Diseño e Implementación de un Sistema MVC para la Cadena de Hoteles Hilton

Jhonatan Carvajal Bonilla

Katherin Morcillo Quiroga

Servicio Nacional de Aprendizaje

Análisis y Desarrollo de Software

Isaura María Suarez Novoa

Bogotá D.C., 16 de Julio de 2025

Tabla de contenido

[**1.** **Responsables** 4](#_Toc203685229)

[**2.** **Modulos** 4](#_Toc203685230)

[ **Módulo de Gestión de Usuarios:** 4](#_Toc203685231)

[ **Modulo Gestión de hoteles:** 4](#_Toc203685232)

[ **Modulo Gestión de categorías:** 4](#_Toc203685233)

[ **Requerimientos No Funcionales** 4](#_Toc203685234)

[**3.** **Diseño UI/UX (Próxima Fase)** 6](#_Toc203685235)

[**4.** **Lista de vistas:** 7](#_Toc203685236)

[**5.** **LISTA DE RUTAS:** 8](#_Toc203685237)

[**6.** **Contextualización** 13](#_Toc203685238)

[**7.** **Creación de vistas con Bootstrap** 16](#_Toc203685239)

[**8.** **Creación de la Base de datos** 17](#_Toc203685240)

[**9.** **Creación del Modelo Vista Controlador** 17](#_Toc203685241)

[ **Organización de carpetas** 17](#_Toc203685242)

[ **Configuración de conexión de la base de datos** 18](#_Toc203685243)

[ **Importación de datos .env** 19](#_Toc203685244)

[ **Archivo que corre la aplicación** 20](#_Toc203685245)

[ **Archivo con las librerías de importación** 20](#_Toc203685246)

[ **Activación del entorno virtual** 21](#_Toc203685247)

[ **Archivo init** 23](#_Toc203685248)

[ **Templates** 24](#_Toc203685249)

[ **static** 25](#_Toc203685250)

[ **Pagina Principal** 26](#_Toc203685251)

[ *Controlador* 26](#_Toc203685252)

[ **administración** 27](#_Toc203685253)

[ *Modelo* 27](#_Toc203685254)

[ *Controlador* 28](#_Toc203685255)

[ *Template* 29](#_Toc203685256)

[ **Categorías** 30](#_Toc203685257)

[ *Modelo* 30](#_Toc203685258)

[ *Controlador* 33](#_Toc203685259)

[ *Template* 38](#_Toc203685260)

[**10.** **FLUJO GENERAL DE LA APLICACIÓN** 40](#_Toc203685261)

[ **FLUJO DETALLADO DEL MÓDULO DE HOTELES** 41](#_Toc203685262)

# **Responsables**

|  |  |
| --- | --- |
| EQUIPO | ROL |
| **Katherine Morcillo Quiroga** | Desarrollador |
| **Jhonatan Carvajal Bonilla** | Desarrollador |

# **Modulos**

**El sistema permitirá a los usuarios:**

# **Módulo de Gestión de Usuarios:**

* 1. RF01: Registrar nuevos usuarios.
  2. RF02: Iniciar sesión.

# **Modulo Gestión de hoteles:**

* 1. RF03: Registrar hoteles.
  2. RF04: Editar hoteles registrados.
  3. RF05: Visualizar hoteles registrados.
  4. RF06: Cambiar estado a los hoteles registrados.

# **Modulo Gestión de categorías:**

* 1. RF07: Registrar categorías.
  2. RF08: Editar categorías registrados.
  3. RF09: Visualizar categorías registrados.
  4. RF10: Cambiar estado a las categorías registrados.

# **Requerimientos No Funcionales**

* + - Usabilidad
    - Seguridad
    - Rendimiento
    - Escalabilidad
    - Compatibilidad
    - Mantenibilidad
    - Accesibilidad
    - Respaldo y recuperación de datos

# **Diseño UI/UX (Próxima Fase)**

Durante esta fase se desarrollará la interfaz del sistema, que incluye: Diseño visual atractivo y responsivo,

Colorimetría y uso

* Color primario **Azul marino profundo (#0c1178)**

Autoridad, confianza, estabilidad y profesionalismo. Ideal para menús porque transmite solidez y guía visual clara.

* Color secundario **blanco humo azulado(#f7f8fa)**

Limpieza, claridad, neutralidad y modernidad. Ideal como fondo porque no distrae y aporta una sensación de amplitud y profesionalismo.

* Color de acción **azul violáceo intenso (#5d64f0)**

Riqueza, exclusividad, sofisticación y llamado a la acción.

Tipografía clara y moderna.

Paleta de colores adecuada al público objetivo. Mockups y prototipo navegable.

Codificación (Etapa Final)

La etapa final será el desarrollo web utilizando tecnologías como HTML, CSS, JavaScript y Bootstrap, enfocándonos en:

Integración de componentes responsivos. Flujo de navegación intuitivo.

Validaciones seguras.

Conexión con backend para gestión de datos.

# **Lista de vistas:**

1. Registro\_usuario
2. Inicio\_sesion
3. Dashboard
4. Editar hotel
5. registrar hotel
6. Hoteles
7. Principal
8. EditarCategoria
9. RegistroCategoria
10. vistaCategoria

Elección de patrón de diseño arquitectónico Se llego a la conclusión del patrón de diseño MVC:

* Se separa las responsabilidades
* Se separa en tres componentes que están interconectados los cuales son: los modelos, las vistas y el controlador
* Curva de aprendizaje bajo
* Facilita la organización
* Se puede mantener
* Mayor flexibilidad
* escalabilidad en el desarrollo del sistema
* Se puede separar por módulos
* Adaptable

El modelo MVC se encarga de la lógica y los datos Vista: Es la interfaz de los usuarios

Controlador: Maneja las interacciones y la comunicación entre ambos (logica) Modelo: Base de datos

# **LISTA DE RUTAS:**

**Requerimiento: Registrar usuario**

**¿Qué acción describe?**  
a) Ingresar datos solicitados para crear una cuenta en la plataforma

**¿Qué módulo está relacionado?**  
a) Registro

**¿Qué verbo identifica la acción?**  
a) Registrar

**¿Qué ruta crees que necesita?**  
a) Ruta: /registro  
b) Método: GET  
c) Qué hace: Muestra el formulario para registrar un nuevo usuario en la plataforma.

a) Ruta: /registrar\_usuario  
b) Método: POST  
c) Qué hace: Registra al usuario en el sistema y guarda la información en la base de datos.

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de usuario (user\_controller.py).

**Requerimiento: Iniciar sesión**

**¿Qué acción describe?**  
a) Pide al usuario su información de registro

**¿Qué módulo está relacionado?**  
a) Registro

**¿Qué verbo identifica la acción?**  
a) Autenticar

**¿Qué ruta crees que necesita?**  
a) Ruta: /login  
b) Método: GET y POST  
c) Qué hace: Muestra el formulario y valida credenciales del usuario. Si son correctas, inicia sesión.

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de usuario (user\_controller.py).

**Requerimiento: Cerrar sesión**

**¿Qué acción describe?**  
a) Finaliza la sesión del usuario actual

**¿Qué módulo está relacionado?**  
a) Registro

**¿Qué verbo identifica la acción?**  
a) Cerrar

**¿Qué ruta crees que necesita?**  
a) Ruta: /logout  
b) Método: GET  
c) Qué hace: Elimina los datos de sesión del usuario y redirige al login

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de usuario (user\_controller.py)

**Requerimiento: Acceder al panel de administración**

**¿Qué acción describe?**  
a) Visualiza estadísticas generales del sistema

**¿Qué módulo está relacionado?**  
a) Administración

**¿Qué verbo identifica la acción?**  
a) Visualizar

**¿Qué ruta crees que necesita?**  
a) Ruta: /administracion  
b) Método: GET  
c) Qué hace: Muestra un resumen de usuarios, hoteles y categorías.

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de administración (admin\_controller.py)

**Requerimiento: Registrar hotel**

**¿Qué acción describe?**  
a) Ingresar datos para registrar un hotel

**¿Qué módulo está relacionado?**  
a) Gestión de hoteles

**¿Qué verbo identifica la acción?**  
a) Registrar

**¿Qué ruta crees que necesita?**  
a) Ruta: /registrar\_hotel  
b) Método: GET y POST  
c) Qué hace: Muestra el formulario de registro de hotel y guarda los datos ingresados

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de hoteles (hotel\_controller.py)

**Requerimiento: Ver lista de hoteles**

**¿Qué acción describe?**  
a) Mostrar todos los hoteles registrados en el sistema

**¿Qué módulo está relacionado?**  
a) Gestión de hoteles

**¿Qué verbo identifica la acción?**  
a) Visualizar

**¿Qué ruta crees que necesita?**  
a) Ruta: /hoteles  
b) Método: GET  
c) Qué hace: Lista todos los hoteles registrados en la base de datos

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de hoteles (hotel\_controller.py)

**Requerimiento: Eliminar hoteles**

**¿Qué acción describe?**  
a) Ocultar hoteles seleccionados del listado

**¿Qué módulo está relacionado?**  
a) Gestión de hoteles

**¿Qué verbo identifica la acción?**  
a) Eliminar

**¿Qué ruta crees que necesita?**  
a) Ruta: /eliminar\_hoteles  
b) Método: POST  
c) Qué hace: Oculta uno o más hoteles del sistema sin eliminarlos físicamente

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de hoteles (hotel\_controller.py)

**Requerimiento: Editar hotel**

**¿Qué acción describe?**  
a) Modificar los datos de un hotel existente

**¿Qué módulo está relacionado?**  
a) Gestión de hoteles

**¿Qué verbo identifica la acción?**  
a) Editar

**¿Qué ruta crees que necesita?**  
a) Ruta: /editar\_hotel/<int:id>  
b) Método: GET y POST  
c) Qué hace: Muestra un formulario con los datos del hotel para editarlos y actualiza los cambios

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de hoteles (hotel\_controller.py)

**Requerimiento: Ver categorías de hoteles**

**¿Qué acción describe?**  
a) Mostrar todas las categorías existentes

**¿Qué módulo está relacionado?**  
a) Gestión de categorías

**¿Qué verbo identifica la acción?**  
a) Visualizar

**¿Qué ruta crees que necesita?**  
a) Ruta: /CategoriaHotel  
b) Método: GET  
c) Qué hace: Lista todas las categorías de hotel registradas en la base de datos

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de categorías (categoria\_controller.py)

**Requerimiento: Registrar categoría**

**¿Qué acción describe?**  
a) Ingresar los datos de una nueva categoría de hotel

**¿Qué módulo está relacionado?**  
a) Gestión de categorías

**¿Qué verbo identifica la acción?**  
a) Registrar

**¿Qué ruta crees que necesita?**  
a) Ruta: /create\_categoria  
b) Método: GET y POST  
c) Qué hace: Muestra el formulario de categoría y guarda la información en la base de datos

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de categorías (categoria\_controller.py)

**Requerimiento: Procesar acción en categorías**

**¿Qué acción describe?**  
a) Ejecutar acción de editar o eliminar sobre categorías seleccionadas

**¿Qué módulo está relacionado?**  
a) Gestión de categorías

**¿Qué verbo identifica la acción?**  
a) Procesar

**¿Qué ruta crees que necesita?**  
a) Ruta: /procesar\_accion  
b) Método: POST  
c) Qué hace: Dependiendo de la opción seleccionada, elimina o redirige para editar la categoría

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de categorías (categoria\_controller.py)

**Requerimiento: Editar categoría**

**¿Qué acción describe?**  
a) Modificar los datos de una categoría existente

**¿Qué módulo está relacionado?**  
a) Gestión de categorías

**¿Qué verbo identifica la acción?**  
a) Editar

**¿Qué ruta crees que necesita?**  
a) Ruta: /editar\_categoria/<int:idCategoria>  
b) Método: GET y POST  
c) Qué hace: Carga los datos actuales de la categoría para editarla y guarda los cambios

**¿Dónde se debería ubicar esta ruta (archivo)?**  
a) Dentro de la carpeta controllers en el controlador de categorías (categoria\_controller.py)

# **Contextualización**

Este informe detalla la conceptualización y diseño de un sistema basado en el patrón Modelo-Vista-Controlador (MVC) que utilizara Flash, para la gestión operativa de la cadena de Hoteles Hilton, la cual describe una cadena hotelera con:

* Hoteles
* categorías de hotel
* Habitaciones (de múltiples tipos)
* Clientes (personas naturales y jurídicas como agencias)
* Reservas
* IVA según el perfil del cliente y normatividad
* Motivos de viaje
* Temporadas

Teniendo esto presente, se puede crear módulos funcionales como los siguientes:

1. Gestión de Hoteles

CRUD de hoteles (nombre, dirección, teléfono, año, etc.)

-Relación con su categoría

2. Gestión de Categorías de Hotel

CRUD de categorías ( Boutique, Ecohotel, etc.)

- Afecta la visualización y filtro de hoteles

3. Gestión de Habitaciones

CRUD de habitaciones: código, tipo, número, piso

-Relación con el hotel

-Visualización por estado (disponible/ocupada)

4. Gestión de Tipos de Habitación

CRUD de tipos de habitación (suite, doble, individual, familiar, etc.)

-Se asocian a habitaciones

-Permite filtrado por comodidad o capacidad

5. Gestión de Clientes

-CRUD de personas naturales

-CRUD de agencias de viajes (personas jurídicas)

-Registro del tipo de cliente (nacional, extranjero)

-Determina si aplica o no el IVA

6. Gestión de Reservas

-Crear reservas con fecha inicio/fin, habitación, cliente

-Registrar motivo de viaje (vacaciones, negocios, etc.)

-Asociar precio, tipo de cliente (natural/jurídico)

-Aplicar IVA según perfil (residente/no residente)

7. Gestión de Motivos de Viaje

-CRUD de motivos: turismo, negocios, salud, luna de miel, etc.

-Seleccionable al crear la reserva

8. Gestión de Temporadas

-Registrar temporadas (alta, media, baja)

-Puede modificar tarifas dependiendo de la lógica aplicada

9. Facturación y Cálculo del IVA

-Determinar si aplica IVA o no según el tipo de cliente y la zona

-Simular total facturado

-Generar factura con o sin IVA (según turista nacional/extranjero, región, etc.)

10. Usuarios del sistema y autenticación (solo admins o recepcionistas)

-Login/logout con sesión

-Mostrar nombre del usuario logueado

11. Reportes (avanzado)

-Reservas por temporada, por cliente, por hotel

-Total facturado en un rango de fechas

-Habitaciones ocupadas vs disponibles

Este informe se centrará en la definición del modelo de datos, la estructura de las vistas y los controladores principales para las funcionalidades de Gestión de Hoteles y Gestión de Categorías, sentando las bases para la expansión a otros módulos

# **Creación de vistas con Bootstrap**

Con base en el análisis contextual, se ha procedido a diseñar y desarrollar las interfaces de usuario (Vistas) de la aplicación, empleando el framework Bootstrap para asegurar un diseño responsivo y moderno, como lo es:

* Vista de Index,
* Vista de Login
* Vista de registro
* Dashboard con las gestiones correspondientes
* Vista de Gestión de Hoteles
* Vista de Gestión de categorías

En estas dos últimas vistas nos enfocaremos para la realización del CRUD sobre el MVC

# **Creación de la Base de datos**

El siguiente procedimiento es la creación de la base de datos relacional, la cual será implementada en MySQL, la cual ha sido diseñada considerando las entidades y relaciones identificadas en la contextualización de la cadena de "Hoteles Hilton", esta se retomará más adelante, en el Modelo de la arquitectura MVC se materializa a través de la Base de Datos.

# **Creación del Modelo Vista Controlador**

A continuación, se explicará la creación del Modelo Vista Controlador sobre la cadena de Hoteles Hilton

# **Organización de carpetas**

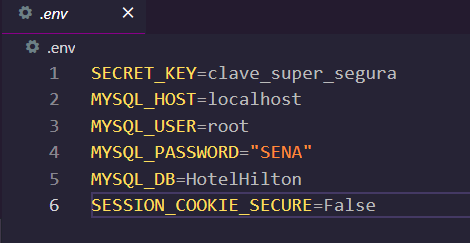
Para comenzar con la creación del modelo Vista controlador, se debe crear una carpeta y se debe tener de manera óptima la estructura de carpetas, a continuación, se mostrará como debe quedar organizada la estructura:

Texto

El contenido generado por IA puede ser incorrecto.

# **Configuración de conexión de la base de datos**

Se realizara el archivo “.env” el cual es el encargado de las configuraciones de las variables el cual nos permitirá tener acceso a la base de datos con la que se trabajara, esta información va de acuerdo a la realización de la instancia en MYSQL WORKBENCH:



-SECRET\_KEY=clave\_super\_segura

Esta variable nos ayudara a realizar la conexión a la base de datos

- MYSQL\_HOST=localhost

Representa la instancia de la base de datos, en este caso como tiene la instancia el puerto 3306, se pone localhost

- MYSQL\_USER=root

Es el usuario con el que se configuro la base de datos, en este caso, la base de datos se configuro como root

-MYSQL\_PASSWORD="SENA"

Es la clave con la que se configuro MYSQL, en este caso, es SENA

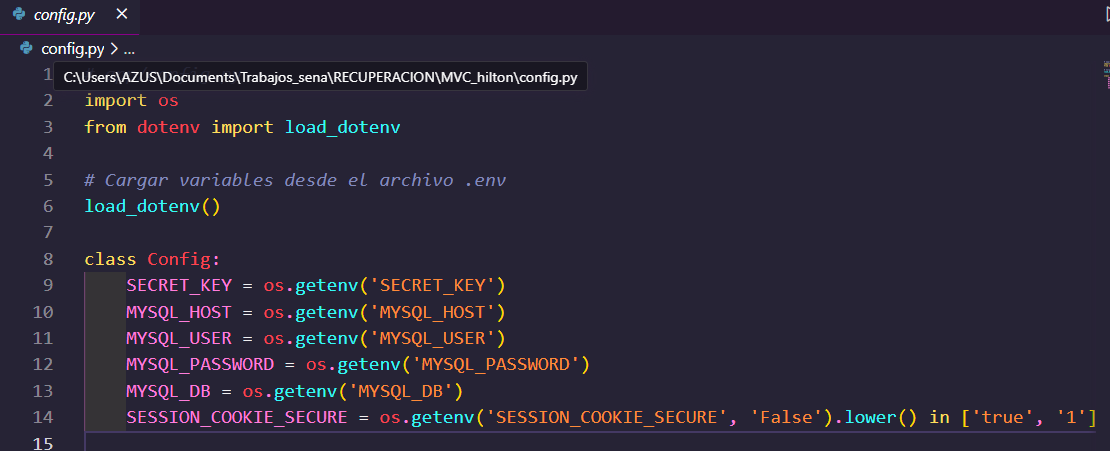
-MYSQL\_DB=HotelHilton

Es el nombre de la base de datos que nos vamos a conectar

-SESSION\_COOKIE\_SECURE=False

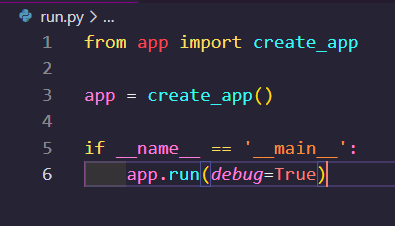
Es para manejar la aplicación de forma segura

# **Importación de datos .env**



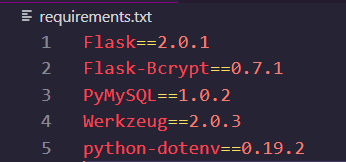
En el archivo config.py, Se importa la librería “os” que nos permite realizar una conexión a la base de datos, además se instala la librería “dotenv “para poder utilizar el archivo .env, guardamos las variables que vamos a llamar en una clase llamada config y traemos la variable con lo que tiene el archivo .env con os.getenv

# **Archivo que corre la aplicación**



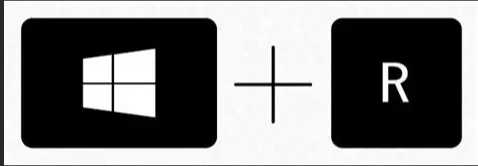
Run.py es el archivo principal el cual se le importara create\_app, luego en la variable app se llamara una función llamada create\_app y se realizara una función el debug verifica que no halla errores y los visualice, hasta que no se solucione los errores.

# **Archivo con las librerías de importación**



En el archivo requirements.txt se importan las librerías que se va a utilizar, con sus respectivas versiones, como flask ya que se trabajara con ella, flask-bcrypt para encriptar la contraseña, pyMySQL el cual nos ayudara a hacer conexión con la base de datos, werkzeug el cual nos ayudara a la seguridad de la aplicación y Python-dotenv para utilizar el archivo .env

# **Activación del entorno virtual**

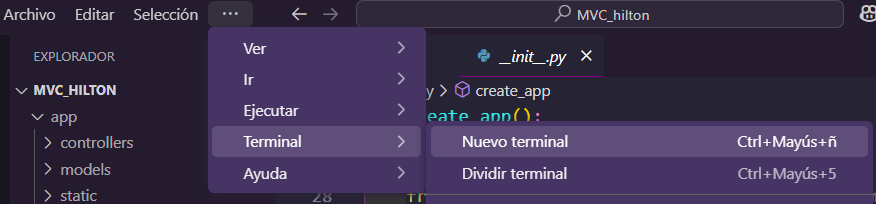


Antes de realizar la activación del entorno virtual, se debe entrar al CMD con el comando windows + r y se ejecuta, luego en el CMD se escribe python --version o py –version, ya depende de como lo reconozca el CDM, según lo que salga el CDM se debe poner en la terminal de Visual Studio Code

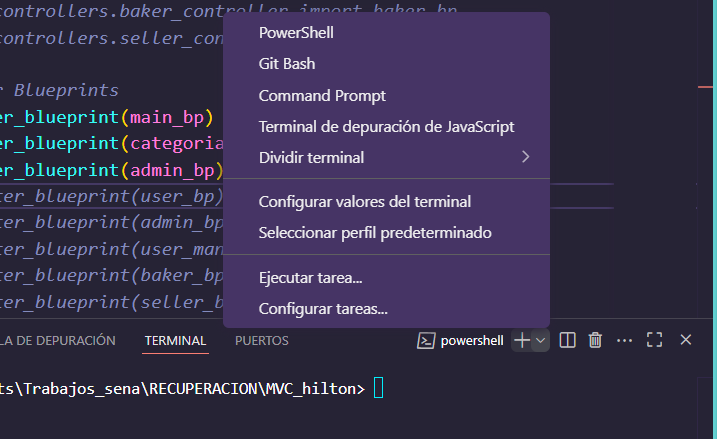
Texto

El contenido generado por IA puede ser incorrecto.

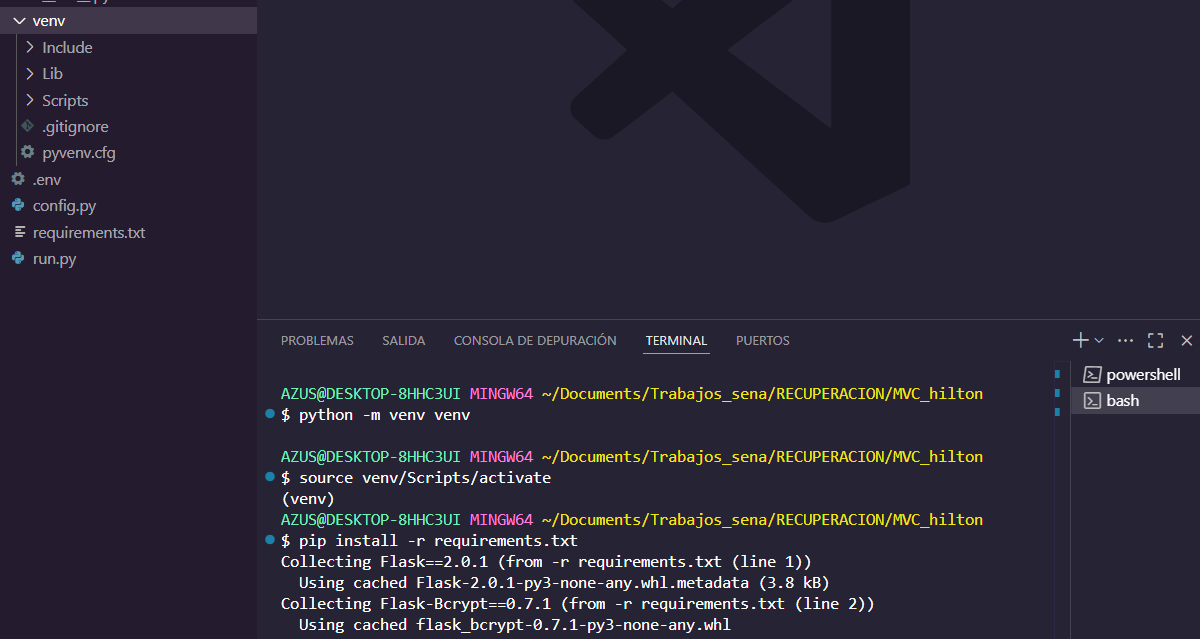
En visual Studio Code, debemos entrar a los 3 puntos, luego a terminal y a nuevo terminal



Una vez adentro de la terminar, nos dirigimos a el + y oprimimos la flecha abajo, luego se selecciona git bash

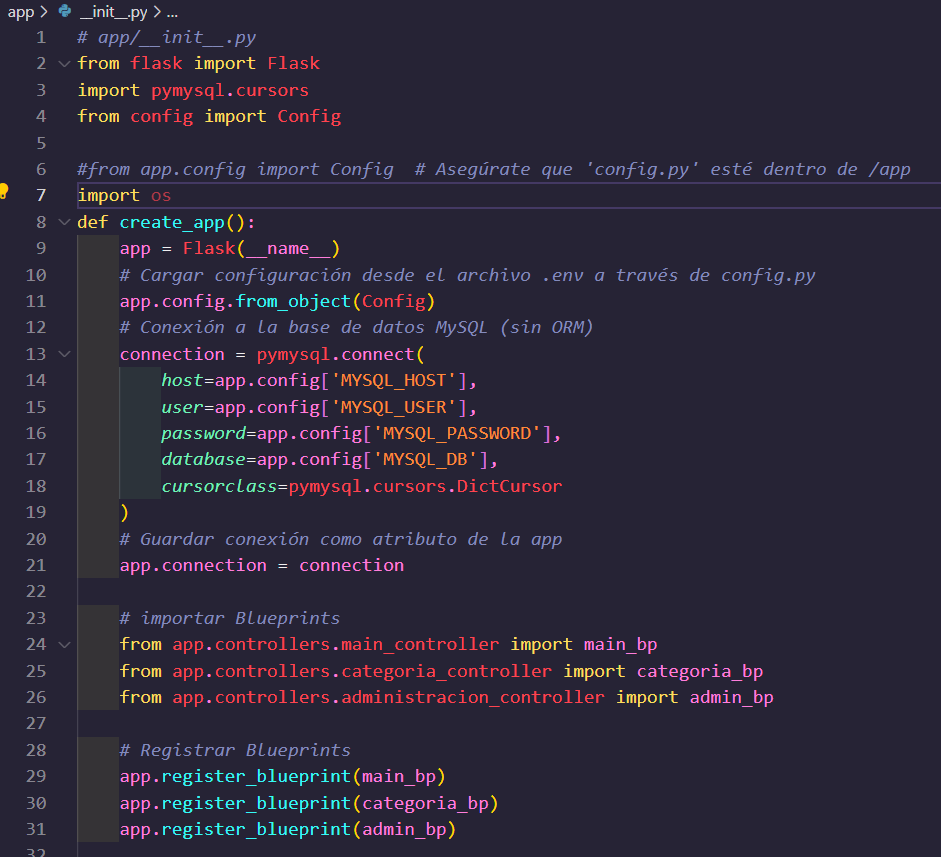


Luego se escribe dependiendo lo que nos salió en el CMD Python o py y escribimos el comando Python -m venv venv, el primer venv es una palabra reservada de Python y el segundo sera el nombre de la carpeta, apenas se mande el comando saldrá una carpeta con el nombre venv, luego se activa el entorno con el comando source venv/Scripts/actívate, e instalamos las librerías del documento requirements.txt con el comando pip install -r requirements.txt



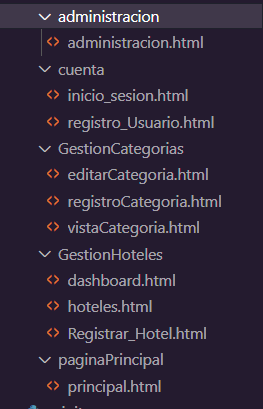
# **Archivo init**

En el archivo \_\_init\_\_.py haremos la función create\_app() que se escribió en el archivo run.py , se empieza a importar flask, pymsql.cursors, se importa de config la clase config y se importa el os, luego se crea la función create\_app, y se crea un variable app= flask(\_\_name\_\_) que cuando se ejecute , el cual \_\_name\_\_ esta también en run.py, levanta todo lo de create\_app , traemos las configuraciones del archivo .env mediante el archivo config y hacemos la conexión a la base de datos connection = pymysql.connect() y llamamos las variables de config y lo guardamos en app.connection, luego importamos todos los blueprints y los registramos y lo retornamos a app

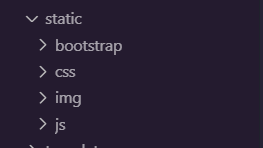


# **Templates**

En esta carpeta se crearon los html que se habian mencionado anteriormente con el framework de boostratp, además, se debe tener en cuenta que para llamar lo que esta en static, se escribira lo siguiente, por ejemplo  *href*="{{url\_for('static', filename='css/katherin/principal.css' )}}"



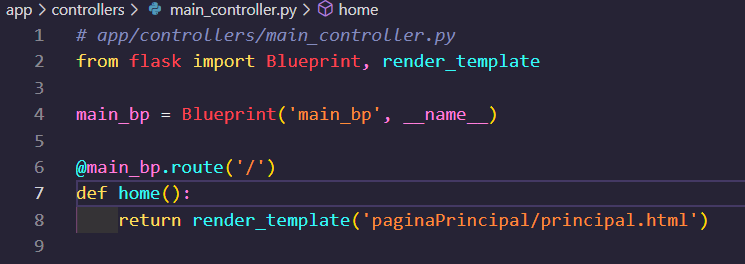
# **static**



Junto a los templates, se crearon los static donde se encuentra el archivo mimificado de Bootstrap, los css, img y los js

# **Pagina Principal**

### *Controlador*

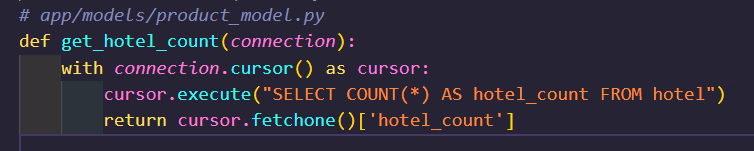


Para el controlador de la pagina principal, la cual se abrirá apenas corramos la aplicación sera main\_controller.py, acá se importara desde flask el blueprint, el cual nos ayudara a hacer el manejo de la modularidad y render\_template nos ayudara a renderizar los html que están en la carpeta render\_template, después declaramos un blueprint llamado main\_bp = blueprint(“main\_bp, \_\_name\_\_”) y ponemos el name que es una palabra reservada de flask,

Luego le decimos que vamos a usar el blueprint y le damos la ruta, la ruta sera el /, el cual se mostrara en el navegador: @main\_bp.route('/'),luego hacemos una función llamada home y le ponemos que retorne un template donde se encuentra la pagina principal: return render\_template('paginaPrincipal/principal.html')

# **administración**

### *Modelo*



Texto

El contenido generado por IA puede ser incorrecto.

En esta sección se presenta el modelo de la entidad "Categoría" y “hoteles”, el cual forma parte del patrón de arquitectura MVC (Modelo-Vista-Controlador). Este modelo permite realizar operaciones sobre la base de datos relacionadas con la tabla Categoria.

Cada función del modelo recibe un parámetro connection, que se encarga de conectarse a la base de datos definida previamente en el archivo .env. A través de este objeto se ejecutan consultas SQL utilizando placeholders (%s) para evitar inyecciones SQL y permitir el uso de parámetros dinámicos.

Funciones del Modelo

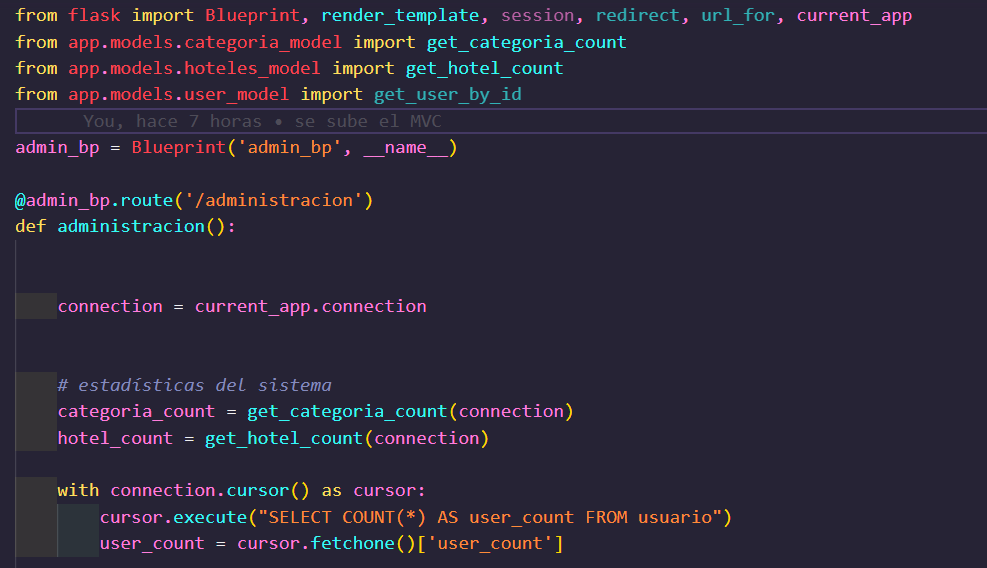
get\_categoria\_count(connection)

Esta función cuenta cuántas categorías existen en la base de datos mediante la sentencia SELECT COUNT(\*) AS categoria\_count FROM Categoria. Luego, se accede al valor de retorno mediante fetchone()['categoria\_count'].

get\_hotel\_count(connection)

Esta función cuenta cuántos hoteles existen en la base de datos mediante la sentencia SELECT COUNT(\*) AS hotel\_count FROM hotel. Luego, se accede al valor de retorno mediante fetchone()['hotel\_count']

### *Controlador*

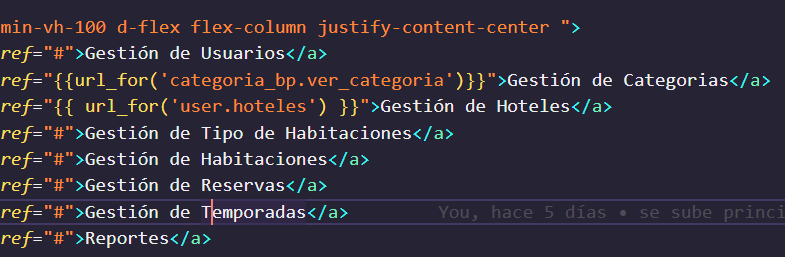


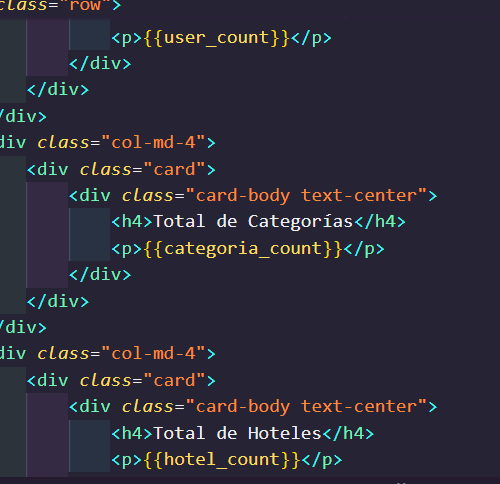
Texto

El contenido generado por IA puede ser incorrecto.

El controlador administracion() pertenece al blueprint admin\_bp, y está asociado a la ruta /administracion. Su función es mostrar una vista administrativa con estadísticas generales del sistema, esto se muestra cuando el usuario entra a administracion. Al acceder a esta ruta, el controlador inicia estableciendo la conexión a la base de datos con current\_app.connection. Luego, llama a dos funciones del modelo: get\_categoria\_count() y get\_hotel\_count() para contar cuántas categorías y hoteles hay registrados. Además, usando un cursor directo, ejecuta una consulta SQL para contar el total de usuarios en la tabla usuario. Todos estos valores se almacenan en variables (categoria\_count, hotel\_count, user\_count) que luego son enviadas a la plantilla administracion.html mediante render\_template. Esta plantilla usará esas variables para mostrar en pantalla un resumen estadístico del sistema.

### *Template*

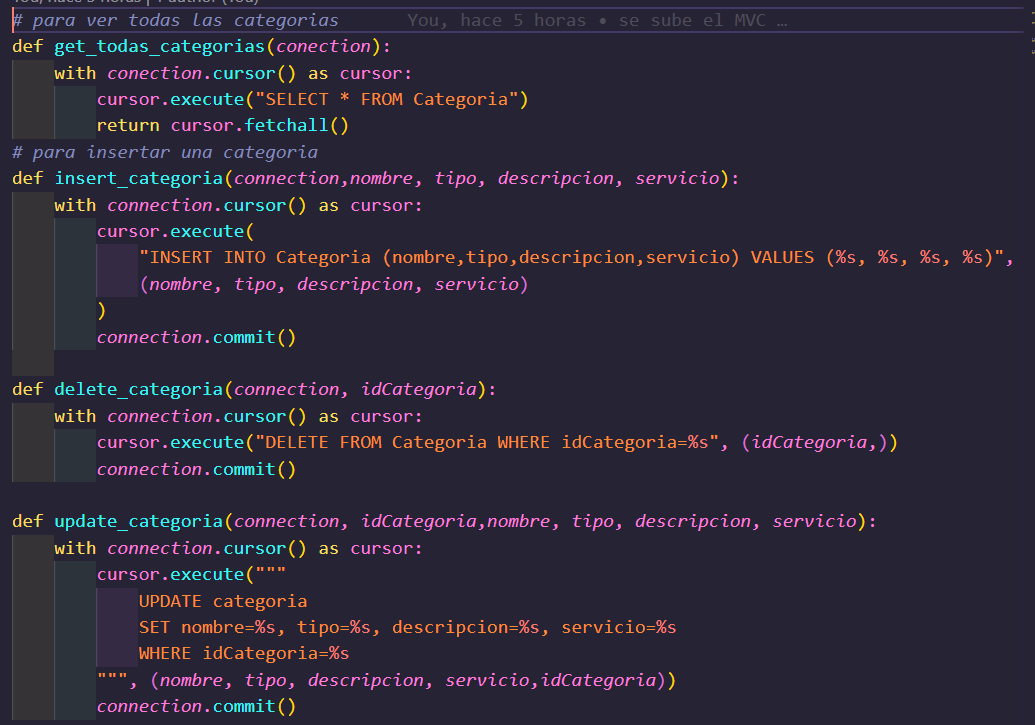


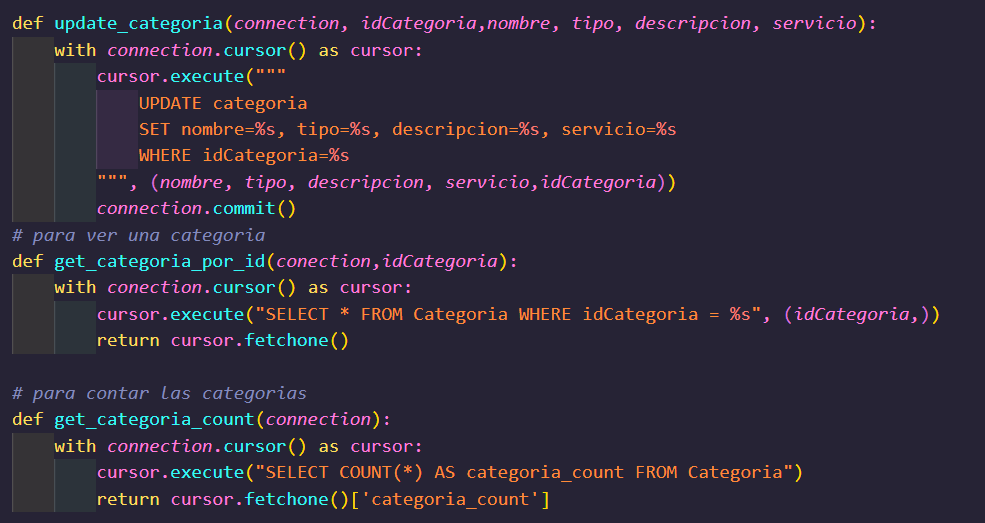


Este template representa la vista del panel de administración del proyecto Hilton y está diseñado para mostrar estadísticas clave y facilitar la navegación entre módulos de gestión.Incluye un menú lateral con enlaces a distintas secciones administrativas como usuarios, categorías y hoteles, donde se llaman desde los blueprint definidos anteriormente, y una sección principal con tarjetas que muestran el total de usuarios, categorías y hoteles registrados, usando las variables user\_count, categoria\_count y hotel\_count. Además, emplea la función url\_for para generar rutas dinámicas seguras y carga estilos personalizados desde la carpeta static, ofreciendo una interfaz clara, funcional y adaptada al rol del administrador.

# **Categorías**

### *Modelo*





En esta sección se presenta el modelo de la entidad "Categoría", el cual forma parte del patrón de arquitectura MVC (Modelo-Vista-Controlador). Este modelo permite realizar operaciones sobre la base de datos relacionadas con la tabla Categoria.

Cada función del modelo recibe un parámetro connection, que se encarga de conectarse a la base de datos definida previamente en el archivo .env. A través de este objeto se ejecutan consultas SQL utilizando placeholders (%s) para evitar inyecciones SQL y permitir el uso de parámetros dinámicos.

Funciones del Modelo

get\_todas\_categorias(connection)

Esta función permite obtener todas las categorías registradas en la base de datos. Se realiza un SELECT \* FROM Categoria y se utiliza fetchall() para retornar una lista con todos los registros encontrados.

insert\_categoria(connection, nombre, tipo, descripcion, servicio)

Esta función permite insertar una nueva categoría. Recibe como parámetros los campos requeridos (nombre, tipo, descripción y servicio) y ejecuta una sentencia INSERT INTO con estos valores. Luego se utiliza connection.commit() para guardar los cambios en la base de datos.

delete\_categoria(connection, idCategoria)

Elimina una categoría específica de la base de datos según su ID. Ejecuta una sentencia DELETE FROM Categoria WHERE idCategoria = %s, y luego confirma la operación con connection.commit().

update\_categoria(connection, idCategoria, nombre, tipo, descripcion, servicio)

Permite actualizar una categoría existente. Recibe el ID de la categoría y los nuevos valores. Se ejecuta un UPDATE con los campos modificados, y se utiliza connection.commit() para guardar los cambios.

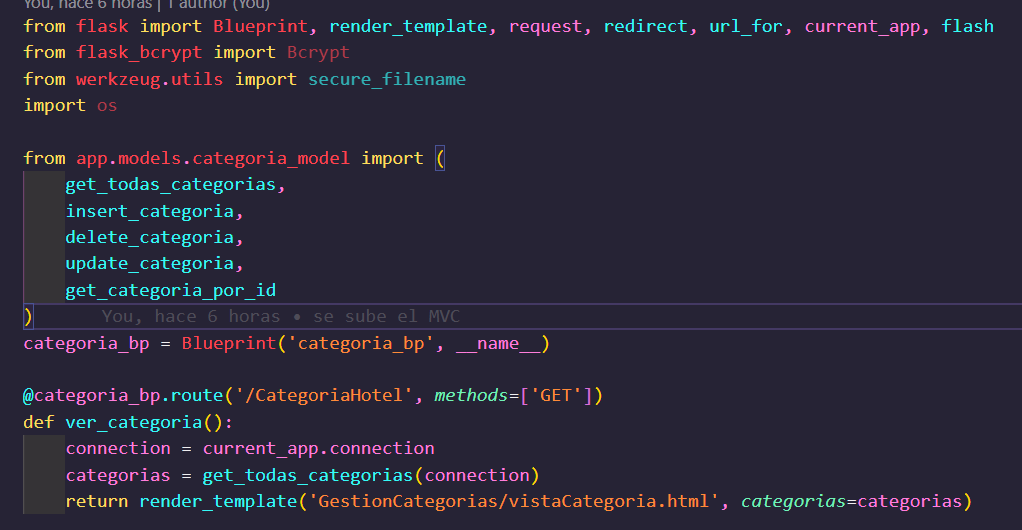
get\_categoria\_por\_id(connection, idCategoria)

Permite consultar una sola categoría según su identificador. Realiza un SELECT \* FROM Categoria WHERE idCategoria = %s y retorna un solo resultado utilizando fetchone().

get\_categoria\_count(connection)

Esta función cuenta cuántas categorías existen en la base de datos mediante la sentencia SELECT COUNT(\*) AS categoria\_count FROM Categoria. Luego, se accede al valor de retorno mediante fetchone()['categoria\_count'].

### *Controlador*



Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

En esta sección se presenta el controlador de la entidad "Categoría", el cual forma parte del patrón de arquitectura MVC (Modelo-Vista-Controlador). Su principal función es actuar como un puente entre el modelo y la vista, lo que hace es recibir las solicitudes del usuario, las procesa y utiliza el modelo para acceder o modificar los datos en la base de datos, para luego enviar la información correspondiente a las vistas que el usuario verá en el navegador.

En este controlador empezamos importando desde flask el blueprint que permite el manejo de la modularidad, el render\_template el cual nos ayudara a renderizar los html que se encuentra en la carpeta render\_template, request que nos permite recibir información del usuario desde un form, redirect, redirigir al usuario a otra ruta después de completar una acción, url\_form, genera la URL dinámica para una función de vista registrada en Flask, current\_app, se usa para acceder a la conexión de base de datos que se ha guardado en la configuración de la aplicación, flash, permite enviar mensajes temporales al usuario.

Empezamos a traer los modelos siguiento la rutas donde están, from app.models.categoria\_model import, traemos las funciones que se menciono anteriormente:

* get\_todas\_categorias,
* insert\_categoria,
* delete\_categoria,
* update\_categoria,
* get\_categoria\_por\_id

después declaramos un blueprint llamado categoria\_bp= blueprint(“categoria\_bp, \_\_name\_\_”) y ponemos el name que es una palabra reservada de flask.

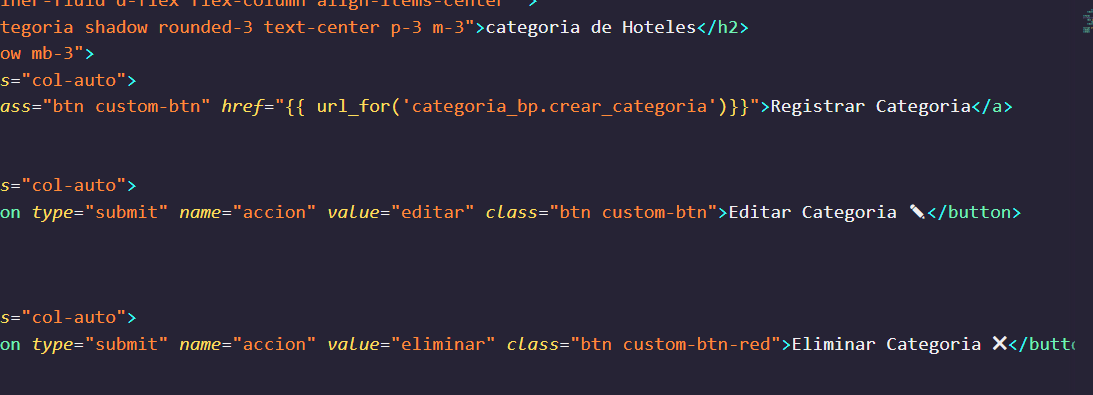
Luego le decimos que vamos a usar el blueprint y le damos la ruta, la ruta sera el /CategoriaHotel, con el método get el cual nos ayuda a visualizar, el cual se mostrara en el navegador, luego hacemos una función llamada ver\_categoria, lo conectamos a la base de datos con connection = current\_app.connection, traemos la función del modelo que importamos: get\_todas\_categorias(connection) y lo guardamos en la variable categorias y lo retornamos con un render\_template al archivo vistaCategoria.html, luego creamos una variable global para usarla en el html  categorias=categorias

Seguimos con otra función llamada crear\_categoria con método get y post para crear las categorías, nos conectamos a la base de datos, lo lleve al template registroCategoria.html y le decimos que si el método es igual a post nos traiga del form los datos como nombre, tipo, descripción y servicio y la inserte, esto lo logramos llamando a la función de insert\_categoria(connection, nombre, tipo, descripcion, servicio) con sus parámetros y si sale algun error, muestre un mensaje flash('Error al crear la categoria: ' + str(e), 'danger') y retorne al blueprint categoria\_bp y valla a la función ver\_categoria, categoria\_bp.ver\_categoria

Continuamos con otra función llamada procesar\_accion, la cual tiene método post, define una ruta (/procesar\_accion), nos conectamos a la base de datos con connection = current\_app.connection, Se obtienen todas las categorías seleccionadas desde un formulario con request.form.getlist('categorias\_seleccionadas'), getlist recupera una lista de valores y categorias\_seleccionadas es el nombre del campo en el formulario HTML, accion = request.form.get('accion') obtiene el valor de la acción que el usuario quiere ejecutar, Si el usuario no ha seleccionado ninguna categoría, se muestra un mensaje de advertencia usando flash y se redirige a la vista de categorías, si la opcion es eliminar, recorre cada idCategoria en la lista y llama a la función delete\_categoria() del modelo ,después, se muestra un mensaje de éxito, si la opcion es editar y el usuario seleccionó más de una categoría, se muestra una advertencia porque solo se permite editar una por vez y si solo se seleccionó una categoría, se redirige a la ruta de edición con ese ID, si ocurre cualquier error durante el proceso, se muestra un mensaje en pantalla con el detalle del error, después de ejecutar cualquier acción, se redirige al usuario nuevamente a la vista de todas las categorías.

La función editar\_categoria del controlador maneja la ruta /editar\_categoria/<int:idCategoria>, el cual mostrara el nombre de la ruta y el id de la categoría en el navegdaor, y permite editar una categoría específica de la base de datos. Esta ruta acepta los métodos GET y POST. Cuando se accede por GET, se espera que se muestre un formulario precargado con los datos actuales de la categoría. En caso de recibir una solicitud POST, es decir, cuando el usuario envía el formulario con los datos modificados, se extraen los campos enviados (nombre, tipo, descripcion y servicio) mediante request.form. Luego, se llama a la función del modelo update\_categoria pasando estos valores junto con el idCategoria. Si la operación es exitosa, se muestra un mensaje de confirmación usando flash; en caso contrario, se captura el error y se muestra un mensaje de advertencia. Finalmente, el usuario es redirigido a la vista principal de categorías (/CategoriaHotel) usando redirect(url\_for(...)).

### *Template*



Texto

El contenido generado por IA puede ser incorrecto.

En el template de vista, para traer el controlador se escribe en un {{ url\_for(‘nombre bp.funcion')}}, para registrar se debe seleccionar un checkbox el cual se guardara con categorias\_seleccionadas, como estamos usando variables globales, se pintara en la pantalla todas las categorias, una vez seleccionado el checkbox, se debe elegir tres de las acciones, esto se procesara con acción ( name="accion") en el controlador y según lo que se escoja, se hará el proceso en el controlador



El propósito del template es permitir al usuario editar una categoría existente del sistema. La información de la categoría se carga automáticamente en el formulario utilizando variables globales, para que se plasme en la pantalla, el id que se escogió, se escriben las variables globales en sus respectivos campos y para que el controlador los identifique se pone en el name= sus respectivos nombres



El propósito del template es permitir al usuario crear una categoría. Al cargarse en el navegador, muestra un formulario con campos para ingresar el nombre de la categoría, tipo (como número de estrellas), descripción y servicios incluidos. El formulario envía los datos al Flask mediante el método POST hacia la ruta definida en categoria\_bp.crear\_categoria.

# **FLUJO GENERAL DE LA APLICACIÓN**

El usuario accede al sistema desde la página principal:

Ruta: /

Vista cargada: principal.html

Desde allí puede ir al formulario de inicio de sesión:

Ruta: /login

Método: GET

Vista: inicio\_sesion.html

Cuando el usuario envía sus credenciales:

Ruta: /login

Método: POST

Acción: Si las credenciales son correctas, se almacena la información del usuario en la sesión (session['usuario']) y se redirige al panel de administración.

En el panel de administración:

Ruta: /administracion

Vista: administracion.html

Información mostrada:

Número total de usuarios

Número total de categorías

Número total de hoteles

La información se obtiene desde consultas directas a la base de datos y modelos como:

get\_user\_by\_id

get\_categoria\_count

get\_hotel\_count

Desde el panel o el menú de navegación, el usuario puede gestionar hoteles mediante las siguientes funcionalidades:

# **FLUJO DETALLADO DEL MÓDULO DE HOTELES**

5.1. Ver lista de hoteles

Ruta: /hoteles

Método: GET

Controlador: lista\_hoteles()

Vista: hoteles.html

Acción: Se consultan todos los hoteles activos y se muestran en una tabla. La información viene desde el modelo obtener\_hoteles(), que realiza un LEFT JOIN con las categorías.

5.2. Registrar un nuevo hotel

Ruta: /registrar\_hotel

Método: GET y POST

Controlador: registrar\_hotel()

Vista: registrar\_hotel.html

Si es GET:

Se muestra un formulario con todos los campos requeridos y un selector desplegable de categorías (obtener\_categorias()).

Si es POST:

Se validan los campos.

Si todo es válido, se ejecuta el modelo insertar\_hotel() para insertar los datos en la base de datos.

Si hay error, se muestra un mensaje usando flash.

5.3. Editar un hotel existente

Ruta: /editar\_hotel/<id>

Método: GET y POST

Controlador: editar\_hotel\_view(id)

Vista: registrar\_hotel.html

Si es GET:

Se obtiene la información del hotel con obtener\_hotel\_por\_id(id).

Se muestran los datos prellenados en el formulario de edición.

Si es POST:

Se validan los datos actualizados.

Se ejecuta editar\_hotel(...) con los nuevos valores.

5.4. Eliminar (ocultar) uno o más hoteles

Ruta: /eliminar\_hoteles

Método: POST

Controlador: eliminar\_hoteles()

Acción:

Se recibe una cadena con los IDs seleccionados.

Se convierten a lista y se pasan a ocultar\_hoteles() para marcarlos como inactivos.

No se eliminan físicamente de la base de datos.