**DjagoGirls Project**

https://tutorial.djangogirls.org/

Run cmd in project folder 🡪 E:\djangoGirls>python -m venv myvenv 🡪 E:\djangoGirls>myvenv\scripts\activate 🡪 (myvenv) E:\djangoGirls>python -m pip install --upgrade pip 🡪 **make a file as:** requirements.txt in djangoGirls folder including: Django~=3.2.10 🡪 (myvenv) E:\djangoGirls>pip install -r requirements.txt 🡪 **make a site:** (myvenv) E:\djangoGirls>django-admin startproject mysite . 🡪 **make an application:** (myvenv) E:\djangoGirls>django-admin startapp blog 🡪 (myvenv) E:\djangoGirls>code .

**Run Server:** new CMD in project main folder (djangoGirls) : E:\djangoGirls>myvenv\scripts\activate 🡪 python manage.py runserver 🡪 **open your server in browser:** http://127.0.0.1:8000/

Make your site Persian: vs Code 🡪 mysite 🡪 setting.py :

Setting.py

LANGUAGE\_CODE = 'fa-ir'

TIME\_ZONE = 'Asia/Tehran'

STATIC\_URL = '/static/'

STATIC\_ROOT = BASE\_DIR / 'static'

ALLOWED\_HOSTS = ['127.0.0.1', '.pythonanywhere.com']

INSTALLED\_APPS = [

    . . .

    'blog',

]

In first CMD: (myvenv) E:\djangoGirls>python manage.py migrate

### Creating a blog post model

We want to make a blog with following properties: author, title, text, created\_date, published\_date.

# blog/models.py

from django.conf import settings

from django.db import models

from django.utils import timezone

class Post(models.Model):

    author = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

    title = models.CharField(max\_length=200)

    text = models.TextField()

    created\_date = models.DateTimeField(default=timezone.now)

    published\_date = models.DateTimeField(blank=True, null=True)

    def publish(self):

        self.published\_date = timezone.now()

        self.save()

    def \_\_str\_\_(self):

        return self.title

### Create tables for models in your database

The last step here is to add our new model to our database. First we have to make Django know that we have some changes in our model. (We have just created it!) Go to your console window and type python manage.py makemigrations blog.

(myvenv) E:\djangoGirls>python manage.py makemigrations blog

Django prepared a migration file for us that we now have to apply to our database. Type python manage.py migrate blog and the output should be as follows:

(myvenv) E:\djangoGirls>python manage.py migrate blog

**Django admin**

To add, edit and delete the posts we've just modeled, we will use Django admin.

Let's open the blog/admin.py file in the code editor and replace its contents with this:

blog/admin.py

from django.contrib import admin

from .models import Post

admin.site.register(Post)

Go to your browser and type the address <http://127.0.0.1:8000/admin/>. You will see a login page.

To log in, you need to create a superuser - a user account that has control over everything on the site. Go back to the command line, type python manage.py createsuperuser

(myvenv) E:\djangoGirls>python manage.py createsuperuser

Return to your browser. Log in with the superuser's credentials you chose; you should see the Django admin dashboard. Go to Posts and experiment a little bit with it. Add five or six blog posts.

# Deploy your website!

Until now, your website was only available on your computer. Now you will learn how to deploy it! Deploying is the process of publishing your application on the Internet so people can finally go and see your app.

As you learned, a website has to be located on a server. There are a lot of server providers available on the internet, we're going to use PythonAnywhere. PythonAnywhere is free for small applications that don't have too many visitors so it'll definitely be enough for you now.

The other external service we'll be using is GitHub, which is a code hosting service. There are others out there, but almost all programmers have a GitHub account these days, and now so will you!

## Installing Git

You can download Git from [git-scm.com](https://git-scm.com/) and install it. You can hit "next" on all steps except for two: in the step where it asks to choose your editor, you should pick Nano, and in the step entitled "Adjusting your PATH environment", choose "Use Git and optional Unix tools from the Windows Command Prompt" (the bottom option). Other than that, the defaults are fine. Checkout Windows-style, commit Unix-style line endings is good.

Do not forget to restart the command prompt or PowerShell after the installation finished successfully.

### Ignoring files in Git:

Git will track changes to all the files and folders in this directory, but there are some files we want it to ignore. We do this by creating a file called .gitignore in the base directory. Open up your editor and create a new file with the following contents:

# Python

\*.pyc

\*~

\_\_pycache\_\_

# Env

.env

myvenv/

venv/

# Database

db.sqlite3

# Static folder at project root

/static/

# macOS

.\_\*

.DS\_Store

.fseventsd

.Spotlight-V100

# Windows

Thumbs.db\*

ehthumbs\*.db

[Dd]esktop.ini

$RECYCLE.BIN/

# Visual Studio

.vscode/

.history/

\*.code-workspace

## Starting our Git repository

Git tracks changes to a particular set of files in what's called a code repository (or "repo" for short). Let's start one for our project. Open up your console and run these commands, in the djangogirls directory:

(myvenv) E:\djangoGirls>git init

Then, you should sign in in github.com. And then:

(myvenv) E:\djangoGirls>git config --global user.name "fAziz1985"

(myvenv) E:\djangoGirls>git config --global user.email fatemeh.azizzadeh1985@gmail.com

(myvenv) E:\djangoGirls>git status

Go to github.com, login and make a new repository: djangoGirlsPr.

The git status command returns information about any untracked/modified/staged files, the branch status, and much more.

And finally we save our changes. Go to your console and run these commands:

(myvenv) E:\djangoGirls>git add .

(myvenv) E:\djangoGirls>git commit -m "first commit"

(myvenv) E:\djangoGirls>git branch -M main

(myvenv) E:\djangoGirls>git remote add origin https://github.com/fAziz1985/djangoGirlsPr.git

(myvenv) E:\djangoGirls>git push -u origin main

# Setting up our blog on PythonAnywhere

## Sign up for a PythonAnywhere account

fAziz1985

Creating a PythonAnywhere API token: login to pythonanywher.com / account / API token / Create new API token

## Configuring our site on PythonAnywhere

Go back to the main [PythonAnywhere Dashboard](https://www.pythonanywhere.com/) by clicking on the logo, and choose the option to start a "Bash" console – that's the PythonAnywhere version of a command line, just like the one on your computer.

PythonAnywhere command-line

$ pip3.8 install --user pythonanywhere

Now we run the helper to automatically configure our app from GitHub:

PythonAnywhere command-line

$ pa\_autoconfigure\_django.py --python=3.8 https://github.com/fAziz1985/djangoGirlsPr.git

$ pa\_autoconfigure\_django.py --python=3.8 https://github.com/fAziz1985/djangoGirlsPr.git --nuke

PythonAnywhere has automatically activated your virtualenv for you, so all you need to do is run:

PythonAnywhere command-line

$ python manage.py createsuperuser

To view your website on internet:

PythonAnywhere / dashboard / all Web Apps / select your website name / Reload …website name…. / then click on your website name here.

Congratulation! Your website is on internet now.

## What is a URL?

A URL is a web address. You can see a URL every time you visit a website – it is visible in your browser's address bar. (Yes! 127.0.0.1:8000 is a URL! And https://djangogirls.org is also a URL.)

We also want to keep the mysite/urls.py file clean, so we will import URLs from our blog application to the main mysite/urls.py file.

Go ahead, add a line that will import blog.urls. You will also need to change the from django.urls… line because we are using the include function here, so you will need to add that import to the line.

Your mysite/urls.py file should now look like this:

mysite/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('blog.urls')),

]

Django will now redirect everything that comes into '<http://127.0.0.1:8000/>' to blog.urls and looks for further instructions there.

## blog.urls

Create a new empty file named urls.py in the blog directory, and open it in the code editor. All right! Add these first two lines:

blog/urls.py

from django.urls import path

from . import views

Here we're importing Django's function path and all of our views from the blog application. (We don't have any yet, but we will get to that in a minute!)

After that, we can add our first URL pattern:

blog/urls.py

urlpatterns = [

path('', views.post\_list, name='post\_list'),

]

# Django views – time to create!

Let's create a view as the comment suggests. Add the following minimal view below it:

blog/views.py

def post\_list(request):

return render(request, 'blog/post\_list.html', {})

## Your first HTML template!

Creating a template means creating a template file. Everything is a file, right? You have probably noticed this already.

Templates are saved in blog/templates/blog directory. So first create a directory called templates inside your blog directory. Then create another directory called blog inside your templates directory:

blog

└───templates

└───blog

And now create a post\_list.html file (just leave it blank for now) inside the blog/templates/blog directory.

Each HTML page is also divided into two elements: **head** and **body**.

* **head** is an element that contains information about the document that is not displayed on the screen.
* **body** is an element that contains everything else that is displayed as part of the web page.

We use <head> to tell the browser about the configuration of the page,and <body> to tell it what's actually on the page.

## Customize your template

You can now have a little fun and try to customize your template! Here are a few useful tags for that:

* <h1>A heading</h1> for your most important heading
* <h2>A sub-heading</h2> for a heading at the next level
* <h3>A sub-sub-heading</h3> …and so on, up to <h6>
* <p>A paragraph of text</p>
* <em>text</em> emphasizes your text
* <strong>text</strong> strongly emphasizes your text
* <br> goes to another line (you can't put anything inside br and there's no closing tag)
* <a href="https://djangogirls.org">link</a> creates a link
* <ul><li>first item</li><li>second item</li></ul> makes a list, just like this one!
* <div></div> defines a section of the page
* <nav></nav> defines a set of navigation links
* <article></article> specifies independent, self-contained content
* <section></section> defines a section in a document
* <header></header> specifies a header for a document or section
* <main></main> specifies the main content of a document
* <aside></aside> defines some content aside from the content it is placed in (like a sidebar)
* <footer></footer> defines a footer for a document or section
* <time></time> defines a specific time (or datetime)

blog/templates/blog/post\_list.html

<!DOCTYPE html>

<html>

<head>

<title>Django Girls blog</title>

</head>

<body>

<header>

<h1><a href="/">Django Girls Blog</a></h1>

</header>

<article>

<time>published: 14.06.2014, 12:14</time>

<h2><a href="">My first post</a></h2>

<p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.</p>

</article>

<article>

<time>published: 14.06.2014, 12:14</time>

<h2><a href="">My second post</a></h2>

<p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut f.</p>

</article>

</body>

</html>

**Deploy changes on internet**

### Commit, and push your code up to GitHub

First off, let's see what files have changed since we last deployed (run these commands locally, not on PythonAnywhere):

$ git status

Make sure you're in the djangogirls directory and let's tell git to include all the changes in this directory:

$ git add .

Before we upload all the files, let's check what git will be uploading (all the files that git will upload should now appear in green):

$ git status

Now it's time to tell it to save this change in its history. We're going to give it a "commit message" where we describe what we've changed.

$ git commit -m "Changed the HTML for the site."

Once we've done that, we upload (push) our changes up to GitHub:

$ git push

### Pull your new code down to PythonAnywhere, and reload your web app

* Open up the [PythonAnywhere consoles page](https://www.pythonanywhere.com/consoles/) and go to your **Bash console** (or start a new one). Then, run:

PythonAnywhere command-line

$ cd ~/<your-pythonanywhere-domain>.pythonanywhere.com

$ git pull

[...]