In Reference to Linkedin Learning Course Django Forms

https://www.linkedin.com/learning-login/share?forceAccount=false&redirect=https%3A%2F%2Fwww.linkedin.com%2Flearning%2Fdjango-forms%3Ftrk%3Dshare_ent_url&account=56973065

Django Form

Getting Started with Forms

- For that first we need to activate our environment.
- Make sure all the dependencies are installed.
- And then create a project by using the following Django command:
    - Django-admin startproject nandiasguarder
- To avoid confusion between the subfolder and folder that is created by the above command rename the project folder by using:
    - Ren nandiasguarden nandiasguarden-project
- Then we can cd into our project folder and run the server by:
    - Python manage.py runserver
- As we need to create a website for customers to order a pizza.
- First, we need is a home page and then an order from where customers can order pizzas.
- First, start with urls.py file.
    - from django.contrib import admin
    - from django.urls import path
    - Importing views from our application.
    - from pizza import views
    - urlpatterns = [
    - path('admin/', admin.site.urls),
    - path('',views.home, name = 'home'),
    - path('order',views.order, name='order'),
    -]
    - So, we added the url patterns for home and order page.
- Then we create views for our page.
    - Go to views.py file and add
    - from django.shortcuts import render
    -
    - def home(request):
    - return render(request, 'pizza/home.html')
    -
    -
    - def order(request):
    - return render(request, 'pizza/order.html')
    - We, created two views home and order.
- Then we create the following templates for them.
- These templates are in pizza(that is inside out project folder and is the name of the application for which we require the views)\templates(don't forget the 's')\pizza\home.html and order.html in the same directory.
- Then add the code in templates:
    - Home.html

    - <h1>Nandia's Guarden</h1>
    - <h2>Order a Pizza
    - Order.html
    - Order a Pizza
- You can use dupl commend for duplicating code or file in visual studio code.
- To do that first install file utils by Steffen Liestner.
- And the use it by ctrl+shift+P key and then type dupl to use the functions.
- So, then we can create a form using the old html basic form creating method.
- This a quick code snippet of all the code we need to create an basic order page.
    - Add this to urls.py in urlpattern
        - path('order',views.order, name='order'),
    - Then Add the view in views.py
        - def order(request):
        - return render(request, 'pizza/order.html')
    - Then add the template for ordering a pizza.
    - <h2>Order a Pizza</h2>
    - So, there are two methods get and post. As this is centric to Django so, this won't be covering that section. For reference on the same take a look at:
    - https://www.w3schools.com/tags/ref_httpmethods.asp
    - <form action="{% url 'order' %}" method="post">
    - So with post we are required to pass csrf_token. Learn more about csrf_token on this : https://docs.djangoproject.com/en/3.0/ref/csrf/
    - {% csrf_token %}
    - <label for="topping1">
    - Topping 1:
    - </label>
    - <input id="topping1" type="text" name="topping1">
    - <label for="topping2">
    - Topping 2:
    - </label>
    - <input id="topping2" type="text" name="topping2">
    - <label for="size">
    - Size:
    - </label>
    - <select id="size" name="size">
    - <option value="Small">Small</option>
    - <option value="Medium">Medium</option>
    - <option value="Large">Large</option>
    - </select>
    - <input type="submit" value="Order Pizza">
    - </form>
- Then when you run the server, we get a form in the order page asking us for topping and pizza size selection.
- But this need improvement and Django can do it better.
- So, to create a Django form we need to first create a file called forms.py and then add the following code into it.
    - We start with importing the Django form into the file.

    - from django import forms
    - Then create a class that inherits the Form class in forms.
    - class PizzaForm(forms.Form):
    - topping1 =forms.CharField(label='Topping 1', max_length=100)
    - topping2 =forms.CharField(label='Topping 2', max_length=100)
    - size = forms.ChoiceField(label='Siza',choices=[('Small','Small'),('Medium','Medium'),('Large','Large')])
- Then we can import the form into our views and pass it as dictionary to the template.
- Just Add the following code into the views and change the template in similar fashion.
    - Views.py
        - def order(request):
        - form = PizzaForm()
        - return render(request, 'pizza/order.html', {'pizzaform':form})
    - Order.html
        - <h2>Order a Pizza</h2>
        - <form action="{% url 'order' %}" method="post">
        - {% csrf_token %}
        - {{pizzaform}}
        - <input type="submit" value="Order Pizza">
        - </form>
- So, now we can make changes to the form in future by just changing the forms.py file.
- Now, we will try to display that your order is placed.
- So, to do that first go to the views.py and add condition to check if the request is a post request.
    - def order(request):
    - So, this condition is to check whether the request method is post.
    - if request.method == 'POST':
    - If so, then we set the filled form details form our post method.
    - filled_form = PizzaForm(request.POST)
    - Then we validate our input.
    - if filled_form.is_valid():
    - Ones validated we can use the information to display a note saying “Your order is received.”
    - note = 'Thanks for ordering! Your %s %s and %s pizza is on its way!'%(filled_form.cleaned_data['size'],
    - filled_form.cleaned_data['topping1'],filled_form.cleaned_data['topping2'])
    - If we want that page to still have the form for ordering we create a new form. And send the note and new form to render.
    - new_form = PizzaForm()
    - return render(request, 'pizza/order.html',{'pizzaform':new_form, 'note':note})
    - else:
    - Or else the form is going to get generated normally.
    - form = PizzaForm()
    - return render(request, 'pizza/order.html', {'pizzaform':form})

Working with Advanced form Features

- For saving the information into our database.
- So, to do that we need to create models in our database.
- We do that by using the following code.
    - Import models fromm Django db.
    - from django.db import models
    - Create a model by inheriting from model
    - class Size(models.Model):
    - title = models.CharField(max_length=100)
    - This id to display the name in out admin panel for the objects that that exist in size table.
    - def __str__(self):
    - return self.title
    - Then we create a pizza with foreign key size.
    - class Pizza(models.Model):
    - topping1 = models.CharField(max_length=100)
    - topping2 = models.CharField(max_length=100)
    - And we set on_delete to models.CASCADE.
    - #if one thing is deleted we are going to delete the corresponidng object too. That's what on_delete = models.CASCADE means.
    - So, if say small size was deleted then all small size pizza's are going to be deleted too.
    - size = models.ForeignKey(Size, on_delete=models.CASCADE)
- Ones we have the model we can use the ModelForm class in forms in Django.
- So, we can get the same form by using this code in forms.py
    - from django import forms
    - We need to import our model that we want to create form for.
    - from .models import Pizza
    - Also we inherit the form from ModelForm.
    - class PizzaForm(forms.ModelForm):
    - We need to create a meta class that has the model and the fields set
    - class Meta:
    - model = Pizza
    -
    - fields = ['topping1', 'topping2','size']
    - This is to specify the labels which are going to show up on screen.
    - Not required but used when we need the name to be a bit different.
    - Here, we did it to get space between the topping and 1 or 2.
    - labels = {'topping1': 'Topping 1', 'topping2': 'Topping 2'}
- How to customize forms using widgets.
- So, just type widgets and inside the forms there are various kinds of widget.
- These help in setting up the view in which way the form elements should be displayed.
- For example, the checkbox multiple option.
- So, for advancement in widget we can try something like this.
    - from django import forms
    - from .models import Pizza, Size
    -
    - class PizzaForm(forms.ModelForm):

    - So, now we have no empty label like "----" and also we have changed the query set to be retrieved from database. This makes it easy to make changes to this in future.
    - Also, we changed the CheckboxSelectMultiple to RadioSelect.
    - size = forms.ModelChoiceField(queryset=Size.objects, empty_label=None, widget=forms.RadioSelect)
    - class Meta:
    - model = Pizza
    - fields = ['topping1', 'topping2','size']
    - labels = {'topping1': 'Topping 1', 'topping2': 'Topping 2'}
- Then to add the capability to add the files we need to first edit the template file of order so that it knows that this page has the capability to something like that.
- So, we add the following line to our form tag.
    - enctype="multipart/form-data"
- Then we then need to install pillow using the pip command. This allows us to work with images.
- Then we add the following field in the forms.py file
    - image = forms.ImageField()
- But there is one more thing that needs to be done.
- There was a problem that our view code was not ready to accept the image.
- So, to do that we need to add this request.FILES with the request.POST in the pizzaform method in the order view.
- Form sets allow us to take a form and repeat over and over. So that people can order multiple pizzas at once.
- To do this, we need to edit the order.html file first.
- Add the following code after our form tag end.
    -

    - This will ask the user it they want to order more than one pizza.
    - Want more then one pizza?
    - If we send then to the pizzas page and we don't need to hide the number of pizzas. So, get method is ok.
    - <form action="{% url 'pizzas' %}" method="GET">
    - {{ multiple_form }}
    - <input type = "submit" value="Get Pizzas">
    - </form>
- Now, as we can see above we are calling a multiple_form but form does not exist yet so lets create a form called multiple_form that takes the number of pizzas that the user wants to order.
- So, for that we add the following code in the forms.py file.
    - class MultiplePizzasForm(forms.Form):
    - number = forms.IntegerField(min_value=2, max_value=6)
    - As, we can see that the mix and min value are set to 2 & 6 so that the pizza input is realistic.
    - If the values are bigger then or smaller than the number's specified then the input is invalid.
- Now, we need to pass this form to the order page.
- To do that import MultiplePizzasForm inside the views.py file.
- Then edit the order method like this:

    - def order(request):
    - Create instance of MultiplePizzasForm and we are calling that instance as multiple_form
    - multiple_form = MultiplePizzasForm()
    - if request.method == 'POST':
    - filled_form = PizzaForm(request.POST)
    - if filled_form.is_valid():
    - filled_form.save()
    - note = 'Thanks for ordering! Your %s %s and %s pizza is on its way!'%(filled_form.cleaned_data['size'],
    - filled_form.cleaned_data['topping1'],filled_form.cleaned_data['topping2'])
    - new_form = PizzaForm()
    - Then we pass the multiple_form in with other thing to the order page.
    - return render(request, 'pizza/order.html',{'pizzaform':new_form, 'note':note, 'multiple_form':multiple_form})
    - else:
    - form = PizzaForm()
    - And we do this for both the cases of render request.
    - return render(request, 'pizza/order.html', {'pizzaform':form, 'multiple_form':multiple_form})
- After that we need to create a view for multiple pizza ordering page.
- So, add the following code into the views.py:
    - Defining the views for multiple pizzas as we mentioned in the urls.py file.
    - def pizzas(request):
    - Setting a default value for the number of pizzas that can be ordered in the case of multi pizza order to 2.
    - number_of_pizzas = 2
    - Now, we take the request and get the filled form from there.
    - filled_multiple_pizza_form = MultiplePizzasForm(request.GET)
    - Checking if the filled form is valid. (min_value=2, max_value=6 or else invalid)
    - if filled_multiple_pizza_form.is_valid():
    - If valid then set the number of pizzas the customer wants to order to that specified number.
    - number_of_pizzas = filled_multiple_pizza_form.cleaned_data['number']
    - Then create a formset by using the formset_factory .This needs to be imported (form django.forms import formset_factory)
    - We are creating a method name PizzaFormSet form formset_factory() which takes the form that we want to be repeated and the number of times it must be repeated.
    - So, in our case. We want to repeat the PizzaForm and the number of pizzas is the number of times we want it to be repeated.
    - PizzaFormSet = formset_factory(PizzaForm, extra= number_of_pizzas)
    - Then form that method we create our new formset.
    - formset = PizzaFormSet()
    - Then we are checking if the request is post.
    - if request.method == 'POST':
    - If so take the form and check if it is valid.
    - filled_formset = PizzaFormSet(request.POST)
    - if filled_formset.is_valid():

    - If the input is valid then we loop through the formset.
    - for form in filled_formset:
    - And print the name of first topping in console for confirmation that everything is working fine.
    - print(form.cleaned_data['topping1'])
    - We can also pass a note to the user that we have received your order.
    - note = 'Pizza have been ordered!'
    - else:
    - Or else if the information filed is invalid or something went wrong then notify the user about that.
    - note = 'Order was not created, plase try again'
    - Then after the if else loop for checking if the form is valid return a render request to multiple pizzas page. We pass the note and the formset to pizzas.html. (template pizzas.html needs to be created)
    - return render(request, 'pizza/pizzas.html', {'note':note, 'formset':formset})
    -
    - else:
    - Or else if the request id not post then return render too.
    - return render(request, 'pizza/pizzas.html', {'formset':formset})
- Now, we need to create the template to view the multiple pizza form.
- Create a file called pizzas.html in the template/pizza folder and write the code as follows.
    - <h1>
    - Order Pizzas
    - </h1>
    - <h2>
    - {{ note }}
    - </h2>
    -
    - <form action="{% url 'pizzas' %}" method="POST">
    - {% csrf_token %}
    - The management form is available as an attribute of the formset itself. When rendering a formset in a template, you can include all the management data by rendering {{ my_formset.management_form }}
    - Refer like : https://docs.djangoproject.com/en/3.0/topics/forms/formsets/ for more information on formset.
    - {{ formset.management_form }}
    - {% for form in formset %}
    - {{ form }}
    -

    - {% endfor %}
    - <input type="submit" value="Order Pizzas">
    -
    - </form>
- Now, we have the option to order multiple pizza but we are not saving this information anywhere so for the next part we will try to save this information into our database.
    - Add this line of code after validating the filled form:
        - filled_form.save()

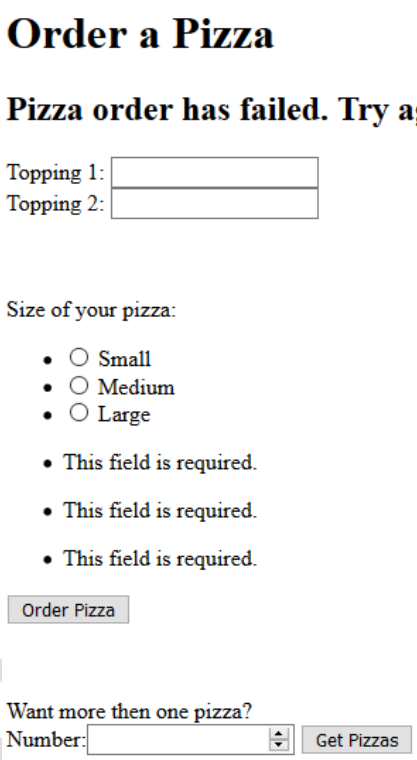
        - And that's it the information will be saved in accordance to the forms as the form is generated according to the models that we have listed. Therefore a database table must exist already before we generate a form.
- How about if the user wants to edit his order.
- So, first create a url for editing the pizza information.
    - path('order/<int:pk>', views.edit_order, name='edit_order'),
- Then edit the order method in the views.py.
    - The place where we were saving the order into the database.
    - created_pizza = filled_form.save()
    - created_pizza_pk = created_pizza.id
    - And then send the pizza id in the directory to the render to order page.
    - new_form = PizzaForm()
    - return render(request, 'pizza/order.html',{'created_pizza_pk':created_pizza_pk,'pizzaform':new_form, 'note':note, 'multiple_form':multiple_form})
- After that we need to create an edit page.
- Create a edit_order.html and add the following code.
    - <h1>
    - Edit Pizza
    - </h1>
    - <form action="{% url 'edit_order' pizza.id %}" method="post">
    - {% csrf_token %}
    - {{ pizzaform }}
    - <input type="submit" value="Edit Order">
    - </form>
- Also, do add the code to view the link to edit order page in the order page.
    - <h2>{{ note }}</h2>
    - So, if the pizza was created before we will get the edit order option.
    - {% if created_pizza_pk %}
    - Edit your order
    - {% endif %}
- Now, we can edit the order we need to notify the user about the successful editing of the order.
- We add the following tag in the edit_order template:
    - <h2>{{ note }}</h2>
- And we add the this code in the views.py to create a note whenever the database is updated with the new edited information.
    - filled_form.save()
    - form = filled_form
    - note = "Order has been updated."
    - return render(request, 'pizza/edit_order.html', {'note':note,'pizzaform':form, 'pizza':pizza})

Customizing and Styling Form Appearance

- Local Validation and errors
    - Django forms provide local validation of data. So, if someone tries to submit the order without filling toppings or going over word limit set in the model we have created then it will not allow submission and will give an error.
    - There are various other fields that you can try like email and url.
    - But if we don't want validation then we can add novaildate into the form tag in the template.
- Server based errors
    - For creating a proper server-side validation, we need to add the following code.
    - So, we edit the views.py file order method in the following way:
        - note = 'Thanks for ordering! Your %s %s and %s pizza is on its way!'%(filled_form.cleaned_data['size'],
        - filled_form.cleaned_data['topping1'],filled_form.cleaned_data['topping2'])
        - We, changed the name form new_form to filled_form.
        - filled_form = PizzaForm()
        - Created an else condition which when form is not valid sets the created_pizza_pk as none. Also, we pass the note that order not created.
        - else:
        - created_pizza_pk = None
        - note = 'Pizza order has failed. Try again.'
        - So, if the form is valid the we'll get the note that order was successful. Else we will get order not successful. Also, the form will be cleared for next order in case the order was successful or else the form will stay with the previous filled information.
        - return render(request, 'pizza/order.html',{'created_pizza_pk':created_pizza_pk,'pizzaform':filled_form, 'note':note, 'multiple_form':multiple_form})
        - else:
        - form = PizzaForm()
        - return render(request, 'pizza/order.html', {'pizzaform':form, 'multiple_form':multiple_form})
- Form rendering
    - We can change the way the form looks with changing in the template of order.html:
        - {{pizzaform.as_p}}
        - This is to view form in paragraph tag.
        - For table view we edit html as:
        - <table>
        - {{pizzaform.as_table}}
        - </table>
        - For order list or unordered list:
            - or
            - {{pizzaform.as_ul}}
            - </table> or
- Customizing forms

- This is how we can customize it completely.(Not Advised but depends on the requirement.)
- For me I used to list all the errors down below.
- {{pizzaform.topping1.label_tag}}
- {{pizzaform.topping1}}
-
- {{pizzaform.topping2.label_tag}}
- {{pizzaform.topping2}}
-
-

- <label for="{{pizzaform.size.id_for_label}}">Size of your pizza:</label>
- {{pizzaform.size}}
- {{pizzaform.topping1.errors}}
- {{pizzaform.topping2.errors}}
- {{pizzaform.size.errors}}



- Spicing up form with css
  - Bootstrap can be used for styling pages quickly.
  - First, install django-widget-tweaks to your environment.
  - Then we copy the code form this link:
  - https://getbootstrap.com/docs/4.4/getting-started/introduction/ in the starter template.
  - Add the new installed application to our project by adding the following in the installed application section in the settings.py file.
  - 'widget_tweaks',
  - Then we edit our order page like this.
  - <!doctype html>
  - <html lang="en">

- <head>
- <!-- Required meta tags -->
- <meta charset="utf-8">
- <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
-
- <!-- Bootstrap CSS -->
- <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
- Change the title.
- <title>Nandia's Garden</title>
- </head>
- <body>
- load the widget_tweaks
- {% load widget_tweaks %}
- Create a container.
-
- <h1>Order a Pizza</h1>
-
- <h2>{{ note }}</h2>
- {% if created_pizza_pk %}
- Edit your order
- {% endif %}
- <form action="{% url 'order' %}" method="post">
- {% csrf_token %}
- Start a for loop for every entry in the form.
- {% for field in pizzaform %}
- Create a form-group div tag for every entry.
-
- Specify the order of entry detail view.
- {{ field.errors }}
- {{ field.label_tag }}
- And set the form controller.
- {% render_field field class="form-control" %}
-
- {% endfor %}
- <input type="submit" value="Order Pizza">
- </form>
-
-

-
- Want more then one pizza?
- <form action="{% url 'pizzas' %}" method="GET">
- {{ multiple_form }}
- <input type = "submit" value="Get Pizzas">

- </form>
-
-
-
- <!-- Optional JavaScript -->
- <!-- jQuery first, then Popper.js, then Bootstrap JS -->
- <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
- <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
- <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
- </body>
- </html>

- Homepage Style
  - Create a html called base.html.
  - And add the following code which is cut out of the order.html:
  - <!doctype html>
  - <html lang="en">
  - <head>
  - <!-- Required meta tags -->
  - <meta charset="utf-8">
  - <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  -
  - <!-- Bootstrap CSS -->
  - <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
  -
  - <title>Nandia's Garden</title>
  - </head>
  -
  - <nav class="navbar navbar-expand-lg navbar-dark" style="background-color: #238a44">
  -
  - Nandia's Garden
  -
  - <ul class="navbar-nav">

- <li class="nav-item active">
- Order Pizza
-
-
-
-
- </nav>
-
- <body>
- {% block 'body' %}
- {% endblock %}
- <!-- Optional JavaScript -->
- <!-- jQuery first, then Popper.js, then Bootstrap JS -->
- <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
- <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
- <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
- </body>
- </html>
- Then edit the rest of the pages in a similar faction and we have a beautiful website.