**Создаем проект**

Если вы правильно установили Django, то после запуска django-admin --version вы увидите текущую версию фреймворка. Теперь создадим проект. Это можно сделать следующим образом: django-admin startproject example\_project.

Как только создание проекта будет завершено, взглянем на директорию нашего проекта:

* example\_project **/\_\_init\_\_.py**— пустой файл, который говорит Python, что данная директория должна восприниматься в качестве пакета.
* example\_project **/settings.py**содержит конфигурацию нашего проекта.
* example\_project **/urls.py**— здесь объявляются URL.
* example\_project **/wsgi.py** — с помощью него приложение может работать с веб-сервером по протоколу WSGI.
* **manage.py**позволяет взаимодействовать с проектом.

Теперь пришло время запустить наше приложение. Для этого в командной строке нужно написать python manage.py runserver. После этого в адресной строке браузера нужно написать: <http://127.0.0.1:8000/>.

**Создаем приложение**

Определим различие между проектом и приложением. Приложение — это программа, которая что-то делает, а проект — это группа приложений.

Итак, приступим к созданию приложения. Это делается следующим образом: python manage.py startapp application.  
Как только приложение создано, давайте напишем простой вид, по правилам Django все виды должны храниться в файле views.py.

*application/views.py*

from django.http import HttpResponse

def index(request):

return HttpResponse("Hello, World!")

Теперь, чтобы привязать наш вид к URL, создадим файл urls.py.

*application/urls.py*

from django.conf.urls import url

from . import views

app\_name = 'application'

urlpatterns = [

url(r'^$', views.index, name='index'),

]

В urls.py мы должны написать следующее:

*example\_project/urls.py*

from django.conf.urls import include, url

from django.contrib import admin

urlpatterns = [

url(r'^application/', include('application.urls'), name=”application”),

url(r'^admin/', admin.site.urls),

]

Теперь, если мы запустим наше приложение http://127.0.0.1:8000/application/, мы увидим «Hello, World!».

**Установка базы данных**

По умолчанию в Django используется SQLite.

Теперь откроем example\_project/settings.py и взглянем на переменную INSTALLED\_APPS, она хранит все приложения, которые активны в текущем проекте. По умолчанию она содержит:

* **django.contrib.admin**— админка, скоро мы ей воспользуемся.
* **django.contrib.auth** — система аутентификации.
* **django.contrib.contenttypes** — фреймворк для content types.
* **django.contrib.sessions** — сессионный фреймворк.
* **django.contrib.messages**— фреймворк для отправки сообщений.
* **django.contrib.staticfiles** — фреймворк для работы со статичными файлами.

Некоторые из этих приложений используют базы данных, но они еще не установлены, поэтому мы и видели «You have unapplied migrations; your app may not work properly until they are applied.». Поправить это можно следующим образом: python manage.py migrate. Вы должны увидеть следующее:

Operations to perform:

Apply all migrations: admin, sessions, auth, contenttypes

Running migrations:

Rendering model states... DONE

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying auth.0007\_alter\_validators\_add\_error\_messages... OK

Applying sessions.0001\_initial... OK

Теперь создадим нашу модель. Для начала создадим Riddle и Option. В Riddle будет содержаться загадка, в Option — один из возможных ответов на нее.

*application/models.py*

from django.db import models

class Riddle(models.Model):

riddle\_text = models.CharField(max\_length=255)

pub\_date = models.DateTimeField('date published')

def \_\_str\_\_(self): return self.riddle\_text

class Option(models.Model):

riddle = models.ForeignKey(Riddle, on\_delete=models.CASCADE)

text = models.CharField(max\_length=255)

correct = models.BooleanField(default=False)

def \_\_str\_\_(self):

if self.correct:

value = "Correct"

else:

value = "Wrong"

return "{} - ({}) [{}]".format(self.text,value,self.riddle.riddle\_text)

Данная модель обеспечивает Django информацией, необходимой для создания схемы базы данных и database-access API для доступа к объектам. Теперь нам нужно привязать наше приложение к нашему проекту, делается это следующим образом:

*example\_project/settings.py*

INSTALLED\_APPS = [

'application.apps.ApplicationConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

После этого нужно сделать миграцию: python manage.py makemigrations application. Вы должны увидеть следующее:

Migrations for 'application':

0001\_initial.py:

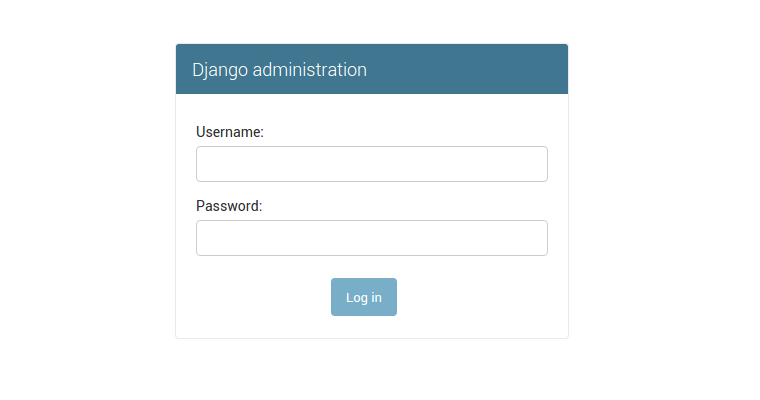
- Create model Option

- Create model Riddle

- Add field riddle to option

Так мы говорим Django, что в моделях были сделаны некоторые изменения, и их нужно сохранить в качестве миграции.

Теперь мы можем начать пользоваться панелью администратора. Но для этого нам нужен пользователь. Создать его можно следующим образом: python manage.py createsuperuser. Username: Alina, Password: VKUSNIYTORTIK97531. После этого запускаем сервер, если он не запущен, и переходим на http://127.0.0.1:8000/admin/. Вы увидите следующее:



Теперь дадим админу возможность изменять наши модели. Делается это так:

*application/admin.py*

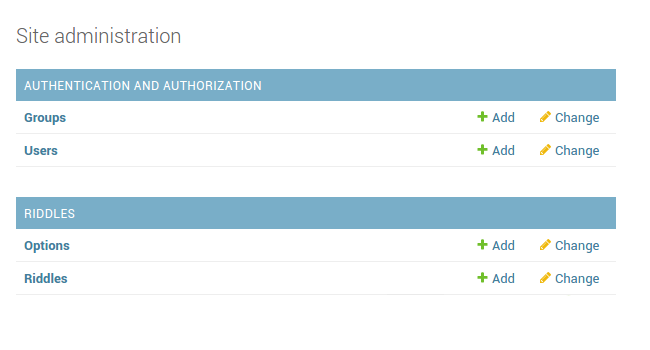
from django.contrib import admin

from .models import Option, Riddle

admin.site.register(Riddle)

admin.site.register(Option)

Вот что получится в итоге:



**Главная страница**

Что нам нужно для создания главной страницы?

* Templates: скелет нашей страницы.
* Views: функция на Python для отображения контента.

Начнем с шаблонов. Создадим папку templates внутри папки application, а в ней создадим index.html.

*application/templates/index.html*

<h1>Available Riddles</h1>

{% if message %}

<p><strong>{{ message }}</strong></p>

{% endif %}

{% if latest\_riddles %}

<ul>

{% for riddle in latest\_riddles %}

<li>

<a href="/application/{{ riddle.id }}/">

{{ riddle.riddle\_text }}

</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>No riddles are available right now.</p>

{% endif %}

Теперь создадим макет для ответов:

*application/templates/answer.html*

<h1>{{ riddle.riddle\_text }}</h1>

{% if error\_message %}

<p>

<strong>{{ error\_message }}</strong>

</p>

{% endif %}

<form action="{% url 'application: answer' riddle.id %}" method="post">

{% csrf\_token %}

{% for option in riddle.option\_set.all %}

<input type="radio" name="option" id="option{{ forloop.counter }}" value="{{ option.id }}" />

<label for="option{{ forloop.counter }}">{{ option.text }}</label><br>

{% endfor %}

<input type="submit" value="Answer" />

</form>

Здесь мы используем csrf\_token, он нужен для защиты от [межсайтовой подделки запроса](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D1%81%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%B4%D0%B5%D0%BB%D0%BA%D0%B0_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0), каждая внутренняя форма должна его использовать. Теперь напишем виды для рендеринга наших шаблонов:

*application/views.py*

from django.http.response import HttpResponse

from django.shortcuts import get\_object\_or\_404, render

from .models import Riddle, Option

def index(request):

return render(request, "index.html", {"latest\_riddles": Riddle.objects.order\_by('-pub\_date')[:5]})

def detail(request, riddle\_id):

return render(request, "answer.html", {"riddle": get\_object\_or\_404(Riddle, pk=riddle\_id)})

def answer(request, riddle\_id):

riddle = get\_object\_or\_404(Riddle, pk=riddle\_id)

try:

option = riddle.option\_set.get(pk=request.POST['option'])

except (KeyError, Option.DoesNotExist):

return render(request, 'answer.html', {'riddle': riddle, 'error\_message': 'Option does not exist'})

else:

if option.correct:

return render(request, "index.html", {"latest\_riddles": Riddle.objects.order\_by('-pub\_date')[:5], "message": "Nice! Choose another one!"})

else:

return render(request, 'answer.html', {'riddle': riddle, 'error\_message': 'Wrong Answer!'})

Давайте пройдемся по каждой функции отдельно:

* **index:**Index использует функцию render. На вход она получает HttpRequest, местонахождение шаблона и его содержимое, а возвращает HttpResponse с окончательным html.
* **detail:**Detail делает практически то же самое, но только функция get\_object\_or\_404 возвращает HttpResponse404, если нужный объект не был найден.
* **answer:** Answer ищет предоставленную загадку (и возвращает 404, если она не найдена) и проверяет правильность ответа.

Теперь добавим наши функции в urls.py:

*application/urls.py*

from django.conf.urls import url

from . import views

app\_name = 'application'

urlpatterns = [

url(r'^$', views.index, name='index'),

url(r'^(?P< riddle\_id>[0-9]+)/$', views.detail, name='detail'),

url(r'^(?P< riddle\_id>[0-9]+)/answer/$', views.answer, name='answer')

]

### Добавим немного стилей

Для начала создадим директорию static, а в ней создадим файл main.css.

riddles/static/main.css

body{

margin:40px auto;

max-width:650px;

line-height:1.6;

font-size:18px;

color:#444;

padding:0 10px;

}

h1,h2,h3{

line-height:1.2;

text-align: center;

}

a {

color: blue;

}

form {

margin: 0 auto;

padding: 1em;

border: 1px solid #CCC;

border-radius: 1em;

}

form div + div {

margin-top: 1em;

}

label {

display: inline-block;

text-align: center;

width: 40%;

}

input {

font: 1em sans-serif;

-moz-box-sizing: border-box;

box-sizing: border-box;

border: 1px solid #999;

width: 50%;

}

input:focus {

border-color: #000;

}

p, div.button {

text-align: center;

}

p.error-message {

color: lightcoral;

}

Немного изменим наши шаблоны:

riddles/templates/index.html

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'main.css' %}" />

<h1>Available Riddles</h1>

{% if message %}

<p><strong>{{ message }}</strong></p>

{% endif %}

{% if latest\_riddles %}

<ul>

{% for riddle in latest\_riddles %}

<li>

<a href="/application/{{ riddle.id }}/">

{{ riddle.riddle\_text }}

</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>No riddles are available right now.</p>

{% endif %}

riddles/templates/answer.html

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'main.css' %}" />

<h1>{{ riddle.riddle\_text }}</h1>

{% if error\_message %}

<p>

<strong>{{ error\_message }}</strong>

</p>

{% endif %}

<form action="{% url 'application:answer' riddle.id %}" method="post">

{% csrf\_token %}

{% for option in riddle.option\_set.all %}

<input type="radio" name="option" id="option{{ forloop.counter }}" value="{{ option.id }}" />

<label for="option{{ forloop.counter }}">{{ option.text }}</label><br>

{% endfor %}

<input type="submit" value="Answer" />

</form>

Первая строка загружает статические файлы, потом мы используем {% static '#' %}, где # — путь к вашему файлу. Аналогичная процедура проводится и для JavaScript.