Django

Что такое веб-приложение?

Разработка веб-приложений — это жизненно важный процесс создания программных приложений, которые работают через веб-браузеры, предлагая пользователям кросс-платформенную совместимость и адаптируясь к мобильным приложениям для обеспечения беспрепятственного доступа между устройствами.

Разница между веб-приложением и веб-сайтом заключается в его функциональности и назначении.

Веб-сайты обычно состоят из статического контента, такого как текст, изображения и видео, которые представлены пользователю в виде структуры с возможностью навигации. Они могут включать интерактивные элементы, такие как формы или панели поиска, но их основная цель — предоставить информацию пользователю.

Веб-приложения — это интерактивные программные приложения, которые позволяют пользователям выполнять определенные задачи или функции. Обычно они более сложны, чем веб-сайты, и для их работы требуется ввод данных пользователем. Примеры веб-приложений включают системы онлайн-банкинга, платформы электронной коммерции и сети социальных сетей, которые позволяют пользователям выполнять такие задачи, как перевод денег, совершение покупок и обмен контентом.

Хотя доступ к веб-сайтам и веб-приложениям осуществляется через веб-браузер и они запускаются на веб-сервере, ключевое различие заключается в их функциональности и назначении. Веб-сайты в первую очередь предназначены для предоставления информации посетителям, а веб-приложения представляют собой интерактивные программные приложения, которые позволяют пользователям выполнять определенные задачи или функции

Клиентские веб-приложения

Это тип веб-приложений, в которых большая часть обработки выполняется на стороне клиента, обычно в веб-браузере пользователя. Эти приложения в значительной степени полагаются на JavaScript, HTML и CSS, обеспечивая интерактивный и динамичный пользовательский интерфейс. Клиентские веб-приложения можно разрабатывать с использованием различных фреймворков и библиотек, таких как React, Angular или Vue.js , которые предоставляют набор инструментов и компонентов для упрощения процесса разработки.

Серверные веб-приложения

Серверные веб-приложения — это тип веб-приложений, в которых большая часть обработки выполняется на стороне сервера, обычно с использованием серверного языка программирования, такого как Python, PHP или Ruby. Эти приложения генерируют динамический контент, который отправляется на сторону клиента в виде HTML, CSS и JavaScript, и для правильной работы они полагаются на связь клиент-сервер.

Серверные веб-приложения часто используются для приложений, управляемых данными, таких как сайты электронной коммерции, социальные сети и системы управления контентом, где большой объем данных необходимо обрабатывать и хранить на стороне сервера. Они также часто используют базы данных для хранения и извлечения данных, такие как MySQL, PostgreSQL или MongoDB .

Серверные веб-приложения можно разрабатывать с использованием различных платформ и библиотек, таких как Ruby on Rails, Django или Laravel, которые предоставляют набор инструментов и компонентов для упрощения процесса разработки. Эти платформы часто предлагают такие функции, как маршрутизация, создание шаблонов, аутентификация и интеграция баз данных, которые могут помочь разработчикам легко создавать сложные приложения.

Фреймворки веб-приложений

При создании веб-приложений разработчики используют различные платформы и другие технологии, чтобы сделать процесс быстрее, проще и эффективнее. Эти технологии можно разделить на интерфейсные и серверные категории.

- Внешний интерфейс (Frontend)
- Серверная часть (Backend)
- Интерфейсы прикладного программирования (API)

Внешний интерфейс (Frontend)

Интерфейс веб-приложения — это та часть, с которой пользователь взаимодействует непосредственно через браузер. Он включает в себя пользовательский интерфейс, макет и общий дизайн приложения. Некоторые из ключевых технологий, используемых для фронтенд-разработки, включают в себя:

- ✓ HTML (язык гипертекстовой разметки) — это стандартный язык разметки, используемый для создания веб-страниц. Он обеспечивает структуру и содержимое веб-страницы и работает в сочетании с CSS и JavaScript для создания общего дизайна и функциональности веб-приложения.
- ✓ CSS (каскадные таблицы стилей) используются для определения визуального стиля и макета веб-страницы. Он работает в сочетании с HTML и JavaScript для создания визуально привлекательного и отзывчивого пользовательского интерфейса. Популярные фреймворки CSS включают Bootstrap и TailWind.
- ✓ JavaScript — это язык программирования, который широко используется для веб-разработки. Он позволяет разработчикам добавлять на веб-страницы интерактивность, анимацию и другие динамические функции. Популярные фреймворки JavaScript включают React, Angular и Vue.js.

Фреймворки веб-приложений

Серверная часть (Backend)

Серверная часть веб-приложения отвечает за обработку и хранение данных, управление аутентификацией и авторизацией пользователей, а также взаимодействие с внешними службами и API. Некоторые из ключевых технологий, используемых для серверной разработки, включают:

- ✓ Языки программирования. Такие языки, как Python, Ruby, PHP и Java, обычно используются для внутренней веб-разработки. Эти языки обеспечивают логику и функциональность, лежащие в основе веб-приложений, и работают в сочетании с такими платформами, как Laravel, Django, Ruby on Rails и Golang.
- ✓ Базы данных. Такие базы данных, как MySQL, PostgreSQL и MongoDB, используются для хранения и извлечения данных в веб-приложениях. Они необходимы для приложений, которым требуется постоянство данных, и часто используются вместе с библиотеками объектно-реляционного сопоставления (ORM), такими как SQLAlchemy и ActiveRecord.

Интерфейсы прикладного программирования (API)

API используются для связи между различными службами и приложениями. Они позволяют веб-приложениям интегрироваться с внешними сервисами, такими как платежные шлюзы, платформы социальных сетей и инструменты анализа данных. Популярные API включают Google Maps API, Stripe API и YouTube API.

Что такое Django?

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC (MTV). Созданный опытными разработчиками, Django берёт на себя большую часть хлопот веб-разработки, поэтому вы можете сосредоточиться на написании своего веб-приложения без необходимости изобретать велосипед. Он бесплатный и с открытым исходным кодом, имеет растущее и активное сообщество, отличную документацию и множество вариантов как бесплатной, так и платной поддержки.

Django был разработан в период с 2003 по 2005 год командой, которая занималась созданием и обслуживанием газетных веб-сайтов. После создания нескольких сайтов, команда начала повторно использовать множество общего кода и шаблонов проектирования. Этот общий код эволюционировал в веб-фреймворк, который превратился в проект «Django» с открытым исходным кодом в июле 2005 года.

Текущей версией фреймворка является версия 6. Каждый новый релиз фреймворка выходит в средним каждые 8 месяцев. Кроме того, постоянно выходят обновления и исправления в безопасности. На настоящий момент последняя актуальная версия 6.0.3.

- Сайт проекта https://www.djangoproject.com/
- Документация https://docs.djangoproject.com/en/6.0/
- Документация https://django.fun/docs/django/5.0/ (на русском)
- Документация https://metanit.com/python/django/ (на русском, но к версиям есть вопросы)

Сайты, использующие Django: Mozilla, National Geographic, Pinterest, OpenStack, Instagram и др.

Django

- ✓ DRY (Don’t Repeat Yourself)
- ✓ ORM (Object-Relational Mapping, объектно-реляционное преобразование)
- ✓ Встроенная панель управления
- ✓ Маршрутизация на основе регулярных выражений
- ✓ Создание проекта на основе нескольких атомарных приложений
- ✓ Свободное распространяемые сообществом пакеты

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по факту является модификацией распространённого в веб-программировании паттерна MVC (Model-View-Controller).

Model View Controller (MVC)

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

1. **Model (Модель)**

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

2. View (Представление)

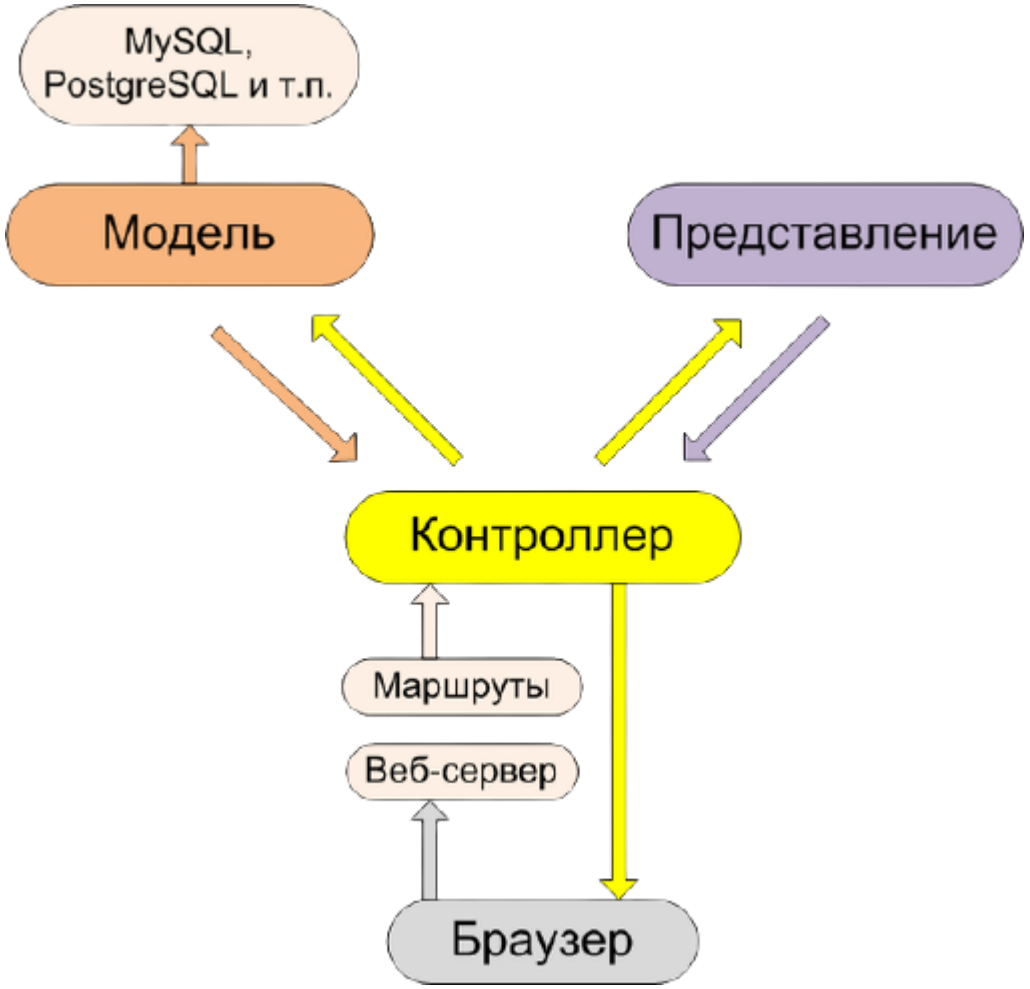
Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Код компонента view определяет внешний вид приложения и способы его использования.

3. Controller (Контроллер)

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.

Model View Controller (MVC)



Model View Template (MVT)

MVT — это архитектурный паттерн, используемый в веб-разработке и широко ассоциируемый с Django. Похож на более известный паттерн "Model-View-Controller", но адаптирован под разработку веб-приложений с некоторыми особенностями Django.

1. Model (Модель)

Представляет структуру данных приложения и бизнес-логику. В Django модели — это Python классы, которые определяют поля и поведение данных, которые вы хотите хранить. Django автоматически предоставляет API для доступа к данным (создание, изменение, удаление, запросы к базе данных) на основе определений моделей.

2. **View (Представление)**

Отвечает за обработку запросов пользователя и возвращение ответов. В Django представления — это Python функции или классы, которые принимают веб-запрос и возвращают веб-ответ. Представления взаимодействуют с моделями и передают данные в шаблоны. *В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).*

3. **Template (Шаблон)**

Отвечают за представление данных. Это текстовые файлы, которые позволяют миксовать HTML с Django Template Language (DTL) — специальным языком шаблонов Django для динамического генерирования HTML страниц. Шаблоны определяют, как данные, полученные от представлений, будут отображаться пользователю. *В MVC этому компоненту соответствует View, то есть представления.*

Работа MVT в Django

- **Запрос пользователя** поступает на веб-сервер и интерпретируется URL маршрутизатором Django.
- **URL маршрутизатор** направляет запрос к соответствующему представлению на основе URL паттерна.
- **Представление** обрабатывает запрос, взаимодействует с моделью для получения запрошенных данных или изменения данных в базе данных.
- После обработки данных представление выбирает **шаблон** для генерации HTML страницы и передает в шаблон необходимые данные.
- **Шаблон** рендерится в HTML, который возвращается пользователю в качестве ответа.

MVT делает веб-разработку в Django структурированной и гибкой. Разделение ответственности между моделями, представлениями и шаблонами облегчает разработку, тестирование и поддержку веб-приложений.

Установка Django

В терминале/консоли

Создали директория для проекта и зашли в нее

```
mkdir django
cd django
```

Установили и активировали виртуальную среду

```
python -m venv .venv
.\.venv\Scripts\activate
```

Установили для данной виртуальной среды Django

```
python -m pip install Django
```

Создание нового проекта

В терминале/консоли

```
(venv) $ django-admin startproject mysite
(venv) $ tree /f mysite
mysite/
├── manage.py
└── mysite
    ├── asgi.py
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Создание приложения в проекте

В терминале/консоли

```
(venv) $ py manage.py startapp polls
(venv) $ tree /f .
|   db.sqlite3
|   manage.py
+---mysite
|   |   asgi.py
|   |   settings.py
|   |   urls.py
|   |   wsgi.py
|   |   __init__.py
|   \---__pycache__
|           settings.cpython-312.pyc
|           urls.cpython-312.pyc
|           wsgi.cpython-312.pyc
|           __init__.cpython-312.pyc
\---polls
    |   admin.py
    |   apps.py
    |   models.py
    |   tests.py
    |   views.py
    |   __init__.py
    |
    \---migrations
            __init__.py
```

Object-Relational Mapping (ORM)

ORM (от английского — Object-Relational Mapping, дословный перевод, объектно-реляционное преобразование) — это технология, которая обеспечивает взаимодействие между объектно-ориентированным кодом и реляционными базами данными и упрощает процесс разработки.

Благодаря преобразованию можно работать с данными в БД, используя объекты в своем программном коде, вместо того, чтобы напрямую писать SQL-запросы. Для этого Object-Relational Mapping автоматически преобразует объекты в записи таблиц базы данных и обратно. Это упрощает процесс разработки в объектно-ориентированных подходах, повышает уровень абстракции, позволяет работать с данными более интуитивным образом.

ORM формирует подобие такой «виртуальной объектной базы данных», которая существует лишь в коде приложения и затем синхронизируется уже с реальной базой данных.

Простыми словами, ORM — это инструмент для оптимизации работы с БД.

ORM стараются скрыть существование базы данных настолько, насколько это возможно. Взамен программисту дают возможность оперировать данными в базе через специальный интерфейс. Вместо построения SQL-запросов, программист вызывает простые методы, а всю остальную работу берёт на себя ORM.

Несмотря на общую цель, ORM бывают очень разными. Django ORM относится к наиболее распространённому и простому типу ORM, реализующему шаблон проектирования Active Record. Этот шаблон базируется на идее, что каждой таблице в приложении соответствует один класс (модель). Этот класс отвечает как за реализацию бизнес логики, так и за взаимодействие с базой данных. Последнее обычно появляется в модели за счёт наследования от базового класса ORM.

Модели данных

Каждая модель представляет собой класс Python, который является подклассом django.db.models.Model. Каждый атрибут модели представляет поле базы данных.

Пример модели:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

SQL запись модели

```
CREATE TABLE myapp_person (
    "id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

- Класс атрибута определяет тип колонки базы данных
- Имя таблицы по умолчанию `myapp_person`
- id создаётся автоматически

Создание объектов в базе данных

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    age = models.IntegerField()



ivan = Person.objects.create(name="Иван", last_name ="Иванов", age=23)

maria = Person(name="Мария", last_name ="Иванова", age=21)
maria.save()
```

Получение объектов:

```
pps = Person.objects.all()

p1 = Person.objects.first()

p2 = Person.objects.get(id=1)
```

Типы полей

- `AutoField()` — целочисленное значение, которое автоматически увеличивается (обычно применяется для первичных ключей)
- `CharField(max_length=N)` — строка длиною не более N символов
- `TextField()` — строка неограниченного размера
- `IntegerField()` — целое число от -2147483648 до 2147483647
- `PositiveIntegerField()` — целое число от 0 до 2147483647
- `BooleanField()` — булева величина
- `DateField()` — дата
- `DateTimeField()` — дата и время
- `EmailField()` — адрес электронной почты
- `FileField()` — хранение файлов (файл хранится на файловой системе)
- `ImageField()` — хранение изображений
- `BinaryField()` — хранение двоичных данных

https://docs.djangoproject.com/en/6.0/ref/models/fields/

https://docs.djangoproject.com/en/6.0/ref/models/fields/#field-types

Параметры полей

Параметры полей: null

Если True, Django будет хранить пустые значения как NULL в базе данных. По умолчанию False
https://docs.djangoproject.com/en/6.0/ref/models/fields/#null

Параметры полей: blank

Если True, поле может быть пустым.
По умолчанию установлено значение False.
Отличие от null. null связана исключительно с базой данных, тогда как blank связана с проверкой.
https://docs.djangoproject.com/en/6.0/ref/models/fields/#blank

Параметры полей: default

Значение поля по умолчанию
https://docs.djangoproject.com/en/6.0/ref/models/fields/#default

Параметры полей: unique

Если указано True, это поле должно быть уникальным во всей таблице.
https://docs.djangoproject.com/en/6.0/ref/models/fields/#unique

Получение объектов из базы данных

- `get()` — получает один объект
- `get_or_create()` — получить объект или создать его
- `all()` — получить все объекты
- `count()` — получить количество объектов
- `filter(), exclude()` — фильтрация объектов
- `in_bulk()` — получить объекты в виде словаря

Обновление данных в базе данных

```
nic = Person.objects.get(id=2)
nic.name = "Николай"
nic.save()

nic = Person.objects.get(id=2)
nic.name = "Николай"
nic.save(update_fields=["name"])

Person.objects.filter(id=2).update(name="Михаил")

values_for_update = {"name": "Михаил", "last_name": "Мишин", "age": 31}
bob, created = Person.objects.update_or_create(id=2, defaults = values_for_update)
```

Удаление данных из базы данных

```
person = Person.objects.get(id=2)
person.delete()

Person.objects.filter(id=4).delete()
```

Отношения

Django предлагает способы определения трех наиболее распространенных типов отношений с базой данных:

- «один ко многим» ("one to many")
- «многие ко многим» ("many to many")
- «один к одному» ("one to one")

Отношение «один ко многим»

```
from django.db import models

class Hotel(models.Model):
    pass

class Room(models.Model):
    hotel = models.ForeignKey(Hotel, on_delete=models.CASCADE)
    # ...
```

ForeignKey: on_delete

- CASCADE — каскадное удаление. Эмулирует SQL ON DELETE CASCADE, а также удаляет объект, содержащий ForeignKey.
- PROTECT — предотвращает удаление объекта, на который есть ссылка, путем вызова ProtectedError.
- RESTRICT — предотвращает удаление указанного объекта путем вызова RestrictedError. В отличие от PROTECT, удаление ссылочного объекта допускается, если он также ссылается на другой объект, который удаляется в той же операции, но через отношение CASCADE.
- SET_NULL — устанавливает ForeignKey в null; должен быть указан параметр null со значением True.
- SET_DEFAULT — устанавливает ForeignKey в значение по умолчанию; значение по умолчанию для ForeignKey должно быть указано в описании поля.
- SET() — устанавливает ForeignKey в указанное переменной или функцией значение.
- DO_NOTHING — не предпринимать никаких действий

Отношение один ко многим

```python
class Question(models.Model):
    question_text = models.CharField("Вопрос", max_length=200)
    pub_date = models.DateTimeField('Дата публикации')

    def __str__(self):
        return f"Вопрос: {self.question_text}"


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField("Ответ", max_length=200)
    votes = models.IntegerField("Количество голосов", default=0)

    def __str__(self):
         return f"Ответ {self.choice_text}, ответили {self.votes}"
```

Отношения «Один ко многим». Прямые

```
>>> choice = Choice.objects.first()
>>> choice
<Choice: Ответ Ничего, ответили 2>
>>> choice.question
<Question: Вопрос: Что нового?>
```

Отношение один ко многим. Обратные

```
>>> q = Question.objects.first()
>>> q.choice_set.all()
<QuerySet [<Choice: Ответ Ничего, ответили 2>, <Choice: Ответ Много всего,
ответили 0>]>
>>> q.choice_set.count()
2
```

Имя `*_set` можно переопределить, задав параметр `related_name` в определении `ForeignKey`. Изменим модель `Choice`:

```
question = models.ForeignKey(Question, on_delete=models.CASCADE, related_name='choices')
```

```
>>> from polls.models import Question, Choice
>>> q = Question.objects.first()
>>> q
<Question: Вопрос: Что нового?>
>>> q.choices.all()
<QuerySet [<Choice: Ответ Ничего, ответили 2>, <Choice: Ответ Много всего, ответили 0>]>
>>>
```

Отношение «один к одному»

```
from django.db import models

class Address(models.Model):
    street = models.CharField(max_length=255)
    building = models.CharField(max_length=10)
    flat = models.CharField(max_length=10)
    zipcode = models.CharField(max_length=6)

    def __str__(self):
        return f"{zipcode}, #{street}, #{building}-#{flat}"

class Restaurant(models.Model):
    name = models.CharField(max_length=128)
    address = models.OneToOneField(Address, on_delete = models.CASCADE,
primary_key = True)
```

Отношение «многие ко многим»

```python
from django.db import models
class Topping(models.Model):
    # ...
    pass
class Pizza(models.Model):
    # ...
    toppings = models.ManyToManyField(Topping)
```

Неважно, какая модель имеет `ManyToManyField`, но вы должны поместить ее **только в одну из моделей, а не в обе.**

Дополнительная информация в отношениях «многие ко многим»

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=128)

    def __str__(self):
        return self.name

class Group(models.Model):
    name = models.CharField(max_length=128)
    members = models.ManyToManyField(Person, through="Membership")

    def __str__(self):
        return self.name

class Membership(models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE)
    group = models.ForeignKey(Group, on_delete=models.CASCADE)
    date_joined = models.DateField()
    invite_reason = models.CharField(max_length=64)
```

https://docs.djangoproject.com/en/6.0/topics/db/models/#extra-fields-on-many-to-many-relationships

Миграции

Миграции – это способ Django распространять изменения, которые вносятся в модели (добавление поля, удаление модели и т.д.), в схему базы данных. Они разработаны, чтобы быть в основном автоматическими, но нужно знать, когда выполнять миграции, когда их запускать и с какими общими проблемами вы можете столкнуться.

Команды

Есть несколько команд, которые используются для взаимодействия с миграциями и обработкой Django схемы базы данных:

- `migrate`, которая отвечает за применение и отмену миграции.
- `makemigrations`, которая отвечает за создание новых миграций на основе изменений, которые вы внесли в свои модели.
- `sqlmigrate`, которая отображает операторы SQL для миграции.
- `showmigrations`, в которой перечислены миграции проекта и их статус.

Можно думать о миграции как о системе контроля версий для схемы базы данных. `makemigrations` отвечает за упаковку изменений модели в отдельные файлы миграции, а `migrate` отвечает за их применение в базе данных.

Файлы миграции для каждого приложения находятся в каталоге «`migrations`» внутри этого приложения и предназначены для передачи и распространения как часть его кодовой базы.

Метаданные модели

```python
class Question(models.Model):
    question_text = models.CharField("Вопрос", max_length=200)
    pub_date = models.DateTimeField('Дата публикации')
    def __str__(self):
        return f"Вопрос: {self.question_text}"

    def was_published_recently(self):
        return self.pub_date >= timezone.now()-datetime.timedelta(days=1)

    class Meta:
        verbose_name = 'Вопрос'
        verbose_name_plural = 'Вопросы'
        ordering = ["-pub_date"]


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE, related_name='choices')
    choice_text = models.CharField("Ответ", max_length=200)
    votes = models.IntegerField("Количество голосов", default=0)

    def __str__(self):
        return f"Ответ {self.choice_text}, ответили {self.votes}"

    class Meta:
        verbose_name = 'Ответ'
        verbose_name_plural = 'Ответы'
        ordering = ["choice_text"]
```

https://docs.djangoproject.com/en/6.0/ref/models/options/#available-meta-options