Simple Forms With Python And Django Tutorial

It's weekend again, and you want to make sure you won't go through next week without playing soccer with your friends in the local playing field. Every week it's the same thing. You send a mail to a twenty-something people mailing list, asking what time and day is best for everyone.

Half an hour later you have 30 emails on 7 different threads with 23 different playing time options. What a mess. If only you could just send a website link to everyone, and come back a day later to see the final results. What if you could set up such a survey in under an hour?

0. Preview

This post is going to give an overview of Python and the widely-used Django as web framework. It's my personal preference of website building blocks and I won't be discussing its pros and cons here. I'm going to try to be as focused as possible, which means I'll give as little customization options as possible so you won't get confused. Security and beauty of websites will also not addressed here for the same reasons. Feel free to ask for elaboration in the comments section, or using Django's extensive documentation.

1. Setup time

So, what do we need for the next hour? First of all, Python version 2.7.5 scripting language installed (32bit and 64bit version). Second and last, Django web framework. Download the framework here and extract it somewhere on your drive (lets say, `c:\code\Django-1.5.1\`). Now install Django as a python package by opening an **admin** cmd.exe and running

```
c:\>c:\python27\python.exe c:\code\Django-1.5.1\setup.py install
c:\>cd c:\code
c:\code\>c:\python27\scripts\djang-admin.py startproject soccer
```

note: You should add c:\python27\ to your system's PATH variable, to avoid typing the full path every time.

Perfect. We now have Django website *project*. Of course every site needs a database to hold.. well.. data. Go ahead and open `c:\code\soccer\soccer\settings.py` with your favorite editor (mine's Notepad++). You'll see a line saying "`DATABASES`", edit the configuration under it so it will look like this (additions in **bold**):

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'c:/code/soccer/soccer.sqlite',   # Or path to database file if using sqlite3.
```

```
        # The following settings are not used with sqlite3:
        'USER': '',
        'PASSWORD': '',
        'HOST': '',                 # Empty for localhost through domain
        'PORT': '',                 # Set to empty string for default.
    }
}
```

Note: Django paths are always using forward slashes ('/'), whether you're using Windows or Linux

Now set up your DB by running the following command. It will ask you for configuring a superuser for your website, you can just choose admin/admin user/pass combination for now, don't worry about it.

```
c:\code\soccer\>python manage.py syncdb
```

Last but not least, we need to create an *application* inside our *project*. Let go to console again and type:

```
c:\code\soccer\>python manage.py startapp survey
```

To let Django know we are going to use this app, open your `settings.py` again and edit:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'survey',
)
```

Note: Remember that python is an "offside rule" language, meaning spaces and indentation is critical. Take extra care when copying&pasting the code from the blog.

At this stage you can start your webserver:

```
c:\code\soccer\>python manage.py runserver 0.0.0.0:8000
```

This runs the webserver on port 8000, and allows access from any IP (after all, a survey for yourself is no survey!). Check that everything went fine by entering `127.0.0.1:8000` in your browser.

Congratulation, you have a website up and running! Now lets add some content to it.

2. Writing our survey system

Lets start with *models*, these are essentially items of information you will want to store in your DB and present to users. Open your `models.py` (in our *application*, which as you remember is in `c:\code\soccer\survey`) and insert the following to the file:

```
from django.db import models
class Vote(models.Model):
  choiceDay = models.CharField(max_length=100)
  choiceTime = models.IntegerField()
  voterName = models.CharField(max_length=100)
```

This bit will add a new table to our website's DB with 3 columns: choiceDay for the desired day for playing ("Sun", "Wed", etc.). choiceTime will be an Integer field because we only want to give 2 options for match time (08:00 and 20:00). voterName will be the name of the specific vote item so we can track no one's been cheating. Because this is a DB modification, you will now need to run the `syncdb` command again (see previous section if you forgot the syntax).

Another building block of Django is a *view*, which is a "type of webpage" inside you website. e.g. in this blog, a view is either a "Blog post", "Blog homepage" or "Sitemap". Views are created in the `views.py` file, next to your previous `models.py`. Lets add our first view now, this one will be in charge of simply displaying the survey options to the voters and asking them to choose. Later on we will create a view that shows the results of the vote. Open `views.py` and add:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render
from survey.models import Vote
def surveyDisplay(request):
  return render(request, 'survey/display.html')
```

This is a very simplistic view, and as you can guess, all it does it direct us to an html page we will now create. This doesn't yet use any of Django's power-ups, but just hang on. For the html pages we will use, create a folder inside "survey" named "templates", and under that another one called "survey" (that's just how Django works). Now create a file called "`display.html`" (It should reside at `c:\code\soccer\survey\templates\survey\display.html`).

```
<html> <head><title>Soccer time survey</title></head>
<body>
Hey, when would you like to play soccer?
<form action="process" method="post">
{% csrf_token %}
Time:
   <input type="radio" name="time" id="choice1" value="1">08:00</input>
<input type="radio" name="time" id="choice2" value="2">20:00</input> </br>
```

WBTutorial.com
Best Website Builder Reviews and Tutorials

```
Day:
<select name=”day” >
<option value=”Sun”>Sun</option>
<option value=”Mon”>Mon</option>
<option value=”Tue”>Tue</option>
<option value=”Wed”>Wed</option>
<option value=”Thu”>Thu</option>
<option value=”Fri”>Fri</option>
<option value=”Sat”>Sat</option>
</select> <br />
 Name:
<input type=”text” name=”name” /> <br />
   <input type=”submit” value=”Vote” />
</form>
</body> </html>
```

As you can see from the html, this is a regular and simple POST form (ignore the `csrf_token` for the moment, its a security cookie Django forces you to use in forms). We are simply asking the user to fill in three fields, and posting our results to a webpage called “`process`“. As you guessed by now, process would be a *view* in our website. Lets write that view now. Open `views.py` and add a new view to handle the survey processing:

```
def surveyProcess(request):
  Vote(choiceDay = request.POST['day'],
choiceTime = int(request.POST['time']),
voterName = request.POST['name']).save()
  return HttpResponseRedirect(“results”)
```

The first expression creates a new Vote object, and fill its data with the information we received from the form. The `save()` function inserts the object to our DB. The last line redirects the user’s webpage to the results page.
Yes. You are guessing correctly, now we will write our third and final view – the survey results.

```
def surveyResults(request):

  votes = Vote.objects.all()
  context = {‘votes’: votes}
  return render(request, ‘survey/results.html’, context)
```

What have we here this time? `Vote.objects.all()` actually pulls from the DB all of the votes objects into as a list. the next two lines should be interpreted together as “Load the `survey/results.html` webpage, giving it the votes list as a context for parsing information”. If you remember `display.html` we just wrote, you remember it was static, meaning it always displayed the same content. One of Django’s strong suits is passing along information (aka context) for the template html file. Lets see how `results.html` (in the same templates/survey directory as its friend `display.html`) should use this context:

```
<html> <head><title>Soccer time results</title></head> <body>
<ul>
{% for vote in votes %}
<li> {{ vote.voterName }} wants to play on {{ vote.choiceDay }} at {% if vote.choiceTime == 1
%} 08:00 {% else %} 20:00 {% endif %}</li>
{% endfor %}
</ul>
<a href=”survey”>Back to main page</a>
</body>  </html>
```

Have you ever seen anything more simple? The Django templates engine allows us to write actual flow-control code in our html templates. Here we create a list, and add a bullet for each entry in the context we were given (by using the field names we chose way back when we created `models.py`). We use the each list item's data either by testing it (using `{% if … %}` or simply printing using `{{vote.name}}`).

Note: There is nothing stopping you from prettifying your templates in any way you see fit. Neither me or Django will help you with that, as it is not in the current scope of discussion, but you can feel free to implement CSS and JavaScript extensions like bootstrap or knockout.

3. Putting it all together

So we have our three views, and two templates to go along with them (remember our second view, the processing one, doesn't display anything to the user so it doesn't need a template). The only thing missing is how to connect everything together. If you think back on what we created, we said a "view" is like a section of our website when thought of from our site of writing the code. From the outside, a "view" is associated with a URL of course. Let's open our websites urls.py (under `c:\code\soccer\soccer\`) and fill in:

```
from django.conf.urls import patterns, include, url
from survey import views
urlpatterns = patterns(”,
    url(r’^survey$’, ‘survey.views.surveyDisplay’),
  url(r’^process$’, ‘survey.views.surveyProcess’),
  url(r’^results$’, ‘survey.views.surveyResults’),
)
```

What have we here? Django loads this page when we run the server, and thus knows which view is to load with every URL the user typed. The first parameter to each url() is a *regular-expression* matching the URL typed, and the second parameter is which view to load. In our website we have the surveyDisplay view for the main page (which now you know will be accessed by <websiteURL>:8000/survey). You remember the post address of the form was "process", and here we connect it to the correct view as well. The third url() you can already figure out by yourself (Hint: we referenced it when redirecting the user after posting results).

That's it! Save all your files, go back to your console and run the server once more. Once you are convinced everything is alright, run `manage.py flush` to reset the DB, and send your website link to everyone you know. Nothing will stand in your way from playing soccer again.