🚀 Fundamentos de Flask

Entornos Virtuales, Rutas y Plantillas

Python Full Stack - Clase 18

¿Qué aprenderemos hoy?

- 🔧 Configurar y usar entornos virtuales con pipenv
- 🌐 Crear nuestra primera aplicación Flask
- 🛣️ Configurar rutas y manejar parámetros en URLs
- 📄 Renderizar plantillas HTML con Jinja2
- 🎨 Trabajar con archivos estáticos (CSS, JS, imágenes)
- 💡 Enviar datos del servidor a las plantillas
- 🔄 Iterar sobre listas y diccionarios en plantillas

🔧 Entornos Virtuales

¿Qué es un Entorno Virtual?

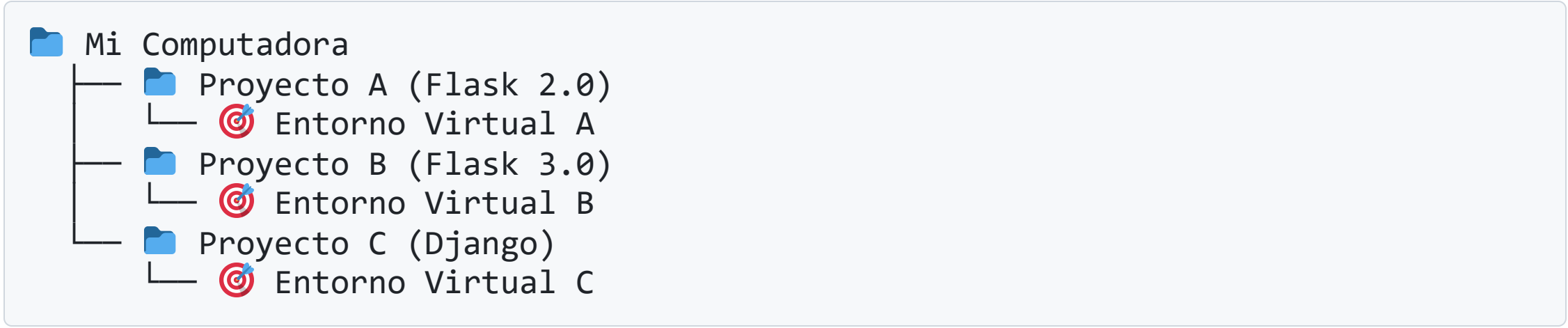
Imagina que tienes **múltiples proyectos** en tu computadora:

- **Proyecto A** necesita Flask versión 2.0
- **Proyecto B** necesita Flask versión 3.0
- **Proyecto C** necesita Django (otro framework)

¿Cómo evitamos conflictos? 🤔

Entornos Virtuales: La Solución

Un **entorno virtual** es como un **aislamiento** para cada proyecto:



Cada proyecto tiene su propio "ambiente" con sus propias dependencias.

Ventajas de los Entornos Virtuales

✅ **Aislamiento**: Cada proyecto tiene sus propias librerías
✅ **Organización**: Evita conflictos entre versiones
✅ **Portabilidad**: Fácil de compartir con otros desarrolladores
✅ **Control**: Sabes exactamente qué versiones usa tu proyecto

Instalando pipenv

pipenv es nuestra herramienta para manejar entornos virtuales.

Windows:

```
pip install pipenv
```

Mac:

```
pip3 install pipenv
```

Nota: Si recibes un error, prueba con:

```
python -m pip install pipenv
# o
python3 -m pip install pipenv
```

Creando Nuestro Primer Proyecto Flask

1. **Crea una carpeta** llamada `hola_flask`
2. **Abre VSCode** en esa carpeta
3. **Abre la terminal** desde VSCode (Terminal → Nueva Terminal)

Asegúrate de estar en la carpeta correcta:

```
cd hola_flask
```

Instalando Flask en el Entorno Virtual

Desde la carpeta `hola_flask`, ejecuta:

```
pipenv install flask
```

Esto hará **dos cosas**:

1. ✅ Crea un entorno virtual
2. ✅ Instala Flask dentro de ese entorno

Archivos Creados por pipenv

Después de ejecutar `pipenv install flask`, verás:

```
hola_flask/
  ├── Pipfile          ← Lista de paquetes instalados
  └── Pipfile.lock     ← Versiones específicas (detallado)
```

Pipfile: Archivo de configuración que muestra qué paquetes tenemos

Pipfile.lock: Versiones exactas de cada paquete (más detallado)

Activando el Entorno Virtual

Para usar el entorno virtual, debemos **activarlo**:

```
pipenv shell
```

¿Cómo saber que está activo?

- **Mac**: Verás `(hola_flask)` al inicio de la línea
- **Windows**: Verás `pipenv` en la parte superior derecha

Desactivando el Entorno Virtual

Para salir del entorno virtual:

```
exit
```

Tu terminal volverá a la normalidad.

⚠️ **Importante**: No crees un entorno virtual dentro de otro. Siempre verifica dónde estás.

¡Hola, Flask!

Creando Nuestro Primer Servidor

Dentro de la carpeta `hola_flask`, crea un archivo llamado `server.py`:

```
from flask import Flask  # Importa Flask

app = Flask(__name__)    # Crea una instancia de Flask

@app.route('/')          # Define la ruta "/"
def hola_mundo():
    return '¡Hola Mundo!'  # Retorna un mensaje

if __name__=="__main__":
    app.run(debug=True)    # Ejecuta en modo debug
```

Desglosando el Código

```
from flask import Flask
```

- Importa la clase Flask que necesitamos

```
app = Flask(__name__)
```

- Crea una instancia de Flask llamada "app"
- `__name__` le dice a Flask dónde buscar recursos

```
@app.route('/')
```

- Decorador que asocia una URL con una función
- La ruta "/" es la raíz del sitio

Más sobre el Código

`def hola_mundo():`

- Función que se ejecuta cuando alguien visita "/"
- Puede tener cualquier nombre

`return '¡Hola Mundo!'`

- Lo que se muestra en el navegador

`app.run(debug=True)`

- Inicia el servidor
- `debug=True` recarga automáticamente cuando cambias código

Ejecutando el Servidor

1. **Asegúrate** de estar en la carpeta `hola_flask`
2. **Activa** el entorno virtual: `pipenv shell`
3. **Ejecuta** el servidor:

```
python server.py
# o
python3 server.py
```

Verás algo como:

```
 * Running on http://127.0.0.1:5000
```

Probando en el Navegador

Abre tu navegador y visita:

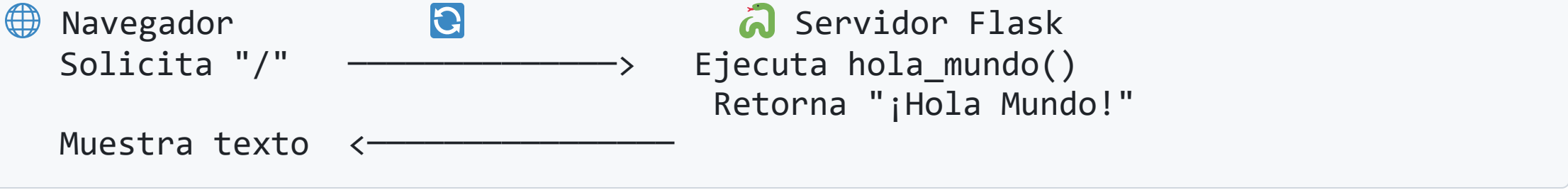
```
http://127.0.0.1:5000/
```

o simplemente:

```
http://localhost:5000/
```

¡Deberías ver "¡Hola Mundo!" en la pantalla! 🎉

¿Qué está Pasando?



1. El navegador solicita la ruta "/"
2. Flask encuentra la función asociada (`hola_mundo()`)
3. La función retorna un texto
4. Flask envía ese texto al navegador
5. El navegador lo muestra

 Rutas en Flask

¿Qué son las Rutas?

Las **rutas** son direcciones que le dicen al servidor **qué hacer**.

Cada ruta tiene dos partes:

1. **Método HTTP** (GET, POST, PUT, DELETE)
2. **URL** (la dirección)

Ejemplo:

- `GET /usuarios` → Ver lista de usuarios
- `POST /usuarios` → Crear un nuevo usuario

Creando Rutas Básicas

Ya creamos la ruta raíz "/". Agreguemos más:

```
@app.route('/exito')
def exito():
    return "¡Éxito!"
```

Ahora si visitas `http://127.0.0.1:5000/exito`, verás "¡Éxito!"

Rutas Variables: El Problema

Imagina que quieres saludar a diferentes personas:

```
@app.route('/saludo/valeria')
def saludo_valeria():
    return "¡Hola Valeria!"

@app.route('/saludo/alfredo')
def saludo_alfredo():
    return "¡Hola Alfredo!"

@app.route('/saludo/nestor')
def saludo_nestor():
    return "¡Hola Nestor!"
```

¡Esto es muy repetitivo! 😫

Rutas Variables: La Solución

Usa **parámetros en la URL**:

```
@app.route('/saludo/<nombre>')
def saludo(nombre):
    print(nombre)  # Imprime en la terminal
    return f'¡Hola {nombre}!'
```

Ahora puedes visitar:

- `/saludo/valeria` → "¡Hola Valeria!"
- `/saludo/alfredo` → "¡Hola Alfredo!"
- `/saludo/cualquier-nombre` → "¡Hola cualquier-nombre!"

Múltiples Parámetros

Puedes tener varios parámetros en una ruta:

```
@app.route('/color/<nombre>/<color>')
def color_favorito(nombre, color):
    return f'Hola {nombre}, tu color favorito es el {color}'
```

Visita: `/color/María/azul`

Resultado: "Hola María, tu color favorito es el azul"

Convertidores de Tipo

Por defecto, los parámetros son **cadenas de texto**. Pero podemos convertirlos:

```
@app.route('/saludo/<nombre>/<int:num>')
def hola_cantidad(nombre, num):
    return f'¡Hola {nombre}!' * num
```

Diferencias:

- `<nombre>` → Siempre es texto
- `<int:num>` → Se convierte a número entero

Ejemplo de Convertidor

```
@app.route('/saludo/<nombre>/<int:num>')
def hola_cantidad(nombre, num):
    return f'¡Hola {nombre}!' * num
```

- `/saludo/Nestor/5` → "¡Hola Nestor!¡Hola Nestor!¡Hola Nestor!¡Hola Nestor!¡Hola Nestor!"
- `/saludo/Nestor/noNumero` → Error 404 (no puede convertir "noNumero" a entero)

💪 Ejercicio: Comprender el Enrutamiento

Crea un archivo `server.py` con:

1. Ruta `/` que muestre "¡Hola desde Flask!"
2. Ruta `/ruta` que muestre "¿Qué ruta estás buscando?"
3. Ruta `/bienvenido/<nombre>` que muestre "Bienvenid@ a esta ruta [nombre]"
4. Ruta `/repite/<int:num>/<palabra>` que repita la palabra "num" veces

Pistas:

- Usa `f'texto {variable}'` para formatear strings
- Recuerda multiplicar strings: `"hola" * 3` = "holaholahola"

⏱️ **Tiempo:** 15 minutos

Renderizar Vistas

De Texto a HTML

Hasta ahora solo retornamos **texto simple**. Pero queremos mostrar **páginas HTML completas** con diseño y estilo.

Flask usa plantillas para esto.

Estructura de Carpetas

Para usar plantillas, Flask busca una carpeta llamada `templates`:

```
hola_flask/
  ├── server.py
  └── templates/          ← Carpeta para HTML
      └── index.html
```

⚠️ **Importante**: La carpeta debe llamarse exactamente `templates` y estar al mismo nivel que `server.py`.

Creando Nuestra Primera Plantilla

Dentro de `templates/index.html`:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>¡Hola Flask!</title>
</head>
<body>
    <h1>Bienvenido a tu plantilla</h1>
</body>
</html>
```

Renderizando la Plantilla

En `server.py`, importa `render_template`:

```
from flask import Flask, render_template  # Agregamos render_template

app = Flask(__name__)

@app.route('/bienvenido')
def bienvenido():
    return render_template('index.html')  # Renderiza el HTML

if __name__=="__main__":
    app.run(debug=True)
```

¿Qué Hace render_template?

Visita `http://127.0.0.1:5000/bienvenido` y verás tu página HTML.

Motores de Plantillas

Enviar Datos al HTML

Hasta ahora mostramos HTML **estático**. Pero queremos mostrar datos **dinámicos** del servidor.

Ejemplo: Mostrar el nombre del usuario en la página.

Pasando Variables a la Plantilla

En `server.py`:

```
@app.route('/bienvenido')
def bienvenido():
    return render_template('index.html',
                           cancion="dale a tu cuerpo alegría macarena",
                           repite=5)
```

Ahora enviamos dos variables:

- `cancion` → texto
- `repite` → número

Sintaxis de Jinja2

Jinja2 es el motor de plantillas de Flask. Usa estas sintaxis:

- `{{ variable }}` → Muestra el valor de una variable
- `{% expresión %}` → Ejecuta lógica (if, for, etc.)
- `{# comentario #}` → Comentarios (no se muestran)

Mostrando Variables en HTML

En `templates/index.html`:

```
<h1>Bienvenido a tu plantilla</h1>

<p>Canción: {{ cancion }}</p>
<p>Repeticiones: {{ repite }}</p>
```

Resultado:

```
Canción: dale a tu cuerpo alegría macarena
Repeticiones: 5
```

Bucles en Plantillas

Podemos usar `for` en las plantillas:

```
{# Repetimos la canción la cantidad de veces indicada #}
{% for x in range(repite) %}
    <h3>{{ cancion }}</h3>
{% endfor %}
```

Esto mostrará la canción 5 veces (porque `repite=5`).

Condicionales en Plantillas

También podemos usar `if`:

```
{# Revisamos si la canción incluye "macarena" #}
{% if "macarena" in cancion %}
    <h4>¡Eh macarena!</h4>
{% endif %}
```

Si la canción contiene "macarena", mostrará el mensaje.

Ejemplo Completo

server.py:

```
@app.route('/bienvenido')
def bienvenido():
    return render_template('index.html',
                           cancion="dale a tu cuerpo alegría macarena",
                           repite=5)
```

index.html:

```
<h1>{{ cancion }}</h1>
{% for x in range(repite) %}
    <p>{{ cancion }}</p>
{% endfor %}
{% if "macarena" in cancion %}
    <h2>¡Eh macarena!</h2>
{% endif %}
```

💡 Buenas Prácticas

⚠️ **No abuses de la lógica en plantillas**

✅ **Mejor**: Lógica compleja en Python (server.py)
✅ **Bien**: Mostrar datos y loops simples en HTML
❌ **Evitar**: Cálculos complejos en plantillas

¿Por qué? Mantiene el código más fácil de mantener y entender.

💪 Ejercicio: Juego de Pelotas

¡Hora de practicar!

Crea un proyecto Flask nuevo:

1. Ruta `/juego` que muestre 3 pelotas rojas (círculos)
2. Ruta `/juego/<int:x>` que muestre "x" pelotas rojas
3. Ruta `/juego/<int:x>/<color>` que muestre "x" pelotas del color indicado

Pistas:

- Usa CSS inline en la plantilla (`<style>` en `<head>`)
- Crea círculos con `border-radius: 50%`
- Usa un bucle `for` para repetir las pelotas

⏱️ **Tiempo:** 20 minutos

Archivos Estáticos

¿Qué son los Archivos Estáticos?

Los **archivos estáticos** son archivos que se envían directamente al navegador **sin modificación**:

- 📄 **CSS** (hojas de estilo)
- 🖼️ **Imágenes** (PNG, JPG, SVG)
- 📜 **JavaScript** (archivos .js)

Flask busca estos archivos en una carpeta llamada `static`

Estructura con Archivos Estáticos

```
hola_flask/
   ├── server.py
   ├── templates/
   │    └── index.html
   └── static/                ← Carpeta para archivos estáticos
        ├── css/
        │    └── estilo.css
        ├── js/
        │    └── script.js
        └── img/
             └── logo.png
```

Enlazando CSS en Plantillas

Para enlazar archivos estáticos, usa `url_for()`:

```
<!-- Enlazar una hoja de estilo CSS -->
<link rel="stylesheet" type="text/css"
      href="{{ url_for('static', filename='css/estilo.css') }}">
```

`url_for('static', filename='...')` genera la ruta correcta automáticamente.

Enlazando JavaScript

```
<!-- Enlazar un archivo JavaScript -->
<script type="text/javascript"
        src="{{ url_for('static', filename='js/script.js') }}">
</script>
```

Enlazando Imágenes

```
<!-- Mostrar una imagen -->
<img src="{{ url_for('static', filename='img/logo.png') }}"
     alt="Logo">
```

Ejemplo Completo

index.html:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <link rel="stylesheet"
          href="{{ url_for('static', filename='css/estilo.css') }}">
</head>
<body>
    <h1>Mi Página</h1>
    <img src="{{ url_for('static', filename='img/logo.png') }}">
    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

⚠️ Caché del Navegador

Si cambias archivos estáticos y no ves los cambios:

Actualización forzada:

- **Windows**: `Ctrl + Shift + R`
- **Mac**: `Cmd + Shift + R`

Esto fuerza al navegador a descargar los archivos nuevamente.

¡Hora de practicar!

Crea un proyecto Flask:

1. Ruta `/loteria` que muestre un tablero 4x4 con colores alternos (azul, amarillo, rosa)
2. Ruta `/loteria/<int:x>` que muestre un tablero 4x"x" filas
3. Usa archivos estáticos para el CSS

Colores sugeridos:

- Azul: `#287fe4`
- Amarillo: `#eadb00`
- Rosa: `#dda8c4`

Pistas:

- Usa `for` anidados para crear filas y columnas
- Alterna colores con condiciones `if`

Más sobre Renderizado

Enviar Listas a Plantillas

Podemos enviar **listas y diccionarios** a las plantillas:

```
@app.route('/listas')
def renderizar_listas():
    listado_estudiantes = [
        {'nombre': 'Florencia', 'edad': 25},
        {'nombre': 'Valentina', 'edad': 30},
        {'nombre': 'José', 'edad': 27},
        {'nombre': 'Patricio', 'edad': 21}
    ]
    return render_template('listas.html',
                           numeros=[7, 15, 22],
                           estudiantes=listado_estudiantes)
```

Iterar sobre Listas

En la plantilla `listas.html`:

```
<h2>Números aleatorios</h2>
{% for n in numeros %}
    <p>{{ n }}</p>
{% endfor %}
```

Esto mostrará:

```
7
15
22
```

Iterar sobre Listas de Diccionarios

```
<h2>Listado de Alumnos</h2>
<ul>
    {% for alumno in estudiantes %}
        <li>{{ alumno['nombre'] }} - {{ alumno['edad'] }}</li>
    {% endfor %}
</ul>
```

Resultado:

- Florencia - 25
- Valentina - 30
- José - 27
- Patricio - 21

Acceder a Diccionarios

Hay dos formas de acceder a valores en diccionarios:

```
{{ alumno['nombre'] }}    <!-- Usando corchetes -->
{{ alumno.nombre }}        <!-- Usando punto (alternativa) -->
```

Ambas funcionan, pero `['clave']` es más explícito.

💪 Ejercicio: Tabla de Países

¡Hora de practicar!

Crea un proyecto Flask:

1. Crea una lista de diccionarios con países y capitales
2. Envía la lista a una plantilla
3. Muestra los datos en una tabla HTML

Países sugeridos:

- Argentina - Buenos Aires
- Brasil - Brasilia
- Chile - Santiago de Chile
- Colombia - Bogotá

Pistas:

- Usa `<table>`, `<tr>`, `<td>` para la tabla
- Itera sobre la lista con `{% for %}`

⏱️ **Tiempo:** 15 minutos

Resumen de Conceptos Clave

✅ **Entornos virtuales**: Aislan dependencias por proyecto
✅ **Rutas**: Asocian URLs con funciones
✅ **Rutas variables**: Reciben parámetros desde la URL
✅ **Plantillas**: Archivos HTML con datos dinámicos
✅ **Jinja2**: Motor de plantillas de Flask
✅ **Archivos estáticos**: CSS, JS, imágenes en carpeta `static`

Lo Más Importante

🎯 Flask es simple pero poderoso

- Fácil de empezar
- Escalable para proyectos grandes

🎯 Organización es clave

- `templates/` para HTML
- `static/` para CSS/JS/imágenes

🎯 Plantillas hacen el trabajo dinámico

- `{{ }}` para mostrar datos
- `{% %}` para lógica

Recursos para Seguir Aprendiendo

📚 **Documentación oficial:**

- Flask: https://flask.palletsprojects.com/
- Jinja2: https://jinja.palletsprojects.com/

💡 **Próximos pasos:**

- Formularios y métodos POST
- Sesiones y cookies
- Conexión con bases de datos

Próximos Pasos

En la siguiente clase veremos:

- 📝 **Formularios** (enviar datos al servidor)
- 🔄 **Métodos POST** (crear y actualizar datos)
- 🔐 **Sesiones** (mantener estado del usuario)
- 🗄️ **Bases de datos** (almacenar información)

Práctica para Casa

🏠 **Proyecto sugerido:**

Crea una aplicación Flask que:

- Tenga una página de inicio con tu información
- Una página de "Sobre mí" con lista de habilidades
- Una página de contacto (por ahora solo mostrar HTML)
- Use archivos estáticos para el diseño

Objetivo: Practicar rutas, plantillas y archivos estáticos.

Consejos Finales

✅ **Siempre activa el entorno virtual** antes de trabajar
✅ **Usa** `debug=True` solo en desarrollo (no en producción)
✅ **Organiza tus archivos** desde el principio
✅ **Practica creando proyectos pequeños** antes de grandes

¡La práctica hace al maestro! 💪

🎉 ¡Felicidades!

Ya sabes crear aplicaciones web con Flask

Has aprendido los fundamentos de desarrollo web backend

❓ Preguntas

¿Alguna duda sobre Flask, rutas o plantillas?

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con formularios y POST!

No olvides completar todos los ejercicios 📝