1. Django's ORM (Object-Relational Mapping)

Definition & purpose

- What is an ORM

ORM stands for Object-Relational Mapping. It's a technique (or layer) in software that allow you to with relational database using the object-oriented paradigm of your programming language - in this case, python.

- Why does Django have an ORM?

Django ORM abstract away much of the SQL you would normally write. Instead of writing SQL queries directly, you use python classes to represent database tables, and Django automatically translates your python code into the necessary SQL.

- Where does it work / How does it function?

The Django ORM work within Django projects. whenever you define a model class in `models.py` effectively defining a table schema.

For example :-

```python
from django.db import models
class Book(models.CharField(max_length=200)
        author = models.charField(max_len=100)
        published_date = models.DateField()
```

- Behind the scenes, Django creates database tables for these classes when run migration.

When you fetch or create data, the ORM converts that to SQL. For example :

```
python
Book.objet.all()
Book.object.create(title = "Django Unleashed",
                   author = "Andrew Pinkham")
```

- The ORM also handles relationships

Benifit of Django's ORM.

1) Less SQL : you do not have to write new raw SQL statement for most common operation.

2) Database portability : If you switch from one database to another, your code is more easily portable.

3) Validation & security. : The ORM help prevent common issue such as SQL injection by escaping parameter automatically.

4) Easier to maintain : Your model class definition are clean and can be version-controlled well.

2. Middleware

* General Definition of Middleware.

- Middleware generally refers to software that site between different layer or component of an application. It can intercept and sometime modify the data or request/respo flowing between these components.

- Primary Purpose.
- Facilitates communication between separate system.
- Add common functionality or cross-cutting concerns without each system needing to implement it separately.

Examples of middleware in a broader software context :

- API Gateways :- That intercept request and add security check or rate-limiting.

- Message-oriented middleware :- that brokes msg betn multiple services.

- Transaction management :- Layer in enterprises apps that ensure data consistency.

Middleware in Django

- In Django, middleware is a series of hooks that process req & responses globally.
- The typicaly req-response lifecycle in Django

1. Request enters Django
2. Middleware runs before Django matches the request to a view.
3. Django calls the appropriate view if the req is allowed through.
4. After the view return a response, that response gose back through midleware again.

- common ex of django middleware.

- Authenticationmiddleware :
  Associate users with req.
- Sessionmiddleware :
  Mages user session access req.
- CRF middleware :
  Add common against Req forgency
- Securitymiddleware :
  Add common security headers.

The concept is similar across frameworks but the implementation details vary. essentially, its about hooking into the flow of data

3) Migrations and migrate in django.

Overview.

Django has a built-in system to handle changes to your database to your base schema over time. This system consists of main steps/ commands :

1. makemigrations :- Create migration files based on change you've made to your models.

2. migrate :- Applies those migration file to the actual database.

what is a migration ?

A migration is a python file that describes the describe the change you want made to your database schema creating a new table, adding a new column, or changing a field

#books / migrations / 0001_initial.py

```
from django.db import migration, model
class migration(migrations.migration):
initial = True
dependencies = [
    ]
operation = [
  migration.createmodel (
      name = 'book'
       field = [
```

```
        ('id', models.AutoField(primary_key=True)),
        ('title', models.CharField(max_length=200)),
        ('author', models.CharField(max_length=100)),
        ('published_date', models.DateField()),
    ],
],
]
```

This tells Django to create a new table named 'Book' with the defined field.

What is the 'migrate' process?

- The migrate command reads the migration files and applies them to your database. This means :

1. If a new table is specified, it creates it.
2. If a field is added, it modifies the tables the structure to add that column
3. If a column is renamed or removed, it performs those operation, etc.

Difference Between migration & migrate.

- migration" :- Usually refers to the definition of the changes you want in your database It's the plan blueprint.

- You create or update this plan by running python manage.py makemigrations.

- This command compares your current model definitions with the previously recorded state in your migrations, then writes out new migration files if there are difference.

- Migrate : Refers to actually applying those migration files to the database.

- You do this by running 'python manage.py migrate'.

★ Why two steps?

By having a sep step to create migration files, Django lets you:

1. Generate migration files in you version control.

2. Inspect or edit them before they are applied to your database

3. Roll back or Roll forward to specific version The migrate step them safely aplied those change. This separation provides better control and conspareney about what database change are being made.

Summary

1. Django's ORM
- An abstraction layer to interact with database through python classes and method.

- Helps avoid raw sql and supports multiple databases easily.

2. Middleware
- Software that intercepts req/responses to add cross-cutting functionality.

- In Django, middleware are classes that process req/responses as they travel in and out of Django view layer.

3. Migrations vs. migrate.
- migrations are python file that describe change to the database schema. You generate or update them with 'python manag.py makemigration.

- migrate is the command that applies these migration files to the actual database.

#1. Middleware (Simple Explanation)

Middleware is like a security checkpoint in a django operation. It sits between the user's req and the respone sent by the server. It can modify, block, or add extra features to request and responses.

Expl use of middleware :

- Checking if a user is logged in before allowing access.

- Protecting against security threats like CFSR attacks.

- Recording logs of all req and responses for debugging.

- Adding extra data the to req or reponses.

In simple word, middleware act as a filter that control what happen to the req before it main logic and what happen to the responses before it is sent back to the user.

#2. ORM (object-Relational Mapping).

ORM is a way to interact with database using python object instead of writing SQL queries. Instead of writing command like 'select * From users', you just ask Django ORM to get all user which python.

#3. HTTP (Simple Explanation)

HTTP is a the language that web browsers and use to communicate. whenever you open a website, your browser sends an HTTP req, and the server responds that an HTTP response.

How HTTP works in Django?

- When a user visits a webpage their browser send a request to the Django server.

- Django processes the request and decides what data to send back.

- The server then sends an HTTP response, which can be a website, data, or an error message.

Common HTTP Method :

- GET → Used to request data from the server

- Post → Used to send data to the server.

- Put/Patch → Used to update existing data.

- DELETE → Used to delete data.

In simple word HTTP is like a msg system where browser and server exchange req & resp. to webpage send forms, or update data.

1. Django's ORM (Object-Relational Mapping)

Definition & purpose.

- What is an ORM

ORM stands for Object-Relational Mapping. It's a technique (or layer) in software that allow you to with relational database using the object-oriented paradigm of your programming language - In this case, python.

- Why does Django have an ORM?

Django ORM abstract away much of the SQL you would normally write. Instead of writing SQL queries directly, you use python classes to represent database tables, and Django automatically translates your python code into the necessary SQL.

- Where does it work / How does it function?

The Django ORM work within Django projects. whenever you define a model class in `models.py` effectively defining a table schema.

For example :-

```python
from django.db import models
class Book (models.CharField (max_length = 200)
            author = models.charfield (max_len=
            published_date = models.Datefield ()
```

- Behind the scenes, Django creates database tables for these classes when run migration.

when you fetch or create data, the ORM converts that to SQL. For example :

```python
Book.objet.all()
Book.object.create(title="Django Unleashed",
                   author="Andrew Pinkham")
```

- The ORM also handles relationships

Benifit of Django's ORM.

1) Less SQL : you do not have to write new raw SQL statement for most common operation.

2) Database portability : If you switch from one database to another, your code is more easily portable.

3) Validation & security. : The ORM help prevent common issue such as SQL injection by escaping parameter automatically.

4) Easier to maintain : Your model class definition are clean and can be version-controlled well.

2. Middleware

* General Definition of Middleware.

- Middleware generally refers to software that site between different layer or component of an application. It can intercept and sometime modify the data or request/respo flowing between these components.

- Primary Purpose
- Facilitates communication between separate system.
- Add common functionality or cross-cutting concerns without each system needing to implement it separately.

Examples of middleware in a broader software context :

- API Gateways :- That intercept request and add security check or rate-limiting.
- Message-oriented middleware :- that brokes msg betn multiple services.
- Transaction management :- Layer in enterprises apps that ensure data consistency.

Middleware in Django

- In Django, middleware is a series of hooks that process req & responses globally.
- The typicaly req-response lifecycle in Django

1. Request enters Django
2. Middleware runs before Django matches the request to a view.
3. Django calls the appropriate view if the req is allowed through
4. After the view return a response, that response gose back through midleware again.

- common ex of django middleware.

- Authenticationmiddleware :
  Associate users with req.
- Sessionmiddleware :
  Mages user session access req.
- CRF middleware :
  Add common against Req forgency
- Securitymiddleware :
  Add common security headers.

The concept is similar across frameworks but the implementation details vary, essentially, its about hooking into the flow of data

3) Migrations and migrate in django.

Overview.

Django has a built-in system to handle changes to your database to your base schema over time. This system consists of main steps/ commands:

1. makemigrations :- Create migration files based on change you've made to your models.

2. migrate :- Applies those migration file to the actual database.

what is a migration?

A migration is a python file that describes the describe the change you want made to your database schema creating a new table, adding a new column, or changing a field

#books / migrations/0001_initial.py

```
from django.db import migration, model
class migration(migrations.migration):
initial = True
dependencies = [
    ]
operation = [
  migration.createmodel (
      name='book'
        field = [
```

```
        ('id', models.AutoField(primary_key=True),
        ('title', models.CharField(max_length=200)),
        ('author', models.CharField(max_length=100)),
        ('published_date', models.DateField()),
    ],
],
]
...
```

This tells Django to create a new table named 'Book' with the defined field.

* What is the 'migrate' process?
- The migrate command reads the migration files and applies them to your database. This means :

1. If a new table is specified, it creates it.
2. If a field is added, it modifies the tables the structure to add that column.
3. If a column is renamed or removed, it performs those operation, etc.

* Difference Between migration & migrate.
- migration" :- Usually refers to the definition of the changes you want in your database. It's the plan blueprint.

- You create or update this plan by running python manage.py makemigrations.

- This command compares your current model definitions with the previously recorded state in your migrations, then writes out new migration files if there are difference.

- Migrate : Refers to actually applying those migration files to the database.

- You do this by running 'python manage.py migrate'.

* Why two steps?

By having a sep step to create migration files, Django lets you:

1. Generate migration files in you version control.

2. Inspect or edit them before they are applied to your database

3. Roll back or Roll forward to specific version The migrate step then safely aplied those change. This separation provides better control and conspareny about what databas change are being made.

SAFFRON | Page No.
Date / /

Summary

1. Django's ORM

- An abstraction layer to interact with database through python classes and method.

- Helps avoid raw sql and supports multiple databases easily.

2. Middleware

- Software that intercepts req/responses to add cross-cutting functionality.

- In Django, middleware are classes that process req/responses as they travel in and out of Django view layer.

3. Migrations vs. migrate.

- migrations are python file that describe change to the database schema. You generate or update them with 'python manag.py makemigration.

- migrate is the command that applies these migration files to the actual database.

#1. Middleware (Simple Explanation)

Middleware is like a security checkpoint in a django operation. It sits between the user's req and the respone sent by the server. It can modify, black, or add extra features to request and responses.

Expl use of middleware :

- Checking if a user is logged in before allowing access.

- Protecting against security threats like CFSR attacks.

- Recording logs of all req and responses for debugging.

- Adding extra data the to req or reponses.

In simple word, middleware act as a filter that contral what happen to the req before it main logic and what happen to the responses before it is sent back to the user.

#2. ORM (object-Relational Mapping).

ORM is a way to interact with database using python object instead of writing SQL queries. Instead of writing command like `select * From users`, you just ask Django ORM to get all user which python.

#3. HTTP (Simple Explanation)

HTTP is a the language that web browseas and use to communicale. whenever you open a website, your browser sends an HTTP req, and the server responds that an HTTP response.

How HTTP works in Django?

- When a user visits a webpage their browser send a request to the Django server.

- Django processes the request and decides what data to send back.

- The server then sends an HTTP response, which can be a website, data, or an error message.

Common HTTP Method :

- GET → Used to request data from the server

- Post → Used to send data to the server.

- Put/Patch → Used to update existing data.

- DELETE → Used to delete data.

In simple word HTTP is like a msg system where browser and server exchange req & resp. to webpage send forms, or update data.