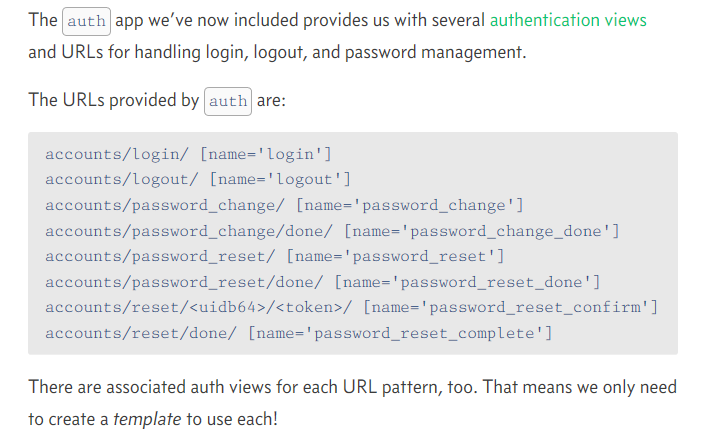
|  |  |  |
| --- | --- | --- |
| Check your django version | Python –m Django --version | Should be 2.1 or higher. |
| Create a new project | Django-admin startproject Django\_project |  |
| Create a new app | Python manage.py startapp blog |  |
| Edit the app’s view.py file | Add the following:  from django.shortcuts import render  from django.http import HttpResponse  def home(request):  return HttpResponse('<h1>Blog Home</h1>')  within your app directory, create a new file and call it urls.py  add the following to the recently created urls.py:  from django.urls import path  from . import views  urlpatterns = [ path('',views.home, name='blog-home'),]  add the following to your project’s urls.py, add:  from django.urls import include  and include the following line to the urlpattern list:  urlpatterns = [  path('admin/', admin.site.urls),  path('blog/',include('blog.urls')),  ] |  |
| Define more views – an About view | Go to views.py  Add the following:  def about(request):  return HttpResponse('<h1>Blog About</h1>')  Go to your apps’ urls.py and add the following highlighted line  urlpatterns = [  path('',views.home, name='blog-home'),  path('about/',views.about, name='blog-about'),  ] |  |
| (Optional) | On project’s urls.py:  urlpatterns = [  path('admin/', admin.site.urls),  path('',include('blog.urls')),  ] |  |
| Creating templates | Create a templates directory within your app directory.  Within the templates directory, create a new directory with your app name.  Appname >> templates >> Appname  This is where our html templates will be stored  Within that path, create two html files: home.html and about.html  home.html:  <!DOCTYPE html>  <html>  <head>  <title></title>  </head>  <body>  <h1>Blog Home</h1>  </body>  </html>  about.html: | Inside html file, if you typed html followed by tab, sublimeText will put a simple structuring of html file for you. |
| Adding your app to the list of installed apps | Check apps.py within your app directory.  Copy the name of the class there.  Open settings.py file  Scroll till you find the list of INSTALLED\_APPS.  Add the following highlighted line:  INSTALLED\_APPS = [  'blog.apps.BlogConfig',  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  ] |  |
| Tell the view which template to load | Edit the views in views.py to look like the following:  def home(request):  return render(request,'blog/home.html')  Do the same thing to the about view. Replace return HttpResponse with return render.  def about(request):  return render(request, 'blog/about.html')  [you no longer need to import HttpResponse up there] | The third argument to pass to the render method is the context. Context (which is optional) is a way to pass information to the template.  Context is a Python dictionary. |
| Try passing some data from the view into the template | Add the following to the top of your view:  posts = [{  'author':'Tagwa',  'title': 'title #1',  'content' : 'content ####1',  'date\_posted' : '26/5/2019'  },  {  'author':'AdminT',  'title': 'title #2',  'content' : 'content ####2',  'date\_posted' : '27/5/2019'    }]  And update your home view:  def home(request):  context = {'posts':posts}  return render(request,'blog/home.html', context)  Go to your home.html template, update it to look like the following:  <!DOCTYPE html>  <html>  <head>  <title></title>  </head>  <body>  {% for post in posts %}  <h1>{{ post.title}}</h1>  <p>By {{ post.author}} on {{post.date\_posted}}</p>  <p>{{ post.content}}</p>  {% endfor %}  </body>  </html> | Double curly braces are templates way of accessing variables defined in the view  The post variable we are looping over is the context’s dictionary key |
| Playing more with passing variables from views into templates  [showing a page title] | On the view:  def about(request):  return render(request, 'blog/about.html', {'title':'About'})  On the template:  <!DOCTYPE html>  <html>  <head>  {% if title %}  <title>Django Blog - {{ title }}</title>  {% else %}  <title>Django Blog</title>  {% endif %}  </head>  <body>  <h1>The About Page is Here!</h1>  </body>  </html> | Refresh localhost/about on your browser and you will notice the title of the page up there. |
| Templates inheritance | Create a new template and call it base.html and place there all the repeated sections within home.html and about.html.  [the only thing that is not repeated between the two is the body section ]  Base.html will be like the following:  <!DOCTYPE html>  <html>  <head>  {% if title %}  <title>Django Blog - {{ title }}</title>  {% else %}  <title>Django Blog</title>  {% endif %}    </head>  <body>    </body>  </html>  In **home.html**, remove everything that is already in the base.html template. We will be left with this:  {% extends "blog/base.html" %}  {% block content %}  {% for post in posts %}  <h1>{{ post.title}}</h1>  <p>By {{ post.author}} on {{post.date\_posted}}</p>  <p>{{ post.content}}</p>  {% endfor %}  {% endblock content%}  And use the **base.html** template to extend it:  <!DOCTYPE html>  <html>  <head>  {% if title %}  <title>Django Blog - {{ title }}</title>  {% else %}  <title>Django Blog</title>  {% endif %}    </head>  <body>  {% block content %}{% endblock %}  </body>  </html>  The new **about.html** template:  {% extends "blog/base.html" %}  {% block content %}  <h1>The About Page is Here!</h1>  {% endblock content%} | Both templates have partially the same code.  The more pages we have with repeated code, the worse that problem would be. |
| Bootstrap starter template | Copy from starter template  Paste in base.html  Grab the jaavscript tags from starter template, and paste it into the body right before its end.  Place the content block into a div tag | Link:  <https://getbootstrap.com/docs/4.0/getting-started/introduction/#starter-template> |
| Adding a navigation bar + a main section for content | Use bootstrap.  [an open source toolkit for developing with HTML, CSS, and JS] |  |
| User registration | 1. Create an app, call it **users** 2. Create a template for users, call it **register.html** 3. Edit the **views.py**, and define the method for user registration form. 4. Leave the **models.py** untouched. You don’t need to edit it. 5. Create a file within your app directory. Call it **forms.py**   from django import forms  from django.contrib.auth.models import User  from django.contrib.auth.forms import UserCreationForm  class UserRegisterForm(UserCreationForm):  email = forms.EmailField()  class Meta:  model = User  fields = ['username', 'email', 'password1', 'password2']   1. Registering your app in your project’s settings.py file.   INSTALLED\_APPS = [ 'blog.apps.BlogConfig', 'users.apps.UsersConfig',  'crispy\_forms',  'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles',] | \*\*\*crispy forms was mentioned once in the **register.html** template. |

|  |  |  |
| --- | --- | --- |
| Login - logout authentication | 1. Adding the following to the user’s **views.py:**   @login\_required def profile(request): return render(request, 'users/profile.html')   1. Edit the models.py file, and include the following   from django.db import models  from django.contrib.auth.models import User  class Profile(models.Model):  user = models.OneToOneField(User, on\_delete=models.CASCADE)  image = models.ImageField(default='default.jpg', upload\_to='profile\_pics')    def \_\_str\_\_(self):  return f'{self.user.username} Profile'   1. Register the Profile class in models.py within the admin.py so that you are able to access it through Admin’s GUI   from django.contrib import admin from .models import Profile admin.site.register(Profile)   1. Create a file within the apps directory, and call it **signals.py.**   Add the following to signals.py:  from django.db.models.signals import post\_save  from django.contrib.auth.models import User  from django.dispatch import receiver  from .models import Profile  @receiver(post\_save, sender=User)  def create\_profile(sender, instance, created, \*\*kwargs):  if created:  Profile.objects.create(user=instance)  @receiver(post\_save, sender=User)  def save\_profile(sender, instance, \*\*kwargs):  instance.profile.save()   1. Three more templates need to be added: **login.html**, **logout.html**, and **profile.html** 2. Editing the models.py:   from django.db import models  from django.contrib.auth.models import User  class Profile(models.Model):  user = models.OneToOneField(User, on\_delete=models.CASCADE)  image = models.ImageField(default='default.jpg', upload\_to='profile\_pics')    def \_\_str\_\_(self):  return f'{self.user.username} Profile' |  |
|  |  |  |
|  |  |  |

Login-logout user authentication: <https://wsvincent.com/django-user-authentication-tutorial-login-and-logout/>

Another login-logout tutorial: <https://www.techiediaries.com/django-authentication/>

Simple file uploader – stackoverflow: <https://stackoverflow.com/questions/5871730/need-a-minimal-django-file-upload-example>



File uploaders:

