



Урок 7

Собственная админка

Интеграция нового приложения в проект. Ограничение доступа к админке. Реализация механизма CRUD для пользователей и категорий товаров.

[Создание нового приложения adminapp](#)

[Диспетчер адресов](#)

[Контроллеры](#)

[Шаблоны](#)

[Добавление ссылки в главное меню и ограничение доступа](#)

[Админка: работа с объектами пользователей](#)

[Админка: работа с объектами категорий](#)

[Практическое задание](#)

[Дополнительные материалы](#)

Создание нового приложения adminapp

В нашем магазине работают практически все гиперссылки, и уже можно делать покупки. Пришло время создать инструмент для наполнения его контентом — админку. Сделаем ее в виде отдельного приложения **adminapp**. Большую часть урока будем закреплять пройденный материал.

Как только создали приложение, добавляем его имя в список **INSTALLED_APPS** файла конфигурации проекта **settings.py** и корректируем адрес в главном диспетчере URL:

geekshop/urls.py

```
...  
path('admin/', include('adminapp.urls', namespace='admin'))  
...
```

Диспетчер адресов

Создаем диспетчер адресов в папке с приложением.

adminapp/urls.py

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

app_name = 'adminapp'

urlpatterns = [
    path('users/create/', adminapp.user_create, name='user_create'),
    path('users/read/', adminapp.users, name='users'),
    path('users/update/<int:pk>', adminapp.user_update, name='user_update'),
    path('users/delete/<int:pk>', adminapp.user_delete, name='user_delete'),

    path('categories/create/', adminapp.category_create,
name='category_create'),
    path('categories/read/', adminapp.categories, name='categories'),
    path('categories/update/<int:pk>', adminapp.category_update,
name='category_update'),
    path('categories/delete/<int:pk>', adminapp.category_delete,
name='category_delete'),

    path('products/create/category/<int:pk>', adminapp.product_create,
name='product_create'),
    path('products/read/category/<int:pk>', adminapp.products,
name='products'),
    path('products/read/<int:pk>', adminapp.product_read, name='product_read'),
    path('products/update/<int:pk>', adminapp.product_update,
name='product_update'),
    path('products/delete/<int:pk>', adminapp.product_delete,
name='product_delete'),
]
```

Получилось громоздко. Здесь мы создаем точки входа контроллеров CRUD трех частей админки:

- работа с объектами пользователей;
- работа с объектами категорий;
- работа с объектами продуктов.

Для продуктов мы сделали два контроллера **Read**:

```
path('products/read/category/<int:pk>', adminapp.products, name='products'),
path('products/read/<int:pk>', adminapp.product_read, name='product_read'),
```

Первый выводит список продуктов в категории, а второй — страницу выбранного продукта.

Контроллеры

Следующим шагом создадим контроллеры. Некоторые из них пока будут заглушками.

adminapp/views.py

```
from authapp.models import ShopUser
from django.shortcuts import get_object_or_404, render
from mainapp.models import Product, ProductCategory

def users(request):
    title = 'админка/пользователи'

    users_list = ShopUser.objects.all().order_by('-is_active', '-is_superuser',
    '-is_staff', 'username')

    content = {
        'title': title,
        'objects': users_list
    }

    return render(request, 'adminapp/users.html', content)

def user_create(request):
    pass

def user_update(request, pk):
    pass

def user_delete(request, pk):
    pass

def categories(request):
    title = 'админка/категории'

    categories_list = ProductCategory.objects.all()

    content = {
        'title': title,
        'objects': categories_list
    }

    return render(request, 'adminapp/categories.html', content)

def category_create(request):
```

```

pass

def category_update(request, pk):
    pass

def category_delete(request, pk):
    pass

def products(request, pk):
    title = 'админка/продукт'

    category = get_object_or_404(ProductCategory, pk=pk)
    products_list = Product.objects.filter(category__pk=pk).order_by('name')

    content = {
        'title': title,
        'category': category,
        'objects': products_list,
    }

    return render(request, 'adminapp/products.html', content)

def product_create(request, pk):
    pass

def product_read(request, pk):
    pass

def product_update(request, pk):
    pass

def product_delete(request, pk):
    pass

```

В контроллере **users()** отсортировали результат запроса по нескольким полям — так лучше воспринимается.

В контроллере **products()** передали в шаблон кроме списка продуктов **products_list** еще и сам объект категории **category**. Это нужно, чтобы вывести ее имя в меню админки и сформировать гиперссылку создания нового продукта. Она должна содержать **pk** категории, в которой он будет добавлен. Напомним, что ее шаблон в диспетчере URL имеет вид:

```
'products/create/category/<int:pk>/'
```

Остальные контроллеры будем создавать в следующих шагах. Сейчас самое главное — понять структуру админки и работу диспетчера URL.

Шаблоны

Создадим структуру шаблонов. Начнем с базового.

adminapp/templates/adminapp/base.html

```
...
<body>
  <div class="admin_container">
    <div class="h2 text-center head">
      Админка
    </div>
    {% block menu %}
      <div class="admin_menu">
        {% include 'adminapp/includes/inc_menu.html' %}
      </div>
    {% endblock %}

    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

Подшаблон меню.

adminapp/templates/adminapp/includes/inc_menu.html

```
<ul class="menu">
  <li>
    <a href="{% url 'main' %}">
      на сайт
    </a>
  </li>
  <li>
    <a href="{% url 'admin:users' %}"
      class="{% if request.resolver_match.url_name == 'users' %}
        active
      {% endif %}">
      пользователи
    </a>
  </li>
  <li>
    <a href="{% url 'admin:categories' %}"
      class="{% if request.resolver_match.url_name == 'categories' or
        category.name %}
        active
      {% endif %}">
      категории
      {% if category %}
        :<span>{{ category.name }}</span>
      {% endif %}
    </a>
  </li>
</ul>
```

Реализуем интерфейс админки при помощи трех элементов меню. С объектами продуктов будем взаимодействовать через ссылки на странице категорий. Этот код не должен вызывать у вас вопросов (если нужны пояснения — стоит повторить материал предыдущих уроков). Названия категорий в меню при просмотре списка продуктов выводятся за счет условия:

```
{% if category %}
  :<span>{{ category.name }}</span>
{% endif %}
```

Шаблон страницы «Пользователи».

adminapp/templates/adminapp/users.html

```
{% extends 'adminapp/base.html' %}
{% load staticfiles %}

{% block content %}
    <div class="users_list">
        <button class="new_user">
            <a href={% url 'admin:user_create' %}>
                НОВЫЙ ПОЛЬЗОВАТЕЛЬ
            </a>
        </button>
        {% for object in objects %}
            <div
                class="user_record {% if not object.is_active %}not_active{% endif %}">
                
                <div class="user_role">
                    {% if object.is_superuser %}
                        администратор
                    {% else %}
                        пользователь
                    {% endif %}
                <button>
                    <a href={% url 'admin:user_update' object.pk %}>
                        редактировать
                    </a>
                </button>
                <button>
                    <a href={% url 'admin:user_delete' object.pk %}>
                        удалить
                    </a>
                </button>
            </div>
            <div class="user_name">
                <p><strong>{{ object.username }}</strong></p>
                <p>
                    {% if object.first_name %}
                        <b>{{ object.first_name|title }}</b>
                    {% if object.last_name %}
                        <p>{{ object.last_name|title }}</p>
                    {% endif %}
                </p>
            </div>
            <div class="user_age">
                age:<br>
                {{ object.age }} лет
            </div>
        </div>
    </div>
</div>
```



```

</div>
<div class="user_email">
  email:<br>
  {{ object.email }}
</div>
<div class="user_lastlogin">
  logged:<br>
  {{ object.last_login }}
</div>
</div>
{% endfor%}
</div>
{% endblock %}

```

В верхней части страницы расположили кнопку «Новый пользователь». Дальше в цикле выводим данные о каждом пользователе: аватарка, роль, имя, возраст, почта, время последнего входа в систему. Также в блоке `<div class="user_role">` располагаем кнопки «Редактировать» и «Удалить». На что стоит обратить внимание:

- если аватарки нет — выводим изображение по умолчанию:

```

{{ object.avatar|default:'users_avatars/default.jpg' }}

```

- в цикле называем текущий объект именем **object**, а не **user** — это в будущем упростит повторное применение кода.

Шаблон страницы «Категории».

adminapp/templates/adminapp/categories.html

```
{% extends 'adminapp/base.html' %}
{% load staticfiles %}

{% block content %}
    <div class="categories_list">
        <button class="new_category">
            <a href={% url 'admin:category_create' %}>
                новая категория
            </a>
        </button>

        {% for object in objects %}
            <div class="category_record">
                <div class="category_name">
                    {{ object.name|title }}
                </div>
                <div class="category_actions">
                    <button>
                        <a href={% url 'admin:products' object.pk %}>
                            товары категории
                        </a>
                    </button>
                    <button>
                        <a href={% url 'admin:category_update' object.pk %}>
                            редактировать
                        </a>
                    </button>
                    <button>
                        <a href={% url 'admin:category_delete' object.pk %}>
                            удалить
                        </a>
                    </button>
                </div>
                <div class="category_desc">
                    {{ object.description|title }}
                </div>
            </div>
        {% endfor %}
    </div>
{% endblock %}
```

Структура шаблона повторяет страницу с пользователями: в верхней части расположили кнопку «Новая категория», дальше — список объектов-категорий. В блок **<div class="category_actions">** поместили кнопки действий: «Редактировать» и «Удалить». Тут же расположим переход на страницу «Товары категории»:

adminapp/templates/adminapp/products.html

```

{% extends 'adminapp/base.html' %}
{% load staticfiles %}

{% block content %}
<div class="products_list">
  <button class="new_product">
    <a href={% url 'admin:product_create' category.pk %}>
      новый продукт
    </a>
  </button>

  {% for object in objects %}
  <div class="product_record">
    
    <div class="product_name">
      {{ object.name|title }}
    </div>
    <div class="product_actions">
      <button>
        <a href={% url 'admin:product_read' object.pk %}>
          подробнее
        </a>
      </button>
      <button>
        <a href={% url 'admin:product_update' object.pk %}>
          редактировать
        </a>
      </button>
      <button>
        <a href={% url 'admin:product_delete' object.pk %}>
          удалить
        </a>
      </button>
    </div>
    <div class="summary">
      <b>цена</b>
      <p>{{ object.price }} руб</p>
      <b>количество</b>
      <p>{{ object.quantity }}</p>
    </div>
    <div class="product_desc">
      {{ object.short_desc|title }}
    </div>
  </div>
  {% endfor%}
</div>
{% endblock %}

```

Код шаблона простой. Выводим краткую информацию о продукте и кнопки с действиями: «Подробнее», «Редактировать», «Удалить».

Все шаги выполнены — можно проверять работу админки, прописав в адресной строке:

```
127.0.0.1:8000/admin/users/read/
```

Должны увидеть страницу с меню и списком пользователей. Перейдите на страницу «Категории» и попробуйте открыть «Товары категории» — все должно работать.

Добавление ссылки в главное меню и ограничение доступа

Пропишем в подшаблоне главного меню код:

mainapp/templates/mainapp/includes/inc_menu.html

```
...
{% if user.is_authenticated %}
<li>
  <a href="{% url 'auth:edit' %}">
    {{ user.first_name|default:'Пользователь' }}
  </a>
</li>
{% endif %}
{% if user.is_superuser %}
<li>
  <a href="{% url 'admin:users' %}">админка</a>
</li>
{% endif %}
<li>
  {% if user.is_authenticated %}
    <a href="{% url 'auth:logout' %}">ВЫЙТИ</a>
  {% else %}
    <a href="{% url 'auth:login' %}">ВОЙТИ</a>
  {% endif %}
</li>
...
```

Теперь суперпользователи будут видеть в главном меню ссылку на админку. Для всех остается возможность попасть в нее по url-адресу, как при проверке в предыдущем шаге. Ограничим доступ при помощи декоратора [user_passes_test](#):

```
...
from django.contrib.auth.decorators import user_passes_test

@user_passes_test(lambda u: u.is_superuser)
def users(request):
    ...
    return render(request, 'adminapp/users.html', content)
```

В качестве первого позиционного аргумента декоратору необходимо передать функцию, возвращающую логическое значение. Мы написали лямбда-функцию. Если доступ пытается получить не суперпользователь — произойдет переход по адресу, который прописали в константе **LOGIN_URL**. Благодаря механизму, реализованному на прошлом уроке, после логина мы окажемся на запрашиваемой изначально странице. В нашем случае это админка.

Теперь можно прописать декоратор перед каждым контроллером админки.

Админка: работа с объектами пользователей

Итак, у нас заработал механизм админки: есть точка входа, есть меню, и оно работает, есть индексные страницы со списками объектов. Реализацию механизма CRUD начнем с объектов пользователей магазина:

```
...
from django.shortcuts import HttpResponseRedirect
from django.urls import reverse
from authapp.forms import ShopUserRegisterForm
from adminapp.forms import ShopUserAdminEditForm
...

def user_create(request):
    title = 'пользователи/создание'

    if request.method == 'POST':
        user_form = ShopUserRegisterForm(request.POST, request.FILES)
        if user_form.is_valid():
            user_form.save()
            return HttpResponseRedirect(reverse('admin:users'))
    else:
        user_form = ShopUserRegisterForm()

    content = {'title': title, 'update_form': user_form}

    return render(request, 'adminapp/user_update.html', content)
```

```

def user_update(request, pk):
    title = 'пользователи/редактирование'

    edit_user = get_object_or_404(ShopUser, pk=pk)
    if request.method == 'POST':
        edit_form = ShopUserAdminEditForm(request.POST, request.FILES, \
                                           instance=edit_user)

        if edit_form.is_valid():
            edit_form.save()
            return HttpResponseRedirect(reverse('admin:user_update', \
                                              args=[edit_user.pk]))
    else:
        edit_form = ShopUserAdminEditForm(instance=edit_user)

    content = {'title': title, 'update_form': edit_form}

    return render(request, 'adminapp/user_update.html', content)

def user_delete(request, pk):
    title = 'пользователи/удаление'

    user = get_object_or_404(ShopUser, pk=pk)

    if request.method == 'POST':
        #user.delete()
        #вместо удаления лучше сделаем неактивным
        user.is_active = False
        user.save()
        return HttpResponseRedirect(reverse('admin:users'))

    content = {'title': title, 'user_to_delete': user}

    return render(request, 'adminapp/user_delete.html', content)
...

```

Здесь импортируем формы и работаем с ними по знакомой схеме. При редактировании не забывайте передавать в форму полученные файлы **request.FILES** и сам объект в именованном аргументе **instance**, например:

```
ShopUserAdminEditForm(request.POST, request.FILES, instance=edit_user)
```

Мы уже знаем, что вместо удаления записей из базы правильнее пометить их неактивными. Этот механизм реализован в контроллере:

```

user.is_active = False
user.save()

```

Из кода контроллеров видно, что для создания и редактирования пользователя будет использоваться один шаблон **'adminapp/user_update.html'**.

В будущем необходимо стремиться делать универсальные шаблоны — тогда их количество будет уменьшаться, а поддержка и развитие проекта станут проще.

Чтобы добавить возможностей для редактирования пользователя, создадим в приложении **adminapp** отдельную форму **ShopUserAdminEditForm**:

adminapp/forms.py

```
from django import forms
from authapp.models import ShopUser
from authapp.forms import ShopUserEditForm

class ShopUserAdminEditForm(ShopUserEditForm):
    class Meta:
        model = ShopUser
        fields = '__all__'
```

Просто создали класс на базе уже существующего **ShopUserEditForm** и расширили список полей в нем. При этом получили все методы класса-родителя.

Шаблоны.

adminapp/templates/adminapp/user_update.html

```
{% extends 'authapp/base.html' %}
{% load staticfiles %}

{% block content %}
    <form class="form-horizontal" method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ update_form.as_p }}
        <input class="form-control" type="submit" value="сохранить">
    </form>
    <button class="btn btn-round form-control last">
        <a href="{% url 'admin:users' %}">
            к списку пользователей
        </a>
    </button>
{% endblock %}
```

Для универсального использования формы мы не прописываем атрибут **action** — данные автоматически будут направлены по текущему URL страницы.

```
{% extends 'adminapp/base.html' %}
{% load staticfiles %}

{% block content %}
<div class="user_delete">
    Уверены, что хотите удалить {{ user_to_delete.username }}
    {% if user_to_delete.first_name %}
        ({{ user_to_delete.first_name|title }}</b>
    {% if user_to_delete.last_name %}
        ({{ user_to_delete.last_name|title }}
    {% endif %})
    {% endif %}
    ?
    <form action="{% url 'admin:user_delete' user_to_delete.pk %}"
        method="post">
        {% csrf_token %}
        <input class="btn btn-danger" type="submit" value="удалить">
    </form>
    <button class="btn btn-success">
        <a href="{% url 'admin:users' %}">
            Отмена
        </a>
    </button>
</div>
{% endblock %}
```

Можно было обойтись без шаблона `user_delete.html`. Но он позволяет защитить проект от удаления объектов простым GET-запросом. Удаление произойдет, только когда контроллер `user_delete()` получит запрос методом **POST**:

```
if request.method == 'POST':
    ...
```

Если это кажется сложным — попробуйте сами переписать код контроллера, чтобы удаление работало сразу по GET-запросу.

Админка: работа с объектами категорий

Следующим шагом реализуем механизм CRUD для категорий магазина. Начнем с формы редактирования категории.

adminapp/forms.py

```
...
from mainapp.models import ProductCategory

...
class ProductCategoryEditForm(forms.ModelForm):
    class Meta:
        model = ProductCategory
        fields = '__all__'

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for field_name, field in self.fields.items():
            field.widget.attrs['class'] = 'form-control'
            field.help_text = ''
```

Код контроллеров и шаблонов берем из предыдущего шага.

Важно: необходимо добавить атрибут **is_active** к модели **ProductCategory**, чтобы реализовать механизм удаления из предыдущего шага:

mainapp/models.py

```
from django.db import models

class ProductCategory(models.Model):
    name = models.CharField(verbose_name='имя', max_length=64, unique=True)
    description = models.TextField(verbose_name='описание', blank=True)
    is_active = models.BooleanField(verbose_name='активна', default=True)
    ...
```

Не забудьте выполнить миграции.

Чтобы выделить неактивные категории на странице админки, добавим в шаблон **categories.html** условие:

adminapp/templates/adminapp/categories.html

```
...
{% for object in objects %}
    <div class="category_record
        {% if not object.is_active %}
            not_active
        {% endif %}">
    <div class="category_name">
        {{ object.name|title }}
    </div>
...

```

Остается прописать стиль **'category_record.not_active'** в файле **styles.css**, например:

```
opacity: 0.5;
```

С объектами продуктов поработаем на следующем занятии.

Практическое задание

1. Создать приложение админки и интегрировать его в проект.
2. Реализовать механизм CRUD для объектов пользователей магазина. Можно полностью удалять объекты (не использовать свойство **is_active**).
3. Реализовать механизм CRUD для объектов категорий товара. Можно полностью удалять объекты (не использовать свойство **is_active**).
4. Защитить доступ к админке декоратором **@user_passes_test**.
5. * Реализовать удаление через свойство **is_active**.
6. * Реализовать в админке «подсветку» неактивных объектов пользователей и категорий.

Дополнительные материалы

Все то, о чем сказано в методичке, но подробнее:

1. [Декоратор @user_passes_test](#).