INFORME PROYECTO RUEDA DE LA VIDA

MICHELLE RAMIREZ AGUDELO

YOSKAR ORDOSGOITTI IDLER

SEBASTIAN ARANGO TABORDA

MEDELLIN ANTIOQUIA

SENA (CTMA)

2025

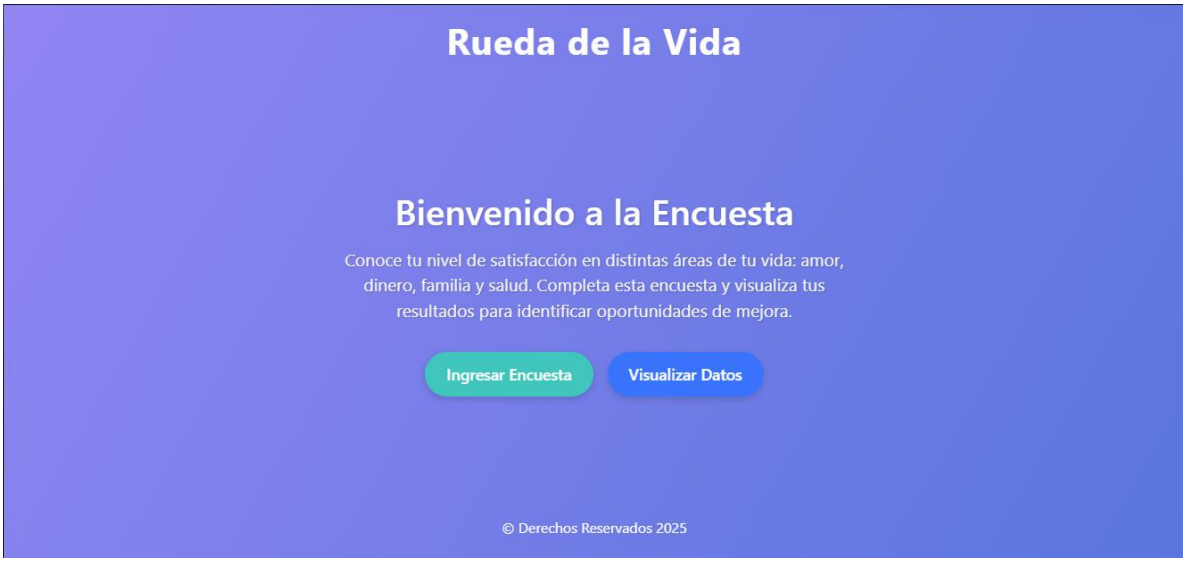
INTRODUCCION

El presente informe detalla de manera integral el proceso de desarrollo del proyecto *La Rueda de la Vida*, una iniciativa que fusiona la tecnología y el autoconocimiento para evaluar aspectos fundamentales de la existencia humana. En esta versión del proyecto, se han incorporado nuevas funcionalidades a los formularios interactivos, permitiendo ahora el almacenamiento y gestión de datos a través de una base de datos diseñada específicamente para este propósito. Se documenta la creación de esta base de datos, la definición de sus campos y su integración con los formularios desarrollados con HTML, Bootstrap y CSS.

Además, se han implementado nuevos formularios para mejorar la experiencia del usuario: un formulario de bienvenida que direcciona a la encuesta y a los resultados, un formulario de resultados que muestra la evaluación obtenida y un formulario de despedida que almacena la información final en la base de datos. Todo el desarrollo se ha llevado a cabo dentro de un entorno virtual, optimizando la organización y ejecución del proyecto.

DESARROLLO

Formulario main.html

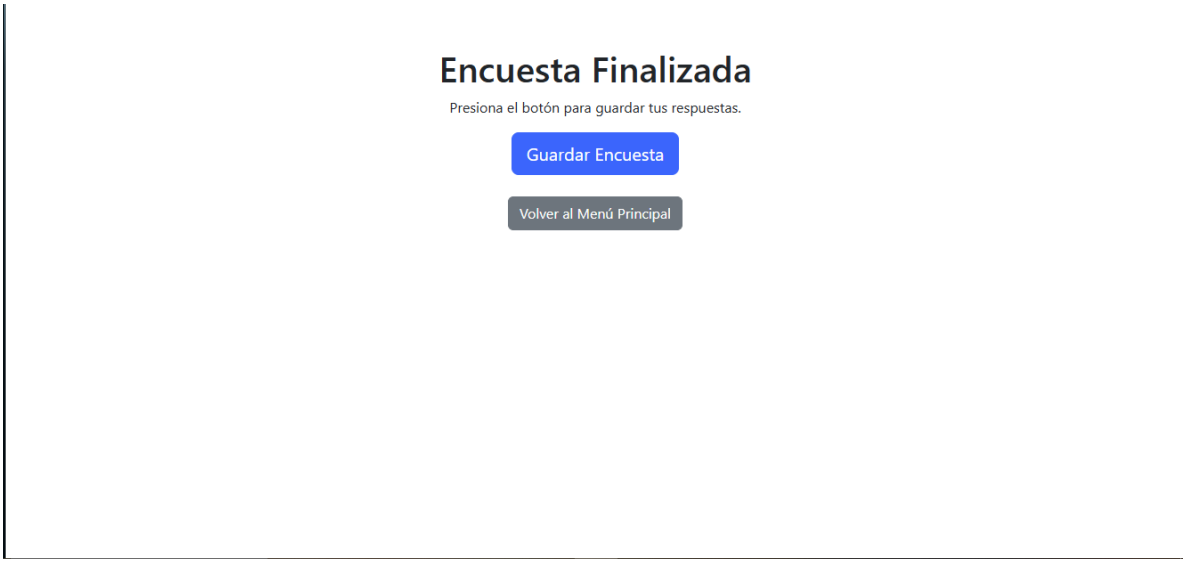


Se agrega un nuevo formulario de presentación que nos muestra un mensaje de bienvenida a la encuesta teniendo las opciones de dirigirnos a las encuestas o a ver los resultados de las personas que han hecho la encuesta.

Código formulario main.hmtl

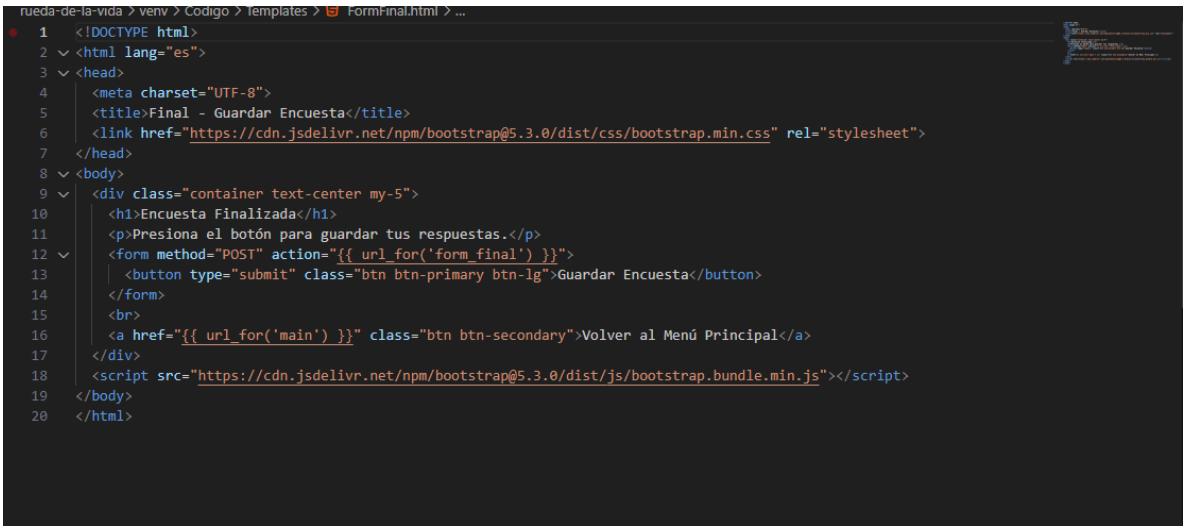
```
122   <!-- Sección principal estilo "hero" -->
123   <div class="hero">
124     <h2>Bienvenido a la Encuesta</h2>
125     <p>
126       Conoce tu nivel de satisfacción en distintas áreas de tu vida: amor, dinero, familia y salud.
127       Completa esta encuesta y visualiza tus resultados para identificar oportunidades de mejora.
128     </p>
129     <div class="btn-container">
130       <a href="{{ url_for('personal') }}" class="btn btn-custom btn-ingresar">Ingresar Encuesta</a>
131       <a href="{{ url_for('ver_resultados') }}" class="btn btn-custom btn-visualizar">Visualizar Datos</a>
132     </div>
133   </div>
134
135   <!-- Footer -->
136   <footer>
137     <p>&copy; Derechos Reservados 2025</p>
138   </footer>
139
140   <!-- Bootstrap JS -->
141   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
142 </body>
143 </html>
144
```

Formulario final FormFinal.html



Este nuevo formulario agrega muestre dos botones luego de finalizar con la encuesta, estos dos botones sirven para guardar la encuesta y volver al menú principal

Código FormFinal.hmtl



```
rueda-de-la-vida > venv > Codigo > Templates > FormFinal.html > ...
1   <!DOCTYPE html>
2   <html lang="es">
3   <head>
4     <meta charset="UTF-8">
5     <title>Final - Guardar Encuesta</title>
6     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
7   </head>
8   <body>
9     <div class="container text-center my-5">
10      <h1>Encuesta Finalizada</h1>
11      <p>Presiona el botón para guardar tus respuestas.</p>
12      <form method="POST" action="{{ url_for('form_final') }}">
13        <button type="submit" class="btn btn-primary btn-lg">Guardar Encuesta</button>
14      </form>
15      <br>
16      <a href="{{ url_for('main') }}" class="btn btn-secondary">Volver al Menú Principal</a>
17    </div>
18    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
19  </body>
20  </html>
```

Formulario de resultados resultados.html

Este formulario muestra los datos de las personas que realizaron la encuesta incluyendo su resultado en la encuesta también cuenta con un botón de eliminar y la opción de redirigirse al menú principal.

Código resultados.html

```
<body>

  <div class="container">
    <h1 class="text-center">Resultados Guardados</h1>
    <table class="table table-striped table-bordered table-hover">
      <thead class="table-dark">
        <tr>
          <th>ID</th>
          <th>Nombre</th>
          <th>Edad</th>
          <th>Departamento</th>
          <th>Amor</th>
          <th>Dinero</th>
          <th>Familia</th>
          <th>Salud</th>
          <th>Acciones</th>
        </tr>
      </thead>
      <tbody>
        {% if respuestas %}
          {% for r in respuestas %}
          <tr>
            <td>{{ r.id }}</td>
            <td>{{ r.nombre_completo }}</td>
```

Función convertir_a_numero

```
Tabnine | Edit | Test | Explain | Document
def convertir_a_numeros(lista):
    """Convierte una lista de strings en una lista de enteros."""
    return [int(x) for x in lista if x and x.isdigit()]
```

Esta función convierte una lista de cadenas de texto en una lista de números enteros. Para ello, verifica que cada elemento de la lista contenga únicamente dígitos antes de convertirlo en entero. Finalmente, retorna una nueva lista con los valores convertidos.

Función calcular porcentaje

```
Tabnine | Edit | Test | Explain | Document
def calcular_porcentaje(lista):
    """Calcula el porcentaje de una categoría."""
    if not lista:
        return 0
    return (sum(lista) / 50) * 25  # 50 es el máximo puntaje posible
```

Esta función calcula el porcentaje de una categoría a partir de los valores de una lista. Primero, verifica si la lista está vacía y, en ese caso, retorna 0. Luego, suma los valores de la lista y los escala en función de un puntaje máximo de 50, devolviendo el resultado como porcentaje.

Función procesar_datos

```
Tabnine | Edit | Test | Explain | Document
def procesar_datos():
    """Convierte respuestas a números y calcula los porcentajes."""
    global respuestas_amor, respuestas_dinero, respuestas_familia, respuestas_salud
    global porcentaje_amor, porcentaje_dinero, porcentaje_familia, porcentaje_salud

    respuestas_amor = convertir_a_numeros(respuestas_amor)
    respuestas_dinero = convertir_a_numeros(respuestas_dinero)
    respuestas_familia = convertir_a_numeros(respuestas_familia)
    respuestas_salud = convertir_a_numeros(respuestas_salud)

    porcentaje_amor = calcular_porcentaje(respuestas_amor)
    porcentaje_dinero = calcular_porcentaje(respuestas_dinero)
    porcentaje_familia = calcular_porcentaje(respuestas_familia)
    porcentaje_salud = calcular_porcentaje(respuestas_salud)
```

Convierte las respuestas almacenadas en listas de números y calcula los porcentajes correspondientes.

Función main

```python
@app.route("/")
def main():
    return render_template("Main.html")
```

Muestra la página principal de la aplicación web.

Funciona personal

```python
def personal():
    global global_nombre, global_edad, global_departamento
    if request.method == "POST":
        global_nombre = request.form["nombre"]
        global_edad = int(request.form["edad"])
        global_departamento = request.form["departamento"]
        print(f"Datos personales: {global_nombre}, {global_edad}, {global_departamento}")
        return redirect(url_for("form_amor"))
    return render_template("FormInfoPersonal.html")
```

Recibe y almacena los datos personales ingresados en el formulario, luego redirige al formulario de la categoría “amor”.

Funciona form_amor

```python
Tabnine | Edit | Test | Explain | Document
@app.route("/amor", methods=["GET", "POST"])
def form_amor():
    global respuestas_amor
    if request.method == "POST":
        respuestas_amor = [
            request.form.get("pregunta1"),
            request.form.get("pregunta2"),
            request.form.get("pregunta3"),
            request.form.get("pregunta4"),
            request.form.get("pregunta5")
        ]
        return redirect(url_for("form_dinero"))
    return render_template("FormAmor.html")
```

Guarda las respuestas del formulario de la categoría “amor” y redirige al formulario de la categoría “dinero”.

Función form_dinero

```python
@app.route("/dinero", methods=["GET", "POST"])
def form_dinero():
    global respuestas_dinero
    if request.method == "POST":
        respuestas_dinero = [
            request.form.get("pregunta1"),
            request.form.get("pregunta2"),
            request.form.get("pregunta3"),
            request.form.get("pregunta4"),
            request.form.get("pregunta5")
        ]
        return redirect(url_for("form_familia"))
    return render_template("FormDinero.html")
```

Guarda las respuestas del formulario de la categoría “dinero” y redirige al formulario de la categoría “familia”.

Función form_familia

```python
Tabnine | Edit | Test | Explain | Document
@app.route("/familia", methods=["GET", "POST"])
def form_familia():
    global respuestas_familia
    if request.method == "POST":
        respuestas_familia = [
            request.form.get("pregunta1"),
            request.form.get("pregunta2"),
            request.form.get("pregunta3"),
            request.form.get("pregunta4"),
            request.form.get("pregunta5")
        ]
        return redirect(url_for("form_salud"))
    return render_template("FormFamilia.html")
```

Guarda las respuestas del formulario de la categoría “familia” y redirige al formulario de la categoría “salud”.

Función form_salud

```python
@app.route("/salud", methods=["GET", "POST"])
def form_salud():
    global respuestas_salud
    if request.method == "POST":
        respuestas_salud = [
            request.form.get("pregunta1"),
            request.form.get("pregunta2"),
            request.form.get("pregunta3"),
            request.form.get("pregunta4"),
            request.form.get("pregunta5")
        ]
        return redirect(url_for("form_final"))
    return render_template("FormSalud.html")

```

Guarda las respuestas del formulario de la categoría “salud” y redirige a la página final.

Función form_final

```python
@app.route("/final", methods=["GET", "POST"])
def form_final():
    global global_nombre, global_edad, global_departamento
    if request.method == "POST":
        procesar_datos()
        nueva_respuesta = Respuesta(
            nombre_completo=global_nombre,
            edad=global_edad,
            departamento=global_departamento,
            amor=porcentaje_amor,
            dinero=porcentaje_dinero,
            familia=porcentaje_familia,
            salud=porcentaje_salud
        )
        db.session.add(nueva_respuesta)
        db.session.commit()
        print("✅ Respuesta guardada en la base de datos.")
        # Tras guardar, redirigimos al menú principal
        return redirect(url_for("main"))
    return render_template("FormFinal.html")
```

Procesa y almacena los datos recopilados de los formularios en la base de datos. Luego, redirige al menú principal.

Función ver_resultados

```python
def ver_resultados():
    respuestas = Respuesta.query.all()
    print(f"📄 Se encontraron {len(respuestas)} registro(s) en la base de datos.")
    return render_template("Resultados.html", respuestas=respuestas)

```

Obtiene todas las respuestas almacenadas en la base de datos y las envía a la plantilla Resultados.html para su visualización.

Función elminar_respuesta

```python

Tabnine | Edit | Test | Explain | Document
@app.route("/eliminar/<int:respuesta_id>", methods=["POST"])
def eliminar_respuesta(respuesta_id):
    registro = Respuesta.query.get(respuesta_id)
    if registro:
        db.session.delete(registro)
        db.session.commit()
        print(f"Registro con ID {respuesta_id} eliminado.")
    else:
        print(f"No se encontró registro con ID {respuesta_id}.")
    return redirect(url_for("ver_resultados"))


```

Elimina un registro específico de la base de datos según su ID. Si el registro existe, lo borra y confirma la acción. Si no, muestra un mensaje indicando que no se encontró el registro. Finalmente, redirige a la página de resultados.

Clase respuesta

```python
# Definir modelo de datos
class Respuesta(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  # Agregar autoincrement=True
    nombre_completo = db.Column(db.String(100), nullable=False)
    edad = db.Column(db.Integer, nullable=False)
    departamento = db.Column(db.String(100), nullable=False)
    amor = db.Column(db.Float, nullable=False)
    dinero = db.Column(db.Float, nullable=False)
    familia = db.Column(db.Float, nullable=False)
    salud = db.Column(db.Float, nullable=False)
```

Define la estructura de los datos que se almacenarán en la base de datos, incluyendo campos como nombre, edad, departamento y porcentajes relacionados con diferentes aspectos.

Función _repr_

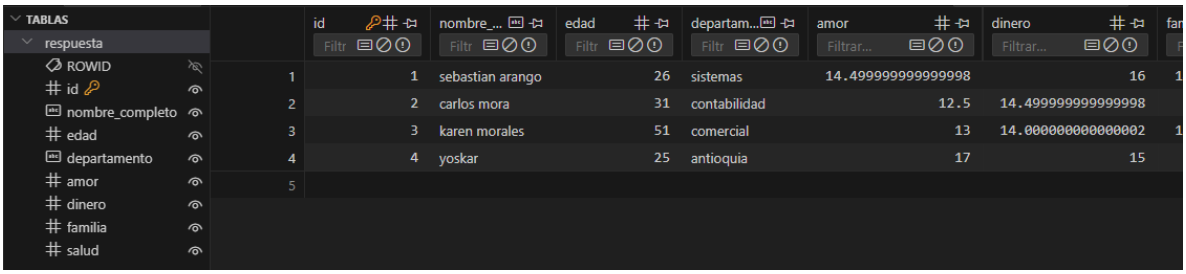
```python
Tabnine | Edit | Test | Explain | Document
def __repr__(self):
    return f'<Respuesta {self.nombre_completo} - Amor: {self.amor}% - Dinero: {self.dinero}% - Familia: {self.famili
```

Devuelve una representación en texto del objeto Respuesta, mostrando los valores almacenados de manera legible.

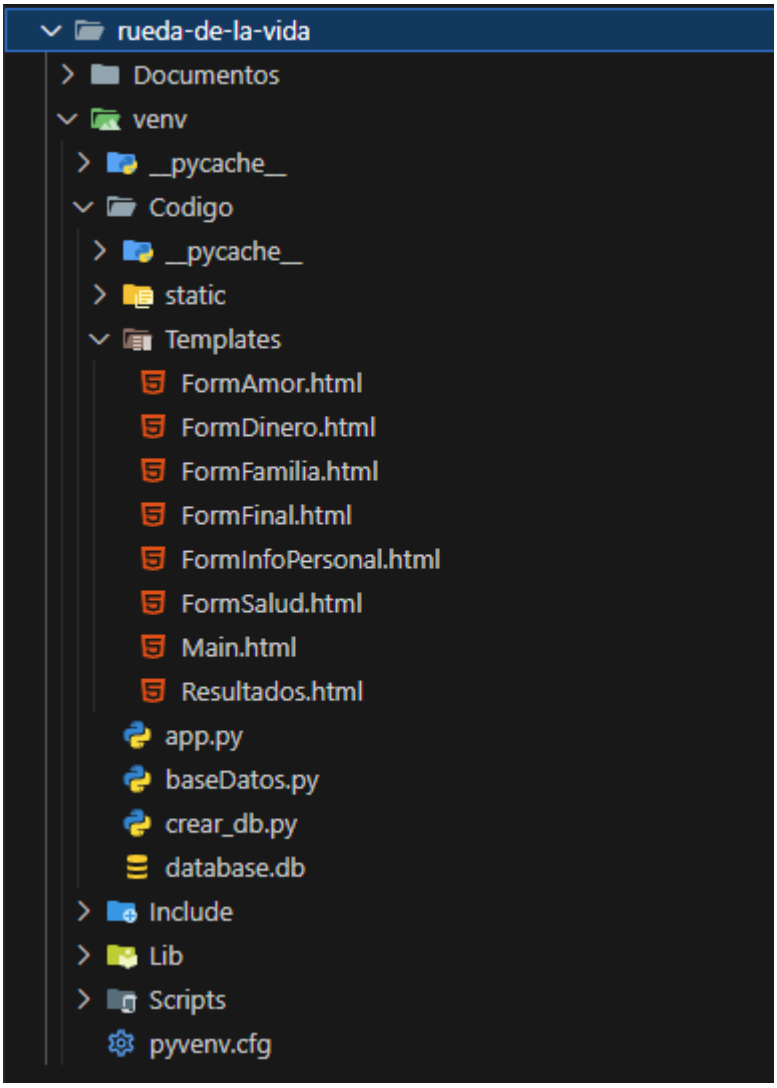
Crea base de datos

Este código crea la base de datos y sus tablas. Primero, importa app y db desde sus respectivos módulos. Luego, usa app.app_context() para establecer un contexto de aplicación, lo que permite que SQLAlchemy interactúe correctamente con la base de datos. Dentro de este contexto, db.create_all() genera las tablas definidas en los modelos de datos. Finalmente, imprime un mensaje confirmando que la base de datos fue creada exitosamente.

Base de tatos



aquí se muestra la tabla respuesta de la base de datos SQLite utilizando la extensión SQLite Viewer en Visual Studio Code. Se pueden ver los registros almacenados con sus respectivos campos (id, nombre_completo, edad, departamento, amor, etc.), lo que confirma que la base de datos ha sido creada y contiene datos.



En la imagen se puede observar la estructura organizada de los directorios y archivos del proyecto “rueda-de-la-vida”. Este proyecto está estructurado de la siguiente manera:

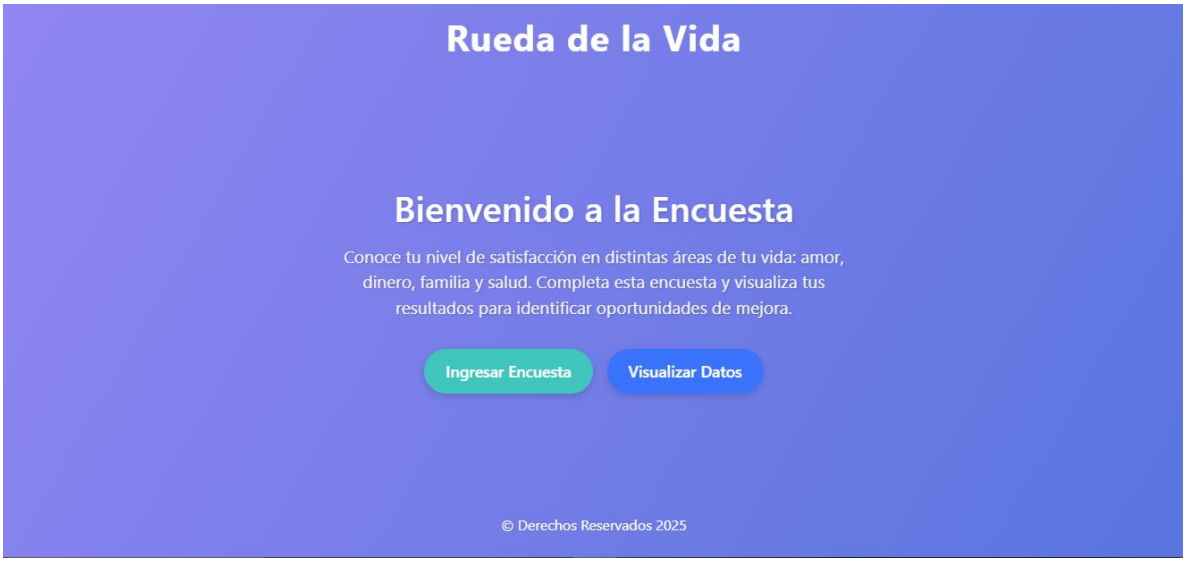
- venv/: Contiene el entorno virtual del proyecto, lo que permite gestionar dependencias de Python de forma aislada sin afectar otras instalaciones en el sistema. Dentro de este directorio se encuentran carpetas como Include, Lib, Scripts, y el archivo pyvenv.cfg, que son esenciales para el funcionamiento del entorno virtual.
- Codigo/: Es el directorio principal del código del proyecto. Dentro de él, encontramos:
    - static/: destinado a almacenar archivos estáticos como imágenes o CSS

- Templates/: Carpeta donde están almacenadas las plantillas HTML utilizadas en la aplicación, incluyendo formularios como FormAmor.html, FormDinero.html, Main.html, y Resultados.html, entre otros.
- app.py: Archivo principal de la aplicación, el punto de entrada donde se configura el servidor y las rutas.
- baseDatos.py y crear_db.py: Scripts relacionados con la gestión de la base de datos.
- database.db: Archivo de la base de datos, que sugiere el uso de SQLite.

Gracias al uso de un entorno virtual (venv), el proyecto puede ejecutarse con versiones específicas de paquetes sin interferir con otros proyectos o configuraciones del sistema.

Vista de formularios

Main.html



FormInfoPersonal.html

formAmor.html

FormDinero.html

formFamilia.html

formSalud.html

formFinal.html

Resultados.hmtl

CONCLUSION

Este proyecto consiste en una aplicación web desarrollada con Flask y SQLAlchemy que permite gestionar registros en una base de datos SQLite. A lo largo del desarrollo, se han implementado funcionalidades clave como la creación de la base de datos, la definición de un modelo de datos con atributos específicos, y la gestión de registros mediante rutas que permiten visualizar, agregar y eliminar datos.

El sistema almacena información de usuarios, incluyendo su nombre, edad, departamento y valores numéricos asociados a diferentes categorías como amor, dinero, familia y salud. Además, se ha utilizado SQLite Viewer en Visual Studio Code para verificar la estructura y contenido de la base de datos.

En conclusión, este proyecto es un ejemplo sólido de cómo combinar Flask, SQLAlchemy y SQLite para gestionar información de manera estructurada y eficiente, sentando las bases para futuras ampliaciones y mejoras, como la integración de una interfaz más avanzada o la adición de nuevas funcionalidades de análisis de datos.