FLASK martes 01 de agosto

A screen shot of a computer

Description automatically generatedSe hace click ahí y eso significa que estoy trabajando en el ambiente virtual de Python con mi proyecto. Acá py se adapta a las librerías que yo estoy usando y punto. Si el ambiente está activado, la carpeta debe mostrarse entre paréntesis y después recién en azul.

PROCEDIMIENTO DE INSTALACIÓN DE UNA HERRAMIENTA DE AMBIENTES VIRTUALES

Se usa un pip para desarrollar pipenv de forma global:

pip install pipenv

Se crea el entorno virtual y se instala flask en la carpeta en la que quiero ejecutar:

pipenv install flask

La primera vez que ejecutemos pipenv install, creará 2 archivos para nosotros, Pipfile y Pipfile.lock. Ambos son necesarios para usar los paquetes instalados, pero la diferencia entre los dos incluye: Pipfile mostrará los paquetes instalados, y Pipfile.lock tendrá los detalles específicos sobre qué versión se está utilizando.

Para usar el entorno, tenemos que activarlo con:

pipenv shell

para ejecutar lo que esta en mi carpeta y que me genere la ruta debo poner Python y el nombre del archivo en cuestión en la terminal. Cuando por alguna razón no me lee, actualizo, hago que corra otra vez y asi

para salir del entorno virtual se usa:

exit

AVISO IMPORTANTE

No crees un entorno virtual con otro entorno virtual.

Dentro de la carpeta "hola\_flask", crea un archivo llamado server.py

Este será nuestro archivo "servidor" donde configuraremos todas nuestras rutas para manejar solicitudes.

Querrás crear una nueva carpeta para cada asignación en el futuro. Parecerá tedioso al principio, pero a medida que agreguemos archivos adicionales a cada proyecto, ¡querremos mantener todo organizado por tarea/proyecto!

Un decorador en Python (@) indica una acción antes de una función.

**También hicimos un par de cosas importantes en el código anterior:**

Importamos la clase Flask. Necesitarás esta línea en cada aplicación que crees con Flask.

Creamos una instancia de la clase Flask llamada "aspp". Necesitarás esta línea en cada aplicación que crees con Flask.

Configuramos una regla de enrutamiento usando el decorador "@" con el método route: @app.route("/route\_string"). La regla de enrutamiento está asociada con la función que la sigue inmediatamente.

¡Finalmente, ejecutamos la aplicación! Esto toma todas nuestras reglas de enrutamiento que configuramos y, de hecho, inicia el servidor.

El profe recomienda siempre usar print y señalar la acción que se ejecuta para saber como está llegando la respuesta, es decir: print(“función NOMBRE”, name)

Puedes utilizar el método int() en Flask para convertir una parte de la URL en un valor numérico que luego puedas usar para repetir una palabra específica la cantidad de veces indicada

#Crea un patrón y una función de URL que puedan manejar los siguientes ejemplos (PISTA: int() puede ser útil Por ejemplo, int("35") devuelve 35):

# localhost:5000/repeat/35/hello: haz que diga "hola" 35 veces

# localhost:5000/repeat/80/bye: haz que diga "adiós" 80 veces

# localhost:5000/repeat/99/dogs: haz que diga "perros" 99 veces

@app.route('/repeat/<int:num>/<word>')

def repeat\_word(num, word):

    return f"{word} " \* num

#aca le pongo http://localhost:5000/repeat/35/hello

HTML Y FLASK

Crear una carpeta llamada templates y cuando uso render templates, le digo que archivo dentro de esa carpeta quiero usar y ya

from flask import Flask, render\_template # agregado render\_template!

app = Flask(\_\_name\_\_)

@app.route('/')

def hola\_mundo():

# En lugar de devolver una cadena,

# devolveremos el resultado del método render\_template, pasando el nombre de nuestro archivo HTML

return render\_template('index.html')

if \_\_name\_\_=="\_\_main\_\_":

app.run(debug=True)

Motores de plantillas

* Pasar datos al html es contextualizar
* from flask import Flask, render\_template
* app = Flask(\_\_name\_\_)
* @app.route('/')
* *def* index():
* return render\_template("index.html", phrase="hola", times=5) # ¡Fíjate en los 2 nuevos argumentos nombrados!
* if \_\_name\_\_=="\_\_main\_\_":
* app.run(debug=True)

recién va a tomar el contexto con dos llaves en el html con el nombre

El render no entiende corchetes entonces si quiero un dato especifico de algún algo, debo usar puntos.

Renderizar datos en una plantilla

Ahora, ¿cómo usamos esos datos en HTML? Hay 2 entradas especiales que podemos usar para insertar código similar a Python en nuestras plantillas de Flask.

{{ alguna variable }}

{% alguna expresión %} {% endfor %}

En el código anterior, usamos las diferentes etiquetas para incrustado para mostrar algunas de nuestras variables, insertar un bucle for y hacer una verificación condicional con una declaración if en nuestra plantilla HTML. Es especialmente importante ver cómo usamos los valores que pasamos a nuestra plantilla desde nuestro archivo de servidor en las etiquetas embed.

Estas etiquetas nos permiten controlar qué se renderiza (sentencias if), cuántas veces se renderiza algo (bucle for) e imprimir valores en nuestro html renderizado.

Aunque técnicamente puedes hacer mucha lógica en tus plantillas, debes intentar limitar esa lógica tanto como sea posible. Haz la mayor parte de tu lógica en tu código Python. Si pones demasiada lógica en tus plantillas, puedes ralentizar el tiempo de respuesta de tu servidor.

Como mencionamos anteriormente, Flask usa un motor de plantillas llamado Jinja. Jinja tiene muchas funciones integradas excelentes que nos permiten poner información dinámica en páginas HTML. Consulta la documentación de Jinja aquí: <https://jinja.palletsprojects.com/en/3.1.x/>

NOTA IMPORTANTE: El uso de comentarios HTML (<!-- -->) NO escribirá comentarios en Jinja. En su lugar, debes utilizar la sintaxis de comentarios de Jinja. <https://jinja.palletsprojects.com/en/3.1.x/templates/>

ARCHIVOS ESTÁTICOS

El contenido estático es cualquier contenido que se pueda entregar al cliente sin ser modificado, generado o procesado por el servidor. Cada marco tendrá su propia forma de servir contenido estático. Flask sirve contenido estático desde un directorio llamado static. Al igual que nuestro directorio templates, el directorio static debe llamarse static. Esta carpeta static se utilizará para servir todas tus hojas de estilo, imágenes y archivos JavaScript.

Como tener disponible archivos estáticos? Con la carpeta static de flask y dentro creo el archivo css. La sintaxis debe ser:

<!-- basado en la estructura de carpetas de la derecha -->

<!-- enlazar una hoja de estilo CSS -->

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='my\_style.css') }}">

<!-- enlazar un archivo javascript -->

<script type="text/javascript" src="{{ url\_for('static', filename='my\_script.js') }}"></script>

<!-- enlazar una imagen -->

<img src="{{ url\_for('static', filename='my\_img.png') }}">

Cual es la diferencia a como hacíamos con js? Que se pone el url for. Siempre que quiera usar un archivo de la ruta static, debo hacerlo de esa manera.

ORGANIZACIÓN

Es común crear algunas carpetas más para organizar nuestros archivos estáticos en categorías según el tipo de documento. Podemos llamarlos css, js e img y alojar los archivos correspondientes en las diferentes carpetas. Podemos reflejar estos cambios en nuestras etiquetas anteriores con los siguientes cambios en los atributos href/src. Observa la forma en que cambiamos el nombre del archivo en la función url\_for:

<!-- enlazar una hoja de estilo CSS -->

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='css/my\_style.css') }}">

<!-- enlazar un archivo javascript --> <script type="text/javascript" src="{{ url\_for('static', filename='js/my\_script.js') }}"></script>

<!-- enlazar una image -->

<img src="{{ url\_for('static', filename='img/my\_img.png') }}">

Nota: Al usar archivos estáticos, es probable que tu navegador los almacene en la memoria caché. Si estás realizando cambios en archivos estáticos y parece que no se actualizan, actualiza la página en tu navegador:ctrl + shift + r (Windows) o cmd + shift + r (Mac).

TAREA DE CHESSBOARD

CODIGO HTML

<body>

    <h1>Chessboard</h1>

    <table>

        {% for i in range(8): %} #creacion de filas por cada iteracion#

        <tr>

            {% for j in range(8): %} #cada iteracion genera una celda en la fila#

            <td class="color{{ (i+j)%2+1 }}"></td>

            {% endfor %}

        </tr>

        {% endfor %}

    </table>

</body>

EXPLICACION

la línea de código **<td class="color{{ (i+j)%2+1 }}"></td>**:

**class="color{{ (i+j)%2+1 }}"**: Aquí se establece el atributo de clase (**class**) de la celda. La clase se utiliza para aplicar estilos CSS específicos a elementos HTML. La parte **color{{ (i+j)%2+1 }}** es la parte interesante aquí:

* + **color**: Es la parte constante de la clase. Todas las celdas tendrán "color" como parte de su clase.
  + **{{ ... }}**: Esto es una interpolación de Jinja2 que permite insertar valores dinámicos dentro de la cadena. Lo que está dentro de las llaves dobles será evaluado por Jinja2 y reemplazado por su valor.
  + **(i+j)%2+1**: Esta es la expresión que se evaluará y cuyo valor se insertará en la clase. Vamos a desglosar esta expresión:
    - **(i+j)**: Esto suma los valores de **i** (valor del bucle exterior) y **j** (valor del bucle interior). Esto es necesario para obtener un número que cambie con cada celda y fila.
    - **%2**: Luego se aplica el operador módulo 2 para obtener el residuo de la división por 2. Esto básicamente alterna entre 0 y 1 para cada iteración del bucle, lo que se corresponde con las clases de colores **color1** y **color2**.
    - **+1**: Finalmente, se le suma 1 al resultado del módulo para que tengas **color1** (si el resultado del módulo es 0) o **color2** (si el resultado del módulo es 1).

Clase jueves 3 de agosto

POST FORM

La estructura en el html debe ser de un formulario y en esta tarea queda así:

<body>

    <h1>Index Page</h1>

    <h3>Create an User</h3>

    <form action="'/users" method="post">

    <label for="name">Name: </label>

    <input type="text" name="name">

    <label for="email">Email: </label>

    <input type="text" name="email">

    <input type="submit" value="create user">

    </form>

</body>

#Action: Esta es la ruta que procesará el formulario (no la que muestra el formulario, que es "/").

#Post: Nuestras opciones son GET y POST; lo más probable es que queramos que sea una solicitud POST (pero si no la configuras, el valor predeterminado es GET)

#Input: Estas son las partes del formulario que realmente recopilan datos del usuario. Consulta las opciones de tipo (<https://www.w3schools.com/tags/att_input_type.asp> ). También verifica (<https://www.w3schools.com/html/html_form_elements.asp> ) otros elementos de formulario como select (menús desplegables) y textarea. Cada elemento debe tener un valor único para su atributo de nombre.

#Enviar\_datos: Puede ser <input type='submit'> o <button>Submit</button>, pero NO <input type='button'>.

Ahora, se debe indicar que va a pasar cuando se envíen los datos, por eso el server se configura así:

    from flask import Flask, render\_template, request, redirect # solicitud agregada

@app.route('/users', methods=['POST'])

def create\_user():

    print("Got Post Info")

    print(request.form)

    # Nunca renderices una plantilla en una solicitud POST

    # En su lugar, redirigiremos a nuestra ruta de índice

    return redirect('/')

#Especificaciones de métodos permitidos [‘POST’]: Si no proporcionamos un valor para los métodos, solo se permiten solicitudes GET. Todo lo que hemos hecho hasta este punto han sido solicitudes GET, por lo que ha estado bien, pero como queremos que este método maneje solicitudes POST, debemos especificar eso. Observa que es una lista; es posible proporcionar más de un valor.

#Acceder a los datos [request.form]: El nombre que le dimos a cada entrada HTML es importante. En el lado del servidor, podemos acceder a los datos que un usuario ingresó en un campo a través del diccionario request.form proporcionando el nombre de la entrada como clave. Para ver qué hay en tu objeto de solicitud, intenta imprimir request.form.

Por último, ten en cuenta que el **tipo de cualquier cosa que ingrese a través de request.form será una "cadena" pase lo que pase.** Si quieres que ese valor se identifique como un número real, tendrás que convertir el tipo.