Создаём своё первое приложение с Django, часть 2

Мы настроим базу данных, создадим свою первую модель и посмотрим на автоматически созданный интерфейс администратора.

Настройка базы данных

Теперь откроем файл mysite/settings.py. Это обычный модуль Python с набором переменных, которые представляют настройки Django.

По умолчанию в настройках указано использование SQLite. Если вы новичок в базах данных, или просто хотите попробовать Django, это самый простой выбор. SQLite уже включен в Python, и вам не нужно дополнительно что-то устанавливать. Однако, создавая свой первый настоящий проект, вам понадобиться более серьезная база данных, например, PostgreSQL.

Если вы хотите использовать другую базу данных, установите *необходимые библиотеки (см. в главе: «Быстрое руководство по установке:- Установка базы данных:- Создание рабочей базы данных»)* и поменяйте следующие ключи в элементе 'default' настройки DATABASES, чтобы они соответствовали настройкам подключения к вашей базе данных:

- ENGINE – один
  из 'django.db.backends.sqlite3', 'django.db.backends.postgresql', 'django.db.backends.mysql' или ' django.db.backends.oracle';

  По умолчанию: '' (Пустая строка)

  Бэкенд базы данных. Django предоставляет следующие бэкенды:

- 'django.db.backends.postgresql'
- 'django.db.backends.mysql'
- 'django.db.backends.sqlite3'
- 'django.db.backends.oracle'

Вы можете использовать сторонние бэкэнды указав в ENGINE путь для импорта (например, mypackage.backends.whatever).

- NAME – название вашей базы данных. Если вы используете SQLite, база данных будет файлов на вашем компьютере, в этом случае NAME содержит абсолютный путь, включая имя, к этому файлу. Значение по умолчанию, os.path.join(BASE_DIR, 'db.sqlite3'), создаст файл в каталоге вашего проекта.

  По умолчанию: '' (Пустая строка)

  Название используемой базы данных. Для SQLite – это полный путь к файлу базы данных. При указывании пути к файлу, всегда используйте обратные слэшы, даже на Windows (например, C:/homes/user/mysite/sqlite3.db).

  Если вы используете не SQLite, вам необходимо указать дополнительно

- USER;

  По умолчанию: '' (Пустая строка)

  Имя пользователя, используемое при подключении к базе данных. Не используется для SQLite.

- PASSWORD;

  По умолчанию: '' (Пустая строка)

  Пароль, используемый при подключении к базе данных. Не используется для SQLite.

- HOST.

  По умолчанию: '' (Пустая строка)

  Имя хоста используемого при подключении к базе данных. Пустая строка подразумевает localhost. Не используется для SQLite.

  Если значение начинается с прямого слэша ('/') и используется MySQL, MySQL будет подключаться через Unix сокет к указанному сокету. Например,

```
"HOST": '/var/run/mysql'
```

  Если вы используете MySQL и значение не начинается с прямого слэша, значение будет использоваться как имя хоста.

  При использовании PostgreSQL, по умолчанию (при пустом HOST) подключение к базе данных будет выполнено через UNIX domain сокет ('local' в pg_hba.conf). Если у вас UNIX domain сокеты находятся не в стандартном каталоге, укажите в этой настройке значение unix_socket_directory из postgresql.conf. Если вы хотите использовать TCP сокеты, укажите в HOST 'localhost' или '127.0.0.1' ('host' в pg_hba.conf). В Windows необходимо указать HOST, т.к. UNIX domain сокеты не доступны.

Примечание

Если вы используете PostgreSQL или MySQL, убедитесь, что вы создали базу данных. Вы можете сделать это, выполнив "CREATE DATABASE database_name;" в консоли вашей базы данных.

Если вы используете SQLite, вам ничего не нужно создавать заранее - файл базы данных будет создан автоматически при необходимости.

Отредактируйте mysite/settings.py и укажите в TIME_ZONE ваш часовой пояс.

TIME_ZONE:

По умолчанию: 'America/Chicago'
Часовой пояс, который будет использоваться в проекте, или None.

Примечание

При первом релизе TIME_ZONE по умолчанию была 'America/Chicago', глобальные настройки (которые используется, если settings.py не содержит настройки) содержат значение 'America/Chicago' для обратной совместимости. Шаблон нового проекта содержит значение 'UTC'.

Заметим, что указанный часовой пояс не обязан совпадать с часовым поясом сервера. Например, один сервер может обслуживать несколько Django-проектов, каждый может использовать свой часовой пояс.

Если USE_TZ равна False, Django будет использовать указанный часовой пояс при сохранении времени. При USE_TZ равной True, указанный часовой пояс будет использоваться по умолчанию при отображении даты в шаблоне и при интерпретации введенных через форму значений.

USE_TZ:

По умолчанию: False
Указывает, используется ли часовой пояс. При True, Django будет использовать объекты даты и времени с указанным часовым поясом. Иначе Django будет использовать объекты даты и времени без учета часового пояса.

Django устанавливает переменной os.environ['TZ'] значение, указанное в настройке TIME_ZONE. Таким образом все ваши представления и модели будут использовать один и тот же часовой пояс. Однако, Django не установит значение переменной окружения TZ при следующих обстоятельствах:

- При ручной конфигурации настроек как описано в *соответствующем разделе*:

Использование настроек без DJANGO_SETTINGS_MODULE:

В некоторых случаях вы можете не использовать переменную окружения DJANGO_SETTINGS_MODULE. Например, если вы используете систему шаблонов без Django, вряд ли вы захотите настраивать переменную окружения для настроек.

В этом случае вы можете явно определить настройки Django. Для этого вызовите:

django.conf.settings.configure**(***default_settings***, *****settings***)**

Например,

```
from django.conf import settings

settings.configure(DEBUG=True)
```

Вы можете передать в configure() любое количество именованных аргументов, которые представляют настройки и их значения. Названия аргументов должны быть в верхнем регистре, как и названия настроек. Если какая-то настройка не была указана в configure(), Django при необходимости будет использовать значение по умолчанию.

Настройка Django таким способом необходима, и рекомендуется использовать, при использовании части фреймверка в другом приложении.

При настройке через settings.configure() Django не меняет переменные окружения (смотрите описание TIME_ZONE, чтобы узнать, когда это происходит). Подразумевается, что вы уже полностью контролируете свое окружение.

- Или если указать TIME_ZONE = None, Django будет использовать системную временную зону. Но при USE_TZ = True конвертация локального времени в UTC будет не такая однозначная.

Если Django не определяет переменную окружения TZ, необходимо самостоятельно убедиться, что ваш код выполняется в правильном окружении.

Также обратите внимание на настройку INSTALLED_APPS в начале файла. Она содержит названия всех приложений Django, которые активированы в вашем проекте. Приложения могут использоваться на разных проектах, вы можете создать пакет, распространить его и позволить другим использовать его на своих проектах.

INSTALLED_APPS:

По умолчанию: [] (Пустой список)

Список строк, который указывают не все приложения Django, используемые в проекте. Каждая строка должна быть полным Python путем к:

- классу настройки приложения(рекомендуется), или
- пакету с приложением.

Используйте реестр приложений

Ваш код не должен использовать INSTALLED_APPS. Вместо этого используйте django.apps.apps.

Если несколько приложений содержат разные версии одних и тех же ресурсов (шаблоны, статические файлы, команды, файлы перевода), будут использоваться ресурсы из приложения, которое указано выше в INSTALLED_APPS.

По умолчанию, INSTALLED_APPS содержит следующие приложение, которые предоставляются Django:

- django.contrib.admin – интерфейс администратора. Скоро мы его будем использовать;
- django.contrib.auth – система авторизации;
- django.contrib.contenttypes – фреймверк типов даных;
- django.contrib.sessions – фреймверк сессии;
- django.contrib.messages – фреймверк сообщений;
- django.contrib.staticfiles – фреймверк для работы со статическими файлами.

Эти приложения включены по умолчанию для покрытия основных задач.

Некоторые приложения используют минимум одну таблицу в базе данных, поэтому нам необходимо их создать перед тем, как мы будем их использовать. Для этого выполним следующую команду:

```
$ python manage.py migrate
```

Команда migrate проверяет настройку INSTALLED_APPS и создает все необходимые таблицы в базе данных, указанной в mysite/settings.py, применяя миграции, которые находятся в приложении (мы расскажем об это ниже). Вы увидите сообщение для кажой примененной миграции. Если вам интересно, запустите консоль базы данных и введите \dt (PostgreSQL), SHOW TABLES; (MySQL), .schema (SQLite),
или SELECT TABLE_NAME FROM USER_TABLES; (Oracle), чтобы посмотреть таблицы, которые создал Django.

Создание моделей

Теперь создадим ваши модели – по сути структуру вашей базы данных с дополнительными мета-данными.

Философия

Модель - это основной источник данных. Он содержит набор полей и поведение данных, которые вы храните. Django следует *принципу DRY* (Каждая отдельная концепция и/или кусок данных следует хранить в одном и лишь одном месте. Избыточность - плохо. Нормализация - хорошо. Фреймворк, в пределах разумного, должен использовать как можно больше из как можно меньшего.) Смысл в том, чтобы определять модели в одном месте.

Частью работы с данными также являются миграции. В отличии от Ruby On Rails, например, миграции вынесены из файла моделей и являются просто историей, которую Django может использовать для изменения базы данных в соответствии с текущей структурой моделей.

В нашем простом приложении голосования, мы создадим две модели: Question и Choice. Questionсодержит вопрос и дату публикации. Choice содержит: текст ответа и количество голосов. Каждый объект Choice связан с объектом Question.

Эти понятия отображаются простыми классами Python. Отредактируйте файл polls/models.py, чтобы он выглядел таким образом:

polls/models.py

```
from django.db import models


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Код очень простой. Каждая модель представлена классом, унаследованным от django.db.models.Model. Каждая модель содержит несколько атрибутов, каждый из которых отображает поле в таблице базы данных.

Каждое поле представлено экземпляром класса Field – например, CharField для текстовых полей и DateTimeField для полей даты и времени. Это указывает Django какие типы данных хранят эти поля.

Названия каждого экземпляра Field (например, question_text или pub_date) это название поля, в “машинном”(machine-friendly) формате. Вы будете использовать эти названия в коде, а база данных будет использовать их как названия колонок.

Вы можете использовать первый необязательный аргумент конструктора класса Field, чтобы определить отображаемое, удобное для восприятия, название поля. Оно используется в некоторых компонентах Django, и полезно для документирования. Если это название не указано, Django будет использовать “машинное” название. В этом примере, мы указали отображаемое название только для поля Question.pub_date. Для всех других полей будет использоваться “машинное” название.

Некоторые классы, унаследованные от Field, имеют обязательные аргументы. Например, CharField требует, чтобы вы передали ему max_length. Это используется не только в схеме базы данных, но и при валидации, как мы скоро увидим.

Field может принимать различные необязательные аргументы; в нашем примере мы указали default значение для votes` равное 0.

Заметим, что связь между моделями определяется с помощью ForeignKey. Это указывает Django, что каждый Choice связан с одним объектом Question. Django поддерживает все основные типы связей: многие-к-одному, многие-ко-многим и один-к-одному.

Активация моделей

Эта небольшая часть кода моделей предоставляет Django большое количество информации, которая позволяет Django:

- Создать структуру базы данных (CREATE TABLE) для приложения;
- Создать Python API для доступа к данным объектов Question и Choice.

Но первым делом мы должны указать нашему проекту, что приложение polls установлено.

Отредактируйте файл mysite/settings.py и измените настройку INSTALLED_APPS добавив строку 'polls'. В результате получим:

```
mysite/settings.py
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Теперь Django знает, что необходимо использовать приложение polls. Давайте выполним следующую команду:

```
$ python manage.py makemigrations polls
```

Вы увидите приблизительно такое:

```
Migrations for 'polls':
  0001_initial.py:
    - Create model Choice
    - Create model Question
    - Add field question to choice
```

Выполняя makemigrations, вы говорите Django, что внесли некоторые изменения в ваши модели (в нашем случае мы создали несколько новых) и хотели бы сохранить их в *миграции*.

Миграции используются Django для сохранения изменений ваших моделей (и структуры базы данных) - это просто файлы на диске. Вы можете изучить миграцию для создания ваших моделей, она находится в файле polls/migrations/0001_initial.py. Не волнуйтесь, вам не нужно каждый раз их проверять. Но их формат удобен для чтения на случай, если вы захотите внести изменения.

В Django есть команда, которая выполняет миграции и автоматически обновляет базу данных - она называется migrate. Мы скоро к ней вернемся, но сначала давайте посмотрим какой SQL выполнит эта миграция. Команда sqlmigrate получает название миграции и возвращает SQL:

```
$ python manage.py sqlmigrate polls 0001
```

Вы увидите приблизительно такое (мы отформатировали результат для читабельности):

```
BEGIN;
--
-- Create model Choice
--
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL
);
--
-- Create model Question
--
CREATE TABLE "polls_question" (
    "id" serial NOT NULL PRIMARY KEY,
    "question_text" varchar(200) NOT NULL,
    "pub_date" timestamp with time zone NOT NULL
);
--
-- Add field question to choice
--
ALTER TABLE "polls_choice" ADD COLUMN "question_id" integer NOT NULL;
ALTER TABLE "polls_choice" ALTER COLUMN "question_id" DROP DEFAULT;
CREATE INDEX "polls_choice_7aa0f6ee" ON "polls_choice" ("question_id");
ALTER TABLE "polls_choice"
  ADD CONSTRAINT "polls_choice_question_id_246c99a640fbbd72_fk_polls_question_id"
    FOREIGN KEY ("question_id")
    REFERENCES "polls_question" ("id")
    DEFERRABLE INITIALLY DEFERRED;

COMMIT;
```

Обратите внимание на следующее:

- Полученные запросы зависят от базы данных, которую вы используете. Пример выше получен для PostgreSQL;
- Названия таблиц созданы автоматически из названия приложения(polls) и названия модели в нижнем регистре – question и choice. (Вы можете переопределить это.);
- Первичные ключи (ID) добавлены автоматически. (Вы можете переопределить и это.);
- Django добавляет "_id" к названию внешнего ключа. (Да, вы можете переопределить и это.);
- Связь явно определена через FOREIGN KEY constraint. Не волнуйтесь о DEFERRABLE, это просто указание для PostgreSQL не применять ограничения FOREIGN KEY до окончания транзакции;
- Учитываются особенности базы данных, которую вы используете. Специфические типы данных такие как auto_increment (MySQL), serial (PostgreSQL), или integer primary key(SQLite) будут использоваться автоматически. Тоже касается и экранирование названий, что позволяет использовать в названии кавычки – например, использование одинарных или двойных кавычек;

- Команда sqlmigrate не применяет миграцию к базе данных - она просто выводит запросы на экран, чтобы вы могли увидеть какой SQL создает Django. Это полезно, если вы хотите проверить что выполнит Django, или чтобы предоставить вашему администратору базы данных SQL скрипт.

Если необходимо, можете выполнить python manage.py check. Эта команда ищет проблемы в вашем проекте, не применяя миграции и не изменяя базу данных.

Теперь, выполните команду migrate снова, чтобы создать таблицы для этих моделей в базе данных:

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, contenttypes, polls, auth, sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

Команда migrate выполняет все миграции, которые ещё не выполнялись, (Django следит за всеми миграциями, используя таблицу в базе данных django_migrations) и применяет изменения к базе данных, синхронизируя структуру базы данных со структурой ваших моделей.

Миграции - очень мощная штука. Они позволяют изменять ваши модели в процессе развития проекта без необходимости пересоздавать таблицы в базе данных. Их задача изменять базу данных без потери данных. Мы ещё вернемся к ним, а пока запомните эти инструкции по изменению моделей:

- Внесите изменения в модели (в models.py).
- Выполните python manage.py makemigrations чтобы создать миграцию для ваших изменений
- Выполните python manage.py migrate чтобы применить изменения к базе данных.

Две команды необходимы для того, чтобы хранить миграции в системе контроля версий.

Поиграемся с API

Теперь, давайте воспользуемся консолью Python и поиграем с API, которое предоставляет Django. Чтобы запустить консоль Python выполните:

```
$ python manage.py shell
```

Мы используем эту команду вместо просто “python”, потому что manage.py устанавливает переменную окружения DJANGO_SETTINGS_MODULE, которая указывает Django путь импорта для файла mysite/settings.py.

Запуск без manage.py

Если вы не желаете использовать manage.py, не проблема. Просто установите переменную окружения DJANGO_SETTINGS_MODULE в mysite.settings, запустите интерпретатор Python и инициализируйте Django:

```
>>> import django
>>> django.setup()
```

Если вы получили исключение AttributeError, возможно вы используете версию Django, которая не соответствует этому учебнику. Вам следует читать старую версию учебника или обновить Django. Запустить python необходимо в каталоге, в котором находится файл manage.py (или убедитесь, что каталог находится в путях Python, и import mysite работает).

Теперь, когда вы в консоли, исследуем API базы данных:

```
>>> from polls.models import Question, Choice   # Import the model classes we just wrote.

# No questions are in the system yet.
>>> Question.objects.all()
[]
# Create a new Question.
# Support for time zones is enabled in the default settings file, so
# Django expects a datetime with tzinfo for pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.
>>> q.save()

# Now it has an ID. Note that this might say "1L" instead of "1", depending
# on which database you're using. That's no biggie; it just means your
# database backend prefers to return integers as Python long integer
# objects.
>>> q.id

# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)
# Change values by changing the attributes, then calling save().
>>> q.question_text = "What's up?"
>>> q.save()

# objects.all() displays all the questions in the database.
>>> Question.objects.all()
[<Question: Question object>]
```

Одну минуту. <Question: Question object> – крайне непрактичное отображение объекта. Давайте исправим это, отредактировав модель Question (в файле polls/models.py) и добавив метод __str__() для моделей Question и Choice:

```
polls/models.py
from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible  # only if you need to support Python 2
class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

@python_2_unicode_compatible  # only if you need to support Python 2
class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Важно добавить метод __str__() не только для красивого отображения в консоли, но так же и потому, что Django использует строковое представление объекта в интерфейсе администратора.

Заметим, это стандартные методы Python. Давайте добавим свой метод, просто для демонстрации:

```
polls/models.py
import datetime

from django.db import models
from django.utils import timezone


class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

Мы добавили import datetime и from django.utils import timezone для использования стандартной библиотеки Python datetime и модуля Django для работы с временными зонами django.utils.timezone соответственно.

Сохраните эти изменения и запустите консоль Python снова, выполнив python manage.py shell:

```
>>> from polls.models import Question, Choice

# Make sure our __str__() addition worked.
>>> Question.objects.all()
[<Question: What's up?>]

# Django provides a rich database lookup API that's entirely driven by
# keyword arguments.
>>> Question.objects.filter(id=1)
[<Question: What's up?>]
>>> Question.objects.filter(question_text__startswith='What')
[<Question: What's up?>]

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>
```

```
    # Request an ID that doesn't exist, this will raise an exception.
    >>> Question.objects.get(id=2)
    Traceback (most recent call last):
        ...
    DoesNotExist: Question matching query does not exist.

    # Lookup by a primary key is the most common case, so Django provides a
    # shortcut for primary-key exact lookups.
    # The following is identical to Question.objects.get(id=1).
    >>> Question.objects.get(pk=1)
    <Question: What's up?>

    # Make sure our custom method worked.
    >>> q = Question.objects.get(pk=1)
    >>> q.was_published_recently()
    True

    # Give the Question a couple of Choices. The create call constructs a new
    # Choice object, does the INSERT statement, adds the choice to the set
    # of available choices and returns the new Choice object. Django creates
    # a set to hold the "other side" of a ForeignKey relation
    # (e.g. a question's choice) which can be accessed via the API.
    >>> q = Question.objects.get(pk=1)

    # Display any choices from the related object set -- none so far.
    >>> q.choice_set.all()
    []

    # Create three choices.
    >>> q.choice_set.create(choice_text='Not much', votes=0)
    <Choice: Not much>
    >>> q.choice_set.create(choice_text='The sky', votes=0)
    <Choice: The sky>
    >>> c = q.choice_set.create(choice_text='Just hacking again', votes=0)

    # Choice objects have API access to their related Question objects.
    >>> c.question
    <Question: What's up?>

    # And vice versa: Question objects get access to Choice objects.
    >>> q.choice_set.all()
    [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]
    >>> q.choice_set.count()
    3

    # The API automatically follows relationships as far as you need.
    # Use double underscores to separate relationships.
    # This works as many levels deep as you want; there's no limit.
    # Find all Choices for any question whose pub_date is in this year
    # (reusing the 'current_year' variable we created above).
    >>> Choice.objects.filter(question__pub_date__year=current_year)
    [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]

    # Let's delete one of the choices. Use delete() for that.
    >>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
    >>> c.delete()
```

Введение в интерфейс администратор Django

Философия

Создание интерфейса администратора для добавления, изменения и удаления содержимого сайта – в основном скучная не креативная задача. Django значительно автоматизирует и упрощает эту задачу.

Django создавался для новостных сайтов, у которых есть разделение между публичными страницами и интерфейсом администратора. Менеджеры используют последний для добавления новостей и другого содержимого сайта, это содержимое отображается на сайте. Django позволяет легко создать универсальный интерфейс для редактирования содержимого сайта.

Интерфейс администратора не предназначен для использования пользователями. Он создан для менеджеров и администраторов сайта.

Создание суперпользователя

Первым делом необходимо создать пользователя, который может заходить на интерфейс администратора. Выполните следующую команду:

```
$ python manage.py createsuperuser
```

Введите имя пользователя и нажмите Enter.

```
Username: admin
```

Теперь введите email:

```
Email address: admin@example.com
```

И наконец введите пароль.

```
Password: **********
Password (again): *********
Superuser created successfully.
```

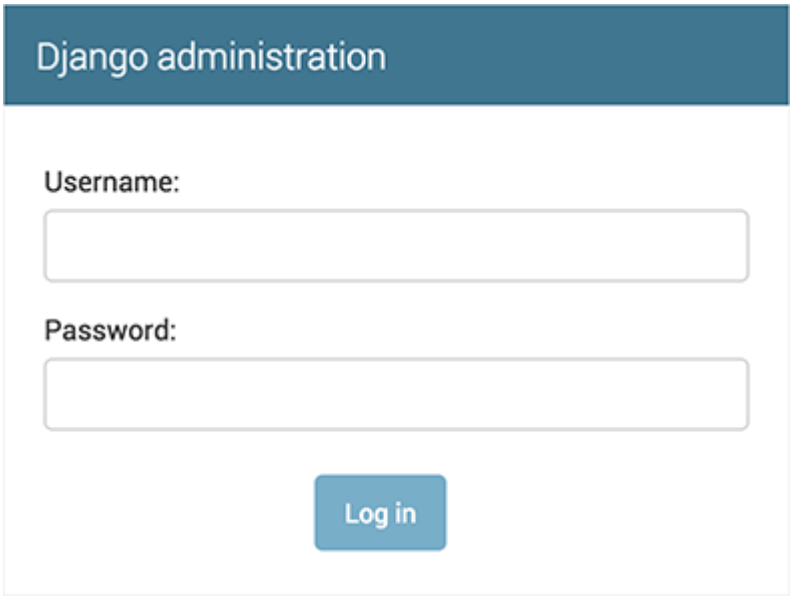
Запускаем сервер для разработки

Интерфейс администратора включен по умолчанию. Давайте запустим встроенный сервер для разработки и посмотрим на него.

Если сервер не запущен, выполните:

```
$ python manage.py runserver
```

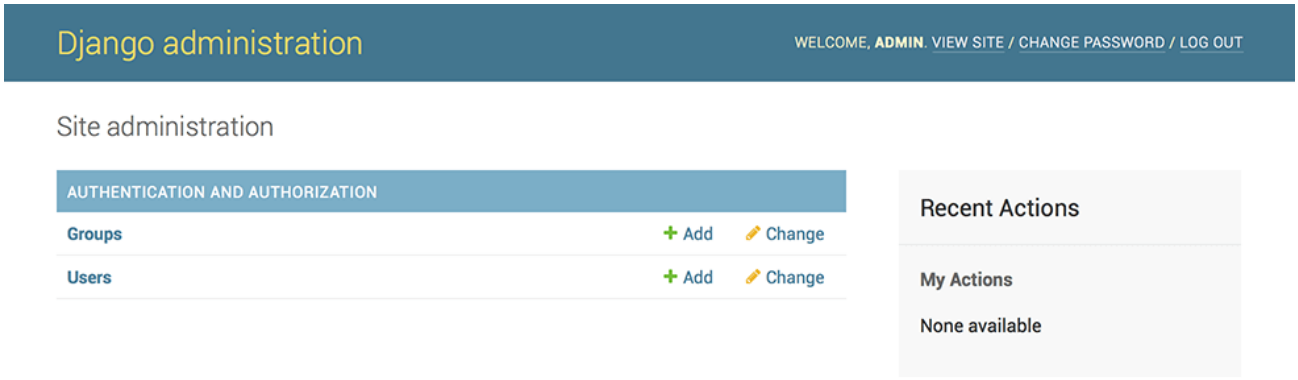
Откроем “/admin/” локального домена в браузере – например, http://127.0.0.1:8000/admin/. Вы должны увидеть страницу авторизации интерфейса администратора:



Т.к. translation включен по умолчанию, страница авторизации может быть на вашем родном языке, зависит от настроек браузера и наличия перевода для вашего языка.

Заходим в интерфейс администратора

Теперь попробуйте войти в админку. Вы должны видеть следующую страницу Django:



Вы должны увидеть несколько разделов: группы и пользователи. Они предоставлены приложением авторизации Django django.contrib.auth.

Добавим приложение голосования в интерфейс администратора

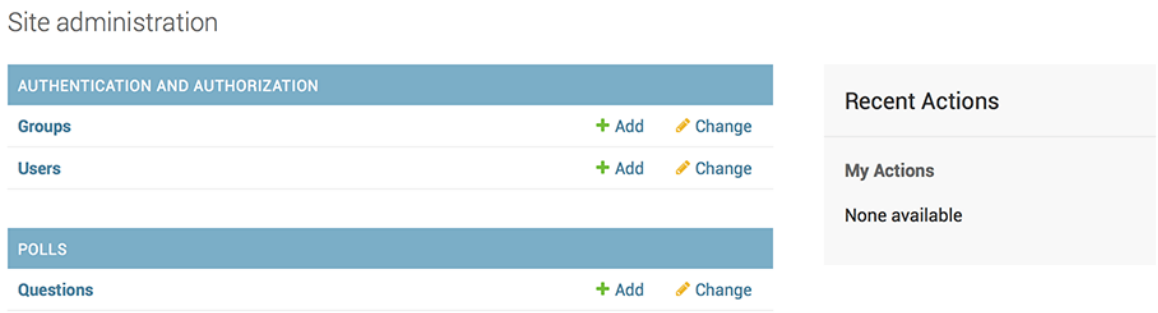
А где же наше приложение голосования? Оно не отображается в интерфейсе администратора.

Нам нужно указать, что объекты модели Question могли редактироваться в интерфейсе администратора. Для этого создадим файл polls/admin.py, и отредактируем следующим образом:

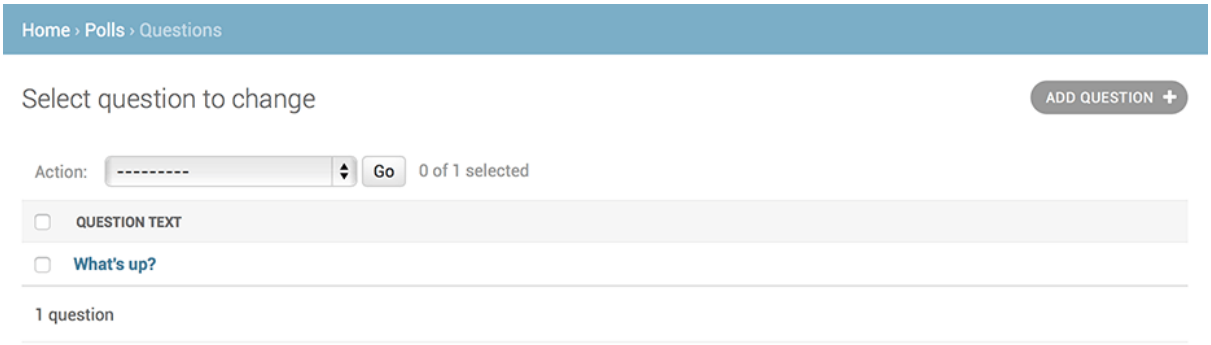
```
polls/admin.py
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

Изучим возможности интерфейса администратора

После регистрации модели Question Django отобразит ее на главной странице:

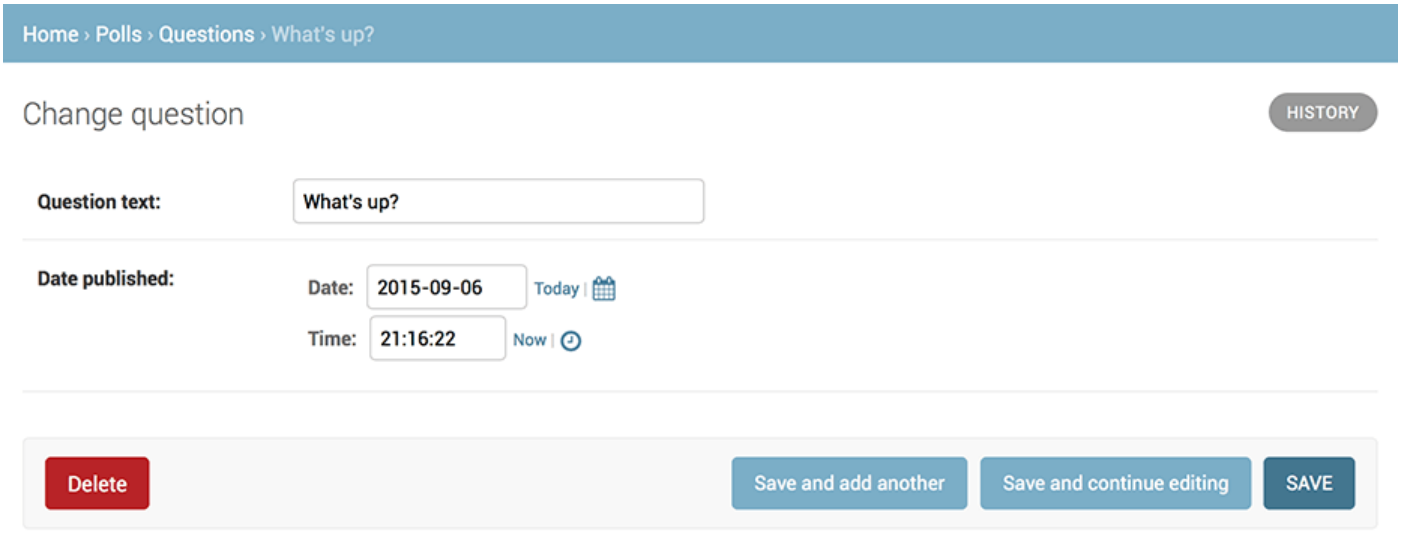


Нажмите “Questions”. Вы попали на страницу “списка объектов” для голосований. Эта страница содержит все объекты из базы данных и позволяет выбрать один из них для редактирования. Мы видим голосование “What’s up?”, которое создали в первой части учебника:



Нажмите “What’s up?” чтобы отредактировать его:

Заметим:



- Поля формы формируются на основе описания модели Question;
- Для различных типов полей модели (DateTimeField, CharField) используются соответствующие HTML поля. Каждое поле знает, как отобразить себя в интерфейсе администратора;
- К полям DateTimeField добавлен JavaScript виджет. Для даты добавлена кнопка “Сегодня” и календарь, для времени добавлена кнопка “Сейчас” и список распространенных значений.

В нижней части страницы мы видим несколько кнопок:

- Save – сохранить изменения и вернуться на страницу списка объектов;
- Save and continue editing – сохранить изменения и снова загрузить страницу редактирования текущего объекта;
- Save and add another – Сохранить изменения и перейти на страницу создания нового объекта;
- Delete – Показывает страницу подтверждения удаления.

Если значение “Date published” не совпадает с временем создания объекта в Части 1 учебника, возможно, вы неверно определили настройку TIME_ZONE. Измените ее и перезагрузите страницу.

Измените “Date published”, нажав “Today” и “Now”. Затем нажмите “Save and continue editing.” Теперь нажмите “History” в правом верхнем углу страницы. Вы увидите все изменения объекта, сделанные через интерфейс администратора, время изменений и пользователя, который их сделал:

Если вы освоили интерфейс администратора, переходите к третьей главе: *«Создаем свое первое приложение с Django, часть 3»*.