Understand

Transactions in Django Framework

Intro of Transactions

In Django, transactions are used to maintain the consistency of the database. It ensure all the database operations within the transaction, either success together or failed together.

ACID Properties

These are the most important transactions properties, that ensure that the database transactions are processed consistently, even system failure.

How to implement transactions in django ORM

Django provides the **atomic decorators** or **context manager** to manage transactions in django ORM. Apart from them it also provides **savepoint**, **commits** and **rollback** methods to manage transactions

Let’s create Model

Here we have a simple BankAccount model, that holds the account number and it’s balance.

```
from django.db import models

class BankAccount(models.Model):
    account_number = models.CharField(max_length=20, unique=True)
    balance = models.DecimalField(max_digits=10, decimal_places=2)
```

So, let’s implement transactions on this model

```python
# views.py
from django.db import transaction
from django.shortcuts import get_object_or_404
from django.http import JsonResponse
from .models import BankAccount

@transaction.atomic
def transfer_funds(request, from_account_number, to_account_number, amount):
    try:
        # Get the bank accounts
        from_account = get_object_or_404(
                    BankAccount, account_number=from_account_number
        )
        to_account = get_object_or_404(
                    BankAccount, account_number=to_account_number
        )

        # Check if the balance of the 'from_account' is sufficient
        if from_account.balance < amount:
            return JsonResponse({'error': 'Insufficient balance'}, status=400)

        # Deduct the amount from the 'from_account'
        from_account.balance -= amount
        from_account.save()

        # Add the amount to the 'to_account'
        to_account.balance += amount
        to_account.save()

        return JsonResponse({'message': 'Funds transferred successfully'})
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)
```

Here we have transfer_funds view, that we have wrap with atomic decorator, and here we are transferring the amount to one account to another account, in between this process, if any part of transactions faill, the entire transaction will be rollback

Other important concepts

- Savepoints - It mark a point within a transaction that you can roll back to later

```python
from django.db import transaction
from .models import Student

@transaction.atomic
def my_transaction_function():
    # Initial transaction
    # Savepoint created automatically here

    try:
        # Some database operations
        stu1 = Student.objects.create(name='John')

        # Create a savepoint
        sid = transaction.savepoint()

        # More database operations
        stu2 = Student.objects.create(name='David')

        # Roll back to the savepoint
        transaction.savepoint_rollback(sid)

    except Exception as e:
        # Roll back entire transaction in case of an exception
        transaction.set_rollback(True)
```

rollback to

in case of any exceptions

- Rollback - It allows you to cancel all changes made in a transaction

- Commit - It makes all changes permanent in the database.

```python
from django.db import transaction
from .models import Student

@transaction.atomic
def my_transaction_function():
    try:
        # Some database operations
        stu = Student.objects.create(name='John Doe')

        # Finalize the transaction by committing the changes
        transaction.commit()
    except Exception as e:
        # Roll back entire transaction in case of an exception
        transaction.set_rollback(True)
```

- Nested Transactions - Using context manager we can even implement nested transactions.

Outer transactions

Inner transactions

```python
from django.db import transaction
from .models import Student

@transaction.atomic
def my_view(request):
    stu1 = Student.objects.create(name="John Doe")

    with transaction.atomic():
        # Perform nested operations
        stu2 = Student.objects.create(name="John Doe")
```

#pythonworld

For more django
related content

Subscribe

To My Youtube
Channel

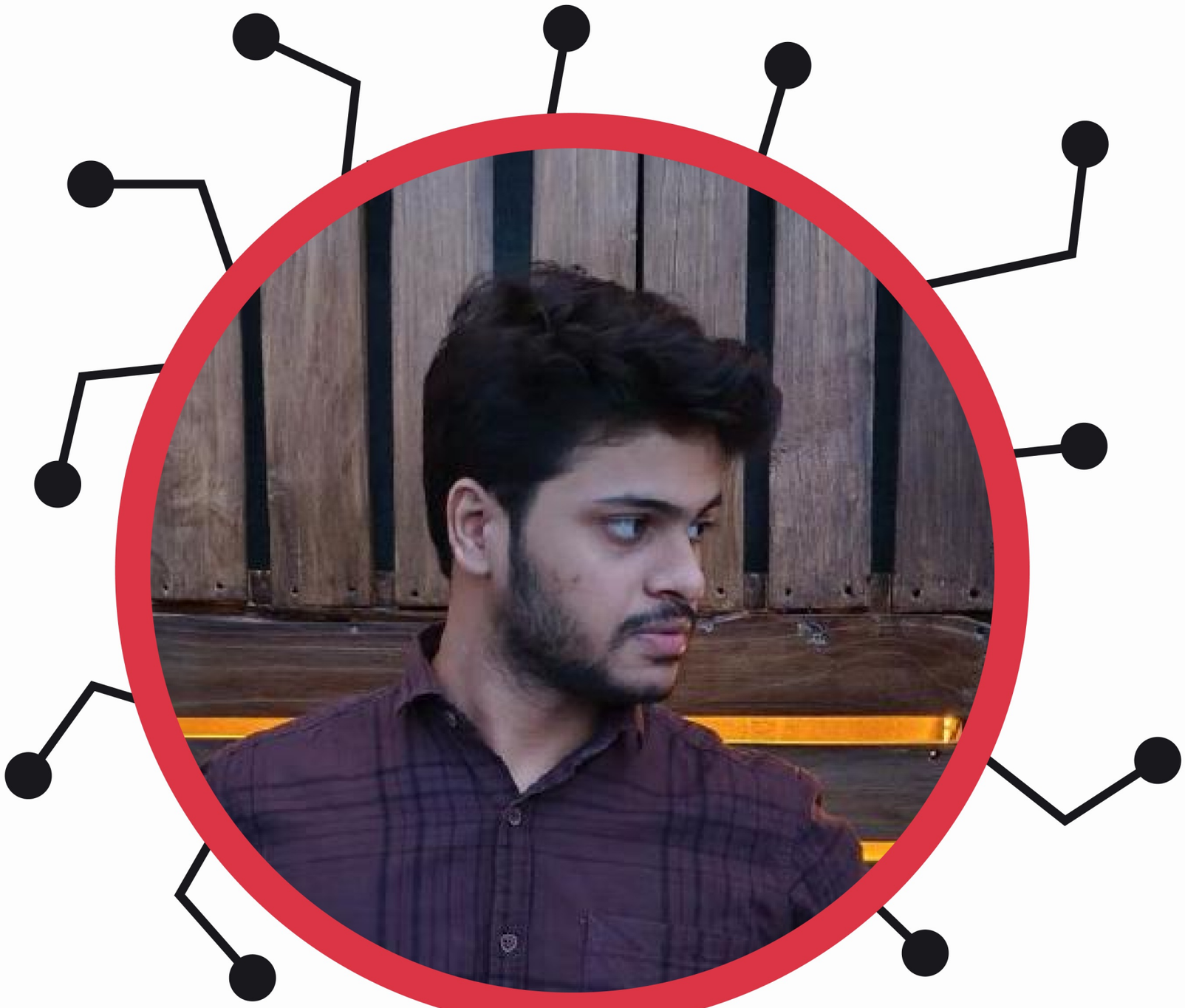
Link in bio

#pythonworld

DID YOU FIND THIS

HELPFUL

Let me know in the comment



Aashish Kumar

Software Engineer

Follow for more ❤️