Understand

Managers in

Django Model Layer

Intro of Model Managers

In Django, a model managers is responsible to perform database query operations like create, read, update, delete and provides many more features.

At least one manager, is associate with each model, which is objects.

#pythonworld

```python
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField(max_length=277)

    objects = models.Manager()
```

Manager name

Manager class

Django by default, create this objects as a manager name in every model.

However, you don't need to define this.

Return all student objects

```python
student = Student.objects.all()
```

Manager

Model

Manager Method

buit-in menthod in manager class

#pythonworld

Rename manager name

If you want to use a name other than objects for the Manager, you can rename it on a per-model basis.

```python
from django.db import models

class Student(models.Model):
  name = models.CharField(max_length=100)
  email = models.EmailField(max_length=277)

  student = models.Manager()
```

```python
student = Student.student.all()
```

#pythonworld

Built-in Manager Methods

Django Model Manager class provides a varieties of buit-in methods, that you can use to query your models and perform database operations.

Popular Manager Methods

- all()
- get()
- filter()
- create()
- update()
- count()
- delete()
- exists()
- values()
- values_list()
- exclude()
- bulk_create()
- annotate()
- aggregate()
- order_by()

Custom Manager

Django let's give you feature to create your own custom manager by just inherit the Manager class and even you create custom methods.

Here we create Author and Editor two custom manager, by just inherit Manager class and here we have overrirde the get_queryset method to return the data based on their roles

```python
from django.db import models


class AuthorManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(role="A")


class EditorManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(role="E")


class Person(models.Model):
    choices = (("A","Author"), ("E", "Editor"))

    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    role = models.CharField(max_length=1, choices=choices)

    objects = models.Manager() # default manager

    authors = AuthorManager() # author manager

    editors = EditorManager() # editor manager
```

and here in our model, we have added our custom managers

This will return person whose role is author

```
Person.authors.all()
```

This will return person whose role is editor

```
Person.editors.all()
```

This will return all person whose role is either author or editor

```
Person.objects.all()
```

#pythonworld

For more django
related content

Subscribe

To My Youtube
Channel

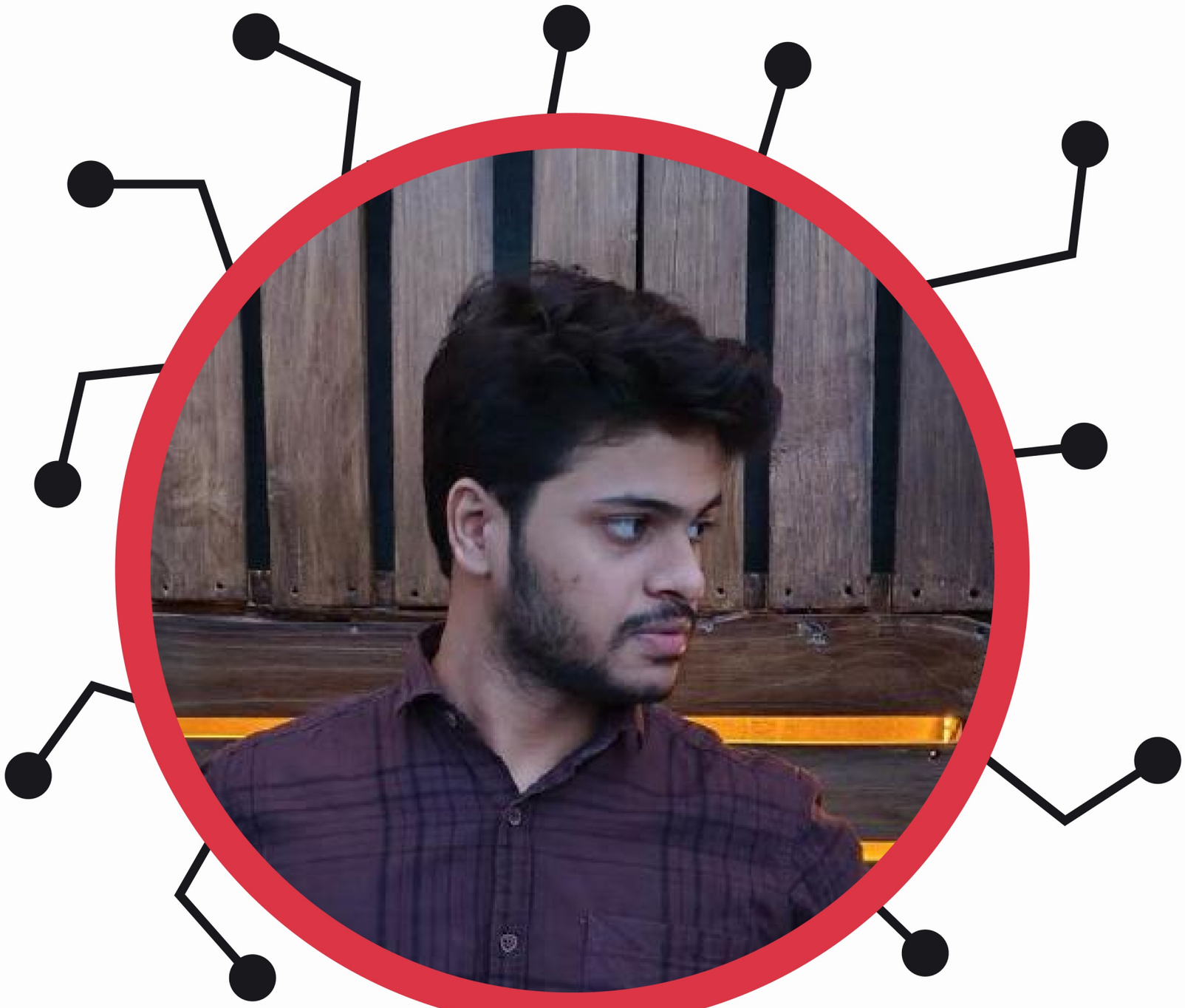
Link in bio

#pythonworld

DID YOU FIND THIS

HELPFUL

Let me know in the comment



Aashish Kumar

Software Engineer

Follow for more ❤️