UNIDAD 3.

Desarrollo de aplicaciones web (Back end).

(El estudiante demostrará el proceso para el desarrollo de una aplicación web.)

Tema I. Lenguajes de programación del lado del servidor.

Evidencia de aprendizaje Tema I –RESUMEN

Temas	Saber	Saber hacer	Resultado de Aprendizaje
Lenguajes de programación del lado del servidor	Identificar la sintaxis del lenguaje de programación del lado del servidor: Variables, expresiones, estructuras de control, clases.	Desarrollar programas del lado del servidor.	Los estudiantes identifican la sintaxis del lenguaje de programación web.
Evidencia de aprendizaje			
A partir de un caso práctico realizar una aplicación web que contenga lo siguiente: Páginas web, hojas de estilos, gestión de contenido de base de datos desde formularios, implementación de seguridad por medio de sesiones.			

Información sobre el profesor

Profesor	Correo	Días de Clase
Bernardo Prado Díaz	Tenor_prado@yahoo.com.mx	Lunes y Martes

SABER: Fundamentos teóricos

¿Qué es la programación del lado del servidor?

Es la lógica que se ejecuta en el servidor web para procesar peticiones, acceder a bases de datos, manejar sesiones y generar respuestas dinámicas. En Django, esta lógica se escribe en Python.

Sintaxis básica de Python (usado en Django)

Variables

Son contenedores de datos.

python

```
nombre = "Ana"        # Variable tipo cadena
edad = 21             # Variable tipo entero
precio = 99.99        # Variable tipo decimal
activo = True         # Variable booleana
```

Expresiones

Combinaciones de valores y operadores que producen un resultado.

python

```
total = precio * 2     # Multiplicación
mensaje = "Hola " + nombre  # Concatenación de cadenas
```

Estructuras de control

Permiten tomar decisiones o repetir acciones.

python

```
# Condicional
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")

# Bucle
for i in range(5):
    print(i)  # Imprime del 0 al 4
```

Clases

Son plantillas para crear objetos con atributos y métodos.

python

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre      # Atributo nombre
        self.precio = precio      # Atributo precio

    def mostrar(self):
        return f"{self.nombre} cuesta ${self.precio}"
```

¿Por qué usar Django?

Django es un framework de alto nivel que permite desarrollar aplicaciones web de forma rápida, segura y escalable. Usa Python como lenguaje principal y ofrece herramientas integradas para manejar rutas, vistas, modelos, formularios, sesiones y más.

SABER HACER: Aplicación práctica en Django

Crear una vista con lógica del lado del servidor

Python

```
# views.py
from django.http import HttpResponse

def saludo(request):
    nombre = "Ana"  # Variable
    mensaje = f"Hola, {nombre}"  # Expresión
    return HttpResponse(mensaje)  # Respuesta al cliente
```

Usar estructuras de control en una vista

python

```
def verificar_edad(request):
    edad = 20  # Variable
    if edad >= 18:  # Estructura de control
        return HttpResponse("Acceso permitido")
    else:
        return HttpResponse("Acceso denegado")
```

Crear una clase modelo en Django

python

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

class Producto(models.Model):
    nombre = models.CharField(max_length=100)  # Atributo tipo texto
    precio = models.DecimalField(max_digits=6, decimal_places=2)  #
Atributo tipo decimal

    def __str__(self):
        return f"{self.nombre} - ${self.precio}"  # Método para mostrar el
objeto
```

Crear una vista que use el modelo

python

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

def lista_productos(request):
    productos = Producto.objects.all()  # Consulta a la base de datos
    return render(request, 'productos.html', {'productos': productos})  #
Envía datos a la plantilla
```

Plantilla HTML para mostrar productos

html

```
<!-- productos.html -->
<h2>Lista de productos</h2>
<ul>
  {% for producto in productos %}
    <li>{{ producto.nombre }} - ${{ producto.precio }}</li>
  {% endfor %}
</ul>
```

Resultado de aprendizaje

- Identificar la sintaxis de Python aplicada en Django (variables, expresiones, estructuras de control, clases).
- Desarrollar vistas, modelos y plantillas que implementen lógica del lado del servidor.
- Comprender cómo se genera contenido dinámico en una aplicación web.
- Aplicar razonamiento lógico para resolver problemas computacionales mediante código.

Vamos a construir un mini proyecto web con Django que integre todo lo que vimos sobre programación del lado del servidor. Será un sistema básico de registro y consulta de productos, ideal para entender cómo se conectan las vistas, modelos, plantillas y lógica del servidor.

Estructura del Proyecto del Tema I.

Nombre del proyecto: inventario_web Funcionalidades:

- Registrar productos (nombre, precio, categoría)
- Listar productos registrados
- Filtrar por categoría

Paso 1: Crear el proyecto Django

Bash

```
django-admin startproject inventario_web
cd inventario_web
python manage.py startapp productos
```

Agrega la app productos en settings.py:

python

```
INSTALLED_APPS = [
    ...
    'productos',
]
```

Paso 2: Crear el modelo de producto

python

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

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=8, decimal_places=2)
    categoria = models.CharField(max_length=50)

    def __str__(self):
        return f"{self.nombre} - ${self.precio} ({self.categoria})"
```

Luego ejecuta:

bash

```
python manage.py makemigrations
python manage.py migrate
```

Paso 3: Crear formularios para registrar productos

python

```
# productos/forms.py
from django import forms
from .models import Producto

class ProductoForm(forms.ModelForm):
    class Meta:
        model = Producto
        fields = ['nombre', 'precio', 'categoria']
```

Paso 4: Crear vistas para registrar y listar productos

python

```
# productos/views.py
from django.shortcuts import render, redirect
from .models import Producto
from .forms import ProductoForm

def registrar_producto(request):
    if request.method == 'POST':
        form = ProductoForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('listar_productos')
    else:
        form = ProductoForm()
    return render(request, 'registrar.html', {'form': form})

def listar_productos(request):
    categoria = request.GET.get('categoria')
    if categoria:
        productos = Producto.objects.filter(categoria=categoria)
```

```
    else:
        productos = Producto.objects.all()
    return render(request, 'listar.html', {'productos': productos})
```

Paso 5: Configurar URLs

python

```
# productos/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('registrar/', views.registrar_producto, name='registrar_producto'),
    path('listar/', views.listar_productos, name='listar_productos'),
]
```

Y en inventario_web/urls.py:

python

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('productos.urls')),
]
```

Paso 6: Crear las plantillas HTML

- registrar.html

```
<h2>Registrar Producto</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Guardar</button>
</form>
```

- listar.html

```
<h2>Lista de Productos</h2>
<form method="get">
  <input type="text" name="categoria" placeholder="Filtrar por categoría">
  <button type="submit">Filtrar</button>
</form>
<ul>
  {% for producto in productos %}
    <li>{{ producto.nombre }} - ${{ producto.precio }} ({{ producto.categoria
}})</li>
  {% endfor %}
</ul>
```

Resultado final

Con este proyecto:

- Se aplican la sintaxis de Python en un entorno real.
- Se Comprende cómo se estructura una app web con Django.
- Se Desarrolla una lógica del lado del servidor para manejar formularios, consultas y vistas.
- Se aprende a conectar modelos, vistas y plantillas para generar contenido dinámico.

Integraremos autenticación de usuarios en tu proyecto Django! Esto permitirá que los usuarios se registren, inicien sesión y accedan a funciones protegidas. Todo lo haremos con herramientas nativas de Django para mantenerlo limpio, seguro y profesional.

¿Qué incluye la autenticación?

- Registro de usuarios
- Inicio de sesión
- Cierre de sesión
- Restricción de acceso a vistas
- Uso de sesiones para mantener al usuario conectado

Paso 1: Activar el sistema de autenticación

Django ya incluye un sistema de usuarios. Solo necesitas usarlo. Asegúrate de tener 'django.contrib.auth' y 'django.contrib.sessions' en INSTALLED_APPS (ya vienen por defecto).

Paso 2: Crear formularios de registro y login

Formulario de registro

python

```
# productos/forms.py
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegistroForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'password1', 'password2']
```

Paso 3: Vistas para registro, login y logout

python

```
# productos/views.py
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.forms import AuthenticationForm
```

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

def registrar_usuario(request):
    if request.method == 'POST':
        form = RegistroForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('iniciar_sesion')
    else:
        form = RegistroForm()
    return render(request, 'registro.html', {'form': form})

def iniciar_sesion(request):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            usuario = form.get_user()
            login(request, usuario)
            return redirect('listar_productos')
    else:
        form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})

def cerrar_sesion(request):
    logout(request)
    return redirect('iniciar_sesion')
```

Paso 4: URLs para autenticación

python

```
# productos/urls.py
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('registrar/', views.registrar_usuario, name='registrar_usuario'),
    path('login/', views.iniciar_sesion, name='iniciar_sesion'),
    path('logout/', views.cerrar_sesion, name='cerrar_sesion'),
    path('listar/', views.listar_productos, name='listar_productos'),
]
```

Paso 5: Proteger vistas con login

python

```
# productos/views.py
from django.contrib.auth.decorators import login_required

@login_required(login_url='iniciar_sesion')
def listar_productos(request):
    ...
```

Esto obliga al usuario a iniciar sesión antes de ver la lista de productos.

Paso 6: Plantillas HTML

registro.html

```
<h2>Registro de Usuario</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Registrarse</button>
</form>
```

login.html

```
<h2>Iniciar Sesión</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Entrar</button>
</form>
```

Agrega botón de logout en listar.html

Cerrar sesión

Resultado final

Con esto, tu aplicación ahora:

- Permite a los usuarios registrarse y autenticarse
- Protege vistas sensibles
- Usa sesiones para mantener la sesión activa
- Refuerza la seguridad y personalización de la experiencia

Vamos a llevar la autenticación de usuarios al siguiente nivel: roles personalizados. Esto te permitirá diferenciar entre tipos de usuarios, como administradores, clientes, empleados, etc., y controlar lo que cada uno puede ver o hacer.

Opción 1: Usar is_staff y is_superuser

Django ya incluye dos roles básicos:

- is_staff: puede acceder al panel de administración.
- is_superuser: tiene todos los permisos.

Puedes usarlos para condicionar vistas:

Python

```
if request.user.is_staff:
   # Mostrar vista especial para staff
```

Pero si necesitas algo más flexible, pasamos a la siguiente opción

Opción 2: Crear roles personalizados con Group

Python

```
from django.contrib.auth.models import Group
# Crear grupos desde el admin o por código
Group.objects.create(name='Clientes')
Group.objects.create(name='Empleados')
```

Luego, asignas usuarios a grupos:

Python

```
usuario.groups.add(Group.objects.get(name='Clientes'))
```

Y en tus vistas:

Python

```
if request.user.groups.filter(name='Clientes').exists():
   # Mostrar contenido exclusivo para clientes
```

Opción 3: Extender el modelo de usuario

Si quieres agregar campos como "tipo de usuario", "empresa", etc., puedes extender el modelo:

Python

```
# productos/models.py
from django.contrib.auth.models import AbstractUser
class UsuarioPersonalizado(AbstractUser):
tipo = models.CharField(max_length=20, choices=[('cliente', 'Cliente'),
('empleado', 'Empleado')])
```

Y en settings.py:

python

```
AUTH_USER_MODEL = 'productos.UsuarioPersonalizado'
```

Lenguajes de Programación del Lado del Servidor con Python y Django

1 SABER – Dimensión Conceptual

Objetivo: Comprender la sintaxis y elementos de un lenguaje de programación del lado del servidor, aplicando Python y Django para desarrollar soluciones funcionales.

A) Conceptos Clave

• Lenguaje de programación del lado del servidor:
Definición: Lenguaje que se ejecuta en el servidor para procesar peticiones, acceder a bases de datos y generar contenido dinámico.
Ejemplo: Ejemplo: Python, PHP, Node.js.

• Variable:
Definición: Espacio de memoria que almacena un valor que puede cambiar durante la ejecución.
Ejemplo: Ejemplo en Python: nombre = 'Juan'

• Expresión:
Definición: Combinación de variables, operadores y funciones que produce un valor.
Ejemplo: Ejemplo: total = precio * cantidad

• Estructuras de control:
Definición: Instrucciones que alteran el flujo normal del programa.
Ejemplo: Ejemplo: if, for, while.

• Clase:
Definición: Molde para crear objetos que encapsulan datos (atributos) y comportamientos (métodos).
Ejemplo: Ejemplo: class Usuario: ...

2 SABER HACER – Dimensión Actuacional

Objetivo: Desarrollar programas del lado del servidor usando Python y Django.

A) Ejemplo de Programa en Python (Django)

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

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    cantidad = models.IntegerField()
```

```
    def calcular_total(self):
        return self.precio * self.cantidad



# archivo views.py
from django.shortcuts import render
from .models import Producto

def lista_productos(request):
    productos = Producto.objects.all()
    return render(request, 'productos.html', {'productos': productos})



<!-- archivo templates/productos.html -->
<table>
    <tr><th>Nombre</th><th>Precio</th><th>Cantidad</th><th>Total</th></tr>
    {% for producto in productos %}
    <tr>
        <td>{{ producto.nombre }}</td>
        <td>{{ producto.precio }}</td>
        <td>{{ producto.cantidad }}</td>
        <td>{{ producto.calcular_total }}</td>
    </tr>
    {% endfor %}
</table>
```

3 SER Y CONVIVIR – Dimensión Socioafectiva

Objetivo: Desarrollar habilidades de trabajo en equipo y razonamiento lógico en el desarrollo de programas del lado del servidor.

- Definir claramente las responsabilidades de cada miembro del equipo (modelos, vistas, plantillas).

- Mantener la comunicación constante durante el desarrollo.

- Usar control de versiones (Git) para la colaboración.

- Realizar revisiones de código entre pares.

- Analizar problemas en conjunto para encontrar la mejor solución.

4 Glosario

- Back-End → Parte del desarrollo web que se ejecuta en el servidor.

- Modelo → Componente en Django que define la estructura de datos.

- Vista → Función o clase en Django que procesa solicitudes y devuelve respuestas.

- Plantilla → Archivo HTML que muestra los datos enviados desde la vista.

- Variable → Espacio de memoria para almacenar datos.

- Clase → Estructura para crear objetos en programación orientada a objetos.