Manual de Backend – SportPro

Elaborado por:

- Diego Armando Pérez Solano
- Juan Sebastián Corredor Sáenz
- Sergio Alejandro Vazquez Pineda

Presentado a:

Néstor German Bolívar Pulgarin

Asignatura:

Programación Orientada a Objetos

Universiada Nacional de Colombia

2025 – 2s

Bogotá, DC

2025

Introducción

El presente manual de backend busca presentar el funcionamiento, uso y descripción del proyecto final para la materia de “programación orientada a objetos”, el propósito de este manual es otorgar una explicación detalla del programa para el docente a cargo de la asignatura.

Además, se incluye informacion relacionada con la preparación y ejecución del aplicativo con el fin de que al terminar de leer el manual se tendrá un entendimiento al menos mínimo del código y como usarlo.

Tecnologías usadas

Para el aplicativo “SportPro” se han usado diferentes librerías, servicios y el lenguaje de programación Python.

LIBRERIAS:

- Types (Modulo estándar de Python que contiene definiciones de tipos especiales y utilidades internas no disponibles los módulos básicos de Python)
- Flask (Librería usada para conectar y trabajar Python con HTML)
- Firebase_admin (Librería usada para usar el servicio de Google para aplicaciones y servidores, Firebase)
- Dotenv (Usada para cargar y usar variables de entorno env (enviroment variables))
- Datetime (Librería usada para manejar fechas y horas de manera precisa)
- Re (Librería de Python usada para manejar las expresiones regulares, patrones de texto)
- Collections (Librería de Python que incluye estructuras de datos avanzadas)
- Os (Librería de Python que facilita y simplifica la navegación por archivos del sistema)
- Requests (Librería que permite realizar peticiones https fácilmente, utilizada para API’s)
- Dataclasses (Introduce el decorador @dataclass y varias herramientas para crear clases con el fin de almacenar datos)
- Typing (Modulo de Python que permite agregar anotados para darle al editor del código una idea de que dato se espera)

SERVICIOS:

- Firebase (Plataforma de Google que facilita el desarrollo de aplicaciones y servidores sin necesidad de crearlos desde cero)
- LM studio (Plataforma que permite el uso de algoritmos IA de manera local en un computador)

LENGUAJES:

- Python (Lenguaje principal del aplicativo, aquí se programó toda la lógica con la que funciona el producto)
- HTML (Lengua de etiquetado con el que se creó la estructura visual del programa)
- CSS (Lenguaje de estilo con el que se creó la parte visual del programa)

Estructura del proyecto

/SportPro > Carpeta principal

- .env > Variables del entorno (detalles importantes y protegidos para el aplicativo)

- Main.py > Código principal que ejecuta el programa

/templates > Contiene todos los HTML que el programa usa para la parte visual

/viewmodels

- rutinas_vm.py > Lógica de las rutinas, construcción para mostrarlas

/models

- firebase.py > Opciones de Firebase e inicialización de este

- rutina_base.py > Base de la rutina, que será remplazada por la construida

- rutinas_baloncesto.py > Rutinas de baloncesto para el programa

- rutinas_futbol.py > Rutinas de futbol para el programa

- rutinas_natacion.py > Rutinas de natación para el programa

- rutinas_tenis.py > Rutinas de tenis para el programa

- rutinas.py > Colección de todas las rutinas que puede revisar el programa

- usuarios.py > Base del usuario, se remplaza por el actual

/static

/css > Archivo css que guarda el estilo a usar

/img

- Imágenes_ejercicios.py > Relación visual-ejercicio

- Logo.png > Logo del aplicativo

/ejercicios > Todos los visuales de los ejercicios

Procesos internos

Una vez que inicia el programa al usuario se le dará un link a la página del programa, una vez le de click primero se le pedirá terminar una encuesta que va a medir su nivel como deportista, una vez realizada la encuesta su cuenta será creada y usted podrá ver las opciones del programa, ejercicios, su cuenta, alarmas, progreso, cronometro y un chat con IA para preguntarle cosas acerca de ejercicios y apoyo para diferentes deportes.

/Main.py

Dentro del main se comunican los HTML con los archivos Python con ayuda de la librería Flask

```python
@app.route("/")
def index():
    return redirect(url_for("encuesta"))


@app.route("/Encuesta", methods=["GET", "POST"])
def encuesta():
    if request.method == "POST":
        usuario = crear_usuario_desde_form(request.form)
        session["usuario_actual"] = usuario.nombre
        return redirect(url_for("rutina"))

    return render_template("Encuesta.html")


@app.route("/Rutina", methods=["GET", "POST"])
def rutina():
    nombre = session.get("usuario_actual")

    if not nombre:
        flash("Completa la encuesta primero.", "warning")
        return redirect(url_for("encuesta"))

    if request.method == "POST":
```

Por eso mismo todas las líneas pertenecientes a este archivo van dictando reglas de recibir y otorgar datos a la página donde se aloja el usuario, por eso mismo también se tienen que importar diferentes métodos del viewmodel y librerías en este archivo, para realizar esa

comunicación con los archivos HTML que al final es lo que Flask muestra debido a que este mismo es el que al final del día muestra lo que se le pida en forma de página web.

Antes de empezar cualquier proceso se importan las siguientes librerías:

- Flask
- Types
- Requests
- Dotenv

1. **Ruta /encuesta**

Lo que realiza la ruta encuesta es renderizar el archivo HTML de la encuesta, cuando renderiza esta manda y envía datos, ¿qué datos? Dentro de la encuesta se encuentran diferentes peguntas para medir el nivel deportivo del usuario así que cuando se termina esta, la ruta pide la informacion de las preguntas y las envía al método para crear un nuevo usuario dotándole de todas las características con las que se respondió.

```python
@app.route("/Encuesta", methods=["GET", "POST"])
def encuesta():
    if request.method == "POST":
        usuario = crear_usuario_desde_form(request.form)
        session["usuario_actual"] = usuario.nombre
        return redirect(url_for("rutina"))

    return render_template("Encuesta.html")
```

Si el método de petición es “POST” entonces con una función especial del archivo Firebase, se coleccionará toda la informacion que se haya dado en la encuesta y se transformará en informacion de usuario

Sesión[usuario] = usuario.nombre

Se recordada al usuario que este usando la aplicación para darle informacion y finalmente después de eso se ira a la url rutina, únicamente si se envía informacion

Si no se esta enviando entonces el Get obtendrá el HTML de la encuesta

2. **Ruta /rutina**

Dentro de rutina se le envía a la página los datos necesarios para generar las rutinas, una vez se hayan creado gracias a la encuesta los parámetros de la rutina se crea una personalizada para el usuario desde el viewmodel, cuando se tenga la rutina lista con un método POST se le pasan estos datos a la página HTML de rutinas y se renderizan los datos necesarios

```python
@app.route("/Rutina", methods=["GET", "POST"])
def rutina():
    nombre = session.get("usuario_actual")

    if not nombre:
        flash("Completa la encuesta primero.", "warning")
        return redirect(url_for("encuesta"))

    if request.method == "POST":
        deporte = request.form.get("deporte")
        tipo_sesion = request.form.get("tipo_sesion")
        titulo = request.form.get("titulo")
        duracion = request.form.get("duracion")

        peso_str = request.form.get("peso", "").strip()
        try:
            peso_actual = int(peso_str) if peso_str else None
        except:
            peso_actual = None

        rutina_obj = SimpleNamespace(
            deporte=deporte,
            tipo_sesion=tipo_sesion,
            titulo=titulo,
            duracion=duracion
        )

        guardar_sesion_rutina(nombre, rutina_obj, peso_actual)
        flash("Rutina registrada en tu historial de progreso.", "success")
        return redirect(url_for("historial"))

    data = obtener_rutinas_para_usuario(nombre)

    return render_template(
        "Rutina.html",
        rutinas=data["rutinas"],
```

Primero se buscará una sesión guardada, si esta no existe entonces se devolverá el usuario a la encuesta junto a un mensaje de advertencia

If Request.method == “POST”

Simboliza que todo debajo de esta condición será informacion enviada por el usuario, las cuatro líneas debajo de Request son valores que se pueden otorgar desde las rutinas, el deporte, el tipo de entrenamiento, titulo de este y finalmente la duración, después se guarda la rutina con los datos entregados y se redirige a el apartado de historial.

Si no esta en POST entonces el método esta en GET y le envía a la pagina un entrenamiento para el usuario ya precargado lista para su uso

3. Ruta /historial

Dentro de la opción historial se usan diferentes librerías para medir a base de las rutinas o ejercicios completados el tiempo utilizado o cantidad de rutinas completadas por el usuario, esta informacion se mide y se le pasa a el HTML del historial, además también se le pide el peso al usuario para agregarlo dentro del historial como una opción para medir progreso

```python
@app.route("/Historial")
def historial():
    nombre = session.get("usuario_actual")
    if not nombre:
        flash("Completa la encuesta primero para generar tu historial.", "warning")
        return redirect(url_for("encuesta"))

    sesiones = obtener_historial_usuario(nombre)

    fechas_peso = []
    pesos = []

    minutos_por_dia = defaultdict(int)
    rutinas_por_dia = defaultdict(int)

    for s in sesiones:
        fecha_str = s.get("fecha", "")
        if not fecha_str:
            continue

        try:
            fecha = datetime.fromisoformat(fecha_str)
            fecha_dia = fecha.strftime("%Y-%m-%d")
        except Exception:
            fecha_dia = fecha_str[:10]

        peso = s.get("peso")
        if peso is not None:
            fechas_peso.append(fecha_dia)
            pesos.append(peso)

        try:
            minutos_por_dia[fecha_dia] += int(s.get("minutos", 0))
        except:
            pass
```

```
        rutinas_por_dia[fecha_dia] += 1

    fechas_tiempo = sorted(minutos_por_dia.keys())
    minutos_diarios = [minutos_por_dia[f] for f in fechas_tiempo]

    fechas_rutinas = sorted(rutinas_por_dia.keys())
    rutinas_diarias = [rutinas_por_dia[f] for f in fechas_rutinas]

    return render_template(
        "Historial.html",
        historial=sesiones,
        fechas_peso=fechas_peso,
        pesos=pesos,
        fechas_tiempo=fechas_tiempo,
        minutos_diarios=minutos_diarios,
        fechas_rutinas=fechas_rutinas,
        rutinas_diarias=rutinas_diarias
    )
```

Primero se verifica que el usuario exista, si no existe entonces se manda un mensaje de advertencia para terminar la encuesta, si el usuario existe entonces se obtiene el historial del usuario y se empiezan a crear las listas para comparar los datos en la sección de historial

Por cada sesión se obtienen las fechas en las que se realizó, tiempos que tomo y cantidad de sesiones como tal, después de todo se preparan los datos para las graficas y definen variables para mandarle al HTML lo cual se hace al final renderizando el HTML del historial.

4. **Ruta /usuario**

Cuando se visita la sección usuario el programa primero identifica si existe un usuario que haya iniciado sesión

```
@app.route("/Usuario")
def usuario():
    nombre = session.get("usuario_actual")
    if not nombre:
        flash("Completa la encuesta primero para ver tu perfil.", "warning")
        return redirect(url_for("encuesta"))
```

O un usuario existente valido

```
data = obtener_usuario_desde_firebase(nombre)
if not data:
    flash("No se encontró información del usuario.", "warning")
    return redirect(url_for("encuesta"))
```

Si las cosas son así, entonces Flask indica la renderización del HMTL de usuario junto a un argumento el cual se define como la informacion que el usuario valido tenga en la base de datos Firebase

```
return render_template("Usuario.html", usuario=data)
```

5. **Ruta /alarmas**

El programa llama al HTML de alarmas

```
@app.route("/Alarmas")
def alarmas():
    return render_template("Alarmas.html")
```

6. **Ruta /Chat**

Cuando se acceda al /API/chat, primero se carga todo lo relacionado con el chat IA que habla con el usuario, cosas como su configuración o lugar de carga, por ejemplo la variable de LM_URL sirve para que si se tiene un modelo IA activado e instalado activar ese modelo para realizar la interacción con el usuario, dentro de las configuraciones el modelo también se define como un ayudante de entrenamiento y niega todo tema que este por fuera de su tema designado, el modelo igualmente define sus tokens y temperatura con la que funcionara

```python
@app.route("/api/chat", methods=["POST"])
def api_chat():
    data = request.get_json()
    user_msg = data.get("message", "")

    if not user_msg:
        return {"reply": "No recibí ningún mensaje."}

    LM_URL = "http://127.0.0.1:1234"

    payload = {
        "model": "qwen2.5-7b-instruct-1m",
        "messages": [
            {
                "role": "system",
                "content": """Actúa como un entrenador personal experto en ejercicio, rutinas, fuerza, movilidad, estiramiento
Responde con mensajes cortos, directos y solo con lo esencial.

Si la pregunta no es sobre ejercicio, deporte o entrenamiento, responde únicamente:
"Solo puedo ayudarte con temas de ejercicio y entrenamiento."

No uses emojis. Sé conciso y profesional."""
            },
            {"role": "user", "content": user_msg}
        ],
        "temperature": 0.4,
        "max_tokens": 150
    }

    try:
        response = requests.post(LM_URL, json=payload)
        res = response.json()

        reply = res["choices"][0]["message"]["content"]
        return {"reply": reply}
```

/MODELS:

La carpeta models cuenta con todo lo relación con modelos que usara el programa, tales como entrenamientos, usuario o informacion de Firebase

Archivos dentro de modelos:

FIREBASE.PY

Conexión con servidor y obtención de datos.

1. **Guardar sesión de la rutina**

Firebase se inicializa en la línea

```python
def guardar_sesion_rutina(nombre_usuario: str, rutina, peso_actual: int | None = None):
    """
    Guarda en /Historial/<usuario> un registro de rutina realizada,
    con fecha, deporte, tipo de sesión, título de rutina, minutos y peso (opcional).
    """
    initialize_firebase()
    ref = db.reference("/Historial").child(nombre_usuario)
```

También define su referencia como /historial/”Usuario actual”

Cuando la referencia esta creada entonces a la hora de que teóricamente se llame este método como se puede ver arriba se le otorgan unos datos, con esos datos se crea un diccionario que contendrá la informacion de la sesión dependiendo de los datos otorgados

```
data = {
    "fecha": datetime.now().isoformat(),
    "deporte": getattr(rutina, "deporte", ""),
    "tipo_sesion": getattr(rutina, "tipo_sesion", ""),
    "rutina": getattr(rutina, "titulo", ""),
    "minutos": minutos,
    "peso": peso_actual
}

ref.push(data)
return data
```

Y finalmente la informacion de la sesión será creada tanto para Firebase así dándole al usuario un historial de progreso, sino que también se retornada este diccionario para usarse

2. **Obtener historial de usuario**

Parecido al anterior, primero se le pasara al programa el usuario actual y se inicializara Firebase , cuando esto este listo se define la referencia a buscar de la exacta misma forma que ya se habia hecho antes y se obtiene la informacion de historial si existe, para usarse esta función debe de haber sido ya ejecutada la 1.

3. **Inicializar Firebase**

Inicializa Firebase con sus credenciales

```python
def initialize_firebase():
    global _firebase_initialized
    if _firebase_initialized:
        return

    cred_path = os.getenv("GOOGLE_APPLICATION_CREDENTIALS")
    db_url = os.getenv("DATABASE_URL")

    if not cred_path:
        raise RuntimeError("Falta GOOGLE_APPLICATION_CREDENTIALS en .env")
    if not db_url:
        raise RuntimeError("Falta DATABASE_URL en .env")

    if not firebase_admin._apps:
        cred = credentials.Certificate(cred_path)
        firebase_admin.initialize_app(cred, {"databaseURL": db_url})

    _firebase_initialized = True
```

4. **Guardar usuario en Firebase**

Primero se ejecuta la función otorgándole unos datos, estos datos es la informacion que otorgo el usuario para la encuesta, de esta forma Firebase se inicia y se crea dentro del

camino /Usuarios un usuario con los datos que se le pasaron a la función

```python
def guardar_usuario_en_firebase(usuario: Usuario):
    """
    Guarda/actualiza los datos del usuario en /Usuarios/<nombre>.
    Incluye los campos nuevos: frecuencia, duracion, calentamiento, discapacidad.
    """
    initialize_firebase()

    ref = db.reference("/Usuarios")

    data = {
        "nombre": usuario.nombre,
        "contrasena": usuario.contrasena,
        "genero": usuario.genero,
        "deporte": usuario.deporte,
        "edad": usuario.edad,
        "nivel": usuario.nivel,
        "complicacion": getattr(usuario, "complicacion", ""),
        "frecuencia": getattr(usuario, "frecuencia", ""),
        "duracion": getattr(usuario, "duracion", ""),
        "calentamiento": getattr(usuario, "calentamiento", ""),
        "discapacidad": getattr(usuario, "discapacidad", "")
    }

    ref.child(usuario.nombre).set(data)
    return data
```

5. **Obtener Usuario**

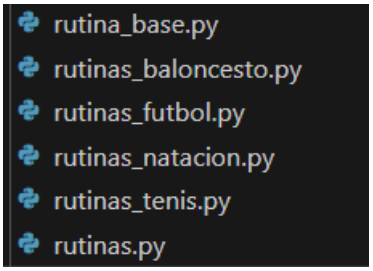
Funcionando junto al .4 con el nombre de la sesión actual se busca dentro de /Usuarios (misma referencia) y se devuelve la informacion del usuario dado

```python
def obtener_usuario_desde_firebase(nombre: str):
    """
    Devuelve el diccionario de datos del usuario almacenado en Firebase,
    o None si no existe.
    """
    initialize_firebase()
    ref = db.reference("/Usuarios")
    return ref.child(nombre).get()
```

Rutinas – todos los archivos de rutinas.py

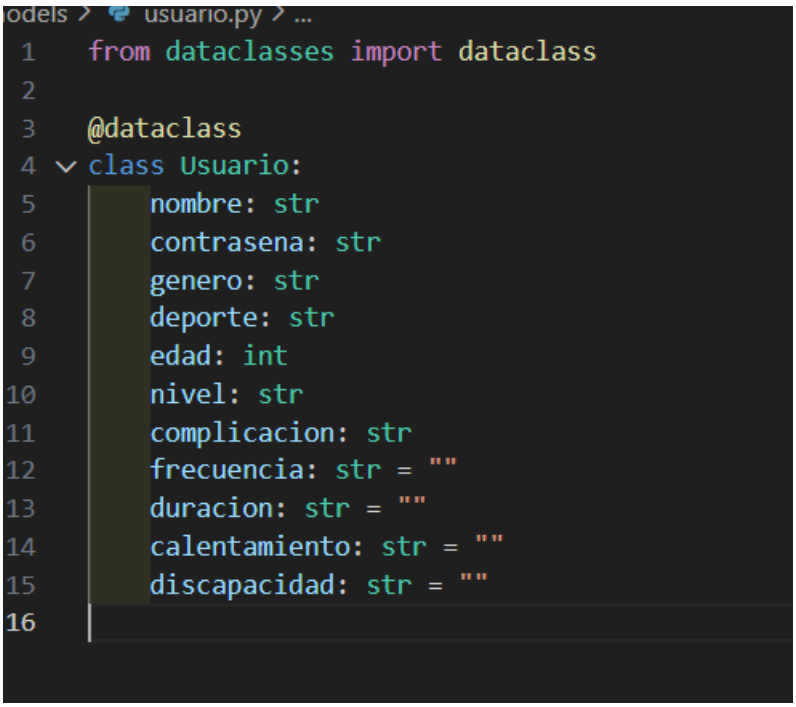
Todos los archivos de rutinas traen configuraciones para las rutinas del usuario, el archivo de rutinas base solo es una base de rutina la cual trae variables preparadas pero vacías

para llenar, de futbol a natación son simplemente diccionarios de datos que traen los ejercicios y finalmente el archivo de rutinas junta todas las posibles rutinas para usarlas después en caso de ser necesario.



usuario.py

Solamente un archivo de base para los usuarios, cuando se va a crear uno se crea una instancia de este archivo y se cambian los valores vacíos por los valores a usar



/Viewmodel

En la sección de viewmodel se encuentra toda la lógica del programa, la relación entre modelos y Flask, ósea trabajo de datos, como por ejemplo obtener datos de Flask y pasar a otorgárselos a Firebase o Usuario.py

```python
import re
from models.rutina_base import Rutina
from models.usuario import Usuario
from models.firebase import guardar_usuario_en_firebase, obtener_usuario_desde_firebase
from models.rutinas_futbol import RUTINAS_FUTBOL
from models.rutinas_baloncesto import RUTINAS_BALONCESTO
from models.rutinas_natacion import RUTINAS_NATACION
from models.rutinas_tenis import RUTINAS_TENIS
from static.img.imagenes_ejercicios import IMAGENES_EJERCICIOS, GIF_POR_DEFECTO
```

Como primer paso el archivo carga la librería re y todos los archivos que necesita para funcionar, la librería re es para manejar listas de una forma mas avanzada

```python
REGLAS_COMPLICACIONES = {
    "fatiga extrema": {
        # Quitamos max_duracion para no eliminar rutinas largas,
        # la reducción de duración la haremos en adaptar_rutina_intensidad.
        "evitar_palabras": ["salto", "explos", "sprint", "velocidad"]
    },
    "golpes de calor": {
        "max_duracion": 30,
        "evitar_palabras": ["sprint", "velocidad", "carrera", "cardio"]
    },
    "complicaciones respiratorias": {
        "evitar_palabras": ["caminadora", "cardio", "crol", "respiración"],
    },
    "complicaciones cardiovasculares": {
        "evitar_palabras": ["velocidad", "sprint", "carrera", "saltos"],
        "nivel_max": "Intermedio"
    }
}
```

Se definen en forma de diccionario reglas para las rutinas, así evitando dar ejercicios muy extremos para ciertas condiciones

```python
EJERCICIOS_ALTERNATIVOS = {
    "sentadilla": "sentadilla guiada",
    "zancada": "extensión de cuádriceps",
    "saltos": "elevaciones de gemelos",
    "sprints": "caminadora",
    "caminadora": "bicicleta",
    "crol": "flotación dorsal",
    "press militar": "elevaciones laterales",
    "peso muerto": "extensión de cuádriceps"
}
```

Se define un diccionario de ejercicios alternativos en case de tener que remplazar algunos

```python
def crear_usuario_desde_form(form):
    nombre = form["usuario"]
    contrasena = form["contrasena"]
    genero = form["genero"]
    deporte = form["deporte"]

    # Edad
    edad_str = form.get("edad", "").strip()
    try:
        edad = int(edad_str)
    except:
        edad = 0

    nivel = mapear_experiencia(form["experiencia"])

    # Complicación médica general
    complicacion = form.get("tipoComplicacion", "") if form["complicaciones"] == "si" else ""

    # NUEVAS VARIABLES
    frecuencia = form.get("frecuencia", "")
    duracion = form.get("duracion", "")
    calentamiento = form.get("calentamiento", "")
    discapacidad = form.get("tipoDiscapacidad", "")

    usuario = Usuario(
        nombre=nombre,
        contrasena=contrasena,
        genero=genero,
        deporte=deporte,
        edad=edad,
        nivel=nivel,
        complicacion=complicacion,
        frecuencia=frecuencia,
        duracion=duracion,
        calentamiento=calentamiento,
        discapacidad=discapacidad
```

La función crear usuario por el formulario se llama desde el main y se le otorgan los datos del formulario, aquí como se ve, nombre, contraseña, genero, deporte, etc.… se dan con el form.get el cual es un método disponible a la hora de obtener datos mediante un POST, cada dato es manejado de la forma necesaria para introducirlos al usuario y al final se crea un objeto de la clase usuario que contendrá todos los datos listos para guardar

```python
def asignar_gif_a_pasos(rutina: Rutina):
    nuevos_pasos = []

    for paso in rutina.pasos:
        if isinstance(paso, dict):
            nombre = paso.get("nombre", "").strip()
            detalle = paso.get("detalle", "").strip()
        else:
            # Separar nombre y detalle si es un string plano tipo "Ejercicio: 3x12"
            if ":" in paso:
                nombre, detalle = map(str.strip, paso.split(":", 1))
            else:
                nombre = paso.strip()
                detalle = ""

        clave_encontrada = None
        paso_lower = nombre.lower()

        for clave in IMAGENES_EJERCICIOS.keys():
            if clave in paso_lower:
                clave_encontrada = clave
                break

        gif = IMAGENES_EJERCICIOS.get(clave_encontrada, GIF_POR_DEFECTO)

        nuevos_pasos.append({
            "nombre": nombre,
            "detalle": detalle,
            "gif": gif
        })

    rutina.pasos = nuevos_pasos
    return rutina
```

Con una lista que se importo llamada IMÁGENES_EJERCICIOS se hace el cálculo del ejercicio a usar por el usuario y se le define su respectiva pareja de imagen para después añadir a los pasos que se mandaran al HTML de rutinas

```python
def reemplazar_ejercicios_peligrosos(rutina: Rutina):
    nuevos_pasos = []

    for paso in rutina.pasos:
        nombre_original = paso["nombre"].lower()
        reemplazo_final = paso["nombre"]

        for clave, reemplazo in EJERCICIOS_ALTERNATIVOS.items():
            if clave in nombre_original:
                reemplazo_final = reemplazo.capitalize()
                break

        nuevos_pasos.append({
            "nombre": reemplazo_final,
            "detalle": paso["detalle"],
            "gif": paso.get("gif", "")
        })

    rutina.pasos = nuevos_pasos
    return rutina
```

En caso de tener ejercicios peligrosos esta función remplaza los ejercicios uniéndolos a la lista nuevos pasos, por cada paso en rutina pasos se busca el nombre original si es necesario cambiar y se le otorga una versión modificada

```python
def adaptar_rutina_intensidad(rutina: Rutina, complicacion: str):
    c = complicacion.lower()
    nueva = rutina  # seguimos trabajando sobre el mismo objeto

    # adaptar duración
    try:
        minutos = int(nueva.duracion.split()[0])
    except:
        minutos = 30

    if "fatiga extrema" in c:
        minutos = max(15, minutos - 15)

        if "(versión ligera)" not in nueva.titulo:
            nueva.titulo += " (versión ligera)"

        texto_extra = " Esta versión reduce la carga para manejar la fatiga."
        if texto_extra.strip() not in nueva.descripcion:
            nueva.descripcion += texto_extra

    elif "golpes de calor" in c:
        minutos = max(15, minutos - 10)

        texto_extra = " Realiza esta rutina en horarios frescos y con buena hidratación."
        if texto_extra.strip() not in nueva.descripcion:
            nueva.descripcion += texto_extra
```

En caso de tener discapacidad, la intensidad de la rutina se modifica cambiando la variable minutos de su máximo normal a uno mas corto y calmado

```python
def adaptar_pasos_intensidad(rutina: Rutina, complicacion: str):
    """
    Reduce automáticamente la carga de los pasos (repeticiones, tiempo, distancia)
    cuando hay fatiga extrema o golpes de calor.
    """
    c = complicacion.lower()
    if "fatiga extrema" not in c and "golpes de calor" not in c:
        return rutina

    nuevos_pasos = []

    for paso in rutina.pasos:
        detalle = paso.get("detalle", "")
        if not detalle:
            nuevos_pasos.append(paso)
            continue

        # Función interna para reducir números
        def reducir(match):
            n = int(match.group())
            # No tocar números muy pequeños (1,2,3,4)
            if n <= 4:
                return match.group()
            # Reducir a un ~60% de la carga original
            nuevo = max(1, int(round(n * 0.6)))
            return str(nuevo)

        detalle_nuevo = re.sub(r"\d+", reducir, detalle)

        paso_mod = paso.copy()
        paso_mod["detalle"] = detalle_nuevo
        nuevos_pasos.append(paso_mod)

    rutina.pasos = nuevos_pasos
    return rutina
```

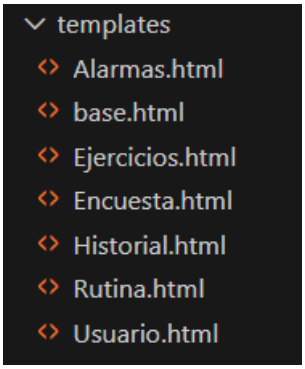
Si es necesario realizar cambios en pasos por discapacidad entonces se obtendrán los datos de los pasos en una rutina y serán modificados por versiones menos complicadas o más adaptadas

```python
def obtener_rutinas_para_usuario(nombre_usuario: str):
    data = obtener_usuario_desde_firebase(nombre_usuario)
    if not data:
        return {"deporte": "", "nivel": "", "complicacion": "", "rutinas": []}

    deporte = data["deporte"]
    nivel = data["nivel"]
    complicacion = data.get("complicacion", "")
    frecuencia = data.get("frecuencia", "")
    duracion_pref = data.get("duracion", "")
    calentamiento = data.get("calentamiento", "")
    discapacidad = data.get("discapacidad", "")

    user_flags = obtener_flags(complicacion, discapacidad)

    # 1. Rutinas base por deporte y nivel + flags (discapacidad, etc.)
    rutinas_usuario = [
        r for r in TODAS_LAS_RUTINAS
        if r.deporte == deporte
        and r.nivel == nivel
        and all(f not in r.evitar_flags for f in user_flags)
    ]

    # 2. REGLAS SEGÚN COMPLICACIÓN
    reglas = REGLAS_COMPLICACIONES.get(complicacion.lower(), {})

    # Evitar tipos de sesión (si alguna complicación lo define)
    if "evitar_tipos" in reglas:
        rutinas_usuario = [
            r for r in rutinas_usuario
            if r.tipo_sesion.lower() not in reglas["evitar_tipos"]
        ]
```

La última función del programa se llama obtener rutinas y se encarga de agrupar todo lo hecho en el archivo, casos especiales o no, al final define la versión final de las listas que se enviaran al HTML, las reglas, las rutinas y los pasos, lentamente revisa cada dato y dependiendo de que cosas se tomaron en cuenta o no y lo agrega.

/Templates

La sección templates guarda todos los HTML, cada uno de ellos son solo cosas visuales combinadas con POST y GET que se comunican con Python mediante la herramienta Flask



Instalación y configuración

REQUISITOS:

Si usted quiere instalar y usar el programa debe de contar con

- Visual studio con Python 3.0
- LM studio
- Pip funcional para Python

INSTALACION:

El programa usa librerías externas por lo que usted tiene que descargarlas usando pip, por eso se le pide que ya tenga instalado el gestor de paquetes para Python pip.

El método correcto para instalar una librería con pip es:

- Pip install “nombre_paquete”

Los paquetes que usted requiere instalar son:

- Flask
- Firebase_admin
- python-dotenv
- requests
- Dataclasses (SOLO SI TIENE UN PYTHON POR DEBAJO DE LA VERSION 3.7)

CONFIGURACION:

Una vez tenga todas las librerías instalas lo último que tendrá que hacer para ejecutar correctamente el programa será preparar el servidor Firebase y la IA en LM studio.

- Firebase:

Cuando descargue el programa y acceda a la carpeta de este mismo usted será capaz de ver un archivo .env, para configurar Firebase usted tendrá que:

1. Abrir el .env con visual studio
2. Extraer el archivo rar de la base de datos
3. Abrir la carpeta que se extrajo
4. Copiar la dirección del archivo Basedatos.json

5. Pegar esa dirección en la variable GOOGLE_APPLICATION_CREDENTIALS

- LM studio:

Para activar la IA de LM studio primero tiene que instalar LM studio desde la siguiente página: LM Studio - Local AI on your computer, una vez el programa este instalado proceda a abrirlo en la configuración Power user, esta configuración desbloqueara una barra de herramientas a la izquierda, de clic a la opción developer y cargue una IA (asegúrese de que la IA no sea tan pesada para su computador) una vez cargada arriba a la izquierda debajo de la opción “cargar modelo” usted cambiara el status de apagado a corriendo (running) y finalmente copiara la ip donde está corriendo el modelo, una vez la tenga copiada abra en visual studio el archivo main.py, vaya a la opción LM_URL y copie aquí la ip donde el modelo está funcionando de la siguiente forma

```
# ENDPOINT CORRECTO PARA CHAT
LM_URL = "http://192.168.68.117:1234/v1/chat/completions"
```

NO TOQUE NADA QUE ESTE DESPUES DEL /V1, únicamente modifique el http:// con la ip que le entrega LM studio, también tiene que modificar la sección

```
    payload = {
        "model": "qwen2.5-7b-instruct-1m",  # usa el nombre EXACTO del modelo cargado
        "messages": [
            {
                "role": "system",
```

Con el nombre del modelo que su computador este usando, model: “Nombre de modelo”

Por último, entre al HTML de rutinas baje hasta la línea 325 donde encontrara la opción model y modifíquela por también el nombre del modelo usado

```
322                 "Content-Type": "application/json"
323             },
324             body: JSON.stringify({
325                 model: "qwen2.5-7b-instruct-1m",  // < Cambia esto por el modelo que uses
326                 messages: [
327                     { role: "user", content: texto }
328                 ]
329             })
```

Cuando todas estas configuraciones estén listas usted podrá correr directamente el archivo main e ir a la ip del aplicativo y empezar a usarla