**Python Django: Ultimate Beginner’s Course - 2022**

<https://www.udemy.com/course/python-django-ultimate-beginners-course-2022/learn/lecture/34354180#overview>

**Project Requirements**:

* How to user X/Y/Z (Django, Shell/CMD, Python)
  + Include what you specifically did/used
* Versions of project for each step
  + Include: notes, comments in code, screenshot/screen recording
  + Include screenshot/video of webpage/result
* Share content before presenting
  + Allows audience to create along or reference as you present

**Highlight Legend**

Red - ERROR information or NOTE

Orange - filename or directory information

Green - CMD commands

Blue - IDE/file code

SuperUser info:

Username: aleecia

Password: projectpassword123

SecondUser info:

Username: aleecia2

Email: test@gmail.com

Password: testpassword123

**October 13 2022**

**Django**

* High-level Python based web framework
* back-end/server-side framework (Eg. Ruby on Rails - Ruby based, Laravel - PHP based, etc.)
* Allows for rapid development (extremely fast and scalable, secure)
* “Batteries-included” framework
  + Built in authentication, functionality, etc.
* IRL : Youtube, Instagram, Spotify
* Default database included in Django projects
  + db.sqlite3

**Python**

* Check if it’s already installed
  + In CMD, type “py --version”
* If not already installed
  + Install and set up python
  + Install python 3.9.13
* Already have 3.10.5 installed on my PC

**PIP**

* “Python install package”
* Used to install and manage Python packages
* To install “package”
  + In CMD, type “pip install packageName”

**CMD commands**

* cd “directoryName”
  + Change directory to different directory/folder
* mkdir “directoryName”
  + Make a directory/folder
* rmdir “directoryName”
  + Remove directory/folder
* cls ()
  + Clear the CMD

To deactivate virtual environment

* virtualEnvName\Scripts\deactivate

**Django Project Setup**

1. Open CMD, change directory
   * cd “desktop”
   * Changed mine to d drive
     + d:
2. Create new folder “development”
   * Changed mine to PythonProject > DjangoProject
3. cd “development”
4. pip install virtualenv
   * Allows you to create virtual environments
5. pip install django
   * If **ERROR** below happens, open CMD as Administrator
   * “WARNING: Failed to write executable - trying to use .deleteme logic ERROR: Could not install packages due to an OSError: [WinError 2] The system cannot find the file specified: 'C:\\Python310\\Scripts\\sqlformat.exe' -> 'C:\\Python310\\Scripts\\sqlformat.exe.deleteme'”
6. pip list
   * See list of installed python packages
7. Create Django project
   * django-admin startproject Edenthought
     + Edenthought = name of project
8. cd Edenthought
9. Create virtual environment
   * virtualenv venv
     + venv = name of virtual environment
   * If **ERROR** below happens, type the following: python -m virtualenv venv
   * “'virtualenv' is not recognized as an internal or external command, operable program or batch file.”
10. Activate virtual environment that was created
    * venv\Scripts\activate
11. pip list
    * Check which packages are in the virtual environment
12. pip install django
13. Open VS
14. Run server in CMD (in project folder, venv active)
    * python manage.py runserver
15. Verify server is running on port, open port in browser
    * http://127.0.0.1:8000/

**Configure python interpreter**

1. In VS, press ctrl + shift + p
   * Choose python: select interpreter
2. Go to project folder, click on virtual environment folder
   * Copy folder’s path
3. In VS, choose Enter interpreter path…
   * Paste folder’s path, press enter

**ERROR**: VS not accepting path to project’s virtual environment folder

**October 14 2022**

**Django App**

* Django project
  + Composed of a collection of various settings, configurations, and apps
  + Can have one or more app in a Django project
* Django app
  + An app that serves a unique purpose
    - Blog, simplified application
  + Consists of URLs, views, templates, and models

1. Create a Django app (in project folder, venv active)
   1. django-admin startapp journal
      1. journal = name of app
2. Verify it’s been created in VS

admin.py

* Django admin page
* Do administrative tasks used to manage Django web app

apps.py

* Provide info about app

models.py

* Create migrations in this file
* Create model and database table

tests.py

* Create unit tests
* Do unit testing

views.py

* Manage all HTTP requests and responses

**Configuring App**

1. Go to settings.py in project folder (Edentought > Edenthought > settings.py)
2. Go to INSTALLED\_APPS
3. Add the following to the array:
   1. ‘journal.apps.JournalConfig’,
4. Verify app is running correctly
   1. python manage.py runserver

* URL - locate a particular webpage on our browser
* Routes - /pageName
* View - python function/class used to return a particular response
  + Need to pass a request object, to handle requests

Creating a simple webpage

* Import paths and HTTP response
* Create a view

**Ex:**

from django.urls import path

from django.http import HttpResponse

def register(request):

Return HttpResponse(‘This is the registration page’)

urlpatterns = [

path(‘register’, register)

]

* Pass request object
* “This is the registration page” appears on blank HTML page
* urlpatterns - list of all the URLs in Django project
* path = URL path
* ‘register’ = route
* register = view

**Add a URL path and view**

1. In VS code, go to urls.py
2. Add the following import:
   1. from django.http import HttpResponse
3. Type the following below import statements:
   1. def register(request):
      1. return HttpResponse(“Welcome to the registration page!”)
4. Create URL path for new view
   1. Type the following below path(‘admin/’, admin.site.urls),
      1. path(‘register’, register),

**Set up URLs and Views in Django app**

1. In VS code, go to “journal” (project folder)
2. Create new file “urls.py”
3. Copy previously made view from Edenthought > urls.py
   1. from django.http … path(‘register], register),
   2. Paste previously made views in new views.py file
   3. Delete copied code from Edenthought > urls.py
   4. Delete url path path(‘register], register), from Edenthought > urls.py
4. Copy from django.urls import path from Edenthought > urls.py
   1. Paste it to Endenthought > journal > urls.py
5. Type the following in urls.py:
   1. from . import views
      1. Import views from journal’s view.py
      2. . = in same directory
6. Copy urlpatterns from Edenthought > urls.py
   1. Paste it to Endenthought > journal > urls.py
   2. Change it to:
      1. path(‘register’, views.register),
7. Import include function from django.urls to Edenthought > urls.py
   1. from django.urls import path, include
8. Import journal.urls to Edenthought > urls.py
   1. path(‘’, include(‘journal.urls’))
9. Add path for ‘/’ URL
   1. Add following code to Endenthought > journal > views.py
      1. Def home(request):
         1. return HttpRequest(“This is the Homepage!”)
   2. Add the following code to Endenthought > journal > urls.py
      1. path(‘’, views.home),

**Add URLs and Views to Django Project**

1. In VS code, go to “journal” (app folder)
2. Create new file “urls.py”
3. Add the following import to journal > views.py:
   1. from django.http import HttpResponse
4. Add the following views to journal > views.py:
   1. def register(request):
      1. return HttpResponse(“Welcome to the registration page!”)
   2. def home(request):
      1. return HttpResponse(“This is the homepage!”)

* Pass request object in view function

1. Add the following import to journal > urls.py:
   1. from django.http import HttpResponse
   2. from . import views

* Import views from journal > view.py
* . = within same directory

1. Add the following urlpattern to journal > urls.py:
   1. urlpatterns = [
      1. path(‘’, views.home),
      2. path(‘register’, views.register),
   2. ]

* path = URL path
* ‘’ / ‘register’ = route
* home / register = view

1. Add the following import to Edenthought (project folder) > urls.py
   1. from django.urls import path, include
2. Add the following urlpatterns to Edenthought (project folder) > urls.py
   1. urlpatterns = [
      1. path(‘admin/’, admin.site.urls),
      2. path(‘’, include(‘journal.urls’))
   2. ]

* urlpatterns - list of all the URLs in Django project
* Includes paths in “journal” (app) urls.py file

**October 15 2022**

**NOTE**: From this point forward, urls.py and views.py are the ones in Endenthought > journal

**Templates**

* layout/structure of particular file
* May include simple HTML file

**Configure Django to Render Templates**

1. In VS Code, go to project directory Edenthought
   1. First folder with projectName
2. Create new folder “templates”
3. Go to settings.py
   1. Import os module
      1. import os
   2. Join base directory to templates folder
      1. In ‘DIRS’ array, add the following:
         1. os.path.join(BASE\_DIR, ‘templates’)

* Tells Django where to find templates

**Render Template**

* Create new HTML file “register.html ”in templates folder
* Add HTML to file
  + <h1>Registration Page!</h1>
  + <p>This is where you register</p>
* Replace view’s HTTP response with HTML template
  + Delete the following from Endenthought > journal > views.py
  + return HttpResponse(“Welcome to the Registration Page!”)
  + Replace it with the following:
    - return render(request, ‘register.html’)

**Template Inheritance**

* Create a base template to re-use in other templates
* Header, footer, nav bar, etc.

**Create base template**

1. Create new HTML file “base.html” in templates folder
2. Add HTML to file
   1. <h1>This website is under development<h1>
   2. <hr>
3. Create new HTML file “index.html” in templates folder
4. Replace view’s HTTP response with HTML template
   1. Delete the following from Endenthought > journal > views.py
   2. return HttpResponse(“This is the homepage!”)
   3. Replace it with the following:
      1. return render(request, ‘index.html’)
5. Register base template to HTML templates
   1. Add the following code to both register.html and index.html
   2. {% include ‘base.html’ %}

**Rendering Data into a Template**

* DTL - Django Template Language
  + Mini-language
  + Write logic in HTML templates
    - Pass variables to templates
    - Create if statements to handle conditions
    - Create for loops to loop through data
* Lists and dictionaries
  + List = array
    - Ex: clientList = [1, 2]
  + Dictionary = key: value pair
    - Ex: { ‘id’ : ‘1’, ‘name’ : ‘John Smith’, ‘profession’ : ‘Web developer’ }
* Tags
  + {% %}
  + Allow writing python-styled logic
  + pre-built tags
    - {% csrf\_token %}
    - {% extends %}
    - {% if %} … {% endif %}

**Context dictionary**

* Key : value
* Defined in views.py
* Pass variables in HTML in {{variable}}
* {‘first\_name : ‘Philip’’}
  + Dictionary in views.py
* My first name is {{first\_name}}
  + HTML in template file
* My first name is Philip
  + HTML shown on webpage

**Pass variables to template**

1. In views.py file
   1. Create variable
      1. first\_name = “John Paul”
   2. Create context dictionary
      1. Conext = {‘name’ : first\_name}
   3. Pass dictionary as third parameter in return render statement
      1. return render(request, ‘index.html’, context)
2. In index.html
   1. Add HTML element with context dictionary’s key
      1. <h1>{{name}}</h1>

**Render Data with If/Endif tags**

* In register.html
  + Remove <h1>{{name}}</h1>
  + Add {% if name == ‘John’ %}
  + <h1>Welcome home, John</h1>
  + {% else %}
  + <h1>Access denied! </h1>
  + {% endif %}

**Render Data with Lookup**

* Dictionary lookup
  + {{dictionary.key}}
* List index lookup
  + {{list.2}}
* Attribute lookup
  + {{object.attribute}}

**Render Data with for/endfor tags**

* Add list of dictionaries named clientList to views.py
* Clean up code
  + Remove context dictionary and its references
  + Remove if/endif logic
  + Remove <p>Welcome</p>
* Add the following code to views.py
  + conext = {‘clients’ : clientList}
  + return render(request, ‘index.html’, context)
* Add the following code to index.html
  + {% for client in clients %}
    - <p>{{client.id}}</p>
    - <br>
    - <p>{{client.name}}</p>
    - <br>
    - <p>{{client.profession}}</p>
    - <br>
  + {{% endfor %}}

Project uses PostgreSQL database, synced with AWS using Amazon RDS (Amazon relational database service)

**Migrate initial migrations**

* python manage.py migrate
  + Migrates all unapplied migrations to database

**Admin panel**

* Admin page of website
* Do basic admin tasks and operations
* Built-in with django
* To login, create super user

**Create superuser**

1. In CMD, server off, type the following:
   1. python manage.py createsuperuser
      1. Type a username (aleecia)
      2. (Optional) type an email
      3. Type a password (projectpassword123)
      4. Type password again (projectpassword123)
         1. **NOTE**: Can’t use common passwords like password123

**Django model**

* Built in features used to create tables along with their fields
  + Database tables

**Ex**:

From django.db import models

class Register(models.Model):

name = models.CharField()

Id = models.UUIDField()

* charField = string or text field
* UUIDField = unique id/number field

**Create a migration**

1. In VS Code, in models.py
   1. Create class Task(models.Model):
      1. title = models.CharField(max\_length=50)
      2. description = models.TextField(max\_lenght=1000)
      3. created = models.DateTimeField(auto\_now\_add=True)

* Task = class name
* Class inherits Model from models
* TextField() = longer text
* Max\_length = maximum length of CharField/TextField
* DateTimeField() = date field
* auto\_now\_add=True = date/time post was posted will be added automatically

1. In CMD type the following:
   1. python manage.py makemigrations

* Prepare migration files/database files to be pushed/migrated to database
* Like Push in GitHub
* Create Django model, prepare it, and organize the structure before it’s sent to the database

1. In CMD type the following:
   1. python manage.py migrate

* Pushes/sends models to database
* Like Commit in GitHub

**Register Django Models in Admin Panel**

* Access database tables in admin panel

1. In VS Code, go to admin.py
2. Import model
   1. from.models import Task
3. Register model
   1. admin.site.register(Task)

**Foreign Key**

* Links two database tables together

**Create a Foreign Key**

1. In VS Code, go to models.py
2. Create ForeignKey Class
   1. Class Review(models.Model):
      1. reviewer\_name = models.CharField(max\_length=65)
      2. review\_title = models.CharFields(max\_length=100)
      3. task = models.ForiegnKey(Task, on\_delete=models.CASCADE)
3. Prepare migration of model to database
   1. python manage.py makemigrations
4. Migrate model to database
   1. python manage.py migrate
5. Import and register the Review model to admin.py
   1. from.models import Task, Review
   2. admin.site.register(Review)
   * ForiegnKey first parameter = model
   * ForiegnKey second parameter = action taken on

**Database Query**

* Action or selection on a database table
* CRUD

**Django ORM (Object Relational Mapping)**

1. Communicate with database without SQL commands
2. Import model that queries will be performed on
3. Select all object from table
   * OR select object based on parameter

**Ex:**

from .models import User

queryDataAll = User.objects.all()

queryDataSingle = User.objects.get(username = ‘john\_678’)

**Set Up Link to task.html from index.html and Add Query**

1. Clean Code
   1. Remove register.html, register in views.py, register in urls.py
   2. Remove clientList dictionary, context, references to context
   3. Remove for/endfor loop from index.html
2. In VS Code
3. Create ”task.html” in templates folder
   1. Copy code from index.html in task.html
4. Add a view and url path for task.html
5. Add 3rd parameter to path(‘task’, views.task, name=”task”)
6. Add link in index.html to task.html
   1. <a href=”{% url ‘task’ %}”> Go to your tasks! </a>
7. Go to views.py
   1. Import model
   2. Add query to def task(request)
      1. queryAllData = Task.objects.all()
   3. Add context dictionary
      1. context = {‘tasks’ : queryAllData}
   4. Add 3rd parameter to return render(request, ‘task.html’, context)
8. Add for/endfor loop to task.html
   1. {{% for task in tasks%}}
      1. <p>{{% task.title %}}</p>
      2. <br>
      3. <p>{{% task.description %}}</p>
      4. <br>
   2. {{% endfor %}}
9. Add dummy data to admin panel

**CRUD**

* Create, read, update, delete

**Model forms**

* Class used to transform Django model into a form
* Like HTML form
* Allows you to send data to database and query data from database

**Ex:**

from django.forms import ModelForm

from . models import Task

class TaskForm(ModelForm):

Class Meta:

Model = Task

fields = ‘\_\_all\_\_’

**Create Django Form**

1. Create file forms.py
2. In forms.py
   1. Import Django Forms and model
   2. Create class TaskForm, it inherits Django ModelForm
   3. Specify class Meta (what model you’re working with + which fields you want to use)

* ‘\_all\_’ = all fields from model

1. In views.py
   1. import TaskForm
   2. create variable to hold TaskForm()
   3. pass form in context dictionary
   4. pass context in return render() statement
2. In HTML template
   1. Render form

**October 16 2022**

**Create a Django Model Form**

1. In VS Code, go to Journal
2. Create new file “forms.py”
   1. Import Django ModelForm and Model
      1. from django.forms import ModelForm
      2. from .models import Task
   2. Create class “TaskForm”
      1. class TaskForm(ModelForm):
         1. class Meta:
            1. model = Task
            2. fields = ‘\_\_all\_\_’
3. Create new file “task-form.html” in templates folder
   1. Copy-paste HTML from index.html to task-form.html
4. Import Model Form to views.py
   1. from .forms import TaskForm
5. Add the following view to views.py
   1. def task\_form(request):
      1. form = TaskForm()
      2. Context = {‘form’ : form}
      3. return render(request, ‘task-form.html’, context)
6. Add the following path to urls.py
   1. path(‘task-form’, views.task\_form, name = ”task-form”)
7. Add link from index.html to task-form.html
   1. <a href = “{% url ‘task-form’ %}”> Go to my model form! </a>
8. Add a form to task-form.html
   1. <form method = “POST” autocomplete = “off” enctype = “multipart/form-data”>
   2. </form>
9. Pass csrf token (Cross-site Request Forgery)
   1. {% csfr\_token %}

* **MUST** be included when creating forms in Django

1. Render data to form
   * 1. {{ form.title }}
     2. {{ form.description }}
2. Add submit button to form
   * 1. <input type =”submit” value =”Submit”>
3. Better format table
   1. Change {{ form.title }} {{ form.description }} to {{ form.as\_p }}

**Django Model Forms - CRUD**

**Create a new record in database using Django Form**

1. Clean up code
   1. Change name of task-form.html to “create-task.html”
   2. In urls.py change path to (‘create-task’, views.create\_task, name = “create-task”)
   3. In views.py, change def task-form to def create\_task … ‘create.html’
   4. In index-html change link href to ‘create-task’ + change text to “Create a task”
2. In views.py
3. Check the HTTP request and validity of form
   1. If form is valid, save it and return to task.html
   2. Add the following below form = TaskForm()
   3. If request.method = ‘POST’:
      1. form = TaskForm(request.POST)
      2. if form.is\_valid():
         1. form.save()
         2. return redirect(‘task’)

* If **ERROR** below happens, import redirect: from django.shortcuts import redirect:
  + redirect is not defined

**Read a record in database using Django Form**

1. Clean up code
   1. Rename task.html to tasks.html
   2. In index.html change link’s href to ‘tasks’
   3. In views.py change def task to def tasks, task.html to tasks.html, and redirect(‘task’) to ‘tasks’
   4. In urls.py change task’ to ‘tasks’, views.task to views.tasks’, and “task” to “tasks”

**To read all models in a database table**

1. In views.py, include the following code
   1. queryData = Task.objects.all()
   2. context = {‘tasks’ : queryData}
   3. return render(request, ‘tasks.html’, context)
2. In tasks.html, include the following code
   1. {% for task in tasks %}
      1. <h2>{{ task.title }}</h2>
      2. <br>
      3. <p>{{ task.description }}</p>
      4. <br>
   2. {% endfor %}

**To read one model in a database table**

1. In views.py, include the following code
   1. queryData = Task.objects.get(id=3)
   2. context = {‘tasks’: queryData}
   3. return render(request, ‘tasks.html’, context)
2. In tasks.html, include the following code
   1. <h2>{{ task.title }}</h2>
   2. <br>
   3. <p>{{ task.description }}</p>
   4. <br>

**Update a record in database using Django Form**

* To update, need to create a dynamic URL
* Ensures correct object is updated

**Updating with Dynamic URL**

1. Create new file “update-task.html” in templates folder
   1. Copy-paste html from create-task.html to update-task.html
2. In views.py, add a view for update-task
   1. def update\_task(request, pk):
      1. task = Task.objects.get(id = pk)
      2. form = TaskForm(instance = task)
      3. if request.method == ‘POST’:
         1. form = TaskForm(request.POST, instance = task)
         2. if form.is\_valid():
            1. form.save()
            2. return redirect(‘tasks’)
         3. context = {‘form’ : form}
         4. return render(request, ‘update-task.html’, context)
3. In urls.py, create new dynamic URL path
   1. path(‘update-task/<str:pk>’, views.update\_task, name = “update-task”)
   2. pk = primary key
4. In tasks.html
   1. Add update button/link to tasks.html
      1. <a href=”{% url‘update-task’ %}”>UPDATE</a>
   2. Pass (task) object’s PK to the button/link
      1. <a href=”{% url‘update-task’ task.id %}”>UPDATE</a>

**Delete a record from database table using Django Form**

1. Create new file “delete-task.html” in templates folder
   1. Copy-Paste html from update-task.html to delete-task.html
   2. Remove {{ form.as\_p }}
   3. Add paragraph element and a link
      1. <p>Are you sure that you want to delete this task?</p>
      2. <a href =”{% url ‘task’ %}”>Return to tasks</a>
   4. Change input value to DELETE this task
2. In views.py, add a view for delete-task
   1. def delete\_task(request, pk):
      1. task = Task.objects.get(id=pk)
      2. if request.method == ‘POST’:
         1. task.delete()
         2. return redirect(‘taskss)
      3. return render(request, ‘delete-task.html’)
3. In urls.py, add path for delete-task
   1. path(‘delete-task/<str:pk>’, views.delete\_task, name=”delete-task”)
4. In tasks.html, add a link to delete-task.html
   1. <a href=”{% url ‘delete-task’ task.id %}”>DELETE</a>

**Static Files**

* Files that don’t change when a webapp is running
  + CSS, JS, Image files

**Configuring Static Files in Django**

1. Create static folder in root folder (Edenthought)
2. Go to settings.py
   1. Under STATIC\_URL = ‘static/’ add the following code:
      1. STATICFILES\_DIRS = [BASE\_DIR / ‘static’]
      2. **OR** STATICFILES\_DIRS = [os.path.join(BASE\_DIR, ‘static’)]

**Configure and Connect CSS Files**

1. Create new folder “css” in static folder
2. Create new file “styles.css” in css folder
   1. Add css to styles.css
      1. h2 {color : blue;}
3. In index.html, load static file and link styles.css file
   1. {% load static %}
   2. <link rel=”stylesheet” type=”text/css” href=”{% static ‘css/styles.css’ %}”>

**Configure and Connect JS**

1. Create new folder “js” in static folder
2. Create new file “app.js” in js folder
   1. Add js to app.js
      1. alert(“Hello world!”);
3. In index.html, link app.js file
   1. <script src=”{% static js/app.js’ %}”></script>

**Configure and Connect Images**

1. Create new folder “images” in static folder
2. Drag/drop image to image folder
3. In index.html, add image
   1. <img src=”{% static ‘images/plant.jpg’ %}”>

**October 17 2022**

**Create a new user**

Give them a username and password

1. Got to index.html
   1. Remove <img> and Comment <script></script>
2. Create new file “register.html” in templates folder
3. In views.py, add a view
   1. def register(request):
      1. pass

* Pass = lets you pass something in, so not to get errors

1. In urls.py, add a path
   1. path(‘register’, views.register, name=”register”)
2. In forms.py, create a user with built-in Django model form
   1. from django.contrib.auth.forms import UserCreationForm
3. import Django built-in User authentication model (found in settings.py)
   1. from django.contrib.auth.models import User
4. Create class for user creation
   1. Class CreateUserForm(UserCreationForm):
      1. Class Meta:
         1. model = User
         2. fields = [‘username’, ‘email’, ‘password1’, ‘password2’]
         * User model is from Django built-in User
         * Password1 = initial password
         * Password2 = verified password
5. In views.py, import CreateUserForm
   1. from .forms import CreateUserForm
6. Finish body of register function
   1. form = CreateUserForm()
   2. If request.method == ’POST’:
      1. form = CreateUserForm(request.POST)
      2. if form.is\_valid():
         1. form.save()
         2. return redirect(‘’)
      3. conext = {‘form’ : form}
      4. return render(request, ‘register.html’, context)
7. Copy-Paste HTML from create-task.html to register.HTML
8. In index.html, add link to register.html
   1. <a href =”{% url ‘register’ %}”> REGISTER </a>
9. Create new file “success.html” in templates folder
10. Add HTML
    1. <h1>Registration successful!</h1>
11. In views.py add view for success.html and change register’s redirect to ’success’
    1. def success(request):
       1. return render(request, ‘success.html’)
12. In urls.py add path for success.html
    1. path(‘success’, views.success, name=”success”)

**Authentication**

* Can add two factor authentication (can be done in Django project)
* Can authenticate with user/password
* To verify user is created, can check in /admin page to see users

**Create a Log-in Page**

1. Create new file “my-login.html” in templates folder
2. Copy-paste from register.html to my-login.html
3. Create new file “dashboard.html” in templates folder

* Where to go when user is logged in
* Redirect URL

1. Copy-Paste include and header from my-login.html into dashboard.html
2. In forms.py
   1. Import Django’s built-in AuthenticationForm
      1. from django.contrib.auth.forms import AuthenticationForm
   2. Import Django’s widgets for PasswordInput and TextInput
      1. from django.forms.widgets import PasswordInput, TextInput

* Used to input username and password, like HTML text/password inputs
  1. Import Django forms
     1. from django import forms
  2. Create LoginForm class
     1. class LoginForm(AuthenticationForm):
        1. username = forms.CharField(widget=TextInput())
        2. password = forms.CharFields(widget=PasswordInput())

1. In views.py, import LoginForm and auth
   1. from .forms import TaskForm, LoginForm
   2. from django.contrib.auth.models import auth
2. In views.py import authenticate, login, and logout
   1. from django.contrib.auth import authenticate, login, logout
3. In views.py, add a view for my-login
   1. def my\_Login(request):
      1. form = LoginForm()
      2. if request.method == ‘POST’:
         1. form = LoginForm(request, data=request.POST)

* Pass request, and pass data as a POST request
  + - 1. if form.is\_valid ():
         1. username = request.POST.get(‘username’)
         2. password = request.POST.get(‘password’)
         3. user = authenticate(request, username=username, password=password)
* Pass request to authenticate the user
* Check if username in database == username in the form and if password in database == password in the form
* If they match, the user is authenticated
  + - * 1. if user is not None:

auth.login(request, user)

return redirect(“dashboard”)

* If the user exists, login with the request, logging in the user and redirect to dashboard
  + 1. context = {‘form’ : form}
    2. return render(request, “my-login.html”, context)

1. In urls.py, add a path for my-login
   1. path(‘my-login’, views.my\_Login, name=”my-login”)
2. In dashboard.html, add a header confirming login
   1. <h3>You have logged in!</h3>
3. In views.py, create view for dashboard
   1. def dashboard(request):
      1. return render(request, “dashboard.html”)
4. In urls.py, add path for dashboard
   1. path(’dashboard’, views.dashboard, name=”dashboard”)
5. Add link in index-html to my-login.html
   1. <a href=”{% url ‘my-login’ %}”>Log in</a>

**How to Logout**

1. Create a url-path and view for logging out
2. In views.py, create the view for logout
   1. def user\_logout(request):
      1. auth.logout(request)
      2. return redirect(“my-login”)
3. In urls.py, add a path to logout
   1. path(‘user-logout’, views.user\_logout, name=”user-logout”)
4. In dashboard.html add a logout link
   1. <a href=”{% url ‘user-logout’ %}”>LOGOUT</a>

**Protect Views / only allow authorized users to see certain views**

1. In views.py create a decorator
2. Import login required decorator
   1. from django.contrib.auth.decorators import login\_required

* Must be logged-in first to see the view it’s attached to

1. In views.py add decorator to dashboard view
   1. @Login\_required(login\_url=”my-login”)

* If user isn’t logged-in, show the my-login.html

**Pre-Project Cleanup**

* In templates folder, remove everything except index.html and base.html
* In index.html remove all the links
* In journal > urls.py, remove all the paths except for home
* In journal > views.py, remove everything but home view and import redirect, render
* In journal > forms.py, remove everything but ModelForm import
* In journal > admin.py, remove everything but admin import
* In journal > models.py remove everything by models import
* In CMD, close port (ctrl + c)
* In CMD, migrate deletion of models
  + python manage.py makemigrations
  + python manage.py migrate