

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



django

HOSSEIN FORGHANI

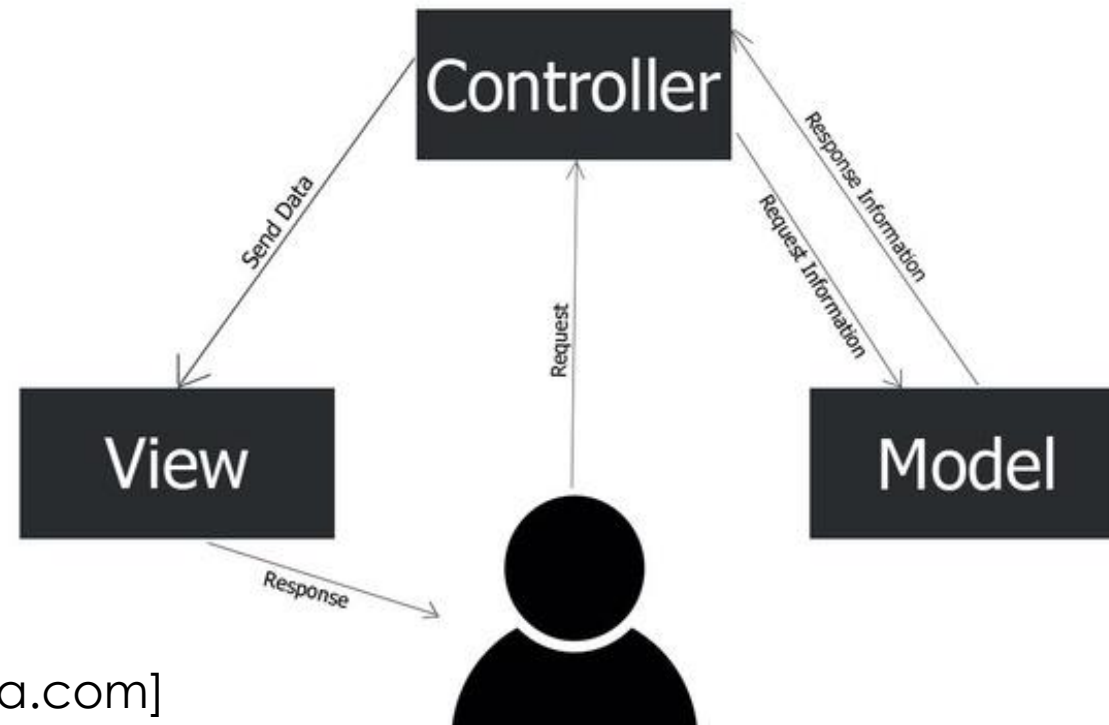
MAKTAB SHARIF

Contents

- ▶ MVC / MVT architecture
- ▶ Setup a Django project
- ▶ Add an app
- ▶ Connect to DB
- ▶ Introduction to Django ORM
- ▶ Introduction to Django admin

MVC

Model-View-Controller

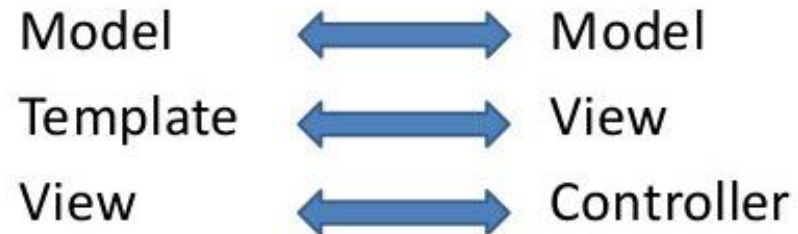


[quora.com]

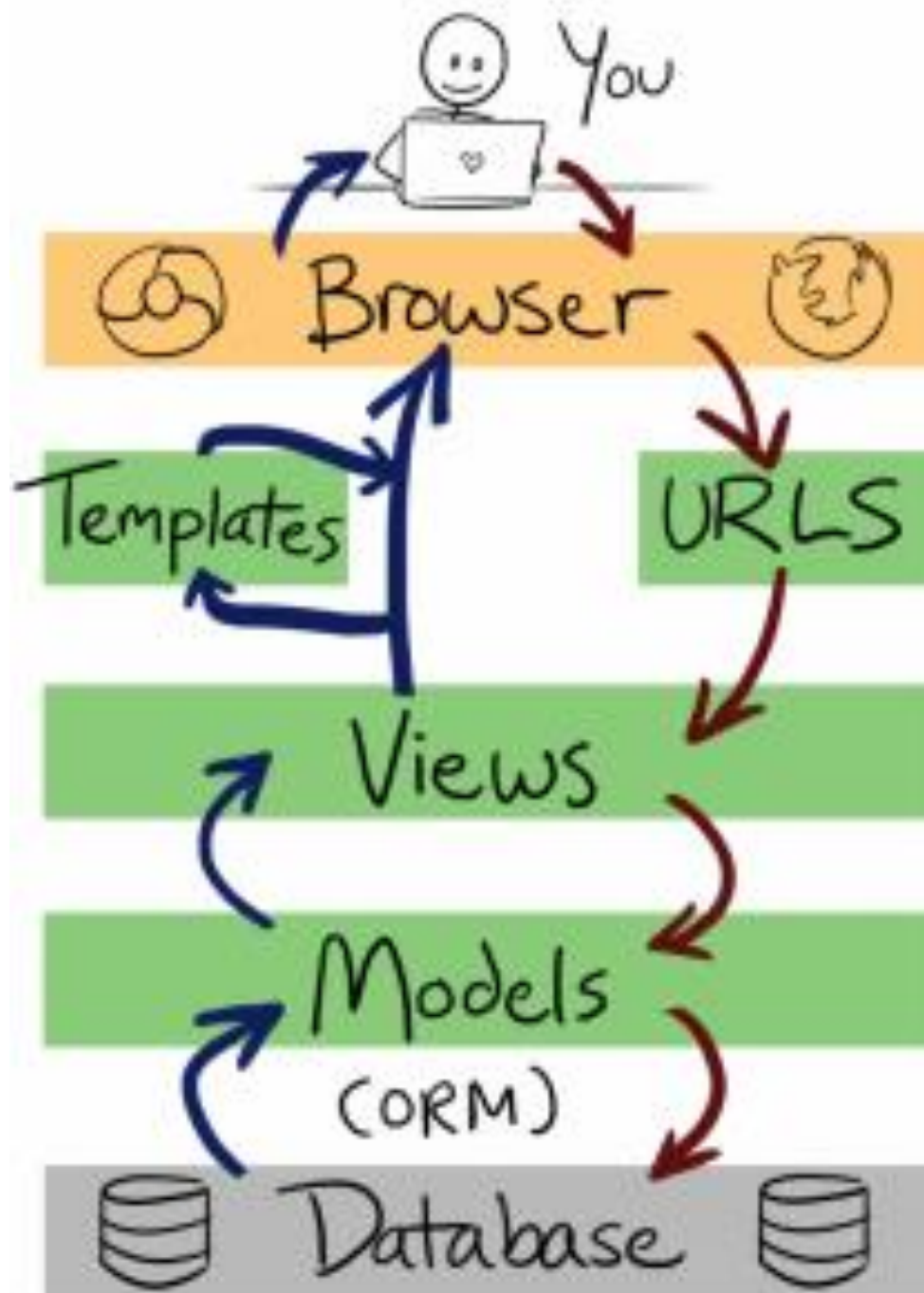
MVT

Is it MVC or MTV??

- In Django it is called MTV rather than MVC.



Models	Describes your data
Views	Controls what users sees
Templates	How user sees it
Controller	URL dispatcher



What will we do?

- ▶ We'll walk you through the creation of a basic poll application
- ▶ Consist of two parts:
 - ▶ A public site that lets people view polls and vote in them
 - ▶ An admin site that lets you add, change, and delete polls

Python & Django Version

- ▶ This tutorial is written for **Django 3.1**
- ▶ **Django 3.1** supports **Python 3.6** and later
- ▶ Check version of Django:

```
$ python -m django --version
```

- ▶ If not installed:

```
$ pip install django
```


Creating a project

- ▶ `cd` into a directory where you'd like to store your code, then:

```
$ django-admin startproject mysite
```

- ▶ This will create a `mysite` directory

container for your project. Its name doesn't matter

`mysite/`

`manage.py`

A command-line utility that lets you interact with this Django project

`mysite/`

Actual Python package for your project

`__init__.py`

Just empty!

`settings.py`

Settings for this Django project

`urls.py`

The URL declarations

`asgi.py`

`wsgi.py`

An entry-point for ASGI & WSGI compatible web servers to serve your project

Development Server

- ▶ Change into the outer mysite directory and run:

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

- ▶ You'll see:

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work
properly until they are applied.
Run 'python manage.py migrate' to apply them.

December 17, 2020 - 15:50:53
Django version 3.1, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Development Server – cont.

- ▶ A lightweight Web server written purely in Python
- ▶ To develop things rapidly
- ▶ Do not use this server in production environment

Visit <http://127.0.0.1:8000/>

django

[View release notes for Django 3.1](#)



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



Django Documentation

Topics, references, & how-to's



Tutorial: A Polling App

Get started with Django



Django Community

Connect, get help, or contribute

Changing the port

- ▶ By default, the runserver command starts on the internal IP at port 8000
- ▶ to change the server's port:

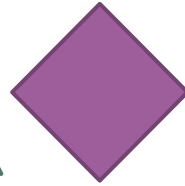
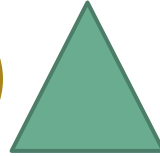
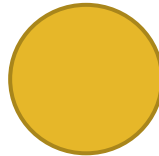
```
$ python manage.py runserver 8080
```

- ▶ to change the server's IP to listen on all available public IPs:

```
$ python manage.py runserver 0:8000
```

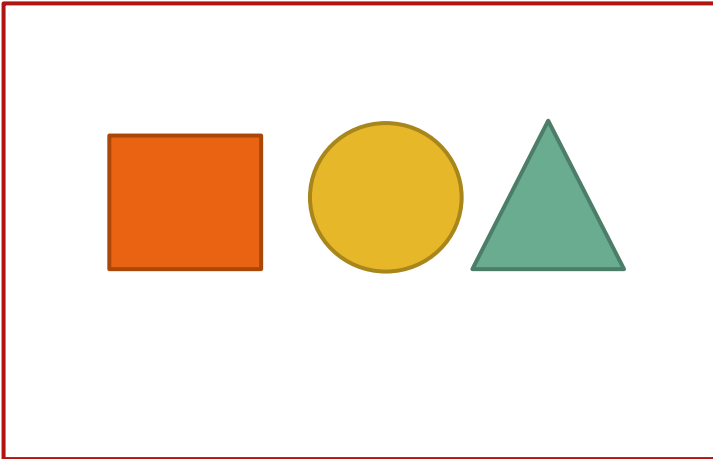
Projects vs Apps

Apps:

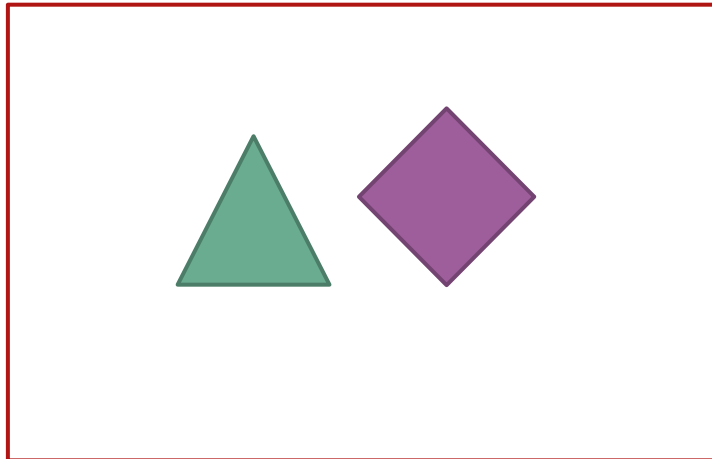


a web application
that does something

Projects:



Project 1



Project 2

a collection of
configuration and apps
for a particular website.

Creating the Polls app

- ▶ A utility that automatically generates the basic directory structure of an app:

```
$ python manage.py startapp polls
```



```
polls/  
    __init__.py  
    admin.py  
    apps.py  
    migrations/  
        __init__.py  
    models.py  
    tests.py  
    views.py
```


Write your first view

- ▶ A simple view that shows a “Hello, world ...” page!

polls/views.py



```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the polls  
index.")
```

Map view to URL

- ▶ To map your view to a URL, we need a URLconf
- ▶ Create a file called urls.py

polls/urls.py



```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Including in root URL conf

- ▶ Next step: to point the root URLconf at the polls.urls module

mysite/urls.py

```
from django.contrib import admin
from django.urls import include, path
```

```
urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

URL prefix

include() is not needed
only for admin.site.urls

path() function

```
path(route, view [, kwargs, name])
```

- ▶ **route**: a string containing a URL pattern
- ▶ **view**: view function to call
- ▶ **kwargs**: arbitrary keyword arguments passed to the target view
- ▶ **name**: a name for URL to refer it from elsewhere

Database setup

- ▶ In `mysite/settings.py`
- ▶ If you are not using SQLite:
 - ▶ additional settings such as `USER`, `PASSWORD`, and `HOST` must be added
 - ▶ You **must have created** the empty database

DB engine

For Postgres: `'django.db.backends.postgresql'`

The name of your database
For SQLite: DB file name

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

Other Settings

- ▶ set `TIME_ZONE` and `LANGUAGE_CODE` to your own ones
- ▶ `SECRET_KEY` is used for sessions. It is randomly generated once.
- ▶ `DEBUG = True` is set for development mode

```
# LANGUAGE_CODE = 'en-us'  
LANGUAGE_CODE = 'fa-ir'
```

```
# TIME_ZONE = 'UTC'  
TIME_ZONE = 'Asia/Tehran'
```

Installed Apps

- ▶ Holds the names of all Django applications that are activated in this Django instance
- ▶ By default:
 - ▶ Add your apps to this list
 - ▶ You can comment out any of default apps

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Apply Initial Migrations

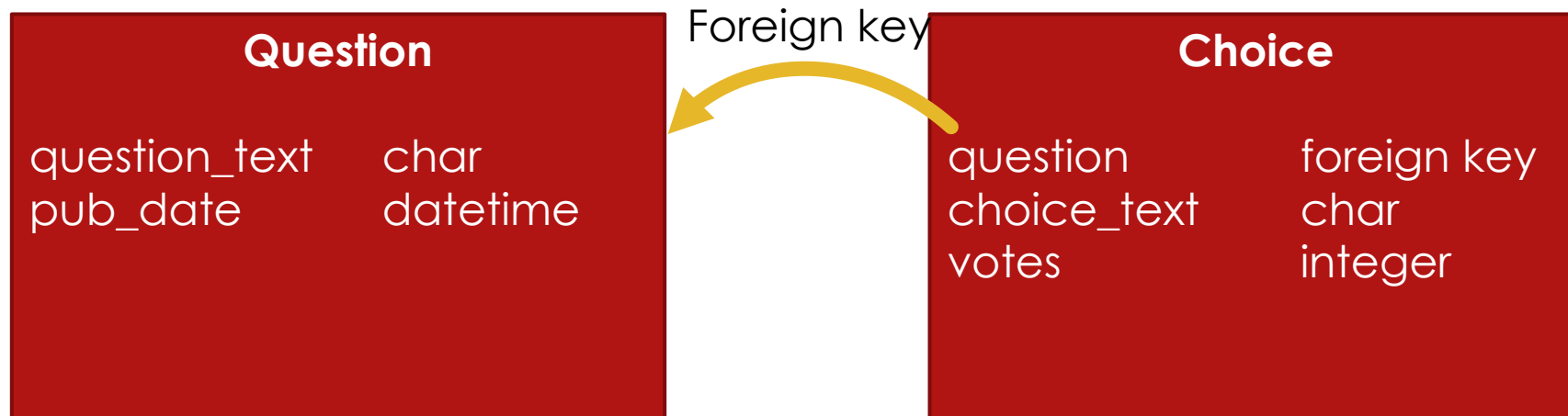
- ▶ Apps may use some DB **tables**
- ▶ The **migrate command** looks at the INSTALLED_APPS setting and creates any necessary database tables:

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

- ▶ This uses database **migrations** shipped with the app (in migrations folder)

Creating Models

- ▶ A model is the single, definitive source of truth about your data.



Creating Models – cont.

- ▶ Each model is represented by a class that subclasses `django.db.models.Model`
- ▶ Put your models in `polls/models.py`

Machine-readable
column name

One-to-many relationship
Django supports also
many-to-many and one-to-one

```
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)
```

Human-readable name


Activating Models

- ▶ Django apps are **pluggable**
- ▶ To include the app in our project, we need to add a reference to its configuration class in **INSTALLED_APPS**

PollsConfig class is in **polls/apps.py**

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',  
]
```



Activating Models – cont.

- ▶ Now Django knows to include the polls app. Then:

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

```
Migrations for 'polls':  
  polls/migrations/0001_initial.py  
    - Create model Question  
    - Create model Choice
```

- ▶ By running **makemigrations**, you're telling Django that you've made some changes to your models

Migrations














- ▶ Migrations are stored at polls/migrations folder
- ▶ You'll see now the file polls/migrations/0001_initial.py
- ▶ Don't worry, You're not expected to read them!
- ▶ To see what SQL that migration would run:

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

- ▶ To apply all migrations on DB:

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

After Migration

- ▼  Tables (12)
 - >  auth_group
 - >  auth_group_permissions
 - >  auth_permission
 - >  auth_user
 - >  auth_user_groups
 - >  auth_user_user_permissions
 - >  django_admin_log
 - >  django_content_type
 - >  django_migrations
 - >  django_session
 - >  polls_choice
 - >  polls_question

→ Migrations have been applied

→ Newly added

Migration Benefits

- ▶ Let you change your models over time
- ▶ No need to delete your database or tables and make new ones
- ▶ You'll commit migrations to your version control system
- ▶ 3 Step guide to making model changes:
 - ▶ Change your models
 - ▶ Run `python manage.py makemigrations` to create migrations
 - ▶ Run `python manage.py migrate` to apply

Django Shell

- ▶ To invoke the Python shell:

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

- ▶ Then you can explore the database API:

```
>>> from polls.models import Choice, Question # Import the models  
>>> Question.objects.all()  
<QuerySet []>
```


Inserting into DB

- ▶ Create a new Question:

```
>>> from django.utils import timezone  
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
```

- ▶ Save the object into the database. Now it has an ID:

```
>>> q.save()  
>>> q.id  
1  
>>> q.question_text  
"What's new?"
```

Changing Values

- ▶ Change values by changing the attributes, then calling `save()`:

```
>>> q.question_text = "What's up?"  
>>> q.save()  
>>> Question.objects.all()
```

```
<QuerySet [<Question: Question object (1)>]>
```

→ isn't a helpful
representation

Representation for Model

- ▶ adding a `__str__()` method to your models:
- ▶ objects' representations are used throughout Django's automatically-generated admin

polls/models.py

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Custom Method

- ▶ For example you can add a method to check the question is published within 1 day ago

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)
```

Database Lookup

- ▶ Django provides a rich database lookup API that's entirely driven by keyword arguments:

```
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>]>

>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>
```

Getting by ID

- ▶ 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. The following is identical to `Question.objects.get(id=1)`

```
>>> q = Question.objects.get(pk=1)
>>> q
<Question: What's up?>
```

choice_set

- ▶ Django creates a set to hold the "other side" of a ForeignKey relation:

```
>>> q.choice_set.all()  
<QuerySet []>
```

- ▶ 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:

```
>>> 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_set – cont.

```
>>> c.question  
<Question: What's up?>
```

- ▶ And vice versa: Question objects get access to Choice objects:

```
>>> q.choice_set.all()  
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>  
>>> q.choice_set.count()  
3
```


Magical Double Underscores!

- ▶ 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
- ▶ Find all Choices for any question whose pub_date is in this year:

```
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>
```

- ▶ Let's delete one of the choices:

```
>>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()
```

Django Admin

- ▶ Projects usually have an **admin site** including some CRUD pages
- ▶ Creating **CRUD pages** is tedious work that doesn't require much creativity
- ▶ Django entirely **automates** that
- ▶ The admin isn't intended to be used by site visitors. It's for **site managers**

Create an Admin User

- ▶ Create a user who can login to the admin site:

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

```
Username: admin
```

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

```
Password: ****
```

```
Password (again): ****
```

```
Superuser created successfully.
```

Login Page

- ▶ After running the server, go to <http://127.0.0.1:8000/admin/>
- ▶ Since translation is turned on by default and you set LANGUAGE_CODE, the login screen will be displayed in the given language



The screenshot shows the Django Admin login interface in Persian. At the top, a dark blue header contains the text "مدیریت Django". Below this, the form is centered on a light gray background. It features two input fields: the first is labeled "نام کاربری:" (Username) and the second is labeled "گذرواژه:" (Password). Below the password field is a blue button with the text "ورود" (Login).

مدیریت وبگاه

بررسی اصالت و اجازه‌ها	
کاربرها	<div> <div>اضافه کردن</div> <div>تغییر</div> </div>
گروه‌ها	<div> <div>اضافه کردن</div> <div>تغییر</div> </div>

فعالیت‌های اخیر

فعالیت‌های من

چیزی در دسترس نیست

Add Poll App to Admin

- ▶ Open the `polls/admin.py` file, and edit it to look like this:

```
polls/admin.py
```

```
from django.contrib import admin
```

```
from .models import Question
```

```
admin.site.register(Question)
```

مدیریت Django

مدیریت وبگاه

<p>فعالیت‌های اخیر</p>	<div>POLLS</div> <div>Questions</div> <div>  تغییر  اضافه کردن </div>
<p>فعالیت‌های من</p> <p>چیزی در دسترس نیست</p>	<div>بررسی اصالت و اجازه‌ها</div> <div>کاربرها</div> <div>  تغییر  اضافه کردن </div> <div>گروه‌ها</div> <div>  تغییر  اضافه کردن </div>

References

- ▶ <https://docs.djangoproject.com/en/3.1>

Any Question?