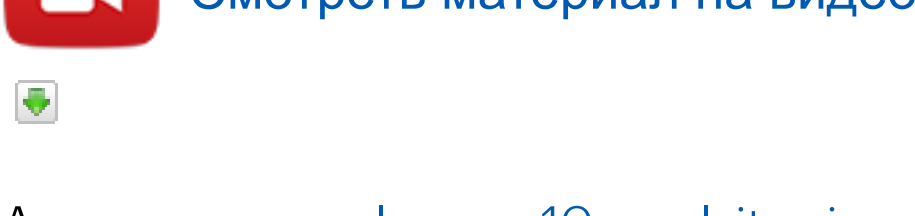


## Начинаем работу с админ-панелью



Архив проекта: [lesson-10-coolsite.zip](#)

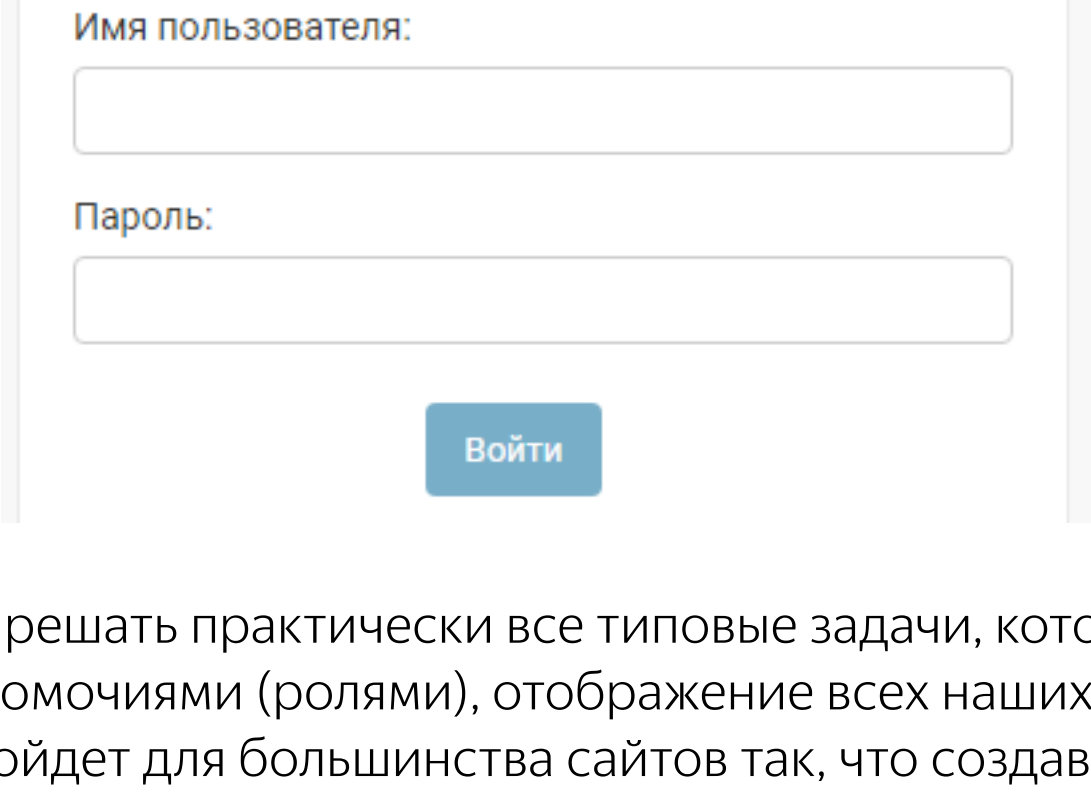
На этом занятии мы затронем еще один ключевой момент фреймворка Django – работа со встроенной админ-панелью. Она поставляется сразу «в коробке» и доступна для нашего сайта по адресу:

`http://127.0.0.1:8000/admin/`

Причем, по умолчанию, фреймворк использует английский язык. Чтобы переключиться на русскую локализацию, нужно в пакете конфигурации в файле `settings.py` изменить константу `LANGUAGE_CODE`, установив ее в значение:

```
LANGUAGE_CODE = 'ru'
```

И при обновлении страницы увидим русскоязычный вариант админ-панели.



С помощью встроенной админки можно решать практически все типовые задачи, которые нам могут потребоваться. Это и создание пользователей с разными полномочиями (ролями), отображение всех наших приложений, добавление, удаление и изменение записей и так далее. Она подойдет для большинства сайтов так, что создавать свою админ-панель в Django попросту нет необходимости.

Итак, для входа нам нужно ввести имя пользователя и пароль. Но у нас пока их нет и вообще нет ни одного зарегистрированного пользователя в админ-панели. Поэтому, сначала необходимо создать суперпользователя – администратора сайта. Для этого я перейду в терминал и в новой вкладке консоли выполню команду (из каталога проекта `coolsite`):

```
python manage.py createsuperuser
```

Здесь нам будет предложено указать логин (имя пользователя). В учебных целях я укажу `root`. Далее, нужно ввести адрес электронной почты, пусть это будет `root@coolsite.ru`. (Почта может быть не существующей в учебном проекте, а вообще, нужно указывать адрес реальной почты, на которую будут приходить служебные сообщения). И, наконец, вводим пароль (1234) и повторяем его. Конечно, такой простой пароль и имя пользователя я использую сугубо в рамках этих занятий. В реальности старайтесь указывать необычные, уникальные имена и большой, сложный пароль, иначе, получив доступ к админ-панели, злоумышленник легко нарушит работу вашего сайта.

Все, суперпользователь создан и мы теперь можем войти в админку, указав `root` и пароль `1234`. На главной странице мы видим два зарегистрированных приложения: «Группы» и «Пользователи». То есть, мы можем создавать новых пользователей уже непосредственно здесь, а также назначать им определенные группы.

Однако, здесь нет нашего приложения `women`. Почему? Дело в том, что его нужно зарегистрировать, то есть, указать, что мы хотим его видеть в админ-панели. Для этого откроем файл `women/admin.py`, который отвечает за взаимосвязь приложения с админкой, где мы будем регистрировать наши модели. Вначале импортируем их:

```
from .models import *
```

и зарегистрируем модель `Women`:

```
admin.site.register(Women)
```

Возвращаемся в админку и при обновлении главной страницы видим наше приложение и зарегистрированную модель `Women`. Если щелкнуть по ней, то видим список наших записей и здесь же можем их редактировать, удалять и добавлять новые. Это очень удобно.

И, смотрите, при выборе определенной записи, мы видим кнопку «Смотреть на сайте». Эта кнопка появляется только тогда, когда в модели определена функция `get_absolute_url()`. Если мы ее прокомментируем и обновим админку, то эта кнопка пропадет. Так, что эта функция достаточно часто используется в различных модулях фреймворка Django.

Давайте немного настроим админ-панель, чтобы она выглядела приятнее и дружелюбнее. Первое, что мы сделаем – это заменим название `Women` на «Известные женщины». Для этого перейдем в файл с моделями (`women/models.py`) и в классе `Women` пропишем специальный вложенный класс `Meta`, который используется админкой для более тонкой настройки модели:

```
class Women(models.Model):
    ...
    class Meta:
        verbose_name = 'Известные женщины'
```

Здесь `verbose_name` – это специальный атрибут, отвечающий за название модели. Однако, мы видим, что админ-панель по-прежнему добавляет латинскую `s` в конец нашего имени, то есть, пытается сделать его множественное число. Чтобы явно указать, как будет звучать название во множественном числе, определим еще один атрибут:

```
class Meta:
    verbose_name = 'Известные женщины'
    verbose_name_plural = 'Известные женщины'
```

Так стало гораздо лучше. Добавим сортировку записей по дате их создания и по заголовку с помощью еще одного атрибута `ordering`:

```
class Meta:
    verbose_name = 'Известные женщины'
    verbose_name_plural = 'Известные женщины'
    ordering = ['time_create', 'title']
```

Как происходит упорядочение по нескольким полям? Сначала идет сортировка по первому полю (в данном случае дате создания), но если даты оказываются равными, то учитывается следующее поле (заголовок) и так далее. Перейдем в админ-панель и видим, что особо ничего не поменялось, записи и так шали в порядке возрастания времени создания. Изменим сортировку по первому полю на противоположную (ставим знак минус):

```
ordering = ['-time_create', 'title']
```

и порядок отображения постов также поменялся. Причем, обратите внимание, указанная сортировка будет также влиять на порядок отображения статей и на сайте. Если перейти на главную страницу, то увидим посты в той же последовательности, что и в админке.

Далее, мы поменяем отображение заголовка `women` приложения на другой – «Женщины мира». Для этого перейдем в файл `women/apps.py` и в классе конфигурации приложения пропишем атрибут:

```
class WomenConfig(AppConfig):
    name = 'women'
    verbose_name = 'Женщины мира'
```

Здесь тоже обратите внимание, что все это работает благодаря подключению приложения в пакете конфигурации (файл `settings.py`) через этот класс:

```
INSTALLED_APPS = [
    ...
    'women.apps.WomenConfig',
]
```

Если бы мы написали просто `'women'`, то заголовок «Женщины мира» в админ-панели не появился бы.

Следующим шагом добавим в списке записей дополнительные поля: `id`, `title`, время создания, изображение, флаг публикации. Для этого нужно открыть файл `women/admin.py` и объявить класс, унаследованный от `ModelAdmin`:

```
class WomenAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'time_create', 'photo', 'is_published')
    list_display_links = ('id', 'title')
    search_fields = ('title', 'content')
```

Здесь в атрибуте `list_display` указываем список отображаемых полей, в атрибуте `list_display_links` – список полей в виде ссылки для перехода к конкретной записи, а атрибут `search_fields` определяет поля, по которым можно будет производить поиск записей. (В этом классе есть и другие атрибуты, вы можете изучить их самостоятельно).

Зарегистрируем этот класс в функции `register`:

```
admin.site.register(Women, WomenAdmin)
```

причем, `WomenAdmin` должен обязательно идти после класса модели `Women`. Возвращаемся в админку и видим все эти поля. Но они имеют английское написание, а нам бы хотелось иметь русские названия. Поправим. Перейдем в класс определения модели `Women` (`women/models.py`) и у каждого поля в конструкторе соответствующих классов добавим именованный параметр `verbose_name`:

```
class Women(models.Model):
    title = models.CharField(max_length=255, verbose_name="Заголовок")
    content = models.TextField(blank=True, verbose_name="Текст статьи")
    photo = models.ImageField(upload_to="photos/%Y/%m/%d/", verbose_name="Фото")
    time_create = models.DateTimeField(auto_now_add=True, verbose_name="Время создания")
    time_update = models.DateTimeField(auto_now=True, verbose_name="Время изменения")
    is_published = models.BooleanField(default=True, verbose_name="Публикация")
    cat = models.ForeignKey('Category', on_delete=models.PROTECT, null=True, verbose_name="Категории")
    ...
```

Теперь имена полей отображаются с нашими значениями.

По аналогии зарегистрируем вторую модель `Category`. В файле `women/admin.py` пропишем:

```
class CategoryAdmin(admin.ModelAdmin):
    list_display = ('id', 'name')
    list_display_links = ('id', 'name')
    search_fields = ('name',)
```

И вызовем функцию `register`:

```
admin.site.register(Category, CategoryAdmin)
```

Переходим в админку и видим, что появилось второе приложение с набором указанных полей. Также скорректируем все названия. Перейдем в файл `women/views.py` и в класс `Category` добавим вложенный класс `Meta`:

```
class Meta:
    verbose_name = 'Категория'
    verbose_name_plural = 'Категории'
    ordering = ['id']
```

Кроме того, у поля `name` добавим параметр `verbose_name`:

```
name = models.CharField(max_length=100, db_index=True, verbose_name="Категория")
```

Все, теперь мы можем полноценно работать и с рубриками в нашей админ-панели.

Однако, измененное метаописание полей модели (параметр `verbose_name`) желательно внести и в таблицы БД. Создадим еще один файл миграции командой:

```
python manage.py makemigrations
```

и, затем, применим все миграции:

```
python manage.py migrate
```

Все, в целом админка у нас настроена и давайте теперь добавим новую запись. Введем в поле `title` «Ариана Гранде», заполним другие поля, кроме картинки и нажмем добавить. Админка сообщит об ошибке, т.к. поле `photo` у нас идет как обязательное. Поэтому выберем картинку и сохраним. Запись была добавлена успешно, в поле `photo` мы видим путь к изображению, а в проекте появилась папка `media` с загруженной фотографией.

Видите как это удобно: нам не нужно вручную прописывать механизм загрузки изображений, фреймворк Django все берет на себя и делает это автоматически. Это очень круто! Правда, здесь может возникнуть одна проблема, когда загружаются изображения с разными именами и попадают в одну папку. Как это обойти мы позже разберем. А сейчас добавим отображение картинок в списке статей на главной странице. Делается это очень просто, в шаблон `index.html` добавим строчки:

```
{% if p.photo %}
    <p></p>
{% endif %}
```

Мы здесь вначале проверяем: имеется ли вообще какой-либо путь к изображению (так как не у всех наших записей он присутствует), а затем, используем встроенный атрибут `url`, который возвращает корректный URL-адрес к текущему изображению. Я загружу все изображения для каждой статьи и получим уже более полноценное отображение постов.

Давайте еще немного улучшим нашу админку и сделаем так, чтобы поле `is_published` можно было редактировать прямо в списке статей. Выполняется это очень просто. В классе `WomenAdmin` (файл `women/admin.py`) добавим еще один атрибут:

```
class WomenAdmin(admin.ModelAdmin):
    ...
    list_editable = ('is_published',)
```

И теперь в списке статей в панели появились чекбоксы, где мы можем убирать или ставить галочки, то есть, поле стало редактируемым.

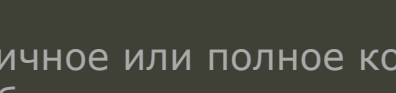
Далее, добавим поля, по которым сможем фильтровать наш список статей. Опять же в классе `WomenAdmin` прописываем атрибут:

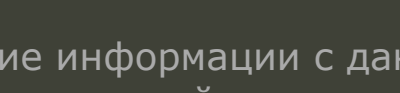
```
list_filter = ('is_published', 'time_create')
```

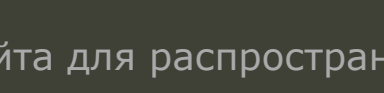
где указываем список полей для фильтрации. Переходим в админку и видим, что справа появилась панель с возможностью выбирать записи по публикации и времени создания.

Вот так в базовом варианте можно выполнять настройку админ-панели и работать через нее с записями таблиц БД.

### Видео по теме

- 

#1. Django - что это такое, порядок установки
- 

#2. Модель MTV. Функции представления
- 

#3. Маршрутизация обработки исключен запросов,