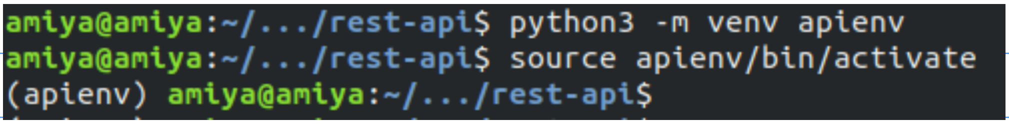
https://www.udemy.com/django-python/

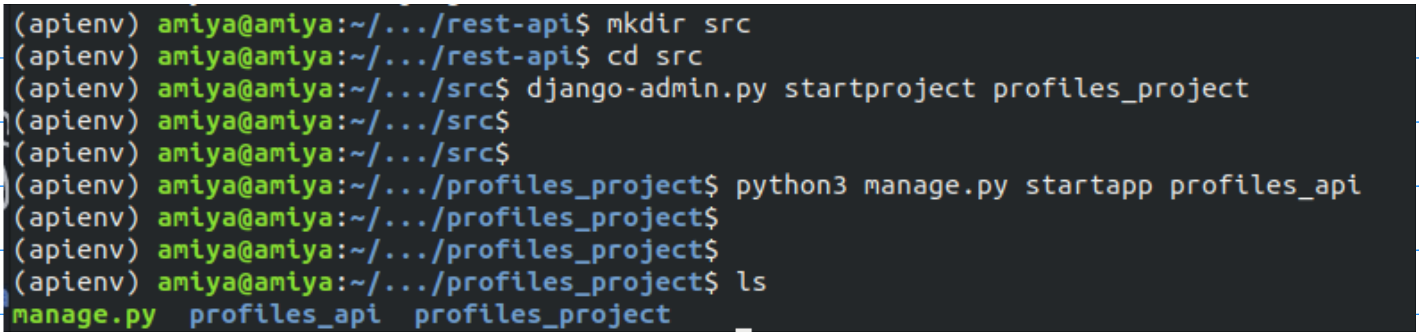
sudo apt-get install python3-venv



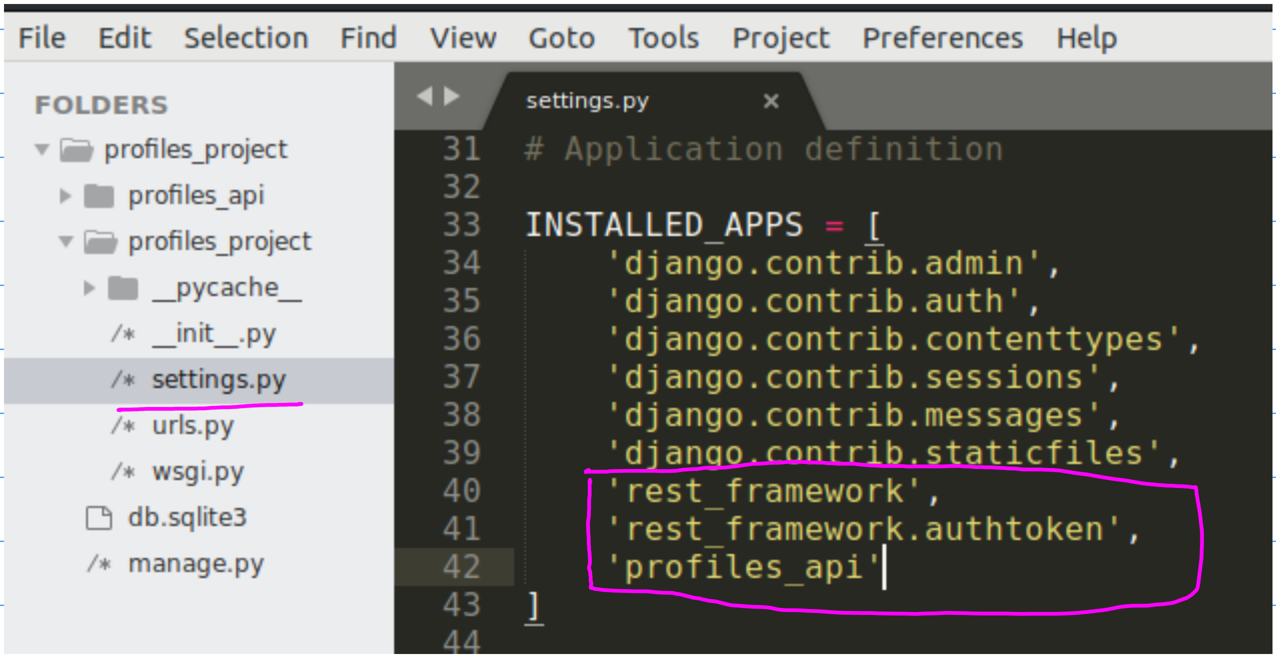
```
amiya@amiya:~/.../rest-api$ python3 -m venv apienv
amiya@amiya:~/.../rest-api$ source apienv/bin/activate
(apienv) amiya@amiya:~/.../rest-api$
```

pip3 install django

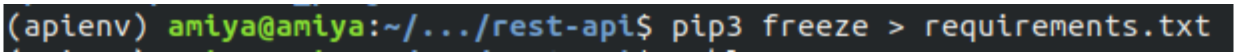
pip3 install djangorestframework



```
(apienv) amiya@amiya:~/.../rest-api$ mkdir src
(apienv) amiya@amiya:~/.../rest-api$ cd src
(apienv) amiya@amiya:~/.../src$ django-admin.py startproject profiles_project
(apienv) amiya@amiya:~/.../src$
(apienv) amiya@amiya:~/.../src$
(apienv) amiya@amiya:~/.../profiles_project$ python3 manage.py startapp profiles_api
(apienv) amiya@amiya:~/.../profiles_project$
(apienv) amiya@amiya:~/.../profiles_project$
(apienv) amiya@amiya:~/.../profiles_project$ ls
manage.py  profiles_api  profiles_project
```



```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'profiles_api'
]
```



```
(apienv) amiya@amiya:~/.../rest-api$ pip3 freeze > requirements.txt
```

What are APIViews?

Uses standar HTTP Methods for functions

GET, POST, PUT, PATCH, DELETE

Gives you the most control over the logic:

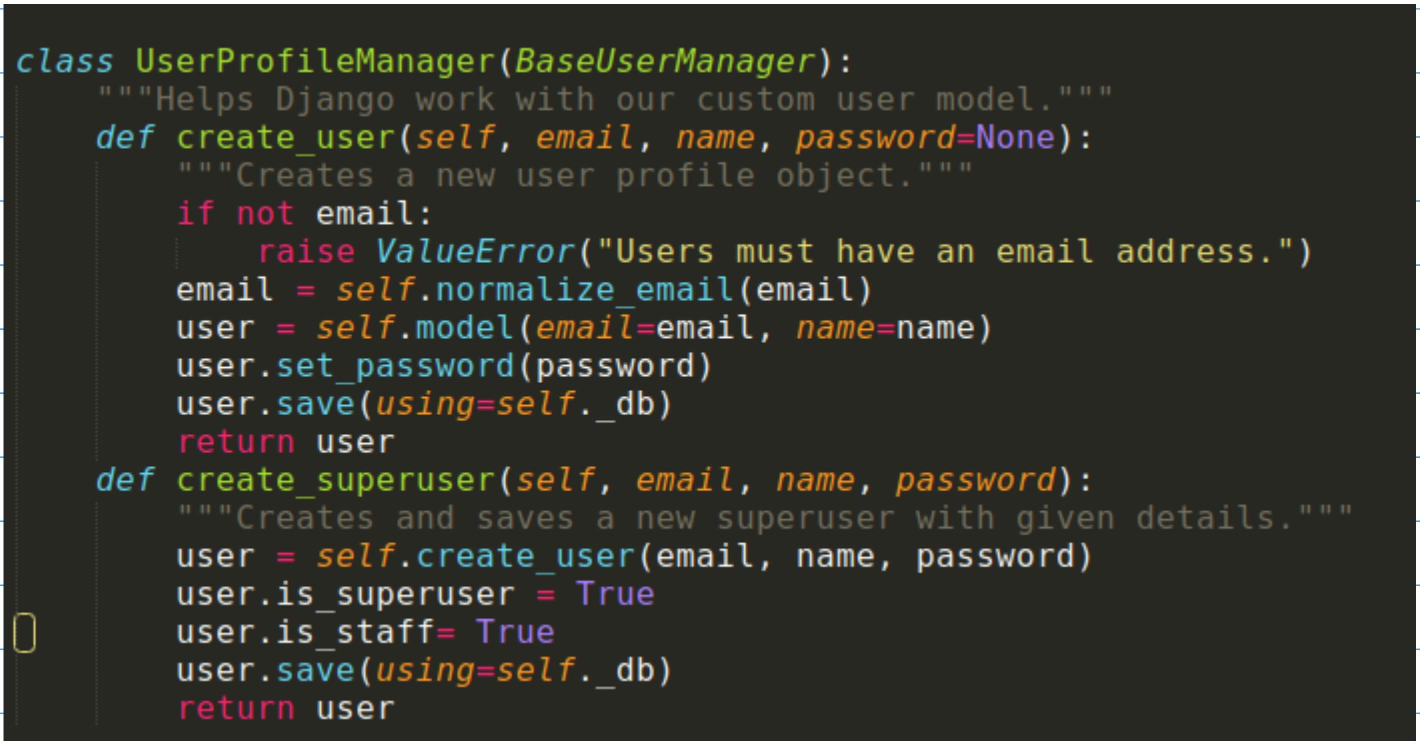
Perfect for implementing complex logic

Calling other APIs

When to use APIViews?

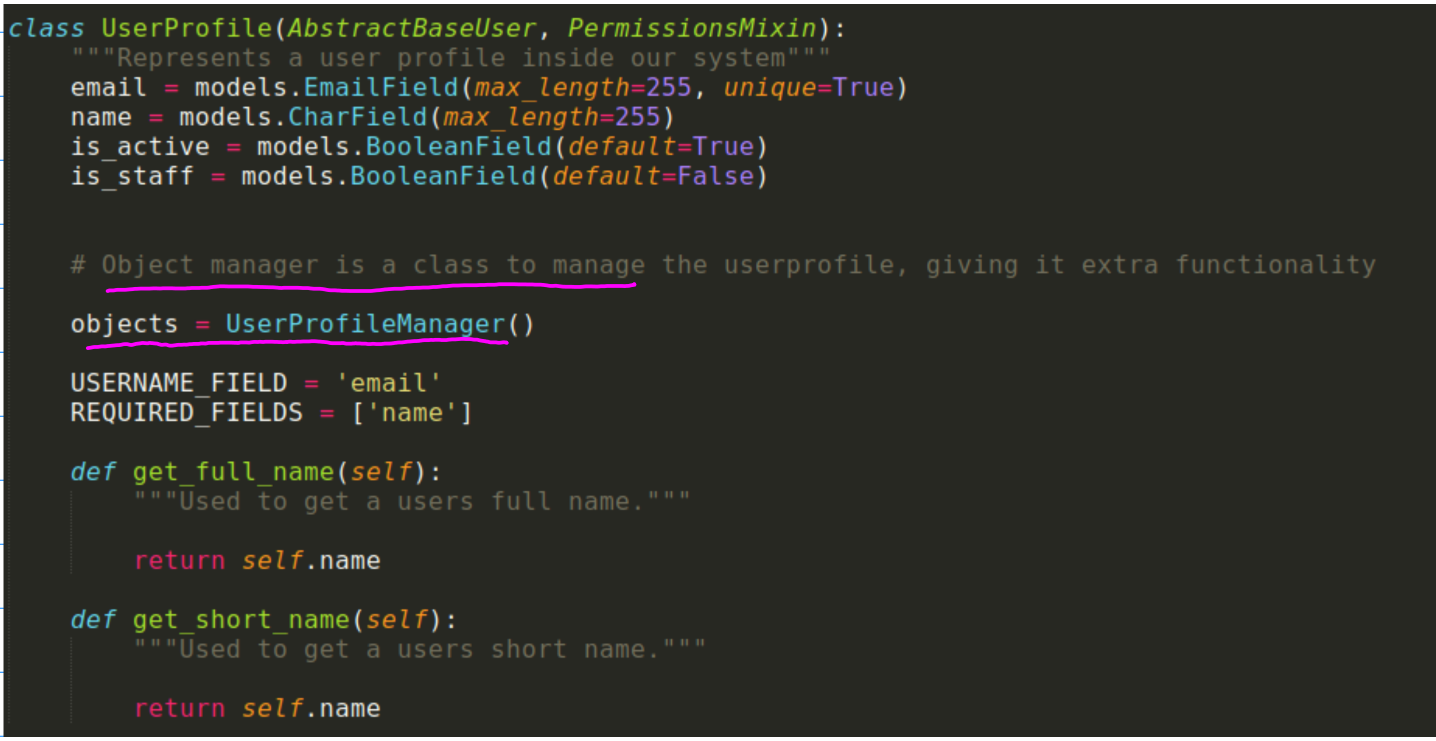
Some examples of when to use an APIView:
-- You need the full control over the logic.
-- Processing files and rendering a synchronous response.
-- You are calling other APIs/Services.
-- Accessing local files or data.

models.py

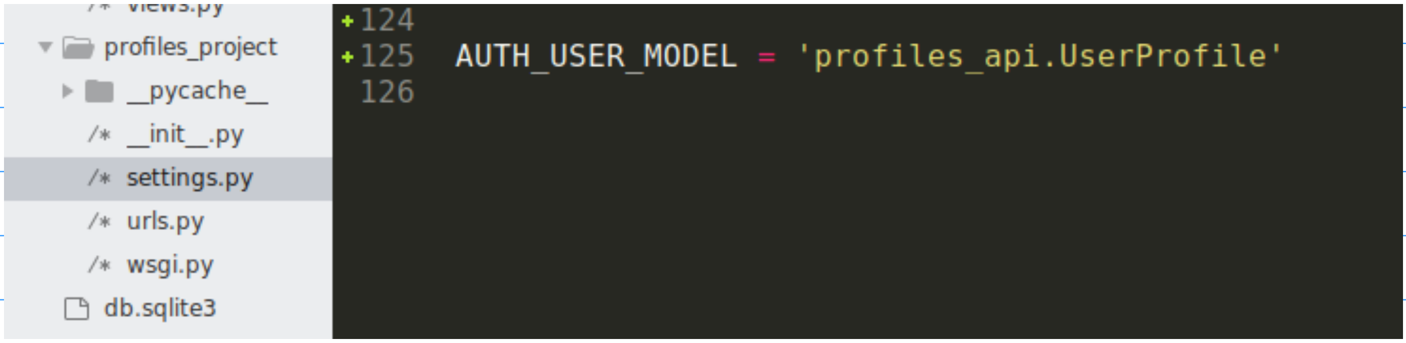


```python
class UserProfileManager(BaseUserManager):
    """Helps Django work with our custom user model."""
    def create_user(self, email, name, password=None):
        """Creates a new user profile object."""
        if not email:
            raise ValueError("Users must have an email address.")
        email = self.normalize_email(email)
        user = self.model(email=email, name=name)
        user.set_password(password)
        user.save(using=self._db)
        return user
    def create_superuser(self, email, name, password):
        """Creates and saves a new superuser with given details."""
        user = self.create_user(email, name, password)
        user.is_superuser = True
        user.is_staff= True
        user.save(using=self._db)
        return user
```

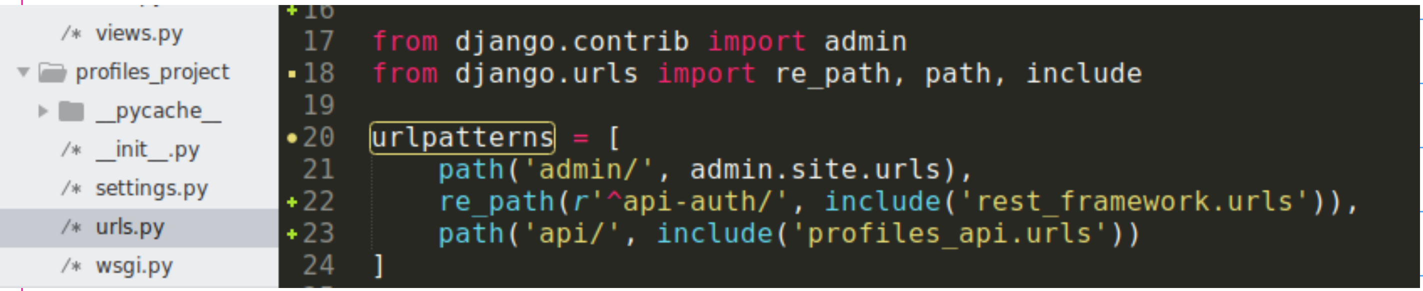
models.py



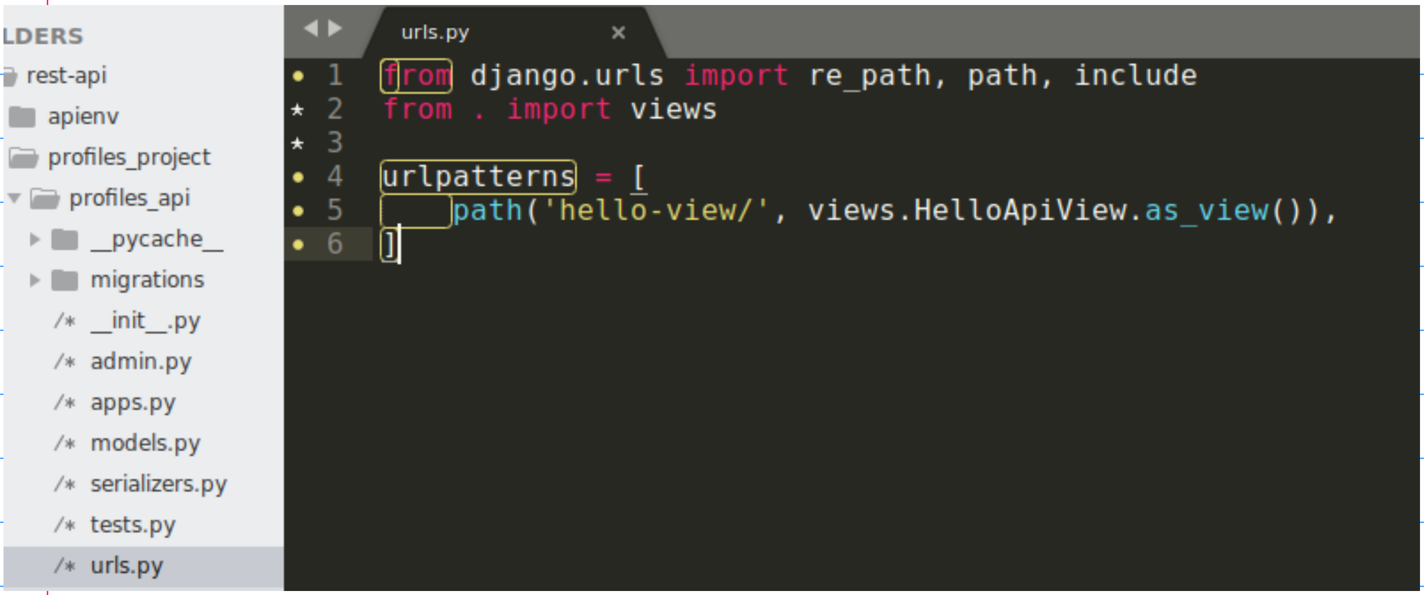
```python
class UserProfile(AbstractBaseUser, PermissionsMixin):
    """Represents a user profile inside our system"""
    email = models.EmailField(max_length=255, unique=True)
    name = models.CharField(max_length=255)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)


    # Object manager is a class to manage the userprofile, giving it extra functionality

    objects = UserProfileManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['name']

    def get_full_name(self):
        """Used to get a users full name."""

        return self.name

    def get_short_name(self):
        """Used to get a users short name."""

        return self.name
```



```python
AUTH_USER_MODEL = 'profiles_api.UserProfile'
```

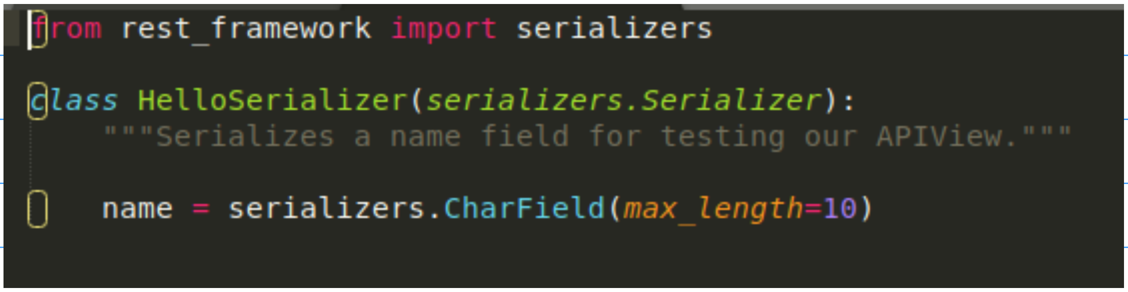


```python
from django.contrib import admin
from django.urls import re_path, path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path(r'^api-auth/', include('rest_framework.urls')),
    path('api/', include('profiles_api.urls'))
]
```



```python
from django.urls import re_path, path, include
from . import views

urlpatterns = [
    path('hello-view/', views.HelloApiView.as_view()),
]
```

profiles_api/serializers.py



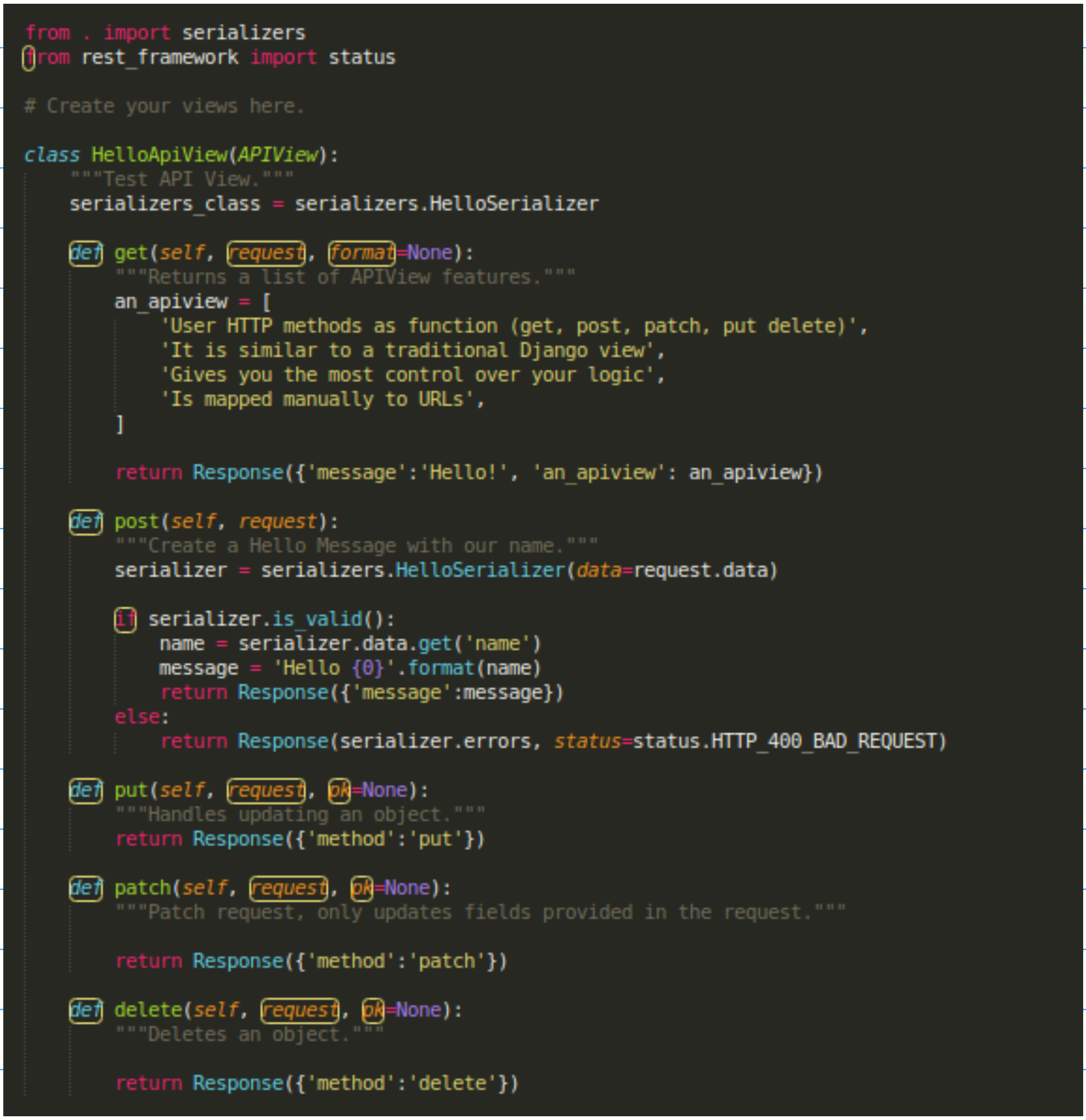
```python
from rest_framework import serializers

class HelloSerializer(serializers.Serializer):
    """Serializes a name field for testing our APIView."""

    name = serializers.CharField(max_length=10)
```

What are APIViews?

Uses standard HTTP Methods for functions

GET, POST, PUT, PATCH, DELETE

profiles_api/views.py



```python
from . import serializers
from rest_framework import status

# Create your views here.

class HelloApiView(APIView):
    """Test API View."""
    serializers_class = serializers.HelloSerializer

    def get(self, request, format=None):
        """Returns a list of APIView features."""
        an_apiview = [
            'User HTTP methods as function (get, post, patch, put delete)',
            'It is similar to a traditional Django view',
            'Gives you the most control over your logic',
            'Is mapped manually to URLs',
        ]

        return Response({'message':'Hello!', 'an_apiview': an_apiview})

    def post(self, request):
        """Create a Hello Message with our name."""
        serializer = serializers.HelloSerializer(data=request.data)

        if serializer.is_valid():
            name = serializer.data.get('name')
            message = 'Hello {0}'.format(name)
            return Response({'message':message})
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def put(self, request, pk=None):
        """Handles updating an object."""
        return Response({'method':'put'})

    def patch(self, request, pk=None):
        """Patch request, only updates fields provided in the request."""

        return Response({'method':'patch'})

    def delete(self, request, pk=None):
        """Deletes an object."""

        return Response({'method':'delete'})
```

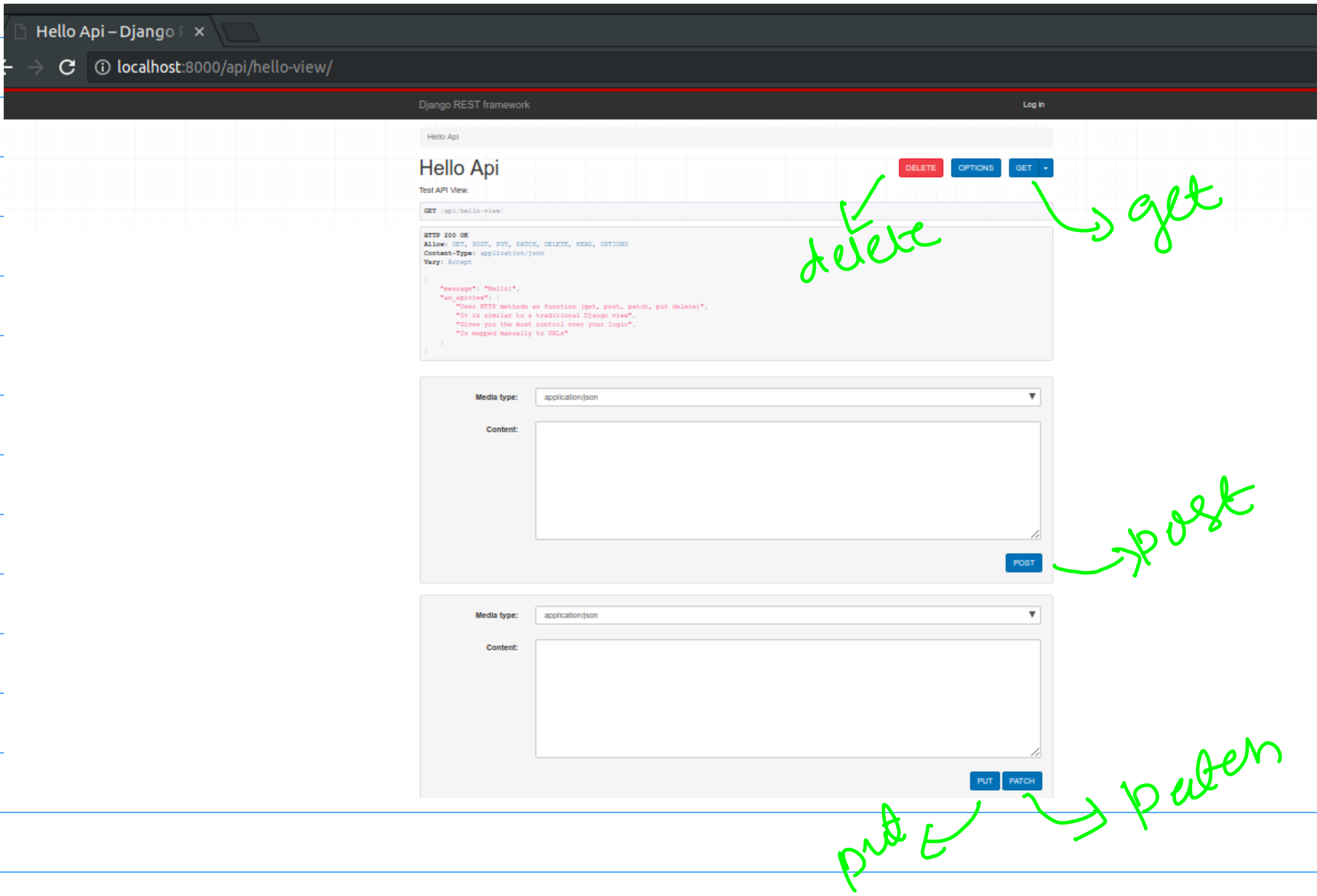
GET

POST

PUT

PATCH

DELETE



Hello Api

Test API View.

```
GET /api/hello-view/

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

{
    "message": "Hello!",
    "an_apiview": [
        "User HTTP methods as function (get, post, patch, put delete)",
        "It is similar to a traditional Django view",
        "Gives you the most control over your logic",
        "Is mapped manually to URLs"
    ]
}
```

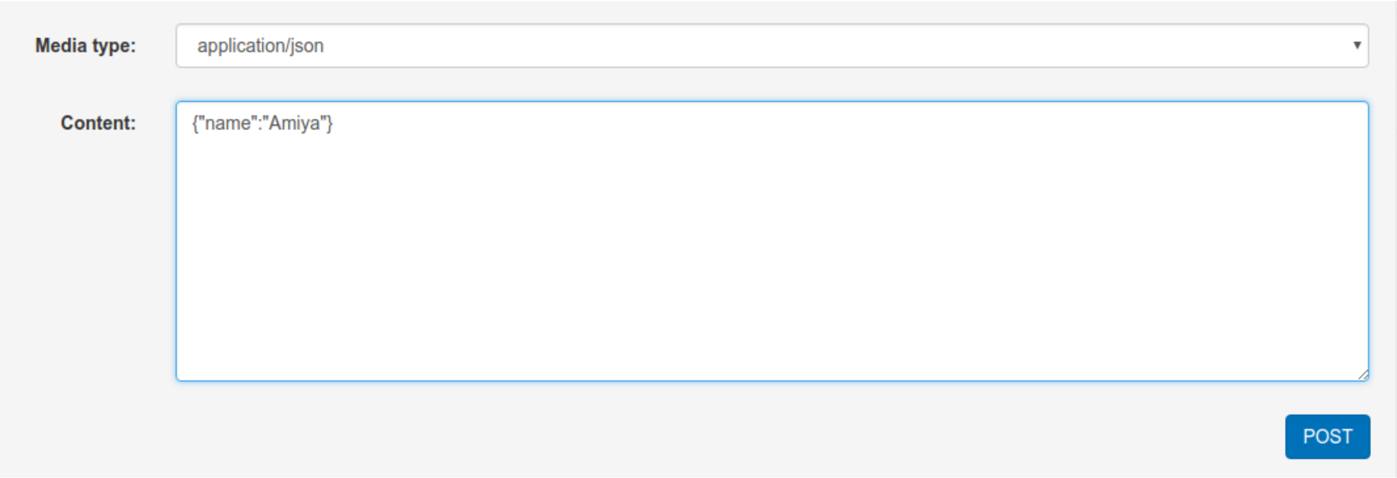
Media type: application/json

Content:

Media type: application/json

Content:

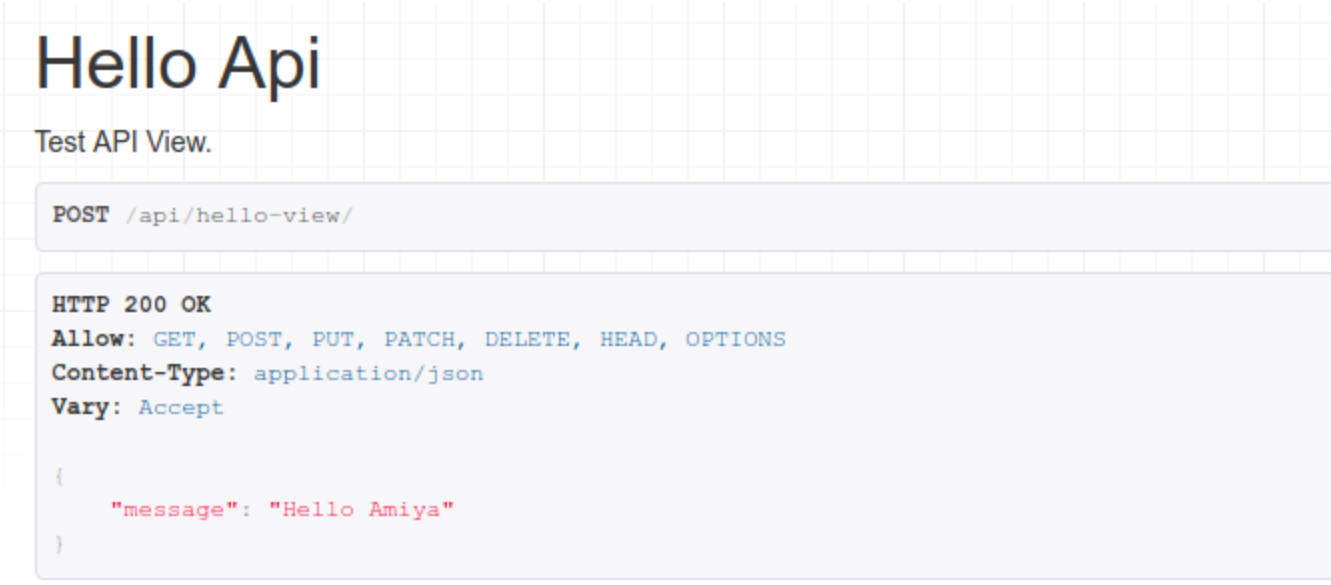
POST



Media type: application/json

Content: {"name":"Amiya"}

POST Results



Hello Api

Test API View.

```
POST /api/hello-view/
```

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

{
    "message": "Hello Amiya"
}
```

Viewsets

Examples of when you might use a Viewset:

-- Your need a simple CRUD interface to your database.
-- You want a quick and simple API.
-- You need little to no customization on the logic.
-- You are working with standard data structures.

Uses model operations for functions:

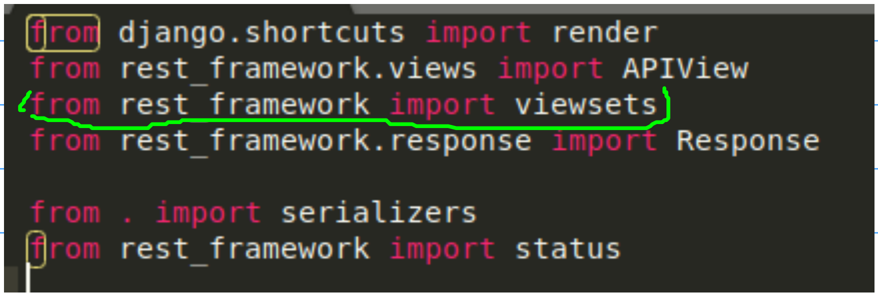
-- List, Create, Retrieve, Update, Parial Update, Destroy

Takes care of lot of typical logic for you:
-- Perfect for standard database operations
-- Fastest way to make a database interface

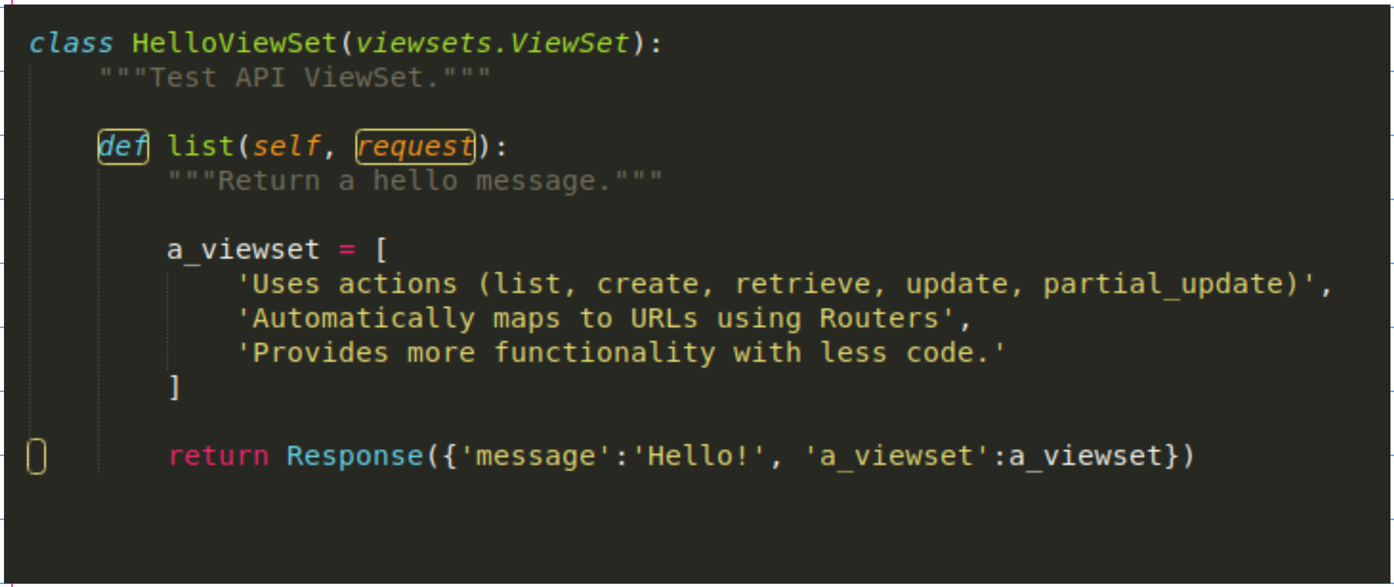
Examples of when you might use a ViewSet:

-- You need a simple CRUD interface to your database.
-- You want a quick and simple API.
-- You need little to no customization on the logic.
-- You are working with standard data structures.

Views:

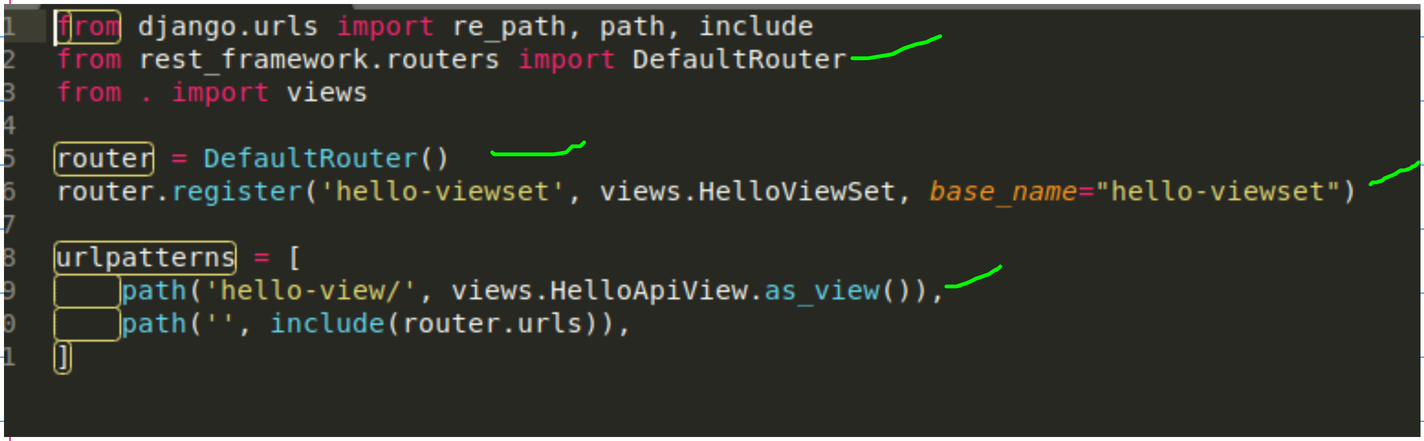


```python
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework import viewsets
from rest_framework.response import Response

from . import serializers
from rest_framework import status
```



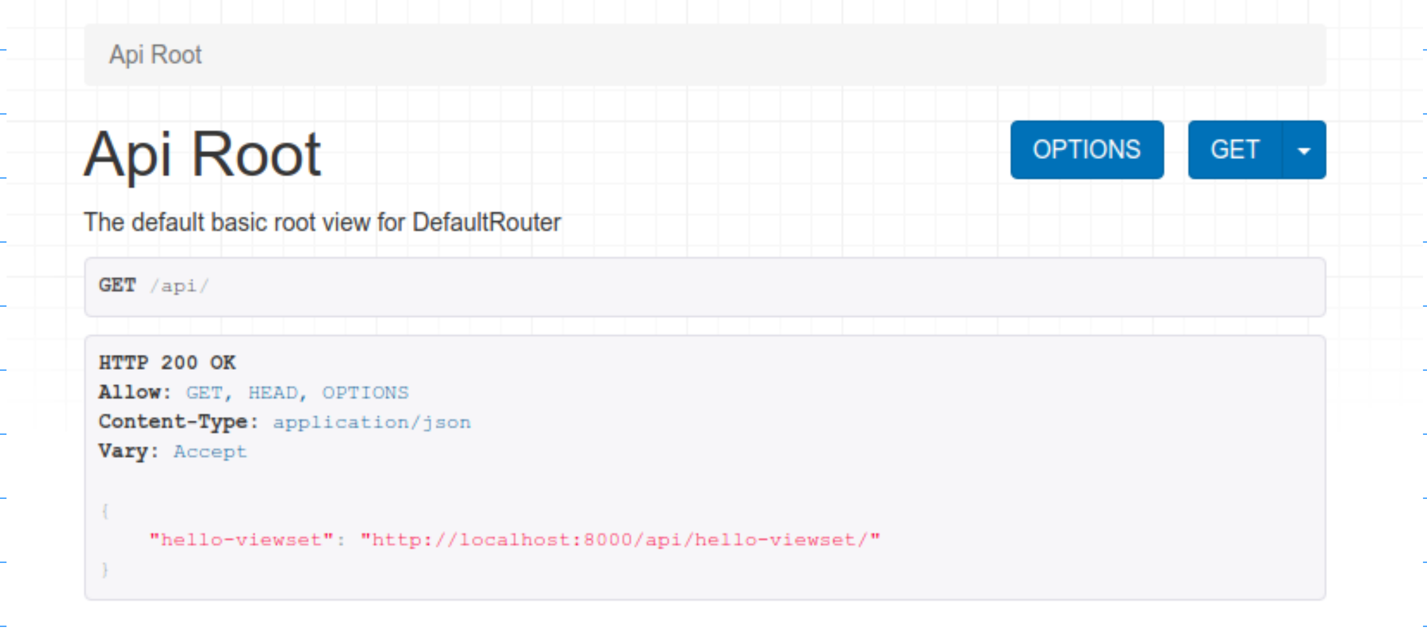
```python
class HelloViewSet(viewsets.ViewSet):
    """Test API ViewSet."""

    def list(self, request):
        """Return a hello message."""

        a_viewset = [
            'Uses actions (list, create, retrieve, update, partial_update)',
            'Automatically maps to URLs using Routers',
            'Provides more functionality with less code.'
        ]

        return Response({'message':'Hello!', 'a_viewset':a_viewset})
```

urls.py



```python
from django.urls import re_path, path, include
from rest_framework.routers import DefaultRouter
from . import views

router = DefaultRouter()
router.register('hello-viewset', views.HelloViewSet, base_name="hello-viewset")

urlpatterns = [
    path('hello-view/', views.HelloApiView.as_view()),
    path('', include(router.urls)),
]
```

http://localhost:8000/api/



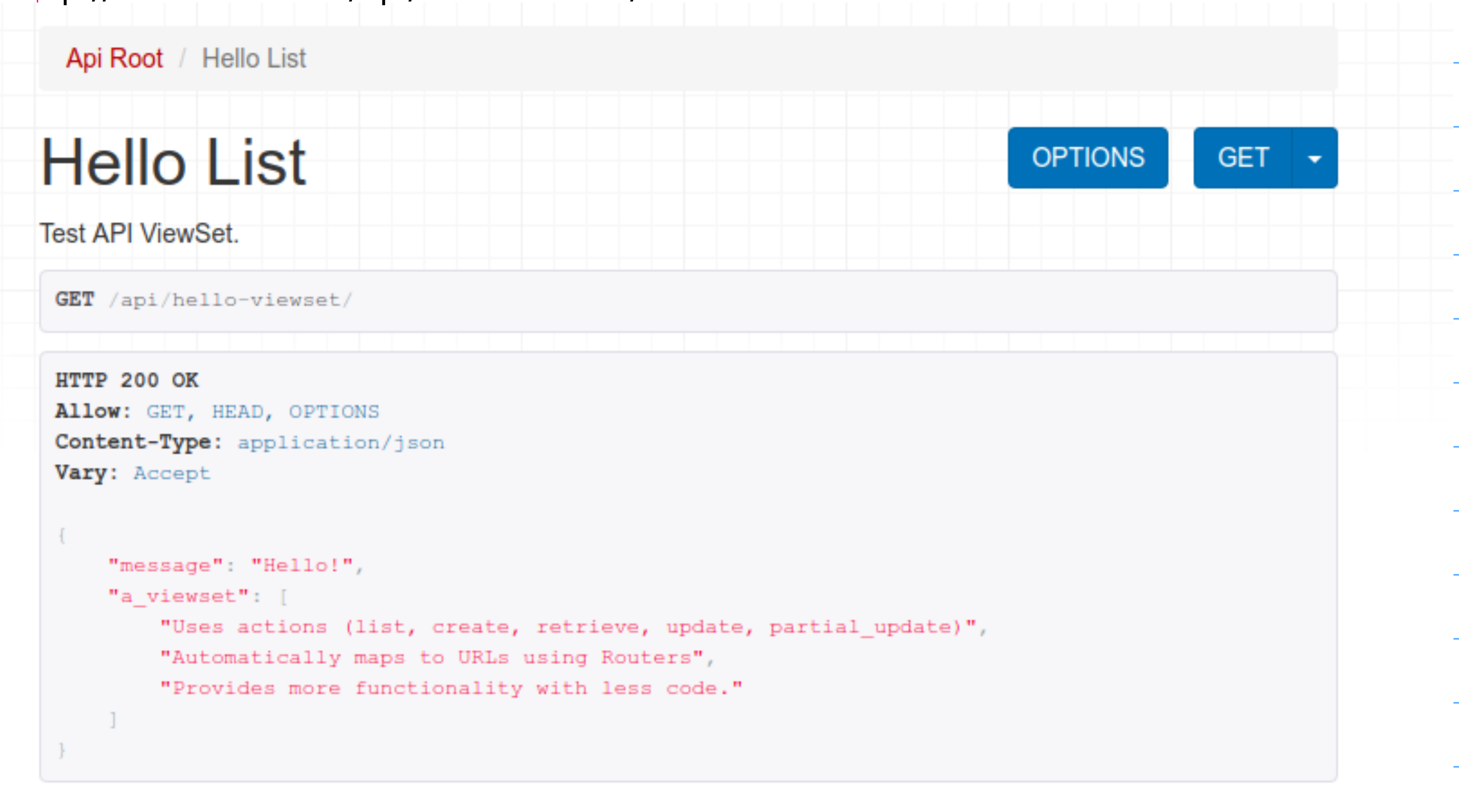
Api Root

The default basic root view for DefaultRouter

```
GET /api/
```

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

{
    "hello-viewset": "http://localhost:8000/api/hello-viewset/"
}
```

http://localhost:8000/api/hello-viewset/

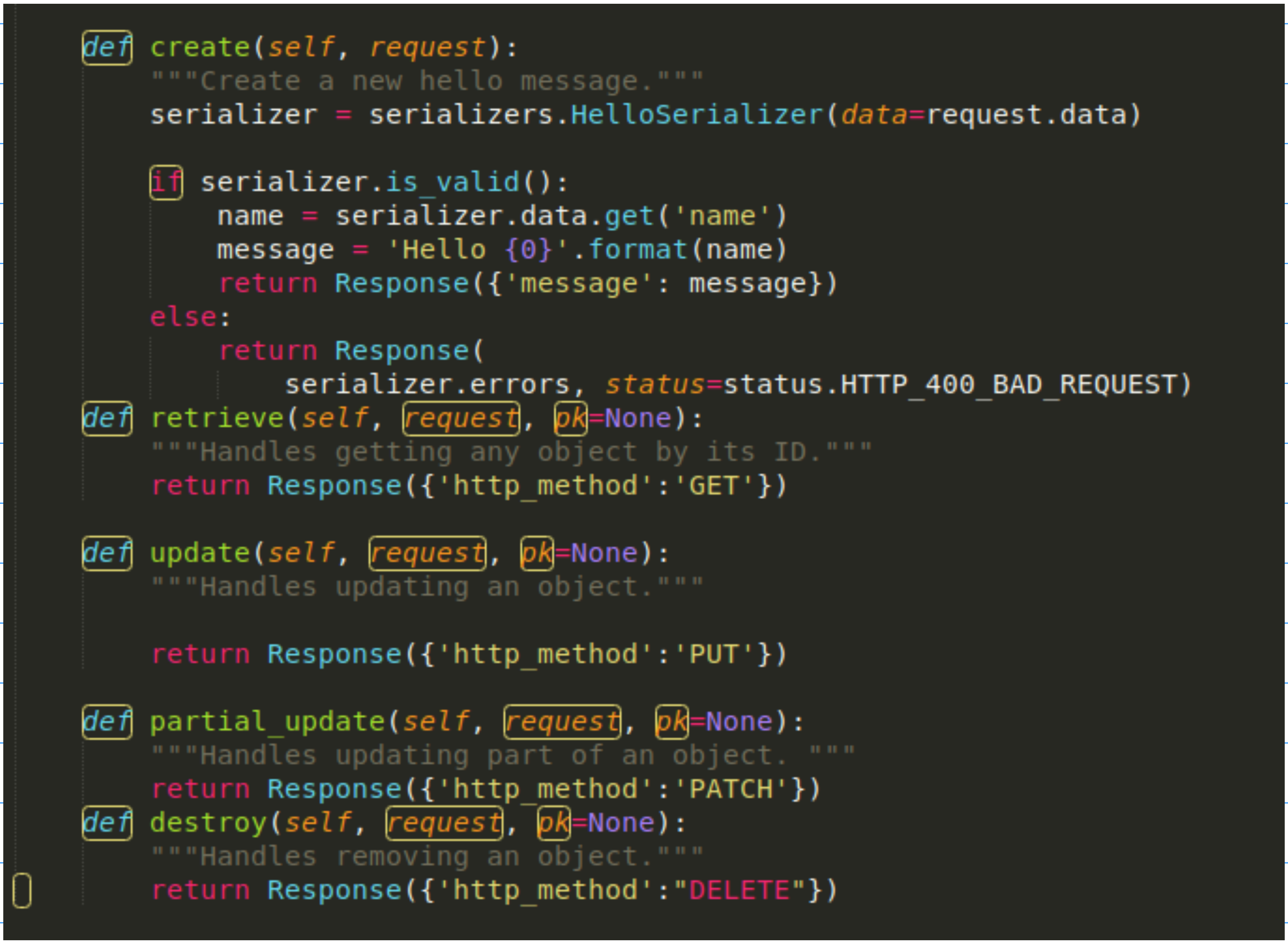


Hello List

Test API ViewSet.

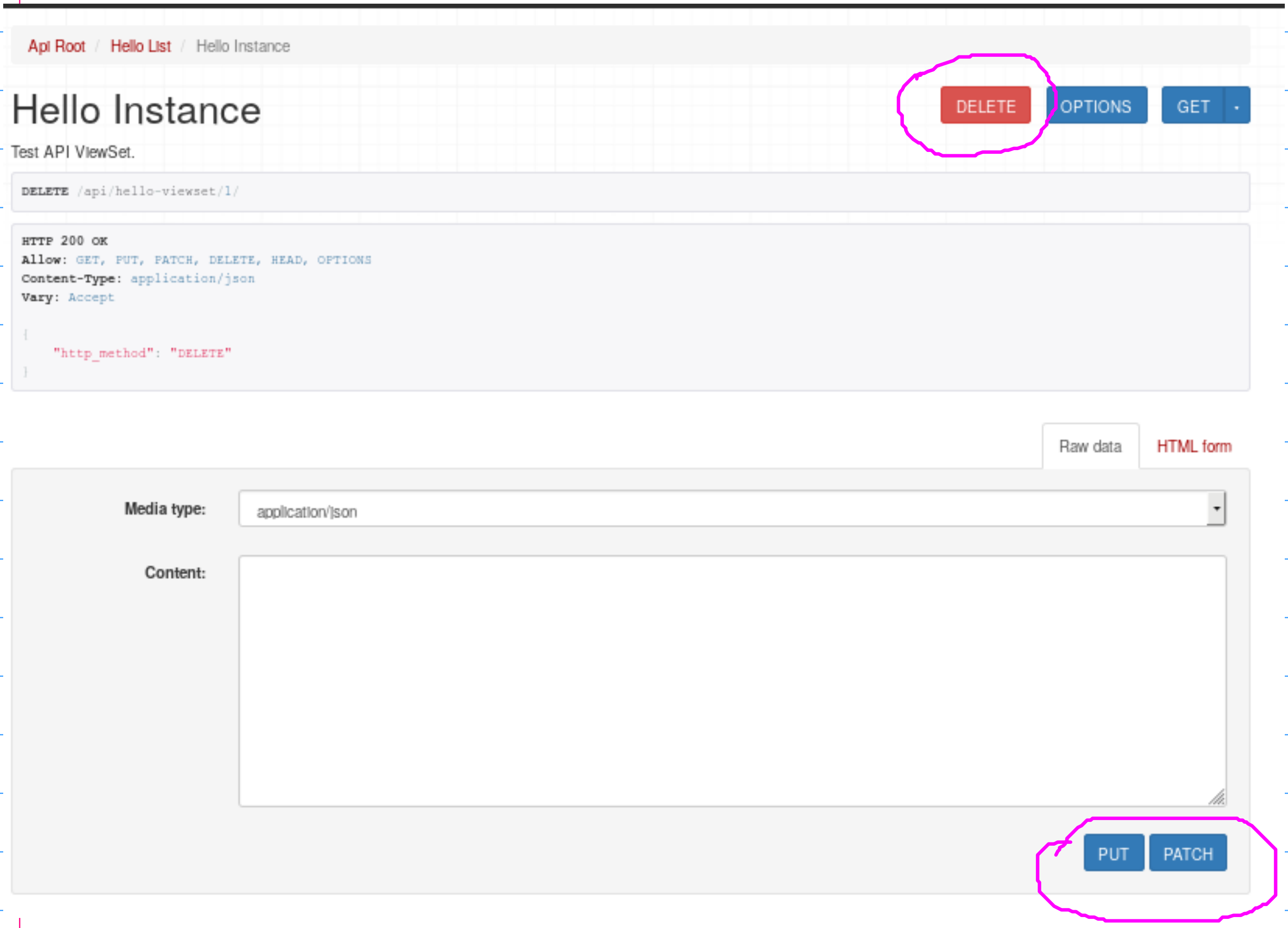
```
GET /api/hello-viewset/
```

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

{
    "message": "Hello!",
    "a_viewset": [
        "Uses actions (list, create, retrieve, update, partial_update)",
        "Automatically maps to URLs using Routers",
        "Provides more functionality with less code."
    ]
}
```



```python
    def create(self, request):
        """Create a new hello message."""
        serializer = serializers.HelloSerializer(data=request.data)

        if serializer.is_valid():
            name = serializer.data.get('name')
            message = 'Hello {0}'.format(name)
            return Response({'message': message})
        else:
            return Response(
                serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    def retrieve(self, request, pk=None):
        """Handles getting any object by its ID."""
        return Response({'http_method':'GET'})

    def update(self, request, pk=None):
        """Handles updating an object."""

        return Response({'http_method':'PUT'})

    def partial_update(self, request, pk=None):
        """Handles updating part of an object. """
        return Response({'http_method':'PATCH'})
    def destroy(self, request, pk=None):
        """Handles removing an object."""
        return Response({'http_method':"DELETE"})
```

http://127.0.0.1:8000/api/hello-viewset/1/



Hello Instance

Test API ViewSet.

```
DELETE /api/hello-viewset/1/
```

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

{
    "http_method": "DELETE"
}
```

Media type: application/json

Content:

Plan our profiles-ap

Basic Requirements

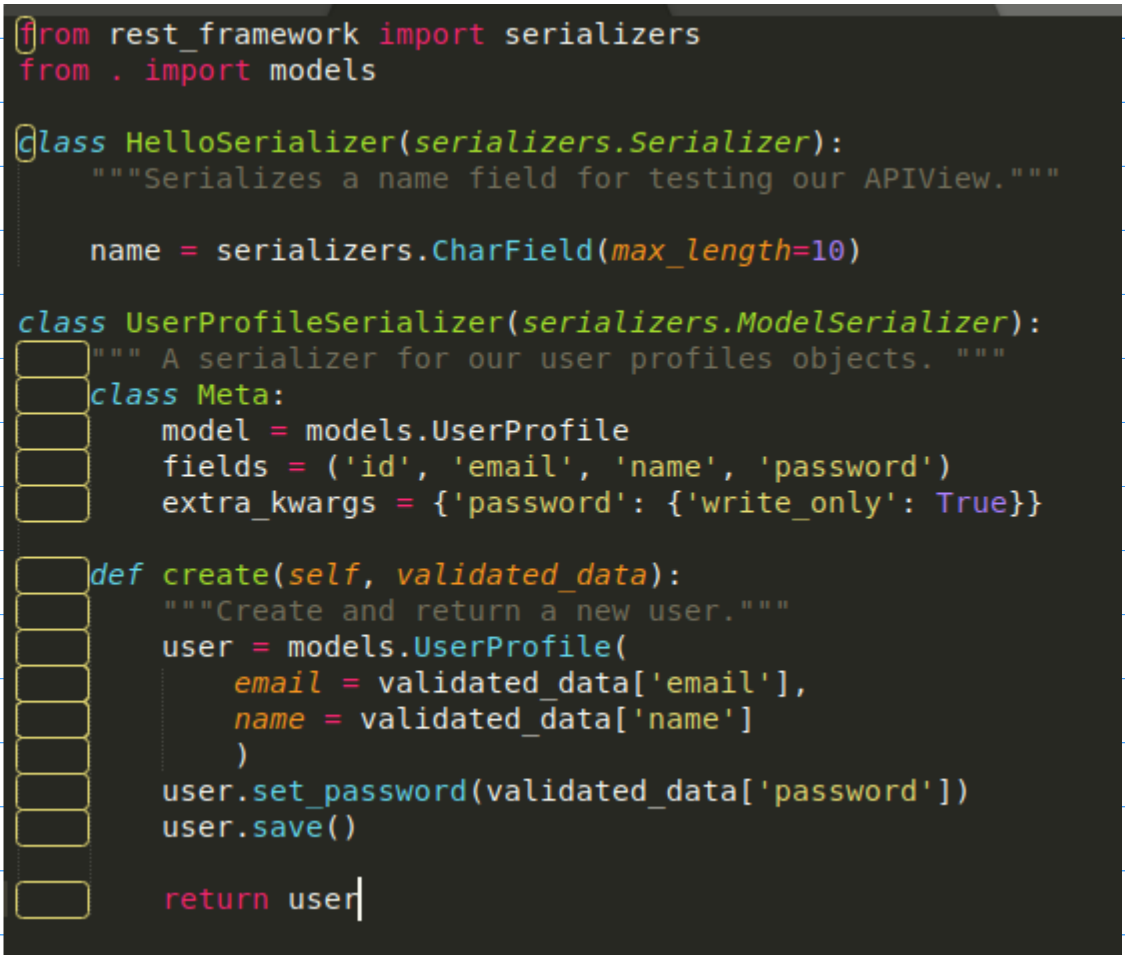
-- Create new profile
        -- Validate profile data
-- List existing profiles
        --- Search for profiles
-- View specific profile
-- Update my profile of logged in user
                -- Update name/email address
                -- Change password
-- Delet profile

URLs for our API:

-- /api/profile/ -- list all profiles
        - GET (list profiles)
        - POST (create profile)

- /api/profile/<profile_id>/ - manage specific profile

- GET (view specific profile)
- PUT/ PATCH (update profile)
- DELETE (remove profile)

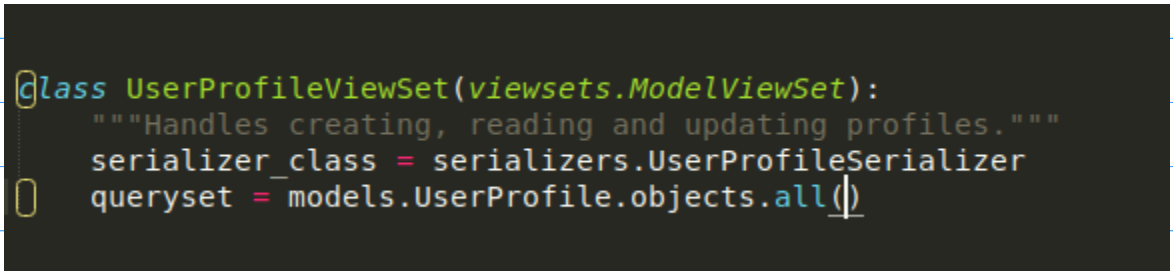


```python
from rest_framework import serializers
from . import models

class HelloSerializer(serializers.Serializer):
    """Serializes a name field for testing our APIView."""

    name = serializers.CharField(max_length=10)

class UserProfileSerializer(serializers.ModelSerializer):
    """ A serializer for our user profiles objects. """
    class Meta:
        model = models.UserProfile
        fields = ('id', 'email', 'name', 'password')
        extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data):
        """Create and return a new user."""
        user = models.UserProfile(
            email = validated_data['email'],
            name = validated_data['name']
            )
        user.set_password(validated_data['password'])
        user.save()

        return user
```

Password: write_only

What creat func do?

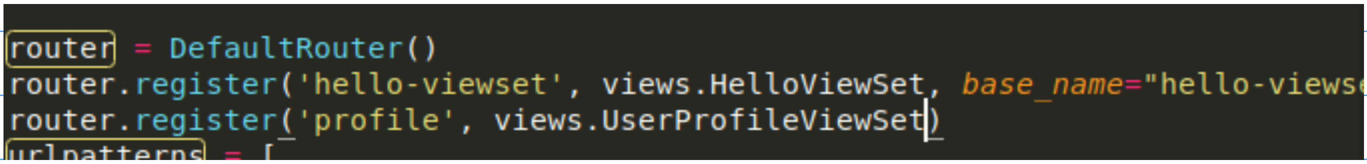
Userprofile viewsets



```python
class UserProfileViewSet(viewsets.ModelViewSet):
    """Handles creating, reading and updating profiles."""
    serializer_class = serializers.UserProfileSerializer
    queryset = models.UserProfile.objects.all()
```

It uses viewsets.ModelViewSet, NOT viewsets.ViewSet as earlier

Register profile viewset with the url router:

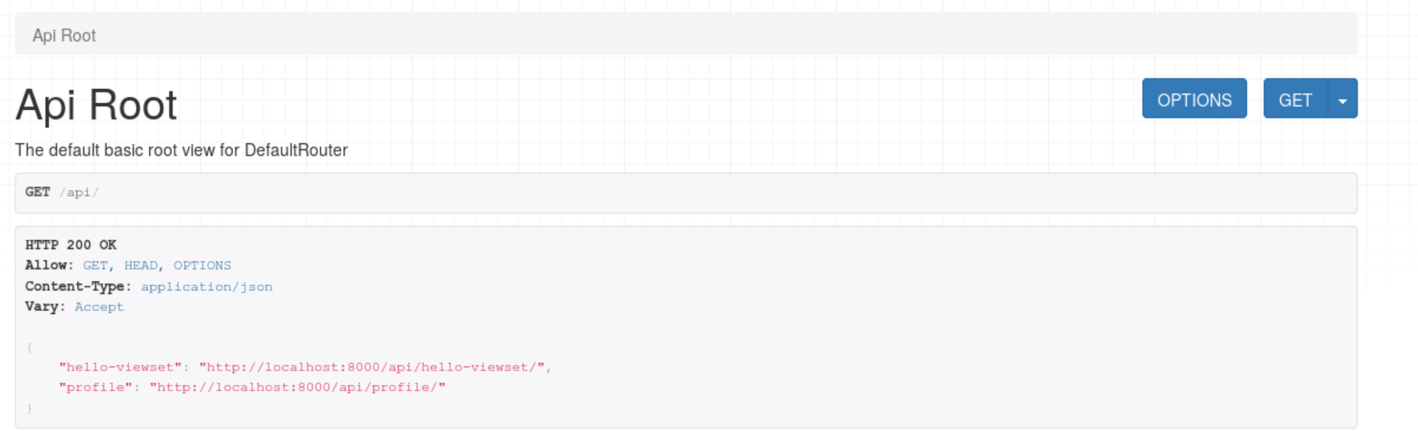


```
router = DefaultRouter()
router.register('hello-viewset', views.HelloViewSet, base_name="hello-viewse
router.register('profile', views.UserProfileViewSet)
urlpatterns = [
```

No need of base_name in model viewsets, rest framework can automatically figure out from the model.

http://localhost:8000/api/

Look that we have a new entry called "profile"



Api Root

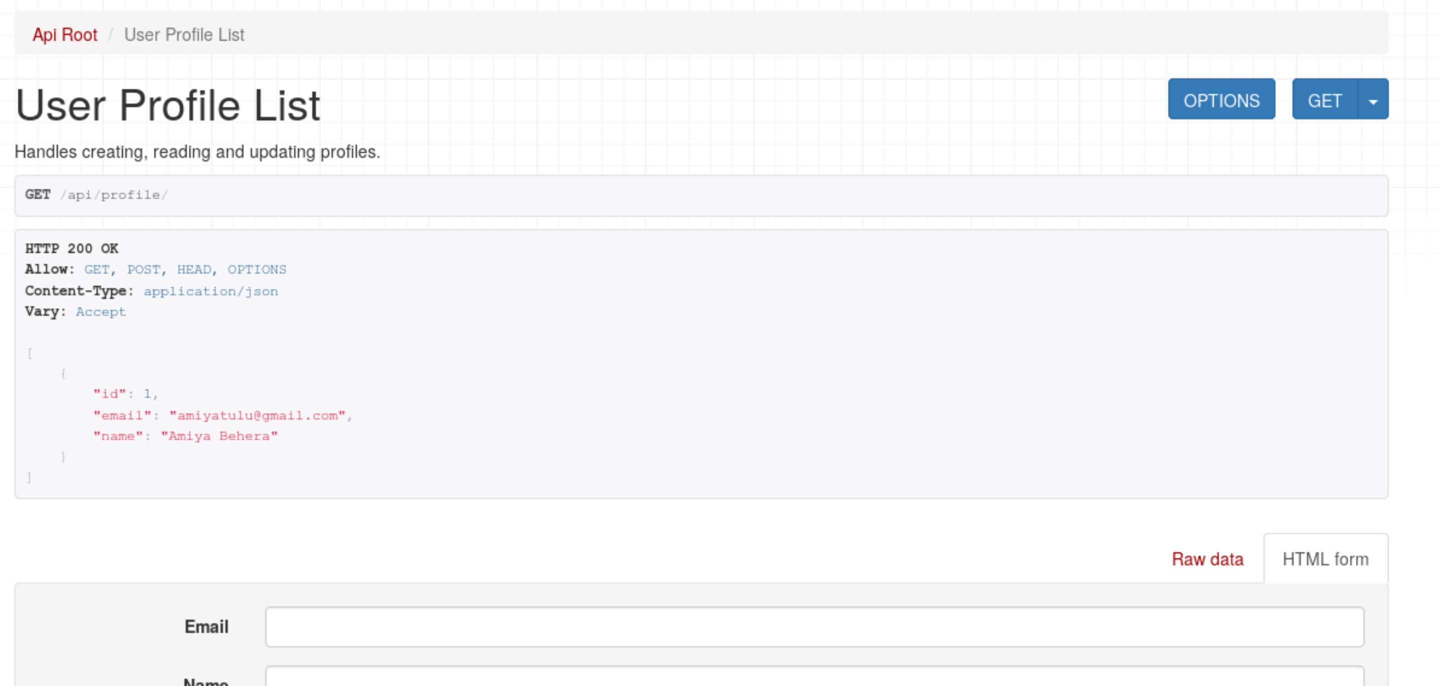
Api Root

The default basic root view for DefaultRouter

```
GET /api/
```

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

{
    "hello-viewset": "http://localhost:8000/api/hello-viewset/",
    "profile": "http://localhost:8000/api/profile/"
}
```

http://localhost:8000/api/profile/
Profile list view



Api Root / User Profile List

User Profile List

Handles creating, reading and updating profiles.

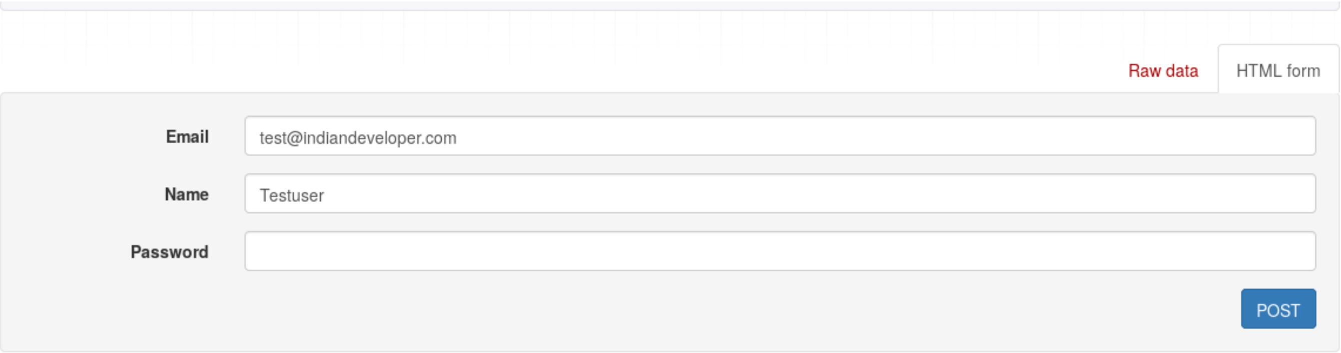
```
GET /api/profile/
```

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

[
    {
        "id": 1,
        "email": "amiyatulu@gmail.com",
        "name": "Amiya Behera"
    }
]
```

Email

Name

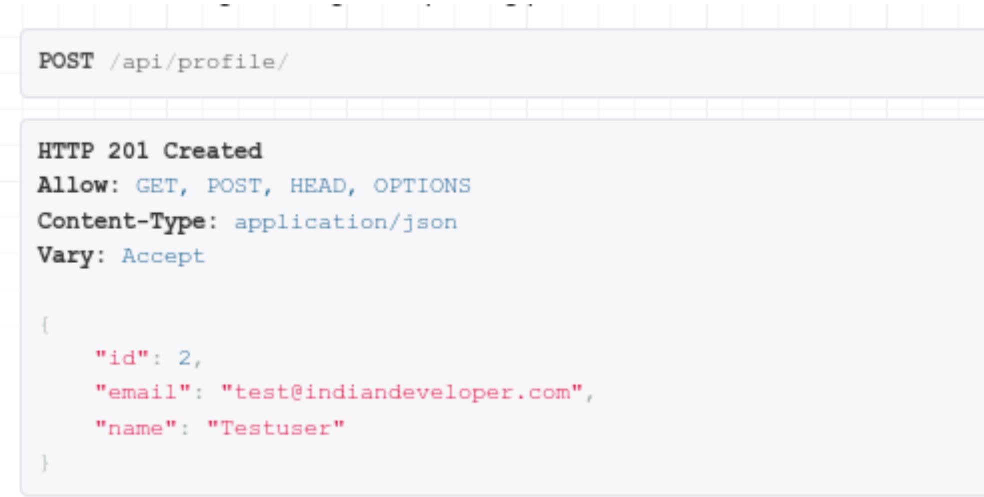
Post a user profile: Email, Name and Password



Email: test@indiandeveloper.com

Name: Testuser

Password:

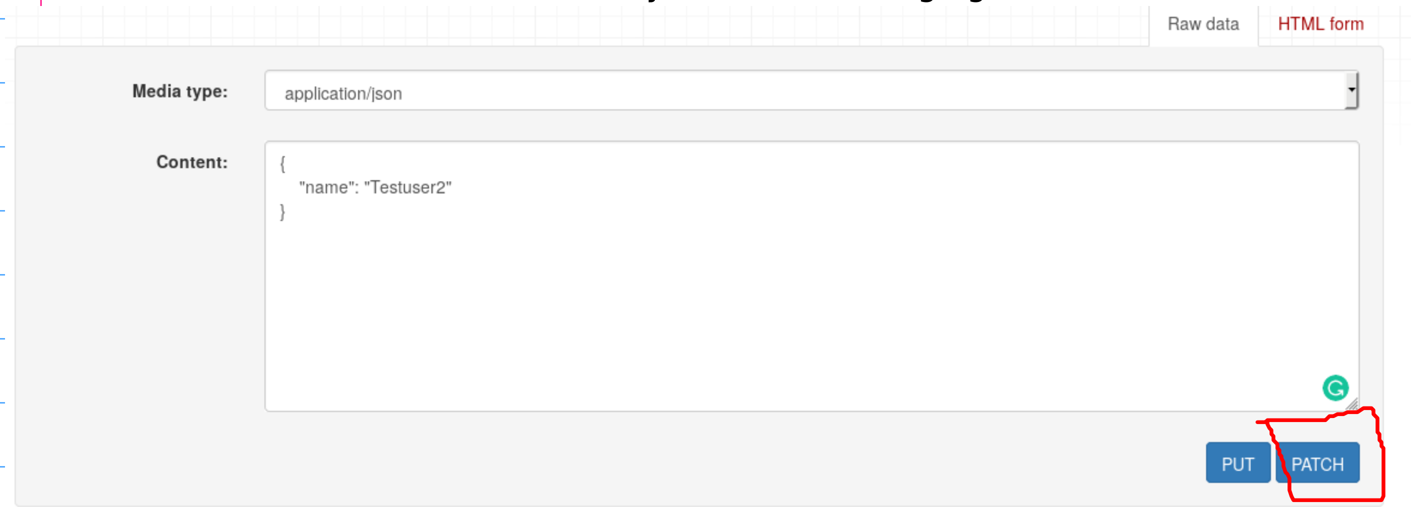


```
POST /api/profile/

HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 2,
    "email": "test@indiandeveloper.com",
    "name": "Testuser"
}
```

See that it doesn't return password, as its write_only

http://localhost:8000/api/profile/2/

Use PATCH to update the profile with id 2
Remove id and email field, because you are not changing it.

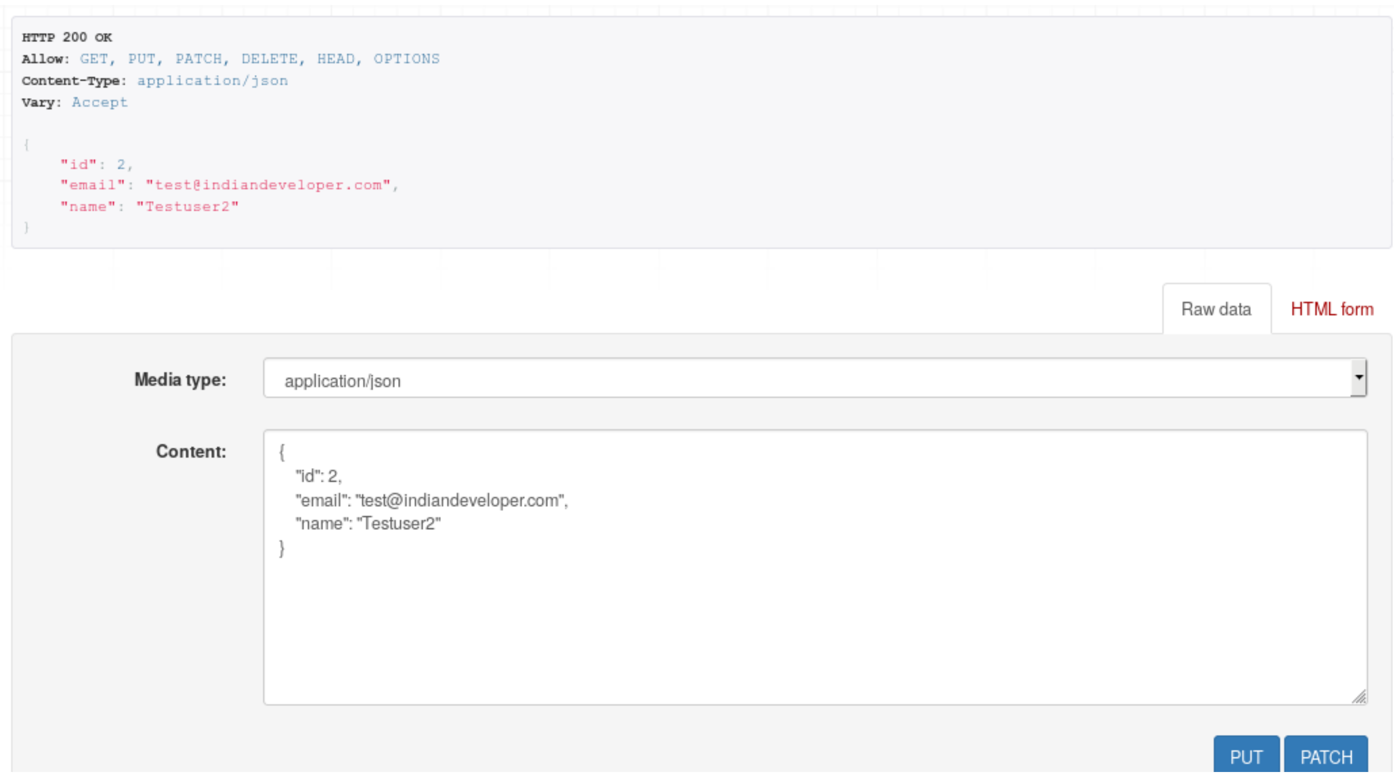


Media type: application/json

Content:

```
{
    "name": "Testuser2"
}
```

Result:



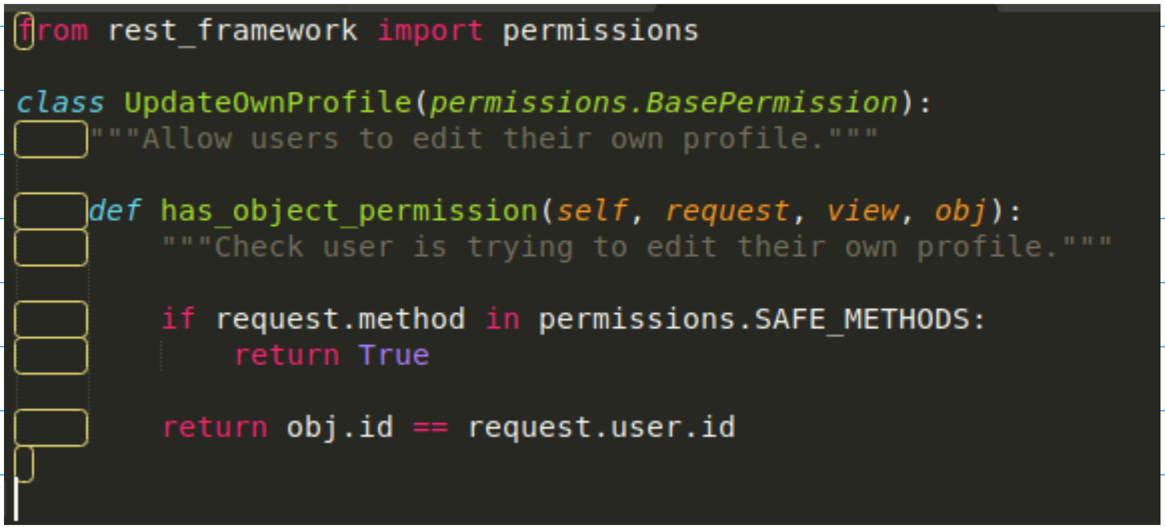
```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 2,
    "email": "test@indiandeveloper.com",
    "name": "Testuser2"
}
```

Media type: application/json

Content:

```
{
  "id": 2,
  "email": "test@indiandeveloper.com",
  "name": "Testuser2"
}
```

Permission class



```python
from rest_framework import permissions

class UpdateOwnProfile(permissions.BasePermission):
    """Allow users to edit their own profile."""

    def has_object_permission(self, request, view, obj):
        """Check user is trying to edit their own profile."""

        if request.method in permissions.SAFE_METHODS:
            return True

        return obj.id == request.user.id
```

has_object_permission does following things:

the request object includes type of request that is made to the api (e.g. get)
If it comes under SAFE_METHODS, i.e. it is going get request, it will return true
next we gonna check if the user is updating his own profile.