

**UNIVERSIDAD INTERAMERICANA DE PANAMÁ FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISEÑO**

**CURSO DE:**

**PROGRAMACIÓN DE COMPUTADORAS IV**

**(CÓDIGO 301-00072)**

**TEMA:**

**PROYECTO FINAL: DOCUMENTACIÓN TÉCNICA**

**REALIZADO POR:**

**IRVING BUITRAGO 8-989-2047**

**BEATRIZ SANCHEZ 8-833-1723**

**RUT ÁLVAREZ 8-997-2205**

**ARIADNA ACEVEDO 8-712-1473**

**RICARDO GUO 8-975-1976**

**DOCENTE:**

**ABDEL GADIEL MARTÍNEZ LASSONDE**

**FECHA DE ENTREGA:**

**LUNES 21 AGOSTO 2023**

**INTRODUCCIÓN**

El presente informe técnico describe el desarrollo de un proyecto universitario para la clase de Programación IV, enfocado en la creación de una aplicación web para registrar citas en una clínica dental. El objetivo principal de esta aplicación es administrar la agenda de un médico dentista y sus pacientes, permitiendo un mejor control y organización de las citas y tratamientos dentales.

La aplicación web fue desarrollada utilizando el lenguaje de programación Python y el microframework Flask, que facilita la creación de aplicaciones web en Python. Se emplearon las siguientes bibliotecas y APIs: Flask, render\_template, request, redirect, url\_for, sesion, sqlite3 y passlib.hash.pbkdf2\_sha256. Además, se utilizaron diseños en HTML para la interfaz de usuario.

El proyecto se basa en la necesidad de mejorar la gestión de citas y tratamientos en clínicas dentales, ya que una agenda mal organizada puede generar retrasos, estrés y pérdida de ingresos. La aplicación permite a la asistente administrativa programar citas de pacientes de forma sencilla y eficiente, mientras que el médico dentista puede administrar su agenda y mantener un registro adecuado de las fichas médicas y tratamientos de sus pacientes.

En resumen, este informe técnico presenta el proceso de desarrollo y las tecnologías empleadas durante el cuatrimestre en las clases del profesor Martínez, que se aplicaron en la creación de una herramienta web para gestionar citas y tratamientos dentales. La aplicación busca mejorar la organización y eficiencia en la atención al paciente, así como facilitar la administración de la agenda del médico dentista y sus pacientes.

**DISEÑO DE LA APLICACIÓN**

Para la creación de esta aplicación para registrar citas para una clínica dental y administrar la agenda de un dentista se basó en dos archivos .py escritos en lenguaje de programación Python cuyos nombres son base.py y my\_table.py. También se creo una base de datos y en las carpetas que se visualizan en la imagen en templates van las plantillas HTML para los diseños de interfaz de usuario y en .idea contiene diversos archivos XML que fueron creados al momento de utilizar el software Pycharm.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

A continuación compartiremos cada uno de los elementos de forma resumida y comentada para explicar de forma sencilla su uso, importancia e implementación en la creación de la aplicación:

**Código Python: base.py**

Por su extension hemos divido el código en 3 secciones o parte las cuales vamos a explicar brevemente a continuación:

***PARTE I.***

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

import sqlite3 as sql

from passlib.hash import pbkdf2\_sha256

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

app.secret\_key = 'YfeItpdUsWnmgfQ'

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

def index():

return render\_template('login.html')

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

def login():

if request.method == "POST":

try:

username = request.form['username']

password = request.form['password']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

password\_hash = cur.execute('SELECT PASSWORD FROM USER\_DATA WHERE USERNAME = ?',(username,)).fetchone()

if password\_hash and pbkdf2\_sha256.verify(password, password\_hash[0]):

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

else:

return render\_template('login.html', error='Credenciales invalidas')

except:

return render\_template('login.html', error='Credenciales invalidas')

return render\_template('login.html')

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

def dashboard():

if request.method == "POST":

opcion = int(request.form['opcion'])

if opcion == 1:

return render\_template('dashboard.html')

elif opcion == 2:

return redirect(url\_for('appointment\_form'))

elif opcion == 3:

return redirect(url\_for('patient\_form'))

else:

return render\_template('dashboard.html')

return render\_template('dashboard.html')

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

def patient\_form():

return render\_template('patient.html')

@app.route('/patient/search', methods=["GET", "POST"])

def search\_patient():

if request.method == 'POST':

try:

name\_last\_name = request.form['name\_last\_name']

name, last\_name = name\_last\_name.split()

with sql.connect('db\_clinica') as con:

cur = con.cursor()

review = cur.execute('SELECT \* FROM PATIENT WHERE NAME = ? AND LAST\_NAME = ?', (name,last\_name)).fetchone()

if review:

return render\_template('patient.html', review=review)

else:

return render\_template('patient.html', error='paciente no encontrado')

except:

return render\_template('patient.html', error='paciente no encontrado')

return render\_template('patient.html')

@app.route('/patient/add', methods=["GET", "POST"])

def add\_patient():

if request.method == 'POST':

try:

name = request.form['name']

last\_name = request.form['last\_name']

id = request.form['id']

birthdate = request.form['birthdate']

address = request.form['address']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

review = cur.execute('SELECT \* FROM PATIENT WHERE ID = ?', (id,)).fetchone()

if review:

return render\_template('add\_patient.html', error='registrado anteriormente')

else:

insert = cur.execute('INSERT INTO PATIENT (name, last\_name, id, birthdate, address) VALUES (?, ?, ?, ?, ?)', (name, last\_name, id, birthdate, address))

con.commit()

return render\_template('patient.html', msg='paciente registrado con exito')

except:

return render\_template(error= 'No se agrego el paciente correctamente')

return render\_template('add\_patient.html')

@app.route('/patient/edit', methods=["GET", "POST"])

def edit\_patient():

if request.method == 'POST':

try:

column\_update = request.form['column\_update']

new\_data = request.form['new\_data']

name = request.form['name']

last\_name = request.form['last\_name']

birthdate = request.form['birthdate']

id = request.form['id']

address = request.form['address']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

update = cur.execute(f'UPDATE PATIENT SET {column\_update} = ? WHERE NAME = ? AND LAST\_NAME = ? AND BIRTHDATE = ? AND ID = ? AND ADDRESS = ?', (new\_data, name, last\_name, birthdate, id, address))

con.commit()

return render\_template('patient.html', update=update)

except:

return render\_template('patient.html', error= 'No se edito el paciente correctamente')

return render\_template('patient.html')

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

def delete\_patient():

try:

name = request.form['name']

last\_name = request.form['last\_name']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

delete = con.execute('DELETE FROM PATIENT WHERE NAME = ? AND LAST\_NAME = ?', (name, last\_name))

con.commit()

return render\_template('patient.html')

except:

return render\_template('patient.html', error= 'No se elimino el paciente correctamente')

**Explicación Parte I.**

Este código forma parte de la aplicación web desarrollada en Python utilizando el microframework Flask. La aplicación permite a los usuarios iniciar sesión y gestionar citas y pacientes en una clínica dental. A continuación, se describen las principales partes del código:

* Importación de bibliotecas y APIs necesarias, como Flask, render\_template, request, redirect, url\_for, session, sqlite3 y pbkdf2\_sha256.
* Creación de la instancia de la aplicación Flask y configuración de la clave secreta.
* Definición de rutas y funciones asociadas para manejar diferentes acciones en la aplicación, como iniciar sesión, acceder al panel de control, buscar pacientes, agregar pacientes, editar pacientes y eliminar pacientes.
* Uso de la base de datos SQLite para almacenar y recuperar información sobre pacientes y citas.
* Uso de la biblioteca Passlib para verificar las contraseñas de los usuarios utilizando el algoritmo pbkdf2\_sha256.
* La aplicación permite a los usuarios iniciar sesión y realizar diversas acciones relacionadas con la administración de pacientes y citas, utilizando una base de datos SQLite para almacenar la información y Passlib para verificar las contraseñas de los usuarios.

***PARTE II.***

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

def appointment\_form():

return render\_template('appointment.html')

@app.route('/appointment/search', methods=["GET", "POST"])

def search\_appointment():

if request.method == 'POST':

try:

name\_last\_name = request.form['name\_last\_name']

name, last\_name = name\_last\_name.split()

with sql.connect('db\_clinica') as con:

cur = con.cursor()

review\_id = cur.execute('SELECT ID FROM PATIENT WHERE NAME = ? AND LAST\_NAME = ?', (name, last\_name)).fetchone()

if review\_id:

id\_patient = review\_id[0]

appo\_review = cur.execute('SELECT \* FROM APPOINTMENT WHERE ID\_PATIENT = ?', (id\_patient,)).fetchone()

if appo\_review:

return render\_template('appointment.html', name\_last\_name=name\_last\_name, appo\_review=appo\_review)

else:

return render\_template('appointment.html', error= 'No se encontraron citas agendadas')

else:

return render\_template('appointment.html')

except:

return render\_template('appointment.html', error= 'No se encontro el paciente')

return render\_template('appointment.html')

@app.route('/appointment/add', methods=["GET", "POST"])

def add\_appointment():

if request.method == 'POST':

try:

name\_last\_name = request.form['name\_last\_name']

name, last\_name = name\_last\_name.split()

appo\_date = request.form['appo\_date']

time = request.form['time']

reason = request.form['reason']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

patient\_id = cur.execute('SELECT ID FROM PATIENT WHERE NAME = ? AND LAST\_NAME = ?', (name, last\_name)).fetchone()

if patient\_id:

exist\_appo = cur.execute('SELECT \* FROM APPOINTMENT WHERE APPO\_DATE = ? AND TIME = ?',(appo\_date, time)).fetchone()

if exist\_appo:

return render\_template('add\_appointment.html', msg='Ya hay una cita registrada en ese horario.')

else:

insert = cur.execute('INSERT INTO APPOINTMENT (ID\_PATIENT, APPO\_DATE, TIME, REASON) VALUES (?, ?, ?, ?)', (patient\_id[0], appo\_date, time, reason))

con.commit()

return render\_template('appointment.html', msg='Cita agregada con éxito')

else:

return render\_template('add\_appointment.html', msg='Paciente no encontrado')

except:

return render\_template('add\_appointment.html', msg= 'La cita no se agrego correctamente')

return render\_template('add\_appointment.html')

**Explicación Parte II.**

Esta segunda parte del código también forma parte de la aplicación web en Python utilizando Flask para gestionar citas y pacientes en una clínica dental. En esta parte, se definen rutas y funciones asociadas para manejar acciones relacionadas con las citas, como mostrar el formulario de citas, buscar citas, y agregar citas. A continuación, se describen las principales partes del código:

* Definición de la ruta y función appointment\_form() para mostrar el formulario de citas utilizando la plantilla 'appointment.html'.
* Definición de la ruta y función search\_appointment() para buscar citas de un paciente específico. La función busca el paciente por su nombre y apellido, y luego consulta la base de datos SQLite para encontrar las citas asociadas a ese paciente. Si se encuentran citas, se muestran en la plantilla 'appointment.html'; de lo contrario, se muestra un mensaje de error.
* Definición de la ruta y función add\_appointment() para agregar nuevas citas a la base de datos. La función recopila información sobre el paciente y la cita, como el nombre y apellido del paciente, la fecha y hora de la cita y el motivo de la cita. Luego, verifica si el paciente existe en la base de datos y si ya hay una cita programada en el mismo horario. Si no hay conflictos, se inserta la nueva cita en la base de datos y se muestra un mensaje de éxito en la plantilla 'appointment.html'.
* Es decir, esta segunda parte del código proporciona funcionalidades adicionales para la aplicación web en Python que utiliza Flask para gestionar citas y pacientes en una clínica dental.

***PARTE III. FINAL***

@app.route('/appointment/edit', methods= ["GET", "POST"])

def edit\_appointment():

if request.method == "POST":

try:

column\_update = request.form['column\_update']

new\_data = request.form['new\_data']

name\_last\_name = request.form['name\_last\_name']

name, last\_name = name\_last\_name.split()

appo\_date = request.form['appo\_date']

time = request.form['time']

reason = request.form['reason']

with sql.connect('db\_clinica') as con:

cur = con.cursor()

update = cur.execute(f'UPDATE APPOINTMENT SET {column\_update} = ? WHERE APPO\_DATE = ? AND TIME = ? AND REASON = ?', (new\_data, appo\_date, time, reason))

con.commit()

return render\_template('appointment.html', update=update)

except:

return render\_template('appointment.html', error='No se edito la cita correctamente')

return render\_template('appointment.html')

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

def delete\_appointment():

try:

name = request.form['name']

last\_name = request.form['last\_name']

with sql.connect('db\_clinica.db') as con:

cur = con.cursor()

patient\_id = cur.execute('SELECT ID FROM PATIENT WHERE NAME = ? AND LAST\_NAME = ?', (name, last\_name)).fetchone()

if patient\_id:

id\_patient = patient\_id[0]

exist\_appo = cur.execute('SELECT ID FROM APPOINTMENT WHERE ID\_PATIENT = ?', (id\_patient,)).fetchone()

if exist\_appo:

delete = cur.execute('DELETE FROM APPOINTMENT WHERE ID\_PATIENT = ?', (id\_patient,))

con.commit()

return render\_template('appointment.html', msg='Cita eliminada')

else:

return render\_template('appointment.html', msg='Cita NO eliminada')

else:

return render\_template('appointment.html', msg='Cita NO encontrada')

except:

return render\_template('appointment.html', error='No se elimino correctamente')

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

app.run(debug=True)

**Explicación Parte III.**

En esta parte, se definen rutas y funciones asociadas para manejar acciones relacionadas con la edición y eliminación de citas. A continuación, se describen las principales partes del código:

* Definición de la ruta y función edit\_appointment() para editar citas existentes en la base de datos. La función recopila información sobre la cita y el paciente, como el nombre y apellido del paciente, la fecha y hora de la cita, y el motivo de la cita. Luego, actualiza la columna especificada con los nuevos datos proporcionados en la base de datos SQLite y muestra un mensaje de éxito o error en la plantilla 'appointment.html'.
* Definición de la ruta y función delete\_appointment() para eliminar citas existentes en la base de datos. La función busca el paciente por su nombre y apellido, y luego consulta la base de datos SQLite para encontrar las citas asociadas a ese paciente. Si se encuentra una cita, se elimina de la base de datos y se muestra un mensaje de éxito en la plantilla 'appointment.html'; de lo contrario, se muestra un mensaje de error.
* Ejecución de la aplicación Flask en modo de depuración, lo que permite a los desarrolladores ver los errores y realizar cambios en tiempo real sin tener que reiniciar la aplicación.

**Código Python: my\_table.py**

El siguiente código en Python utiliza la biblioteca sqlite3 para crear y gestionar una base de datos SQLite llamada db\_clinica. Además, utiliza la biblioteca passlib.hash para generar y verificar hashes de contraseñas utilizando el algoritmo pbkdf2\_sha256. A continuación, se presenta el código con comentarios:

import sqlite3

from passlib.hash import pbkdf2\_sha256

# Conectar a la base de datos SQLite llamada 'db\_clinica'

conn = sqlite3.connect('db\_clinica')

# Crear las tablas 'PATIENT', 'APPOINTMENT' y 'USER\_DATA' si no existen

# conn.execute('''CREATE TABLE IF NOT EXISTS PATIENT

# (ID INTEGER PRIMARY KEY NOT NULL,

# NAME TEXT NOT NULL,

# LAST\_NAME TEXT NOT NULL,

# BIRTHDATE DATE NOT NULL,

# ADDRESS TEXT NOT NULL);''')

#

# conn.execute('''CREATE TABLE IF NOT EXISTS APPOINTMENT

# (ID\_APPO INTEGER PRIMARY KEY AUTOINCREMENT,

# ID\_PATIENT INTEGER,

# APPO\_DATE DATE NOT NULL,

# TIME TEXT NOT NULL,

# REASON TEXT NOT NULL,

# FOREIGN KEY (ID\_PATIENT) REFERENCES PATIENT(ID));''')

# conn.commit()

# print('hecho')

#

# conn.execute('''CREATE TABLE IF NOT EXISTS USER\_DATA

# (USERNAME TEXT PRIMARY KEY NOT NULL,

# PASSWORD TEXT NOT NULL);''')

# Función para registrar un nuevo usuario en la tabla 'USER\_DATA'

# def register\_user(username, password):

# try:

# password\_hash = pbkdf2\_sha256.hash(password) # Generar el hash de la contraseña

#

# conn.execute('INSERT INTO USER\_DATA (USERNAME, PASSWORD) VALUES (?, ?)', (username, password\_hash))

# conn.commit()

# conn.close()

#

# print("Usuario registrado con éxito")

# except Exception as e:

# print("Error al registrar el usuario:", e)

#

# # Registrar un nuevo usuario

# register\_user('admin\_Admin', 'admin456')

**Explicación de Código my\_table.py**

Este código crea y gestiona una base de datos SQLite llamada db\_clinica, que contiene tres tablas: PATIENT, APPOINTMENT y USER\_DATA. La tabla PATIENT almacena información sobre los pacientes, como su ID, nombre, apellido, fecha de nacimiento y dirección. La tabla APPOINTMENT almacena información sobre las citas, como el ID de la cita, el ID del paciente, la fecha y hora de la cita y el motivo de la cita. La tabla USER\_DATA almacena información sobre los usuarios, como su nombre de usuario y contraseña.

La función register\_user() permite registrar un nuevo usuario en la tabla USER\_DATA. La función toma como argumentos el nombre de usuario y la contraseña, genera un hash de la contraseña utilizando el algoritmo pbkdf2\_sha256 y luego inserta el nombre de usuario y el hash de la contraseña en la tabla USER\_DATA. Si el registro se realiza con éxito, se imprime un mensaje de confirmación. En caso contrario, se muestra un mensaje de error.

Este código crea y gestiona una base de datos SQLite para almacenar información sobre pacientes, citas y usuarios en una clínica dental. Utiliza la biblioteca passlib.hash para generar y verificar hashes de contraseñas de manera segura.

**Bases de Datos db\_clinica**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Templates**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Add\_appointment.html**

El siguiente código HTML muestra un formulario para agregar citas en una clínica dental. Utiliza el framework W3.CSS y la fuente Raleway para darle estilo al formulario. Este código HTML crea un formulario para agregar citas en una clínica dental. El formulario contiene campos para ingresar el nombre del paciente, la fecha de la cita, la hora y el motivo de la cita. El formulario se presenta en el centro de la pantalla con una imagen de fondo y utiliza colores y estilos personalizados para mejorar la experiencia del usuario.

<!DOCTYPE html>

<html>

<head>

<title>Inicio</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">

<style>

body, h1 {

font-family: "Raleway", sans-serif;

margin: 0;

padding: 0;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr');

background-size: cover;

}

form {

background-color: rgba(0, 0, 0, 0.8);

padding: 20px;

border-radius: 10px;

width: 500px;

box-shadow: 0px 0px 10px rgba(0, 255, 255, 0.8);

text-align: center;

}

label {

display: block;

margin-bottom: 10px;

background-color: #0000;

text-align: center;

color: white;

font-weight: bold;

margin-bottom: 20px;

align-self: center;

}

input, textarea {

display: block;

width: 100%; /\* Added to make inputs take full width \*/

padding: 8px;

margin-bottom: 20px;

background-color: #00000;

color: black;

font-weight: bold;

border: 1px solid white; /\* Added border for better visibility \*/

border-radius: 5px;

}

input[type="submit"] {

background-color: #007bff;

color: #ffffff;

border: none;

padding: 10px 20px;

border-radius: 3px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #0056b3;

}

</style>

</head>

<body>

<form method="post" action="/appointment/add">

<label for="name\_last\_name">Paciente:</label>

<input type="text" name="name\_last\_name" required>

<label for="appo\_date">Fecha de Cita:</label>

<input type="date" name="appo\_date" required>

<label for="time">Hora:</label>

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

<label for="reason">Motivo:</label>

<textarea name="reason" required></textarea>

<input type="submit" value="Agregar Cita">

</form>

</body>

</html>

**Add\_patient.html**

Este código HTML crea un formulario para agregar pacientes en una clínica dental. El formulario contiene campos para ingresar el nombre, apellido, ID del paciente, fecha de nacimiento y dirección del paciente. El formulario utiliza el framework W3.CSS para darle estilo y la fuente Raleway para mejorar la apariencia del texto. El formulario se presenta en el centro de la pantalla con una imagen de fondo y utiliza colores y estilos personalizados para que sea memorable la experiencia del usuario.

<!DOCTYPE html>

<html>

<head>

<title>Agregar Paciente</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">

<style>

body, h1 {

font-family: "Raleway", sans-serif;

margin: 0;

padding: 0;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr');

background-size: cover;

}

form {

background-color: rgba(0, 0, 0, 0.8);

padding: 20px;

border-radius: 10px;

width: 500px;

box-shadow: 0px 0px 10px rgba(0, 255, 255, 0.8);

text-align: center;

}

label {

display: block;

margin-bottom: 10px;

background-color: #0000;

text-align: center;

color: white;

font-weight: bold;

margin-bottom: 20px;

align-self: center;

}

input, textarea {

display: block;

width: 100%; /\* Added to make inputs take full width \*/

padding: 8px;

margin-bottom: 20px;

background-color: #00000;

color: black;

font-weight: bold;

border: 1px solid white; /\* Added border for better visibility \*/

border-radius: 5px;

}

input[type="submit"] {

background-color: #007bff;

color: #ffffff;

border: none;

padding: 10px 20px;

border-radius: 3px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #0056b3;

}

</style>

</head>

<body>

<form method="post" action="/patient/add">

<label for="name">Nombre:</label>

<input type="text" id="name" name="name" required>

<br>

<label for="last\_name">Apellido:</label>

<input type="text" id="last\_name" name="last\_name" required>

<br>

<label for="id">ID del Paciente:</label>

<input type="text" id="id" name="id" required>

<br>

<label for="birthdate">Fecha de Nacimiento:</label>

<input type="date" id="birthdate" name="birthdate" required>

<br>

<label for="address">Dirección:</label>

<input type="text" id="address" name="address" required>

<br>

<input type="submit" value="Agregar Paciente">

</form>

</body>

</html>

**Appointment.html**

El siguiente código HTML muestra una página de citas para una clínica dental. La página permite buscar citas por nombre y apellido del paciente y muestra información sobre las citas encontradas. Además, proporciona botones para agregar una nueva cita, acceder al panel de control y ver la lista de pacientes. El diseño de la página utiliza estilos personalizados y una imagen de fondo.

Este código utiliza estilos personalizados para dar formato a la página y utiliza la sintaxis de plantillas Jinja2 para mostrar información dinámica sobre las citas. Los botones permiten navegar a diferentes partes de la aplicación, como agregar una nueva cita o ver la lista de pacientes.

<!DOCTYPE html>

<html>

<head>

<title>Citas</title>

<style>

body {

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1');

background-size: cover;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

background-color: rgba(255, 255, 255, 0.8);

padding: 20px;

border-radius: 10px;

display: flex;

flex-direction: column;

align-items: center;

}

.bar-link {

display: block;

width: 200px;

padding: 10px;

background-color: #2096CD;

border: 1px solid gray;

text-align: center;

text-decoration: none;

color: white;

font-weight: bold;

margin-bottom: 5px;

align-self: flex-start;

}

.content {

background-color: rgba(54, 43, 43, 0.8);

padding: 20px;

border-radius: 10px;

display: flex;

flex-direction: column;

align-items: center;

}

h1,label, .w3-card-4, .errores, .modificar {

color: white;

}

</style>

</head>

<body>

<div class="content">

<h1>Citas</h1>

<form method="post" action="/appointment/search">

<label for="name\_last\_name"></label>

<input type="text" name="name\_last\_name" required>

<input type="submit" value="Buscar">

</form>

<div class="modificar">

{% if name\_last\_name %}

<p><strong>Nombre y Apellido:</strong> {{ name\_last\_name }}</p>

{% endif %}

{% if appo\_review %}

</div>

<div class="w3-card-4">

<p><strong>ID del Paciente:</strong> {{ appo\_review[1] }}</p>

<p><strong>Fecha de Cita:</strong> {{ appo\_review[2] }}</p>

<p><strong>Hora:</strong> {{ appo\_review[3] }}</p>

<p><strong>Motivo:</strong> {{ appo\_review[4] }}</p>

</div>

<div class="errores">

{% else %}

<p>{{ error }}</p>

{% endif %}

</div>

<form method="get" action="/appointment/add">

<button type="submit" style="background-color: #007bff; color: #ffffff; border: none; padding: 10px 20px; border-radius: 3px; cursor: pointer;">Agregar Cita</button>

</form>

<form method="get" action="{{ url\_for('dashboard') }}">

<button type="submit" style="background-color: #007bff; color: #ffffff; border: none; padding: 10px 20px; border-radius: 3px; cursor: pointer;">Dashboard</button>

</form>

<form method="get" action="{{ url\_for('patient\_form') }}">

<button type="submit" style="background-color: #007bff; color: #ffffff; border: none; padding: 10px 20px; border-radius: 3px; cursor: pointer;">Pacientes</button>

</form>

</div>

</body>

</html>

**Dashboard.html**

El siguiente código HTML muestra una página de inicio o "Dashboard" para una aplicación web de citas en una clínica dental. La página presenta un menú con enlaces para navegar a diferentes secciones de la aplicación, como citas y pacientes, y un botón para cerrar sesión. El diseño de la página utiliza estilos personalizados y una imagen de fondo. Esta página de inicio o "Dashboard" sirve como punto de entrada para la aplicación web de citas en una clínica dental. Los usuarios pueden navegar a diferentes secciones de la aplicación, como citas y pacientes, utilizando los enlaces proporcionados en el menú. Además, pueden cerrar sesión utilizando el botón "Log Out". El diseño de la página es atractivo y fácil de usar, con estilos personalizados y una imagen de fondo que mejora la experiencia del usuario.

<!DOCTYPE html>

<html>

<head>

<title>Dashboard</title>

<style>

body {

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1');

background-size: cover;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

background-color: rgba(255, 255, 255, 0.8);

padding: 20px;

border-radius: 10px;

display: flex;

flex-direction: column;

align-items: center;

}

.bar-link {

display: block;

width: 200px;

padding: 10px;

background-color: #2096CD;

border: 1px solid gray;

text-align: center;

text-decoration: none;

color: white;

font-weight: bold;

margin-bottom: 5px;

align-self: flex-start;

}

</style>

</head>

<body>

<div class="container">

<h1>Bienvenido al sistema de citas para la clínica dental</h1>

<form action="/dash" method="post">

<a class="bar-link" href="#" onclick="selectOption(1)">Inicio</a>

<a class="bar-link" href="#" onclick="selectOption(2)">Citas</a>

<a class="bar-link" href="#" onclick="selectOption(3)">Pacientes</a>

<input type="hidden" id="selected\_option" name="opcion">

</form>

<form action="/logout" method="get">

<button type="submit" class="bar-link" style="background-color: red;">Log Out</button>

</form>

</div>

<script>

function selectOption(option) {

document.getElementById("selected\_option").value = option;

document.querySelector("form").submit();

}

</script>

</body>

</html>

**Login.html**

El siguiente código HTML muestra una página de inicio de sesión para una aplicación web. La página presenta un formulario de inicio de sesión con campos para ingresar el nombre de usuario y la contraseña. El diseño de la página utiliza estilos personalizados y una imagen de fondo. Esta página de inicio de sesión permite a los usuarios ingresar su nombre de usuario y contraseña para acceder a la aplicación web. El diseño de la página es atractivo y fácil de usar, con estilos personalizados y una imagen de fondo que mejora la experiencia del usuario.

<!DOCTYPE html>

<html>

<head>

<title>Inicio</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">

<style>

body, h1 {

font-family: "Raleway", sans-serif;

margin: 0;

padding: 0;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr');

background-size: cover;

}

form {

background-color: rgba(0, 0, 0, 255);

padding: 20px;

border-radius: 5px;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

text-align: center;

}

label {

font-weight: bold;

}

input {

width: 100%;

padding: 10px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

background-color: #007bff;

color: #ffffff;

border: none;

padding: 10px 20px;

border-radius: 3px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

p {

color: #ff0000;

text-align: center;

}

</style>

</head>

<body>

<div class="bgimg w3-display-container w3-animate-opacity w3-text-white">

<form method="post" action="/login" class="w3-display-middle">

<label for="username">Usuario:</label>

<input type="text" id="username" name="username" required>

<br>

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

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

<br>

<button type="submit" class="w3-button w3-blue">Iniciar sesión</button>

</form>

</div>

</body>

</html>

**Patient.html**

El siguiente código HTML muestra una página de búsqueda y agregar pacientes para una aplicación web de citas en una clínica dental. La página presenta un formulario para buscar pacientes por nombre y apellido y muestra información sobre los pacientes encontrados. Además, proporciona un botón para agregar un nuevo paciente. El diseño de la página utiliza estilos personalizados y una imagen de fondo.

<!DOCTYPE html>

<html>

<head>

<title>Búsqueda y Agregar de Paciente</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">

<style>

body, h1 {

font-family: "Raleway", sans-serif;

margin: 0;

padding: 0;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

background-image: url('https://images.pexels.com/photos/3845810/pexels-photo-3845810.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr');

background-size: cover;

}

.content {

background-color: rgba(0, 0, 0, 0.8);

padding: 20px;

border-radius: 10px;

width: 500px;

box-shadow: 0px 0px 10px rgba(0, 255, 255, 0.8);

text-align: center;

margin: 20px auto;

color: white

}

label {

display: block;

margin-bottom: 10px;

background-color: #0000;

text-align: center;

color: white;

font-weight: bold;

margin-bottom: 20px;

align-self: center;

}

input, select, textarea {

display: block;

width: 100%;

padding: 8px;

margin-bottom: 20px;

background-color: #00000;

color: black;

font-weight: bold;

border: 1px solid white;

border-radius: 5px;

}

input[type="submit"] {

background-color: #007bff;

color: #ffffff;

border: none;

padding: 10px 20px;

border-radius: 3px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #0056b3;

}

</style>

</head>

<body>

<div class="content">

<form method="post" action="/patient/search">

<label for="name\_last\_name">Paciente:</label>

<input type="text" id="name\_last\_name" name="name\_last\_name" required>

<input type="submit" value="Buscar Paciente">

</form>

{% if error %}

<p>{{ error }}</p>

{% endif %}

<form method="get" action="/patient/add">

<input type="submit" value="Agregar Paciente">

</form>

{% if review %}

<h2>Resultado de la búsqueda:</h2>

<table>

<tr>

<th>Nombre</th>

<th>ID</th>

<th>Fecha de nacimiento</th>

<th>Direccion</th>

</tr>

<tr>

<td>{{ review[1] }} {{ review[2] }}</td>

<td>{{ review[0] }}</td>

<td>{{ review[3] }}</td>

<td>{{review[4] }}</td>

</tr>

</table>

<h2>Editar Paciente:</h2>

<form method="post" action="/patient/edit">

<select name="column\_update">

<option value="name">Nombre</option>

<option value="last\_name">Apellido</option>

<option value="id">ID</option>

<option value="birthdate">Fecha de Nacimiento</option>

<option value="address">Dirección</option>

</select>

<input type="text" name="new\_data" placeholder="Nuevo valor">

<input type="hidden" name="name" value="{{ review[1] }}">

<input type="hidden" name="last\_name" value="{{ review[2] }}">

<input type="hidden" name="id" value="{{ review[0] }}">

<input type="hidden" name="birthdate" value="{{ review[3] }}">

<input type="hidden" name="address" value="{{ review[4] }}">

<input type="submit" value="Editar">

</form>

<h2>Eliminar Paciente:</h2>

<form method="post" action="/patient/delete">

<input type="hidden" name="name" value="{{ review[1] }}">

<input type="hidden" name="last\_name" value="{{ review[2] }}">

<input type="submit" value="Eliminar" onclick="return confirm('¿Estás seguro de que deseas eliminar este paciente?')">

</form>

{% endif %}

<form method="get" action="{{ url\_for('dashboard') }}">

<button type="submit" style="background-color: #007bff; color: #ffffff; border: none; padding: 10px 20px; border-radius: 3px; cursor: pointer;">Dashboard</button>

</form>

<form method="get" action="{{ url\_for('appointment\_form') }}">

<button type="submit" style="background-color: #007bff; color: #ffffff; border: none; padding: 10px 20px; border-radius: 3px; cursor: pointer;">Citas</button>

</form>

</div>

</body>

</html>

**CONCLUSIONES**

* La creación y desarrollo de la aplicación web para el registro de citas en el ámbito odontológico utilizando tecnologías como Flask, una API personalizada, HTML y la gestión de la base de datos a través de DBeaver ha resultado en una solución eficiente y versátil. A lo largo de este proyecto, se ha logrado combinar de manera exitosa la potencia de la programación backend con la experiencia de usuario ofrecida por el frontend, proporcionando una herramienta integral para la gestión de citas en clínicas dentales.
* La elección de Flask como framework backend permitió una implementación ágil y modular, facilitando la creación de rutas y endpoints para manejar las operaciones de la aplicación. La incorporación de una API personalizada fortaleció la conectividad entre los distintos componentes de la aplicación, garantizando una comunicación fluida y segura.
* El desarrollo del frontend utilizando HTML posibilitó la creación de interfaces amigables y accesibles para el usuario final. La presentación clara de la información y la interacción intuitiva a través de registro de citas mejorando la experiencia del usuario y fomentando una adopción más sencilla de la aplicación.
* La elección de DBeaver como herramienta de gestión de la base de datos permitió un modelado y diseño eficiente de la estructura de datos subyacente. La capacidad de DBeaver para interactuar con diversas bases de datos facilitó la elección de la opción más adecuada para el proyecto y simplificó la manipulación de los datos de las citas y los registros de los pacientes.
* En conjunto, este proyecto no solo logró cumplir con los objetivos establecidos, sino que también sentó las bases para una solución escalable y adaptable en el ámbito de la gestión de citas dentales. La colaboración entre las tecnologías backend y frontend, respaldada por una base de datos sólida, demostró ser esencial para crear una experiencia integral y satisfactoria tanto para los profesionales de la salud dental como para los pacientes.
* En conclusión, la aplicación web desarrollada con Flask, una API personalizada, HTML y la base de datos DBeaver se erige como un ejemplo exitoso de cómo la integración de tecnologías diversas puede dar lugar a una solución completa y efectiva en el ámbito de la gestión de citas. El resultado final no solo cumple con los requisitos funcionales, sino que también sienta las bases para futuras mejoras y adaptaciones en respuesta a las necesidades cambiantes de la industria odontológica.

**BIBLIOGRAFÍA**

Flask (2023). Flask Documentation. Recuperado de https://flask.palletsprojects.com

DBeaver (2023). DBeaver Community | Free Universal Database Tool. Recuperado de https://dbeaver.io

DigitalOcean (2021). How To Use Templates in a Flask Application. Recuperado de https://www.digitalocean.com/community/tutorials/how-to-use-templates-in-a-flask-application

GeeksforGeeks (2023). Python Flask - Request Object. Recuperado de https://www.geeksforgeeks.org/python-flask-request-object/

Full Stack Python (2012). flask.templating render\_template Example Code. Recuperado de https://www.fullstackpython.com/flask-templating-render-template-examples.html

Wikipedia (2010). Flask (web framework). Recuperado de https://en.wikipedia.org/wiki/Flask\_(web\_framework)

Wikipedia (2017). DBeaver. Recuperado de https://en.wikipedia.org/wiki/DBeaver

MDN Web Docs (2023). MDN Web Docs. Recuperado de https://developer.mozilla.org