# Instalar Flask

**Instalar el entorno de virtual :**

pasos

1. pip install virtualenv

2. virtualenv "env"→nombre del entorno

3. env\Scripts\activate

4. imprime en pantalla el nombre del entorno que seria (env)

si no funciona abre el powershell

pasos

1. Get-ExecutionPolicy -list

2. Set-ExecutionPolicy RemoteSigned -Force

**Instalar pip install Flask:**

pasos

1. pip install flask

# Escribe tu primer Hola mundo con Flask.

Creas un archivo de Python y escribir el siguiente código:

from flask import Flask #→ es obligatorio llamar la libreria

app = Flask(\_\_name\_\_)#→ Activamos la dependencia

@app.route("/") #→Ruta Url "Creamos una ruta"

def hello\_world():#→Funcion de salida"

    return "<p>Hello, World!</p>"

if \_\_name\_\_ == '\_\_main\_\_':#→Recomiendo poder ejecutar el servidor "

    app.run()

# Crear Rutas o paginas

Simplemente debemos copiar @app.route y luego hacer la función con la salida.

from flask import Flask #→ es obligatorio llamar la libreria

app = Flask(\_\_name\_\_)#→ Activamos la dependencia

@app.route("/") #→Ruta Url "Creamos una ruta"

def hello\_world():#→Funcion de salida"

    return "<p>Hello, World!</p>"

#creamos mas rutas

@app.route("/Hola") #→Ruta Url "Creamos la ruta 2"

def ruta2():#→Funcion de salida"

    return "<p>Hello, RUTA 2!</p>"

if \_\_name\_\_ == '\_\_main\_\_':#→Recomiendo poder ejecutar el servidor "

    app.run()

# Activar el modo debug en Flask

poder ver los cambios mientras la api esta desplegada

if \_\_name\_\_ == '\_\_main\_\_':#→Recomiendo poder ejecutar el servidor "

    app.run(debug = True) #→ agregamos el modo debug "

# Activar varias rutas en un index

Para que la pagina principal sea

app = Flask(\_\_name\_\_)#→ Activamos la dependencia

@app.route("/") #→Ruta Url "Creamos una ruta"

@app.route("/main") #→Ruta Url "creamos varias rutas"

def hello\_world():#→Funcion de salida"

    return "<p>Hello, World!</p>"

# enviar variable argumentos

Hacemos una práctica de enviar parámetros podrás ver el curso “ Flask - Construye aplicaciones web profesionales con Python” en la clase “10. Variables en rutas” en el minuto “0:20”

#creamos mas rutas

@app.route("/Hola/<name>") #→Ruta Url "Creamos la ruta 2"

def ruta2(name):#→Funcion de salida"

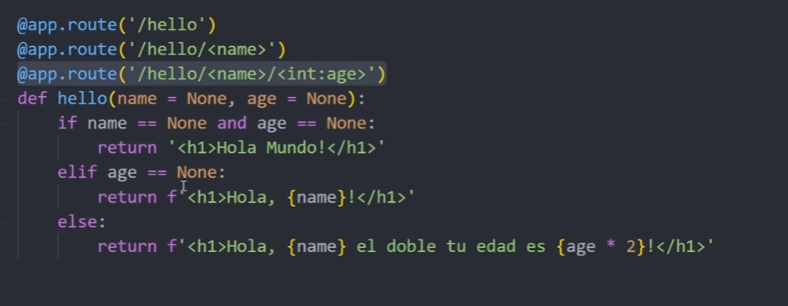
    return f"<p>Hello, {name}!</p>"

asi se vera en la web :

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Como pasarle los valores , y además hacer unos condicionales



# Plantillas html

debes crear una carpeta en la ruta del proyecto llamada templade, creamos el archivo html e inportamos una nueva librería :

from flask import Flask,render\_template #→ es obligatorio llamar la libreria

luego creamos un , creamos una ruta de esta manera :

#creamos mas rutas con Html

@app.route("/html") #→Ruta Url "llamamos nuestro html "

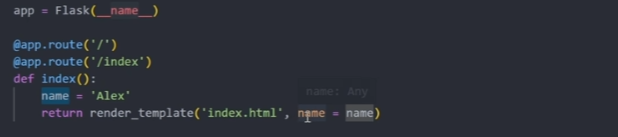
def html():#→Funcion de salida"

    return render\_template("Index.html")

if \_\_name\_\_ == '\_\_main\_\_':#→Recomiendo poder ejecutar el servidor "

    app.run(debug = True) #→ agregamos el modo debug "

podemos pasarle ejemplo variables , y todo con argumentos esa parte la podrás encontrarla , ver el curso “ Flask - Construye aplicaciones web profesionales con Python” en la clase “14. Creación de plantillas” en el minuto “3:00”:



Y en la plantilla índex como agregamos name:



# Lógica a las plantillas html como IF,BUCLES y demás

Para tener esta información te cuento franklin del futuro que busques la sección del curso “ Flask - Construye aplicaciones web profesionales con Python” de Udemy :

Imagen que contiene Interfaz de usuario gráfica

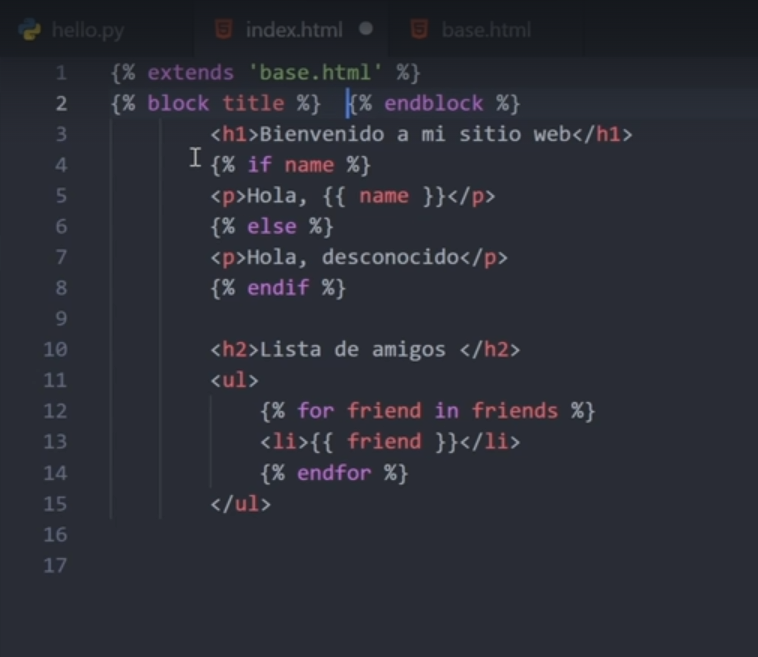
Descripción generada automáticamente

1. llamar una función de desde HTML :

{{url\_for('registro')}}

# Html Base – Herencia de plantilla

Para tener esta información te cuento franklin del futuro que busques la sección del curso “ Flask - Construye aplicaciones web profesionales con Python” de Udemy , en la clase “16. Uso de herencia de plantillas” en el minuto “0:00” donde te explicaran como hacer un html , base y no tengas que reescribir código.



Ejemplo de archivo base :

Nota: aquí implementamos boostrap para el diseño.

{% block html %}

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title></title>

    <!-- Bootstrap CSS -->

    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

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

<body class="fondo">

{% endblock %}

<!-- Contenido-->

{% block contenido %}

{% endblock %}

<!-- /Contenido-->

<!-- Mensaje error -->

{%for massage in get\_flashed\_messages() %}

<div class="container mt-3">

  <div class="row">

    <div class="col-md-6 mx-auto">

      <div class="alert alert-danger" role="alert">

        {{ massage }}

      </div>

    </div>

  </div>

</div>

{% endfor %}

<!-- /Mensaje error -->

{% block footer %}

    <!-- Bootstrap JS -->

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

{% endblock %}

Para llamar la plantilla base :

{% extends 'base.html' %}

{% block contenido %}

<!—aqui pones el contenido del html -->

{% endblock %}

Para hacer mensaje de error en el Dom html:

<!-- Mensaje error -->

{%for massage in get\_flashed\_messages() %}

<div class="container mt-3">

  <div class="row">

    <div class="col-md-6 mx-auto">

      <div class="alert alert-danger" role="alert">

        {{ massage }}

      </div>

    </div>

  </div>

</div>

{% endfor %}

<!-- /Mensaje error -->

# API-REST con Flask

1. Es instalar la librería en el entorno
2. se hace los Model , modificar la tabla dependiendo a tus necesidades:

from flask import Flask ,request,jsonify

from flask\_sqlalchemy import SQLAlchemy

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///Students.db'

db=SQLAlchemy(app)

class Student(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

“@Nombre de la columna“ = db.Column(db.String(50), nullable=False)

    def serialize(self):

        return{

            'id':self.id,

            '“Nombre de la columna“ ':self.name,

          }

3.se crea una función para la creación de todas la tabla :

with app.app\_context():

    db.create\_all()

4.para crear un metodo get obteniendo las tablas, el ejemplo es con una tabla llamada Student :

@app.route("/Student" ,methods =['GET'])

def get\_Student():

    Students=Student.query.all()

    return jsonify({'Student':[Student.serialize() for Student in Students ]})

5.para hacer un post :

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

def Create\_Student():

    data =request.get\_json()

    student = Student(

        “@Nombre de la columna“ = data['name'],

    db.session.add(student)

    db.session.commit()

    return jsonify({'message':'Contacto registrado con exito ','Student':Student.serialize()})

6.para obtener un modificar datos :

@app.route('/Student/<int:id>', methods = ['PUT', 'PATCH'])

def edit\_Student(id):

    Students = Student.query.get\_or\_404(id)

    data = request.get\_json()

    if '“@Nombre de la columna name' in data:

        Students.“@Nombre de la columna = data['name']

    # Guardar los cambios en la base de datos

    db.session.commit()

  return jsonify({'message':'Contacto actualizado con éxito', 'contact':

Students.serialize()})

6.Como eliminar la clase de ejemplo es Student :

@app.route('/Student/<int:id>', methods = ['DELETE'])

def delete\_Student(id):

    Students = Student.query.get(id)

    if not Students:

        return jsonify({'message':'estudiante no encontrado'}), 404

    db.session.delete(Students)

    db.session.commit()

    return jsonify({'message':'estudiante eliminado con éxito'})

7.para finalizar mostramos todo el proyecto:

from flask import Flask ,request,jsonify

from flask\_sqlalchemy import SQLAlchemy

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///Students.db'

db=SQLAlchemy(app)

class Student(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String(50), nullable=False)

    lastName = db.Column(db.String(50), nullable=False)

    email = db.Column(db.String(50), nullable=False)

    codigoPhone= db.Column(db.String(11), nullable=False)

    phone= db.Column(db.String(11), nullable=False)

    password= db.Column(db.String(50), nullable=False)

    def serialize(self):

        return{

            'id':self.id,

            'name':self.name,

            'lastName':self.lastName,

            'email':self.email,

            'codigoPhone':self.codigoPhone,

            'phone':self.phone,

            'password':self.password

        }

with app.app\_context():

    db.create\_all()

@app.route("/Student" ,methods =['GET'])

def get\_Student():

    Students=Student.query.all()

    return jsonify({'Student':[Student.serialize() for Student in Students ]})

@app.route('/Student/<int:id>', methods = ['GET'])

def get\_StudentID(id):

    Students = Student.query.get(id)

    if not Students:

        return jsonify({'message':'Contacto no encontrado'}), 404

    return jsonify(Students.serialize())

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

def Create\_Student():

    data =request.get\_json()

    student = Student(

        name = data['name'],

        lastName = data['lastName'],

        email = data['email'],

        codigoPhone = data['codigoPhone'],

        phone = data['phone'],

        password = data['password'])

    db.session.add(student)

    db.session.commit()

    return jsonify({'message':'Contacto registrado con exito ','Student':Student.serialize()})

@app.route('/Student/<int:id>', methods = ['PUT', 'PATCH'])

def edit\_Student(id):

    Students = Student.query.get\_or\_404(id)

    data = request.get\_json()

    if 'name' in data:

        Students.name = data['name']

    if 'lastName' in data:

        Students.lastName = data['lastName']

    if 'email' in data:

        Students.email = data['email']

    if 'codigoPhone' in data:

        Students.codigoPhone = data['codigoPhone']

    if 'phone' in data:

        Students.phone = data['phone']

    if 'password' in data:

        Students.phone = data['password']

    # Guardar los cambios en la base de datos

    db.session.commit()

    return jsonify({'message':'Contacto actualizado con éxito', 'contact': Students.serialize()})

@app.route('/Student/<int:id>', methods = ['DELETE'])

def delete\_Student(id):

    Students = Student.query.get(id)

    if not Students:

        return jsonify({'message':'estudiante no encontrado'}), 404

    db.session.delete(Students)

    db.session.commit()

    return jsonify({'message':'estudiante eliminado con éxito'})

@app.route("/Cursos", methods =['GET'])

def Cursos():

    return "<p>Cursos</p>"

@app.route("/Certificados")

def Certificados():

    return "<p>Certificados</p>"

Como auteticarse con firebase= <https://www.youtube.com/watch?v=HltzFtn9f1c>

* Herramientas necesarias para el despliegue: Git, Github y PythonAnywhere

Texto

Descripción generada automáticamente

# Enviar datos de un formulario en Flask

Se deben tener encuenta los id del html en el form por ejemplo aquí tenemos el formulario en html :

 <form class="form-floating" action="" method="post">

        <div class="mb-3">

        <label class="text-start text-primary" for="items"><strong>Código:</strong></label>

        <input class="input-group-text" type="text" id="items" name="items" required>

        </div>

        <div class="mb-3">

        <label class="text-start text-primary" for="articulo"><strong>Artículo:</strong></label>

        <input class="input-group-text" type="text" id="articulo" name="articulo" required>

        </div>

        <div class="mb-3">

        <label class="text-start text-primary" for="ctd\_stickers"><strong>Cantidad de etiquetas a Imprimir:</strong></label>

        <input class="input-group-text" type="number" id="ctd\_stickers" name="ctd\_stickers" >

        </div>

        <div class="mb-3">

        <label class="text-start text-primary" for="ctd\_altura"><strong>Altura a Imprimir:</strong></label>

        <input class="input-group-text" type="number" id="ctd\_altura" name="ctd\_altura" required value="200">

        </div>

        <div class="mb-3">

        <label class="text-start text-primary" for="ctd\_ancho"><strong>Ancho a Imprimir:</strong></label>

        <input class="input-group-text" type="number" id="ctd\_ancho" name="ctd\_ancho" required value="150">

        </div>

        <input class="btn btn-info" type="submit" value="Convertir a etiquetas">

      </form>

Franklin del futuro recuerda tomar los nombres de id y ahora mira la función en Python :

def StikerSmall():

    items=request.form['items']

    articulo=request.form['articulo']

    ctd\_stickers=request.form['ctd\_stickers']

    ctd\_altura=request.form['ctd\_altura']

    ctd\_ancho=request.form['ctd\_ancho']

    out= db.stikersmall(items,articulo,ctd\_stickers,ctd\_altura,ctd\_ancho)

    return out

# Adjuntar archivos en Flask

Mira esta excelente referencia en este video en youtube explica paso a paso

Link: <https://www.youtube.com/watch?v=Yyul17TBHrE&ab_channel=RoCode>

Lo primero que debes hacer es agregar esta librerías :

import os

from werkzeug.utils import secure\_filename

paso dos agregar y la este código y onfiguracion y crea una carpeta en el respositorio del proyecto:

app.config['UPLOAD\_FOLDER'] = 'uploads/'  # Directorio donde se guardarán los archivos subidos

ALLOWED\_EXTENSIONS= set(['png','jpg','pdf'])

Luego vez crea la función que captura el form :

soportes1=request.files['soportes1']

print(soportes1.filename)

soportes1.save(os.path.join(app.config['UPLOAD\_FOLDER'],soportes1.filename))

así queda en html :

<div class="form-floating mb-3"></div>

        <label for="soportes">Carta de solicitud de cesantías donde se especifique, motivo de retiro, monto y fondo:</label>

        <input class="form-control rounded-3" type="file" id="soportes1" name="soportes1" accept=".jpg .pdf .png">

        <br>

# Seguridad en el login Flask

Los usuarios no podrán acceder a ningún endpoind si lo tenemos asegurado sigue el paso a paso para implementarlo.

Aquí tienes un **paso a paso** detallado para implementar autenticación en tu aplicación Flask desde cero usando Flask-Login:

**1. Instalar las dependencias necesarias**

Asegúrate de que tienes instalados los paquetes necesarios para manejar la autenticación y la seguridad de contraseñas.

bash

Copiar código

pip install flask flask-login werkzeug flask-sqlalchemy

**2. Configurar la aplicación Flask**

Crea el archivo app.py y comienza configurando tu aplicación Flask. Asegúrate de tener una clave secreta y la configuración necesaria para tu base de datos.

python

Copiar código

from flask import Flask, render\_template, redirect, url\_for, request, flash

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, UserMixin, login\_user, logout\_user, login\_required, current\_user

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

app.secret\_key = 'clave\_secreta' # Cambia esto por una clave segura

# Configuración de la base de datos

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///users.db'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

# Inicializar la base de datos

db = SQLAlchemy(app)

# Inicializar el gestor de inicio de sesión

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login' # Redirigir a la página de login si no está autenticado

**3. Crear el modelo de usuario**

Define un modelo de usuario que implemente UserMixin para que funcione con Flask-Login. Este modelo manejará las credenciales de los usuarios.

python

Copiar código

from werkzeug.security import generate\_password\_hash, check\_password\_hash

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

email = db.Column(db.String(100), unique=True)

password = db.Column(db.String(100))

def \_\_init\_\_(self, email, password):

self.email = email

self.password = generate\_password\_hash(password) # Almacenar contraseña hasheada

**Nota:** Las contraseñas se almacenan de manera segura utilizando hashing.

**4. Crear la base de datos**

Antes de continuar, asegúrate de crear la base de datos. Ejecuta este código en un entorno Python interactivo o añádelo temporalmente en tu archivo app.py:

python

Copiar código

with app.app\_context():

db.create\_all() # Crear la tabla de usuarios

**5. Cargar el usuario actual**

Flask-Login necesita una función para cargar el usuario actual basado en su ID de sesión. Define la función user\_loader para hacerlo:

python

Copiar código

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id)) # Buscar usuario por ID

**6. Crear las vistas de inicio de sesión y cierre de sesión**

Implementa las rutas para el **inicio de sesión** y **cierre de sesión** utilizando Flask-Login.

python

Copiar código

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

def login():

if request.method == 'POST':

email = request.form['email']

password = request.form['password']

user = User.query.filter\_by(email=email).first()

if user and check\_password\_hash(user.password, password):

login\_user(user) # Iniciar sesión

return redirect(url\_for('dashboard')) # Redirigir a una página protegida

else:

flash('Credenciales incorrectas, intenta de nuevo.')

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user() # Cerrar sesión

return redirect(url\_for('login'))

La vista de **inicio de sesión** verifica el correo y la contraseña del usuario. Si son correctos, login\_user() inicia la sesión. En la vista de **cierre de sesión**, logout\_user() cierra la sesión.

**7. Crear vistas protegidas con login\_required**

Usa el decorador @login\_required para proteger las rutas que solo deben estar disponibles para usuarios autenticados.

python

Copiar código

@app.route('/dashboard')

@login\_required

def dashboard():

return f'Bienvenido {current\_user.email} al tablero.'

Si un usuario intenta acceder a /dashboard sin estar autenticado, será redirigido automáticamente a la página de login.

**8. Crear el formulario de registro de usuario**

También es útil tener un formulario para que los nuevos usuarios se registren. Implementa una vista para **registrar nuevos usuarios**:

python

Copiar código

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

def register():

if request.method == 'POST':

email = request.form['email']

password = request.form['password']

# Verificar si el usuario ya existe

existing\_user = User.query.filter\_by(email=email).first()

if existing\_user is None:

new\_user = User(email=email, password=password)

db.session.add(new\_user)

db.session.commit()

flash('Usuario creado con éxito, ahora puedes iniciar sesión.')

return redirect(url\_for('login'))

flash('El usuario ya existe, intenta con otro correo.')

return render\_template('register.html')

**9. Plantillas HTML**

Crea las plantillas **login.html** y **register.html** para manejar el formulario de login y registro.

**login.html**:

html

Copiar código

<form method="POST" action="{{ url\_for('login') }}">

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

<input type="email" name="email" required>

<label for="password">Contraseña:</label>

<input type="password" name="password" required>

<button type="submit">Iniciar sesión</button>

</form>

<a href="{{ url\_for('register') }}">¿No tienes cuenta? Regístrate</a>

**register.html**:

html

Copiar código

<form method="POST" action="{{ url\_for('register') }}">

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

<input type="email" name="email" required>

<label for="password">Contraseña:</label>

<input type="password" name="password" required>

<button type="submit">Registrarse</button>

</form>

**10. Probar la aplicación**

Ejecuta tu aplicación para probar la funcionalidad de autenticación:

bash

Copiar código

python app.py

Ve a http://localhost:5000/login para ver el formulario de inicio de sesión y a http://localhost:5000/register para registrar un nuevo usuario.

**Resumen de los pasos:**

1. **Instalar dependencias**: Instala Flask, Flask-Login y SQLAlchemy.
2. **Configurar la aplicación**: Configura la base de datos y Flask-Login.
3. **Crear el modelo de usuario**: Crea el modelo de usuario con hash de contraseñas.
4. **Cargar usuario autenticado**: Implementa la función user\_loader.
5. **Crear vistas de login/logout**: Implementa rutas de login y logout.
6. **Proteger rutas**: Usa @login\_required para proteger las vistas.
7. **Registro de usuarios**: Implementa una ruta para registrar nuevos usuarios.
8. **Crear plantillas**: Crea formularios HTML para login y registro.
9. **Ejecutar la aplicación**: Prueba la autenticación en tu aplicación Flask.

Con estos pasos, habrás implementado un sistema básico de autenticación en tu aplicación Flask utilizando Flask-Login.

# Produccion en Flask

<https://flask.palletsprojects.com/en/2.3.x/tutorial/deploy/>

llamar librería :

from waitress import serve

código para ejecutar en produccion

if \_\_name\_\_ == '\_\_main\_\_':

    # Se obtiene la configuración de debug desde el archivo config.py

    debug = config.config["debug"]

    if debug:

        logging.info("mode de debug esta True ")

        print("mode de debug esta True ")

        app.run(host="0.0.0.0",debug=debug, port=5000)

    else:

        # Inicia el servidor Flask con debug activado (según configuración) en el puerto 5000

        logging.info("mode de debug esta falso , aplicacion en producciion")

        print("mode de debug esta falso , aplicacion en producciion")

        serve(app,host="0.0.0.0", port=5000 , threads=2)

# JINJA2 Avanzado.

Escribieros las sentencias mas útiles para hacen html desde jinja:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{% set name = “franklin” %} → variables

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{%

Condicionales con html

{% if numero < 0 %}

<a> es verdadero </a>

{% else %}

<a> es falso </a>

{% endif %}

**Ejemplo con elif en Jinja:**

jinja

Copiar código

{% if numero < 0 %}

<a>El número es negativo</a>

{% elif numero == 0 %}

<a>El número es cero</a>

{% else %}

<a>El número es positivo</a>

{% endif %}

**Explicación:**

* **{% if %}**: Evalúa si la condición es verdadera.
* **{% elif %}**: Evalúa otra condición si la primera es falsa.
* **{% else %}**: Ejecuta el bloque si todas las condiciones anteriores son falsas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{% for empresa in empresas %}

<option value="{{empresa.empresa}}">{{ empresa.empresa }}</option>

For each, para datos o hacer lista en html.

{% endfor %}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Para mandar funciones por html.

<a href="{{ url\_for('admin') }}">Ir al panel de administración</a>

# Pasar parámetros a varios plantillas ala vez.

**Paso 1: Crear la aplicación Flask**

Primero, necesitas configurar una aplicación básica de Flask. Aquí es donde definimos las rutas, las sesiones y la lógica para pasar variables manualmente.

python

Copiar código

from flask import Flask, session, render\_template, redirect, url\_for

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

app.secret\_key = 'tu\_clave\_secreta' # Es importante configurar una clave secreta

**Paso 2: Ruta de inicio de sesión**

Define una ruta /login que simule el proceso de autenticación del usuario y almacene el nombre del usuario en la sesión. Esto es necesario para que luego se pueda pasar esa información a los templates.

python

Copiar código

@app.route('/login')

def login():

# Simulación de autenticación exitosa

session['username'] = 'franklin' # Asignas el nombre del usuario en la sesión

return redirect(url\_for('reporte')) # Redirige al reporte después de iniciar sesión

**Paso 3: Crear la ruta logout**

Añade una ruta para permitir que el usuario cierre sesión y se elimine la información de la sesión.

python

Copiar código

@app.route('/logout')

def logout():

session.pop('username', None) # Elimina el nombre de usuario de la sesión

return redirect(url\_for('login')) # Redirige al login después de cerrar sesión

**Paso 4: Crear las rutas que rendericen templates y pasen variables manualmente**

Para hacer esto de forma manual, tendrás que pasar explícitamente la variable username desde cada ruta al template cuando lo renderices. Por ejemplo, puedes crear rutas para reporte.html, usuarios.html y otros templates.

**Ruta para reporte.html**

python

Copiar código

@app.route('/reporte')

def reporte():

# Verifica si el usuario ha iniciado sesión

if 'username' in session:

return render\_template('reporte.html', username=session['username'])

else:

return redirect(url\_for('login')) # Si no está autenticado, redirige al login

**Ruta para usuarios.html**

python

Copiar código

@app.route('/usuarios')

def usuarios():

# Verifica si el usuario ha iniciado sesión

if 'username' in session:

return render\_template('usuarios.html', username=session['username'])

else:

return redirect(url\_for('login')) # Si no está autenticado, redirige al login

**Paso 5: Crear los templates y usar la variable username**

Ahora necesitas crear los templates (HTML) donde se mostrará la información del usuario que estás pasando manualmente desde las rutas.

**reporte.html**

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Reporte</title>

</head>

<body>

{% if username %}

<h1>Bienvenido al reporte, {{ username }}!</h1>

<p>Aquí está tu reporte personalizado.</p>

<a href="{{ url\_for('logout') }}">Cerrar sesión</a>

{% else %}

<p>No estás autenticado.</p>

{% endif %}

</body>

</html>

**usuarios.html**

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Usuarios</title>

</head>

<body>

{% if username %}

<h1>Lista de usuarios, {{ username }}!</h1>

<p>Aquí puedes ver todos los usuarios.</p>

<a href="{{ url\_for('logout') }}">Cerrar sesión</a>

{% else %}

<p>No estás autenticado.</p>

{% endif %}

</body>

</html>

**Paso 6: Probar la aplicación**

Una vez que tengas las rutas y los templates listos, ejecuta la aplicación Flask y pruébala.

bash

Copiar código

export FLASK\_APP=app.py

flask run

Accede a las siguientes rutas en tu navegador para probar:

* **/login**: Inicia sesión y establece la variable username en la sesión.
* **/reporte**: Muestra el reporte si el usuario ha iniciado sesión.
* **/usuarios**: Muestra la lista de usuarios si el usuario ha iniciado sesión.
* **/logout**: Cierra la sesión y redirige al login.

**Resumen del Proceso**

1. **Configura la clave secreta**: Esto es esencial para manejar sesiones en Flask.
2. **Define la ruta de login**: Guarda la información del usuario en la sesión.
3. **Define la ruta de logout**: Elimina la sesión cuando el usuario cierra sesión.
4. **Pasa la variable manualmente a los templates**: Desde cada ruta, pasa la variable username manualmente al llamar a render\_template.
5. **Usa la variable en los templates**: En cada archivo HTML, accede a la variable con {{ username }}.

Este enfoque **manual** te permite tener control total sobre qué variables se pasan a cada template sin necesidad de usar una solución global como context\_processor. Cualquiera que implemente este enfoque en su aplicación Flask podrá asegurarse de que la variable de sesión username esté disponible en los templates que especifique explícitamente.

**PWA en flask**

Link = https://medium.com/@tristan\_4694/how-to-create-a-progressive-web-app-pwa-using-flask-f227d5854c49