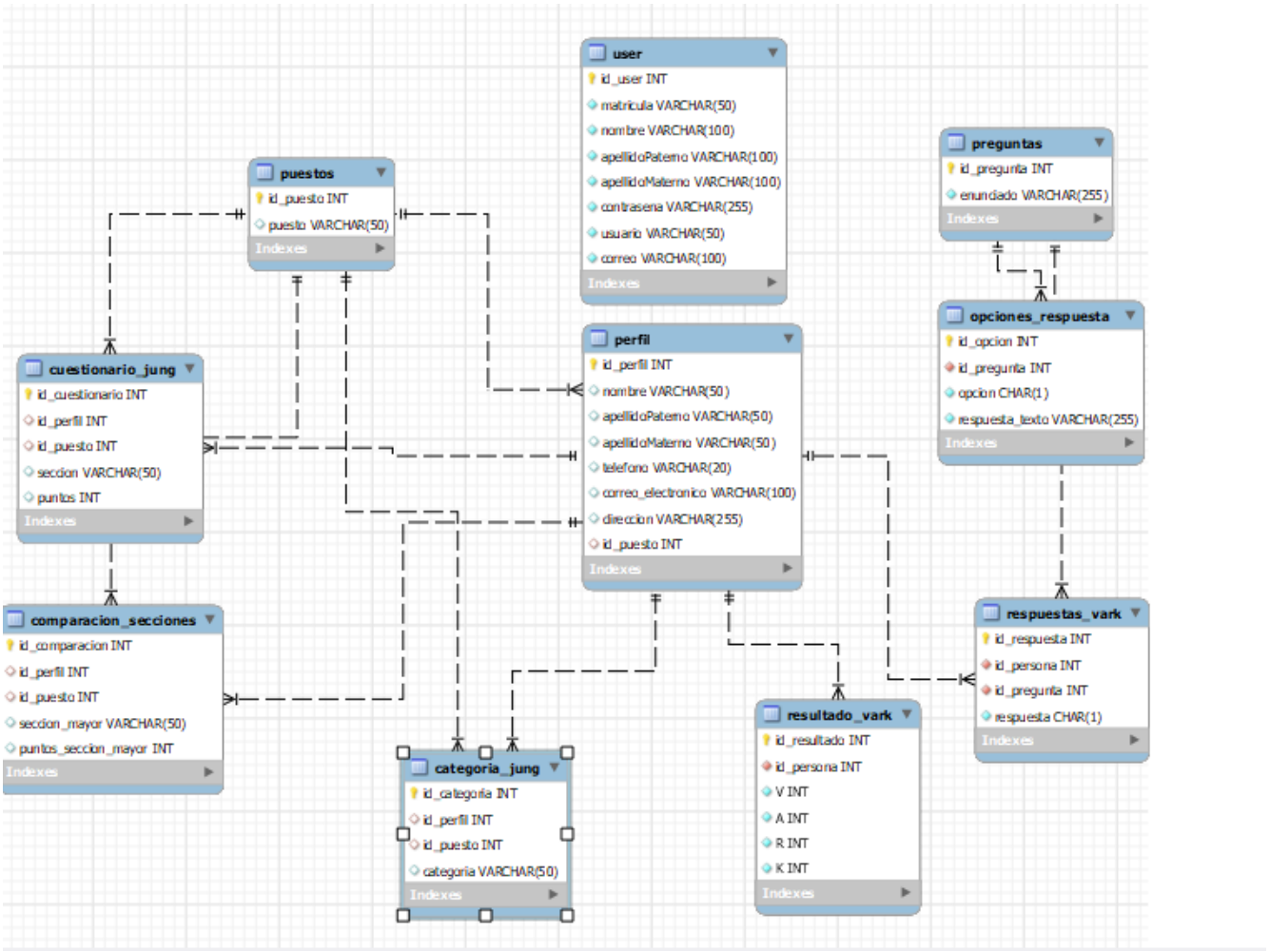
Proyecto ordinario
Desarrollo web

Matricula 17332507

Jose Adrian Ontiveros moran

Base de datos
por un principio tengo una base de datos que contiene



La base de datos se divide en dos en VARK y cuestionario jung,en el Vark tenemos tablas donde se guardan las preguntas y en otras sus inicisos y tabla donde se va guardadndo su resultado donde nos permite ir clasificando los resultados que nos ayudara después en los dashboard, y en el cuestionario jung nos va registrando cada sección de la base de datos de cada sección la mayor y ahí la vamos clasificando en categoría jung que es ahí donde podemos categorisar el perfil.

Empezando con el programa en Python o flask y con su pagina web empezamos declarando la clase con su conexión a la base de datos se ultilizo un patron de arquitectura singletoon,esto nos ayuda a no inicialisar a acada rato al conexión a ala base de datos y con esta clase solo vamos creando nuevas instancias y siempre usaremos la misma clase.

```
class Database:
    __instance = None

    @staticmethod
    def get_instance():
        if Database.__instance is None:
            Database()
        return Database.__instance

    def __init__(self):
        if Database.__instance is not None:
            raise Exception("lA CLASE ES SINGUELTOON!")
        else:
            Database.__instance = self
            # Asegúrate de que la cadena de conexión es correcta
            self.engine = create_engine('mysql+pymysql://root:123456@localhost/JyV',
echo=True)
            self.Base = declarative_base()
            self.Session = sessionmaker(bind=self.engine)
            self.session = self.Session()

    def get_session(self):
        return self.session
```

Librerias

Flask: Flask es el framework web que se está utilizando para construir la aplicación. Proporciona herramientas para el enrutamiento, manejo de solicitudes y respuestas, así como un conjunto de utilidades para facilitar el desarrollo web en Python.

render_template: Esta función se utiliza para renderizar plantillas HTML. Permite incrustar datos dinámicos en las plantillas antes de enviarlas al cliente.

request: La biblioteca request se utiliza para manejar las solicitudes HTTP entrantes. Proporciona acceso a datos como parámetros de consulta, datos de formulario y cookies.

redirect, url_for: Estas funciones se utilizan para redirigir a los usuarios a otras rutas dentro de la aplicación. redirect redirige a una URL específica, mientras que url_for genera la URL para una vista específica basada en su nombre.

session, flash: session se utiliza para almacenar datos de sesión del usuario entre solicitudes HTTP. flash se utiliza para enviar mensajes temporales al usuario, que generalmente se muestran en la siguiente solicitud.

send_file, send_from_directory: Estas funciones se utilizan para enviar archivos al cliente como respuesta a una solicitud HTTP. send_file envía un archivo específico, mientras que send_from_directory se utiliza para enviar archivos desde un directorio específico en el servidor.

jsonify: jsonify se utiliza para convertir objetos de Python en formato JSON, que es útil para devolver datos estructurados desde una vista Flask.

model.model: Esta es una importación personalizada de un módulo llamado model que parece contener definiciones de modelos de datos utilizando SQLAlchemy, que es un ORM (Mapeo

Objeto-Relacional) para Python.

Database, Perfil, Pregunta, ComparacionSecciones, RespuestaVark, ResultadoVark, Puesto, CuestionarioJung, CategoriaJung, User: Estos son nombres de clases que probablemente representan entidades en la base de datos del proyecto. Estas clases se utilizan para interactuar con la base de datos utilizando SQLAlchemy.

sqlalchemy: SQLAlchemy es una biblioteca popular de Python para trabajar con bases de datos relacionales. Proporciona una forma de mapear objetos de Python a tablas en una base de datos relacional y realizar consultas utilizando Python.

func: func es una función proporcionada por SQLAlchemy que permite utilizar funciones de base de datos en las consultas SQLAlchemy.

io, BytesIO: Estas bibliotecas se utilizan para trabajar con flujos de bytes en Python. BytesIO es un búfer de bytes en memoria que se puede utilizar para leer y escribir datos de bytes.

reportlab.pdfgen, reportlab.lib.pagesizes: Estas bibliotecas se utilizan para generar archivos PDF en Python. reportlab.pdfgen proporciona herramientas para crear documentos PDF programáticamente, mientras que reportlab.lib.pagesizes proporciona tamaños de página predefinidos.

flask_mail, Message: Estas bibliotecas se utilizan para enviar correos electrónicos desde una aplicación Flask. flask_mail proporciona una interfaz para enviar correos electrónicos, y Message se utiliza para construir y enviar mensajes de correo electrónico.

```
from flask import Flask, render_template, request, redirect, url_for, session,
flash,send_file,jsonify,send_from_directory
from model.model import Database, Perfil, Pregunta, ComparacionSecciones,
RespuestaVark, ResultadoVark, Puesto,CuestionarioJung,CategoriaJung,User
from sqlalchemy import func
from sqlalchemy.orm.exc import NoResultFound
from io import BytesIO
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from flask_mail import Mail, Message
```

Después de ahí empezamos a mapear la base de datos utilizando un orm que nos ayuda a tener sentencias mas fáciles y tener mejor funcionanmiento yu mas orden con las querys oh sentencias para ir manipulando la base de datos.

```
# Define las clases ORM después de la definición de la clase Database
Base = Database.get_instance().Base

class Puesto(Base):
    __tablename__ = 'Puestos'
    id_puesto = Column(Integer, primary_key=True, autoincrement=True)
    puesto = Column(String(50))
    # Definir la relación con la clase Perfil
    perfiles = relationship("Perfil", back_populates="puesto")


class Perfil(Base):
    __tablename__ = 'Perfil'
    id_perfil = Column(Integer, primary_key=True, autoincrement=True)
    nombre = Column(String(50))
    apellidoPaterno = Column(String(50))
    apellidoMaterno = Column(String(50))
    telefono = Column(String(20))
    correo_electronico = Column(String(100))
    direccion = Column(String(255))
    id_puesto = Column(Integer, ForeignKey('Puestos.id_puesto'))
    puesto = relationship("Puesto", back_populates="perfiles")

class CuestionarioJung(Base):
```

```
    __tablename__ = 'Cuestionario_jung'
    id_cuestionario = Column(Integer, primary_key=True, autoincrement=True)
    id_perfil = Column(Integer, ForeignKey('Perfil.id_perfil'))
    id_puesto = Column(Integer, ForeignKey('Puestos.id_puesto'))
    seccion = Column(String(50))
    puntos = Column(Integer)

class CategoriaJung(Base):
    __tablename__ = 'Categoria_Jung'
    id_categoria = Column(Integer, primary_key=True, autoincrement=True)
    id_perfil = Column(Integer, ForeignKey('Perfil.id_perfil'))
    id_puesto = Column(Integer, ForeignKey('Puestos.id_puesto'))
    categoria = Column(String(50))

class ComparacionSecciones(Base):
    __tablename__ = 'Comparacion_secciones'
    id_comparacion = Column(Integer, primary_key=True, autoincrement=True)
    id_perfil = Column(Integer, ForeignKey('Perfil.id_perfil'))
    id_puesto = Column(Integer, ForeignKey('Puestos.id_puesto'))
    seccion_mayor = Column(String(50))
    puntos_seccion_mayor = Column(Integer)

class Pregunta(Base):
    __tablename__ = 'Preguntas'
    id_pregunta = Column(Integer, primary_key=True, autoincrement=True)
    enunciado = Column(String(255), nullable=False)
    opciones = relationship("OpcionRespuesta", back_populates="pregunta")


class OpcionRespuesta(Base):
    __tablename__ = 'Opciones_respuesta'
    id_opcion = Column(Integer, primary_key=True, autoincrement=True)
    id_pregunta = Column(Integer, ForeignKey('Preguntas.id_pregunta'))
    opcion = Column(CHAR(1), nullable=False)
    respuesta_texto = Column(String(255), nullable=False)
    pregunta = relationship("Pregunta", back_populates="opciones")


class RespuestaVark(Base):
    __tablename__ = 'Respuestas_vark'
    id_respuesta = Column(Integer, primary_key=True, autoincrement=True)
    id_persona = Column(Integer, ForeignKey('Perfil.id_perfil'))
    id_pregunta = Column(Integer, ForeignKey('Preguntas.id_pregunta'))
    respuesta = Column(CHAR(1), nullable=False)

class ResultadoVark(Base):
    __tablename__ = 'Resultado_vark'
    id_resultado = Column(Integer, primary_key=True, autoincrement=True)
    id_persona = Column(Integer, ForeignKey('Perfil.id_perfil'))
```

```
    V = Column(Integer, nullable=False)
    A = Column(Integer, nullable=False)
    R = Column(Integer, nullable=False)
    K = Column(Integer, nullable=False)

class User(Base):
    __tablename__ = 'user'
    id_user = Column(Integer, primary_key=True, autoincrement=True)
    matricula = Column(String(50), nullable=False)
    nombre = Column(String(100), nullable=False)
    apellidoPaterno = Column(String(100), nullable=False)
    apellidoMaterno = Column(String(100), nullable=False)
    contrasena = Column(String(255), nullable=False)
    usuario = Column(String(50), nullable=False)
    correo = Column(String(100), nullable=False)

# Setup database connection and create tables
def setup_database():
    db_instance = Database.get_instance()
    db_instance.Base.metadata.create_all(db_instance.engine)
```

ahora pasamos a la primer sección del código en el que empezamos inicialisando los menu que nos ayudan manejar la pagina web y una vez que se registra se abre un submenú para ir registrando cada uno de los cuestionarios

```
app.route('/')
def menu():
    return render_template('menu.html')
#------------menu principal-----------------------------------------------------#
#------------Registrar perfil---------------------------------------------------#
@app.route('/')
def index():
    return redirect(url_for('registrar_perfil'))

# Ruta para registrar un perfil
@app.route('/registrar_perfil', methods=['GET', 'POST'])
def registrar_perfil():
    global id_persona

    if request.method == 'POST':
        nombre = request.form['nombre']
```

```
        apellido_paterno = request.form['apellido_paterno']
        apellido_materno = request.form['apellido_materno']
        telefono = request.form['telefono']
        correo = request.form['correo']
        id_puesto = int(request.form['id_puesto'])

        nuevo_perfil = Perfil(nombre=nombre,
                              apellidoPaterno=apellido_paterno,
                              apellidoMaterno=apellido_materno,
                              telefono=telefono,
                              correo_electronico=correo,
                              id_puesto=id_puesto)
        session_db.add(nuevo_perfil)
        session_db.commit()

        # Capturar el id del nuevo perfil
        id_persona = nuevo_perfil.id_perfil

        flash('Perfil registrado correctamente', 'success')
        return redirect(url_for('menu_cuestionario',
id_perfil=nuevo_perfil.id_perfil))
    else:
        puestos = session_db.query(Puesto).all()
        return render_template('registrar_perfil.html', puestos=puestos)


# Ruta para el menú de opciones del cuestionario
@app.route('/menu_cuestionario/<int:id_perfil>')
def menu_cuestionario(id_perfil):
    perfil = session_db.query(Perfil).filter_by(id_perfil=id_perfil).first()
    return render_template('menu_cuestionario.html', perfil=perfil)
```

ahora podemos explicar la parte vark en la que pide de la tabla preguntas sus preguntas con sus respectivos asertivos,para ir llenando y ya con las respuestas va conparando en una matris el numero y que letra es y así poder ir verificando que tipo es

```
# Tabla VARK
tabla = [
    {"#": 1, "V": "b", "A": "a", "R": "c", "K": "d"},
    {"#": 2, "V": "b", "A": "a", "R": "a", "K": "b"},
    {"#": 3, "V": "c", "A": "d", "R": "c", "K": "a"},
    {"#": 4, "V": "c", "A": "a", "R": "b", "K": "a"},
    {"#": 5, "V": "d", "A": "c", "R": "b", "K": "a"},
```

```
    {"#": 6, "V": "b", "A": "d", "R": "c", "K": "a"},
    {"#": 7, "V": "d", "A": "b", "R": "c", "K": "c"},
    {"#": 8, "V": "d", "A": "d", "R": "a", "K": "c"},
    {"#": 9, "V": "a", "A": "b", "R": "d", "K": "c"},
    {"#": 10, "V": "b", "A": "b", "R": "c", "K": "d"},
    {"#": 11, "V": "d", "A": "c", "R": "b", "K": "a"},
    {"#": 12, "V": "c", "A": "a", "R": "b", "K": "d"},
    {"#": 13, "V": "d", "A": "c", "R": "b", "K": "a"},
    {"#": 14, "V": "c", "A": "d", "R": "b", "K": "a"},
    {"#": 15, "V": "d", "A": "c", "R": "a", "K": "b"},
    {"#": 16, "V": "d", "A": "c", "R": "a", "K": "b"}
]

pregunta_index = 0
id_persona = None
#------------menu principal----------------------------------------------------#
@app.route('/')
def menu():
    return render_template('menu.html')
#------------menu principal----------------------------------------------------#
#------------Registrar perfil--------------------------------------------------#
@app.route('/')
def index():
    return redirect(url_for('registrar_perfil'))

# Ruta para registrar un perfil
@app.route('/registrar_perfil', methods=['GET', 'POST'])
def registrar_perfil():
    global id_persona

    if request.method == 'POST':
        nombre = request.form['nombre']
        apellido_paterno = request.form['apellido_paterno']
        apellido_materno = request.form['apellido_materno']
        telefono = request.form['telefono']
        correo = request.form['correo']
        id_puesto = int(request.form['id_puesto'])

        nuevo_perfil = Perfil(nombre=nombre,
                              apellidoPaterno=apellido_paterno,
                              apellidoMaterno=apellido_materno,
                              telefono=telefono,
                              correo_electronico=correo,
                              id_puesto=id_puesto)
        session_db.add(nuevo_perfil)
        session_db.commit()

        # Capturar el id del nuevo perfil
        id_persona = nuevo_perfil.id_perfil
```

```
        flash('Perfil registrado correctamente', 'success')
        return redirect(url_for('menu_cuestionario',
id_perfil=nuevo_perfil.id_perfil))
    else:
        puestos = session_db.query(Puesto).all()
        return render_template('registrar_perfil.html', puestos=puestos)


# Ruta para el menú de opciones del cuestionario
@app.route('/menu_cuestionario/<int:id_perfil>')
def menu_cuestionario(id_perfil):
    perfil = session_db.query(Perfil).filter_by(id_perfil=id_perfil).first()
    return render_template('menu_cuestionario.html', perfil=perfil)
#------------------VARK-----------------------------------------------------------
#
@app.route('/mostrar_siguiente_pregunta')
def mostrar_siguiente_pregunta():
    global pregunta_index

    if pregunta_index < len(preguntas):
        pregunta = preguntas[pregunta_index]
        opciones_pregunta = pregunta.opciones  # Cargar las opciones de respuesta
junto con la pregunta asociada
        pregunta_index += 1
        return render_template('pregunta.html', pregunta=pregunta,
opciones=opciones_pregunta)
    else:
        return redirect(url_for('calcular_perfil'))

@app.route('/guardar_respuesta', methods=['POST'])
def guardar_respuesta():
    global pregunta_index, id_persona
    respuesta = request.form['opcion']
    guardar_respuesta_db(preguntas[pregunta_index - 1].id_pregunta, respuesta,
id_persona)
    return redirect(url_for('mostrar_siguiente_pregunta'))

def guardar_respuesta_db(id_pregunta, respuesta, id_persona):
    try:
        respuesta = RespuestaVark(id_persona=id_persona, id_pregunta=id_pregunta,
respuesta=respuesta)
        session_db.add(respuesta)
        session_db.commit()
    except Exception as e:
        flash(f"No se pudo guardar la respuesta: {e}", 'error')

@app.route('/calcular_perfil')
def calcular_perfil():
```

```
    global id_persona

    respuestas =
session_db.query(RespuestaVark).filter_by(id_persona=id_persona).all()

    perfil = {"V": 0, "A": 0, "R": 0, "K": 0}
    for respuesta in respuestas:
        modo = obtener_modo_vark(respuesta.id_pregunta, respuesta.respuesta)
        if modo:
            perfil[modo] += 1

    # Insertar el perfil VARK en la tabla Resultado_VARK
    try:
        resultado_vark = ResultadoVark(id_persona=id_persona, V=perfil['V'],
A=perfil['A'], R=perfil['R'], K=perfil['K'])
        session_db.add(resultado_vark)
        session_db.commit()
        flash("Perfil VARK calculado y guardado correctamente.", 'success')
        return redirect(url_for('mostrar_resultado'))
    except Exception as e:
        flash(f"No se pudo guardar el perfil VARK en la base de datos: {e}", 'error')
        return redirect(url_for('index'))

def obtener_modo_vark(id_pregunta, respuesta):
    for fila in tabla:
        if fila["#"] == id_pregunta:
            if respuesta == fila["V"]:
                return "V"
            elif respuesta == fila["A"]:
                return "A"
            elif respuesta == fila["R"]:
                return "R"
            elif respuesta == fila["K"]:
                return "K"
    return None

@app.route('/mostrar_resultado')
def mostrar_resultado():
    global id_persona

    try:
        perfil_vark =
session_db.query(ResultadoVark).filter_by(id_persona=id_persona).first()
        return render_template('resultado.html', perfil_vark=perfil_vark)
    except Exception as e:
        flash(f"No se pudo obtener el perfil VARK: {e}", 'error')
        return redirect(url_for('index'))
#---------------FIN DE VARK-----------------------------------------------------
---------------#
```

Y en la parte del cuestionario jung, solo va comparando cada sección cual es mayor y luego va y selecciona la mayor y de ahí vamos comparando de acuerdo a sus combinaciones de sección que tipo de personalidad va mas aorde a el.

```
def obtener_perfil(id_perfil=None):
    if id_perfil is not None:
        return session_db.query(Perfil).filter_by(id_perfil=id_perfil).first()

    max_id_perfil = session_db.query(func.max(Perfil.id_perfil)).scalar()
    if max_id_perfil:
        return session_db.query(Perfil).filter_by(id_perfil=max_id_perfil).first()
    return None

def obtener_puesto(id_puesto=None):
    if id_puesto is not None:
        return session_db.query(Puesto).filter_by(id_puesto=id_puesto).first()

    max_id_puesto = session_db.query(func.max(Puesto.id_puesto)).scalar()
    if max_id_puesto:
        return session_db.query(Puesto).filter_by(id_puesto=max_id_puesto).first()
#---------------Inicio de Jung-------------------------------------------------------
---------------#
@app.route('/Seccion1_2')
def seccion_1_2():
    session = Database.get_instance().get_session()
    perfil = session.query(Perfil).order_by(Perfil.id_perfil.desc()).first()
    puesto = session.query(Puesto).order_by(Puesto.id_puesto.desc()).first()
    return render_template('seccion1_2.html', perfil=perfil, puesto=puesto)

@app.route('/sumar_secciones', methods=['POST'])
def sumar_secciones():
    session = Database.get_instance().get_session()
    seccion1_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(1,
10)])
    seccion2_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(10,
19)])

    id_perfil = request.form.get('id_perfil')
    id_puesto = request.form.get('id_puesto')

    if id_perfil and id_puesto:
```

```
        try:
            id_perfil = int(id_perfil)
            id_puesto = int(id_puesto)
            guardar_cuestionario_jung(seccion1_suma, seccion2_suma, id_perfil,
id_puesto, session)
            return redirect(url_for('seccion_3_4'))
        except ValueError:
            flash("ID de perfil o puesto no son valores enteros.", 'danger')
    else:
        flash("No se proporcionaron ID de perfil o puesto.", 'danger')

    return redirect(url_for('seccion_1_2'))

@app.route('/Seccion3_4')
def seccion_3_4():
    session = Database.get_instance().get_session()
    perfil = session.query(Perfil).order_by(Perfil.id_perfil.desc()).first()
    puesto = session.query(Puesto).order_by(Puesto.id_puesto.desc()).first()
    return render_template('seccion3_4.html', perfil=perfil, puesto=puesto)

@app.route('/sumar_secciones3_4', methods=['POST'])
def sumar_secciones3_4():
    session = Database.get_instance().get_session()
    seccion3_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(1,
10)])
    seccion4_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(10,
19)])

    id_perfil = request.form.get('id_perfil')
    id_puesto = request.form.get('id_puesto')

    if id_perfil and id_puesto:
        try:
            id_perfil = int(id_perfil)
            id_puesto = int(id_puesto)
            guardar_cuestionario_seccion3_4(seccion3_suma, seccion4_suma, id_perfil,
id_puesto, session)
            return redirect(url_for('seccion_5_6'))
        except ValueError:
            flash("ID de perfil o puesto no son valores enteros.", 'danger')
    else:
        flash("No se proporcionaron ID de perfil o puesto.", 'danger')

    return redirect(url_for('seccion_3_4'))

@app.route('/Seccion5_6')
def seccion_5_6():
    session = Database.get_instance().get_session()
    perfil = session.query(Perfil).order_by(Perfil.id_perfil.desc()).first()
```

```
    puesto = session.query(Puesto).order_by(Puesto.id_puesto.desc()).first()
    return render_template('seccion5_6.html', perfil=perfil, puesto=puesto)

@app.route('/sumar_secciones5_6', methods=['POST'])
def sumar_secciones5_6():
    session = Database.get_instance().get_session()
    seccion5_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(37,
46)])
    seccion6_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(46,
55)])

    id_perfil = request.form.get('id_perfil')
    id_puesto = request.form.get('id_puesto')

    if id_perfil and id_puesto:
        try:
            id_perfil = int(id_perfil)
            id_puesto = int(id_puesto)
            guardar_cuestionario_seccion5_6(seccion5_suma, seccion6_suma, id_perfil,
id_puesto, session)
            return redirect(url_for('seccion_7_8'))
        except ValueError:
            flash("ID de perfil o puesto no son valores enteros.", 'danger')
    else:
        flash("No se proporcionaron ID de perfil o puesto.", 'danger')

    return redirect(url_for('seccion_5_6'))

@app.route('/Seccion7_8')
def seccion_7_8():
    session = Database.get_instance().get_session()
    perfil = session.query(Perfil).order_by(Perfil.id_perfil.desc()).first()
    puesto = session.query(Puesto).order_by(Puesto.id_puesto.desc()).first()
    return render_template('seccion7_8.html', perfil=perfil, puesto=puesto)

@app.route('/sumar_secciones7_8', methods=['POST'])
def sumar_secciones7_8():
    session = Database.get_instance().get_session()
    seccion7_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(55,
64)])
    seccion8_suma = sum([int(request.form.get(f'pregunta{i}', 0)) for i in range(64,
73)])

    id_perfil = request.form.get('id_perfil')
    id_puesto = request.form.get('id_puesto')

    if id_perfil and id_puesto:
        try:
            id_perfil = int(id_perfil)
```

```
            id_puesto = int(id_puesto)
            guardar_cuestionario_seccion7_8(seccion7_suma, seccion8_suma, id_perfil,
id_puesto, session)
            asignar_categoria_perfil_actual(id_perfil, id_puesto, session)
            return redirect(url_for('resultados'))
        except ValueError:
            flash("ID de perfil o puesto no son valores enteros.", 'danger')
    else:
        flash("No se proporcionaron ID de perfil o puesto.", 'danger')

    return redirect(url_for('seccion_7_8'))

@app.route('/resultados')
def resultados():
    session = Database.get_instance().get_session()
    perfil = session.query(Perfil).order_by(Perfil.id_perfil.desc()).first()
    puesto = session.query(Puesto).order_by(Puesto.id_puesto.desc()).first()
    cuestionarios =
session.query(CuestionarioJung).filter_by(id_perfil=perfil.id_perfil).all()
    categoria_perfil =
session.query(CategoriaJung).filter_by(id_perfil=perfil.id_perfil).first()
    return render_template('resultados.html', perfil=perfil, puesto=puesto,
cuestionarios=cuestionarios, categoria=categoria_perfil)

def guardar_cuestionario_jung(seccion1_suma, seccion2_suma, id_perfil, id_puesto,
session):
    if seccion1_suma > seccion2_suma:
        seccion = "Sección 1"
        mayor_puntuacion = seccion1_suma
    else:
        seccion = "Sección 2"
        mayor_puntuacion = seccion2_suma

    cuestionario = CuestionarioJung(id_perfil=id_perfil, id_puesto=id_puesto,
seccion=seccion, puntos=mayor_puntuacion)
    session.add(cuestionario)
    session.commit()

def guardar_cuestionario_seccion3_4(seccion3_suma, seccion4_suma, id_perfil,
id_puesto, session):
    if seccion3_suma > seccion4_suma:
        seccion = "Sección 3"
        mayor_puntuacion = seccion3_suma
    else:
        seccion = "Sección 4"
        mayor_puntuacion = seccion4_suma

    cuestionario = CuestionarioJung(id_perfil=id_perfil, id_puesto=id_puesto,
seccion=seccion, puntos=mayor_puntuacion)
```

```
    session.add(cuestionario)
    session.commit()

def guardar_cuestionario_seccion5_6(seccion5_suma, seccion6_suma, id_perfil,
id_puesto, session):
    if seccion5_suma > seccion6_suma:
        seccion = "Sección 5"
        mayor_puntuacion = seccion5_suma
    else:
        seccion = "Sección 6"
        mayor_puntuacion = seccion6_suma

    cuestionario = CuestionarioJung(id_perfil=id_perfil, id_puesto=id_puesto,
seccion=seccion, puntos=mayor_puntuacion)
    session.add(cuestionario)
    session.commit()

def guardar_cuestionario_seccion7_8(seccion7_suma, seccion8_suma, id_perfil,
id_puesto, session):
    if seccion7_suma > seccion8_suma:
        seccion = "Sección 7"
        mayor_puntuacion = seccion7_suma
    else:
        seccion = "Sección 8"
        mayor_puntuacion = seccion8_suma

    cuestionario = CuestionarioJung(id_perfil=id_perfil, id_puesto=id_puesto,
seccion=seccion, puntos=mayor_puntuacion)
    session.add(cuestionario)
    session.commit()

def asignar_categoria_perfil_actual(id_perfil, id_puesto, session):
    secciones =
session.query(CuestionarioJung.seccion).filter_by(id_perfil=id_perfil).all()
    secciones_perfil = [seccion[0] for seccion in secciones]
    categoria = None

    if "Sección 3" in secciones_perfil and "Sección 6" in secciones_perfil:
        categoria = "Apoyo"
    elif "Sección 3" in secciones_perfil and "Sección 5" in secciones_perfil:
        categoria = "Técnico analítico"
    elif "Sección 4" in secciones_perfil and "Sección 5" in secciones_perfil:
        categoria = "Controlador"
    elif "Sección 4" in secciones_perfil and "Sección 6" in secciones_perfil:
        categoria = "Social"
    elif "Sección 5" in secciones_perfil and "Sección 4" in secciones_perfil:
        categoria = "Técnico analítico"
    elif "Sección 5" in secciones_perfil and "Sección 3" in secciones_perfil:
        categoria = "Técnico analítico"
```

```
    elif "Sección 5" in secciones_perfil and "Sección 4" in secciones_perfil:
        categoria = "Controlador"
    elif "Sección 6" in secciones_perfil and "Sección 3" in secciones_perfil:
        categoria = "Apoyo"
    elif "Sección 6" in secciones_perfil and "Sección 4" in secciones_perfil:
        categoria = "Social"

    if categoria:
        nuevo_categoria = CategoriaJung(id_perfil=id_perfil, id_puesto=id_puesto,
categoria=categoria)
        session.add(nuevo_categoria)
        session.commit()
```

Para mmandar un correo automático usamos flask mail donde previamente configuramos con la contraseña de aplicación y con esta función pudimos ir mandado un contacto form.

```
@app.route('/contact_form')
def contact_form():
    return render_template('contact_form.html')

# Ruta para procesar el formulario de contacto
@app.route('/contacto', methods=['POST'])
def contacto():
    nombre = request.form['nombre']
    correo = request.form['correo']
    asunto = request.form['asunto']
    mensaje = request.form['mensaje']

    # Enviar correo electrónico
    msg = Message(subject=asunto,
                  sender=app.config['MAIL_USERNAME'],
                  recipients=['aweb978@gmail.com'])  # Cambia 'destinatario@gmail.com'
por el correo del destinatario
    msg.body = f"Nombre: {nombre}\nCorreo: {correo}\nMensaje: {mensaje}"
    mail.send(msg)

    return f"¡Gracias {nombre}! Tu mensaje ha sido enviado."
```

Aquí es donde empieza el lado del admin con un login y un registro en el que le di de variante iniciar con la matricuala,y de ahí despliega una nuevo menú de admin.

```python
#-----------------LOGIN-------------------------------------------------------------------------------#
# Ruta para el registro de usuarios
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        matricula = request.form['matricula']
        nombre = request.form['nombre']
        apellidoPaterno = request.form['apellidoPaterno']
        apellidoMaterno = request.form['apellidoMaterno']
        contrasena = request.form['contrasena']
        correo = request.form['correo']
        usuario = request.form['usuario']

        # Verificar si el usuario ya existe en la base de datos
        existing_user = session_db.query(User).filter_by(matricula=matricula).first()
        if existing_user:
            return 'Ya existe un usuario con esta matrícula'

        # Crear un nuevo usuario
        new_user = User(matricula=matricula, nombre=nombre,
apellidoPaterno=apellidoPaterno,
                        apellidoMaterno=apellidoMaterno, contrasena=contrasena,
correo=correo, usuario=usuario)

        # Guardar el nuevo usuario en la base de datos
        session_db.add(new_user)
        session_db.commit()

        return 'Registro exitoso'

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        matricula = request.form['matricula']
        contrasena = request.form['contrasena']

        # Verificar las credenciales del usuario
        user = session_db.query(User).filter_by(matricula=matricula,
contrasena=contrasena).first()
        if user:
            session['matricula'] = user.matricula
            session['nombre'] = user.nombre  # Guarda el nombre del usuario en la
sesión
            return redirect('/bienvenido')  # Redirige al usuario al inicio exitoso
        else:
            return 'Credenciales incorrectas'
```

```
    return render_template('login.html')

@app.route('/bienvenido')
def bienvenido():
    if 'matricula' in session:
        nombre_usuario = session['nombre']  # Obtén el nombre del usuario de la sesión
        return render_template('bienvenido.html', nombre=nombre_usuario)
    else:
        return redirect('/login')  # Si no hay una sesión activa, redirige al usuario
al inicio de sesión
#-------------------------------nusqueda-----------------------------------------
---------#
```

Y en este neva parte del lado del admin es checar secciones especificas para ir haciendo dashboard con chsr js y así ir podiendo secmentar a los usuarios de la materia.

```
#-------------------------------nusqueda-----------------------------------------
---------#
@app.route('/index8')
def index8():
    # Obtener la sesión de la base de datos
    session = Database.get_instance().get_session()

    # Consultar los datos de la tabla CategoriaJung
    categorias = session.query(CategoriaJung.categoria).all()

    # Procesar los datos
    categoria_counts = {}
    for categoria in categorias:
        if categoria[0] in categoria_counts:
            categoria_counts[categoria[0]] += 1
        else:
            categoria_counts[categoria[0]] = 1

    # Preparar los datos para Chart.js
    labels = list(categoria_counts.keys())
    values = list(categoria_counts.values())

    return render_template('index8.html', labels=labels, values=values)
```

```python
@app.route('/categorias', methods=['GET'])
def obtener_categorias():
    db = Database.get_instance()
    session = db.get_session()
    categorias = session.query(CategoriaJung.categoria).distinct().all()
    return jsonify(categorias=[categoria[0] for categoria in categorias])

# Obtener las categorías por puestos
@app.route('/categorias_puestos', methods=['GET'])
def obtener_categorias_puestos():
    db = Database.get_instance()
    session = db.get_session()
    categorias_puestos = session.query(Puesto.puesto, CategoriaJung.categoria,
func.count(CategoriaJung.id_categoria)) \
        .join(CategoriaJung, CategoriaJung.id_puesto == Puesto.id_puesto) \
        .group_by(Puesto.puesto, CategoriaJung.categoria) \
        .all()

    categorias_por_puesto = {}
    for puesto, categoria, count in categorias_puestos:
        if puesto not in categorias_por_puesto:
            categorias_por_puesto[puesto] = []
        categorias_por_puesto[puesto].append({'categoria': categoria, 'count': count})

    return jsonify(categorias_por_puesto)

# Ruta para renderizar la página HTML
@app.route('/graficas')
def mostrar_pagina2():
    return render_template('graficas.html')
@app.route('/getChartData')
def get_chart_data():
    session = Database.get_instance().get_session()

    # Consulta para obtener los nombres de usuario y los puntajes VARK por puesto
    data = session.query(
        User.usuario,
        Puesto.puesto,
        func.sum(ResultadoVark.V).label('total_V'),
        func.sum(ResultadoVark.A).label('total_A'),
        func.sum(ResultadoVark.R).label('total_R'),
        func.sum(ResultadoVark.K).label('total_K')
    ).join(ResultadoVark, ResultadoVark.id_persona == User.id_user)\
    .join(Puesto, Puesto.id_puesto == User.id_puesto)\
    .group_by(User.usuario, Puesto.puesto).all()
```

```
    # Preparar los datos para enviar al cliente
    labels = [f"{entry[0]} ({entry[1]})" for entry in data]
    total_V = [entry[2] for entry in data]
    total_A = [entry[3] for entry in data]
    total_R = [entry[4] for entry in data]
    total_K = [entry[5] for entry in data]

    return jsonify(labels=labels, total_V=total_V, total_A=total_A, total_R=total_R,
total_K=total_K)

@app.route('/graficas2')
def graficas2():
    return render_template('graficas2.html')


# Obtener los resultados VARK por puesto
def get_vark_totals_by_puesto():
    db_instance = Database.get_instance()
    session = db_instance.get_session()

    results = session.query(
        Puesto.puesto,
        func.sum(ResultadoVark.V).label('total_V'),
        func.sum(ResultadoVark.A).label('total_A'),
        func.sum(ResultadoVark.R).label('total_R'),
        func.sum(ResultadoVark.K).label('total_K')
    ).join(Perfil, Perfil.id_perfil == ResultadoVark.id_persona
    ).join(Puesto, Puesto.id_puesto == Perfil.id_puesto
    ).group_by(Puesto.puesto).all()

    return results

@app.route('/graficas3')
def graficas3():
    results = get_vark_totals_by_puesto()
    return render_template('graficas3.html', results=results)

# Ruta para la búsqueda por correo electrónico
@app.route('/buscar_por_correo', methods=['GET', 'POST'])
def buscar_por_correo():
    if request.method == 'POST':
        correo = request.form['correo']
        session = Session()
        perfiles = session.query(Perfil).filter_by(correo_electronico=correo).all()
        return render_template('resultados2.html', perfiles=perfiles)
    return render_template('buscar_por_correo.html')

def eliminar_perfil(id_perfil):
```

```
    session = Session()

    # Eliminar instancias relacionadas en la tabla CuestionarioJung
    session.query(CuestionarioJung).filter_by(id_perfil=id_perfil).delete()

    # Eliminar instancias relacionadas en la tabla CategoriaJung
    session.query(CategoriaJung).filter_by(id_perfil=id_perfil).delete()

    # Eliminar instancias relacionadas en la tabla ComparacionSecciones
    session.query(ComparacionSecciones).filter_by(id_perfil=id_perfil).delete()

    # Eliminar instancias relacionadas en la tabla RespuestaVark
    session.query(RespuestaVark).filter_by(id_persona=id_perfil).delete()

    # Eliminar instancias relacionadas en la tabla ResultadoVark
    session.query(ResultadoVark).filter_by(id_persona=id_perfil).delete()

    # Eliminar el perfil en sí
    perfil = session.query(Perfil).get(id_perfil)
    if perfil:
        session.delete(perfil)
        session.commit()

    return redirect(url_for('buscar_por_correo'))


@app.route('/editar_perfil/<int:id_perfil>', methods=['GET', 'POST'])
def editar_perfil(id_perfil):
    session = Session()
    perfil = session.query(Perfil).get(id_perfil)
    if request.method == 'POST':
        perfil.nombre = request.form['nombre']
        perfil.apellidoPaterno = request.form['apellido_paterno']
        perfil.apellidoMaterno = request.form['apellido_materno']
        perfil.telefono = request.form['telefono']
        perfil.correo_electronico = request.form['correo_electronico']
        perfil.direccion = request.form['direccion']
        perfil.id_puesto = request.form['id_puesto']
        session.commit()
        return redirect(url_for('buscar_por_correo'))
    puestos = get_puestos(session)
    return render_template('editar_perfil.html', perfil=perfil, puestos=puestos)

def get_puestos(session):
    return session.query(Puesto).all()

def handle_edit_profile_request():
    session = Session()  # Si estás usando SQLAlchemy
    puestos = session.query(Puesto).all()
```

```
    # O si tienes una función get_puestos() definida en tu clase Database:
    # puestos = Database.get_instance().get_puestos()

    # Renderizar el template HTML pasando los puestos como contexto
    return render_template("editar_perfil.html", puestos=puestos)
##########################################################################
```

Y aquí es donde se termina el proyecto este proyecto tiene la capacidad de escalar a mas, ya que ayudaría a entender necesidades especificas en plantas, escuelas y luegares enm las que tengan la necesidad de aprender con un sistema de aprendizaje o conocer su personalidad especifica oh como tratar en varias personas
y por ultimo aquí es donde muestro el video

```
@app.route('/video')
def video_page():
    return render_template('video.html')

@app.route('/static/<path:filename>')
def static_files(filename):
    return send_from_directory('static', filename)
if __name__ == '__main__':
```