Работаем над контентом

Работаем над заполнением сайта реальной информацией. Реализуем подробную информацию по каждому объекту. Добавляем в админку функционал для: добавления/редактирования/удаления объектов из базы данных. Пагинация.

[Создаем модели](#_c713xnuw3xem)

[Поэкспериментируем в консоли](#_14atu5yryjd4)

[Выведем на странице](#_byjowx90zavf)

[Доработаем админку](#_elwx96r5vupd)

[Пагинация](#_bw66wlgvi4z9)

[Теория](#_1zryeyfatjew)

[Практическая часть урока](#_fe0hdt828xhu)

[Домашнее задание](#_owpsqd9opgcf)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_e7skojz6dozc)

# 

# Создаем модели

Следующей задачей является наполнение сайта контентом. Для хранения и отображение объектов (в моем случае, драгоценных камней) нужны модели.

Все камни будем делить на категории. Каждая категория будет иметь название и описание. Рейтинг будет изменяться в зависимости от оценок пользователей.

**models.py**

|  |
| --- |
| **class** Gem(models.Model):  name = models.CharField(verbose\_name=**u'название'**, max\_length=32, unique=**True**)  category = models.ForeignKey(**'Category'**)  image = models.ImageField(blank=**True**)  rating = models.PositiveIntegerField(verbose\_name=**'рейтинг'**, default=0)  description = models.TextField(verbose\_name=**'описание'**, blank=**True**)  **class** Category(models.Model):  name = models.CharField(verbose\_name=**u'название'**, max\_length=16, unique=**True**)  description = models.TextField(verbose\_name=**'описание'**, blank=**True**) |

# Выведем на странице

Для отображения драг. камней сделаем отдельную страницу gems\_page.html

|  |
| --- |
| {% extends "base.html" %}  {% block title %}  Gems  {% endblock %}  {% block content %}  <**div class="row"**>  {% for gem in gems %}  <**div class="gem\_container"**>  <**div class="name"**>  {{ gem.name }}  </**div**>  <**div class="image"**>  <**img src="/static/images/{{ gem.image | default:"dfav.png**" }}">  </**div**>  {{ gem.category.name }}  </**div**>  {% endfor %}  </**div**>  {% endblock %} |

# Доработаем админку

Наполнять сайт данными, используя консоль, не особо удобно. Сделаем для этого хороший инструмент в админке.

Первым делом, создадим форму на основании модели Gem, которую будем использовать для создания объектов. Принцип создания идентичен форме регистрации, ведь создание пользователей, по сути, ничем не отличается от создания драгоценных камней, только поля разные.

дополняем forms.py

|  |
| --- |
| **from** django **import** forms  **from** .models **import** Gem  **class** GemsForm(forms.ModelForm):  **class** Meta:  model = Gem  fields = (**'\_\_all\_\_'**) |

дополняем views.py

|  |
| --- |
| **def** gems(request, category\_id):  gems = Gem.objects.filter(category\_\_id=category\_id)  categories = Category.objects.all()  **return** render(request, **'gems\_page.html'**, {**'categories'**: categories, **'gems'**: gems}) |

Думаю, не стоит подробно описывать, как дописать urls.py.

Создаем шаблон для отображения формы.

Обратите внимание, чтобы добавить новую категорию, придется добавить ещё одну форму. Это довольно не удобно. Как это сделать правильно, рассмотрим ниже.

Теперь мы можем добавлять объекты на наш сайт, используя удобные визуальные формы, но не стоит забывать про консоль, особенно, если начнёт происходить что-то непонятное. Консоль намного информативнее, привыкайте с ней работать, это сэкономит кучу времени и нервов.

После этого реализуем добавление, удаление, обновление информации о драгоценных камнях. В данном случае мы не будем использовать ajax, а реализуем более стандартный подход.

views.py

|  |
| --- |
| **def** admin\_gems(request):  gems = Gem.objects.all()  **return** render(request, **'admin\_gems.html'**, {**'gems'**: gems})  **def** admin\_gems\_create(request):  **if** request.method == **'POST'**:  form = GemsForm(request.POST)  **if** form.is\_valid():  form.save()  **return** HttpResponseRedirect(**'/admin/gems/'**)  context = {**'form'**: form}  **return** render(request, **'admin\_gems\_create.html'**, context)  context = {**'form'**: GemsForm()}  **return** render(request, **'admin\_gems\_create.html'**, context)  **def** admin\_gems\_delete(request, id):  gem = get\_object\_or\_404(Gem, id=id)  gem.delete()  **return** HttpResponseRedirect(**'/admin/gems/'**)  **def** admin\_gems\_update(request, id):  gem = get\_object\_or\_404(Gem, id=id)  **if** request.method == **'POST'**:  *# form = GemsForm(request.POST or None, instance=gem)*  form = GemsForm(request.POST, instance=gem)  **if** form.is\_valid():  gem.save()  **return** HttpResponseRedirect(**'/admin/gems/'**)  context = {**'form'**: form}  **return** render(request, **'admin\_gems\_update.html'**, context)  context = {**'form'**: GemsForm(instance=gem)}  **return** render(request, **'admin\_gems\_update.html'**, context)  **def** admin\_gems\_detail(request, id):  gem = get\_object\_or\_404(Gem, id=id)  **return** render(request, **'admin\_gem\_detail.html'**, {**'gem'**:gem}) |

urls.py

|  |
| --- |
| urlpatterns += [  url(**r'admin/gems/'**, admin\_gems, name=**'admin\_gems'**),  url(**r'^admin/create/gems$'**, admin\_gems\_create, name=**'gems\_create'**),  url(**r'^admin/delete/gems/(\d+)$'**, admin\_gems\_delete, name=**'gems\_delete'**),  url(**r'^admin/update/gems/(\d+)$'**, admin\_gems\_update, name=**'gems\_update'**),  url(**r'^admin/detail/gems/(\d+)$'**, admin\_gems\_detail, name=**'gems\_detail'**)  ] |

Шаблоны достаточно простые, их мы разбирали на предыдущих уроках (можно посмотреть в приложенном мною проекте)

# Пагинация

Когда камней станет достаточно много, они будут занимать очень много места, не стоит заставлять пользователя долго-долго прокручивать длиннющую страницу. В таких случаях отображаемую страницу разбивают на несколько страниц - это и есть пагинация.

## Теория

Django предоставляет несколько классов, которые помогают реализовать постраничный вывод данных, т.е. данных, распределённых на несколько страниц с ссылками «Предыдущая/Следующая». Эти классы располагаются в django/core/paginator.py.

Здесь приведён более сложный пример использования класса Paginator в представлении для постраничного вывода результатов запроса. Мы показали представление и его шаблон, чтобы показать, как вы можете отображать результаты. Данный пример предполагает наличие у вас модели Contacts, которая уже была импортирована.

## 

## 

## Практическая часть урока

Функция представления может выглядеть так:

|  |
| --- |
| from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger  def listing(request):  contact\_list = Contacts.objects.all()  # Show 25 contacts per page  paginator = Paginator(contact\_list, 25)  page = request.GET.get('page')  try:  contacts = paginator.page(page)  except PageNotAnInteger:  # If page is not an integer, deliver first page.  contacts = paginator.page(1)  except EmptyPage:  # If page is out of range (e.g. 9999), deliver last page of results.  contacts = paginator.page(paginator.num\_pages)  return render\_to\_response('list.html', {"contacts": contacts}) |

В шаблоне list.html подключен блок навигации по страницам:

|  |
| --- |
| {% for contact in contacts %}  {# Each "contact" is a Contact model object. #}  {{ contact.full\_name|upper }}<br />  ...  {% endfor %}  <div class="pagination">  <span class="step-links">  {% if contacts.has\_previous %}  <a href="?page={{ contacts.previous\_page\_number }}">previous</a>  {% endif %}  <span class="current">  Page {{ contacts.number }} of {{ contacts.paginator.num\_pages }}.  </span>  {% if contacts.has\_next %}  <a href="?page={{ contacts.next\_page\_number }}">next</a>  {% endif %}  </span>  </div> |

Объект Paginator подробно описан [здесь](http://djbook.ru/rel1.4/topics/pagination.html" \l "paginator-objects).

# 

# 

# Домашнее задание

Продолжаем разработку своего проекта:

1. Доработать свою админку, чтобы иметь возможность добавлять/удалять/редактировать ваши объекты.
2. Сделать отображение всех объектов на странице
3. Реализовать постраничный вывод. Добавить более 20-ти объектов, чтобы проверить пагинацию.
4. \*(усложненное) У вас будет длинное описание, что плохо смотрится при большом количестве отображаемых объектов. Используя фильтры, сократить отображаемый текст до 10-15 символов (постараться не обрезать слова).

Все проблемные или не до конца понятные элементы снабжайте комментариями. Что-то хотели сделать, но не получилось, не удаляйте код - закомментируйте, снабдив подробными комментариями. При проверке ДЗ, я постараюсь решить ваши проблемы и рассказать о наиболее частых проблемах в начале следующего занятия.

# Дополнительные материалы

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

1. [Создание форм из моделей](http://djbook.ru/rel1.6/topics/forms/modelforms.html)
2. [Пагинация](http://djbook.ru/rel1.4/topics/pagination.html" \l "paginator-objects)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Django Book(rus)](http://djbook.ru/)