## 

Table of Contents

[Activating the Virtual Environment 1](#_Toc128726679)

[Adding Bootstrap 1](#_Toc128726680)

[Registration 7](#_Toc128726681)

[Creating a Login/Logout Page 11](#_Toc128726682)

[Hiding or Showing Content in Views 12](#_Toc128726683)

[Restricting Access Based on Authentication 13](#_Toc128726684)

## Activating the Virtual Environment

The following examples require you to be in an active virtual environment.

On Mac

source mypython/bin/activate

On Windows (this is the path the Activate.ps1)

mypython\Scripts\Activate

When on PythonAnywhere the following command will activate the virtual environment:

workon mysite-virtualenv

To deactivate your environment type the following in the PyCharm terminal:

**deactivate**

## Adding Bootstrap

Bootstrap is a style framework. For non-designers like me, it offers pre-built style templates and nice looking controls without almost any work.

This tutorial uses version 4.0 though I am going to use documentation here:

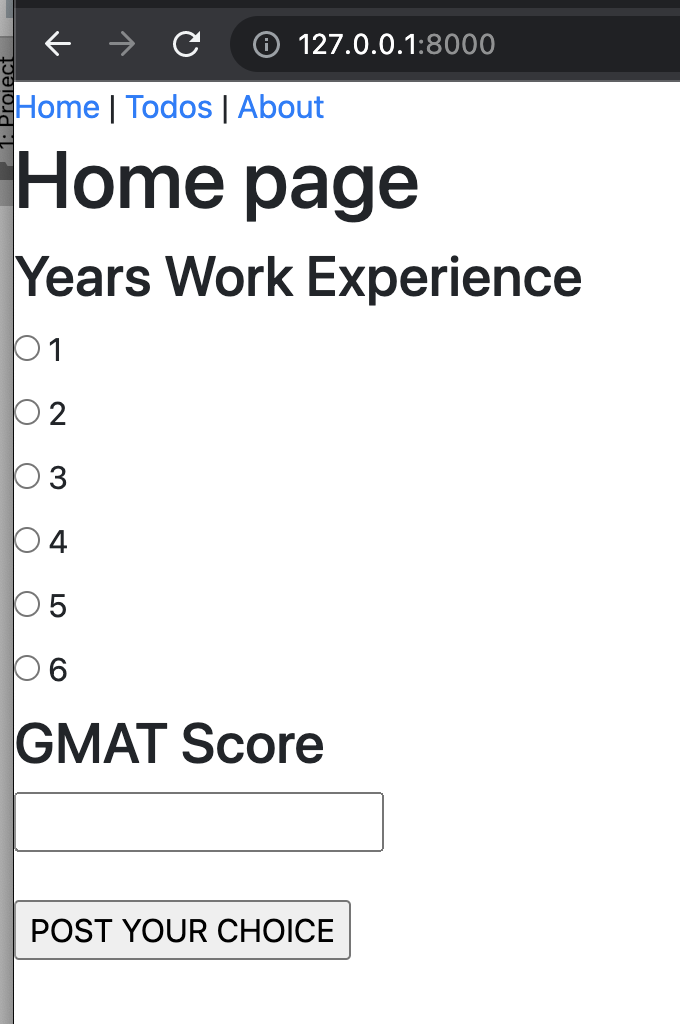
<https://getbootstrap.com/docs/4.6/getting-started/introduction/>

Example 1: Adding Styling to the Django Site

To add Bootstrap, replace the code in **templates\base.html** with the following.

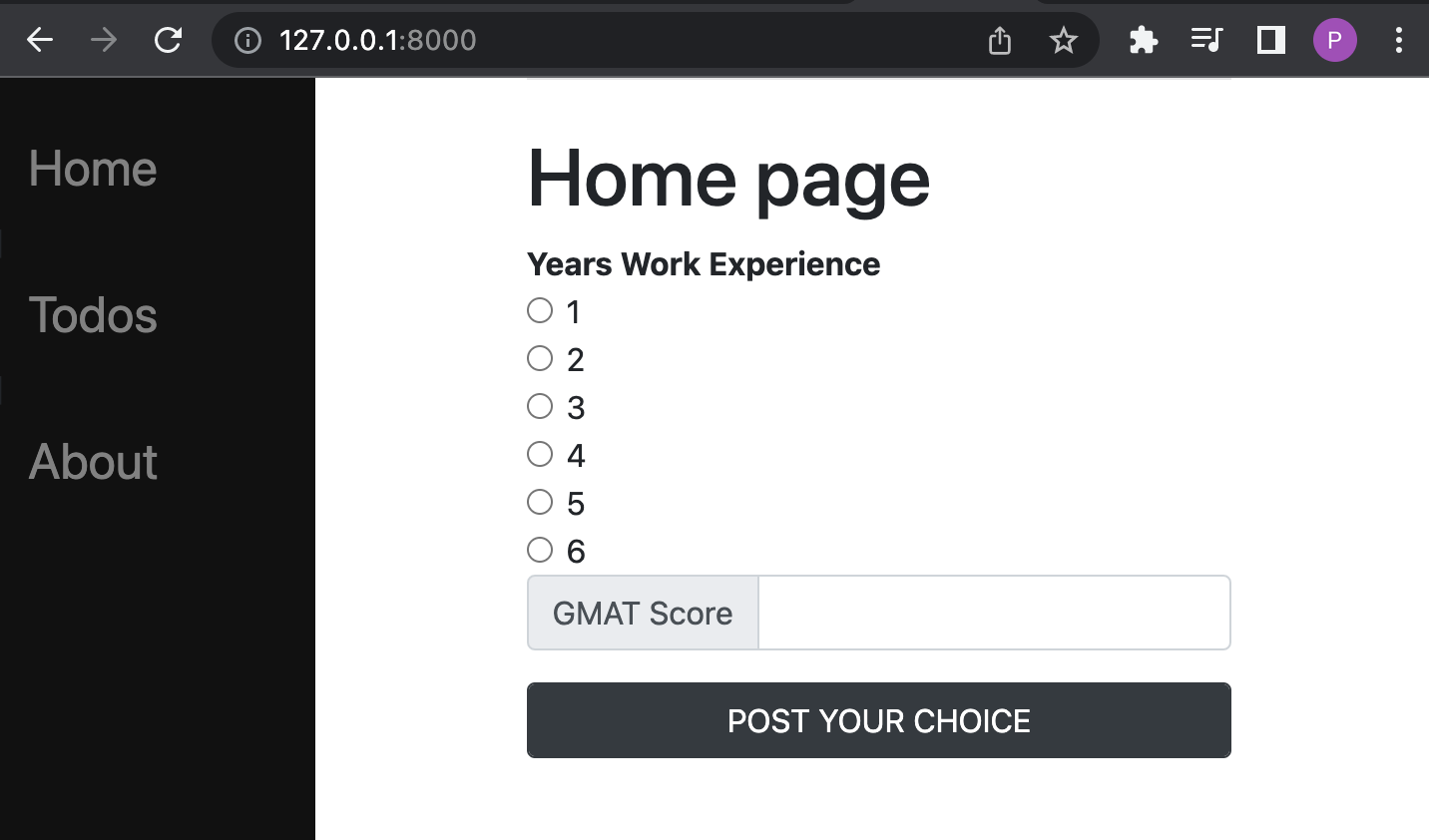
|  |
| --- |
| <!doctype html>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">  <!-- templates/base.html -->  <a href="{% url 'home' %}">Home</a> |  <a href="{% url 'todos' %}">Todos</a> |  <a href="{% url 'about' %}">About</a>  {% block content %} {% endblock content %}  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script> |

The presentation is still not great but the basic styling is in place for further improvement.



Example 2: Improving the Styling

This example improves the look of the site by using the styling provided by Tim at <https://www.techwithtim.net/tutorials/django/adding-bootstrap/>

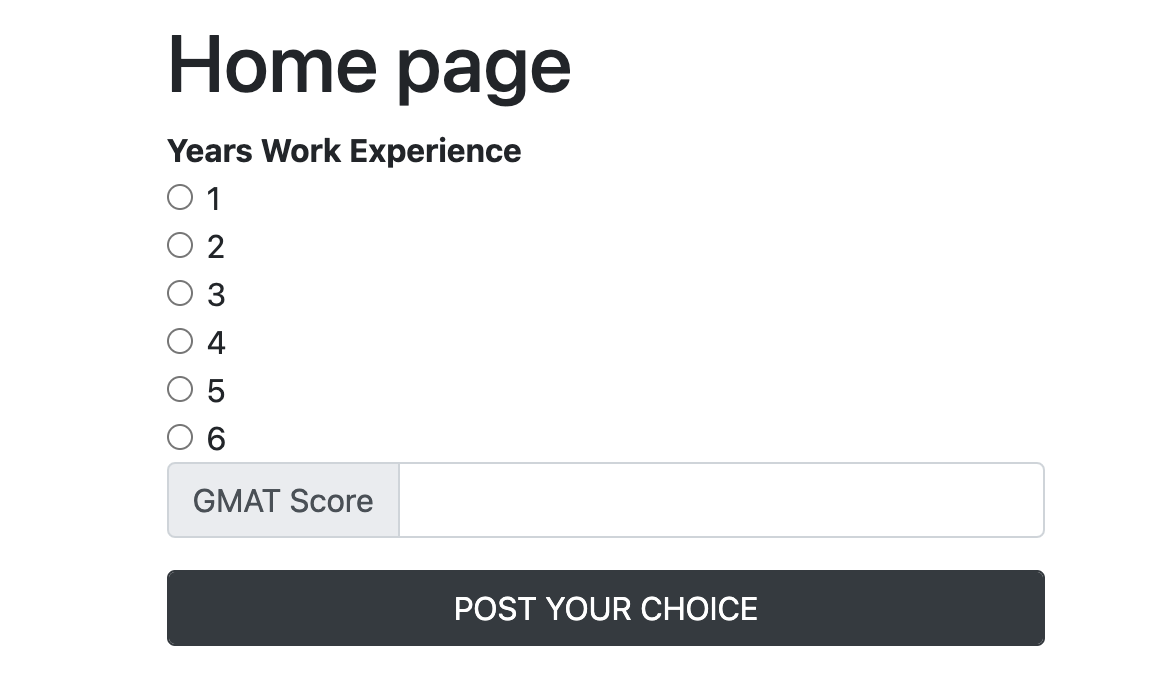


To add the side bar, replace the code in templates/base.html with the following code and change the title of the page to your name. See the green highlight.

|  |
| --- |
| <!-- base.html -->  <!doctype html>  <html>  <head>  <style type="text/css">  .sidenav {  height:100%;  width:160px;  position: fixed;  z-index:1;  top:0;  left:0;  background-color:#111;  overflow-x: :hidden;  padding-top:20px;  }  .sidenav a {  padding:6px 8px 6px 16px;  text-decoration: none;  font-size:25px;  color: #818181;  display:block;  }  .sidenav a:hover{  color:#f1f1f1;  }  .main{  margin-left:160px;  padding: 0px 10px;  }  </style>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">  <title>{% block title %}Pat's Site{% endblock %}</title>  </head>  <body>  <div class="sidenav">  <a href="{% url 'home' %}">Home</a> |  <a href="{% url 'todos' %}">Todos</a> |  <a href="{% url 'about' %}">About</a>  </div>  <div id="content" name="content" class="main">  <div class="row justify-content-center">  <div class="col-8">  <hr class="mt-0 mb-4">  {% block content %}  {% endblock %}  </div>  </div>  </div>  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>  </body>  </html> |

Example 3: Fixing the Controls on the Home Page

I am not a pro with Bootstrap but it is pretty easy to figure out. These are some quick modifications that I made to improve the appearance of the home page. This is what I ended up with:



Using the button documentation at

<https://getbootstrap.com/docs/4.6/components/buttons/>

Inside **home.html**, I replaced the existing button with this one:

|  |
| --- |
| <button type="submit" class="btn btn-dark">POST YOUR CHOICE</button> |

Next, I want to this page to figure out how to improve the appearance of the text input

<https://getbootstrap.com/docs/4.6/components/input-group/>

Note that I had to maintain the name property that is highlighted in yellow to ensure the data can be read when the form is submitted to the action function.

|  |
| --- |
| <div class="input-group mb-3">  <div class="input-group-prepend">  <span class="input-group-text" id="inputGroup-sizing-default">GMAT Score</span>  </div>  <input type="text" name="gmat" class="form-control" aria-label="Sizing example input" aria-describedby="inputGroup-sizing-default">  </div> |

This looked pretty good so I later decided to put the button inside the div tag with hopes that the size would be okay and thankfully that worked.

|  |
| --- |
| <div class="input-group mb-3">  <button type="submit" class="btn btn-dark form-control">POST YOUR CHOICE</button>  </div> |

Next, I found this page to find information about styling the radio buttons.

<https://getbootstrap.com/docs/4.6/components/forms/#checkboxes-and-radios>

The yellow highlights are modifications that I made to ensure the labelling and references would work with the action functions in views.py.

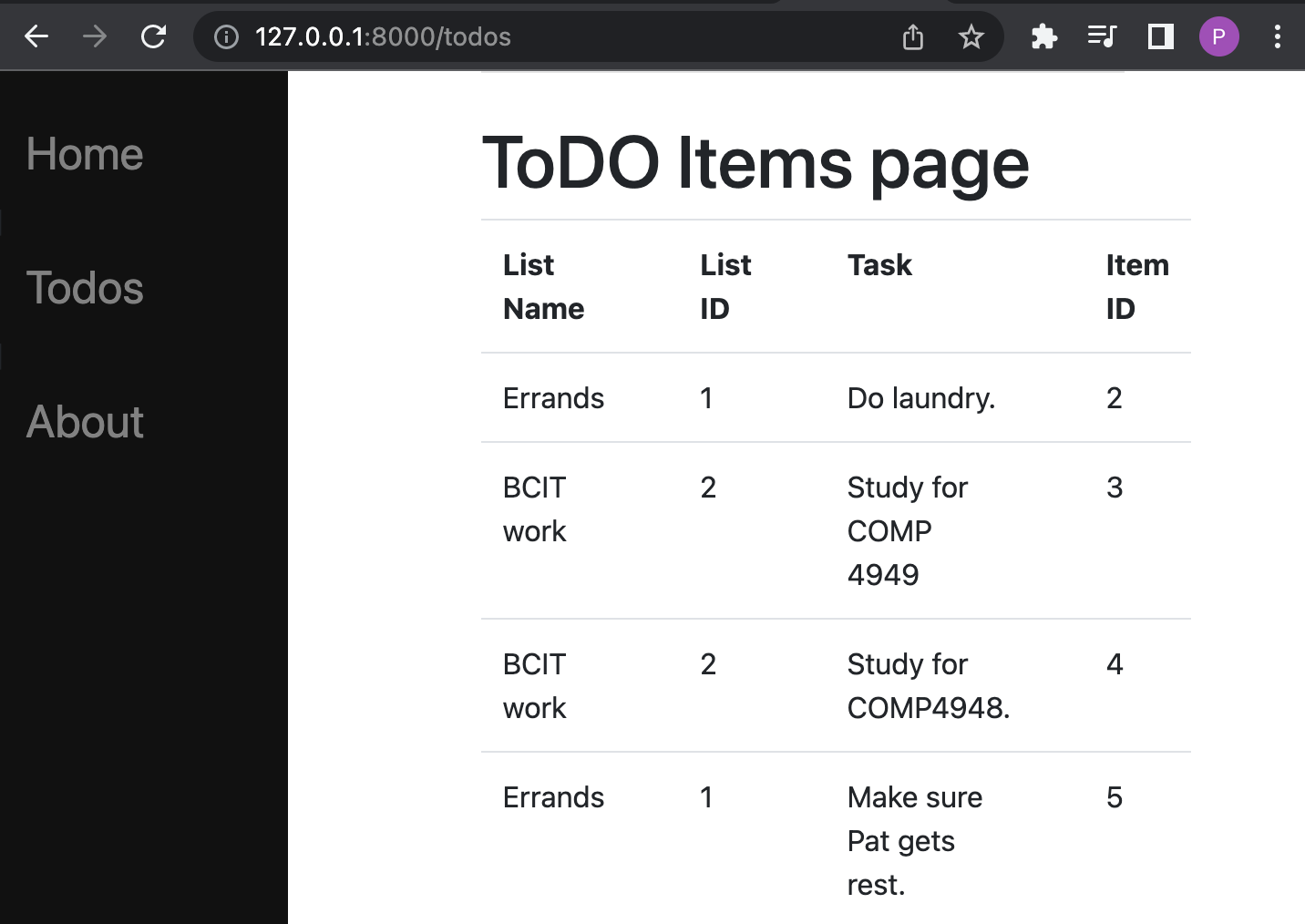
Here is what I came up with for **templates/home.htm**l.

|  |
| --- |
| <div class="form-check">  <input class="form-check-input" type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice }}">  <label class="form-check-label" for="choice{{ forloop.counter }}">  {{ choice }}  </label>  </div> |

This looks okay but could use more work. I think I will leave it for now.

Example 4: Fixing the Appearance of the Table

I made quick adjustment to fix the table on the todos page.



On the ToDoItems.html page I added class=”table” and scope=”col” to improve the appearance.

<https://getbootstrap.com/docs/4.6/content/tables/>

There are actually mobile responsive grids that could be used instead but due to time I left it. Still the functionality for easily implementing mobile responsive tables is available if you need it.

**templates/todos.html**

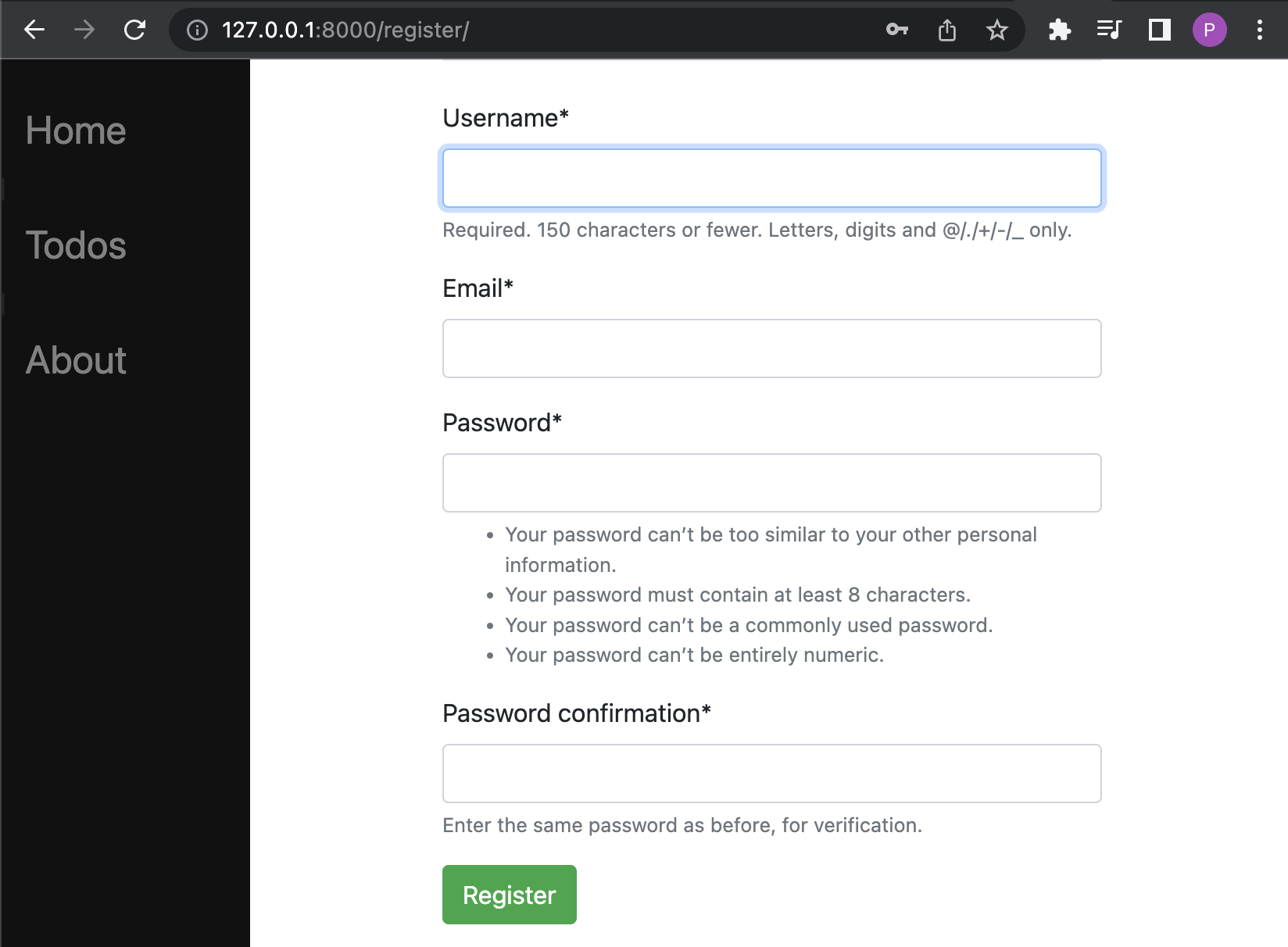
|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <h1>ToDO Items page</h1>  <table class="table">  <!-- table headers -->  <tr>  <th scope="col">List Name</th>  <!-- empty column for spacing-->  <th scope="col">&nbsp;</th>  <th scope="col">List ID</th>  <th scope="col">&nbsp;</th>  <th scope="col">Task</th>  <th scope="col">&nbsp;</th>  <th scope="col">Item ID</th>  </tr>  {% for todo in ToDoItemDetail %}  <!-- rows of data -->  <tr>  <!-- cells -->  <td scope="col">{{todo.todolist.name}}</td>  <!-- empty column for spacing-->  <td scope="col">&nbsp;</td>  <td scope="col">{{todo.todolist.id}}</td>  <td>&nbsp;</td>  <td scope="col">{{todo.text}}</td>  <td>&nbsp;</td>  <td scope="col">{{todo.id}}</td>  </tr>  {% endfor %}  </table>  {% endblock content %} |

## Registration

Example 5: Creating a Registration Form

This section is based heavily on the tutorial at <https://www.techwithtim.net/tutorials/django/user-registration/>

In this example you will build a registration form.



To enable nicely styled forms run these installations.

pip install django-crispy-forms

pip install crispy\_bootstrap4

The installations are light one so they will not use up much of your space quota remotely if you install it PythonAnywhere.)

Then add references to the local settings.py file.

**helloworld/config/settings.py**

|  |
| --- |
| INSTALLED\_APPS = [  "django.contrib.admin",  "django.contrib.auth",  "django.contrib.contenttypes",  "django.contrib.sessions",  "django.contrib.messages",  "django.contrib.staticfiles",  'crispy\_forms',  'crispy\_bootstrap4',  "pages.apps.PagesConfig", # new  ] |

Also, for styling, add this reference anywhere in **helloworld/config/settings.py**

|  |
| --- |
| CRISPY\_TEMPLATE\_PACK="bootstrap4" |

In the pages folder create a forms.py file add this content to enable the automated generation registration of a form which enforces validation for the form inputs. This form can be extended but for simplicity it will be left as is.

**pages/forms.py**

|  |
| --- |
| from django import forms  from django.contrib.auth.forms import UserCreationForm  from django.contrib.auth.models import User  class RegisterForm(UserCreationForm):  email = forms.EmailField()  class Meta:  model = User  fields = ["username", "email", "password1", "password2"] |

Now create a registration folder and a register.html file inside it. The set up will matter so the spelling and location are essential since some of the automation template generation will require it later.

**templates/registration/register.html**

|  |
| --- |
| {% extends "base.html" %}  {% block title %}Create an Account{% endblock %}  {% load crispy\_forms\_tags %}  {% block content %}  <form method="POST" class="form-group" >  {% csrf\_token %}  {{ form|crispy }}  <button type="submit" class="btn btn-success">Register</button>  </form>  {% endblock %} |

Then add this code to views.py for the registration view. If a GET request is made the empty registration form is displayed. When a POST request is made the register() function will receive the data and save it if the data is valid. If the data is invalid the registration form is displayed with error messages. The error messages were enabled by the automation template used earlier in forms.py.

**pages/views.py**

|  |
| --- |
| from django.shortcuts import render, redirect  from .forms import RegisterForm  def register(response):  # Handle POST request.  if response.method == "POST":  form = RegisterForm(response.POST)  if form.is\_valid():  form.save()  return redirect("../") # Go to home page  # Handle GET request.  else:  form = RegisterForm()  return render(response, "registration/register.html", {"form":form}) |

Next, import the register view and add the register route to the registration form can be dispatched if requested. See the highlights.

**pages/urls.py**

|  |
| --- |
| # pages/urls.py  from django.urls import path  from .views import homePageView, aboutPageView, results, homePost, todos, register  urlpatterns = [  …  path("register/", register, name="register"), # <-- added  ] |

Example 6: Adding a Registration Confirmation

When trying this out you may notice that you are redirected to the home page after registering but there is no confirmation. This example will add a multi-purpose message page for informing users of events such as confirming their registration.



First add the html template.

**templates/message.html**

|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <br/><br/><br/><br/>  <h3>{{title}}</h3>  {{msg}}  {% endblock content %} |

Then, in pages/views.py add the action function to **pages/views.py**

|  |
| --- |
| def message(request, msg, title):  return render(request, 'message.html', {'msg': msg, 'title': title }) |

Next, the router needs to know about this message action function so import the message reference.

**pages/urls.py**

|  |
| --- |
| from .views import homePageView, aboutPageView, results, homePost, todos, register, message |

Then add the route with the *message* and *title* parameters included.

**pages/urls.py**

|  |
| --- |
| path('message/<str:**msg**>/<str:**title**>/', message, name="message"), # <-- added |

Now a successful registration can inform users when their registration is successful. To enable this adjustment, in **pages/views.py** replace the redirect for a successful registration from:

|  |
| --- |
| return redirect("../home/") # Go to home page |

To

|  |
| --- |
| return HttpResponseRedirect(reverse('message',  kwargs={'msg': "Your are registered.", 'title': "Success!"}, )) |

When registering a new user, the message page should now appear to confirm a successful registration.

## Creating a Login/Logout Page

<https://www.techwithtim.net/tutorials/django/login-logout/>

The process of adding a login and logout page is even more automated than the registration form.

Example 7: Creating a Login/Logout Page

Django comes with a few built in applications that automatically generate specific user authentication forms like a login and logout form. To enable these forms, add the following code to the **pages/urls.py** file.

|  |
| --- |
| # pages/urls.py  from django.urls import path, include  from .views import homePageView, aboutPageView, results, homePost, todos, register  urlpatterns = [  …  path('', include("django.contrib.auth.urls")), # <-- added  ] |

Then add the login view here **/templates/registration/login.html.** (Be sure to place it in the registration folder)

|  |
| --- |
| # login.html  {% extends "../base.html" %}  {% block title %}  Login Here  {% endblock %}  {% load crispy\_forms\_tags %}  {% block content %}  <form method="post" class="from-group">  {% csrf\_token %}  {{form|crispy}}  <p>Don't have an accoount? Create one <a href="../register/">here</a></p>  <button type="submit" class="btn btn-success">Login</button>  </form>  {% endblock %} |

When this addition has been made you can then navigate to login and logout:

<http://localhost:8000/login>

<http://localhost:8000/logout>

For ease of access, add the following links to templates/base.html

|  |
| --- |
| <a href="{% url 'login' %}">Login</a>  <a href="{% url 'logout' %}">Logout</a> |

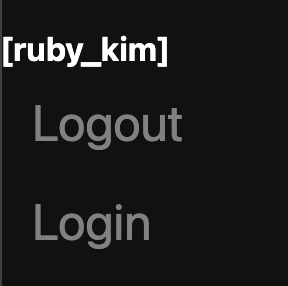
Also, be sure to add a login / logout redirect. To do this, add this code to anywhere in **settings.py**

|  |
| --- |
| LOGIN\_REDIRECT\_URL = "/"  LOGOUT\_REDIRECT\_URL = "/" |

## Hiding or Showing Content in Views

Django has several features to detect if a user is authenticated and to hide/show or redirect users depending on their authentication status.

Example 8: Showing Content if a User is Authenticated



|  |
| --- |
| {% if user.is\_authenticated %}  <b style="color:white;">[{{ user.username }}]</b>  <a href="{% url 'logout' %}">Logout</a>  {% endif %} |

Exercise 1 (4 marks)

Use Bootstrap to format the logged in user name more nicely. Show your code here:

|  |
| --- |
| <!-- base.html --> <!doctype html> <html>  <head>  <style type="text/css">  .sidenav {  height: 100%;  width: 160px;  position: fixed;  z-index: 1;  top: 0;  left: 0;  background-color: #111;  overflow-x:hidden;  padding-top: 20px;  }   .sidenav a, #username {  padding: 6px 8px 6px 16px;  text-decoration: none;  font-size: 25px;  color: #833737;  display: block;  }   .sidenav a:hover {  color: #f1f1f1;  }   .main {  margin-left: 160px;  padding: 0px 10px;  }  </style>   <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"  integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">   <title>{% block title %}Victor's Site{% endblock %}</title> </head>  <body>  <div class="sidenav">  <a href="{% url 'home' %}">Home</a> |  <a href="{% url 'todos' %}">Todos</a> |  <a href="{% url 'about' %}">About</a>  <a href="{% url 'login' %}">Login</a>  {% if user.is\_authenticated %}  <b id="username">{{ user.username }}</b>   <a href="{% url 'logout' %}">Logout</a>  {% endif %}  </div>   <div id="content" name="content" class="main">  <div class="row justify-content-center">  <div class="col-8">  <hr class="mt-0 mb-4">  {% block content %}  {% endblock %}  </div>  </div>  </div>   <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"  integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"  crossorigin="anonymous"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"  integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"  crossorigin="anonymous"></script>  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"  integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"  crossorigin="anonymous"></script> </body>  </html> |

Show a screenshot of your logged in user name with your name in it.

|  |
| --- |
|  |

Exercise 2 (3 marks)

Show the code needed to hide the Login link when logged in and to show the Login when logged out.

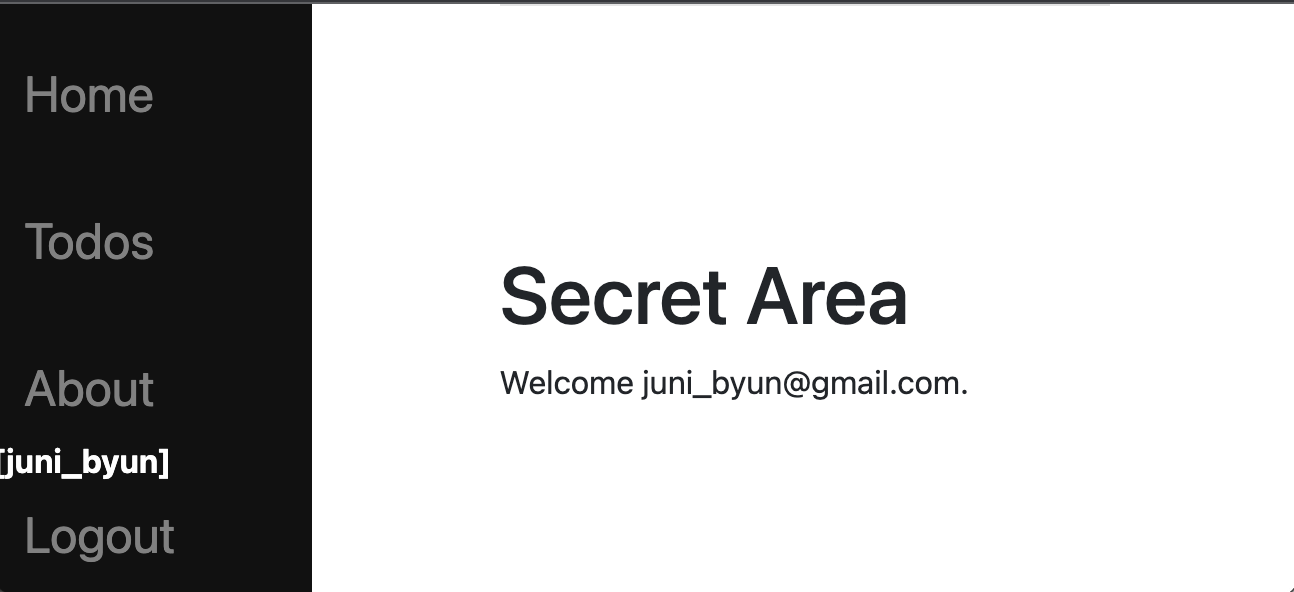
|  |
| --- |
| <div class="sidenav">  <a href="{% url 'home' %}">Home</a> |  <a href="{% url 'todos' %}">Todos</a> |  <a href="{% url 'about' %}">About</a>  <a href="{% url 'login' %}">Login</a>  {% if user.is\_authenticated %}  <b id="username">{{ user.username }}</b>  <a href="{% url 'logout' %}">Logout</a>  {% endif %} </div> |

## Restricting Access Based on Authentication

Django has several features to detect if a user is authenticated in Python code and to redirect users depending on their authentication status.

Example 9: Detecting if a User is Authenticated and Redirecting if Not

This example only allows users to access the secret area if they are authenticated.



First create a template that receives useremail from the action function.

**pages/secret.html**

|  |
| --- |
| <!-- templates/base.html -->  {% extends "base.html" %}  {% block content %}  <br/><br/><br/><br/>  <h1>Secret Area</h1>  Welcome {{ useremail }}.  {% endblock content %} |

Then in **pages/views.py** add a secretArea action to check a user’s authentication status and redirect accordingly. In this case, a user-friendly response advising the user to login if provided if they are not logged in.

|  |
| --- |
| def secretArea(request):  if not request.user.is\_authenticated:  return HttpResponseRedirect(reverse('message',  kwargs={'msg': "Please login to access this page.",  'title': "Login required."}, ))  return render(request, 'secret.html', {'useremail': request.user.email }) |

The router must have the reference to the secretArea and a path so the dispatcher can find the appropriate action function.

|  |
| --- |
| from .views import homePageView, aboutPageView, results, homePost, todos, register, message, secretArea  …  path("secret/", secretArea, name="secret"), |

Exercise 3 (3 marks)

Add a hyperlink to the secret area. Only make it visible when the user is logged in. Show a screenshot of the link and the secret area when you are logged in with an account **that has your name on it**.

|  |
| --- |
|  |

Show the code for your hyperlink that is included in the pages/base.html file.

|  |
| --- |
| <div class="sidenav">  <a href="{% url 'home' %}">Home</a> |  <a href="{% url 'todos' %}">Todos</a> |  <a href="{% url 'about' %}">About</a>  <a href="{% url 'login' %}">Login</a>  {% if user.is\_authenticated %}  <b id="username">{{ user.username }}</b>  <a href="{% url 'secret' %}">Secret</a>  <a href="{% url 'logout' %}">Logout</a>  {% endif %} </div> |