
Proyecto Final Python

Versión 1.0

Pablo Techera

02 de septiembre de 2024

Contenidos:

1. Proyecto de Análisis de Composición Corporal	1
1.1. Descripción General del Proyecto	1
1.2. Objetivos del Proyecto	1
1.3. Alcance del Proyecto	2
2. Stack Tecnológico y Alternativas Evaluadas	3
3. Modelo de Datos	5
4. Requisitos de la Aplicación	7
4.1. pip install -r requirements.txt	9
5. Manual de Instalación	11
6. Código Fuente y Funcionalidad	13
7. Base de Datos y archivo models.py	19
8. Conclusiones	23
9. Evolutivos del Proyecto	25

Proyecto de Análisis de Composición Corporal

1.1 Descripción General del Proyecto

«**Análisis de Composición Corporal**» es una aplicación de escritorio desarrollada en Python, diseñada para obtener resultados de salud mediante cálculos biométricos.

Esta herramienta permite a profesionales de la salud, entrenadores personales, nutricionistas (como lo soy yo), calcular y hacer seguimiento de diversos parámetros de composición corporal. Además, se podrá escalar a mejoras en funcionalidad, estética de la interfaz de usuario y en términos de seguridad en la base de datos.

1.2 Objetivos del Proyecto

Los principales objetivos de este proyecto son:

- Proporcionar una interfaz intuitiva para la entrada de datos biométricos.
- Calcular con precisión diversos indicadores de composición corporal, incluyendo:
 - Índice de Masa Corporal (IMC)
 - Porcentaje de Grasa Corporal
 - Masa Muscular
 - Tasa Metabólica Basal (TMB)
 - FFMI (calidad de masa muscular y tope genético)
 - Ratio Cintura/Cadera
 - Reparto de Macronutrientes según objetivo.
- Almacenar y gestionar perfiles de clientes para un seguimiento a largo plazo (en desarrollo).
- Generar informes detallados sobre la composición corporal y su evolución en el tiempo (CSV).
- Ofrecer recomendaciones e interpretaciones personalizadas basadas en los resultados del análisis.

1.3 Alcance del Proyecto

Esta aplicación está diseñada para ser utilizada como herramienta complementaria por profesionales de la salud, como en clínicas de nutrición, gimnasios, consultas médicas, entrenadores personales, y cualquier persona que necesite medir parámetros biométricos.

Funcionalidades principales:

- Registro y autenticación de usuarios.
- Entrada y almacenamiento de datos de clientes.
- Cálculos automáticos de composición corporal.
- Visualización de resultados y tendencias.
- Generación de informes.
- Extracción de archivos.

Stack Tecnológico y Alternativas Evaluadas

Stack Tecnológico Utilizado:

- **Python:** Lenguaje de programación principal.
- **Tkinter:** Utilizado para la interfaz gráfica de usuario.
- **SQLite:** Base de datos utilizada para almacenar la información del usuario.

Alternativas Evaluadas:

- **Flask o Django:** Para una futura versión web de la aplicación.
- **Matplotlib:** Para implementar gráficos y visualizaciones en futuras versiones.

CAPÍTULO 3

Modelo de Datos

La aplicación utiliza **SQLite** como sistema de gestión de bases de datos para almacenar los datos de los usuarios. El modelo de datos incluye:

- **Usuarios:** Información personal del usuario, como nombre, edad, y sexo.
- **Mediciones:** Almacena las mediciones de composición corporal, incluyendo peso, altura, circunferencias, etc.
- **Resultados:** Resultados de los análisis realizados, como IMC, porcentaje de grasa corporal, entre otros.

Requisitos de la Aplicación

Requisitos de Hardware:

- Procesador: Intel o AMD de 2 GHz o superior.
- Memoria RAM: 4 GB o más recomendados.
- Espacio en Disco: 100 MB de espacio disponible.

Requisitos de Software:

- **Sistema Operativo:** Windows, macOS o Linux.
- **Python:** Versión 3.12.
- **Dependencias:** Las siguientes librerías deben ser instaladas para ejecutar la aplicación:
 - *alabaster==0.7.16*
 - *altgraph==0.17.4*
 - *anyio==4.4.0*
 - *attrs==24.2.0*
 - *babel==2.16.0*
 - *bcrypt==4.2.0*
 - *beautifulsoup4==4.12.3*
 - *bleach==6.1.0*
 - *certifi==2024.8.30*
 - *charset-normalizer==3.3.2*
 - *click==8.1.7*
 - *colorama==0.4.6*
 - *defusedxml==0.7.1*
 - *docutils==0.20.1*
 - *fastjsonschema==2.20.0*
 - *greenlet==3.0.3*

- *h11*==0.14.0
- *idna*==3.8
- *imagesize*==1.4.1
- *iniconfig*==2.0.0
- *Jinja2*==3.1.4
- *jsonschema*==4.23.0
- *jsonschema-specifications*==2023.12.1
- *jupyter_client*==8.6.2
- *jupyter_core*==5.7.2
- *jupyterlab_pygments*==0.3.0
- *macholib*==1.16.3
- *MarkupSafe*==2.1.5
- *mistune*==3.0.2
- *nbclient*==0.10.0
- *nbconvert*==7.16.4
- *nbformat*==5.10.4
- *nbsphinx*==0.9.5
- *packaging*==24.1
- *pandocfilters*==1.5.1
- *platformdirs*==4.2.2
- *pluggy*==1.5.0
- *Pygments*==2.18.0
- *pyinstaller*==6.10.0
- *pyinstaller-hooks-contrib*==2024.8
- *pytest*==8.3.2
- *python-dateutil*==2.9.0.post0
- *pyzmq*==26.2.0
- *referencing*==0.35.1
- *requests*==2.32.3
- *rpds-py*==0.20.0
- *setuptools*==72.2.0
- *six*==1.16.0
- *sniffio*==1.3.1
- *snowballstemmer*==2.2.0
- *soupsieve*==2.6
- *Sphinx*==7.4.7
- *sphinx-autobuild*==2024.4.16
- *sphinx-rtd-theme*==2.0.0
- *sphinxcontrib-applehelp*==2.0.0

- *sphinxcontrib-devhelp*==2.0.0
- *sphinxcontrib-htmlhelp*==2.1.0
- *sphinxcontrib-jquery*==4.1
- *sphinxcontrib-jsmath*==1.0.1
- *sphinxcontrib-qthelp*==2.0.0
- *sphinxcontrib-serializinghtml*==2.0.0
- *SQLAlchemy*==2.0.31
- *starlette*==0.38.4
- *tinycss2*==1.3.0
- *tornado*==6.4.1
- *traitlets*==5.14.3
- *typing_extensions*==4.12.2
- *urllib3*==2.2.2
- *uvicorn*==0.30.6
- *watchfiles*==0.24.0
- *webencodings*==0.5.1
- *websockets*==13.0.1

Para instalar estas dependencias, ejecute el siguiente comando en la terminal:

4.1 pip install -r requirements.txt

Manual de Instalación

Esta sección ofrece dos opciones para instalar y ejecutar la aplicación: clonar el repositorio desde GitHub o descomprimir el archivo `.zip` y ejecutar el proyecto desde un IDE.

Opción 1: Clonar el Repositorio desde GitHub

1. **Clonar el Repositorio:** - Primero, clona el repositorio desde GitHub a tu máquina local:

```
` git clone https://github.com/pablotech80/analisis_corporal.git `
```

2. **Navegar al Directorio del Proyecto:** - Cambia al directorio del proyecto clonado:

```
` cd tu_proyecto `
```

3. **Crear un Entorno Virtual (Opcional pero Recomendado):** - Crea y activa un entorno virtual para evitar conflictos con otras versiones de Python o dependencias instaladas globalmente.

```
` python -m venv .venv source .venv/bin/activate # En macOS/Linux .venv\Scripts\activate # En Windows `
```

4. **Instalar las Dependencias:** - Instala las dependencias necesarias para ejecutar la aplicación:

```
` pip install -r requirements.txt `
```

5. **Ejecutar la Aplicación:** - Inicia la aplicación ejecutando el archivo principal `main.py`:

```
` python main.py `
```

6. **Uso de la Aplicación:** - Sigue las instrucciones en pantalla para ingresar los datos biométricos y realizar el análisis de composición corporal.

Opción 2: Descomprimir el Archivo `.zip` y Ejecutar desde un IDE

1. **Descomprimir el Archivo `.zip`:** - Descomprime el archivo `.zip` que contiene el proyecto en tu máquina local.

2. **Abrir el Proyecto en un IDE:** - Abre un IDE como PyCharm, VSCode, o cualquier otro de tu preferencia.

3. **Configurar el Entorno Virtual (Opcional pero Recomendado):** - Configura un entorno virtual dentro del IDE:

- En PyCharm: Ve a *File > Settings > Project: tu_proyecto > Python Interpreter* y selecciona *Add Interpreter* para crear un entorno virtual.
- En VSCode: Usa la paleta de comandos (`Ctrl+Shift+P`) y selecciona *Python: Select Interpreter*, luego selecciona o crea un entorno virtual.

4. **Instalar las Dependencias:** - Instala las dependencias necesarias utilizando la terminal integrada del IDE:

```
` pip install -r requirements.txt `
```

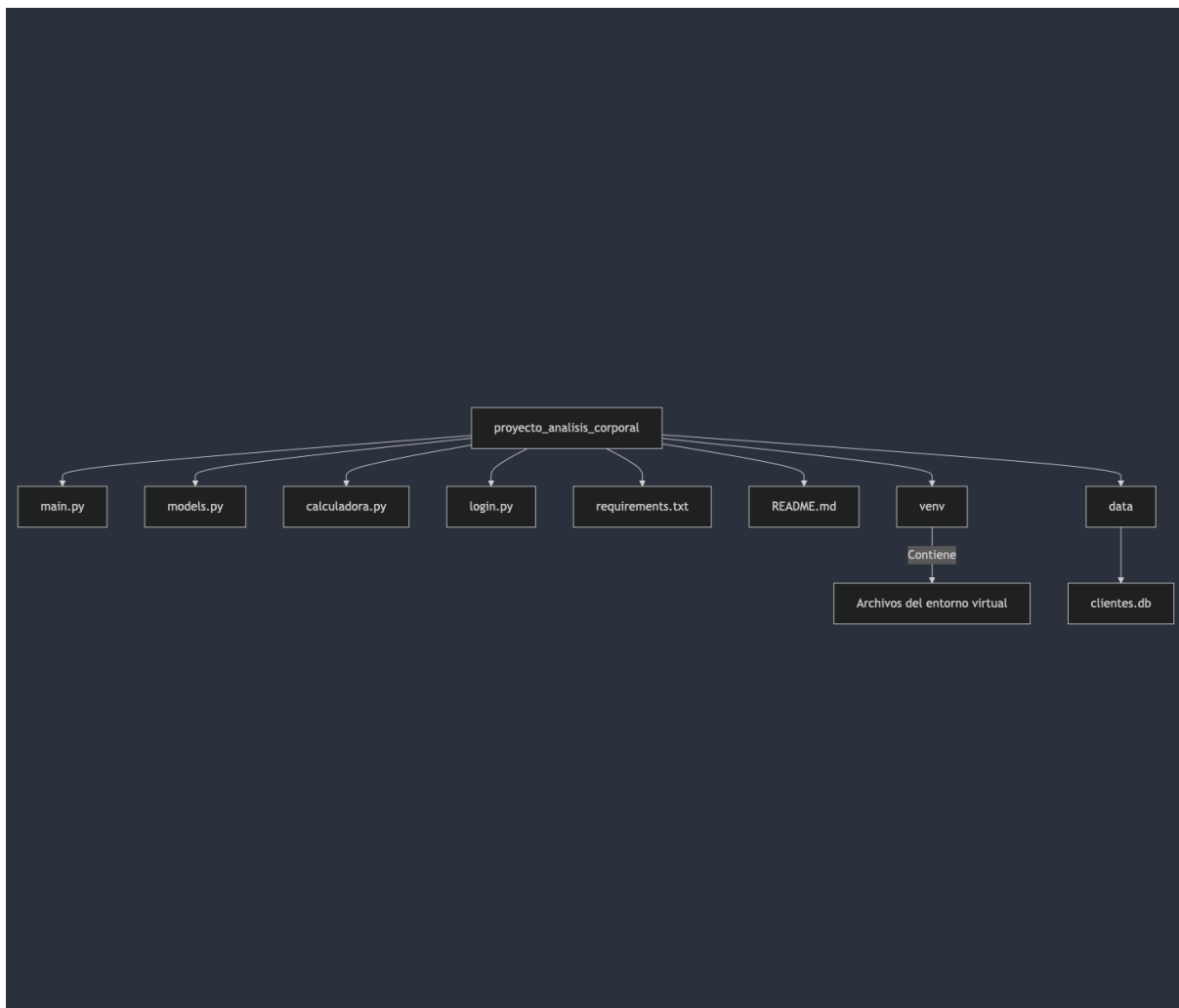
5. **Ejecutar la Aplicación:** - Ejecuta el archivo *main.py* directamente desde el IDE utilizando la opción de «Run» o «Ejecutar».
6. **Uso de la Aplicación:** - Sigue las instrucciones en pantalla para ingresar los datos biométricos y realizar el análisis de composición corporal.

Nota: Asegúrate de que tu entorno tiene acceso a Python 3.12 y las dependencias especificadas en el archivo *requirements.txt* para evitar problemas de compatibilidad.

Código Fuente y Funcionalidad

Recorrido por el flujo del programa aportando imágenes.

Directorio principal y los ficheros principales que componen la aplicacion.



Este diagrama muestra el flujo general del programa desde la entrada del usuario hasta la generación de los resultados.



main.py

Puedes ver el código completo en el siguiente enlace:

[Ver main.py en HTML](#)

El archivo *main.py* es el punto de entrada principal de la aplicación.

Captura de Pantalla: Interfaz Principal y vista de la clase `MainApplication`

```

class MainApplication:
    def setup_input_fields(self):
        """Configura los campos de entrada para recibir los datos del usuario."""
        # Nombre
        ttk.Label(self.frame, text="Nombre Completo:").grid(row=0, column=0, sticky=tk.W)
        self.nombre = tk.StringVar()
        ttk.Entry(self.frame, textvariable=self.nombre).grid(row=0, column=1, sticky=(tk.W, tk.E))

        # Peso
        ttk.Label(self.frame, text="Peso (kg):").grid(row=1, column=0, sticky=tk.W)
        self.peso = tk.StringVar()
        ttk.Entry(self.frame, textvariable=self.peso).grid(row=1, column=1, sticky=(tk.W, tk.E))

        # Altura
        ttk.Label(self.frame, text="Altura (cm):").grid(row=2, column=0, sticky=tk.W)
        self.altura = tk.StringVar()
        ttk.Entry(self.frame, textvariable=self.altura).grid(row=2, column=1, sticky=(tk.W, tk.E))

        # Edad
        ttk.Label(self.frame, text="Edad:").grid(row=3, column=0, sticky=tk.W)
        self.edad = tk.StringVar()
        ttk.Entry(self.frame, textvariable=self.edad).grid(row=3, column=1, sticky=(tk.W, tk.E))

        # Género
        ttk.Label(self.frame, text="Género (H/M):").grid(row=4, column=0, sticky=tk.W)
        self.genero = tk.StringVar()
        self.genero_entry = ttk.Entry(self.frame, textvariable=self.genero)
        self.genero_entry.grid(row=4, column=1, sticky=(tk.W, tk.E))
        self.genero_entry.bind('<KeyRelease>', self.actualizar_panel)

        # Objetivo
        ttk.Label(self.frame, text="Objetivo (mantener/perder/ganar):").grid(row=5, column=0, sticky=tk.W)
        self.objetivo = tk.StringVar()
        self.objetivo_entry = ttk.Entry(self.frame, textvariable=self.objetivo)
        self.objetivo_entry.grid(row=5, column=1, sticky=(tk.W, tk.E))
        self.objetivo_entry.bind('<KeyRelease>', self.actualizar_panel)

```

Captura de Pantalla: Vista de la clase LoginWindow y los métodos que controlan la interfaz de usuario.

```

class LoginWindow:
    """Gestiona la interfaz de usuario para el proceso de inicio de sesión y registro.

    Esta clase crea una ventana de inicio de sesión que permite a los usuarios ingresar
    o registrar credenciales para acceder a la aplicación de Análisis de Composición Corporal.

    Atributos:
        root (tk.Tk): La ventana raíz en la que se ejecuta la interfaz de inicio de sesión.
        frame (ttk.Frame): Contenedor para los widgets de entrada y botones.
        username (tk.StringVar): Variable de control para el nombre de usuario.
        password (tk.StringVar): Variable de control para la contraseña.
    """
    def __init__(self, root):
        """Inicializa la ventana de login con la configuración básica de la UI.

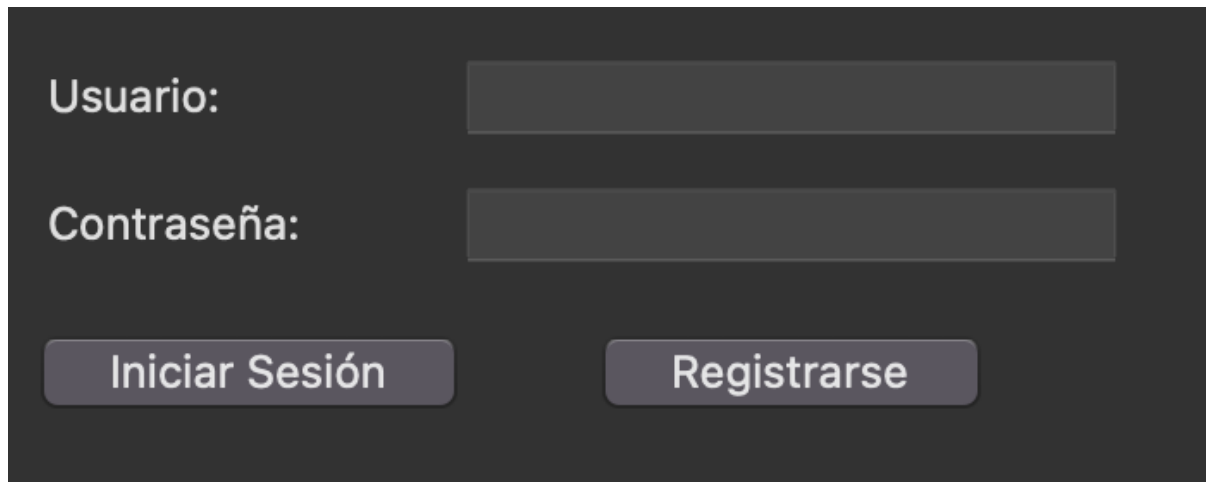
        Args:
            root (tk.Tk): La ventana principal de la aplicación.
        """
        self.root = root
        self.root.title("Login - Análisis de Composición Corporal")
        self.setup_ui()

    def setup_ui(self):
        """Configura los widgets de la interfaz de usuario."""
        self.frame = ttk.Frame(self.root, padding="10")
        self.frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        ttk.Label(self.frame, text="Usuario:").grid(row=0, column=0, sticky=tk.W, pady=5)
        self.username = tk.StringVar()
        ttk.Entry(self.frame, textvariable=self.username).grid(row=0, column=1, sticky=(tk.W, tk.E), pady=5)

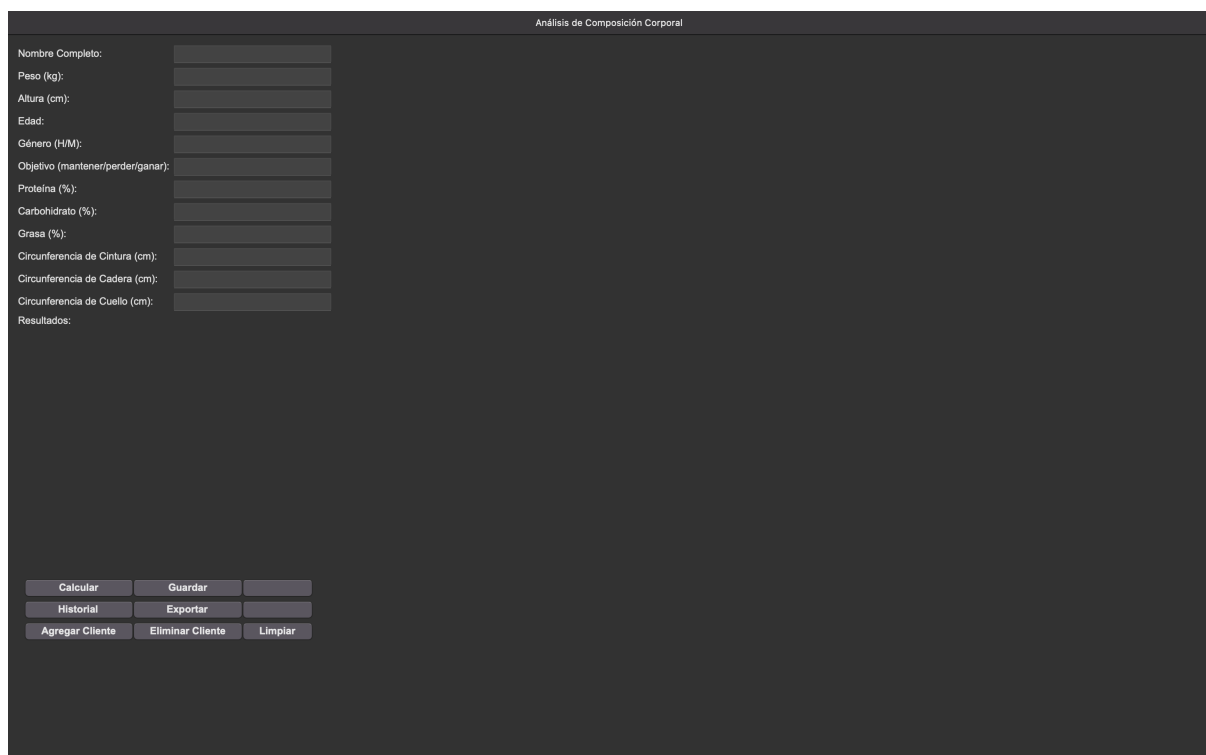
```

Captura de Pantalla: Vista de Inicio de la aplicación, donde se ingresan los datos de usuario.



A dark-themed login and registration form. It features two input fields: 'Usuario:' and 'Contraseña:'. Below these fields are two buttons: 'Iniciar Sesión' and 'Registrarse'.

Captura de Pantalla: Vista al lograr registrarse e inicializar el usuario, frame principal de la aplicación.



The main application frame, titled 'Análisis de Composición Corporal'. It contains a list of input fields for user data: 'Nombre Completo:', 'Peso (kg):', 'Altura (cm):', 'Edad:', 'Género (H/M):', 'Objetivo (mantener/perder/ganar):', 'Proteína (%)', 'Carbohidrato (%)', 'Grasa (%)', 'Circunferencia de Cintura (cm):', 'Circunferencia de Cadera (cm):', and 'Circunferencia de Cuello (cm):'. Below these fields is a 'Resultados:' section. At the bottom, there is a grid of buttons: 'Calcular', 'Guardar', 'Historial', 'Exportar', 'Agregar Cliente', 'Eliminar Cliente', and 'Limpiar'.

Captura de Pantalla: Vista con ingreso de datos del cliente, y salida de resultados del Botón «Calcular».

Análisis de Composición Corporal	
Nombre Completo:	Pablo Techera
Peso (kg):	89
Altura (cm):	165
Edad:	44
Género (H/M):	H
Objetivo (mantener/perder/ganar):	Perder
Proteína (%):	40
Carbohidrato (%):	40
Grasa (%):	20
Circunferencia de Cintura (cm):	98
Circunferencia de Cuello (cm):	41
Resultados: TMB: 1822.74 kcal/día IMC: 32.69 Porcentaje de Grasa: 25.89% (Alto) Peso de Grasa Corporal: 23.04 kg Masa Muscular: 65.96 kg Agua Total del Cuerpo: 46.06 litros FFM: 24.23 Interpretación FFM: Muy cerca del máximo potencial. Interpretación IMC: El IMC es alto, pero puede estar influenciado por una alta masa muscular. Peso Saludable: 50.37 kg - 67.79 kg Sobrepeso: 21.21 kg Relación Cintura/Cadera: 0.0 (N/A) Ratio Cintura/Altura: 0.59 (Moderado riesgo) Calorías Diarias Necesarias: 1749.83 kcal Macronutrientes: Proteínas: 174.98g, Carbohidratos: 174.98g, Grasas: 38.89g Porcentaje de grasa corporal dentro del rango normal.	
<div> <div>Calcular</div> <div>Guardar</div> </div> <div> <div>Historial</div> <div>Exportar</div> </div> <div> <div>Agregar Cliente</div> <div>Eliminar Cliente</div> <div>Limpiar</div> </div>	

#calculadora.py`

Puedes ver el código completo en el siguiente enlace:

[Ver calculadora.py en HTML](#)

El archivo calculadora.py contiene todos los métodos donde calculan con funciones matemáticas los resultados del análisis, mediante la toma de medidas del usuario.

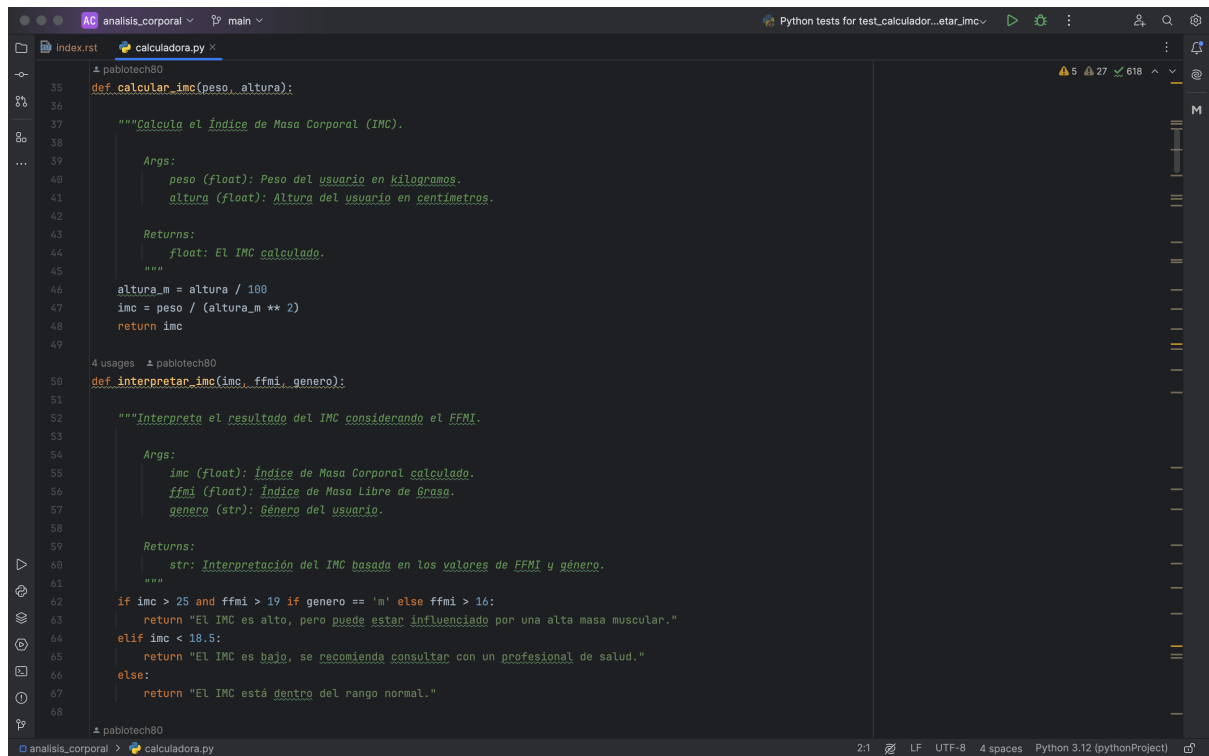
```

16 def calcular_tmb(peso, altura, edad, genero):
17
18     """Calcula la Tasa Metabólica Basal (TMB) usando la fórmula de Harris-Benedict.
19
20     Args:
21         peso (float): Peso del usuario en kilogramos.
22         altura (float): Altura del usuario en centímetros.
23         edad (int): Edad del usuario en años.
24         genero (str): Género del usuario ('h' para hombre, 'm' para mujer).
25
26     Returns:
27         float: La TMB calculada
28     """
29
30     if genero == 'h':
31         tmb = 88.362 + (13.397 * peso) + (4.799 * altura) - (5.677 * edad)
32     else:
33         tmb = 447.593 + (9.247 * peso) + (3.098 * altura) - (4.330 * edad)
34     return tmb
35
36
37 def calcular_imc(peso, altura):
38
39     """Calcula el Índice de Masa Corporal (IMC).
40
41     Args:
42         peso (float): Peso del usuario en kilogramos.
43         altura (float): Altura del usuario en centímetros.
44
45     Returns:
46         float: El IMC calculado.
47     """
48
49     altura_m = altura / 100
50     imc = peso / (altura_m ** 2)
51     return imc

```

Este archivo contiene las funciones para realizar los cálculos biométricos.

Captura de Pantalla: Cálculo de IMC



```
35 def calcular_imc(peso, altura):
36
37     """Calcula el Índice de Masa Corporal (IMC).
38
39     Args:
40         peso (float): Peso del usuario en kilogramos.
41         altura (float): Altura del usuario en centímetros.
42
43     Returns:
44         float: El IMC calculado.
45     """
46     altura_m = altura / 100
47     imc = peso / (altura_m ** 2)
48     return imc
49
50 4 usages 1 pablotech80
51 def interpretar_imc(imc, ffmi, genero):
52
53     """Interpreta el resultado del IMC considerando el FFMI.
54
55     Args:
56         imc (float): Índice de Masa Corporal calculado.
57         ffmi (float): Índice de Masa Libre de Grasa.
58         genero (str): Género del usuario.
59
60     Returns:
61         str: Interpretación del IMC basada en los valores de FFMI y género.
62     """
63     if imc > 25 and ffmi > 19 if genero == 'm' else ffmi > 16:
64         return "El IMC es alto, pero puede estar influenciado por una alta masa muscular."
65     elif imc < 18.5:
66         return "El IMC es bajo, se recomienda consultar con un profesional de salud."
67     else:
68         return "El IMC está dentro del rango normal."
```

Base de Datos y archivo models.py

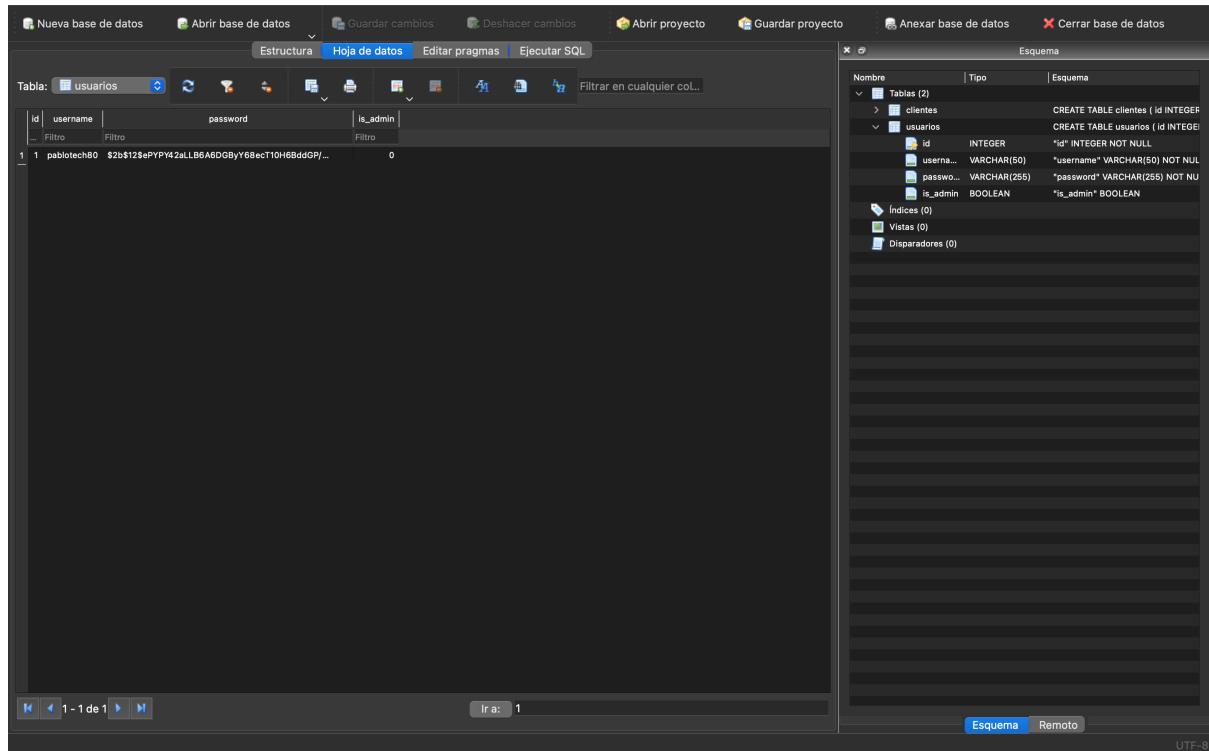
La aplicación utiliza SQLite para gestionar la información relacionada con los usuarios y sus mediciones.

Puedes ver el código completo en el siguiente enlace:

[Ver models.py en HTML](#)

Estructura de la Base de Datos

Captura de Pantalla: Estructura de la Base de Datos clase Usuario



Captura de Pantalla: Estructura de la Base de Datos clase Clientes

La base de datos incluye las siguientes tablas:

- **Usuarios:** Almacena la información personal de los usuarios, como nombre, edad y sexo.
- **Mediciones:** Contiene las mediciones de los usuarios, como peso, altura, circunferencias, etc.
- **Resultados:** Guarda los resultados de los cálculos realizados, incluyendo IMC, porcentaje de grasa corporal, y otros indicadores.

Conexión con la Base de Datos

El archivo *models.py* gestiona las interacciones con la base de datos, incluyendo la creación de tablas, inserción de datos y consultas. A continuación, se muestra un ejemplo de cómo se define un modelo de usuario:

Captura de Pantalla: Estructura de la Base de Datos clase Usuario


```
6 Base = declarative_base()
7
8 class Usuario(Base):
9
10     """
11     Representa un usuario en la base de datos, incluyendo administrador (admin no se ha configurado todavía)
12     y usuarios normales.
13
14     Atributos:
15     id (Integer): Clave primaria, identificador único del usuario.
16     username (String(50)): Nombre de usuario, debe ser único.
17     password (String(255)): Contraseña del usuario, almacenada de manera segura.
18     is_admin (Boolean): Indica si el usuario tiene privilegios de administrador.
19
20     Métodos:
21     set_password(self, password): Encripta y establece la contraseña del usuario.
22     check_password(self, password): Verifica si una contraseña proporcionada coincide con la almacenada.
23
24     """
25     __tablename__ = 'usuarios'
26     id = Column(Integer, primary_key=True)
27     username = Column(String(50), unique=True, nullable=False)
28     password = Column(String(255), nullable=False)
29     is_admin = Column(Boolean, default=False)
30
31     @pablotech80
32     def set_password(self, password):
33         self.password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')
34
35     @pablotech80
36     def check_password(self, password):
37         return bcrypt.checkpw(password.encode('utf-8'), self.password.encode('utf-8'))
38
39 class Cliente(Base):
40     """
41     Representa un cliente en la base de datos, almacenando información detallada
42     """
```

Captura de Pantalla: Estructura de la Base de Datos clase Clientes

```
68 class Cliente(Base):
69     __tablename__ = 'clientes'
70     id = Column(Integer, primary_key=True)
71     nombre = Column(String, nullable=False)
72     fecha = Column(DateTime, nullable=False)
73     peso = Column(Float, nullable=False)
74     altura = Column(Float, nullable=False)
75     edad = Column(Integer, nullable=False)
76     genero = Column(String(1), nullable=False)
77     cintura = Column(Float, nullable=False)
78     cadera = Column(Float, nullable=False)
79     cuello = Column(Float, nullable=False)
80     tab = Column(Float)
81     porcentaje_grasa = Column(Float)
82     peso_grasa = Column(Float)
83     masa_muscular = Column(Float)
84     agua_total = Column(Float)
85     fpmi = Column(Float)
86     peso_min = Column(Float)
87     peso_max = Column(Float)
88     sobrepeso = Column(Float)
89     rcc = Column(Float)
90     ratio_cintura_altura = Column(Float)
91     calorías_diarias = Column(Float)
92     proteínas = Column(Float)
93     carbohidratos = Column(Float)
94     grasas = Column(Float)
95
96     """Configuración de la base de datos"""
97
98     engine = create_engine('sqlite:///clientes.db')
99     Base.metadata.create_all(engine)
100
101     """Creo una sesión en la base de datos"""
102
103     Session = sessionmaker(bind=engine)
```


CAPÍTULO 8

Conclusiones

El desarrollo de esta aplicación proporciona una herramienta poderosa para analizar la composición corporal, ayudando a los usuarios a comprender mejor su estado físico y mental. Además, facilita a los entrenadores personales la personalización de planes de entrenamiento y nutrición.

Esta versión inicial del proyecto cumple con los objetivos propuestos, aunque hay margen para futuras mejoras y ampliaciones.

Evolutivos del Proyecto

En futuras versiones, se planea incluir:

- **Gráficos de Progreso:** Para un mejor seguimiento de la evolución del usuario, usando librería Matplotlib.
- **Versión Web:** Implementación de una versión web utilizando el framework Django.
- **Nuevas Métricas de Salud: Integración de más indicadores de salud y bienestar, como por ejemplo:**
 - Medición de nivel de estrés.
- **Mejoras en la Interfaz:** Mejorar la estética y la usabilidad de la interfaz gráfica.
- **Mejoras en la Funcionalidad:** Creación de Administrador, funciones como Agregar y Eliminar clientes.