

Nupic Web Application development

Contents

Focus in.....	1
Why to build a Web Application?	1
The common data flow schema	1
Tools	2
Preparations	2
Download/Install Django.....	2
Check if Django is installed.....	2
Create a Project.....	3
Customize IDE.....	3
Run webserver	7
Homepage	8
Test page	9
webApp structure investigation.....	9
Application workflow schema	10
webApp	11
helloworld folder.....	16
templates	20

Focus in

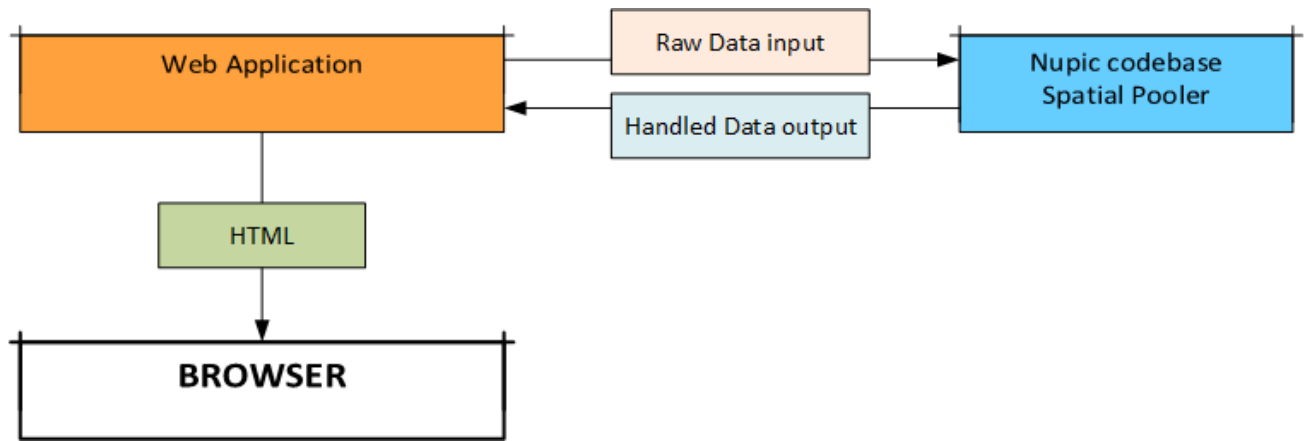
This tutorial is focused mainly on building web application *based* on Nupic, not on Nupic as platform. Consider it as one more tool for learning Nupic and data visualization in most (probably) convenient way. It will not be useful for those who has great experience on web-development in Python, but can help very much for others who wants try Nupic in such a way.

Why to build a Web Application?

There are two main reasons — flexibility and imagery.

You may build/rebuild your application almost without any limitations; the Nupic codebase is encapsulated from it. And you have total control on output data appearance using all richness of HTML/JavaScript/CSS etc.

The common data flow schema



The detailed diagram of the process is given [further](#).

Tools

This tutorial is suited for:

- OS — Windows 7 or Linux (Ubuntu 13.04) with Nupic being preinstalled.
- Platform — Python 2.7x
- IDE — Pycharm Community Edition (optional — you can use any other).
- Web framework — Django 1.6x.

If you wonder — why Django? — the reason is that this framework takes care (and makes you free of) about lots technical things, in particular — setting/using webserver. It just works after installation and you don't need to spend your time to customize it. Actually if you are more familiar with some other python-based web-framework — no problem, it is up to you.

Preparations

First of all you have Python to be installed on your system.

Download/Install Django

<https://www.djangoproject.com/download/>

<https://docs.djangoproject.com/en/1.6/intro/install/>

Note for linux users:

Probably the simplest way is:

```
sudo pip install django
```

Check if Django is installed

```
>>>python
```

```
>>>import django
>>>print(django.get_version())
[version of django]
```

If you get the Django version here, then everything OK; go to the next step — project creating.

Create a Project

1. Create the directory for your project.
2. **For Windows users:**

Copy the **django-admin.py** file from the directory where Django had been installed

```
C:\Python27\Lib\site-packages\django\bin\django-admin.py
C:\Python27\Scripts\django-admin.py
```

Note: both files are the same.

- ...to the your project's directory. Let's name it "webApp"
3. Create the Django project
 - a. Go to the directory where your project should be located:

```
>cd path_to_project/webApp
```

- b. Make a project:

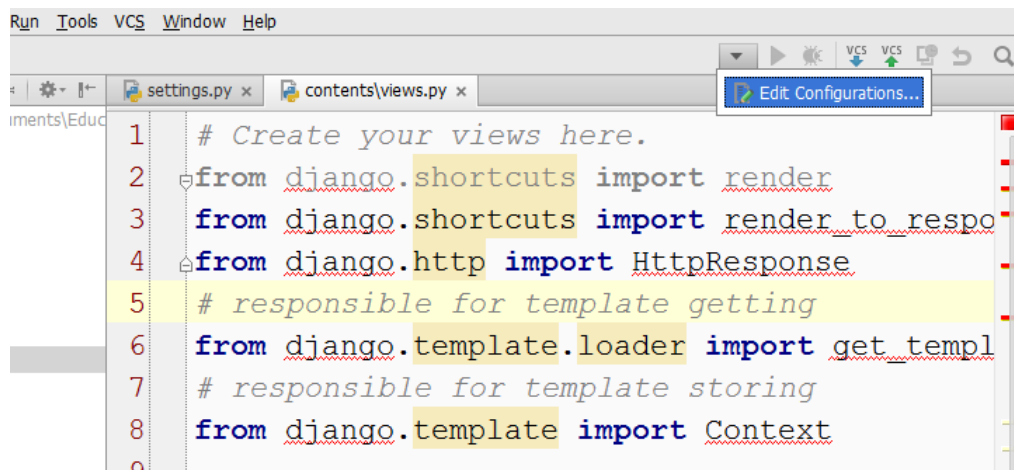
```
>python django-admin.py startproject webApp
```

The official tutorial is here:

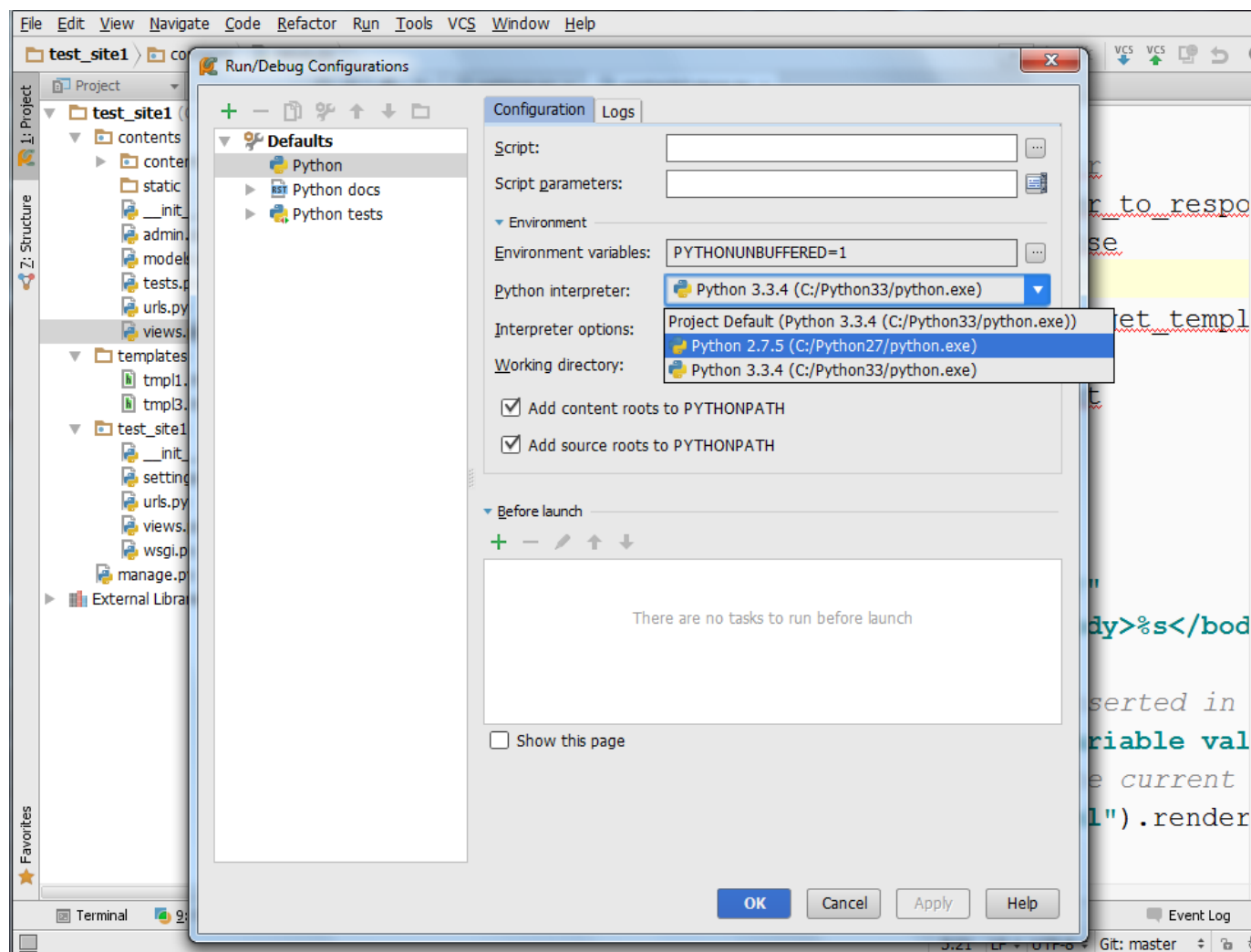
<https://docs.djangoproject.com/en/1.6/intro/tutorial01/#creating-a-project>

Customize IDE

Check Python version for the project:

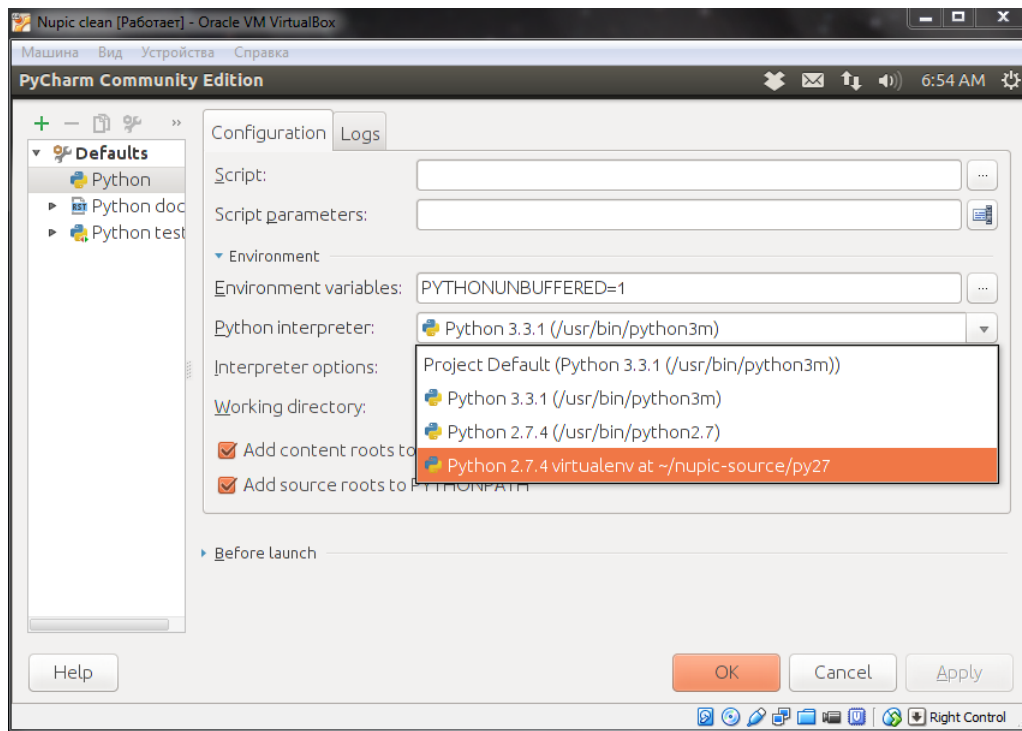


```
1 # Create your views here.
2 from django.shortcuts import render
3 from django.shortcuts import render_to_response
4 from django.http import HttpResponseRedirect
5 # responsible for template getting
6 from django.template.loader import get_template
7 # responsible for template storing
8 from django.template import Context
```

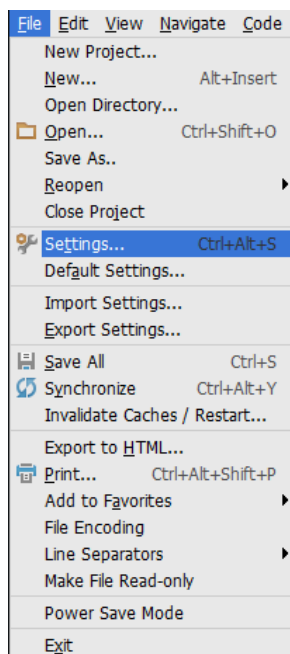


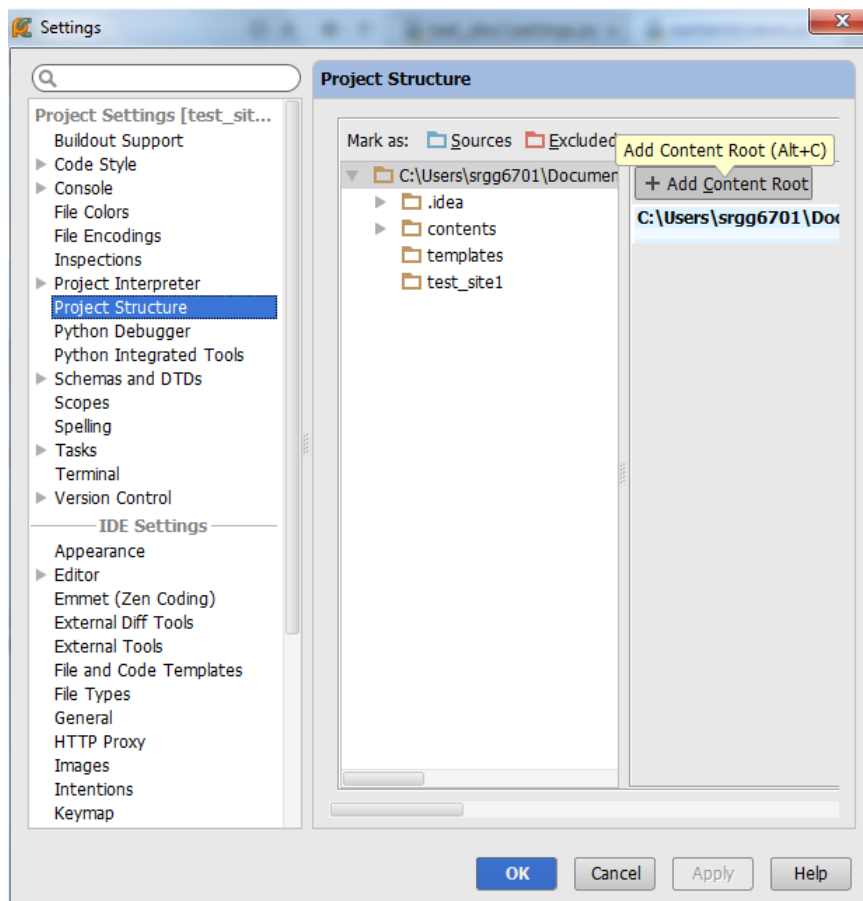
Note for VM users:

Point out the **virtualenv** at the **nupic** location:

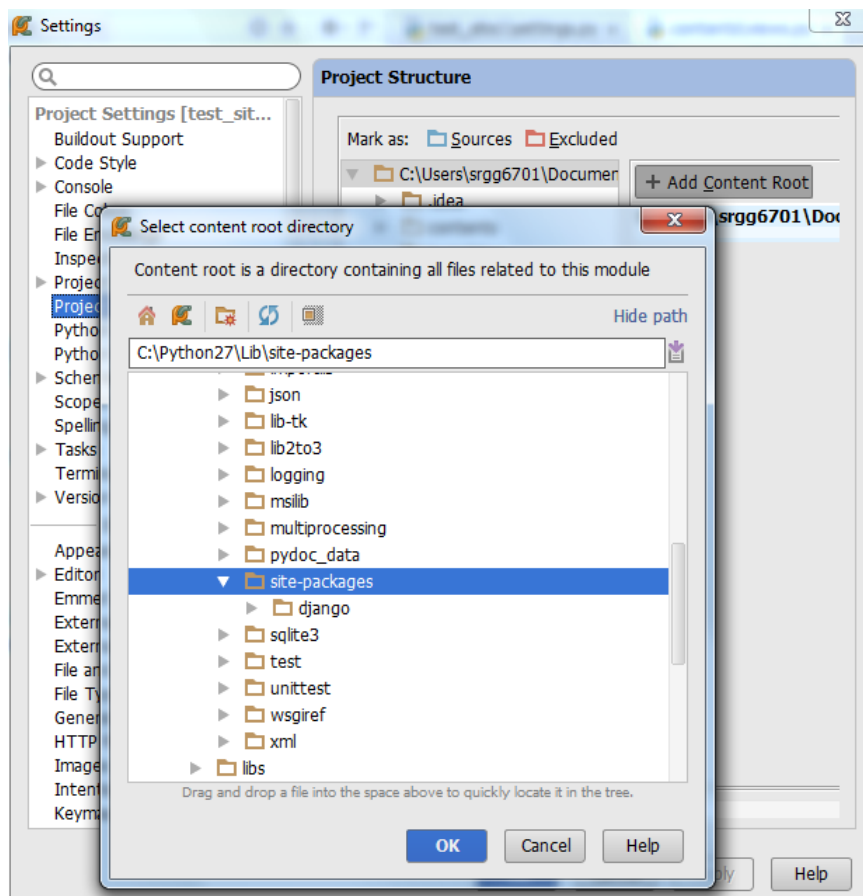


Add the Django installation directory to the project content root:

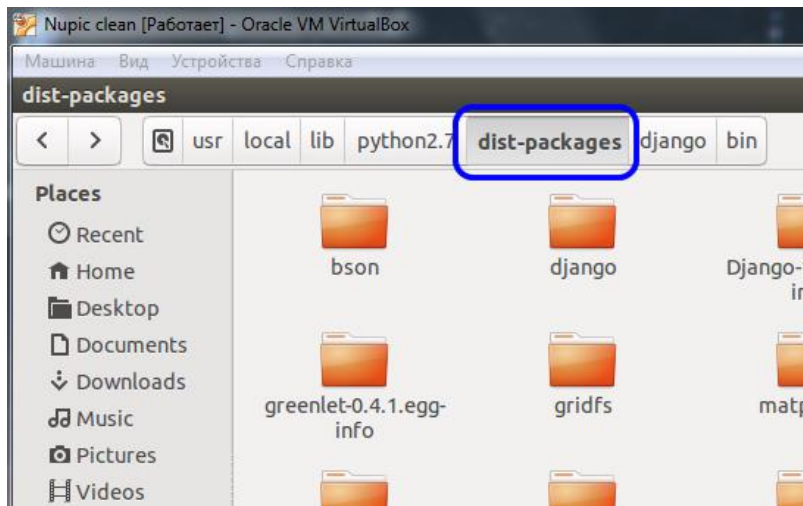




Windows:



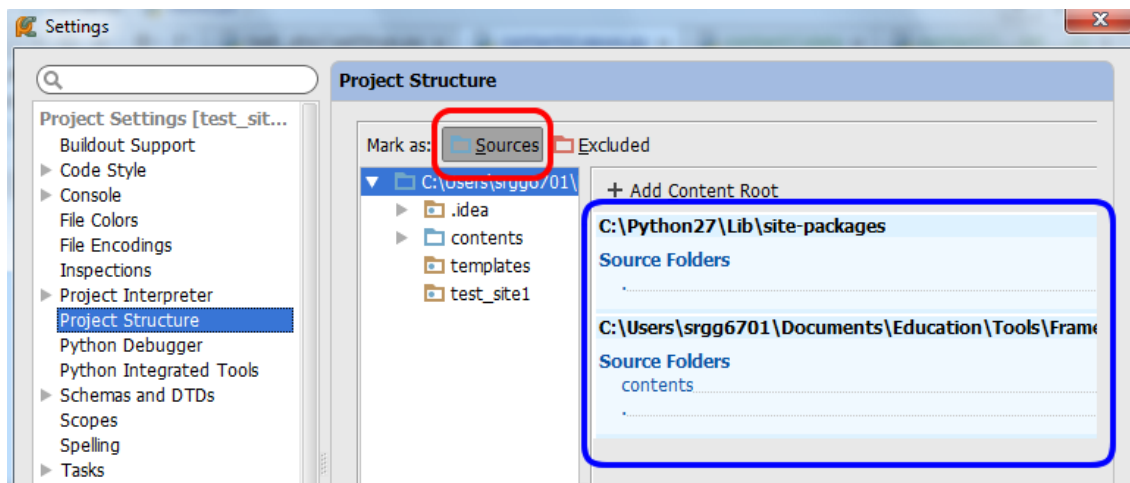
Linux:



After that your IDE has to see all imports:

```
1  # Create your views here.
2  from django.shortcuts import render
3  from django.shortcuts import render_to_response
4  from django.http import HttpResponse
5  # responsible for template getting
6  from django.template.loader import get_template
7  # responsible for template storing
8  from django.template import Context
```

If it doesn't, make sure that all your paths are marked as "sources".



Run webserver

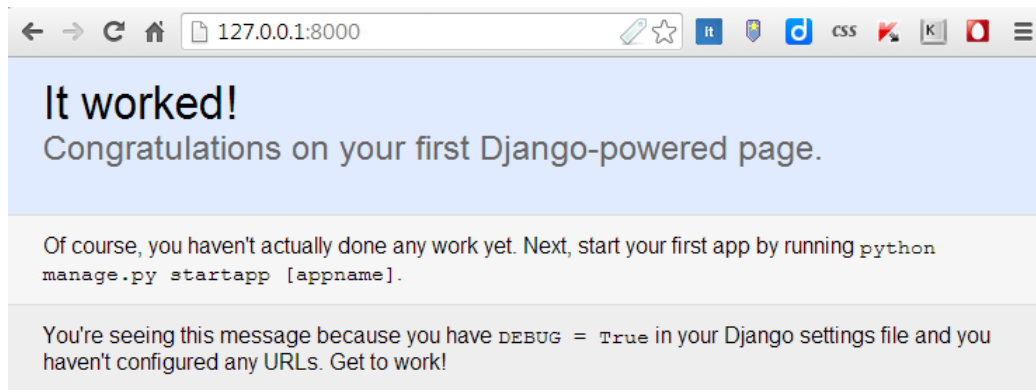
Now go to the project directory (in this location should be the file which is named **manage.py**) and make a command:

```
>python manage.py runserver
```

You have to see the message in your console/terminal:

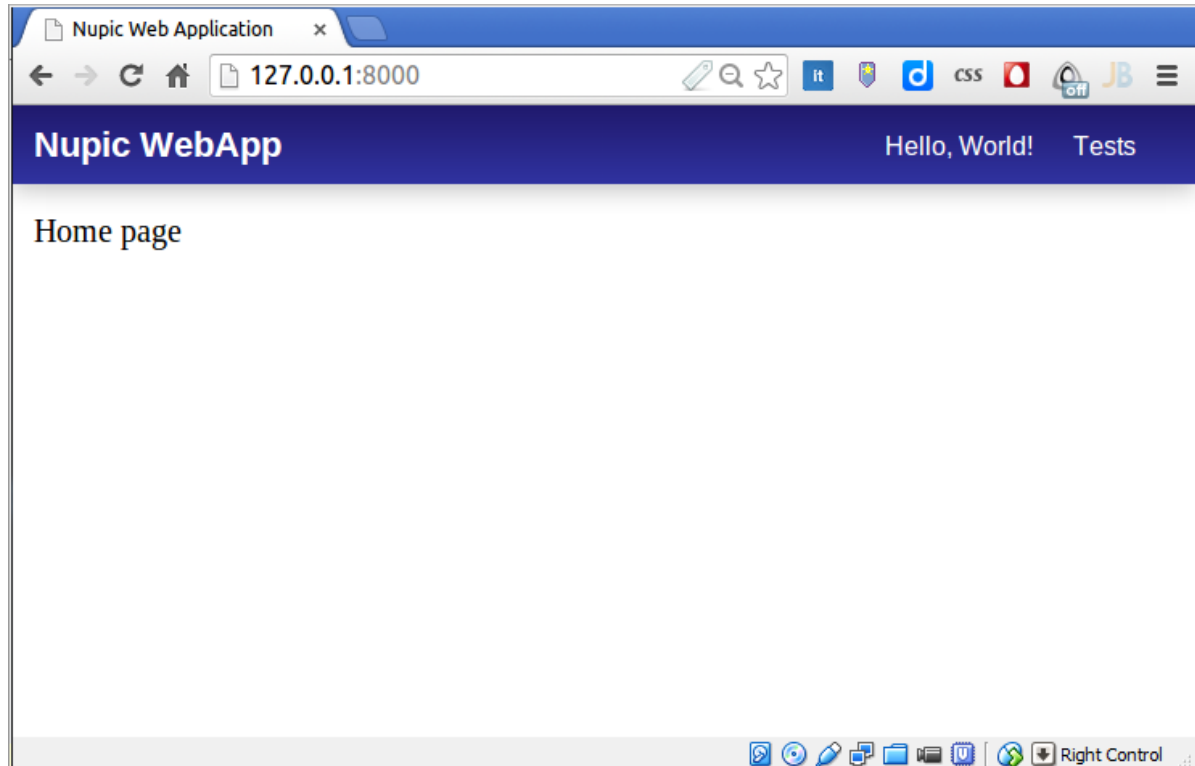
```
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.
```

Try to run a browser and go to this address. You have to see something like this:



Homepage

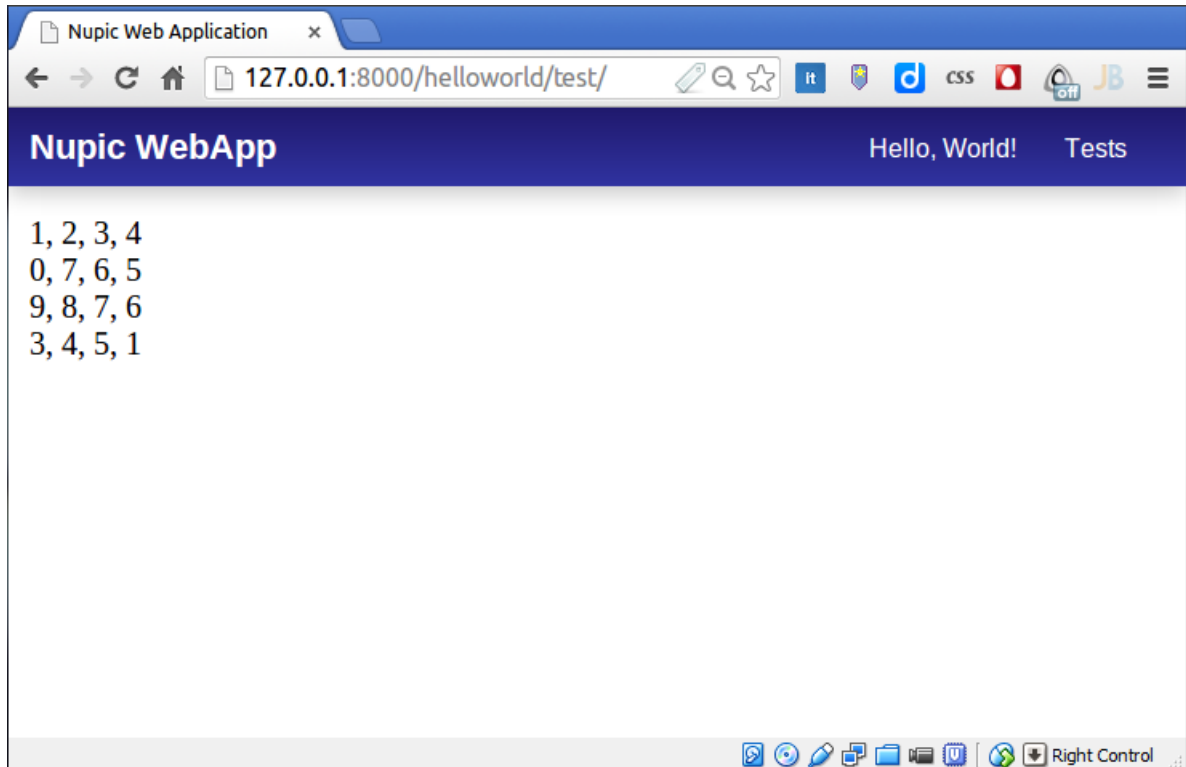
As you got the site prototype you may clone its content from a remote repository (of course you can also continue to build it by yourself). In this case your main page will look like this:



There are two links now. The second one is for testing.

Test page

If you click it you should see something like this:

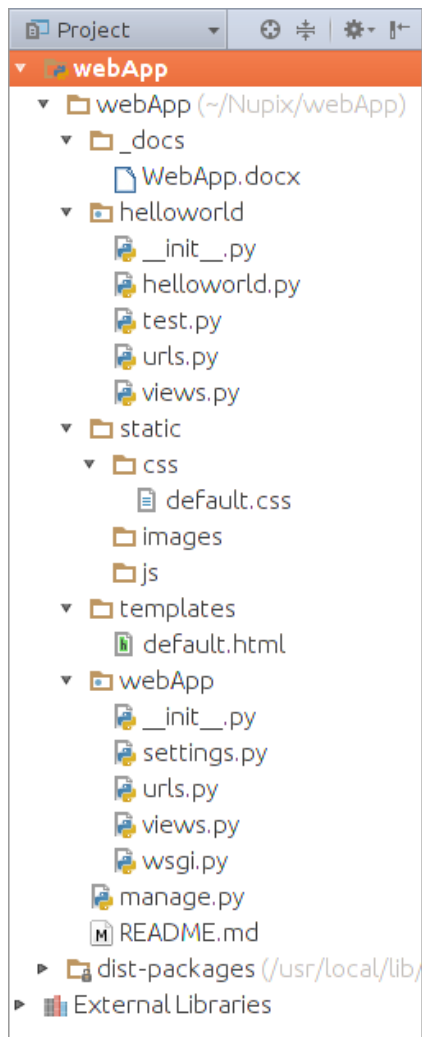


Its content comes from the **helloworld/test.py** file.

webApp structure investigation

OK, it is the time to focus on the project's structure.

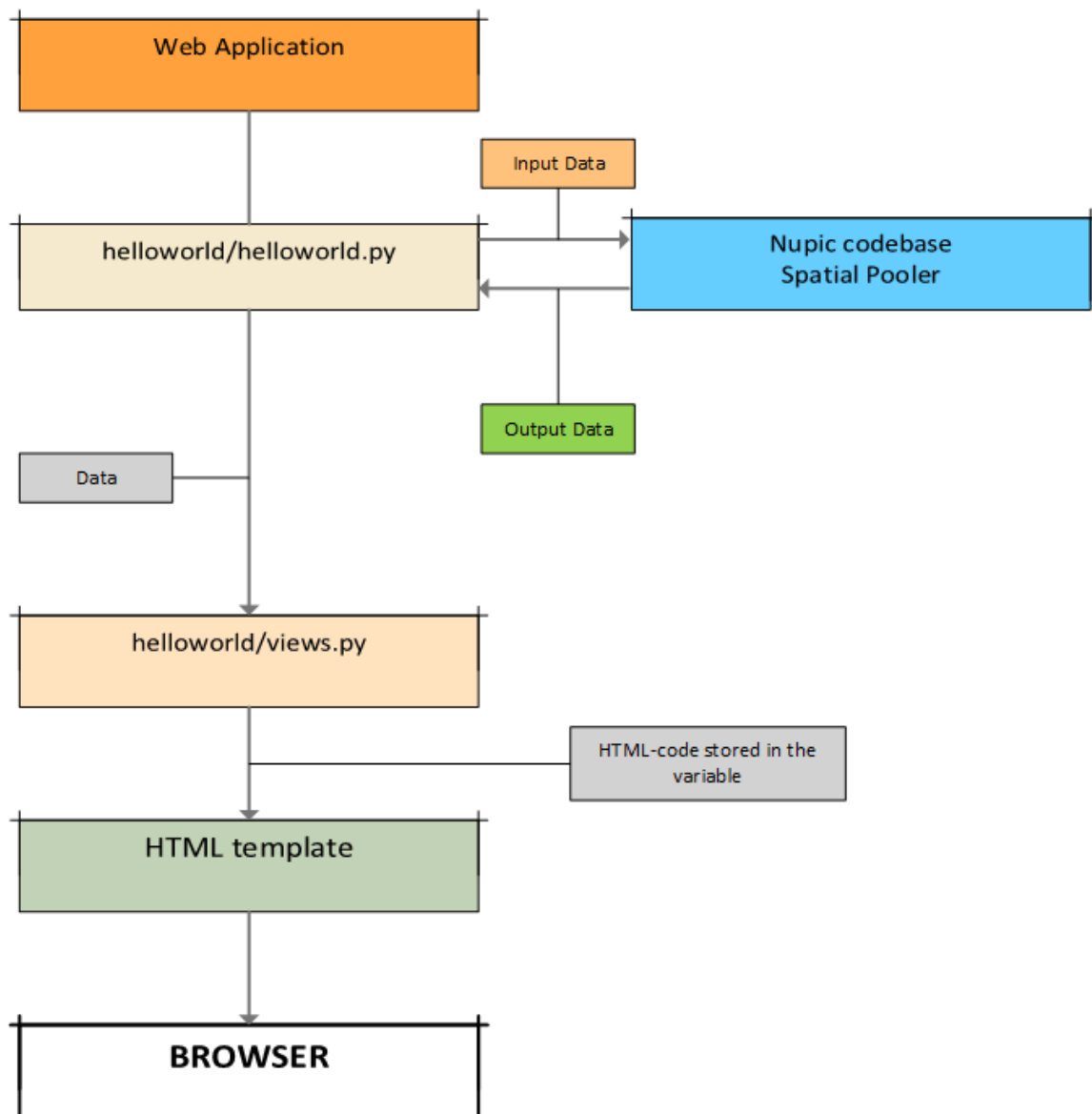
If you've got the site from a remote repository you have to see something like this:



Let's investigate our web application's folders and files!

Note that Python (and Django) let you design it as you prefer, so such a structure may be changed if you want.

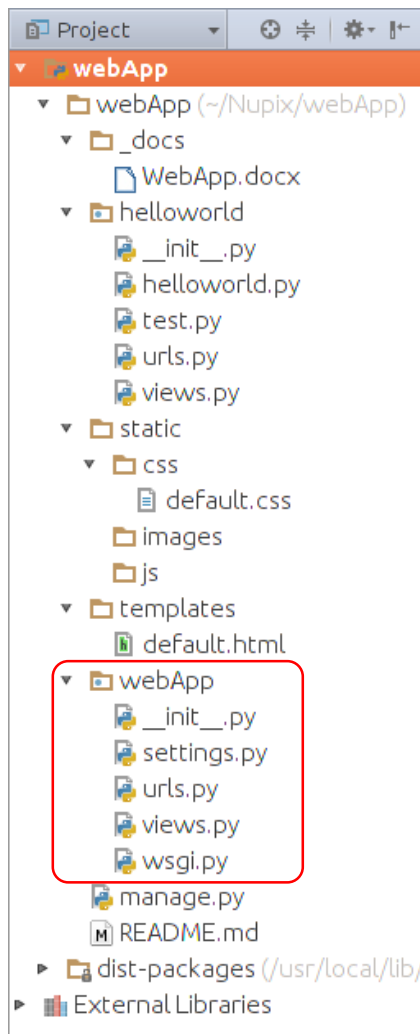
Application workflow schema



As mentioned earlier, the Nupic codebase is encapsulated from Web application. It gets inputs and returns handled data. First of all we have to consider the folder which contains the application settings.

webApp

We have two such directories. One is our project's root directory while another one is the "application" directory (the details about this term are [bellow](#)).

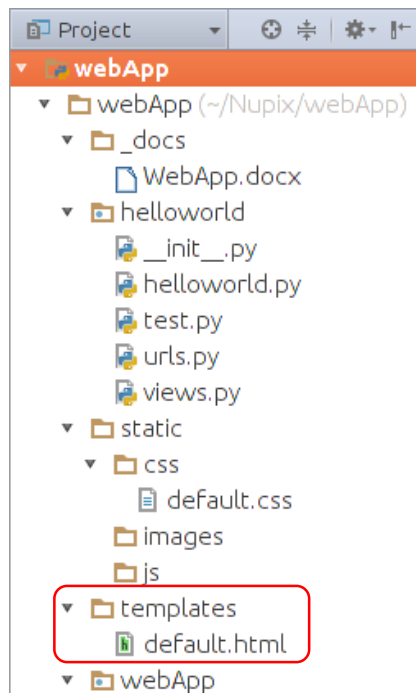


The file **wsgi.py** will not be considered here because it is applied only with Django itself. You can find the full information about it in the [official site](#). But files **settings.py**, **urls.py** and **views.py** are interesting for us.

settings.py

As its name suggests it is the place where the application settings live. Some of them are set while creating project (i.e. it happens automatically), other must be set manually. In our case the latter ones are:

The HTML-[templates directory](#)



It sets here:

```
TEMPLATE_DIRS = ('/home/nupic/Nupix/webApp/templates')
```

So-called “installed applications”

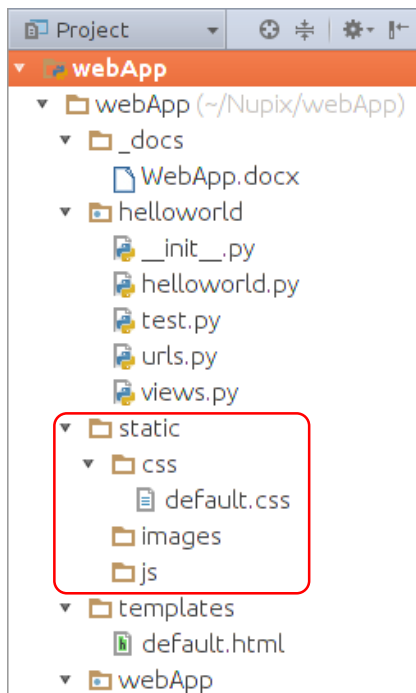
You may consider them here just as directories, which should be accessible for the application:

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

Our one has the single such directory named [helloworld](#).

Note that in the reality this is the more sophisticated concept. If you are interested of it, look the additional information here: <https://docs.djangoproject.com/en/1.6/intro/tutorial01/#creating-models> (see the section “Projects vs. apps”). But for our current purposes the information mentioned above is enough.

“static” directory



```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.6/howto/static-files/
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    ('static', '/home/nupic/Nupix/webApp/static'),
)
```

More detailed about such a kind of static folder/files is [here](#).

urls.py

This is the main (default) application router. It uses URL patterns to do following 2 things:

1. Assign the function which should form the HTML-content which must be injected into the template.
2. Tells the browser a location which it must go to.

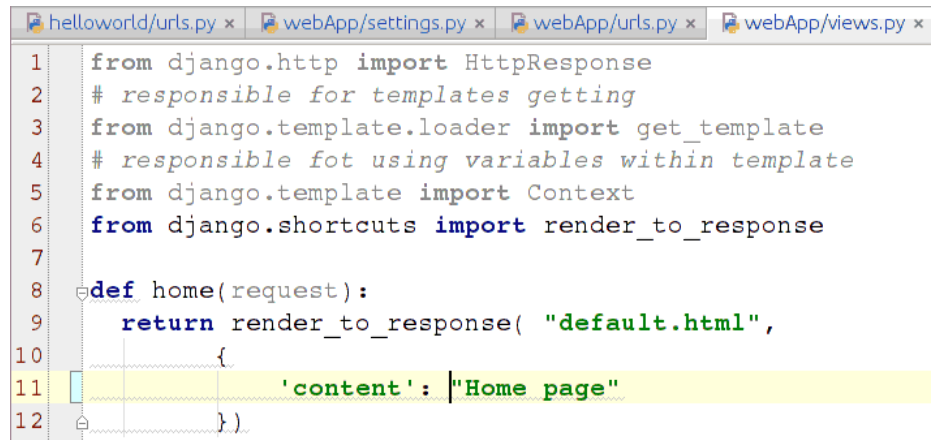
```
1 from django.conf.urls import patterns, include, url
2
3 urlpatterns = patterns('',
4     url(r'^$', 'webApp.views.home', name='home'),
5     url(r'^helloworld/', include('helloworld.urls')),
6 )
```

You can see here two routes:

The route #1

The string in the row #4 means:

If the location contains only a **[domain_name]** (may be with “/” in the end; in our case it is <http://127.0.0.1:8000/>) it should call the function **home()** which is placed in the file **webApp/views.py**:



```
1 from django.http import HttpResponse
2 # responsible for templates getting
3 from django.template.loader import get_template
4 # responsible for using variables within template
5 from django.template import Context
6 from django.shortcuts import render_to_response
7
8 def home(request):
9     return render_to_response( "default.html",
10                               {
11                                   'content': "Home page"
12                               })
```

This function assigns the HTML-content for the variable “content” which is used within HTML-template:



```
1 {% load staticfiles %}
2 <!DOCTYPE html><!-- TODO resolve the issue above -->
3 <html>
4 <head>
5     <title>Nupic Web Application</title>
6     <link rel="stylesheet" type="text/css" href="{% static
7         </style>
8 </head>
9 <body>
10 <div id="logo">
11     <a href="/" class="logo">Nupic WebApp</a>
12     <menu class="menu">
13         <li><a href="/helloworld">Hello, World!</a></li>
14         <li><a href="/helloworld/test">Tests</a></li>
15     </menu>
16 </div>
17 <main>
18     {{ content }}
19     {{ html|safe }}
20 </main>
21 </body>
22 </html>
```

As a result the string “Home page” [appears](#).

The route #2

The [string in the row #5](#) means:

If the location is **[domain_name]/helloworld** (it also may be terminated by “/”) the file [helloworld/urls.py](#) must be included.

Pay attention that every “application” ([mentioned earlier](#)) within a project should contain such a file. You may think about it as a *second-level* router, which handles **URLs** within its *own application* scope. More information [follows further](#).

views.py

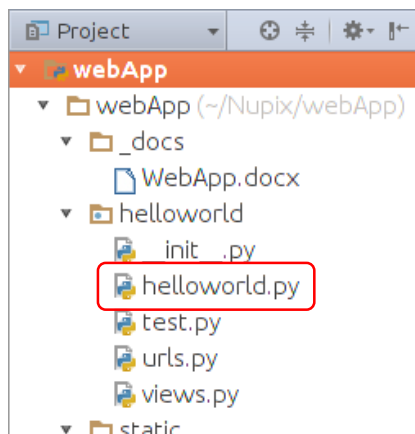
Its purpose has been described [above](#).

Notice, that as in the [case with urls.py](#) such files should be placed within every [application](#) in a project.

helloworld folder

In fact this is the most important place in our simplest application. It contains the file **helloworld.py** which, in turn, contains the script connected to the Nupic base application.

helloworld.py



This script in essence is the example of another simple application (built just on Python) which name speaks for itself — **nupichelloworld**. You can get it here: <https://github.com/loneword/nupichelloworld>. The only difference between its file **helloworld.py** and the file **helloworld/helloworld.py** in webApp is that the native one outputs resulting code in the console while the second one transmits it to the function **helloworld()** in the [helloworld/views.py](#).

The code difference is:

native helloworld.py

```
def run(self):
    """Run the spatial pooler with the input vector"""

    #activeArray[column]=1 if column is active after
    self.sp.compute(self.inputArray, True, self.activeArray)

    print self.activeArray.nonzero()

example = Example((32, 32), (64, 64))

#Trying random vectors
for i in range(3):
    example.create_input()
    example.run()

print "-"*75+"Using identical input vectors"+"-"*75

#Trying identical vectors
for i in range(2):
    example.run()
```

webapp helloworld/helloworld.py

```
def run(self):
    """Run the spatial pooler with the input vector"""

    #activeArray[column]=1 if column is active after
    self.sp.compute(self.inputArray, True, self.activeArray)

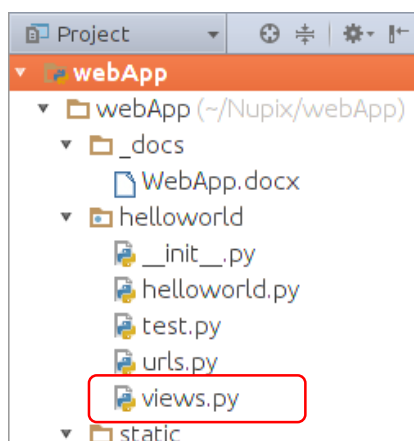
    return self.activeArray.nonzero()

example = Example((32, 32), (64, 64))

#Initialize a list of random data for views
arr_data_random=[]
for i in range(3):
    example.create_input()
    arr_data_random.append(example.run())
    #cnt+=1

#Initialize a list of random data for views
arr_data_identical=[]
for i in range(2):
    example.run()
    arr_data_identical.append(example.run())
```

views.py



So the [previously described file](#) (which can be considered as **Model** in the MVC terminology) returns the data stored in the two arrays: **arr_data_random** and **arr_data_identical**. They are handled here by the **helloworld()** function:

```
# get data from the helloworld.py module
def helloworld(request):
    import helloworld
    html=makeHTML("Random vectors", helloworld.arr_data_random)
    html+=makeHTML("Identical vectors", helloworld.arr_data_identical)
    return render_to_response("default.html", { 'html': html })
```

...which passes them to the next function — **makeHTML()** that does all the job and returns the HTML:

```
def makeHTML(header,array):
    html= "<h3>"+header+"</h3>"
    html+="<div>"
    for lst in array:
        for arr in lst:
            cnt = 0
            for d in arr:
                if cnt:
                    html+=","
                cnt+=1
                html+=str(d)
            html+="<hr/>"
    html+="</div>"
    return html
```

So what you will see is what the Nupic codebase returns. In our simplest case it looks something like this:

Nupic Web Application x

127.0.0.1:8000/helloworld/

Nupic WebApp Hello, World! Tests

Random vectors

5, 22, 65, 237, 389, 437, 483, 522, 647, 668, 702, 742, 863, 899, 1062, 1070, 1140, 1213, 1254, 1255, 1256, 1353, 1389, 1409, 1494, 1528, 1594, 1700, 1711, 1723, 1800, 1826, 1845, 1851, 1894, 1899, 2074, 2077, 2198, 2215, 2353, 2392, 2408, 2426, 2451, 2526, 2540, 2574, 2592, 2607, 2636, 2650, 2658, 2682, 2720, 2762, 2785, 2808, 2883, 2897, 2904, 2918, 2958, 3051, 3066, 3170, 3305, 3308, 3325, 3363, 3414, 3415, 3463, 3523, 3709, 3727, 3748, 3783, 3871, 3935, 4039

32, 52, 61, 123, 128, 139, 213, 297, 411, 427, 455, 463, 494, 559, 566, 577, 616, 658, 700, 718, 745, 758, 812, 873, 893, 895, 983, 1063, 1145, 1155, 1176, 1262, 1356, 1459, 1492, 1592, 1596, 1616, 1626, 1745, 1757, 1761, 1777, 1801, 1813, 1888, 1947, 2239, 2243, 2260, 2465, 2495, 2539, 2614, 2652, 2723, 2727, 2792, 2801, 2864, 2966, 2982, 3044, 3081, 3118, 3127, 3152, 3301, 3324, 3336, 3456, 3545, 3581, 3586, 3604, 3613, 3634, 3716, 3737, 3820, 4062

62, 166, 204, 216, 276, 531, 549, 564, 600, 629, 675, 680, 682, 714, 919, 974, 991, 1019, 1028, 1037, 1099, 1116, 1135, 1174, 1204, 1246, 1300, 1347, 1363, 1433, 1463, 1579, 1618, 1701, 1737, 1794, 1910, 1930, 2016, 2132, 2183, 2238, 2253, 2295, 2315, 2447, 2476, 2520, 2559, 2639, 2838, 2880, 2887, 2951, 2985, 3025, 3026, 3110, 3122, 3145, 3329, 3345, 3396, 3458, 3513, 3518, 3567, 3573, 3662, 3725, 3760, 3775, 3784, 3799, 3800, 3811, 3822, 4001, 4003, 4042, 4082

Identical vectors

10, 60, 218, 288, 323, 329, 368, 384, 403, 462, 493, 511, 532, 582, 585, 589, 611, 659, 732, 739, 901, 923, 1022, 1094, 1178, 1211, 1380, 1434, 1521, 1539, 1584, 1654, 1704, 1790, 1802, 1867, 1960, 2083, 2117, 2158, 2167, 2173, 2256, 2319, 2320, 2419, 2446, 2497, 2501, 2510, 2549, 2576, 2612, 2657, 2679, 2694, 2807, 2828, 2921, 2962, 2977, 3022, 3048, 3063, 3082, 3139, 3282, 3385, 3580, 3603, 3667, 3671, 3673, 3685, 3705, 3738, 3745, 3746, 4017, 4090, 4095

24, 34, 56, 68, 141, 256, 333, 338, 342, 383, 467, 479, 526, 542, 569, 602, 623, 640, 653, 735, 747, 804, 807, 851, 992, 1030, 1172, 1206, 1225, 1232, 1271, 1373, 1490, 1518, 1541, 1667, 1675, 1804, 2008, 2031, 2061, 2154, 2223, 2268, 2273, 2335, 2562, 2583, 2598, 2642, 2663, 2708, 2736, 2788, 2853, 2885, 2936, 2939, 3033, 3053, 3109, 3146, 3176, 3182, 3199, 3209, 3225, 3237, 3238, 3337, 3460, 3494, 3570, 3590, 3601, 3740, 3761, 3888, 3941, 3955, 3968

You can consider it as a starting point for own investigations and improvement of this web application.

Also there is another function — **test()**.

```
# get data from the test.py module
def test(request):
    import test
    html=""
    for arr in test.arr1:
        html+=''<div>'
        cnt = 0
        for d in arr:
            if cnt:
                html+=', '
            html = html+str(d)
            cnt +=1
        html+=''</div>'
    return render_to_response("default.html", {'html': html})
```

It just handles data coming from testing file considered [next](#).

test.py

As being said [earlier](#) it is created purely for testing purposes.

```
helloworld/views.py x helloworld/test.py x
1 #content = "Hello! This is the test content from test.py file!"
2 #      Let's look at the array!"
3
4 arr1 = [
5     [1,2,3,4],[0,7,6,5],[9,8,7,6],[3,4,5,1]
6 ]
7
```

This is used by function [test\(\)](#) in the [helloworld/views.py](#) file. This function just returns some content and injects it into the [HTML-template](#).

urls.py

This file is for routing within the additional (inner) application in the project.

```
helloworld/urls.py x
1 from django.conf.urls import patterns, url
2 import views
3 urlpatterns = patterns('',
4     #helloworld
5     url(r'^$', 'helloworld.views.helloworld'),
6     #helloworld/test
7     url(r'^test/', 'helloworld.views.test'),
8 )
```

It is used by Django after the [default router](#) is being handled.

The code in the string #5 means:

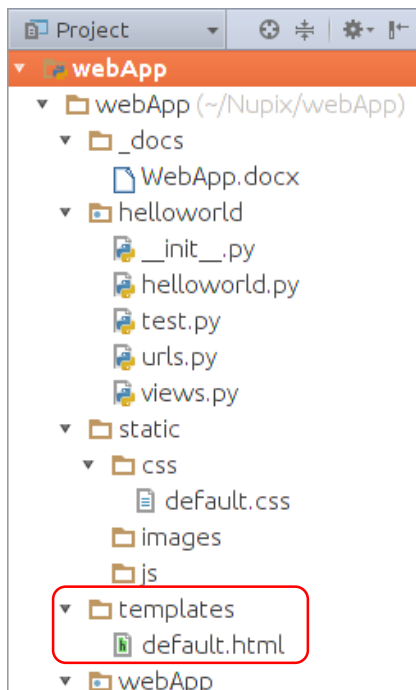
If page location is **[domain_name]/[application_name]** (in our case it is <http://127.0.0.1:8000/helloworld>) call the function [**helloworld\(\)**](#) in the file **helloworld/views.py** ([considered earlier](#) as well).

The code in the string #7 means:

If page location is **[domain_name]/[application_name]/test** then call the function [**test\(\)**](#) in the file **helloworld/views.py** ([considered earlier](#) as well).

templates

We have here the single file:



But you can add own templates if it is necessary.

default.html

So Django injects content returning by the [**helloworld\(\)**](#) function (and the [**test\(\)**](#) [function](#) is used for that also) into HTML-template:

```

html body main
1  {% load staticfiles %}
2  <!DOCTYPE html><!-- TODO resolve the issue above -->
3  <html>
4  <head>
5      <title>Nupic Web Application</title>
6      <link rel="stylesheet" type="text/css" href="{% static
7          </style>
8  </head>
9  <body>
10 <div id="logo">
11     <a href="/" class="logo">Nupic WebApp</a>
12     <menu class="menu">
13         <li><a href="/helloworld">Hello, World!</a></li>
14         <li><a href="/helloworld/test">Tests</a></li>
15     </menu>
16 </div>
17 <main>
18     {{ content }}
19     {{ html|safe }}
20 </main>
21 </body>
22 </html>

```

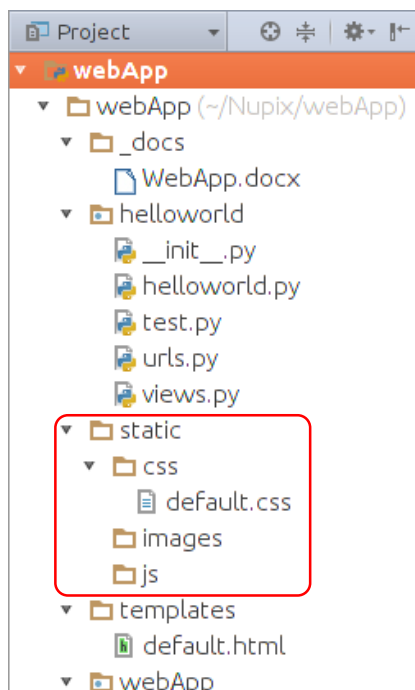
Used when the default page is loaded

Filter, preserving HTML

Generated content

static

This is a special folder to hold such files like **css**, **images**, **javascripts** etc, which are considered as “static”, i.e. not being handled by Python itself:



Now there is only one file which contains CSS-styles for our application. To include it into template the following directive is used:

```
<link rel="stylesheet" type="text/css" href="{% static 'static/css/default.css"
```

As you can see there is a Django directive:

```
{% static 'static/css/default.css' %}
```

which tells Python to include this file into HTML.

Two another folders are reserved for further development.

That's all. Happy coding!