Django Start

Django is an MTV framework (models, templates, views)

-models build database tables and handle all interaction with the database

-templates are served in their complete form to the client. Django’s templating allows us to perform some logic and inject data into our HTML documents

-views manage traffic. It’s the view’s job to get the right data for the right route delivered in the right form. Views handle the data we deliver to the user to view

One of the big advantages of Django over Flask is that it compartmentalizes the work into different apps. Flask kept most of the python in one file (server.py) to manipulate the HTML. By separating the functionality into different specialized areas, it’s much easier to work with for large projects/websites.

Django takes a few moves to set up.

First, you will need to set up a virtual Django environment.

> python -m virtualenv djangoEnv

> call djangoEnv/Scripts/activate

(djangoEnv)>pip install Django (if not already installed)

Next, instantiate your project in the folder you want to work in

> django-admin startproject project\_name(your choice!)

Navigate to the project\_name folder you just made and create an apps directory and inside that create an \_\_init\_\_.py with *nul>*

!! You will need to do the below steps for each app!!

Navigate to your apps directory to start a new app (each app is a set of compartmentalized instructions for your routing and login processing.

python ../manage.py startapp first\_app(name the app whatever)

This app is not visible yet to your main project so go to settings.py in the main directory and add your app to the list of INSTALLED APPS with the format:

*‘apps.app\_name’,*

Great! Now it is visible to the main project. You can test here that it is working by navigating to your main folder in the command line and running: python manage.py runserver. It will let you know where the project lives (usually local.host:8000)

For your URL routing, open up the main project .urls file and import *include* in addition to *urls.* Under urlpatterns, change the line to: url(r'^', include('apps.first\_app.urls'copy)) which will tell the routing to go through your app’s urls file.

Your urls file doesn’t exist yet so create that in the apps folder (nul> urls.py). In that file, add the following to direct your routes:

from django.conf.urls import url

from . import views

urlpatterns = [

url(r'^$', views.index)

]

This will be where you add your future routes (inside your urlpatterns). With good regex work, you can really direct your workflow. The above says ‘if there is no route name, use the index function inside your views file’.. (r’^dashboard$’, views.dashboard) would run the dashboard function whenever a redirect to localhost.8000/dashboard is performed.

You can also pass variables along (r’^user/(?P<id>\d+)$’, views.user) passes along a variable ‘id’ whenever a route /user/’somenumber’ is passed.

**Views**

Views handle all of your logic (adding to database, creating session data, telling which html template to load, etc).

Start it out something like this:

from django.shortcuts import render, HttpResponse, redirect

# the index function is called when root is visited

*def* index(request):

response = "Hello, I am your first request!"

return HttpResponse(response)

You will add a lot of functions that will do everything from create new users and other objects as defined by your models.py (coming up), passing info from page to page, verify passwords and more.

**Templates**

To serve up your static files, make a folder called templates for html files and a folder called static for css and js.

*Project\_name/apps/app\_name/templates/app\_name/file\_name.ext*

*Project\_name/apps/app\_name/static/app\_name/css(or js)/file\_name.txt*

In your html file you can link your css/js by:

{% load static %}

<!-- The line above tells Django to be ready to listen for static files -->

<link rel="stylesheet" href="{% static 'ourApp/css/main.css' %}" media="screen" title="no title" charset="utf-8">

**Form and Session**

<form action="/blogs/create" method="post">

{% csrf\_token %}

Blog Name: <input type="text" name="name" placeholder="blog name">

Description: <textarea name="desc" placeholder="description"></textarea>

<input type="submit" value="submit">

</form>

You need to add the csrf token as above to your forms. This is protected by middleware that authenticates your post data and keeps it safe. You can access any of your ‘post’ info inside your views. Just call request.POST[‘name\_from\_form’] (or request.GET[], request.method, etc)

To be able to use session data (and your models), quit your server and run the following in your directory for the project—python manage.py makemigrations (it will check that your data is valid) and then—python manage.py migrate to complete your migrations and activate session data.

Now you can start storing any extra data you want during the user session. Simply set your data to request.session[‘name’]! You can store values, names, dictionaries or whatever!

To call it in your html, {{request.session.name}}! Session data is saved even if you leave the browser so you may want a del request.session[‘key’] to clear out old session data.

**Models**

At some point, you will need want to create data for your database to use. This is easy in Django. Open up a models.py file in your app and build the data you want to create!

The default database is SQLite, to see what is inside, I’ve been using DB Browser

# Inside models.py

from \_\_future\_\_ import unicode\_literals

from django.db import models

# Create your models here.

class Blog(models.Model):

name = models.CharField(max\_length=255)

desc = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

class Comment(models.Model):

comment = models.CharField(max\_length=255)

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

# Notice the association made with ForeignKey for a one-to-many relationship

# There can be many comments to one blog

blog = models.ForeignKey(Blog, related\_name = "comments")

class Admin(models.Model):

first\_name = models.CharField(max\_length=255)

last\_name = models.CharField(max\_length=255)

email = models.CharField(max\_length=255)

blogs = models.ManyToManyField(Blog, related\_name = "admins")

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

Read up more on foreign key and many to many relationships to be able to create classes that stay linked though auto-generated incrementing ids.

You will need to make and run migrations before the classes can be used.

Now these classes can be called and used like an object. Below are a few basic commands. <https://tutorial.djangogirls.org/en/django_orm/>

user = User.objects.create()- creates a new user without any values. You can add values inside the parens or instantiate the new user and assign your own values (e.g. user.name = …, user.email = …). ID and created\_at/updated\_at are automatically created. You will need to user.save() in order to add it to the database.

User.objects.all() will reference all users.

B = User.objects.get(id=1) or (name=’mike’) or whatever identifying term you want.

**Validations**

In order to runs some logic to test passwords, you may want to add a Manager class to handle flash messages.

# Inside your app's models.py file

from \_\_future\_\_ import unicode\_literals

from django.db import models

#Our new manager!

#No methods in our new manager should ever catch the whole request object with a parameter!!! (just parts, like request.POST)

class BlogManager(models.Manager):

*def* basic\_validator(self, postData):

errors = {}

if len(postData['name']) < 5:

errors["name"] = "Blog name should be more than 5 characters"

if len(postData['desc']) < 10:

errors["desc"] = "Blog desc should be more than 10 characters"

return errors

class Blog(models.Model):

name = models.CharField(max\_length=255)

desc = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Connect an instance of BlogManager to our Blog model overwriting

# the old hidden objects key with a new one with extra properties!!!

objects = BlogManager()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In views:

# Inside your app's views.py file

from django.shortcuts import render, HttpResponse, redirect

from .models import Blog

*def* update(request):

errors = Blog.objects.basic\_validator(request.POST)

if len(errors):

for tag, error in errors.iteritems():

messages.error(request, error, extra\_tags=tag)

return redirect('/blog/edit/'+id)

else:

blog = Blog.objects.get(id = id)

blog.name = request.POST['name']

blog.desc = request.POST['desc']

blog.save()

return redirect('/blogs')

Any questions, refer to the docs!