

Лекция 5. Работа с административной панелью



На этой лекции мы

1. Узнаем об административной панели Django
2. Разберемся в настройке админ панели
3. Изучим добавление пользовательских моделей в панель
4. Узнаем о способах персонализации админ панели

Краткая выжимка, о чём говорилось в предыдущей лекции

На прошлой лекции мы:

1. Узнали о формах Django
2. Разобрались в создании классов форм
3. Изучили поля и виджеты форм
4. Узнали о обработке данных из формы
5. Разобрались в сохранении файлов

План лекции

[На этой лекции мы](#)

[Краткая выжимка, о чём говорилось в предыдущей лекции](#)

[План лекции](#)

[Подробный текст лекции](#)

[Что такое административная панель в Django?](#)

[Зачем нужна административная панель?](#)

[Вместо старта](#)

[Настройка административной панели](#)

[Первоначальная настройка](#)

[Настройка языка](#)

[Создание суперпользователя](#)

[Таблица пользователя в БД](#)

[Сброс пароля](#)

[Стандартная админка](#)

[Добавить пользователя](#)

[Просмотр и редактирование пользователя](#)

[Удаление пользователя](#)

[Добавление пользовательских моделей
в административную панель](#)

[Создание моделей](#)

[Создаём и применяем миграции](#)

[Подключение моделей к административной панели](#)

[Добавление записи без внешнего ключа](#)

[Добавление записи с внешним ключом](#)

[Подключение моделей из разных приложений](#)

[Персонализация моделей в админ панели](#)

[Изменение модели](#)

[Создаём миграции](#)

[Применяем миграции](#)

[Правим существующие продукты и добавляем новые](#)

[Настройка списка изменения](#)

[Отображение дополнительных полей](#)

[Сортировка строк](#)

[Добавление фильтрации в список изменения](#)

[Текстовый поиск](#)

[Добавление новых действий](#)

[Настройка изменения отдельной записи](#)

[Отображение полей](#)

[Детальная настройка отображения полей](#)

[Вывод](#)

[Домашнее задание](#)

Подробный текст лекции

Что такое административная панель в Django?

Django — это один из наиболее популярных фреймворков для создания веб-приложений на Python. Он предоставляет множество инструментов и функциональных возможностей, которые значительно упрощают процесс разработки. Одним из таких инструментов является административная панель.

Административная панель в Django — это готовый интерфейс, который позволяет управлять данными вашего приложения. Она предоставляет возможность создавать, редактировать и удалять записи в базе данных, а также осуществлять множество других действий, связанных с управлением приложением. Среди разработчиков административную панель часто называют коротким словом “админка”. Этот же термин используют и те, кто не создаёт административные панели, но пользуется ими — пользователи с доступом к админке.

Зачем нужна административная панель?

Во-первых, она значительно упрощает работу разработчиков, позволяя им быстро и удобно управлять данными.

Во-вторых, административная панель может быть использована в качестве удобного инструмента для администрирования приложения, что позволяет упростить жизнь не только разработчикам, но и другим пользователям приложения. В этой лекции мы рассмотрим основы работы с административной панелью в Django, настроим ее и создадим пользовательские модели. Также мы рассмотрим примеры использования административной панели и обсудим основные моменты ее работы.

Вместо старта

Если вы создавали новое приложение для каждого занятия, выполните команды:

```
>cd myproject
>python manage.py startapp myapp5
```

Отлично! Новое приложение создано в проекте. Сразу подключим его в настройках `setting.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'myapp5',
]
```

Всё готово к началу изучения админки на практике.

Настройка административной панели

Перед тем как начать работать с административной панелью впервые, необходимо сделать несколько обязательных действий. Далее в админку можно будет просто заходить для внесения правок в данные приложения.

Первоначальная настройка

Адрес по которому мы можем зайти в админ панель был настроен при создании проекта (начало лекции 1). Заглянем в `urls.py` проекта, уедемся что часть работы Django сделал за нас.

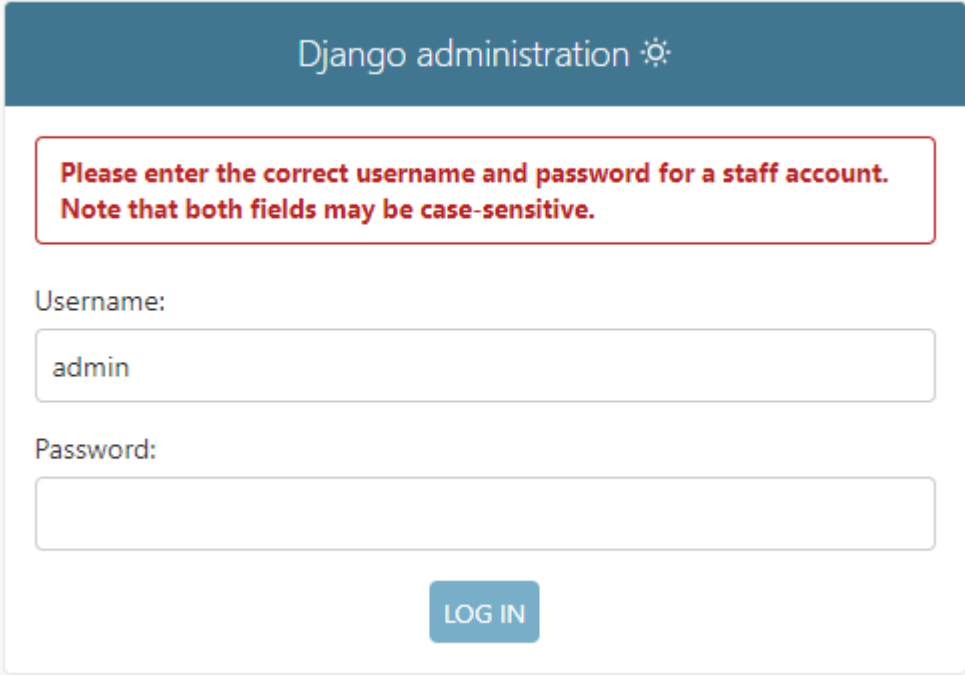
```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    ...
]
```

- импортирован модуль admin из django.contrib
- в urlpatterns добавлена строка, которая будет указывать на адрес административной панели

Стартуем сервер и переходим по адресу административной панели:
<http://127.0.0.1:8000/admin>

Мы автоматически попадаем на форму ввода логина и пароля. Но какие бы данные мы не ввели, доступ будет закрыт.



Настройка языка

А почему мы изучаем курс на русском языке, а Django общается с нами на английском? В фреймворке есть возможность использовать не только английский, но и другие популярные языки. Откроем файл settings.py и изменим языковую константу с английского на русский:

```
LANGUAGE_CODE = 'ru-ru'
```

Перезагружаем сервер. Теперь при попытке зайти в админ даже предупреждение мы видим на русском:

Пожалуйста, введите корректные имя пользователя и пароль учётной записи. Оба поля могут быть чувствительны к регистру.

Но смена языка не помогла на попасть в админку.



Внимание! Вы можете оставить язык по умолчанию или указать тот, который вам удобен. На код в рамках курса это не повлияет. Только на стандартный текст.

Создание суперпользователя

По умолчанию доступ к административной панели имеет только суперпользователь (superuser). Выполним в командной строке команду:

```
python manage.py createsuperuser
```

Далее введём имя пользователя, электронную почту, пароль и повтор пароля

```
Имя пользователя (leave blank to use 'pk-user'): Admin
Адрес электронной почты: admin@mail.ru
Password:
Password (again):
Введённый пароль слишком похож на имя пользователя.
Введённый пароль слишком короткий. Он должен содержать как минимум 8 символов.
Введённый пароль слишком широко распространён.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```



Внимание! При вводе пароля на экране ничего не отображается. Но Django учитывает нажатие каждой клавиши.



Важно! В примере использован пароль admin как пример антипаттерна. Не используйте короткий, распространённые пароли. Если пароль не проходит валидацию, придумайте более надёжный.

Таблица пользователя в БД

Если зайти в базу данных, можно найти введенные данные в таблице `auth_user`. Её мы создали при выполнении команды `migrate` в первый раз. Django подготовили миграции для пользователя заранее. Это 1 из 18 миграций, о которых нам сообщал Django на первой лекции курса.

Так выглядит наш суперпользователь:

```
{
  "id": 1,
  "password": "pbkdf2_sha256$600000$j7TQ1ugSw6bM24Unk0YAml$yrUbxt8/+7GPip5XahVoij7QuL6Z2QTArkWggHft8Sk=",
  "last_login": "2023-05-19 15:17:28.471108",
  "is_superuser": 1,
  "username": "Admin",
  "last_name": "",
  "email": "admin@mail.ru",
  "is_staff": 1,
  "is_active": 1,
  "date_joined": "2023-05-19 15:12:15.653025",
  "first_name": ""
}
```

Django побеспокоился о безопасности конфиденциальных данных, пароль хранится в виде хеша.

Кроме логина и пароля в таблице есть поля для имени и фамилии, несколько флагов, а также фиксация времени регистрации и последней активности.

Сброс пароля

Частая ситуация новичка, забыть свой пароль суперпользователя. В этом случае можно воспользоваться консольной командой:

```
python manage.py changepassword <username>
```

Вместо `<username>` надо подставить имя пользователя, чей пароль надо изменить.

```
python manage.py changepassword Admin
Changing password for user 'Admin'
Password:
Password (again):
```

Отлично! Переходим из консоли в браузер.

Стандартная админка

Ещё раз переходим по адресу <http://127.0.0.1:8000/admin/> и вводим логин и пароль. Мы оказываемся на главной странице админки.

Раздел “Пользователи и группы” доступен по умолчанию. Администратор сайта может делать всю цепочку CRUD: создавать, просматривать, изменять и удалять с группами и пользователями.

На самом деле CRUD доступен для любых моделей, которые зарегистрированы в админке. Но об этом в следующей главе лекции. А пока провернём цепочку CRUD с пользователем.

Добавить пользователя

Попробуем добавить обычного пользователя. Для этого надо кликнуть на кнопку добавить с символом “плюс”. Сначала введите имя пользователя и пароль. Затем вы сможете ввести больше информации о пользователе.

- Имя пользователя - Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/-/_.
- Пароль - Пароль не должен быть слишком похож на другую вашу личную информацию. Ваш пароль должен содержать как минимум 8 символов. Пароль не должен быть слишком простым и распространенным. Пароль не может состоять только из цифр.
- Подтверждение пароля - Для подтверждения введите, пожалуйста, пароль ещё раз.

Мы можем просто сохранить запись в базе данных, либо Сохранить и добавить другой объект, либо Сохранить и продолжить редактирование.

Просмотр и редактирование пользователя

Чтение и обновление пользователя происходят на одной и той же странице. В нашем примере это адрес <http://127.0.0.1:8000/admin/auth/user/2/change/>. На него мы можем попасть через кнопку изменить и последующий клик по имени пользователя.

Данные в форме сгруппированы:

- Имя и скрытый пароль
- Персональная информация
- Права доступа
- Важные даты

Любой пользователь, у которого активен “Статус персонала” (is_staff) может входить в административную панель. Изменим статус нашего пользователя и

попробуем зайти в админку под его профилем.



Внимание! Для выхода из административной панели можно использовать адрес <http://127.0.0.1:8000/admin/logout/> или нажать ссылку в правом верхнем углу страницы.

Введя логин и пароль созданного нами пользователя мы скорее всего увидим запись:

Администрирование сайта

У вас недостаточно полномочий для просмотра или изменения чего либо.

Система прав пользователя определяет какие из действий в административной панели разрешены пользователю. По умолчанию запрещено всё.



Важно! Статус суперпользователя указывает, что пользователь имеет все права без явного их назначения. Делайте суперпользователями только администраторов сайта.

Удаление пользователя

Удалить пользователя можно как из списка пользователей, так и из режима редактирования пользователя.

В первом случае выбираем пользователя из списка, активируем галочку, далее в списке действий выбираем “Удалить выбранного пользователя” и нажимаем кнопку выполнить.

Во втором случае нажимаем красную кнопку удалить в правом нижнем углу профиля пользователя.

В любом из вариантов необходимо подтвердить удаление записи из базы данных нажав на кнопку: “Да, я уверен”.



Внимание! Часто удаление данных из базы не является хорошим решением. Такие данные можно восстановить только из резервной копии, а это затратная операция. Проще пометить данные удалёнными. Например для стандартной модели пользователя есть флаг `is_active`. Его отключение

сохраняет данные в БД, но не активный пользователь не сможет зайти на сайт, словно он удалён.

Добавление пользовательских моделей в административную панель

До этого момента мы не написали ни одной строчки кода. Да, Django даёт огромный простор при работе с административной панелью на основе того, что уже есть во фреймворке. Но можно ли добавлять в админку свои таблицы? Ответа — да!

Создание моделей

Добавим в проект пару моделей как мы это делали начиная с лекции про Модели. Например Категории и Продукты. Таблица "категории" будет иметь уникальный идентификатор каждой категории, а таблица "продукты" будет иметь столбец, который связывает каждый продукт с уникальным идентификатором его категории. Открываем models.py приложения и пишем код:

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=50, unique=True)

    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=50)
    category = models.ForeignKey(Category,
    on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

Мы специально сделали модели максимально простыми:

- Модель Category содержит только одно поле name, которое является уникальным строковым идентификатором категории.

- Модель Product содержит два поля: name, которое является строковым именем продукта, и category, которое является внешним ключом на модель Category. Внешний ключ определяется с помощью поля ForeignKey, которое указывает на модель Category и использует on_delete=models.CASCADE, чтобы обеспечить каскадное удаление, когда категория удаляется.

Создаём и применяем миграции

Убедимся, что приложение добавление в список в settings.py проекта:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    ...  
    'myapp5',  
]
```

Далее создадим миграции

```
>python manage.py makemigrations myapp5  
Migrations for 'myapp5':  
  myapp5\migrations\0001_initial.py  
    - Create model Category  
    - Create model Product
```

И финальный штрих — применение миграций:

```
>python manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, myapp2,  
  myapp3, myapp4, myapp5, sessions  
Running migrations:  
  Applying myapp5.0001_initial... OK
```

Подключение моделей к административной панели

Подключим модели категории и товара к админке приложения. Открываем файл admin.py приложения. Его создал Django в процессе выполнения команды startapp.

Пропишем внутри следующий код:

```
from django.contrib import admin  
from .models import Category, Product
```

```
admin.site.register(Category)
admin.site.register(Product)
```

Первая строка импортирует доступ к административным свойствам и методам.

Далее импортируем классы моделей, которые создали ранее.

Две нижние строки регистрируют модели в административной панели сайта.

Запускаем сервер и переходим в админ панель <http://127.0.0.1:8000/admin>

Теперь мы видим раздел с именем приложения и таблицы категории и продукта с стандартными кнопками добавить и изменить.



Внимание! Названия таблиц будут на английском языке даже при включении русской локали в настройках. Для перевода создаваемых разработчиком данных, в том числе названий моделей и полей необходимо воспользоваться функцией `gettext` из `django.utils.translation`. Материал перевода выходит за рамки курса. Но вы можете изучить его самостоятельно по ссылке <https://docs.djangoproject.com/en/4.2/topics/i18n/translation/>

Добавление записи без внешнего ключа

Модель категория содержит единственное поле “имя”. Через кнопку добавить мы можем создать несколько категорий продуктов. Например, мясо, молоко, овощи, фрукты.



Внимание! При добавлении нескольких записей одной модели удобнее пользоваться кнопкой “Сохранить и добавить другой объект”.

Добавление записи с внешним ключом

Модель продукт содержит помимо имени ссылку - внешний ключ на модель категория. Админка Django автоматически добавляет поле выбора для создания связи между продуктом и его категорией.

Например мы можем добавить колбасу и выбрать ей категорию мясо. А далее добавить сыр, сметану и творог и выбрать им категорию молоко.

При этом кнопка “плюс” рядом с категорией позволяет при редактировании продуктов, создавать новые категории. Добавим торт и создадим ему категорию десерты. В отдельно открывшемся окне прописываем категорию “десерты”, нажимаем сохранить. Категория автоматически подставляется для продукта “торт”.

Подключение моделей из разных приложений

Вернёмся к коду лекции 3. Если вы создавали для каждого занятия свой проект, в файле `models.py` из каталога `myapp3` у вас будет храниться модель автора и статьи. Подключим их к админке через `admin.py` приложения `myapp3`.

```
from django.contrib import admin
from .models import Author, Post

admin.site.register(Author)
admin.site.register(Post)
```

Теперь админ панель добавила раздел `myapp3` с возможностью изменять записи авторов и постов.

Обратите внимание, что при выводе списка записей Django используете дандер `__str__` модели. Например у автора прописано:

```
...
def __str__(self):
    return f'Name: {self.name}, email: {self.email}'
```

В результате перейдя по адресу <http://127.0.0.1:8000/admin/myapp3/author/> мы видим строки:

- Name: Author_1, email: mail1@mail.ru
- Name: Author_2, email: mail2@mail.ru
- ...

Как изменить представление по умолчанию узнаем через несколько абзацев.

Персонализация моделей в админ панели

Не всегда таблицы базы данных бывают небольшими. Они могут состоять из десятков, а иногда и сотен столбцов.

Изменение модели

Например продукт может иметь не только имя и категорию. Вполне возможна следующая структура:

- Название продукта
- Описание продукта
- Цена продукта
- Категория продукта
- Количество на складе
- Дата добавления продукта
- Рейтинг продукта

Внесём изменения в модель Продукт нашего приложения myapp5, файл models.py:

```
from django.db import models

...
class Product(models.Model):
    name = models.CharField(max_length=50)
    category = models.ForeignKey(Category,
on_delete=models.CASCADE)
    description = models.TextField(default='', blank=True)
    price = models.DecimalField(default=999999.99, max_digits=8,
decimal_places=2)
    quantity = models.PositiveSmallIntegerField(default=0)
    date_added = models.DateTimeField(auto_now_add=True)
    rating = models.DecimalField(default=5.0, max_digits=3,
decimal_places=2)

    def __str__(self):
        return self.name
```

Теперь класс Product имеет следующие поля:

- name — строка (CharField) длиной не более 50 символов, обязательное поле, содержащее название товара.
- category — внешний ключ (ForeignKey) на модель Category с параметром on_delete=models.CASCADE, обязательное поле, содержащее категорию товара.
- description — текстовое поле (TextField) с параметром default="", то есть значение поля по умолчанию пустая строка, и параметром blank=True, то есть поле может быть пустым.
- price — числовое поле с плавающей запятой (DecimalField) с параметрами default=999999.99 (значение поля по умолчанию), max_digits=8

(максимальное число цифр в числе) и `decimal_places=2` (количество цифр после запятой), обязательное поле, содержащее цену товара.

- `quantity` — положительное целочисленное поле (`PositiveSmallIntegerField`) с параметром `default=0` (значение поля по умолчанию), содержащее количество товара. Для поля `PositiveSmallIntegerField` допускается диапазон чисел от 0 до 32767 включительно.
- `date_added` — поле даты и времени (`DateTimeField`) с параметром `auto_now_add=True`, то есть при создании объекта будет автоматически заполнено текущей датой и временем.
- `rating` — числовое поле с плавающей запятой (`DecimalField`) с параметрами `default=5.0` (значение поля по умолчанию), `max_digits=3` (максимальное число цифр в числе) и `decimal_places=2` (количество цифр после запятой), содержащее рейтинг товара.

Мы уже добавили несколько записей в таблицу продуктов. Для вновь созданных полей обязательно прописать значения по умолчанию, `default`. Они будут применены к существующим в базе данных записям.

Создаём миграции

Параметр `default` добавлен для всех новых полей кроме `date_added`. При создании миграций Django предложит нам выбор:

- 1) самостоятельно добавить значение по умолчанию для существующих строк
- 2) прервать создание миграций и прописать `default` параметр в модели

Мы выбираем первый вариант и соглашаемся с `timezone.now` как датой добавления существующих записей.

```
>py manage.py makemigrations myapp5
It is impossible to add the field 'date_added' with
'auto_now_add=True' to product without providing a default. This
is because the database needs something to populate existing
rows.
  1) Provide a one-off default now which will be set on all
existing rows
  2) Quit and manually define a default value in models.py.
Select an option: 1
Please enter the default value as valid Python.
Accept the default 'timezone.now' by pressing 'Enter' or provide
another value.
The datetime and django.utils.timezone modules are available, so
it is possible to provide e.g. timezone.now as a value.
Type 'exit' to exit this prompt
```



```
[default: timezone.now] >>>
Migrations for 'myapp5':

myapp5\migrations\0002_product_date_added_product_description_pro
duct_price_and_more.py
    - Add field date_added to product
    - Add field description to product
    - Add field price to product
    - Add field quantity to product
    - Add field rating to product
```

Применяем миграции

```
>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp2,
myapp3, myapp4, myapp5, sessions
Running migrations:
                                Applying
myapp5.0002_product_date_added_product_description_product_price_
and_more... OK
```

Изменения в базе данных зафиксированы. Можно стартовать сервер и проверять админку <http://127.0.0.1:8000/admin>

Правим существующие продукты и добавляем новые

Перейдём в админку <http://127.0.0.1:8000/admin/myapp5/product/> и отредактируем уже существующие продукты. Заодно добавим несколько новых.

В зависимости от типа поля в модели, админка создаёт соответствующие поля:

- для строкового имени поле ввода имеет обычный тип `type="text"`.
- для `ForeignKey` поле выбора `select`.
- для `TextField` описание поле ввода текста `textarea`.
- для цены, количества и рейтинга поля ввода с `type="number"`.
- поле `date_added` не отображается в админке. Параметр `auto_now_add=True` автоматически заполняет поле.

При этом админка следит за валидацией данных. Например попытаемся ввести данные, выходящие за обозначенные в модели рамки <http://127.0.0.1:8000/admin/myapp5/product/5/change/> Мы получим предупреждения

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ADMIN. ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Начало > Муapp5 > Products > торт

Начните печатать для фильтрации...

MYAPP3

Authors + Добавить

Posts + Добавить

MYAPP5

Categorys + Добавить

Products + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы + Добавить

Пользователи + Добавить

История

Имя: торт

Категория: десерты

Описание:

Цена: 1000000,01

Количество: -1

Рейтинг: 15,00

Сохранить

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

Удалить

Изменим существующие продукты, чтобы они прошли валидацию. Например так

id	name	category_id	date_added	description	price	quantity	rating
1	колбаса	1	2023-05-20 08:58:26.184100	Много мяса, мало бумаги	1699,5	325	7.77
2	сыр	2	2023-05-20 08:58:26.184100	Сыр из молока	999.97	50	9.7
3	сметана	2	2023-05-20 08:58:26.184100	Вкуснейший творог	250,01	800	7.9
4	творог	2	2023-05-20 08:58:26.184100	Вкусный творог	150,99	125	8.5
5	торт	5	2023-05-20 08:58:26.184100	Просто торт	753,77	4	7.02

id	name	category_id	date_added	description	price	quantity	rating
6	бисквит	5	2023-05-20 09:22:10.261009	Как торт, но вкуснее	425.25	70	6.95

Настройка списка изменения

Рассмотрим возможности админки в рамках списка значений, например списка продуктов, основанного на модели Product и хранящегося в базе данных в таблице myapp5_product.

Отображение дополнительных полей

Вернёмся к тому, что в списке выводится информация из дандер `__str__` модели. Например по адресу <http://127.0.0.1:8000/admin/myapp5/product/> мы видим содержимое поля name для каждого продукта. Перейдём в admin.py приложения и внесём правки в код:

```
from django.contrib import admin
from .models import Category, Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ['name', 'category', 'quantity']

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)
```

Импорты остались без изменения. Создаём класс ProductAdmin как дочерний для admin.ModelAdmin. Он позволит изменить работу с продуктами в админке, не меняя модель Продукты. Следовательно нам не надо делать миграции, вносить изменения в базу данных.

Переменная list_display является зарезервированным именем. Django автоматически найдёт её и прочтает содержимое списка. Как вы догадались, в списке имена полей модели Продукты. Помимо имени мы хотим видеть категорию продукта и сколько его осталось на складе.

Обновим страницу <http://127.0.0.1:8000/admin/myapp5/product/> Django отображает три столбца, вместо одного.

Сортировка строк

Новый список можно упорядочить. При клике по заголовку столбца срабатывает сортировка. Текстовые поля сортируются по алфавиту, числовые по значению, дата и время по хронологии. При этом никакой код писать не нужно. Возможны сортировки по убыванию и по возрастанию значения. Кроме того возможна сложная сортировка по нескольким полям.

Но мы можем прописать сортировку по умолчанию в коде административной модели, файле `admin.py` приложения:

```
from django.contrib import admin
from .models import Category, Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ['name', 'category', 'quantity']
    ordering = ['category', '-quantity']

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)
```

Переменная `ordering` так же является зарезервированным именем. Django автоматически найдёт её и прочтает содержимое списка. В нашем примере используется двухуровневая сортировка продуктов. Вначале по категориям по возрастанию первичного ключа, далее по количеству по убыванию внутри категории.



Внимание! Сортировка таблиц базы данных с большим количеством значений может значительно замедлить выполнение запроса. Если сортируются не индексированные поля, скорость также снижается. Не стоит злоупотреблять сортировкой данных!

Добавление фильтрации в список изменения

Снова откроем файл `admin.py` и добавим ещё одну переменную: `list_filter`. Она сохраняет список для фильтрации записей.

```
from django.contrib import admin
```

```

from .models import Category, Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ['name', 'category', 'quantity']
    ordering = ['category', '-quantity']
    list_filter = ['date_added', 'price']

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)

```

Справа от списка изменений появился раздел Фильтр с двумя блоками: дата добавления и цена. При этом модель адаптируется под тип поля и его содержимое.

Для даты видим возможность фильтрации:

- Любая дата
- Сегодня
- Последние 7 дней
- Этот месяц
- Этот год

Для цены Django подбирает возможность отфильтровать значения по числовым значениям из базы данных.

Текстовый поиск

Финальный штрих — добавления поля поиска значений в базе данных. Аналог LIKE запроса в SQL.

Добавим ещё одну строку в административную модель продукта, файл admin.py приложения:

```

from django.contrib import admin
from .models import Category, Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ['name', 'category', 'quantity']
    ordering = ['category', '-quantity']
    list_filter = ['date_added', 'price']
    search_fields = ['description']
    search_help_text = 'Поиск по полю Описание продукта (description)'

```

```
admin.site.register(Category)
admin.site.register(Product, ProductAdmin)
```

Список `search_fields` определяет по каким полям будет проходить поиск текста. В нашем примере мы ищем совпадение введённой строки со значениями поля `description`. Тут же можно указать подсказку по поиску. Для этого используем переменную `search_help_text`.

Добавление новых действий

По умолчанию над списком есть поле выбора действия с единственным вариантом “Удалить выбранные объекты”. Если отметить галочкой несколько записей, выбрать действие “Удалить...” и нажать кнопку выполнить, мы попадём на страницу подтверждения удаления. Стандартное предупреждение, дублирование информации об удаленных объектах и кнопки выбора:

Вы уверены?

Вы уверены, что хотите удалить products? Все следующие объекты и связанные с ними элементы будут удалены:

Краткая статистика

Products: 3

Объекты

Product: колбаса

Product: сметана

Product: творог

Кнопки выбора

Да, я уверен

Нет, отменить и вернуться к выбору

Помимо стандартного удаления можно создать собственные методы, которые расширят список возможных действий над группой выбранных объектов. Пропишем в `admin.py` метод сбрасывания количества товаров в ноль.

```
from django.contrib import admin
from .models import Category, Product
```

```

@admin.action(description="Сбросить количество в ноль")
def reset_quantity(modeladmin, request, queryset):
    queryset.update(quantity=0)

class ProductAdmin(admin.ModelAdmin):
    """Список продуктов."""
    list_display = ['name', 'category', 'quantity']
    ordering = ['category', '-quantity']
    list_filter = ['date_added', 'price']
    search_fields = ['description']
    search_help_text = 'Поиск по полю Описание продукта (description)'
    actions = [reset_quantity]

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)

```

Создаём функцию `reset_quantity`, которая принимает три обязательных параметра: модель админа, запрос и объект запроса в базу данных `queryset`. Для функции прописываем декоратор `@admin.action`. Так Django поймёт, что мы создали новое действие. В качестве аргумента декоратора передаём текст, который будет виден в раскрывающемся списке действий. Сама функция использует объект `queryset` для того чтобы обновить количество товара, установив в качестве значения ноль. Чтобы действие появилось в списке возможных, прописываем у административной модели переменную `actions`. В список складываем добавляемые действия.



Внимание! В отличие от других списков, имя функции в списке не заключается в кавычки. Также не нужны круглые скобки после имени. Мы передаём функцию как объект.

Настройка изменения отдельной записи

А теперь поговорим о возможностях модификации формы изменения объекта. В нашем случае речь пойдёт о форме `Продукт`. Главой ранее мы пользовались её для изменения существующих и добавления новых продуктов.

Отображение полей

Мы можем изменить порядок полей, скрыть некоторые из них или сделать их неизменяемыми.

```
from django.contrib import admin
from .models import Category, Product

...

class ProductAdmin(admin.ModelAdmin):
    """Список продуктов."""
    list_display = ['name', 'category', 'quantity']
    ...

    """Отдельный продукт."""
    fields = ['name', 'description', 'category', 'date_added',
'rating']
    readonly_fields = ['date_added', 'rating']

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)
```

Новое поле `fields` определяет порядок вывода элементов формы. Если опустить какие-то поля, они перестанут отображаться. Например мы больше не видим цену и количество товара.

Переменная `readonly_fields` так же содержит список полей. Эти поля можно просматривать, но нельзя изменять. Мы сделали неизменяемым рейтинг. Поле `'date_added'` изначально было неизменяемым, так как дата проставляется автоматически в момент создания записи. Подобное поведение мы указали в модели:

```
...
date_added = models.DateTimeField(auto_now_add=True)
...
```

Если добавить дату добавления в `fields`, но не добавлять в `readonly_fields`, получим ошибку вида `FieldError`:

```
'date_added' cannot be specified for Product model form as it is
a non-editable field. Check fields/fieldsets/exclude attributes
of class ProductAdmin.
```


Будьте внимательны при выводе полей. Редактируемое поле можно сделать нередактируемым, добавив в `readonly_fields` список. Наоборот сделать не получится.

Детальная настройка отображения полей

Если необходимо более индивидуально настроить отображение полей модели, можно воспользоваться переменной `fieldsets`. Рассмотрим вариант представления, который задействует максимум возможностей формы. В очередной раз изменяем файл `admin.py` приложения:

```
from django.contrib import admin
from .models import Category, Product

...
class ProductAdmin(admin.ModelAdmin):
    """Список продуктов."""
    list_display = ['name', 'category', 'quantity']
    ...

    """Отдельный продукт."""
    # fields = ['name', 'description', 'category', 'date_added',
    'rating']
    readonly_fields = ['date_added', 'rating']
    fieldsets = [
        (
            None,
            {
                'classes': ['wide'],
                'fields': ['name'],
            },
        ),
        (
            'Подробности',
            {
                'classes': ['collapse'],
                'description': 'Категория товара и его подробное
описание',
                'fields': ['category', 'description'],
            },
        ),
        (
            'Бухгалтерия',
            {
                'fields': ['price', 'quantity'],
```

```

        },
        (
            'Рейтинг и прочее',
            {
                'description': 'Рейтинг сформирован автоматически  
на основе оценок покупателей',
                'fields': ['rating', 'date_added'],
            }
        ),
    ]

admin.site.register(Category)
admin.site.register(Product, ProductAdmin)

```

Использование данного кода для класса ProductAdmin в админке Django приведет к следующим изменениям в отображении формы:

- Поля "date_added" и "rating" отобразятся только для чтения, так как они будут указаны в параметре `readonly_fields`.
- Все поля будут разбиты на четыре группы (fieldset):
 - Первая группа будет содержать только поле "name", она будет иметь класс "wide", что означает, что она будет занимать все доступное место на странице.
 - Вторая группа будет содержать поля "category" и "description", они будут скрыты по умолчанию (класс "collapse"), но можно будет развернуть эту группу, нажав на соответствующий заголовок. Под заголовком отобразится описание группы.
 - Третья группа будет содержать поля "price" и "quantity". Они выводятся в одну строку, потому что переданы как элемент кортежа.
 - Четвертая группа будет содержать поля "rating" и "date_added", она также содержит описание, которое будет отображаться под заголовком.

Таким образом, форма будет иметь более структурированный и четкий вид, что упростит ее использование для администраторов.

Как вы видите, административная панель Django обладает достаточно широким набором инструментов. Часть из них позволяют пользоваться админкой написав минимум строк кода. Другая позволяет активно изменять и настраивать админ панель под любые нужды.

Вывод

На этой лекции мы:

1. Узнали об административной панели Django
2. Разобрались в настройке админ панели
3. Изучили добавление пользовательских моделей в панель
4. Узнали о способах персонализации админ панели

Домашнее 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 административной панелью.