REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Despliegue de Aplicaciones con Flask

INCRUSTAR FUNCIONALIDAD PYTHON EN HTML CON JINJA2 (FLASK)

Tabla de contenido

Documento elaborado por: Alvaro Pérez Niño - Instructor

REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Despliegue de Aplicaciones con Flask

INCRUSTAR FUNCIONALIDAD PYTHON EN HTML CON JINJA2 (FLASK)

Contexto: ¿Por qué Jinja2 en Flask?

Flask utiliza **Jinja2** como motor de plantillas para separar la lógica de negocio (Python) de la capa de presentación (HTML).
En lugar de escribir Python puro dentro del HTML (lo cual es mala práctica), se usan **etiquetas especiales** que permiten:

- Mostrar variables enviadas desde las vistas.
- Hacer bucles y condiciones.
- Aplicar filtros de formato.
- Reutilizar estructuras de HTML con herencia de plantillas.

Así mantenemos una **arquitectura en capas**:

- Python (rutas / servicios) → prepara los datos.
- Jinja2 (plantillas HTML) → se encarga solo de la presentación.

1. Renderizar plantillas desde Flask

En Flask, la función principal para trabajar con Jinja2 es render_template.

Ejemplo en app.py: En este ejemplo, la plantilla inicio.html recibirá las variables usuario y cursos para ser usadas con Jinja2.

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def inicio():
    usuario = {"nombre": "Ana", "rol": "admin"}
    lista_cursos = ["Python", "Flask", "HTML"]
    return render_template("inicio.html",
                           usuario=usuario,
                           cursos=lista_cursos)
```

INCRUSTAR FUNCIONALIDAD PYTHON EN HTML CON JINJA2 (FLASK)

2. Mostrar variables con {{ … }}

Las **dobles llaves** sirven para imprimir datos del contexto que la vista envía.

inicio.html:

```
<p>Hola, {{ usuario.nombre }} 👋</p>
<p>Tu rol es: {{ usuario.rol }}</p>

<p>Curso destacado: {{ cursos[0] }}</p>
```

Jinja2 permite acceder a:

- Atributos de objetos: usuario.nombre
- Índices de listas: cursos[0]
- Claves de diccionarios: datos["clave"] o datos.clave

3. Control de flujo con {% … %}

Las etiquetas {% %} se usan para instrucciones de **control de flujo** (bucles, condicionales, bloques, etc.).

3.1 Bucles for

```
<ul>
  {% for curso in cursos %}
    <li>{{ curso }}</li>
  {% endfor %}
</ul>
```

Dentro del `for` puedes usar el objeto especial `loop`:

```
<ul>
  {% for curso in cursos %}
    <!-- loop.index inicia en 1 -->
    <li>{{ loop.index }}. {{ curso }}</li>
  {% endfor %}
</ul>
```

3.2 Condicionales `if`, `elif`, `else`

```
{% if usuario.rol == "admin" %}
  <p>Bienvenido, administrador.</p>
{% elif usuario.rol == "instructor" %}
  <p>Bienvenido, instructor.</p>
{% else %}
  <p>Bienvenido, usuario.</p>
{% endif %}
```

4. Comentarios en plantillas Jinja2

Los comentarios **no se renderizan** en el HTML final:

```
{# Este es un comentario interno de la plantilla #}
<p>Contenido visible para el navegador.</p>
```

5. Filtros en Jinja2

Los **filtros** permiten transformar valores al momento de mostrarlos. Se utilizan con el operador `|`.

INCRUSTAR FUNCIONALIDAD PYTHON EN HTML CON JINJA2 (FLASK)

Ejemplos:

```
<p>Nombre en mayúsculas: {{ usuario.nombre|upper }}</p>
<p>Texto en minúsculas: {{ usuario.nombre|lower }}</p>
<p>Longitud de la lista: {{ cursos|length }}</p>
```

También puedes formatear fechas (si envías objetos `datetime` desde Python):

```
<p>Fecha: {{ fecha_actual.strftime("%d/%m/%Y") }}</p>
```

6. Archivos estáticos con Flask y Jinja2

En Flask, los archivos estáticos se sirven generalmente desde el directorio static/.
Para referenciarlos en plantillas Jinja2 se usa url_for('static', ...).

Estructura de carpetas sugerida:

```
project/
├─ app.py
├─ templates/
│  ├─ base.html
│  └─ inicio.html
└─ static/
   ├─ css/
   │  └─ estilos.css
   └─ images/
      └─ logo.png
```

INCRUSTAR FUNCIONALIDAD PYTHON EN HTML CON JINJA2 (FLASK)

```
<head>
  <link rel="stylesheet"
        href="{{ url_for('static', filename='css/estilos.css') }}">
</head>
<body>
  <img src="{{ url_for('static', filename='images/logo.png') }}" alt="Logo">
</body>
```

7. Buenas prácticas al usar Jinja2

1. Evitar lógica compleja en las plantillas
    - Haz las operaciones en Python (vistas, servicios, modelos) y passa solo los resultados a Jinja2.
    - La plantilla debe centrarse en mostrar datos, no en procesarlos.
2. Preparar los datos en la vista
3. Usar herencia de plantillas y includes
    - Centraliza cabeceras, menús y pie de página.
    - Facilita el mantenimiento.
4. Comentar solo lo necesario
    - Usa {# ... #} para explicar partes importantes sin saturar el código.
5. Mantener una arquitectura en capas
    - Rutas → Servicios → Plantillas (Jinja2).
6. Ayuda a mantener el proyecto ordenado y facilita pruebas y mantenimiento.

Referentes bibliográficos

1. Palet, A. (2023). *Jinja Documentation*. Palet Software.
2. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2ª ed.). O'Reilly Media.
3. Django Software Foundation. (s.f.). *Template Engines*.
4. Mozilla Developer Network (MDN). (s.f.). *HTML: HyperText Markup Language*.

Documento elaborado por: Alvaro Pérez Niño - Instructor