In Reference to Linkedin Learning Course Building a personal portfolio with Django

https://www.linkedin.com/learning-login/share?forceAccount=false&redirect=https%3A%2F%2Fwww.linkedin.com%2Flearning%2Fbuilding-a-personal-portfolio-with-django%3Ftrk%3Dshare_ent_url&account=56973065

Building a personal portfolio with Django

Staring a new project:

- First, we use the following command to start a new project.
  - Django-admin startproject Website Name
- Then we can run this by using
  - Python manage.py runserver

Create Django Application in our project:

- A Website can have many elements like
  - Blog
  - Event
  - Accounts
- So, in this the project is the website and the elements are the applications.
- So, we can change a particular application without changing the other.
- To create a new application:
  - Django-admin startapp jobs
- Then after you have created an applications name jobs in the project folder. Then you need to add it to the setting file in the project folder.
- To do that go to setting.py and navigate to the INSTALLED_APP. There are already going to be some preinstalled applications there.
- Add the name of the application into the list.
- And add a comma at the end off the string.

Setting up URLs in your Django project

- For this we need to go to the urls.py file and then we create url patterns.
- For this example, we used path and set a name like in nick in the string. But this is just for example. After this we set the path like jobs.views.nick to specify the view where the request should go.
- We also set a name this is used in future.
- After this we need to create a view in jobs.views.
  - For this we create a class name nick
  - And set a return that calls to render (which takes two attribute 'request' this is the request object that is passed by pattern, and the template name where the request will be routed 'for this example jobs/home.html')
- Now we need to create a html file for template.
- And this code is similar to html code with some options of calling and running python and Django stuff which will run in real-time to generate appropriate response according to available data.

Create a model in Django

- For this we go to jobs.models.py file and then we create a class name Jobs. It's not necessary to be of the same name as application it's just a coincidence.
- Then we created a variable to take image as input and another for text summery.
- For image we need to install pillow using pip.
- Then we set the path where we want the images to be saved "here images/ which is in the jobs folder"
- Then for the summary we set the max_lenght =200.

Postgres setup for Django

- For windows the installation is a bit different but you can look into the link:
  - https://www.postgresqltutorial.com/install-postgresql/
- For using u can either use the Postgres Admin 4 or shell
- Then using the GUI we can create a database.
- So, after we have done that we need to then setup our configuration in the setting.py file.
- The following information needs to be changed.
  - 'ENGINE': 'django.db.backends.postgresql',
  - 'NAME': 'PortfolioDB',
  - 'USER': 'postgres',
  - 'PASSWORD': 'admin',
  - 'HOST': 'localhost',
  - 'PORT': '5432',
- After that you are required to install psycopg2 using pip command.
- And that it then you can run the server to check if everything is fine.

Make Django migrations and migrate

- So, when you create a model or change a model you need to migrate the changes.
- To do that we use the following command:
  - Python manage.py makemigrations
- Then to migrate:
  - Python manage.py migrate
- So even when you are making changes like change of database there is a requirement to migrate these changes to do that, we use the above commands.
- After we have done that, we can now start the development for storing the data into the database.
- So, the migrations are changes that are forwarded to the database.]

Setup an admin panel in Django

- The first step is to create a super user.
- To do that we run the following command:
  - Python manage.py createsuperuser
  - Then this command is going to take use through the process of creating the user.
- To do this we need to import Job form the .models file
- As we can see that the Job is the class name in the models.py file
- And then add:
  - Admin.site.register(Job)

- So, by doing this we have added the Job model to the Django admin panel form where we can edit the data.

Create a model object via admin panel Django

- So, this is just to show that we can use the admin panel to save the data and also for the image that we needed to be saved at the particular location is also working and we need not worry about the folder as that is also created.

Pulling objects from database in Django

- So, to pull the data from the database and to show the results in the web page we need to modify the views.
- The first thing we do is to import the model from models.py using the following line of code:
    - From .models import Job
- Then we fetch the required data from the model by:
    - Jobs (Variable)= Job.objects
- After that we can pass this data to html template to create a viewable page out of it:
- We do this by passing the data as dictionary to our render function which was called in return.
- return render(request, 'jobs/home.html', {'jobs':jobs})
- Now we need to change the template to view this information:
    - <h1>
    - All my Jobs
    - </h1>
    - {% for job in jobs.all %}
    -
    - {{job.summary}}
    -
    - {% endfor %}

Designing your Django project

- So, to make the website look good we can use something called as Bootstrap.
- Go to the bootstrap website and then select the stile (here we selected album).
- Then we need the html code for that.
- Right click on the page and view source.
- Copy all the code and then save it below the home page you already have.
- But we ae not done yet we still need the Css file for it to render properly.
- For that go to the home page in bootstrap website and then click on the get started.
- From there you can see that there is code for CSS and Java Script.
- The CSS code is to be pasted below the Bootstrap core Css comment.
- Also remove the code that is already present there.
- Then for the Java Script go to the end of the code in the home.html page where the code for the album page was pasted in the previous instruction.
- There do the same remove the previously mentioned code and put the code copied from the get started page.
- Then if you refresh the page you will see the similar page like in the bootstrap album page.
- Then we will change the content of the page to suit our requirement.

- To do that we first started with the title and changed that to what is relevant to us. Then we changed heading and paragraph below it.
- Also, we removed the header and footer but this all changes depend on what you want to do with the page.
- Also, removed the redundant boxes for the image and text.
- As, we want the boxes to appear according to the content in the database.
- We removed one of the secondary button and then changed the first button content to Email me. And changed the link to mailto:temp@email.com.
    - Email Me
- To add something that is static.
- That is, it is not connected to our database.
- Inside the jobs folder create a new folder called static.
- And this is were we will add static stuff.
- Then for this example I put the image of Django in the static folder.
- Now we need to use this image in our website.
- So, this static thing is not just for images. The important stiff like the Css, Javascript and other things can also be placed there.
- So, if you go to the setting in the project you can see at the end there is a code like :
    - STATIC_URL = '/static/'
    - So, this is to identify the image javascript and css in the project
- This other line is to say where we want our static to be stored.
- Now, we add a STATIC_ROOT = now we specify where this is but instead of just a path we should always give it from out base of our project. So, even if we move it to server we done face any issues.
- So, add something like
    - STATIC_ROOT = os.path.join(BASE_DIR,'static')
    - This says that we want the stuff to be in the base directory and in the folder name static.
- Now we move to need to edit the url so that we can specify at which url they should be displayed at.
- So, first we import our setting and then we import the static.
    - from django.conf import settings
    - from django.urls.static import static
- Then we add the following after the urlpatterns.
    - + static(settings.STATIC_URL, document_root = settings.STATIC_ROOT)
    - So, this code is to specify that these are the things we need to use when we are rendering the above pages.
    - Now we need to collect static files.
    - So, this is like if you have many applications in your project then collect static would bring them all to the same place.
    - To do that type python manage.py collectstatic.
    - For the first time run the admin part has a lot of static so its going to show crazy number of files.
    - Then a new folder will show up in the root directory of the project.
    - So, changing the location in the setting will change the function of this command.
    - Then we go to the home.html template.
    - And add the following after the start of html tag.

    - {% load static %}
    - This is to say to the Django that there is a static been used in this page
    - Then in the place where we had the email button we can add an image tag and code it like this:
        - <img src="{% static '1_25Le7KoMK_z6BIaM8x74RA.png' %}" height=300></img>
- And that's it now we can run the server and check everything.

Bootstrap as a static asset in Django

- Go to the Bootstrap home page and then click on download and then download the compressed file for css and javascript.
- Then go to jquery.com and click on downloads button.
- And click on the download the compressed production jquery (version number)
- Probably the first link.
- And then goto popper.js.org.
- In the top you can see install popper.
- There is a link in heading is CDN.
- It will open a file in your browser. Save that file.
- The default name should come out to be popper.min.js.
- After that we can edit the css and java script tags as:
    - <link rel="stylesheet" href="{% static '/css/bootstrap.min.css' %}">
    - <script src="{% static '/jquery-3.5.0.min.js' %}"></script>
    - <script src="{% static '/popper.min.js' %}"></script>
    - <script src="{% static '/js/bootstrap.min.js' %}"></script>
- Now we need to edit the column where the jobs are going to show up:
- To do that we first delete the extra code in the column and change it look something like this:
    -
    - {% for job in jobs.all %}
    -
    -
    - <img class= "card-img-top" src = ""></img>
    -
    - {{job.summary}}
    -
    -
    -
    -
    -
    -
    - {% endfor %}
- Now for the image to show up in accordance to the database we need to do dome more work.
- We will start with the settings.py file.
- And add the following line:
    - MEDIA_URL = '/media/'
    - MEDIA_ROOT = BASE_DIR

- This similar to what we did for the first time in the static addition.
- And in the url make changes like this and this is just a cleaning and addition of media in the as the static resource:
    - urlpatterns = [
    - path('admin/', admin.site.urls),
    - path('nick', jobs.views.nick, name = 'nick'),
    - path('', jobs.views.nick, name = 'nick'),
    -]
    -
    - urlpatterns += static(settings.STATIC_URL,document_root = settings.STATIC_ROOT)
    -
    - urlpatterns += static(settings.MEDIA_URL,document_root = settings.MEDIA_ROOT)
- And in the home template we change the img tag like this:
    - <img class= "card-img-top" src = "{{job.image.url}}"></img>
    - Just specifying the url of the image in src.

Connecting URL and templates in Django

- So, as we have some entries in the database now it is not right seeing entries in the form of job object 1 and so on.
- To change this, we define the Job model to contain a function like given below:
    - def __str__(self):
    - return self.summary
    - This is in the job class
- Adding a new path:
    - So, to get individual jobs and assign then a url we need to add the following code:
    - path(',jobs/<int:job_id>',
    - this needs to be added in the url patterns in urls.py file.
    - So, what this says is if some one comes to the website and goes to jobs/number/
    - Then we want to save that number as job_id.
    - So now we need to add the code that specifies that were to go with this request.
    - To do that modify the above code to:
        - path('jobs/<int:job_id>', jobs.views.detail, name = 'detail')
    - Now this is talking about a view name detail. But we have nothing like that so we add a view.
        - def detail(request,job_id):
            - print(job_id)
            - return render(request, 'jobs/home.html')
    - This takes the job_id variable that we just created in the url patterns
    - So, it's just to check if everything is working. Therefore, we are just printing the number.

Creating View in Django

- So, to view a particular job and view it in separate page.
- We change the import at the top to:
    - from django.shortcuts import render , get_object_or_404
    - This it will try to get a singular particular object from the database if it works it will return that object else give a 404 page.

    - So now we edit the detail function like this:
        - def detail(request,job_id):
        - job_detail = get_object_or_404(Job, pk = job_id)
        - return render(request, 'jobs/detail.html',{'job':job_detail})
- After we have done that, we need to then create the detail template in the jobs folder in the template folder.
- Then we add the following line in html:
    - {{ job.summary }}

Designing objects details in views

- So now we need to make the detail page look good.
- To edit the html page detail for this and we can copy paste most of the stuff from the home.html.
- And the edit that to something like this:

- <!doctype html>
- <html lang="en">
- {% load static %}
- <head>
- <meta charset="utf-8">
- <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
- <meta name="description" content="">
- <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
- <meta name="generator" content="Jekyll v3.8.6">
- <title>Nick Walter</title>
-
- <link rel="canonical" href="https://getbootstrap.com/docs/4.4/examples/album/">
-
- <!-- Bootstrap core CSS -->
- <link rel="stylesheet" href="{% static '/css/bootstrap.min.css' %}">
-
- <!-- Favicons -->
-
- <meta name="theme-color" content="#563d7c">
-
-
- <style>
- .bd-placeholder-img {
- font-size: 1.125rem;
- text-anchor: middle;
- -webkit-user-select: none;
- -moz-user-select: none;
- -ms-user-select: none;
- user-select: none;
- }

```
@media (min-width: 768px) {
  .bd-placeholder-img-lg {
    font-size: 3.5rem;
  }
 }
</style>
<!-- Custom styles for this template -->
<link href="album.css" rel="stylesheet">
</head>
<body>
<main role="main">

<section class="jumbotron text-center">
  <div class="container">
    <h1>Job Detail</h1>
    <p class="lead text-muted">{{job.summary}}</p>
    <img src="{{job.image.url}}" height=300></img>
    <p>
      <a href="{% url 'nick' %}" class="btn btn-primary my-2">Back</a>
    </p>
  </div>
</section>
</main>
<script src="{% static '/jquery-3.5.0.min.js' %}"></script>
<script src="{% static '/popper.min.js' %}"></script>
<script src="{% static '/js/bootstrap.min.js' %}"></script>
</html>
```

- Also, that the name that we were giving to the url patterns can now be used to refer them for example we use one in the back button.

URL paths with parameters

- So, when do the image. For the places where we refer to data that has been passed to the page, we use the {{}} also we use the {%%}.
- So, to create the click in the home page
- We edit the home page like this:

```
<div class="row">
  {% for job in jobs.all %}
  <div class="col-md-4">
    <a href="{% url 'detail' job.id %}">
      <div class="card mb-4 shadow-sm">
        <img class= "card-img-top" src = "{{ job.image.url }}"></img>
        <div class="card-body">
          <p class="card-text">{{job.summary}}</p>
```

```
            </div>
          </div>
        </a>
      </div>
      {% endfor %}
    </div>
```