INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

LICENCIATURA EN CIENCIA DE DATOS

UNIDAD DE APRENDIZAJE

DESARROLLO DE APLICACIONES WEB

PRÁCTICA CRUD

NOMBRE DEL ALUMNO:

DE LUNA OCAMPO YANINA

PROFESOR:

IVÁN BLANCO ALMAZÁN

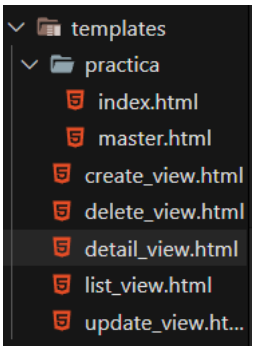
GRUPO:

4CDM1

FECHA:

07/06/2022

Creamos estos html para poner a funcionar lo que queremos implementar. Junto con los .py que se explicarán a continuación.



A partir del proyecto creado en clase, creamos en models.py lo siguiente:

```python
class GeeksModel(models.Model):

    # fields
    title = models.CharField(max_length = 200)
    description = models.TextField()

    def __str__(self):
        return self.title
```

Declarando el modelo que implementaremos para aplicarlo al CRUD. Con makemigrations y migrate, lograremos crear la base de datos para nuestro proyecto. Creamos un forms que será lo que meteremos en la página.

```python
from django import forms
from .models import GeeksModel


# creating a form
class GeeksForm(forms.ModelForm):

    class Meta:
        model = GeeksModel

        fields = [
            "title",
            "description",
        ]
```

Creamos la vista para crear una instancia en la base de datos.

```python
def create_view(request):
    context ={}

    form = GeeksForm(request.POST or None)
    if form.is_valid():
        form.save()

    context['form']= form
    return render(request, "create_view.html", context)
```

Creamos las plantillas que utilizaremos.

```
<form method="POST" enctype="multipart/form-data">

    {% csrf_token %}

    {{ form.as_p }}

    <input type="submit" class="btn btn-primary" value="Submit">
</form>
```

Obtenemos:

Title:

Description:

Enumeramos todas las instancias de una tabla de la base de datos o en un orden particular.

```
def list_view(request):
    context ={}

    context["dataset"] = GeeksModel.objects.all()

    return render(request, "list_view.html", context)
```

Vemos una lista en orden de cada una de las cosas y tareas que vayamos agregando.

Visualizaremos lo que hemos escrito.

```
def detail_view(request, id):
    context ={}

    context["data"] = GeeksModel.objects.get(id = id)

    return render(request, "detail_view.html", context)
```

Actualizamos la vista.

Aplicamos también el borrar por si queremos eliminar una tarea que hemos escrito mal o que ya no necesitamos, etc.

Update:

```python
def update_view(request, id):
    context ={}

    obj = get_object_or_404(GeeksModel, id = id)

    form = GeeksForm(request.POST or None, instance = obj)

    if form.is_valid():
        form.save()
        return HttpResponseRedirect("/"+id)

    context["form"] = form

    return render(request, "update_view.html", context)
```

Vista update:

```html
<div class="main">
    <form method="POST">
        {% csrf_token %}

        {{ form.as_p }}
        <button type="submit" class="btn btn-primary">Update</button>
    </form>

</div>
```

Delete:

```python
def delete_view(request, id):
    context ={}

    obj = get_object_or_404(GeeksModel, id = id)


    if request.method =="POST":
        obj.delete()
        return HttpResponseRedirect("/")

    return render(request, "delete_view.html", context)
```

Vista de delete:

```html
<div class="main">
    <form method="POST">
        {% csrf_token %}
        Are you want to delete this item ?
        <button type="submit" class="btn btn-danger">Yes</button>
        <a href="/">Cancel </a>
    </form>
</div>
```

Mostrando mediante Bootstrap un diseño de botón específico en cada caso dado y aplicación dada, siguiente el formato.

```
<input type="submit" class="btn btn-primary" value="Submit">
```

Cambiando únicamente la parte de class, añadiendo el formato de botón que necesites.