Пајтон сервер на Ђангу

* Install [python 3](http://python.org/)
* Install virtual environment:



|  |  |
| --- | --- |
| 1 | # pip install virtualenv |

* create an empty folder
* create and activate a virtual environmemt:



|  |  |
| --- | --- |
|  | # virtualenv -p /usr/local/bin/python3 env    # source env/bin/activate  (za windows pogledati uputstvo dole) |

* install Django



|  |  |
| --- | --- |
| 1 | # pip install django |

* create the app



|  |  |
| --- | --- |
| 1  2  3 | # django-admin startproject helloapp    # python manage.py startapp first |

* Add the module name (first) to the installed apps (setting.py):



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | INSTALLED\_APPS = [      'django.contrib.admin',      'django.contrib.auth',      'django.contrib.contenttypes',      'django.contrib.sessions',      'django.contrib.messages',      'django.contrib.staticfiles',      'first',  ] |

* Add migration so you can change your database schema easily:



|  |  |
| --- | --- |
| 1 | # python manage.py migrate |

* Run the server to see the first app:



|  |  |
| --- | --- |
| 1 | # python manage.py runserver |

You can now open your web browser and go to http://localhost:8000 to see its working.

Add some views to your app:

* Add new entry to urls.py (urlpatterns):



|  |  |
| --- | --- |
| 1 | url(r'^', include('first.urls')) |

* Add file urls.py to the App folder (first):



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from django.conf.urls import url  from first import views    urlpatterns = [      url(r'^$', views.HomePageView.as\_view()),      url(r'^links/$' , views.LinksPageView.as\_view()),  ] |

* Create a templates folder on the module folder(first) and add your html files (index.html, links.html)
* On the views.py file add the following class:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from django.shortcuts import render  from django.views.generic import TemplateView    # Create your views here.  class HomePageView(TemplateView):      def get(self, request, \*\*kwargs):          return render(request, 'index.html', context=None)    class LinksPageView(TemplateView):      def get(self, request, \*\*kwargs):          return render(request, 'links.html', context=None) |

* Run the server again and see the results (try home page and /links address

Instalacija virtualenv na Windows OS

virtualenv

In your Command Prompt enter:

pip install virtualenv

Launch virtualenv

In your Command Prompt navigate to your project:

cd your\_project

Within your project:

virtualenv env

Activate your virtualenv:

on Windows, virtualenv creates a batch file

\env\Scripts\activate.bat

to activate virtualenv on Windows, activate script is in the Scripts folder:

\path\to\env\Scripts\activate

Example:

C:\Users\'Username'\venv\Scripts\activate.bat

[ANGULAR AND DJANGO – ADDING NUMPY, PANDAS, MATPLOTLIB AND SEABORN](https://devarea.com/angular-and-django-adding-numpy-pandas-matplotlib-and-seaborn/)

BY [LIRAN B.H](https://devarea.com/author/liran/) | [AUGUST 28, 2017](https://devarea.com/2017/08/28/) | [2 COMMENTS](https://devarea.com/angular-and-django-adding-numpy-pandas-matplotlib-and-seaborn/#comments) | [ANGULAR](https://devarea.com/category/angular/), [DJANGO](https://devarea.com/category/django/), [PYTHON](https://devarea.com/category/python/)

In the [first post](https://devarea.com/building-a-simple-website-with-angular-4-and-django-and-deploy-it-to-heroku/) I covered the process of building a simple site with Angular 4 on the client and Django on the server and how to deploy it to [heroku](http://heroku.com/). One great benefit to use python is the huge amount of packages it has

Python is great choice for data analysis, it has some popular packages to make development very easy:

* [NumPy](http://www.numpy.org/) – package for creating and using multi-dementional arrays
* [Matplotlib](https://matplotlib.org/) – package for creating graphs
* [Pandas](http://pandas.pydata.org/) – package for data handling
* [Seaborn](http://seaborn.pydata.org/) – package for data visualization , build on matplotlib

In this post I will cover the process of integrating the above packages on the server side and use it from Angular

Using NumPy

First we need to install it on our virtual environment:



|  |  |
| --- | --- |
| 1  2 | (env) # pip install numpy  (env) # pip freeze > requirements.txt |

Then we can add code to use numpy arrays for example – adding new view:



|  |  |
| --- | --- |
| 1  2  3  4 | def getNums(request):          n = np.array([2, 3, 4])          name1 = "name-" + str(n[1])          return HttpResponse('{ "name":"' + name1 + '", "age":31, "city":"New York" }') |

Add url to match the view:



|  |  |
| --- | --- |
| 1 | url(r'^getnum/$',views.Customers.getNums), |

Run the server and test the result by using the url: 127.0.0.1:8000/getnum

https://devarea.com/wp-content/uploads/2017/08/Screen-Shot-2017-08-27-at-16.05.23-300x43.png

**More useful example**:

We want to send list of numbers to manipulate it on server – for example calculate average: we build a simple form:

https://devarea.com/wp-content/uploads/2017/08/Screen-Shot-2017-08-28-at-15.05.07-300x26.png

**The Angular side:**

HTML:



|  |  |
| --- | --- |
| 1  2  3 | <input type="text" [(ngModel)]='str1'>    <button (click)="getavg()">Get Average</button>    <input type="text" value={{resavg}}> |

JavaScript:



|  |  |
| --- | --- |
| 1  2  3 | getavg(){     this.\_http.get("/getavg?val=" + this.str1).map(r=>r.text()).subscribe(res => this.resavg= res)    } |

**The server side:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | def getAvg(request):          s1=request.GET.get("val","")          if len(s1)==0:              return HttpResponse("none")          l1=s1.split(',')          ar=np.array(l1,dtype=int)            return HttpResponse(str(np.average(ar))) |

of course it  is just an example of something you can easily do on the client but if you need something more complex (linear algebra etc.) this is the way

Using Matplotlib

Matplotlib package helps you create many types of graphs. it is based on Numpy Arrays.

Now we want to generate a graph on http request.

The first thing to do is installing matplotlib in our virtual environment:



|  |  |
| --- | --- |
| 1  2 | (env) # pip install matplotlib  (env) # pip freeze > requirements.txt |

Then we need to import matplotlib and also add the following command:



|  |  |
| --- | --- |
| 1  2  3 | import matplotlib as pl  pl.use('Agg')  import matplotlib.pyplot as plt |

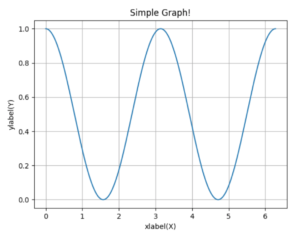
You need to use a non-interactive backend ‘Agg’ to generate png files

The following code generate graph as a view:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | def getimage(request):          x = np.arange(0, 2 \* np.pi, 0.01)          s = np.cos(x) \*\* 2          plt.plot(x, s)            plt.xlabel('xlabel(X)')          plt.ylabel('ylabel(Y)')          plt.title('Simple Graph!')          plt.grid(True)            response = HttpResponse(content\_type="image/jpeg")          plt.savefig(response, format="png")          return response |

Add url for the new view and test it locally:



Load the graph from Angular code:

Add an img tag:



|  |  |
| --- | --- |
| 1 | <img src={{imgsrc}} width="500"> |

Add new data member to the class and connect a button click event:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | export class AppComponent {    ...    imgsrc='';    getimage(){     this.imgsrc = '/getimg';    }    ...      } |

Build the angular code and copy to the static folder of your app:



|  |  |
| --- | --- |
| 1  2 | # ng build  # cp dist/\*.js ~/projects/hello/helloapp/helloapp/static/ |

Using Pandas

Pandas is a great package for data analysis – one object it defines is a dataframe – a 2 dimension table with many methods act on it like SQL.

We need to install it:



|  |  |
| --- | --- |
| 1  2 | (env) # pip install pandas  (env) # pip freeze > requirements.txt |

Now we define a view to return a dataframe in html format:



|  |  |
| --- | --- |
| 1  2  3  4  5 | def getData(request):          samp = np.random.randint(100, 600, size=(4, 5))          df = pd.DataFrame(samp, index=['avi', 'dani', 'rina', 'dina'],                            columns=['Jan', 'Feb', 'Mar', 'Apr', 'May'])          return HttpResponse(df.to\_html(classes='table table-bordered')) |

In the above code we define a numpy array with random numbers, create a DataFrame and convert it to html. The classes we define belongs to bootstrap

Integrate With Angular

We define a span and bind the innerHTML attribute to the pandas table:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <span innerHtml={{pandas}}></span>  ...    getDF()  {     this.\_http.get("./getdata").map(r => r.text()).subscribe(v => this.pandas = v);  } |

Using Seaborn

Seaborn helps with creating statistic graphs. It is built on Matplotlib and support some interesting graphs

Install:



|  |  |
| --- | --- |
| 1  2 | (env) # pip install seaborn  (env) # pip freeze > requirements.txt |

To use seaborn, we will use a CSV file taken from [kaggle.com](https://www.kaggle.com/c/titanic) – Titanic Dataset

To load the file into pandas dataframe  we need to add the file to the project files – we simply put it on the same folder as views.py



|  |  |
| --- | --- |
| 1  2  3 | module\_dir = os.path.dirname(\_\_file\_\_)  file\_path = os.path.join(module\_dir, 'titanic\_train.csv')  df = pd.read\_csv(file\_path) |

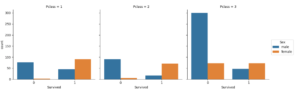
we want to see who survived group by sex and Pclass so we generate a factorplot graph:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | def getSBData(request):          module\_dir = os.path.dirname(\_\_file\_\_)          file\_path = os.path.join(module\_dir, 'titanic\_train.csv')          df = pd.read\_csv(file\_path)          gr=sb.factorplot(x='Survived', hue='Sex', data=df, col='Pclass', kind='count')          response = HttpResponse(content\_type="image/jpeg")          gr.savefig(response, format="png")          return response |

And Again we declare img tag on Angular, bind it to class member and set it on click

This is the result:



Like the above, we can add any package we want , manipulate it on the server and make an http request from the client to load it.

**Final server code(views.py):**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | from django.http import HttpResponse  from django.shortcuts import render  from django.views.generic import TemplateView  import matplotlib as pl  pl.use('Agg')  import pandas as pd  import os  import seaborn as sb      import matplotlib.pyplot as plt  import numpy as np    # Create your views here.  class HomePageView(TemplateView):      def get(self, request, \*\*kwargs):          return render(request, 'index.html', context=None)    class LinksPageView(TemplateView):      def get(self, request, \*\*kwargs):          return render(request, 'links.html', context=None)      class Customers(TemplateView):      def getCust(request):          name='liran'          return HttpResponse('{ "name":"' + name + '", "age":31, "city":"New York" }')      def getNums(request):          n = np.array([2, 3, 4])          name1 = "name-" + str(n[1])          return HttpResponse('{ "name":"' + name1 + '", "age":31, "city":"New York" }')        def getAvg(request):          s1=request.GET.get("val","")          if len(s1)==0:              return HttpResponse("none")          l1=s1.split(',')          ar=np.array(l1,dtype=int)            return HttpResponse(str(np.average(ar)))        def getimage(request):          x = np.arange(0, 2 \* np.pi, 0.01)          s = np.cos(x) \*\* 2          fig,ax = plt.subplots()          plt.plot(x, s)            plt.xlabel('xlabel(X)')          plt.ylabel('ylabel(Y)')          plt.title('Simple Graph!')          plt.grid(True)            response = HttpResponse(content\_type="image/jpeg")          plt.savefig(response, format="png")          return response        def getData(request):          samp = np.random.randint(100, 600, size=(4, 5))          df = pd.DataFrame(samp, index=['avi', 'dani', 'rina', 'dina'],                            columns=['Jan', 'Feb', 'Mar', 'Apr', 'May'])          return HttpResponse(df.to\_html(classes='table table-bordered'))        def getSBData(request):          module\_dir = os.path.dirname(\_\_file\_\_)          file\_path = os.path.join(module\_dir, 'titanic\_train.csv')          df = pd.read\_csv(file\_path)          gr=sb.factorplot(x='Survived', hue='Sex', data=df, col='Pclass', kind='count')          response = HttpResponse(content\_type="image/jpeg")          gr.savefig(response, format="png")          return response |

urls.py



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | from django.conf.urls import url  from first import views    urlpatterns = [      url(r'^$', views.HomePageView.as\_view()),      url(r'^links/$' , views.LinksPageView.as\_view()),      url(r'^getcust/$',views.Customers.getCust),      url(r'^getnum/$',views.Customers.getNums),      url(r'^getimg/$', views.Customers.getimage),      url(r'^getdata/$', views.Customers.getData),      url(r'^getsbdata/$', views.Customers.getSBData),      url(r'^getavg/$', views.Customers.getAvg),    ] |

**Client side** HTML:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <div style="text-align:center">    <h1>      Welcome to {{title}}!    </h1>    <button (click)="click1()" >click me</button>    <input type="text" [(ngModel)]='str1'>    <button (click)="getavg()">Get Average</button>    <input type="text" value={{resavg}}>    <button (click)="getimage()">load graph</button>    <button (click)="getDF()">load table</button>    <button (click)="getsbimage()">seaborn graph</button>    <br/>  </div>  <img src={{imgsrc}} width="500">  <h2>Test App: {{c1.name}}</h2>  <span innerHtml={{pandas}}></span>  <img src={{imgsbsrc}} width="1000"> |

TypeScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | import { Component } from '@angular/core';  import {Http, Response} from '@angular/http';  import {Observable} from 'rxjs';  import 'rxjs/add/operator/map';      @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {      constructor(private \_http:Http) {      this.c1.name = "eli"     }    title = 'app';    imgsrc='';    pandas='';    imgsbsrc='';    str1='';    resavg='';    getimage(){     this.imgsrc = '/getimg';    }    c1:Cust = new Cust();    click1(){      this.getAllBooks().subscribe(b => this.c1 = b)      }    getavg(){     this.\_http.get("/getavg?val=" + this.str1).map(r=>r.text()).subscribe(res => this.resavg= res)    }    getDF(){      this.\_http.get("./getdata").map(r => r.text()).subscribe(v => this.pandas = v);    }    getAllBooks()    {      return this.\_http            .get("./getcust")            .map(r => <Cust>r.json())      }    getsbimage()    {     this.imgsbsrc = '/getsbdata';    }  }    export class Cust{      name:string;      age:number;      city:string; |

https://docs.djangoproject.com/en/2.2/intro/tutorial01/