ELECTIVA DE PROFUNDIZACION TECNOLOGICA

ENSAYO

PARCIAL FINAL

JHONIER ARLEY PASOS PERENGUES

INSTITUTO TECNOLÓGICO DEL PUTUMAYO

TECNOLOGIA EN DESARROLLO DE SOFTWARE

28 DE NOVIEMBRE DEL 2023

MOCOA-PUTUMAYO

Contenido

1. INTRODUCCION.

Desde a importancia del desarrollo web estructurado junto al uso de Python ‘Django’ lo que implica la creación de aplicaciones web en conjunto de herramientas que nos permiten construir proyectos escalables y mantenibles, se dará conocimiento previo de ‘Django’ en general hasta la creación de un proyecto con la aplicación de los procedimientos construidos en clase, con fin de comprender la estructura del proyecto y la configuración inicial.

Resaltando que en ‘Django’ se encuentran los modelos, quienes son las piezas fundamentales que definen la estructura de la base de datos. Adquiriendo la capacidad de exportar los modelos permite compartir y colaborar en el desarrollo más efectivo junto al manejo de formularios simples que facilitan la iteración con el usuario exploraremos para que se usan y por qué son más efectivos, el uso de ChoiceField para la gestión predefinida de listas de manera más eficiente para poder utilizarlos en el campo de los formularios, y sumergiéndonos un poco en el uso del date time y el envió de variables atravez de las funciones.

Programación estructurada.

La programación estructurada, surgida en los años 70 con Dijkstra como uno de sus impulsores, revolucionó la forma en que se codifican algoritmos al reemplazar la confusa instrucción "GOTO" con el concepto de funciones o subrutinas. Este paradigma introdujo una organización jerárquica del código, facilitando la comprensión y mantenimiento del software. Sus contribuciones, como la abstracción, la capacidad de depuración por unidades y la mejora en la velocidad de desarrollo, siguen siendo relevantes hoy. Aunque paradigmas posteriores como la programación orientada a objetos han ganado prominencia, la programación estructurada sigue siendo esencial, utilizada no solo en la enseñanza, sino también en proyectos actuales, como evidencian lenguajes como JavaScript y PHP que la incorporan como base. A pesar de la evolución, la programación estructurada perdura como una piedra angular en la construcción y mantenimiento eficiente del software.

En este contexto, el uso de Django, un marco de desarrollo web basado en Python, se destaca como una herramienta fundamental para la construcción de proyectos que no solo cumplen con los estándares actuales, sino que también establecen nuevas fronteras en términos de eficiencia y mantenibilidad

Django.

Django es un framework de desarrollo web de código abierto que utiliza el lenguaje de programación conocido como Python, este framework es muy popular por que usa el modelo de

vista-controlador (MVC). El cual fue diseñado principalmente para gestionar paginas orientadas a la gestión de noticias y fue liberada al público en julio de 2005 y en 2008 se formó la Django Software Foundation quien se haría cargo de Django en el futuro.

Django requiere Python 2.5 o superior y no se necesitan de otras bibliotecas de Python para poder funcionar de manera básica.

Modelos Django:

Uno de los pilares de Django radica en sus modelos, componentes cruciales que definen la estructura de la base de datos. La capacidad de diseñar y crear modelos no solo implica la representación de datos, sino que también facilita la colaboración al permitir la exportación de modelos. Este aspecto es crucial para compartir y construir de manera efectiva en entornos colaborativos de trabajo, es un tipo especial de objeto que se guarda en la base de datos.

Para la creación de modelos Django primero se define nuestro archivo models.py dentro de nuestra aplicación, cabe resaltar que cada aplicación puede tener su propio modelo que utilizan diferentes tipos de datos entre los más comunes son

```
from django.db import models


class Modelo(models.Model):
    nombre = models.CharField(max_length=255)
    edad = models.IntegerField()
    precio = models.FloatField()
```

fecha_nacimiento = models.DateField()

activo = models.BooleanField(default=True)

autor = models.ForeignKey(Autor, on_delete=models.CASCADE)

esos son algunos de los modelos mas comunes y con un ejemplo de donde se podrían usar también existen las foráneas que son para conectar 2 o mas modelos ejemplo en el modelo anterior el campo de autor es una clave foránea que conecta con otro modelo llamado Autor.

Formularios Django:

En Django los formularios son muy buenos para la manipulación de datos atravez de la web, django ya trae un conjunto de herramientas y clases que facilitan la creación y validación de formularios de manera eficiente.

Un ejemplo de uso de formulario seria si tenemos un modelo llamado libro con campos de título, autor, y fecha publicación lo ideal en ves de crear el método de crear un método único para crear libros podemos crear un formulario de ese modelo para agregar libros.

Estos se crean en un archivo en nuestra aplicación con nombre **forms.py** y dentro de ese archivo se realiza la creación de nuestro formulario importando nuestro modelo

```
from django import forms
from .models import Libro


class LibroForm(forms.ModelForm):
    class Meta:
        model = Libro
        fields = ['titulo', 'autor', 'fecha_publicacion']
```

ya con nuestro formulario creado en nuestro archivo **views.py** importamos nuestro formulario creado y le agregamos el método post asignándole una plantilla html y nuestro formulario.

```
from django.shortcuts import render, redirect
from .forms import LibroForm


def agregar_libro(request):
    if request.method == 'POST':
        form = LibroForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_libros')  # Redirigir a la lista de libros o a donde desees
    else:
        form = LibroForm()
```

```
return render(request, 'agregar_libro.html', {'form': form})
```

resaltando que debemos tener una ruta en nuestro archivo urls.py que nos lleve a ese html.

ChoiceField y Datetime:

El ChoiceField se destaca por la capacidad de gestionar listas predefinidas "que no cambiaran" de manera eficiente. Supongamos que tenemos un modelo llamado **tarea** con un campo de estado que puede ser "En Proceso", "Completada" o "Pendiente". En lugar de permitir al usuario ingresar texto libre para el estado, puedes utilizar un **ChoiceField** para limitar las opciones y facilitar la entrada de datos, ejemplo

```
# models.py
from django.db import models


class Tarea(models.Model):
    ESTADO_CHOICES = [
        ('P', 'Pendiente'),
        ('E', 'En Proceso'),
        ('C', 'Completada'),
    ]
```

titulo = models.CharField(max_length=200)

estado = models.CharField(max_length=1, choices=ESTADO_CHOICES)

fecha_creacion = models.DateTimeField(auto_now_add=True)

aquí podemos ver el choicefield llamado ESTADO_CHOICE el cual es una lista que lo utilizamos en el modelo de Tarea en el campo de estado para limitar las opciones a las opciones predefinidas del choice.

El **datetimefield** esto es lo que necesitas al momento de querer manejar fechas y horas en tu modelo es muy útil por que con este campo puedes representar tanto la fecha como la hora y se puede captar automáticamente de esta forma,

fecha_creacion = models.DateTimeField(auto_now_add=True)

al utilizar choicefield y datetimefield de manera efectiva se puede manejar consistencia de los datos y simplificar la entrada del usuario en tu aplicación.

Enviando Variables a Través de Funciones:

En Python “django” se pueden enviar las variables a través de funciones de muchas maneras, principalmente utilizando vistas y plantillas un ejemplo serio este.

 # views.py

from django.shortcuts import render

```
def mi_vista(request):
    mi_variable = "Hola desde la vista"
    return render(request, 'mi_plantilla.html', {'mi_variable': mi_variable})
```

tenemos una variable definida en mi vista que la enviamos a una plantilla html que la podemos ver llamándola de esta manera <h1>{{ mi_variable }}</h1>

también está el envio de variables de vista a vista quien recibe la variable que nos envía la primera vista y de hay uno puede hacer lo que quiera con ella ejemplo:

```
# views.py
from django.shortcuts import render

def vista_uno(request):
    mi_variable = "Hola desde la vista uno"
    return render(request, 'vista_uno.html', {'mi_variable': mi_variable})

def vista_dos(request, variable_recibida):
    return render(request, 'vista_dos.html', {'variable_recibida': variable_recibida})
```

pero en nuestro archivo url debemos especificar que recibe una variable y el tipo de variable que recibe algo así:

```
# urls.py
from django.urls import path
from .views import vista_uno, vista_dos

urlpatterns = [
    path('vista_uno/', vista_uno, name='vista_uno'),
    path('vista_dos/<str:variable_recibida>/', vista_dos, name='vista_dos'),
]
```

Esta flexibilidad en el manejo de datos entre funciones, vistas y plantillas en Django proporciona un entorno poderoso y versátil para construir aplicaciones web robustas. Es esencial entender las diversas formas de enviar y recibir variables para aprovechar al máximo el marco de desarrollo y cumplir con los requisitos específicos de cada proyecto.

Estáticos:

los archivos estáticos en django son recursos que no cambian dinámicamente con cada solicitud de la página, entre estos los más comunes son los estilos css configuraciones JavaScript, imágenes entre otros archivos multimedia, como beneficio nos ayudan a optimizar como ya lo dije anteriormente a estos no cambiar nos ayudan acelerando los procesos de carga también son

muy importantes porque pueden ser reutilizables en otras plantillas html que tengas predefinida y quieras usar los mismos estilos como base.

Conclusiones:

En el transcurso de este ensayo se exploro la importancia del desarrollo web destacando el papel que juega Django, desde la creación inicial de un proyecto hasta la implementación de herramientas como los formularios y los choices simplificando la tarea de los desarrolladores de tener que crear todo desde 0, esto resalta la importancia de este framework que nos ayuda en la eficiencia en el desarrollo web