Собственная админка

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

[Создание нового приложения adminapp](#_1fob9te)

[Диспетчер адресов](#_8vh9w0xl0o7z)

[Контроллеры](#_2et92p0)

[Шаблоны](#_tyjcwt)

[Добавление ссылки в главном меню и ограничение доступа](#_3dy6vkm)

[Админка: работа с объектами пользователей](#_1t3h5sf)

[Админка: работа с объектами категорий](#_4d34og8)

[Практическое задание](#_2s8eyo1)

[Дополнительные материалы](#_17dp8vu)

# 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**. Б**о**льшую 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 %}">  <img   src="/media/{{ object.avatar|default:'users\_avatars/default.jpg' }}"  alt="{{ object.username }}">  <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 %}  {% endif %}  </p>  </div>  <div class="user\_age">  age:<br>  {{ object.age }} лет  </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">  <img   src="/media/{{object.image|default:'products\_images/default.jpg'}}"  alt="{{ object.name }}">  <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 главном 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](https://docs.djangoproject.com/en/2.1/topics/auth/default/#django.contrib.auth.decorators.user_passes_test):

**adminapp/views.py**

|  |
| --- |
| ... 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:

**adminapp/views.py**

|  |
| --- |
| ... 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 помечать 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.

**adminapp/templates/adminapp/user\_delete.html**

|  |
| --- |
| {% 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](https://docs.djangoproject.com/en/2.1/topics/auth/default/#django.contrib.auth.decorators.user_passes_test).