РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Лекция 1.

- Введение
- История появления
- Установка и настройка Django
- Структура проекта Django
- Структура приложения Django
- Django ORM (Object-Relational Mapping)
- Миграции в Django ORM
- Технология MVT (Model-View-Template)

ВВЕДЕНИЕ

Что такое Джанго?

Django — это высокоуровневый веб-фреймворк Python, который позволяет быстро разрабатывать безопасные и удобные в обслуживании веб-сайты. Созданный опытными разработчиками, Django берет на себя большую часть хлопот веб-разработки, так что вы можете сосредоточиться на написании своего приложения без необходимости изобретать велосипед. Он бесплатный и с открытым исходным кодом, имеет процветающее и активное сообщество, отличную документацию и множество вариантов бесплатной и платной поддержки.

ВВЕДЕНИЕ

Django — это мощный веб-фреймворк, который позволит вам создать программное обеспечение, отвечающее широкому спектру потребностей. Вот ключевые его преимущества:

- **Полный пакет**

Django следует принципу «батарейки включены», предлагая все необходимые инструменты «из коробки». Это означает, что компоненты работают согласованно и предоставляются с актуальной документацией. Вам не нужно искать сторонние решения, так как всё необходимое уже встроено.

- **Универсальность**

С помощью Django можно создать любой тип веб-сайта, от систем управления контентом и новостных порталов до социальных сетей. Он поддерживает различные клиентские фреймворки и может выдавать данные в форматах HTML, JSON, XML и других. Django легко расширяется и настраивается под любые требования.

ВВЕДЕНИЕ

- **Безопасность**

Фреймворк обеспечивает высокую степень безопасности, защищая от распространённых атак, таких как SQL-инъекции и межсайтовый скриптинг. Django использует надёжные методы управления паролями, применяя хэширование и исключая возможность их прямого хранения. Кроме того, встроенные механизмы защиты от кликджекинга и подделки запросов делают его безопасным выбором для разработчиков.

- **Масштабируемость**

Благодаря компонентной архитектуре, Django легко масштабируется. Он подходит для высоконагруженных проектов, позволяя расширять инфраструктуру — от серверов баз данных до кэширования. Среди успешных примеров его масштабирования — Instagram и Disqus.

ВВЕДЕНИЕ

- **Поддерживаемость**

Код в Django структурирован по принципам, способствующим его поддерживаемости и повторному использованию. В частности, следование принципу «Не повторяйся» (DRY) помогает минимизировать дублирование кода, а разделение на модули и приложения упрощает его сопровождение.

- **Портативность**

Поскольку Django написан на Python, он работает на различных платформах — Linux, Windows, macOS. Это делает его гибким в выборе серверной инфраструктуры. Многие провайдеры предлагают специализированные решения для хостинга Django-сайтов, что упрощает развёртывание приложений.

Django — это идеальный инструмент для разработки надёжных и гибких веб-приложений.

ИСТОРИЯ ПОЯВЛЕНИЯ

Django начал свое развитие в период с 2003 по 2005 год, когда группа веб-разработчиков работала над созданием и поддержкой газетных сайтов. По мере работы над несколькими проектами, команда начала замечать, что многие элементы кода и шаблонов могут быть использованы повторно. Этот процесс привел к созданию единой архитектуры для веб-разработки, которая в итоге была открыта миру как проект "Django" в июле 2005 года.

С момента своего первого стабильного релиза версии 1.0 в сентябре 2008 года, Django непрерывно эволюционировал и совершенствовался. Каждый новый релиз привносил новые возможности и улучшения, такие как поддержка дополнительных типов баз данных, усовершенствованные механизмы кэширования, новые шаблонизаторы, а также внедрение «общих» функций и классов представлений, которые значительно сокращают объем кода для выполнения типичных задач.

ИСТОРИЯ ПОЯВЛЕНИЯ

К концу 2023 года с выходом версии 5.0, Django утвердился как один из ведущих фреймворков с открытым исходным кодом, используемый тысячами разработчиков по всему миру. Благодаря активному сообществу участников, фреймворк поддерживает высокую скорость разработки и остается актуальным для современных потребностей веб-разработки. Несмотря на то, что его корни восходят к созданию новостных порталов, сегодня Django — это универсальный инструмент, который подходит для разработки любых типов веб-приложений, от простых блогов до сложных корпоративных систем.

Что общего между youtube, google search, dropbox и instagram ?

Да, все они сделаны с использованием фреймворка Django!

УСТАНОВКА И НАСТРОЙКА DJANGO

1. Выбор редактора кода :

Рекомендую использовать один из следующих редакторов для работы с Django:

- **Visual Studio Code** : бесплатный, многофункциональный редактор с поддержкой расширений для Python.

    Плюсы: простота использования, встроенный терминал, множество расширений.

    Минусы: иногда может потреблять много ресурсов.

Visual Studio Code

УСТАНОВКА И НАСТРОЙКА DJANGO

- **PyCharm** : интегрированная среда разработки, специально адаптированная для Python.

  Плюсы: мощные инструменты для отладки и рефакторинга.

  Минусы: может медленно работать на старых машинах, Django хорошо работает только на платной версии.

- **Sublime Text** : легкий и быстрый редактор, поддерживающий множество языков.

  Плюсы: высокая производительность, возможность настройки.

  Минусы: отсутствие встроенных инструментов отладки.

УСТАНОВКА И НАСТРОЙКА DJANGO

2. **Установка Python :**

Убедитесь, что у вас установлена последняя версия Python (рекомендуется версия 3.8 или выше). Вы можете скачать Python с официального сайта https://www.python.org/downloads/

После установки проверьте версию Python в терминале:

bash

python --version

УСТАНОВКА И НАСТРОЙКА DJANGO

3. **Установка Django :**

Настройте Django с помощью пакетного менеджера pip. В терминале выполните команду:

bash

pip install django

4. Проверьте успешность установки, выполнив команду:

bash

django-admin --version

УСТАНОВКА И НАСТРОЙКА DJANGO

5. **Создание нового проекта Django:**

Для создания нового проекта воспользуйтесь командой:

django-admin startproject <имя_проекта>

bash

django-admin startproject myproject

Перейдите в созданную директорию проекта:

bash

cd myproject

УСТАНОВКА И НАСТРОЙКА DJANGO

6. **Запуск разработки сервера :**

Для работы проверки проекта запустите разработку сервера:

bash

python manage.py runserver

Откройте браузер и перейдите по адресу , чтобы увидеть стартовую страницу Django.
http://127.0.0.1:8000/

УСТАНОВКА И НАСТРОЙКА DJANGO

7. **Настройка виртуального окружения :**

Рекомендуем использовать виртуальные окружения для управления зависимостями вашего проекта. Создайте виртуальное окружение с помощью следующей команды:

bash

python -m venv venv

Активируйте виртуальное окружение:

В Windows:

bash

venv\Scripts\activate

УСТАНОВКА И НАСТРОЙКА DJANGO

8. **Создание приложения :**

Внутри вашего проекта создайте новое приложение с помощью команды:

bash

python manage.py startapp <имя_приложения>

Например, для создания приложения «блог»:

bash

python manage.py startapp blog

УСТАНОВКА И НАСТРОЙКА DJANGO

9. **Регистрация приложений :**

Добавьте ваше приложение в файл вашего проекта, указав его имя в списке : settings.py INSTALLED_APPS

питон

```
INSTALLED_APPS = [
    ...
    'blog',
    ...
]
```

СТРУКТУРА ПРОЕКТА DJANGO

При создании нового проекта Django с помощью команды **django-admin startproject**, автоматически создается набор файлов и директорий. Эти компоненты формируют базовую структуру проекта, обеспечивая четкое разделение логики и управление проектом.

```
myproject/
|
├── manage.py              # Скрипт для управления проектом
├── myproject/             # Папка проекта
|   ├── __init__.py        # Пакетный файл Python
|   ├── settings.py        # Конфигурации проекта
|   ├── urls.py            # Основной файл маршрутизации URL
|   ├── wsgi.py            # Конфигурация для WSGI-сервера
|   └── asgi.py            # Конфигурация для ASGI-сервера
```

СТРУКТУРА ПРОЕКТА DJANGO

1. **manage.py**

Этот файл является основным скриптом для управления вашим проектом.

Назначение:

- Запуск сервера разработки: **python manage.py runserver**.
- Применение миграций: **python manage.py migrate**.
- Создание приложений: **python manage.py startapp <app_name>**.

Примечание: Этот файл позволяет работать с проектом, не указывая полный путь к настройкам.

СТРУКТУРА ПРОЕКТА DJANGO

2. **Папка проекта (например, myproject/)**

Внутри проекта создается папка с именем проекта, содержащая файлы, отвечающие за конфигурацию и маршрутизацию.

- **__init__.py**

Указывает Python, что данная директория является пакетом.

Назначение: Этот файл может быть пустым, но необходим для корректного импорта других модулей проекта.

- **settings.py**

Описание: Файл настроек проекта, в котором указываются все ключевые параметры конфигурации.

СТРУКТУРА ПРОЕКТА DJANGO

Основные параметры:

INSTALLED_APPS: список установленных и активированных приложений.

DATABASES: конфигурация базы данных.

TEMPLATES: настройка шаблонов для фронтенда.

STATIC_URL: URL для доступа к статическим файлам.

MIDDLEWARE: список промежуточного ПО (middleware), обрабатывающего запросы на разных этапах.

- **urls.py**

  Описание: Основной файл маршрутизации (URL-конфигурации) для всего проекта.

  Назначение: Связывает URL-адреса с соответствующими представлениями (views). Позволяет управлять тем, какие страницы будут отображаться по определенным адресам.

СТРУКТУРА ПРОЕКТА DJANGO

Пример кода:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
]
```

СТРУКТУРА ПРОЕКТА DJANGO

- **wsgi.py**

Конфигурация для развертывания проекта с использованием WSGI (Web Server Gateway Interface).

Назначение: Этот файл используется для взаимодействия вашего проекта с веб-сервером на уровне production. Например, если вы используете серверы Apache или Nginx.

- **asgi.py**

Конфигурация для развертывания проекта с использованием ASGI (Asynchronous Server Gateway Interface).

Назначение: Этот файл используется для асинхронных запросов в более современных приложениях. Подходит для работы с WebSocket или другими асинхронными функциями.

СТРУКТУРА ПРОЕКТА DJANGO

Понимание базовой структуры Django-проекта — это первый шаг к эффективной разработке. Эти файлы и каталоги помогают четко разделить логику, настройки и взаимодействие с веб-сервером, создавая удобную для масштабирования и поддержки архитектуру.

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

Приложение в Django — это самостоятельный компонент, отвечающий за определенный функционал. Одно приложение может использоваться в нескольких проектах, что позволяет легко масштабировать и переиспользовать код. Команда **python manage.py startapp <app_name>** создает типовую структуру приложения.

```
myapp/
│
├── admin.py          # Регистрация моделей в панели администратора
├── apps.py           # Конфигурация приложения
├── models.py         # Определение моделей данных
├── views.py          # Определение представлений (views)
├── urls.py           # Маршрутизация для приложения
├── tests.py          # Тесты для приложения
├── migrations/       # Миграции базы данных
│   └── __init__.py   # Файл для инициализации каталога миграций
├── templates/        # HTML-шаблоны
└── static/           # Статические файлы (CSS, JS, изображения)
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **apps.py**

Файл конфигурации приложения.

Назначение: Содержит настройки приложения и используется для его регистрации в проекте.

Пример: В проекте зарегистрировано приложение в settings.py через INSTALLED_APPS. Тогда файл apps.py будет содержать следующий код:

```
from django.apps import AppConfig

class MyAppConfig(AppConfig):
    name = 'myapp'
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **models.py**

Файл, в котором определяются модели данных.

Назначение: Здесь создаются классы моделей, которые представляют таблицы базы данных.

Пример модели:

```
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    published_date = models.DateTimeField(auto_now_add=True)
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **views.py**

Файл, в котором определяются представления (views).

Назначение: Представления обрабатывают запросы и возвращают ответы, такие как HTML-страницы или данные в формате JSON.

Пример:

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **urls.py**

Файл маршрутизации URL-адресов для приложения.

Назначение: Связывает URL-адреса с представлениями (views).

Пример:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
]
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **admin.py**

Файл для регистрации моделей в административной панели Django.

Назначение: Позволяет управлять моделями через встроенную панель администратора.

Пример:

```
from django.contrib import admin
from . models import Article

admin.site.register(Article)
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

migrations/

Каталог, где хранятся файлы миграций базы данных.

Назначение: Миграции управляют изменениями структуры базы данных, синхронизируя её с моделями в приложении.

Пример миграции:

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

```
# 0001_initial.py
from django.db import migrations, models

class Migration(migrations.Migration):
    initial = True

    operations = [
        migrations.CreateModel(
            name='Article',
            fields=[
                ('id', models.AutoField(auto_created=True)),
                ('title', models.CharField(max_length=100)),
            ],
        ),
    ]
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **tests.py**

Файл для написания тестов приложения.

Назначение: Содержит тесты для проверки корректности работы кода.

Пример теста:

```
from django.test import TestCase
from .models import Article

class ArticleTestCase(TestCase):
    def test_create_article(self):
        article = Article.objects.create(title="Test Article", content="Content")
        self.assertEqual(article.title, "Test Article")
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **templates/**

Каталог для хранения HTML-шаблонов приложения.

Назначение: Хранит файлы, которые будут отображены пользователям в ответ на запросы.

Пример шаблона:

html

```
<!-- home.html -->
<h1>Welcome to Home Page</h1>
```

СТРУКТУРА ПРИЛОЖЕНИЯ DJANGO

- **static/**

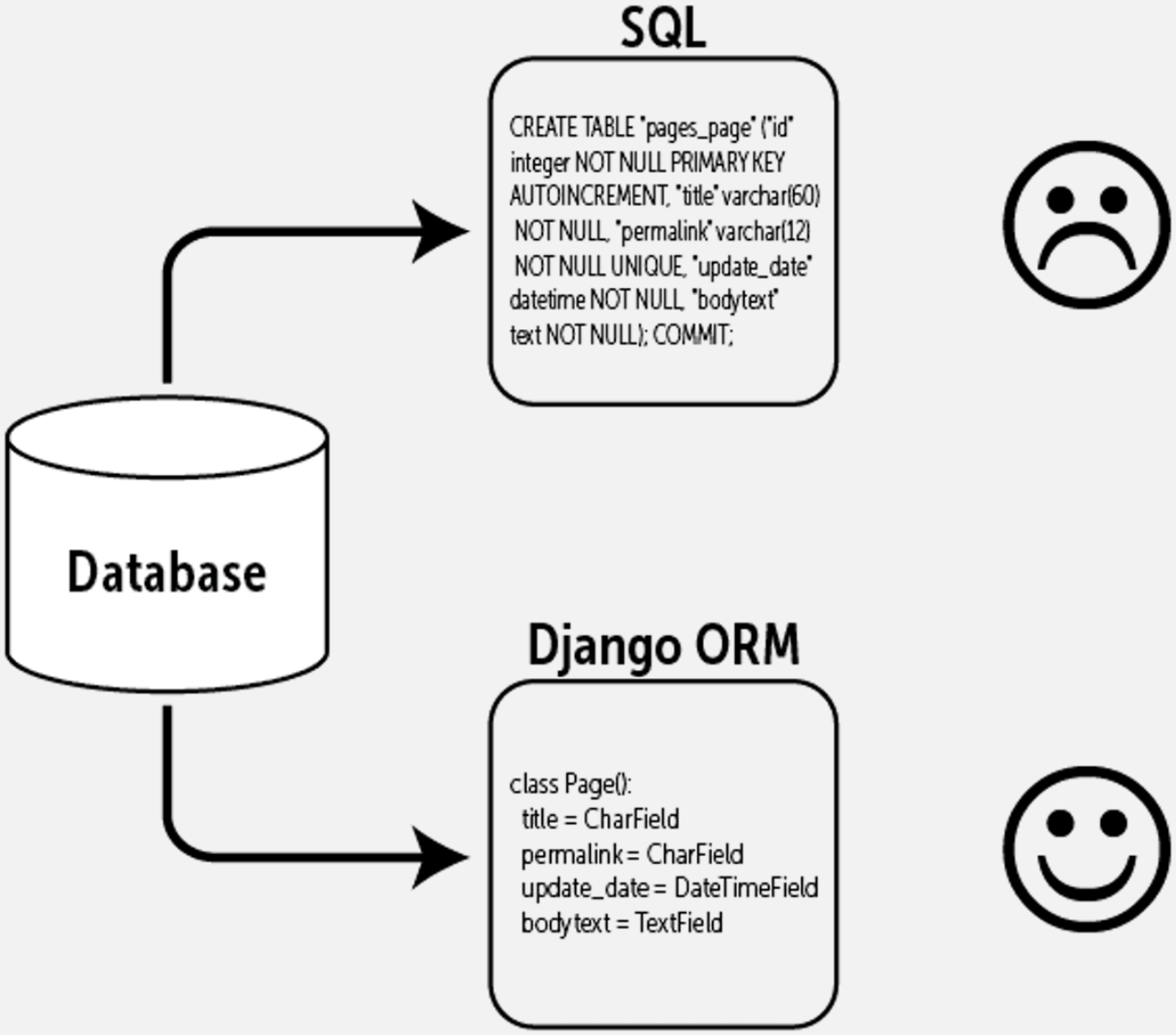
Каталог для хранения статических файлов (CSS, JavaScript, изображения).

Назначение: Статические файлы используются для фронтенд-части приложения, такие как стили и скрипты.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Django ORM (Object-Relational Mapping) — это мощная и удобная система, которая позволяет разработчикам взаимодействовать с базой данных через объектно-ориентированный подход. С ORM можно работать с данными, используя Python-классы и методы, не прибегая к написанию SQL-запросов напрямую. Django автоматически преобразует ваши запросы в SQL, что упрощает работу с базами данных.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)



DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

Модель

В Django ORM модель представляет собой Python-класс, который наследуется от django.db.models.Model. Каждый класс модели соответствует таблице базы данных, а атрибуты класса становятся столбцами таблицы.

Пример модели:

```
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    published_at = models.DateTimeField(auto_now_add=True)
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

- **Модель**

В Django ORM модель представляет собой Python-класс, который наследуется от **django.db.models.Model**. Каждый класс модели соответствует таблице базы данных, а атрибуты класса становятся столбцами таблицы.

Пример модели:

```
from django.db import models


class Article(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    published_at = models.DateTimeField(auto_now_add=True)
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

- **Поле модели**

Поля модели в Django ORM — это атрибуты класса, которые определяют тип данных и поведение соответствующего столбца в таблице базы данных. Существует множество встроенных типов полей, которые позволяют гибко описывать структуру данных.

Примеры полей:

- CharField: Строка с ограничением по длине.
- TextField: Длинный текст.
- IntegerField: Целое число.
- FloatField: Число с плавающей точкой.
- BooleanField: Логический тип данных (истина или ложь).
- DateTimeField: Дата и время.
- ForeignKey: Поле, устанавливающее связь "один ко многим" с другой моделью.
- ManyToManyField: Связь "многие ко многим" между моделями.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

Метаданные модели (Meta class)

Внутри каждой модели можно создать внутренний класс Meta, который позволяет управлять поведением модели на более высоком уровне. Он помогает определять, как Django должен вести себя с моделью — например, какое имя присвоить таблице или как сортировать результаты.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

```
class Article(models.Model):
    title = models.CharField(max_length=255)
    published_at = models.DateTimeField()


    class Meta:
        ordering = ['-published_at']  # Сортировка по дате публикации в обратном порядке
        verbose_name = "Статья"
        verbose_name_plural = "Статьи"
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

Основные параметры Meta:

- ordering: Определяет порядок сортировки объектов при выборке.
- db_table: Имя таблицы в базе данных (по умолчанию Django генерирует имя на основе имени модели).
- verbose_name и verbose_name_plural: Устанавливают отображаемое имя модели в единственном и множественном числе, например, для административной панели.
- unique_together: Указывает уникальность комбинации нескольких полей.
- index_together: Указывает индексацию комбинации полей для ускорения поиска.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

Связи между моделями

Django ORM поддерживает несколько типов связей между моделями:

- **Один ко многим (ForeignKey)**: Один объект связан с множеством других.
- **Многие ко многим (ManyToManyField)**: Одна запись может быть связана с несколькими записями другой модели и наоборот.
- **Один к одному (OneToOneField)**: Связь один к одному между записями.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные понятия Django ORM

```
class Author(models.Model):
    name = models.CharField(max_length=100)


class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)  # Связь один ко многим
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Создание записи**

Чтобы создать новый объект модели, нужно просто вызвать метод create() или использовать конструктор модели.

Пример:

```
article = Article.objects.create(title="My Article", content="Some content")
# Или
article = Article(title="My Article", content="Some content")
article.save()
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Извлечение данных (запросы)**

Django ORM позволяет выполнять запросы к базе данных через менеджеры моделей (обычно это objects).

Пример:

Получить все объекты:

articles = Article.objects.all()

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

Фильтрация данных:

```
recent_articles = Article.objects.filter(published_at__year=2024)
```

Получить один объект по условию:

```
article = Article.objects.get(id=1)
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Основные методы запросов:**

all() — возвращает все записи.

filter(kwargs)** — возвращает записи, соответствующие указанным условиям.

get(kwargs)** — возвращает одну запись, соответствующую условиям, либо вызывает ошибку, если объект не найден.

exclude(kwargs)** — возвращает записи, которые не соответствуют условиям.

order_by(*fields) — сортирует записи по указанным полям.

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Обновление данных**

Для обновления существующего объекта нужно изменить его атрибуты и вызвать метод **save()**.

Пример:

```
article = Article.objects.get(id=1)
article.title = "Updated Title"
article.save()
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Удаление данных**

Удалить объект можно с помощью метода **delete()**.

Пример:

```
article = Article.objects.get(id=1)
article.delete()
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Основные операции с ORM

- **Агрегатные функции**

Django ORM поддерживает различные агрегатные функции для выполнения статистических вычислений, таких как подсчет записей, получение суммы или среднего значения.

Пример:

```
from django.db.models import Count, Avg

# Подсчет количества статей
total_articles = Article.objects.count()

# Среднее значение для поля
average_articles = Article.objects.aggregate(Avg('views'))
```

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Миграции в Django ORM

Django ORM использует механизм миграций для управления схемой базы данных. Миграции автоматически создаются на основе изменений в моделях, и их можно применять для изменения структуры базы данных.

- **Создание миграций**

Чтобы создать миграции после изменения моделей, используется команда:

bash

python manage.py makemigrations

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

Миграции в Django ORM

- **Применение миграций**

Для применения миграций в базе данных используется команда:

bash

python manage.py migrate

DJANGO ORM (OBJECT-RELATIONAL MAPPING)

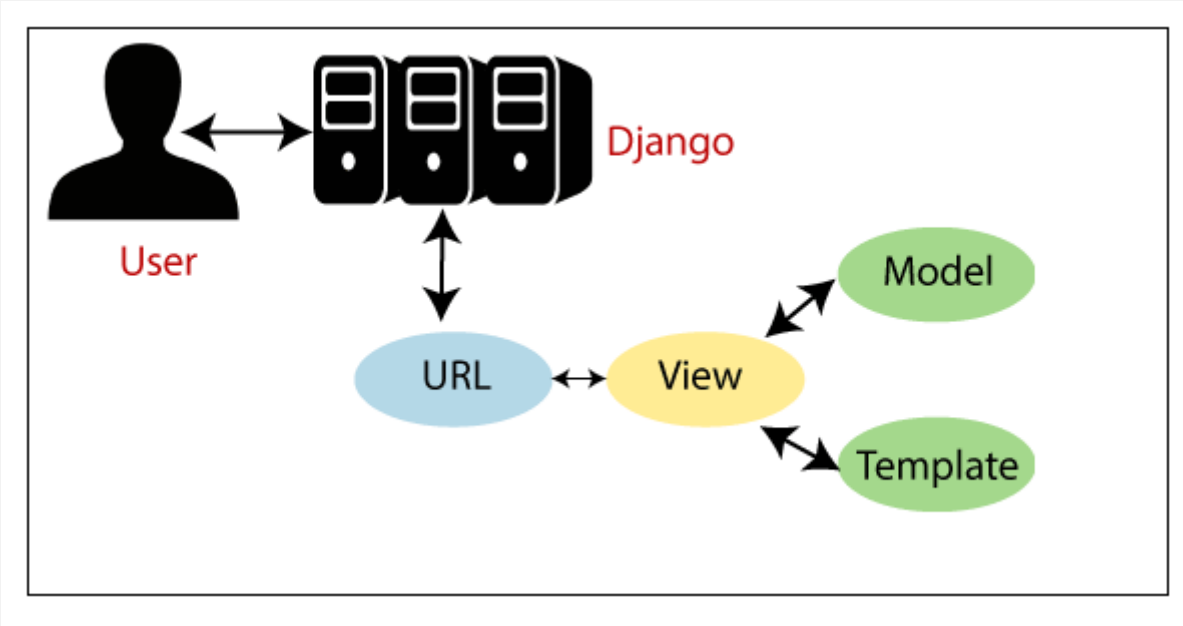
Миграции в Django ORM

Преимущества миграций Django

- Автоматизация: Не нужно вручную писать SQL-запросы для изменения схемы базы данных. Django автоматически генерирует необходимый код.
- Безопасность: Миграции позволяют легко управлять сложными изменениями в схеме базы данных без риска потерять данные или нарушить целостность базы.
- Управляемость: Все изменения фиксируются в миграционных файлах, что облегчает отслеживание и управление версионностью структуры базы данных.
- Совместимость с разными базами данных: Django автоматически адаптирует SQL-запросы для разных типов баз данных.

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Технология MVT (Model-View-Template) — это архитектурный паттерн, который используется в Django для организации кода веб-приложений. MVT является близким родственником более известного паттерна MVC (Model-View-Controller), но с некоторыми особенностями. Этот подход помогает разделить различные аспекты разработки веб-приложений на три независимых компонента: модель, представление и шаблон.



ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

- Model (Модель)

Модель отвечает за управление данными и взаимодействие с базой данных. Она описывает структуру данных, которая будет храниться в базе данных, а также бизнес-логику приложения.

- View (Представление)

Представление обрабатывает запросы от пользователя и возвращает ответ (обычно сгенерированную HTML-страницу). В Django представления — это функции или классы, которые получают HTTP-запросы, обрабатывают данные (при необходимости взаимодействуя с моделями) и возвращают HTTP-ответ.

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

- Template (Шаблон)

Шаблон отвечает за отображение данных. В Django шаблоны — это HTML-файлы с возможностью использования встроенного шаблонизатора для вставки переменных, циклов и условий в код.

Django предоставляет мощный механизм шаблонов, который позволяет динамически генерировать HTML, использовать шаблонные теги и фильтры.

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

- Как MVT работает в Django:

✓ Пользователь отправляет запрос к веб-сайту (например, вводит URL).

✓ Django URLConf (система маршрутизации) определяет, какое представление (view) должно обработать данный запрос, основываясь на URL.

✓ Представление (View) запрашивает данные из модели (Model) (если это необходимо) и затем передает их в шаблон (Template) для отображения.

✓ Шаблон (Template) обрабатывает данные и генерирует HTML-код, который будет возвращен пользователю в виде HTTP-ответа.

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

- Отличие MVT от MVC:

Django следует концепции MVT, которая схожа с MVC, но отличается тем, что в Django роль контроллера берет на себя фреймворк, а разработчик работает с представлениями и шаблонами. Основное отличие заключается в том, что:

- В MVC контроллер (Controller) управляет процессом обработки запроса.
- В MVT этот процесс берет на себя Django, предоставляя представление (View), которое напрямую взаимодействует с моделью (Model) и шаблоном (Template).

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

Преимущества MVT:

- Разделение логики и представления: Логика работы с данными отделена от отображения данных, что упрощает разработку и поддержку.
- Удобная система шаблонов: Шаблоны легко поддерживаются, и их можно переиспользовать в разных частях приложения.
- Упрощенная работа с данными: Django ORM и модели облегчают взаимодействие с базой данных, позволяя работать с данными в объектно-ориентированном стиле.

ТЕХНОЛОГИЯ MVT (MODEL-VIEW-TEMPLATE)

Основные компоненты MVT:

Архитектура MVT помогает организовать код веб-приложений так, чтобы каждая часть системы (модели, представления и шаблоны) была независима и могла развиваться отдельно. Это делает приложения на Django более структурированными, поддерживаемыми и легко расширяемыми.