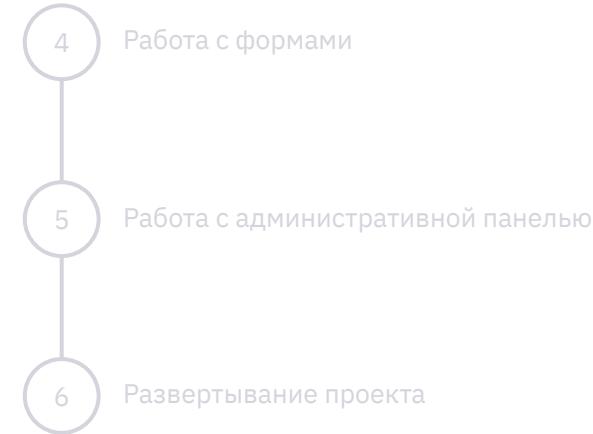
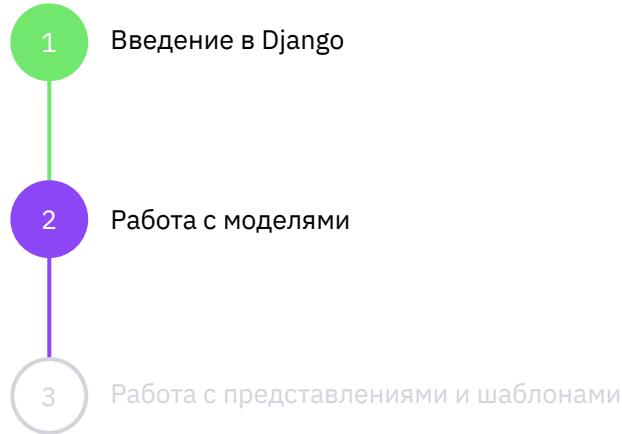


Работа с моделями

Урок 1



План курса





Содержание урока





Что будет на уроке сегодня

- 📌 Узнаем о моделях Django
- 📌 Разберёмся в создании моделей
- 📌 Изучим миграции
- 📌 Узнаем о создании собственных команд
- 📌 Изучим работу с моделями данных, CRUD



Введение в модели Django





Введение в модели Django

Django - это популярный веб-фреймворк, который позволяет разрабатывать мощные и масштабируемые веб-приложения. Одной из ключевых функций Django являются модели, которые обеспечивают удобный способ работы с базами данных.

Модели в Django - это классы Python, которые определяют структуру таблиц базы данных. Каждый атрибут класса соответствует полю таблицы, а экземпляры класса представляют записи в таблице. Django использует ORM (Object-Relational Mapping), чтобы автоматически создавать SQL-запросы для работы с базой данных.





Определение моделей



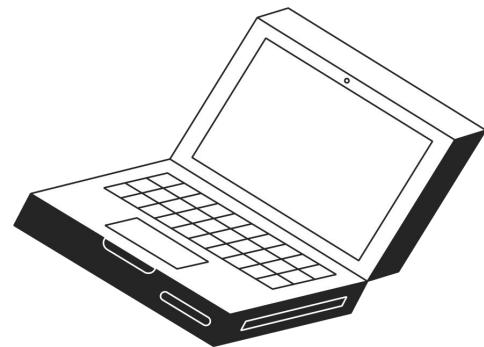


Создание модели

Для создания модели в Django мы должны перейти в файл `models.py` внутри вашего приложения и определить класс Python, который будет наследоваться от базового класса "`models.Model`"

```
from django.db import models

class User(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    password = models.CharField(max_length=100)
    age = models.IntegerField()
```





Описание полей модели

Django предоставляет множество типов полей

- CharField - поле для хранения строковых данных
- EmailField - поле для хранения электронной почты
- TextField - поле для хранения текстовых данных большой длины
- IntegerField - поле для хранения целочисленных данных
- DecimalField - поле для хранения десятичных чисел
- BooleanField - поле для хранения логических значений (True/False)
- DateTimeField - поле для хранения даты и времени
- ForeignKey - поле для связи с другой моделью
- ManyToManyField - поле для связи с другой моделью в отношении "многие-ко-многим"

<https://docs.djangoproject.com/en/4.2/ref/models/fields/#model-field-types>



Миграции



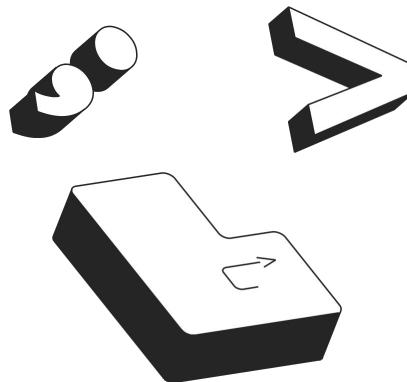


Как создать миграции для моделей?

Для создания миграций для моделей Django используется команда "makemigrations"

```
python manage.py makemigrations myapp2
```

Заглянем в каталог /migrations





Применение миграций

Вспомните запуск сервера командой `python manage.py runserver`

...

```
You have 19 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes,
myapp2, sessions.
```

Run '`python manage.py migrate`' to apply them.





Пара слов о создании
собственных команд
`manage.py`





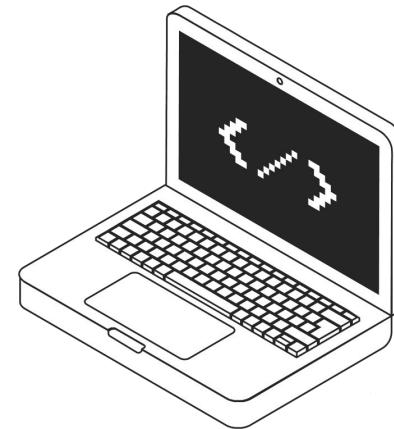
Создаём структуру каталогов

Django предоставляет возможности по созданию собственных команд вида

`python manage.py <command>`

Такие команды должны хранится в специальном месте

```
myproject/
    manage.py
    myapp2/
        __init__.py
        management/
            __init__.py
            commands/
                __init__.py
                my_command.py
...
...
...
```





Создаём файл с кодом команды

Простейшее содержимое файла команды должно быть следующим:

```
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = "Print 'Hello world!' to output."

    def handle(self, *args, **kwargs):
        self.stdout.write('Hello world!')
```



Работа с данными в моделях





Работа с данными в моделях

В Django работа с данными в моделях осуществляется через объекты моделей, которые представляют отдельные записи в базе данных.

CRUD - акроним, обозначающий четыре базовые функции, используемые при работе с базами данных:

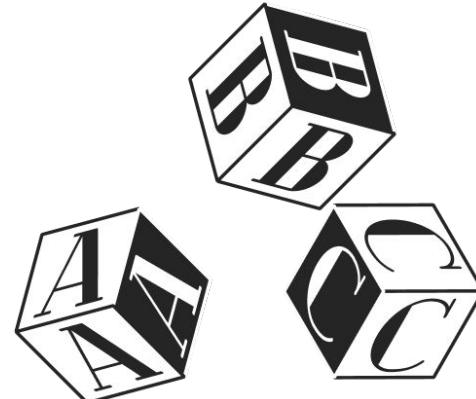
-  создание (create),
-  чтение (read),
-  модификация (update),
-  удаление (delete)



Создание объектов модели, `create`

Для создания нового объекта модели необходимо создать экземпляр класса модели и заполнить его поля значениями

```
user = User(name='John',  
            email='john@example.com',  
            password='secret',  
            age=25)
```





Добавляем читаемое представление в модель

Дандер метод `__str__` определяет формат вывода информации об экземплярах модели

```
...
class User(models.Model):
    ...
    def __str__(self):
        return f'Username: {self.name}, email: {self.email}, age: {self.age}'
...
...
```



Получение объектов модели, read

Для получения объектов модели из базы данных можно использовать методы `.all()`, `.get()`, `.filter()`

- `all()` возвращает все объекты модели
- `get()` возвращает один объект, соответствующий заданным условиям
- `filter()` возвращает объекты подходящие под условия фильтрации



Фильтрация объектов модели

Для фильтрации объектов модели по заданным условиям можно использовать метод `filter()`: `Model.objects.filter(param_filter=value)`

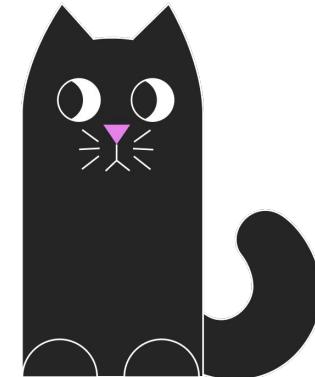
- 💡 exact - точное совпадение значения поля
- 💡 iexact - точное совпадение значения поля без учета регистра
- 💡 contains - значение поля содержит заданный подстроку
- 💡 in - значение поля находится в заданном списке значений
- 💡 gt - значение поля больше заданного значения
- 💡 gte - значение поля больше или равно заданному значению
- 💡 lt - значение поля меньше заданного значения
- 💡 lte - значение поля меньше или равно заданному значению
- 💡 startswith - значение поля начинается с заданной подстроки
- 💡 endswith - значение поля заканчивается на заданную подстроку
- 💡 range - значение поля находится в заданном диапазоне значений
- 💡 date - значение поля является датой, соответствующей заданной дате
- 💡 year - значение поля является годом, соответствующим заданному году



Изменение объектов модели, update

Для изменения объектов модели можно использовать методы поиска `get()` или `filter()` в сочетании с `save()` экземпляра класса модели

```
...
user = User.objects.filter(pk=pk).first()
user.name = new_name
user.save()
...
```





Удаление объектов модели, delete

Для удаления объектов модели можно использовать методы поиска get() или filter() в сочетании с delete() экземпляра класса модели

```
...
user = User.objects.filter(pk=pk).first()
if user is not None:
    user.delete()
...
```





Дополнительные возможности моделей

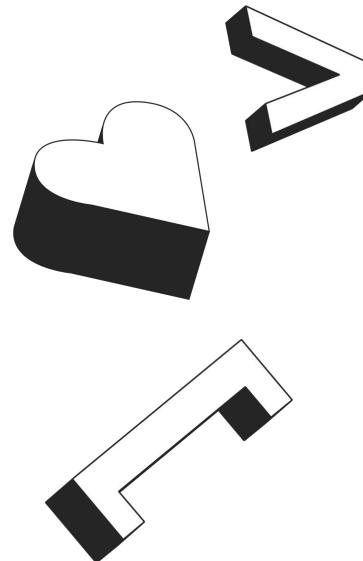




Связи между моделями

Отношения между моделями в Django позволяют описывать связи между объектами разных моделей

- ForeignKey - один ко многим
- ManyToManyField - многие ко многим
- OneToOneField - один к одному





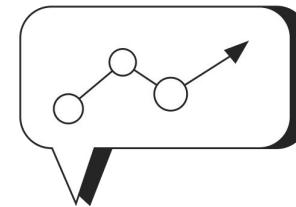
Создание пользовательских методов

Можно создавать пользовательские методы и свойства для моделей, чтобы расширить их функциональность

```
class Post(models.Model):
    ...
    content = models.TextField()
    ...

    def get_summary(self):
        words = self.content.split()
        return f'{ " ".join(words[:12])}...'

    ...
```





Итоги занятия





На этой лекции мы

- 📌 Узнали о моделях Django
- 📌 Разобрались в создании моделей
- 📌 Изучили миграции
- 📌 Узнали о создании собственных команд
- 📌 Изучили работу с моделями данных, CRUD



Задание

1. Для закрепления материалов лекции попробуйте самостоятельно набрать и запустить демонстрируемые примеры.
2. *Загляните в официальную документацию Django и изучите дополнительные возможности работы с моделями.





Спасибо за внимание

