

## Rest API using ModelViewSet

This Django project demonstrates the creation of API Endpoints using ModelViewSet and default router.

### Prerequisites

- Python (version 3.7 or higher)
- Django (version 3.x or higher)
- DjangoRESTframework (version 3.15.2 or higher)
- Virtual environment tool (optional but recommended)

### Setup Instructions

Follow these steps to set up and run the Django "rest\_api\_demo" project:

#### 1. Create and Activate a Virtual Environment (Optional but Recommended)

It's a good practice to create a virtual environment for each project to manage dependencies.

##### a. Create a virtual environment:

```
python -m venv .venv
```

##### b. Activate the virtual environment:

On Windows:

```
.venv\Scripts\activate
```

On macOS/Linux:

```
source .venv/bin/activate
```

#### 2. Install Django

Once the virtual environment is active, install Django:

```
pip install django  
pip install djangorestframework
```

#### 3. Check version

Python 3

```
python3 --version  
python3 -m django --version
```

Earlier than Pthan Python 3

```
python --version
```

```
python -m django --version
```

#### 4. Create a Django Project

To create a new Django project named `rest_api_demo`:

```
django-admin startproject rest_api_demo
```

#### 5. Navigate into the Project Directory

```
cd rest_api_demo
```

#### 6. Create a Django App

Create an app named `student`:

```
python manage.py startapp student
```

#### 7. Configure the App in the Project

Open `rest_api_demo/settings.py` and add the newly created `student` and `rest framework` to the `INSTALLED_APPS` list:

```
INSTALLED_APPS = [  
    # Django default apps  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    # Add your app here
```

```

'rest_framework', # To implement Django rest framework (DRF)

'student',       # App created

]

```

## 8. Define the Models in app

In `student/models.py`, create student profile model as below

```

class Profile(models.Model):
    # Basic Information
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

```

## 9. Define the Profile Serializer in app

In `student/serializers.py`, create student profile serializer as below

```

from rest_framework import serializers # Import the serializers module from Django
REST Framework
from .models import Profile # Import the Profile model from the current app's models

# Define a serializer for the Profile model
class ProfileSerializer(serializers.ModelSerializer):
    # Meta class defines metadata for the serializer
    class Meta:
        model = Profile # Specify the model to serialize
        fields = ['id', 'first_name', 'last_name', 'email'] # Fields to include in
the serialization

    # Custom validation method for the 'email' field
    def validate_email(self, value):
        """
        Check if the email is unique in the Profile model.
        If the email already exists, raise a ValidationError.
        """
        if Profile.objects.filter(email=value).exists(): # Check if a Profile with
this email already exists
            raise serializers.ValidationError("A student with this email already
exists.") # Raise error if not unique
        return value # Return the value if it's unique

```

## 10. Define API Views in app

In `student/api_views.py`, create views to access endpoints to interface

```

from rest_framework import viewsets # Import the viewsets module to create RESTful
views

from .models import Profile # Import the Profile model to interact with the database
from .serializers import ProfileSerializer # Import the ProfileSerializer for data
serialization and deserialization

# Define a ViewSet for the Profile model
class ProfileViewSet(viewsets.ModelViewSet):
    """

```

```

    A ViewSet for handling CRUD operations on the Profile model.
    Inherits from ModelViewSet, which provides default implementations
    for create, retrieve, update, partial_update, destroy, and list actions.
    """
    queryset = Profile.objects.all() # Define the queryset to include all Profile
objects
    serializer_class = ProfileSerializer # Specify the serializer class to handle
Profile instances

```

## 11. Define the URL Pattern in project

A router in Django REST Framework is a utility that automatically handles the URL routing for your API. It works with ViewSets to map HTTP methods (like GET, POST, PUT, DELETE) to the appropriate actions in your views. This eliminates the need to manually define URL patterns for each action in your application.

If you register a ProfileViewSet with the router using the prefix profiles, the following endpoints are automatically created:

HTTP Method	URL	Action	Description
GET	/api/profiles/	list	Fetch a list of profiles.
POST	/api/profiles/	create	Add a new profile.
GET	/api/profiles/{id}/	retrieve	Get details of a profile.
PUT	/api/profiles/{id}/	update	Update an existing profile.
PATCH	/api/profiles/{id}/	partial_update	Partially update a profile.
DELETE	/api/profiles/{id}/	destroy	Delete a profile.

In `student/urls.py`, define URL patterns to access endpoints

```

from django.urls import path, include # Import Django's path and include functions
for URL routing
from rest_framework.routers import DefaultRouter # Import DefaultRouter to generate
URL patterns for ViewSets
from .api_views import ProfileViewSet # Import the ProfileViewSet to register it with
the router

# Create a router instance
router = DefaultRouter() # DefaultRouter provides automatic URL routing for the API
router.register(r'profiles', ProfileViewSet) # Register the ProfileViewSet with the
router
# 'r' indicates a raw string, 'profiles' will be the URL prefix for the API endpoints

# Define the URL patterns
urlpatterns = [
    path('', include(router.urls)), # Include all automatically generated URLs from
the router
    # This maps the base path (') to the router's URLs, enabling all ProfileViewSet
endpoints
]

```

In `rest_api_demo/urls.py`, include the student app URLs:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('student.urls'))
]
```

## 12. Run Database Migrations

Before running the server, apply the initial migrations:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

## 13. Run the Development Server

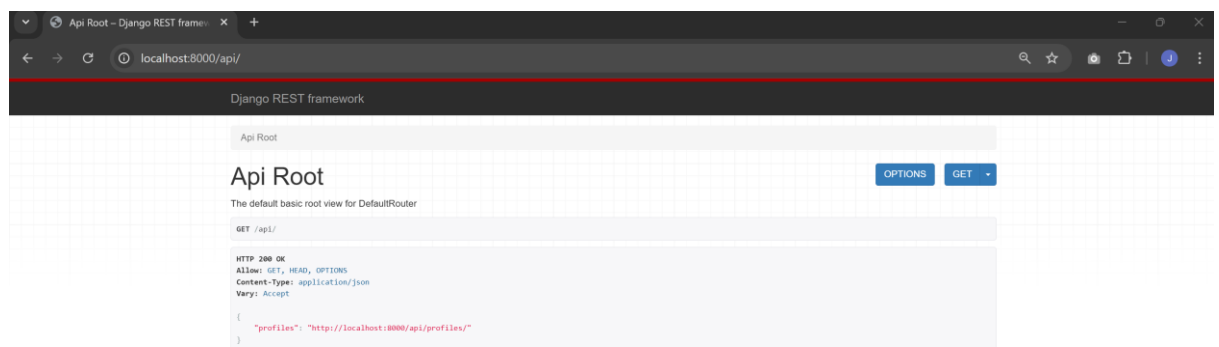
Finally, start the Django development server:

```
python manage.py runserver
```

## 13. Launch the Application and see the result

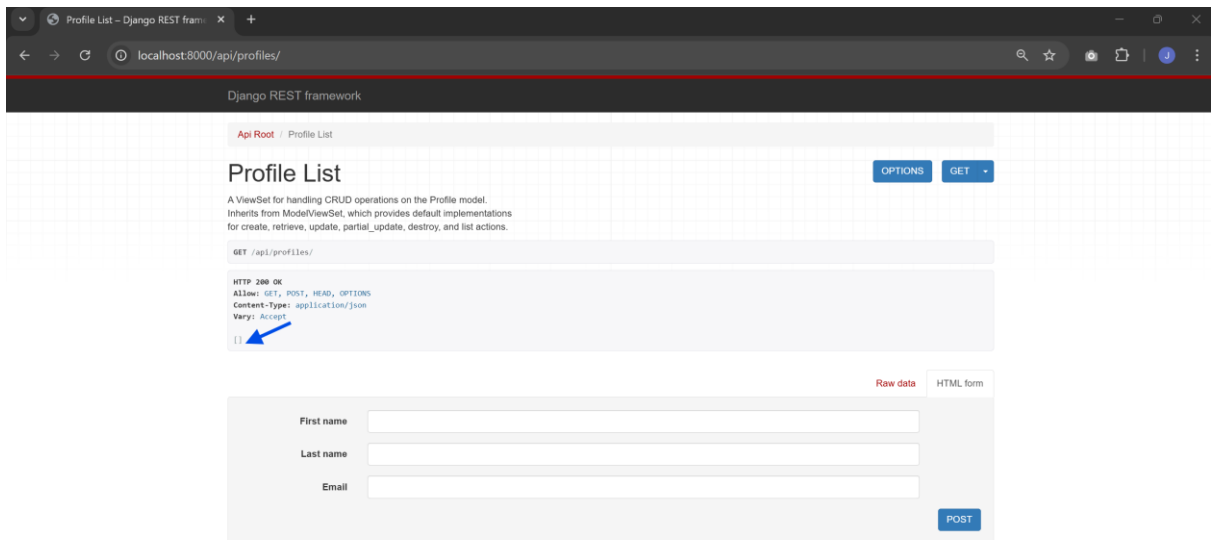
Open your web browser and visit:

```
http://127.0.0.1:8000/api
```

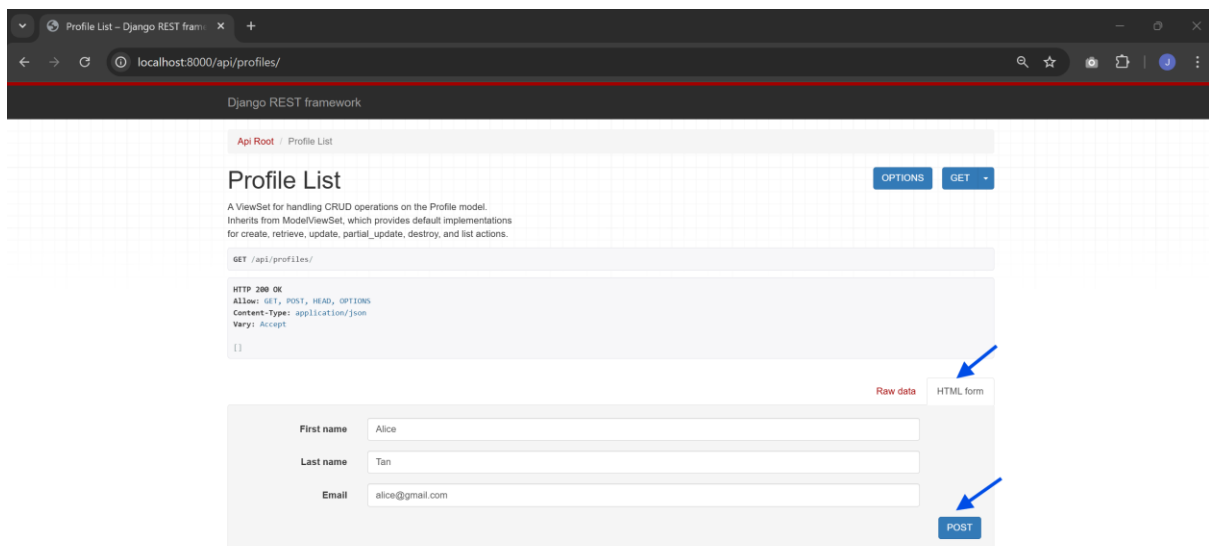


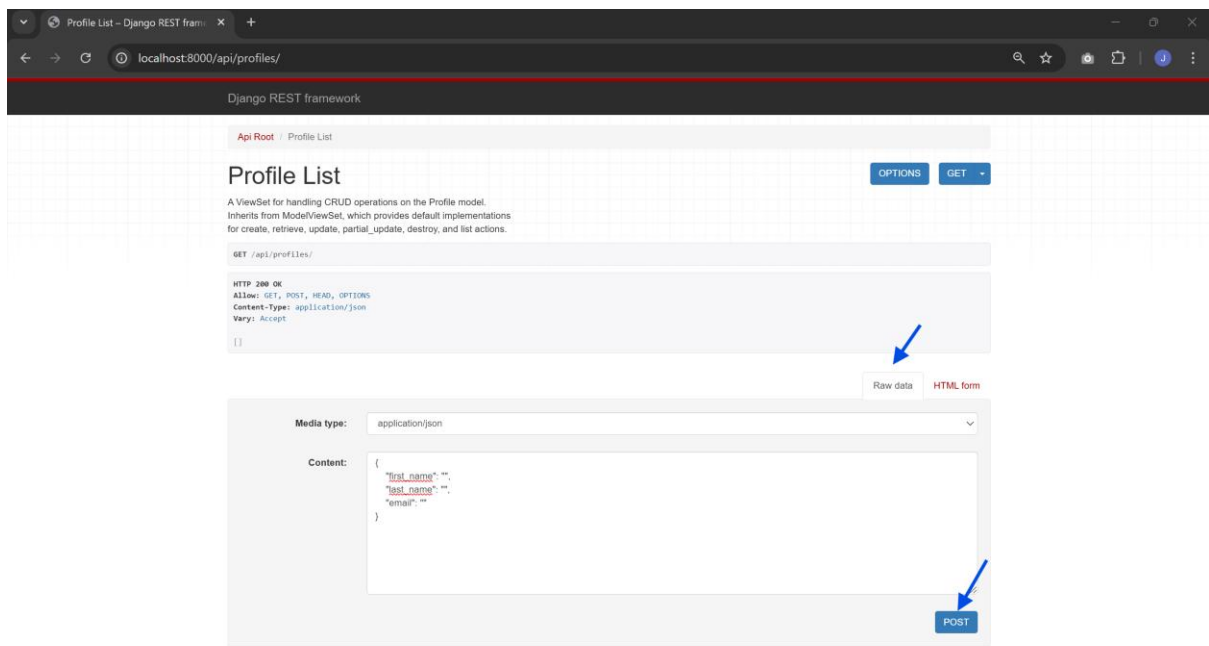
To list all the profiles – <http://127.0.0.1:8000/api/profiles>

You can observe the empty list

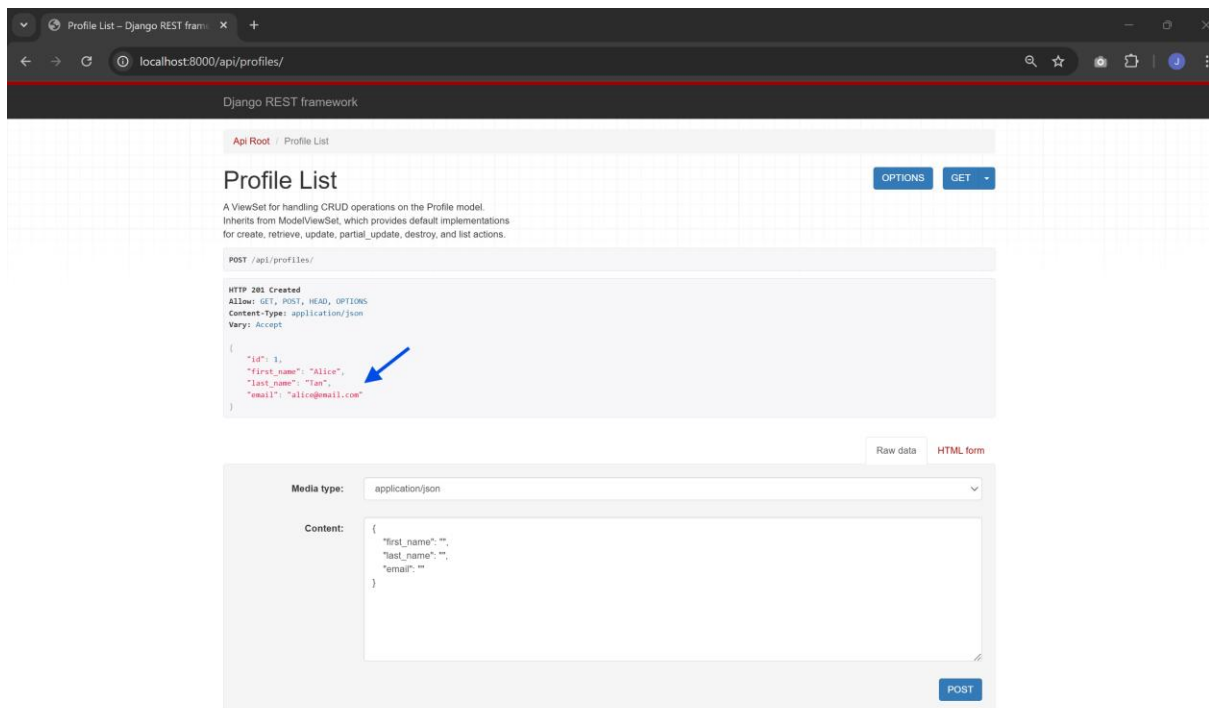


Switch between HTML Form and Raw Data to create the profile. Enter the values for properties and POST.



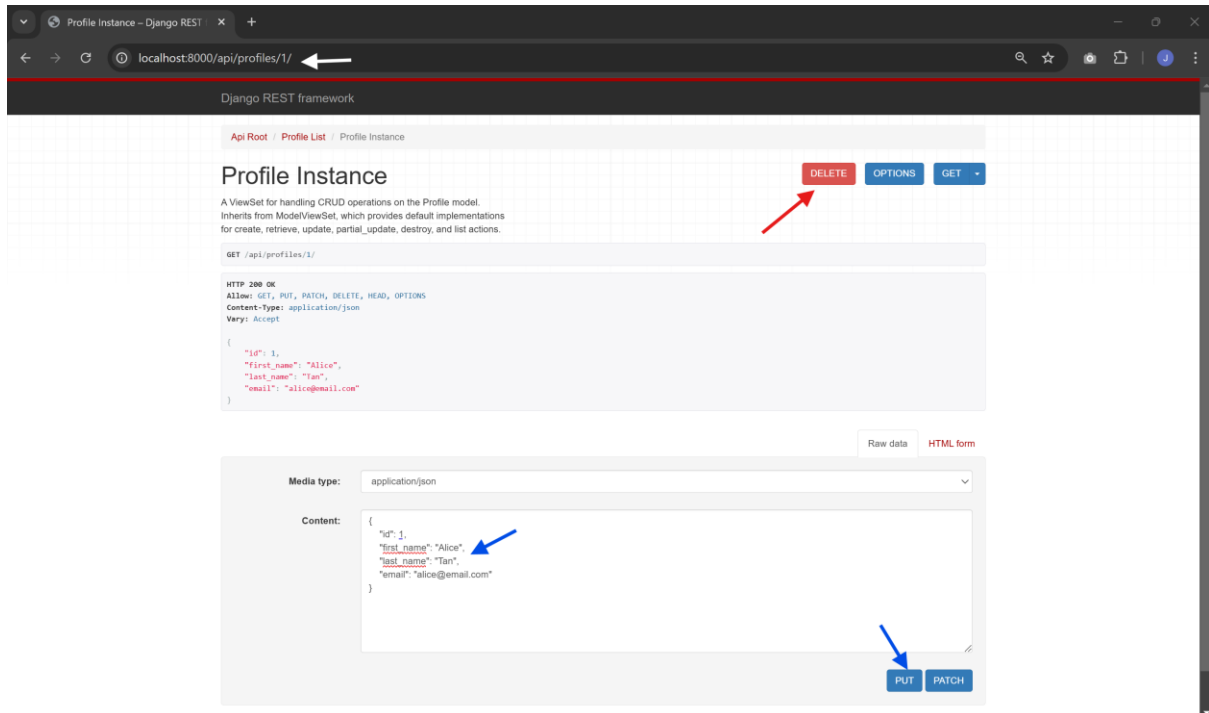


Enter the details and click POST. Observe that the profile being created and listed



Select the profile based on id and make changes to the data to update the profile details.

Select the profile based on id and delete the profile



## 14. Deactivating the Virtual Environment

To deactivate the virtual environment when you're done working:

*Deactivate*