self, request):
        return self.create(request)

class EmployeeDetail(generics.GenericAPIView):
    pass

```

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.



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

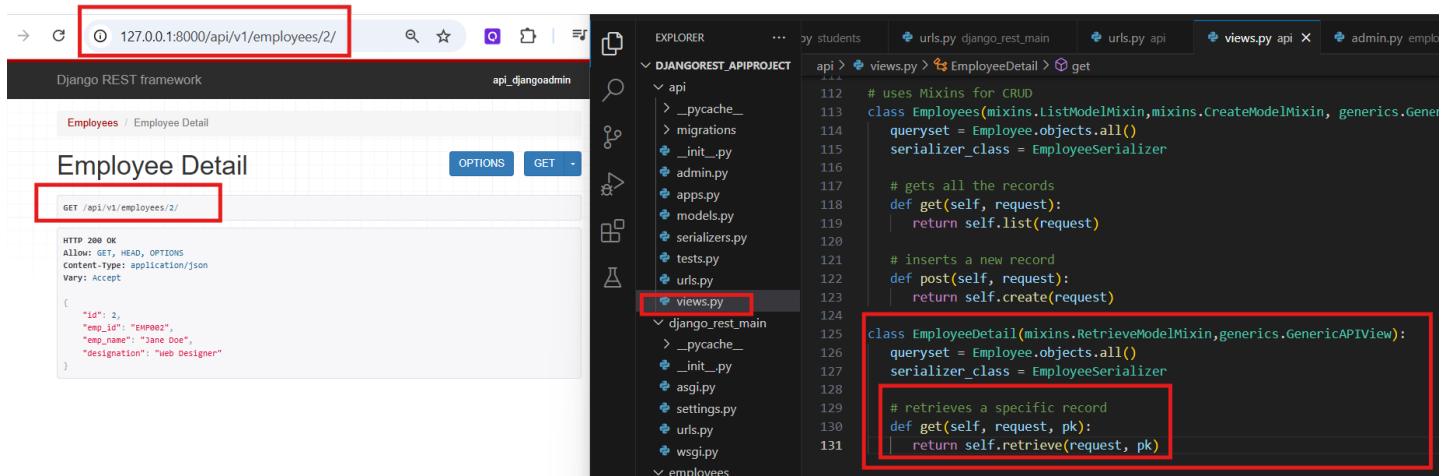
Emp id:

Emp name:

Designation:

POST

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



The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/employees/2/`. The page title is "Employee Detail". The response body shows a JSON object with fields: "id": 2, "emp\_id": "EMP002", "emp\_name": "Jane Doe", and "designation": "Web Designer".

On the right, the Django REST API code is displayed in a code editor:

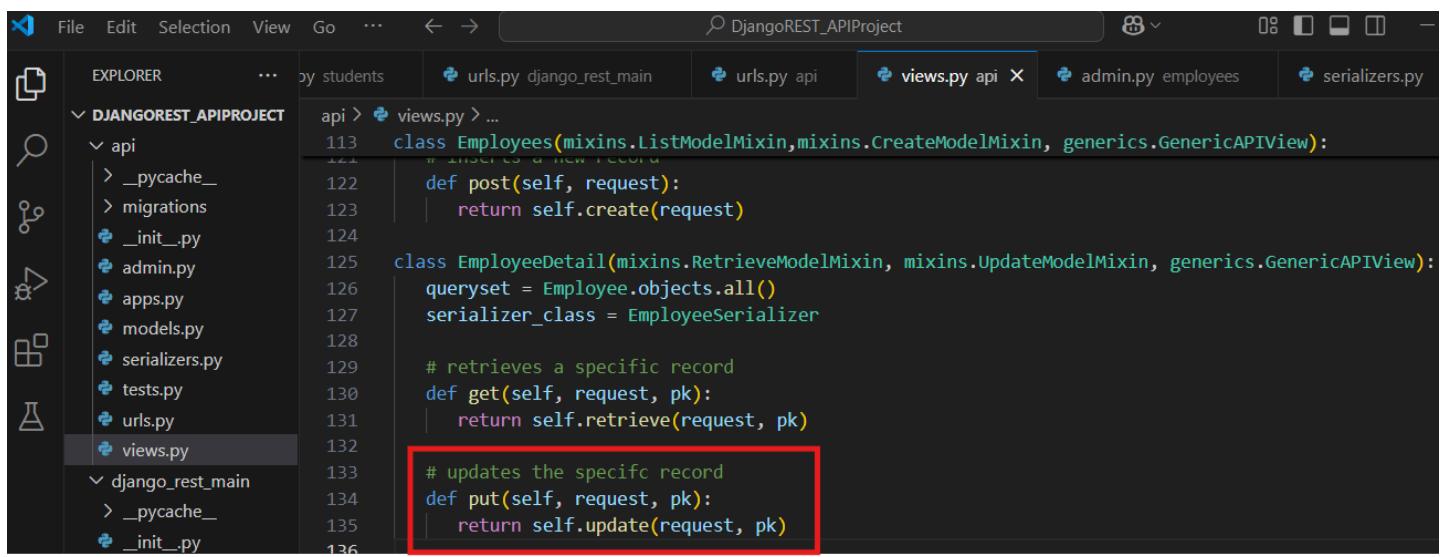
```

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__
    > __init__.py
    > asgi.py
    > settings.py
    > urls.py
    > wsgi.py
  employees
    > __pycache__
    > __init__.py
    > admin.py
    > models.py
    > serializers.py
    > tests.py
    > urls.py
    > views.py
  ...
  api > views.py > EmployeeDetail > get
  # uses Mixins for CRUD
  class Employees(mixins.ListModelMixin,mixins.CreateModelMixin, generics.GenericAPIView):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
    # gets all the records
    def get(self, request):
        return self.list(request)
    # inserts a new record
    def post(self, request):
        return self.create(request)
  class EmployeeDetail(mixins.RetrieveModelMixin,generics.GenericAPIView):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
    # retrieves a specific record
    def get(self, request, pk):
        return self.retrieve(request, pk)

```

The `EmployeeDetail` class and its `get` and `put` methods are highlighted with a red box.

6. To update a specific record, we use the UPDATEMODELMIXIN.



The screenshot shows the Django REST API code in a code editor. The `EmployeeDetail` class is defined with the `UPDATEMODELMIXIN`:

```

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__
    > __init__.py
    > asgi.py
    > settings.py
    > urls.py
    > wsgi.py
  employees
    > __pycache__
    > __init__.py
    > admin.py
    > models.py
    > serializers.py
    > tests.py
    > urls.py
    > views.py
  ...
  api > views.py > ...
  113 class Employees(mixins.ListModelMixin,mixins.CreateModelMixin, generics.GenericAPIView):
  114     # inserts a new record
  115     def post(self, request):
  116         return self.create(request)
  117
  118     class EmployeeDetail(mixins.RetrieveModelMixin, mixins.UpdateModelMixin, generics.GenericAPIView):
  119         queryset = Employee.objects.all()
  120         serializer_class = EmployeeSerializer
  121
  122         # retrieves a specific record
  123         def get(self, request, pk):
  124             return self.retrieve(request, pk)
  125
  126         # updates the specific record
  127         def put(self, request, pk):
  128             return self.update(request, pk)
  129
  130     # gets all the records
  131     def list(self, request):
  132         return self.get(request)
  133
  134     # inserts a new record
  135     def create(self, request):
  136         return self.post(request)

```

The `put` method is highlighted with a red box.

To test:

→   127.0.0.1:8000/api/v1/employees/2/      

Django REST framework api\_djangoadmin

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:

Emp name:   

Designation:

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

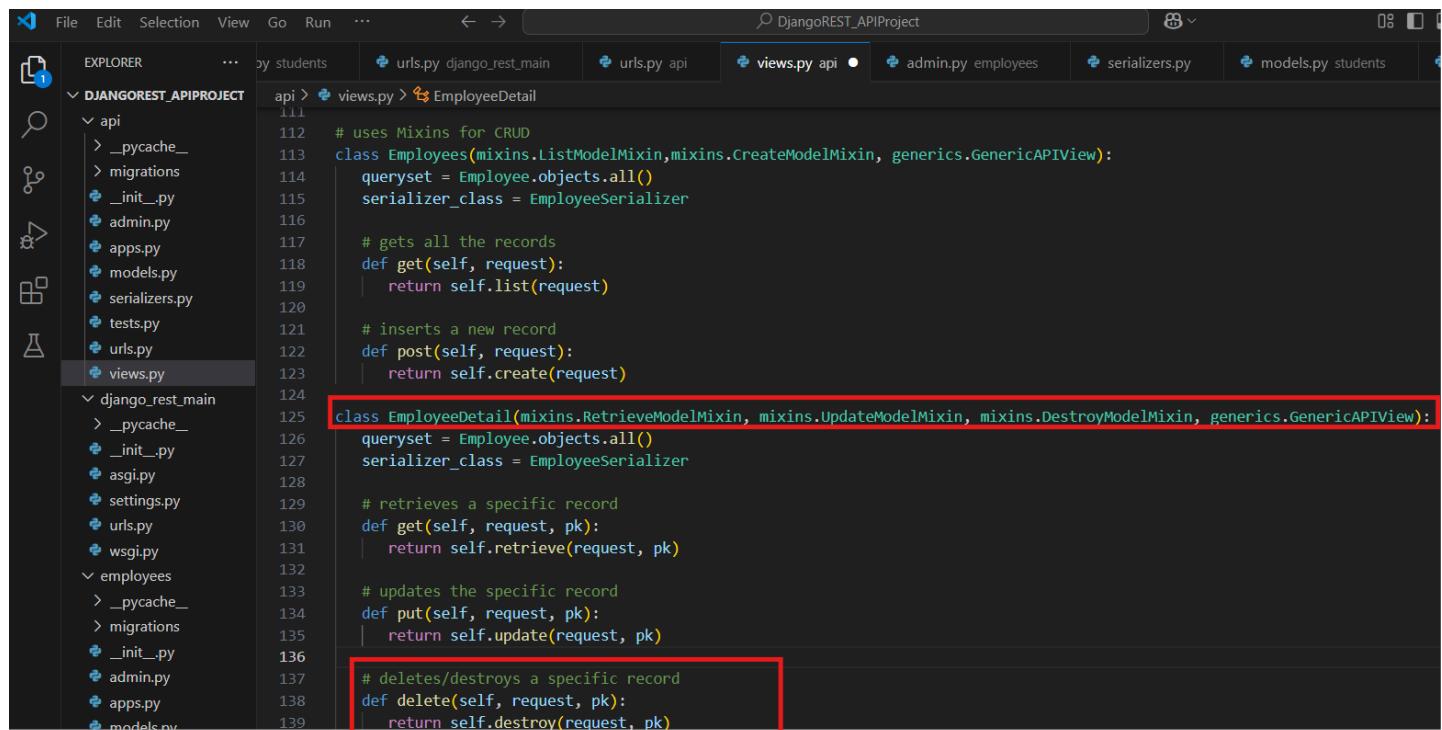
Janet Doe

Designation

Web Designer

[PUT](#)

7. To delete a specific record.



```
File Edit Selection View Go Run ... ← → DjangoREST_APIProject
EXPLORER ... /api/students /urls.py django_rest_main /urls.py api /views.py api ● /admin.py employees /serializers.py /models.py students
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__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  employees
    > __pycache__
    > migrations
    __init__.py
    admin.py
    apps.py
    models.py

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

[PUT](#)

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/employees/2/` in the address bar. The page title is "Django REST framework" and the sub-page title is "api\_djangoadmin". The main content is titled "Employee Detail". On the right, there are three buttons: "DELETE" (red), "OPTIONS" (blue), and "GET" (blue with a dropdown arrow). A red box highlights the address bar and the main content area. The content area shows a "Raw data" section with the following JSON response:

```
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."
}
```

Below this, there is an "HTML form" section with three input fields: "Emp id", "Emp name", and "Designation". A "PUT" button is located to the right of the form. The "Raw data" tab is currently selected.

9.