Sistemas de Negocios

Clase 2A: Beginning Django y programacion de sistemas web para negocios

September 21, 2016

David Olivieri

Uvigo, E.S. Informatica

Django

- MVC pattern into web development
- Python:
  - Clean and elegant syntax.
  - Large standard library of modules that covers a wide range of tasks.
  - Extensive documentation.
  - Mature runtime environment.
  - Support for standard and proven technologies such as Linux and Apache.

Django

- Tight integration between components
- Object-relational Mapper (ORM)
- Clean URL design
- Automatic administration interface
- Advanced development environment
- Multi-language support
- Template engine:
  - Text filtering engine
  - Form generation and validation
  - Extensible authentification
  - Caching system

Getting Started

- We will learn the following topics in this chapter:
  - Installing Python.
  - Installing Django.
  - Installing a database system.
  - Creating your first project.
  - Setting up the database.
  - Launching the development server.

These class notes
follow a great Tutorial series
that can be found on
Youtube:

https://www.youtube.com/user/CodingEntrepreneurs

Python on linux

For APT-based Linux distributions
(such as Debian and Ubuntu),
open a terminal and type:

$ sudo apt-get update
$ sudo apt-get install python

$ sudo apt-get install python-django

You can test your installation by running
this command:

$ django-admin.py --version

Using virtualenv

- A great way to isolate code.

```
sudo easy_install pip
```

“pip” is the python package installer

```
sudo pip install virtualenv
```

The virtualenv is a great way to isolate code inside its own environment.

Using virtualenv

```
virtualenv  ecommsite

cd ecommsite
source bin/activate
```

Activates the virtual environment

```
pip freeze
```

This checks to see what
Is installed when starting.

```
pip install django==1.6.1
```

Install django within the virtual environment that we created.

A session for creating a virtualenv

```
david@treg: ~/Djprog/class
New python executable in class/bin/python
Installing setuptools, pip...done.
david@treg:~/Djprog$ ls
class
david@treg:~/Djprog$ cd class/
david@treg:~/Djprog/class$ ls
bin  include  lib  local
david@treg:~/Djprog/class$ source bin/activate
(class)david@treg:~/Djprog/class$ pip install django==1.6.1
Downloading/unpacking django==1.6.1
  Downloading Django-1.6.1-py2.py3-none-any.whl (6.7MB): 6.7MB downloaded
Installing collected packages: django
Successfully installed django
Cleaning up...
(class)david@treg:~/Djprog/class$ pip freeze
Django==1.6.1
argparse==1.2.1
wsgiref==0.1.2
(class)david@treg:~/Djprog/class$
```

Creating first project

- Objectives:
  - Creating a new project.
  - Creating and managing the project's database.
  - Validating the current project and testing for errors.
  - Starting the development web server.

Creating the project “ecomm”

```
david@treg: ~/Djprog/class/ecomm
Django==1.6.1
argparse==1.2.1
wsgiref==0.1.2
(class)david@treg:~/Djprog/class$ clear

(class)david@treg:~/Djprog/class$ ls
bin  include  lib  local
(class)david@treg:~/Djprog/class$ django-admin.py startproject ecomm
(class)david@treg:~/Djprog/class$ ls
bin  ecomm  include  lib  local
(class)david@treg:~/Djprog/class$ cd ecomm
(class)david@treg:~/Djprog/class/ecomm$ ls
ecomm  manage.py
(class)david@treg:~/Djprog/class/ecomm$
```

Creating an Empty Project

$ django-admin.py startproject ecomm

This command will make a folder named **ecomm** in the current directory, and create the initial directory structure inside it. Let's see what kinds of files are created:

```
ls ecomm/
        __init__.py
        manage.py
        settings.py
        urls.py
```

manage.py This is another utility script used to manage your project. You can think of it as your project's version of django-admin.py.

settings.py This is the main configuration file for your Django project.

url.py This is another configuration file. You can think of it as a mapping between URLs and Python functions that handle them

Database options in “settings.py”

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

syncdb

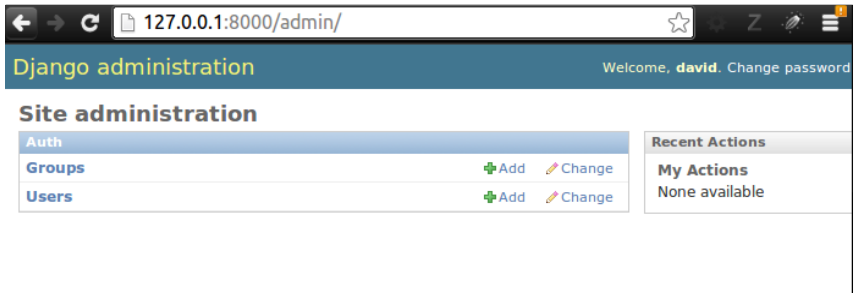
- We setup the database by using the manage.py and synching the database:

```
$python  manage.py  syncdb
```

```
david@treg: ~/Djprog/class/ecomm
(class)david@treg:~/Djprog/class/ecomm$ python manage.py syncdb
Creating tables ...
Creating table django_admin_log
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session

You just installed Django's auth system, which means you don't have any superuse
rs defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'david'):
Email address: d@s.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
(class)david@treg:~/Djprog/class/ecomm$
```

“url.py”

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'ecomm.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

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



Launching the Development Server

To start the server, run the following command:
$ python manage.py runserver

Next, open your browser, and navigate to **http://localhost:8000/.** You should see a welcome message as in the image below:

```
david@treg: ~/Djprog/class/ecomm
(class)david@treg:~/Djprog/class/ecomm$ python manage.py runserver
Validating models...

0 errors found
September 29, 2014 - 08:14:32
Django version 1.6.1, using settings 'ecomm.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Summary of what we have done

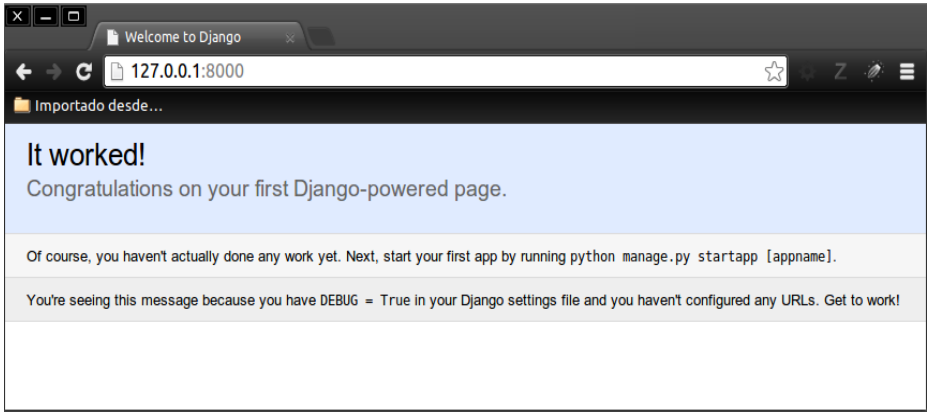
- To start a new Django project, issue the following command:

- **$ django-admin.py startproject <project-name>**
  - To create database tables, issue the following command:
- **$ python manage.py syncdb**
  - To start the development server, issue the following command:
- **$ python manage.py runserver**
  - Django project settings are stored in settings.py. This file is a regular Python source file that can be edited using any source code editor. To change a variable, simply assign the desired value to it.

Runnning the server

python manage.py runserver

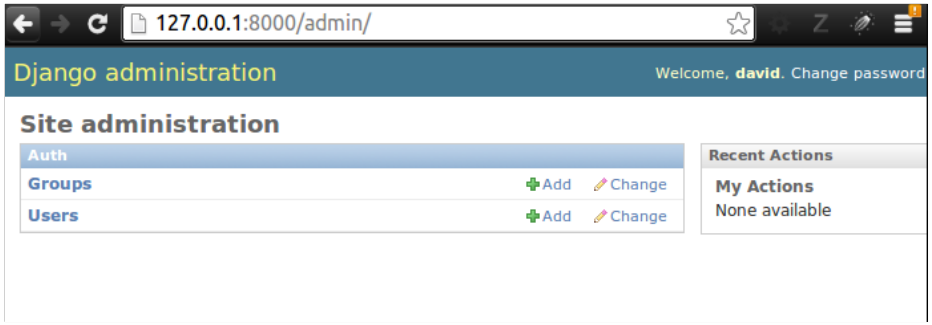
This runs the server and we
Can pull up a browser and see that
It works:



http://127.0.0.1:8000/

We can see an administration page:

http://127.0.0.1:8000/admin



Building First App: “signup” app

- This is just a way of adding a name and info;
- Sign up for a webpage.

About MVC and terminology

•Django is an MVC framework. However, the controller is called the "view", and the view is called the "template".

•The view in Django is the component which retrieves and manipulates data, whereas the template is the component that presents data to the user.

- For this reason, Django is sometimes called an MTV framework (where MTV stands for model template view).

•This different terminology neither changes the fact that Django is an MVC framework, nor affects how applications are developed.
•But keep the terminology in mind to avoid possible confusion if you have worked with other MVC frameworks in the past.

Creating the app

- A **view** in Django terminology is a regular Python function that responds to a page request by generating the corresponding page.

- To write our first Django view for the main page, we first need to create a Django **application** inside our project.
- You can think of an application as a container for views and data models.

Creating An application

```
$ python manage.py startapp bookmarks
```

Django will create a folder named bookmarks inside the project folder with these three files:

The syntax of application creation is very similar to that of project creation. We used startapp as the first parameter to python manage.py, and provided bookmarks as the name of our application

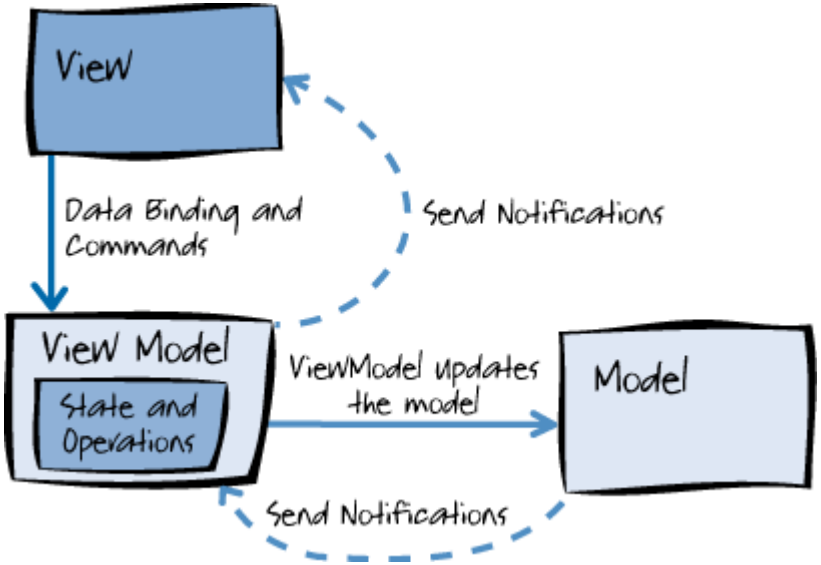
__init__.py:	This file tells Python that bookmarks is a Python package.
views.py:	This file will contain our views.
models.py:	This file will contain our data models.

Overview of the files

__init__.py:	This file tells Python that bookmarks is a Python package.
views.py:	This file will contain our views.
models.py:	This file will contain our data models.
admin.py:	This takes the models and makes them usable

Models are a way to populate the database tables.

Views is used to describe how the website will appear.



Making the app

```
david@treg: ~/Djprog/class/ecomm

(class)david@treg:~/Djprog/class/ecomm$ python manage.py startapp signups
(class)david@treg:~/Djprog/class/ecomm$ ls
db.sqlite3  ecomm  manage.py  signups
(class)david@treg:~/Djprog/class/ecomm$
```

The settings files are here.

The app now is in this directory.

```
(class)david@treg:~/Djprog/class/ecomm/signups$ ls -1
admin.py
__init__.py
models.py
tests.py
views.py
```

The “models.py” file

- This is used to define what looks like sql files.
- Django will take care of forming the tables for the appropriate database backend.

The empty “models.py” file

```
from django.db import models

# Create your models here.
```

You will add your model class here

create the model: signups

models.py

```
from django.db import models
from django.utils.encoding import smart_unicode

class SignUp(models.Model):
    first_name = models.CharField(max_length=120, null=True,blank=True)
    last_name= models.CharField(max_length=120, null=True, blank=True)
    email = models.EmailField()
    timestamp = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=True, auto_now=False)

    def __unicode__(self):
        return smart_unicode(self.email)
```

The **smart_unicode** helps with internationalization; foreign symbols

Now install in settings.py

- We need to look for INSTALLED_APPS and install it there

```
INSTALLED_APPS = (
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.sessions',
   'django.contrib.messages',
   'django.contrib.staticfiles',
  'signups',
)
```

Add this line

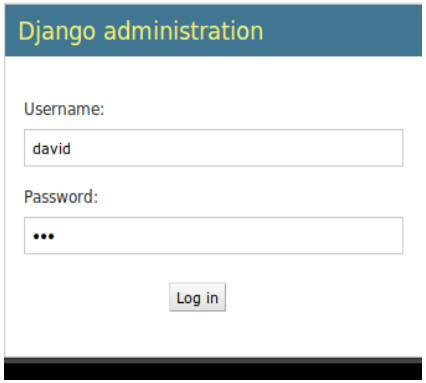
This registers the model in the settings.py file.

Now synch to database

```
$ python manage.py syncdb
```

This will create a table signups_signup

```
(class)david@treg:~/Djprog/class/ecomm$ python manage.py syncdb
Creating tables ...
Creating table signups_signup
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
(class)david@treg:~/Djprog/class/ecomm$
```



We need to import it into the admin app.

```
from django.contrib import admin

# Register your models here.
```

This is the empty “admin.py” file

Add it to the admin

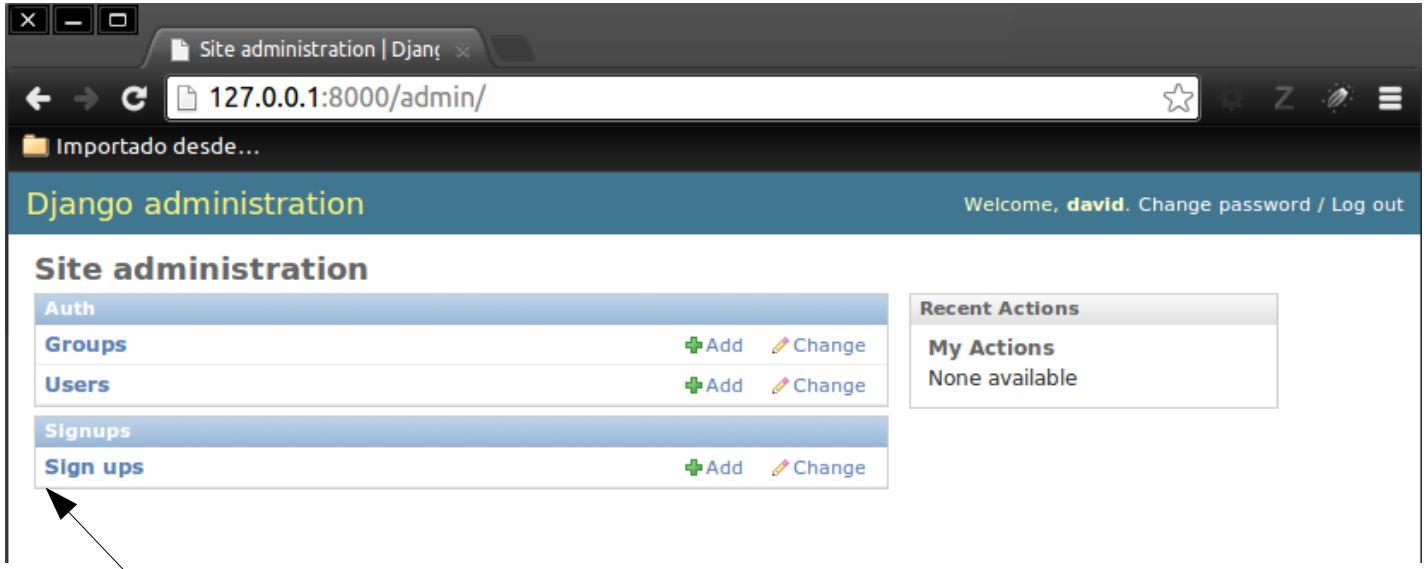
- Open admin.py

```
from django.contrib import admin

# Register your models here.
from .models import SignUp

class SignUpAdmin(admin.ModelAdmin):
    class Meta:
        model = SignUp

admin.site.register(SignUp, SignUpAdmin)
```

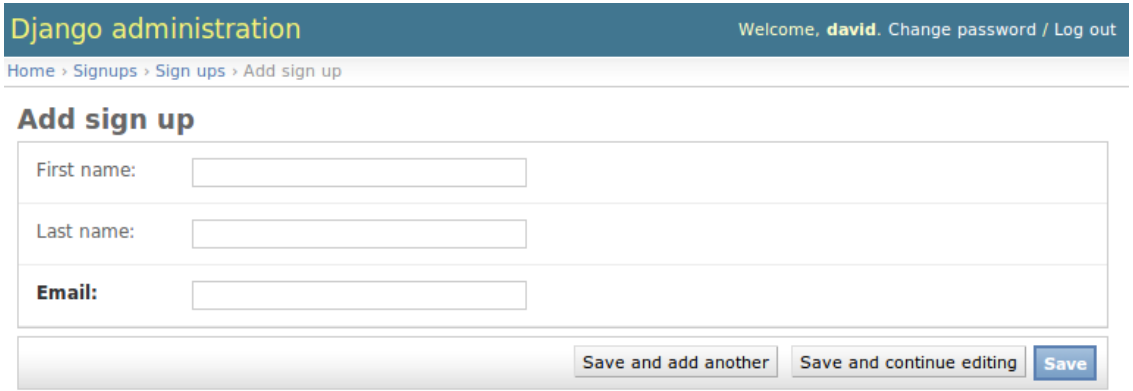
Register both the model and the SignUpAdmin classes

Now open admin

python manage.py runserver



Now our app appears in the admin



Now visualizing in webpage with views

- We want to connect the application to a webpage.
- How can we have a form that will display a form to add information?
- Admin is only for the webpage superuser.

The “views” with Templates

- Making the web application visible
- This will allow anyone to register the email
- We will setup “Templates”

First step: Specify where the “Template” files will be located.

Open the **settings.py** file. Add the following at the bottom.

```
STATIC_URL = ‘/static/’

# Template location
TEMPLATE_DIRS = (
   os.path.join(os.path.dirname(BASE_DIR), “static”,“templates”),
)
```

Must create These two directories

let-’s look at that last sentence

os.path.join(os.path.dirname(BASE_DIR), “static”, “templates”)

Joins the path with the BASE directory then it will look for
The directory “static” and then “templates”

The BASE_DIR was defined before.

BASE_DIR = os.path.dirname(os.path.dirname(__file__))

Create the directories

In the top level dir:

```
mkdir  static
cd static
mkdir templates
```

```
(class)david@treg:~/Djprog/class$ ls
bin  include  lib  local  src  static
```

"ecomm" was renamed "src"

Inside templates, create a new file called "signup.html"

```
<!DOCTYPE html>
  <html>
     <head>
     </head>

     <body>
          <h1>Join Now<h1>
     </body>
  </html>
```

Creating the html file. 1.

We need to render this file when someone goes to the url signup.html

We do this with url.py under main src

```
urlpatterns = patterns( ‘’,
    url(r’^$’,’signups.views.home’,name=’home’),
    url r’^admin/’,include(admin.site.urls)),
)
```

But, we don’t have Anything called home, So we have to open “views.py” and define ‘home’

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^$', 'signups.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

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

Writing the views.py

“views.py”

```
from django.shortcuts import render

# create your views here.
def  home(request):
      return   #something
```

Here we need to
Do a HTTPResponse

Full implementation with details:

```
from django.shortcuts import render, render_to_response, RequestContext

# create your views here.
def  home(request):
      return render_to_response(“signup.html”,
            locals(),
            context_instance=RequestContext(request))
```

Now we can test the page.

Now we need to add a form.

Adding a form

- Method 1: Just the old HTML way.

```
<!DOCTYPE html>
  <html>
      <head>
      </head>
      <body>
            <h1>Join Now<h1>
                <form method=‘POST’ action=‘’>
                     <input type=‘text’>
                     <input type=‘submit’>
                </form>
      </body>
  </html>
```

Method 2. the django way

- We want to create the form directly from our model
  – Create a new file under our app: forms.py

forms.py

Create this file
In the “signups”
App directory

```
from django import forms
from .models import SignUp

class SignUpForm(forms.ModelForm):
    class Meta:
        model= SignUp
```

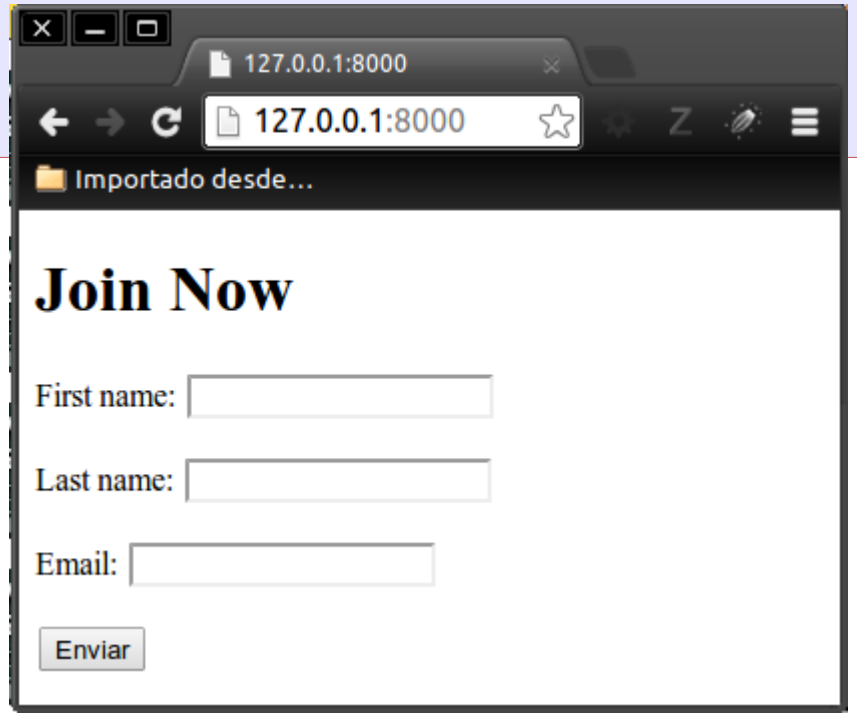
Modify **views.py**

```
from django.shortcuts import render, render_to_response, RequestContext
# create your views here.
from .forms import SignUpForm
def  home(request):
    form = SignUpForm()
    return render_to_response(“signup.html”,
        locals(),
        context_instance=RequestContext(request))
```

Modify the signup.html file

A token that helps against spam... if not we get a csrf error

```
<!DOCTYPE html>
  <html>
     <head>
     </head>
     <body>
          <h1>Join Now<h1>
              <form method='POST' action=''>  {% csrf_token %}
                {{  form.as_p  }}
                <input type='submit'>
              </form>
     </body>
  </html>
```



Run the server again:
python manage.py runserver

And refresh the page.

At the moment, we are not doing anything with the data... We need to program more...

Do something with data

In **views.py**, we had the following:

form = SignUpForm()

But, this doesn't do anything with the data.
We need to modify this.

form = SignUpForm(request.POST or None)

The method POST sends it to the server;
the action (action='') is within
The same page.

```
<form method='POST' action=''> {% csrf_token %}
    {{ form.as_p }}
   <input type='submit'>
```

The server then needs to request
that information. Thus, we need
To check if things are valid"

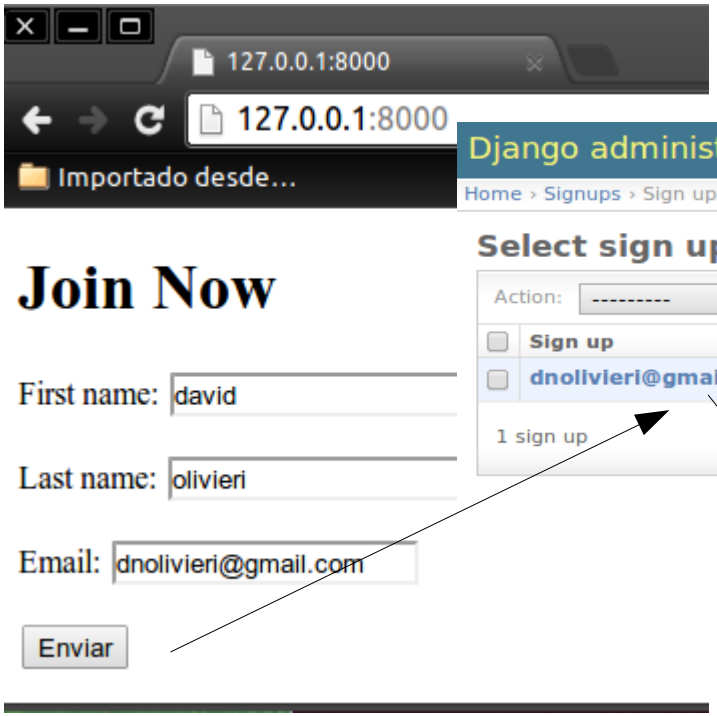
Do something with data

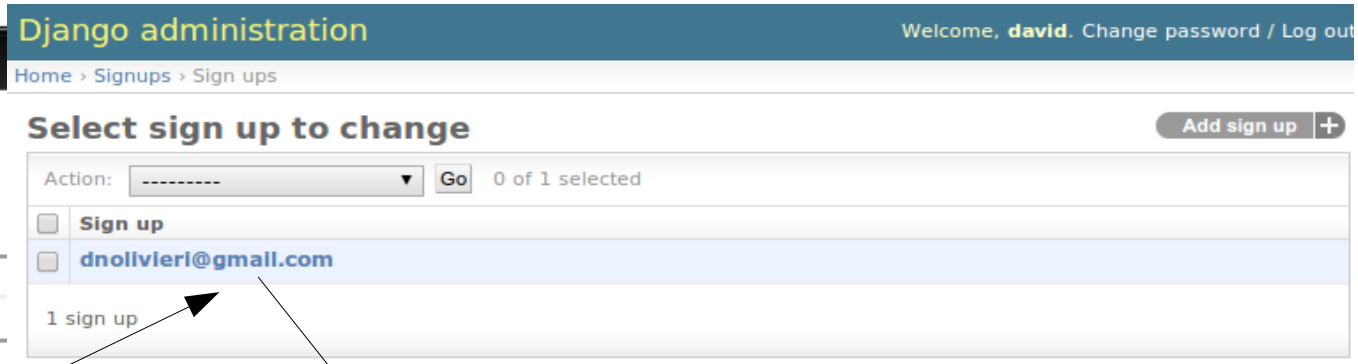
views.py

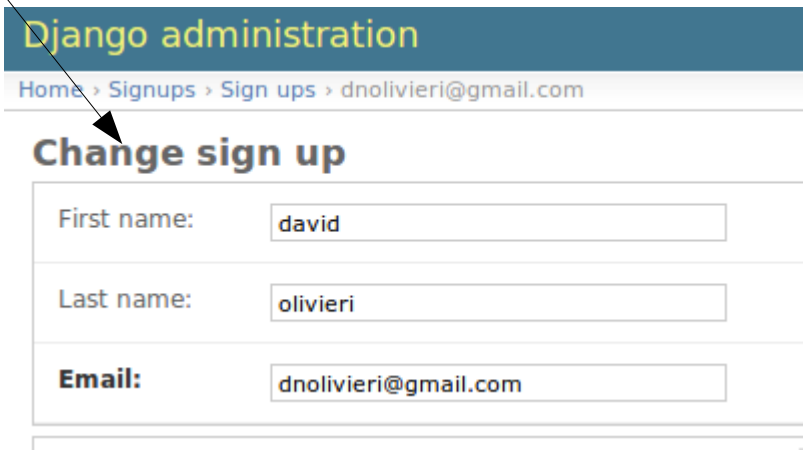
```
from django.shortcuts import render, render_to_response, RequestContext
# create your views here.
from .forms import SignUpForm
def  home(request):
      form = SignUpForm(request.POST or None )
      if form.is_valid():
            save_it = form.save(commit = False)
            save_it.save()
      return render_to_response("signup.html",
            locals(),
            context_instance=RequestContext(request))
```

Now refresh the server.

Try out the app







It worked... We saved
The data from the webpage!

Summary

- We have made a simple app that:
  - connects to a database
  - We can see data in admin app.
  - Connects to a web page through a template.
- Next time:
  - we will improve on templates;
  - Use bootstrap.