UNIVERSIDAD NACIONAL DE INGENIERÍA

“Facultad de Ingeniería Eléctrica y Electrónica”

Proyecto

EDU-Tech: Plataforma de diagnóstico y personalización para el aprendizaje eficaz

Curso

Programación orientada a objetos (BMA15)

Profesor

Tello Canchapoma, Yury Oscar

Fecha

1 de marzo de 2025

Integrantes

Capcha Gutierrez, Sofia Pilar 20230410K

Villarreal Lopez, Alexandra Aracelli 20230438B

Lima-Perú

2025

INDICE

INTRODUCCIÓN

El rendimiento académico es un factor importante en la educación, ya que nos permite medir el avance de los estudiantes y diseñar estrategias para mejorar su aprendizaje. Sin embargo, cada estudiante posee un estilo de aprendizaje diferente, lo que hace que ciertos métodos educativos sean más efectivos que otros dependiendo de la persona. A pesar de la existencia de múltiples recursos y herramientas digitales para el aprendizaje, no siempre están alineadas con las necesidades individuales de cada estudiante.

Este proyecto propone el desarrollo de una plataforma educativa que ayude a los estudiantes a identificar su estilo de aprendizaje mediante el cuestionario de Honey-Alonso. Basado en los resultados de esta evaluación, el sistema recomendará aplicaciones educativas personalizadas que se adapten a su forma de aprender. Además, también permite predecir el rendimiento académico de los estudiantes usando un modelo de Machine Learning basado en KMeans Clustering.

Es por ese motivo que en el proyecto buscamos integrar un enfoque más personalizado en la educación, brindando a los estudiantes herramientas adecuadas para mejorar su rendimiento y estrategias basadas en datos para optimizar su aprendizaje, además al poder predecir notas ellos pueden anticiparse a los resultados. Con la combinación de análisis de datos, algoritmos de agrupamiento y educación personalizada, la plataforma proporcionará una solución innovadora y automatizada para ayudar a los estudiantes a mejorar su desempeño académico de manera efectiva.

OBJETIVOS

Objetivo principal:

Desarrollar una plataforma educativa que ayude a los estudiantes a mejorar su rendimiento académico mediante el uso de apps educativas de acuerdo a su estilo de aprendizaje, el cual será identificado con la encuesta de Honey-Alonso, además predecir su próximo promedio les permitirá anticipar los resultados y mejorarlos.

Objetivos específicos:

- Permitir a los estudiantes registrarse, iniciar sesión y almacenar su información personal y académica, el cual se guardará en un data set para su análisis.
- Presentar el cuestionario de Honey-Alonso para identificar su estilo predominante, el cual consta de 80 preguntas categorizadas en diferentes estilos de aprendizaje: activo, reflexivo, teórico y pragmático.
- Brindar recomendaciones de las aplicaciones educativas según su estilo de aprendizaje predominante.
- Implementar un modelo de Machine Learning basado en KMeans Clustering para predecir el rendimiento académico de los estudiantes en función de su estilo de aprendizaje, las herramientas digitales que utilizan y sus calificaciones previas.

ANTECEDENTES

¿Por qué es importante analizar el rendimiento de los estudiantes?, según Prasad Bharti (2021), el análisis del trabajo de los estudiantes es una parte esencial de la enseñanza. Los profesores asignan, recopilan y examinan el trabajo de los estudiantes todo el tiempo para evaluar, revisar y mejorar la enseñanza.

Por eso realiza el proyecto Student Academic Performance Analysis, el cual analiza los conjuntos de datos que consta de las calificaciones obtenidas por los estudiantes en diversas materias. Para así analizar cuáles son los factores pueden influir en el desempeño de un estudiante, incluida la influencia de los antecedentes educativos de los padres, la preparación para los exámenes, la salud del estudiante, etc.

https://www.kaggle.com/code/bhartiprasad17/student-academic-performance-analysis/output

El tiempo de estudio invertido de un estudiante tiene mucha relación con su desempeño académico, es por ese motivo que Medhat M. (2024), hace una investigación con su modelo de regresión lineal llamado Student Scores Analysis para predecir calificaciones. Este a través de gráficos nos muestra cómo se centra el análisis de un conjunto de datos que relaciona las horas de estudio de los estudiantes con sus calificaciones académicas. Según sus gráficos se encontró una correlación positiva significativa entre las horas de estudio y las calificaciones obtenidas.

https://www.kaggle.com/code/markmedhat/student-scores-analysis/notebook

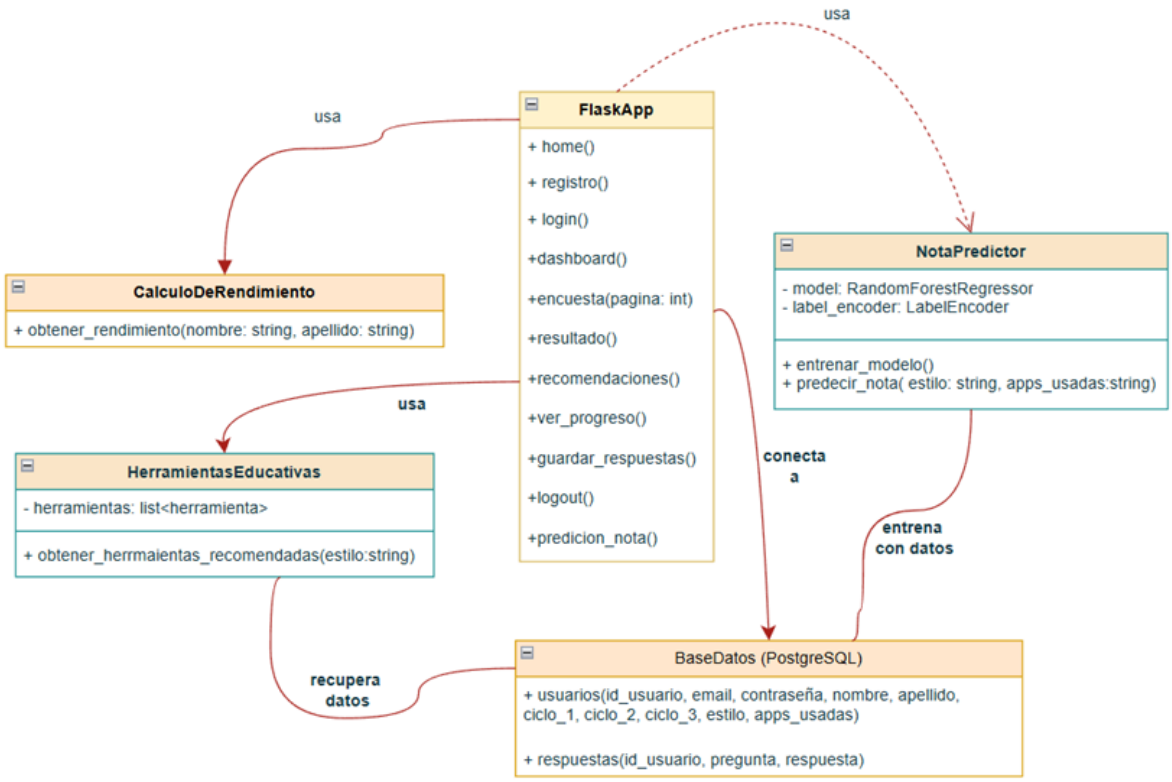
Anticipar el desempeño que podría tener un estudiante es importante para que así se pueda corregir y evitar malos resultados, por eso Martinez Bachmann J. (2018), implementa en su proyecto Predicting Grades for the School Year (Math Subject) un modelo algorítmico simple que predice la calificación de un estudiante individual al final del año, con el objetivo de comprender lo que los datos dicen a través de visualizaciones (plotly, matplotlib, seaborn).

Por eso su proyecto se centra en el análisis del rendimiento académico de los estudiantes, con el objetivo de identificar las variables clave que influyen en sus calificaciones. A través de un enfoque investigativo, se buscan respuestas a preguntas sobre factores como el impacto del consumo de alcohol en las relaciones familiares, así como la influencia del entorno de vivienda (rural o urbano) en el rendimiento de los estudiantes, considerando aspectos como las actividades extracurriculares, el tiempo libre y otros elementos contextuales.

https://www.kaggle.com/code/janiobachmann/predicting-grades-for-the-school-year

DIAGRAMA UML

Diagrama de clases



Este proyecto está basado en 3 paradigmas de la programación: programación orientada a objetos, programación funcional y programación imperativa.

Programación orientada a objetos (POO):

En el caso de programación orientada a objetos, podemos ver 3 clases principales, las cuales son: `FlaskApp`, `CalculoDeRendimiento, HerramientasEducativas, NotaPredictor y BaseDatos.`

1. FlaskApp

Es la clase principal que maneja la interfaz web y las rutas de la aplicación Flask. No está definida explícitamente como una clase en Python, pero podemos considerarla como un controlador que gestiona las funciones.

Métodos (Rutas en Flask):

- `home()` → Muestra la página de bienvenida.
- `registro()` → Permite registrar nuevos usuarios en la base de datos.
- `login()` → Permite el inicio de sesión.
- `dashboard()` → Página principal del usuario después de iniciar sesión.
- `encuesta(pagina: int)` → Muestra la encuesta sobre estilos de aprendizaje.

- `resultado()` → Calcula el estilo de aprendizaje del usuario según sus respuestas.
- `recomendaciones()` → Muestra las herramientas educativas recomendadas.
- `ver_progreso()` → Permite a los usuarios ver su avance en la encuesta.
- `guardar_respuestas()` → Guarda las respuestas del usuario en la base de datos.
- `logout()` → Cierra sesión del usuario.
- `prediccion_nota()` → Predice la nota del usuario basada en su estilo de aprendizaje.

2. Clase CalculoDeRendimiento

Esta clase se encarga de obtener el rendimiento académico del usuario, basándose en sus promedios de los últimos 3 ciclos.

Atributos:

No posee atributos, ya que contiene el método estático "`@staticmethod`". No usa `self`, debido a que no necesita acceder a atributos de la instancia ni de la clase. Se encarga únicamente de hacer una consulta SQL y calcular un promedio, lo cual es independiente del estado del objeto.

Métodos:

- `obtener_rendimiento (nombre, apellido):`
  a) Consulta los promedios del usuario en la base de datos
  b) Calcula el promedio y lo categoriza en 3 niveles: Reprobado (D-), Desaprobado (D), Aprobado (C), Bueno (B), Muy Bueno (A), Excelente (A+).
  c) Retorna a un diccionario con el promedio y tipo de rendimiento.

3. Clase HerramientasEducativas

Esta segunda clase tiene el listado de herramientas educativas recomendadas según el estilo de aprendizaje del usuario.

Atributos:

- `herramientas:` Es la lista de diccionarios donde cada entrada representa una aplicación educativa con su nombre y tipo de aprendizaje.

Métodos:

- `obtener_herramientas_recomendadas(estilo):` Se encarga de retornar una lista de herramientas según el estilo de aprendizaje del usuario.

4. Clase NotaPredictor

Esta clase implementa Machine Learning para predecir la nota promedio del estudiante, basándose en su estilo de aprendizaje y las aplicaciones educativas que usa.

Atributos:

- `model: RandomForestRegressor` → Modelo de Machine Learning para predicción de notas.
- `label_encoder: LabelEncoder` → Convierte el estilo de aprendizaje en valores numéricos.

Métodos:

- `entrenar_modelo()` → Entrena un modelo **RandomForestRegressor** con datos de los usuarios almacenados en la base de datos.
- `predecir_nota(estilo: str, apps_usadas: str) -> float`
  Usa el modelo de ML para predecir la nota de un estudiante en función de su estilo de aprendizaje (Activo, Reflexivo, Teórico, Pragmático) y la cantidad de herramientas educativas que utiliza.

5. BaseDatos (PostgreSQL)

Representa la base de datos del sistema, donde se almacenan los usuarios y sus respuestas.

Esta la tabla de usuarios almacena los datos de usuario como: id_usuario, email, contrasena, nombre, apellido, ciclo_1, ciclo_2, ciclo_3, estilo y apps_usadas.

En la tabla de respuestas se guarda las respuestas de la encuesta de cada usuario: id_usuario, pregunta y respuesta.

Programación funcional:

Se observa el uso de este en varios lugares, especialmente con funciones anónimas (lambda) y operaciones sobre estructuras de datos.

Ejemplos en el código:

- Uso de lambda en cálculos de estadística y normalización de datos.
- Uso de map, apply, y filter para transformar los datos en pandas.
- Eliminación de duplicados con `dict.fromkeys(respuestas)`, aplicando una transformación funcional a listas.

Programación imperativa:

La mayor parte del código sigue este enfoque, ya que describe **cómo** deben hacerse las cosas, con instrucciones paso a paso.

Ejemplos en el código:

- En las rutas de Flask (`@app.route`), se definen funciones que ejecutan instrucciones en orden secuencial.
- En la inicialización de la base de datos `(verificar_base_datos())`, se ejecutan sentencias SQL de manera explícita.

- Uso de estructuras de control como `if, for, y while` para gestionar el flujo del programa.

EXPLICACIÓN DEL CÓDIGO

1. Librerías

El proyecto " EDU-Tech: Plataforma de diagnóstico y personalización para el aprendizaje eficaz", utiliza algunas librerías para su análisis y procesamiento, las cuales son:

a) Os: se encarga de manejar las rutas de los archivos dentro del sistema operativo.

b) Psycopg2: Para conectar la base de datos PostgreSQL.

c) Flask: Define rutas y renderiza las páginas HTML.

d) Pandas y NumPy: Carga y manipula datos en el DataFrame desde bases de datos y archivos CSV.

e) Scikit-learn: Implementación de algoritmos de Machine Learning.

f) Re: Manejo de expresiones regulares.

g) Joblib: Para guardar y cargar modelos de Machine Learning.

```
from flask import Flask, render_template, request, redirect, session, url_for
import psycopg2
import os
import pandas as pd
import numpy as np
import re
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import joblib
from psycopg2 import pool
```

2. Configuración

En nuestro archivo app.py contiene la configuración de la aplicación. En esta parte definimos por seguridad una clave secreta.

```
app = Flask(__name__)
app.secret_key = "supersecreto"
```

También estableceremos las rutas de los archivos de la base de datos CSV llamada datos.csv y de nuestro SQL en database.db, ademas SimpleConnectionPool para manejar multiples conexiones:

```
BASE_DIR = os.path.abspath(os.path.dirname(__file__))
DATABASE_URL = "postgresql://sofia:12345@localhost/mi_base_de_datos"
db_pool = pool.SimpleConnectionPool(1, 10, dsn=DATABASE_URL)
DATASET_PATH = r"C:\Users\sofia\Downloads\codigoFlask\dataset\datos.csv"
MODEL_PATH = os.path.join(BASE_DIR, 'modelo_notas.pkl')
```

Luego se implementan métodos para manejar la base de datos.

```
def get_db_connection():
    return db_pool.getconn()

def release_db_connection(conn):
    if conn and hasattr(conn, 'closed') and not conn.closed:
        db_pool.putconn(conn)
```

3. Creación de tablas

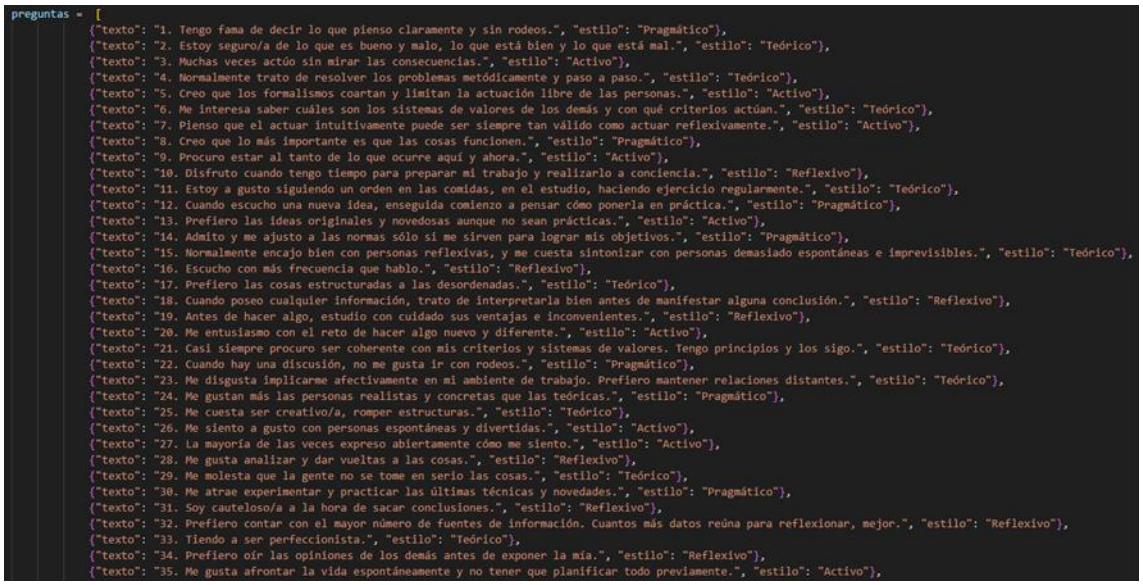
En este método, se asegura que las tablas usuarios y respuestas existan.

```
def verificar_base_datos():
    conn = get_db_connection()  # Usa la nueva conexión
    cursor = conn.cursor()

    # Crear tabla de usuarios (ya está en tu código)
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS usuarios (
        id_usuario SERIAL PRIMARY KEY,
        email TEXT UNIQUE NOT NULL,
        contrasena TEXT NOT NULL,
        nombre TEXT NOT NULL,
        apellido TEXT NOT NULL,
        ciclo_1 REAL,
        ciclo_2 REAL,
        ciclo_3 REAL,
        estilo TEXT,
        Apss_usadas TEXT
    )
    """)
```

```python
# CREAR LA TABLA RESPUESTAS (Asegurar que existe)
cursor.execute("""
CREATE TABLE IF NOT EXISTS respuestas (
    id_usuario INTEGER,
    pregunta TEXT NOT NULL,
    respuesta TEXT,
    PRIMARY KEY (id_usuario, pregunta),
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
)
""")

conn.commit()
release_db_connection(conn)
```

4. Preguntas de la encuesta

Tenemos la variable preguntas la cual contiene la lista de preguntas de la encuesta de estilos de aprendizaje. Cada pregunta está asociado a un tipo de estilo de aprendizaje como activo, reflexivo, teórico o pragmático.



```python
preguntas = [
    {"texto": "1. Tengo fama de decir lo que pienso claramente y sin rodeos.", "estilo": "Pragmático"},
    {"texto": "2. Estoy seguro/a de lo que es bueno y malo, lo que está bien y lo que está mal.", "estilo": "Teórico"},
    {"texto": "3. Muchas veces actúo sin mirar las consecuencias.", "estilo": "Activo"},
    {"texto": "4. Normalmente trato de resolver los problemas metódicamente y paso a paso.", "estilo": "Teórico"},
    {"texto": "5. Creo que los formalismos coartan y limitan la actuación libre de las personas.", "estilo": "Activo"},
    {"texto": "6. Me interesa saber cuáles son los sistemas de valores de los demás y con qué criterios actúan.", "estilo": "Teórico"},
    {"texto": "7. Pienso que el actuar intuitivamente puede ser siempre tan válido como actuar reflexivamente.", "estilo": "Activo"},
    {"texto": "8. Creo que lo más importante es que las cosas funcionen.", "estilo": "Pragmático"},
    {"texto": "9. Procuro estar al tanto de lo que ocurre aquí y ahora.", "estilo": "Activo"},
    {"texto": "10. Disfruto cuando tengo tiempo para preparar mi trabajo y realizarlo a conciencia.", "estilo": "Reflexivo"},
    {"texto": "11. Estoy a gusto siguiendo un orden en las comidas, en el estudio, haciendo ejercicio regularmente.", "estilo": "Teórico"},
    {"texto": "12. Cuando escucho una nueva idea, enseguida comienzo a pensar cómo ponerla en práctica.", "estilo": "Pragmático"},
    {"texto": "13. Prefiero las ideas originales y novedosas aunque no sean prácticas.", "estilo": "Activo"},
    {"texto": "14. Admito y me ajusto a las normas sólo si me sirven para lograr mis objetivos.", "estilo": "Pragmático"},
    {"texto": "15. Normalmente encajo bien con personas reflexivas, y me cuesta sintonizar con personas demasiado espontáneas e imprevisibles.", "estilo": "Teórico"},
    {"texto": "16. Escucho con más frecuencia que hablo.", "estilo": "Reflexivo"},
    {"texto": "17. Prefiero las cosas estructuradas a las desordenadas.", "estilo": "Teórico"},
    {"texto": "18. Cuando poseo cualquier información, trato de interpretarla bien antes de manifestar alguna conclusión.", "estilo": "Reflexivo"},
    {"texto": "19. Antes de hacer algo, estudio con cuidado sus ventajas e inconvenientes.", "estilo": "Reflexivo"},
    {"texto": "20. Me entusiasmo con el reto de hacer algo nuevo y diferente.", "estilo": "Activo"},
    {"texto": "21. Casi siempre procuro ser coherente con mis criterios y sistemas de valores. Tengo principios y los sigo.", "estilo": "Teórico"},
    {"texto": "22. Cuando hay una discusión, no me gusta ir con rodeos.", "estilo": "Pragmático"},
    {"texto": "23. Me disgusta implicarme afectivamente en mi ambiente de trabajo. Prefiero mantener relaciones distantes.", "estilo": "Teórico"},
    {"texto": "24. Me gustan más las personas realistas y concretas que las teóricas.", "estilo": "Pragmático"},
    {"texto": "25. Me cuesta ser creativo/a, romper estructuras.", "estilo": "Teórico"},
    {"texto": "26. Me siento a gusto con personas espontáneas y divertidas.", "estilo": "Activo"},
    {"texto": "27. La mayoría de las veces expreso abiertamente cómo me siento.", "estilo": "Activo"},
    {"texto": "28. Me gusta analizar y dar vueltas a las cosas.", "estilo": "Reflexivo"},
    {"texto": "29. Me molesta que la gente no se tome en serio las cosas.", "estilo": "Teórico"},
    {"texto": "30. Me atrae experimentar y practicar las últimas técnicas y novedades.", "estilo": "Pragmático"},
    {"texto": "31. Soy cauteloso/a a la hora de sacar conclusiones.", "estilo": "Reflexivo"},
    {"texto": "32. Prefiero contar con el mayor número de fuentes de información. Cuantos más datos reúna para reflexionar, mejor.", "estilo": "Reflexivo"},
    {"texto": "33. Tiendo a ser perfeccionista.", "estilo": "Teórico"},
    {"texto": "34. Prefiero oír las opiniones de los demás antes de exponer la mía.", "estilo": "Reflexivo"},
    {"texto": "35. Me gusta afrontar la vida espontáneamente y no tener que planificar todo previamente.", "estilo": "Activo"},
```

5. Clases CalculoDeRendimiento y HerramientasEducativas

En esta parte continuamos con nuestras clases principales, la cual explicamos en nuestro diagrama de clases.

```python
class CalculoDeRendimiento:
    @staticmethod
    def obtener_rendimiento(nombre, apellido):
        conn =  get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT ciclo_1, ciclo_2, ciclo_3
            FROM usuarios WHERE nombre = %s AND apellido = %s
        """, (nombre, apellido))
        rendimiento = cursor.fetchone()
        release_db_connection(conn)

        if rendimiento:
            notas = [nota for nota in rendimiento if nota is not None]
            if notas:
                promedio = sum(notas) / len(notas)
                tipo_rendimiento = pd.cut([promedio], bins=[-float("inf"), 5.99, 9.99, 10.99, 12.99, 13.99, 20],
                                          labels=['Reprobado (D-)', 'Desaprobado (D)', 'Aprobado (C)', 'Bueno (B)', 'Muy Bueno (A)', 'Excelente (A+)'])[0]
                return {
                    "promedio": round(promedio, 2),
                    "tipo_rendimiento": tipo_rendimiento
                }
        return {"promedio": "N/A", "tipo_rendimiento": "Sin datos"}
```

```python
class HerramientasEducativas:
    herramientas = [
        {"nombre": "Sololearn: Es una aplicación y plataforma web que sirve para aprender a programar.", "tipo_app": "Activo"},
        {"nombre": "Duolingo: Es una aplicación gamificada para aprender idiomas con ejercicios interactivos.", "tipo_app": "Activo"},
        {"nombre": "Evernote: Es una aplicación para tomar notas, organizar información y gestionar tareas.", "tipo_app": "Reflexivo"},
        {"nombre": "MindMeister: Es una aplicación para crear mapas mentales y organizar ideas visualmente.", "tipo_app": "Reflexivo"},
        {"nombre": "Khan Academy: Es una plataforma educativa que te ayudará con cursos de matemáticas, ciencias y otros temas, con videos y ejercicios.", "tipo_app": "Teórico"},
        {"nombre": "Wolfram Alpha: Es un Motor de conocimiento computacional que resuelve ecuaciones y responde preguntas científicas.", "tipo_app": "Teórico"},
        {"nombre": "PhET Interactive Simulations: Es una app que realiza simulaciones interactivas para entender conceptos de matemáticas y ciencias.", "tipo_app": "Pragmático"},
        {"nombre": "Virtual ChemLab: Es una app que muestra laboratorio de química virtual para experimentación en un entorno seguro.", "tipo_app": "Pragmático"},
        {"nombre": "Photomath: Te ayudará a resolver problemas matemáticos escaneando con la cámara y muestra paso a paso la solución.", "tipo_app": "Pragmático"},
        {"nombre": "Mathway: Es una calculadora avanzada que resuelve problemas matemáticos y algebraicos con explicaciones.", "tipo_app": "Pragmático"},
        {"nombre": "Demos graphing calculator: Es una calculadora gráfica avanzada para visualizar ecuaciones y funciones matemáticas.", "tipo_app": "Activo"},
        {"nombre": "Periodic tablet: Es una app que muestra una tabla periódica interactiva con propiedades de los elementos.", "tipo_app": "Reflexivo"},
        {"nombre": "ChemCrafter: Es un juego educativo que permite experimentar con reacciones químicas virtuales.", "tipo_app": "Activo"},
        {"nombre": "Chemistry dictionary: Es un diccionario con términos y definiciones de química.", "tipo_app": "Teórico"},
        {"nombre": "Chemistry helper: Es una aplicación para cálculos y referencias en química.", "tipo_app": "Teórico"},
        {"nombre": "MEL Chemistry: Es una plataforma con experiencias de química en realidad aumentada y videos educativos.", "tipo_app": "Activo"},
        {"nombre": "Physics Toolbox: Es una app donde puedes encontrar un conjunto de herramientas para mediciones físicas usando sensores del móvil.\n", "tipo_app": "Activo"},
        {"nombre": "Physics Calculator: Es una app donde puedes usar una calculadora con fórmulas físicas para resolver problemas.", "tipo_app": "Activo"},
        {"nombre": "PHYWIZ: Es una app que te proporciona un asistente de física que resuelve problemas y explica conceptos.", "tipo_app": "Teórico"},
        {"nombre": "Fizyka: Es una aplicación educativa sobre física con simulaciones y explicaciones.", "tipo_app": "Teórico"},
        {"nombre": "Coursera: Es una plataforma con cursos en línea de universidades y empresas reconocidas.", "tipo_app": "Reflexivo"},
        {"nombre": "TED: Es una aplicación con charlas inspiradoras de expertos en diversos campos.", "tipo_app": "Reflexivo"},
        {"nombre": "GeoGebra: Es una herramienta interactiva para álgebra, geometría, cálculo y estadística.", "tipo_app": "Activo"},
        {"nombre": "Algebrator: Es una app que ayuda a resolver ecuaciones algebraicas con explicaciones detalladas.", "tipo_app": "Pragmático"},
        {"nombre": "Chemcollective: Es una app en donde puedes usar herramientas y simulaciones de química para la educación.", "tipo_app": "Reflexivo"},
        {"nombre": "Rosetta Stone: Es una aplicación que usa un método de aprendizaje de idiomas basado en la inmersión.", "tipo_app": "Pragmático"},
        {"nombre": "Babbel: Es una plataforma de aprendizaje de idiomas con cursos estructurados y lecciones interactivas.", "tipo_app": "Pragmático"},
        {"nombre": "Memrise: Es una plataforma de aprendizaje de idiomas con actividades dinamicas y juegos.", "tipo_app": "Activo"},
        {"nombre": "CodeCombat: Es una plataforma para aprender a programar con un enfoque gamificado tipo RPG.", "tipo_app": "Activo"},
        {"nombre": "Brilliant: Es una plataforma interactiva de aprendizaje de mátematica, ciencias y lógica.", "tipo_app": "Pragmatico"},
        {"nombre": "Todoist: Aplicación de gestión de tareas y organización del tiempo.", "tipo_app": "Pragmatico"},
        {"nombre": "Grammarly: Es un corrector gramatical avanzado para mejorar escritura en inglés.", "tipo_app": "Pragmatico"},
        {"nombre": "MyStudyLife: Planificador académico para gestionar tareas, exámenes y horarios.", "tipo_app": "Pragmatico"},
        {"nombre": "Desmos Scientific Calculator: Calculadora científica avanzada para resolver ecuaciones complejas.", "tipo_app": "Pragmatico"},
        {"nombre": "Microsoft Math Solver: Resuelve problemas matemáticos con explicaciones paso a paso.", "tipo_app": "Pragmatico"},
        {"nombre": "GoodNotes: Aplicación para tomar notas digitales organizadas y estructuradas.", "tipo_app": "Reflexivo"},
        {"nombre": "Notion: Herramienta de productividad para organizar proyectos y aprendizaje.", "tipo_app": "Reflexivo"},
        {"nombre": "AnkiDroid: Sistema de tarjetas de memoria para repasar conceptos a largo plazo.", "tipo_app": "Reflexivo"},
        {"nombre": "Coursera: Cursos en línea con contenido detallado y enfoque académico estructurado.", "tipo_app": "Reflexivo"},
        {"nombre": "Google Keep: Aplicación ligera para tomar notas rápidas y estructurarlas en categorías.", "tipo_app": "Reflexivo"},
        {"nombre": "Wolfram Alpha: Motor de búsqueda computacional que resuelve ecuaciones y problemas científicos.", "tipo_app": "Teorico"},
        {"nombre": "Physics Toolbox Suite: Aplicación con herramientas avanzadas para experimentos científicos.", "tipo_app": "Teorico"},
        {"nombre": "Stanford Online: Acceso a material educativo de la Universidad de Stanford.", "tipo_app": "Teorico"},
        {"nombre": "EdX: Plataforma de aprendizaje con cursos universitarios en ciencias y tecnología.", "tipo_app": "Teorico"},
        {"nombre": "MIT OpenCourseWare:Cursos gratuitos del MIT con contenido académico riguroso.", "tipo_app": "Teorico"},
        {"nombre": "Lightbot: Juego educativo que enseña lógica de programación de forma visual.", "tipo_app": "Activo"},
        {"nombre": "Todoist: Aplicación de gestión de tareas y organización del tiempo.", "tipo_app": "Reflexivo"},
        ]

    @classmethod
    def obtener_herramientas_recomendadas(cls, estilo):
        return [h["nombre"].replace("\n", "").strip() for h in cls.herramientas if h["tipo_app"] == estilo]
```

6. Ruta Principal

Definimos los métodos en donde comienza a compilar nuestro programa, con las páginas de bienvenida y para nuestro panel cuando el usuario este logueado.

```python
@app.route('/')
def home():
    return render_template("bienvenida.html")

def home1():
    if "usuario_id" in session:
        return redirect(url_for("dashboard"))  # Si ya está logueado, redirige al dashboard(panel)
    return redirect(url_for("registro"))
```

7. Guardar base de datos en CSV

Antes de iniciar la aplicación, se verifica si las tablas necesarias existen en la base de datos en nuestro método `verificar_base_datos()`:

```python
def guardar_en_csv(nombre, apellido, email, ciclo_1, ciclo_2, ciclo_3, Apps_usadas):
    print("Ejecutando guardar_en_csv()...")  # Debug

    # Si el archivo no existe, crearlo con encabezados
    if not os.path.exists(DATASET_PATH):
        df = pd.DataFrame(columns=["Nombre", "Apellido", "Email", "ciclo_1", "ciclo_2", "ciclo_3", "apps usadas"])
    else:
        try:
            df = pd.read_csv(DATASET_PATH)
        except pd.errors.EmptyDataError:
            df = pd.DataFrame(columns=["Nombre", "Apellido", "Email", "ciclo_1", "ciclo_2", "ciclo_3", "apps usadas"])

    # Agregar nueva fila con los datos
    nueva_fila = pd.DataFrame([{
        "Nombre": nombre,
        "Apellido": apellido,
        "Email": email,
        "ciclo_1": ciclo_1,
        "ciclo_2": ciclo_2,
        "ciclo_3": ciclo_3,
        "apps usadas": Apps_usadas
    }])

    df = pd.concat([df, nueva_fila], ignore_index=True)

    # Guardar datos asegurando codificación UTF-8 y sin índice
    df.to_csv(DATASET_PATH, index=False, encoding='utf-8-sig')
    print(f"Datos guardados en {DATASET_PATH}")
```

8. Registro e inicio de sesión

El usuario puede registrarse en `registro()` y sus datos se guardan en la base de datos y en el archivo CSV:

```python
def registro():
    if request.method == "POST":
        email = request.form.get("email", "").strip().lower()
        contrasena = request.form.get("contrasena", "").strip()
        nombre = request.form.get("nombre", "").strip().title()
        apellido = request.form.get("apellido", "").strip().title()
        Apps_usadas = request.form.get("Apps_usadas", "").strip()

        # Obtener y limpiar ciclos
        def limpiar_nota(cadena):
            cadena = re.sub(r"[^0-9\.,]", "", cadena.strip())
            return float(cadena.replace(",", ".")) if cadena not in ["", ".", ","] else None

        ciclo_1 = limpiar_nota(request.form.get("ciclo_1", ""))
        ciclo_2 = limpiar_nota(request.form.get("ciclo_2", ""))
        ciclo_3 = limpiar_nota(request.form.get("ciclo_3", ""))

        conn = get_db_connection()
        cursor = conn.cursor()

        # Verificar si el usuario ya existe
        cursor.execute("SELECT * FROM usuarios WHERE email = %s", (email,))
        if cursor.fetchone():
            conn.close()
            return render_template("registro.html", error="Este email ya está registrado. Intenta iniciar sesión.")

        # Insertar el nuevo usuario en la base de datos
        cursor.execute("""
            INSERT INTO usuarios (email, contrasena, nombre, apellido, ciclo_1, ciclo_2, ciclo_3, Apps_usadas)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (email, contrasena, nombre, apellido, ciclo_1, ciclo_2, ciclo_3, Apps_usadas))

        conn.commit()
        release_db_connection(conn)
        # Guardar en CSV
        guardar_en_csv(nombre, apellido, email, ciclo_1, ciclo_2, ciclo_3, Apps_usadas)

        return redirect(url_for("login"))
    return render_template("registro.html")
```

El usuario puede iniciar sesión en `login()`:

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form["email"].strip().lower()
        contrasena = request.form.get("contrasena", "").strip()

        conn =  get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT id_usuario, nombre, apellido FROM usuarios WHERE email = %s AND contrasena = %s", (email, contrasena))
        usuario = cursor.fetchone()
        release_db_connection(conn)

        if usuario:
            session["usuario_id"] = usuario[0]
            session["nombre"] = usuario[1]
            session["apellido"] = usuario[2]
            session["email"] = email
            return redirect(url_for("dashboard"))
        else:
            return render_template("login.html", error="⚠ Email o contraseña incorrectos")

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

9. Ruta de login hacia el dashboard (panel)

En esta parte se asegura que el usuario haya iniciado sesión para que pueda entrar a nuestros HTML, donde mostraremos algunas imágenes donde nos explicara cada estilo de aprendizaje.

```python
@app.route('/dashboard')
def dashboard():
    if "usuario_id" not in session:
        return redirect(url_for("login"))  # Si no hay sesión, redirige a login

    nombre = session["nombre"]
    apellido = session["apellido"]

    return render_template("dashboard.html", nombre=nombre, apellido=apellido)

@app.route('/imagen1')
def imagen1():
    return render_template("imagen1.html")

@app.route('/imagen2')
def imagen2():
    return render_template("imagen2.html")

@app.route('/imagen3')
def imagen3():
    return render_template("imagen3.html")

@app.route('/imagen4')
def imagen4():
    return render_template("imagen4.html")
```

10. Encuesta y estilo de aprendizaje

El usuario responde preguntas en encuesta las cuales hemos dividido en 4 partes de la encuesta <`int:pagina`>, y se almacenan sus respuestas:

```python
@app.route('/encuesta/<int:pagina>', methods=['GET', 'POST'])
def encuesta(pagina):
    if "usuario_id" not in session:
        return redirect(url_for("login"))

    usuario_id = session["usuario_id"]

    inicio = (pagina - 1) * 20
    fin = inicio + 20
    preguntas_pagina = preguntas[inicio:fin]

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT pregunta, respuesta FROM respuestas WHERE id_usuario = %s", (usuario_id,))
    respuestas_previas = dict(cursor.fetchall())
    release_db_connection(conn)

    if request.method == "POST":
        conn = get_db_connection()
        cursor = conn.cursor()
        for i, pregunta in enumerate(preguntas_pagina):
            respuesta = request.form.get(f'pregunta{inicio + i + 1}')
            if respuesta:
                cursor.execute("""
                INSERT INTO respuestas (id_usuario, pregunta, respuesta)
                VALUES (%s, %s, %s)
                ON CONFLICT (id_usuario, pregunta)
                DO UPDATE SET respuesta = EXCLUDED.respuesta;
                """, (usuario_id, pregunta["texto"], respuesta))

        conn.commit()
        release_db_connection(conn)
```

```python
    if pagina == 1:
        return redirect(url_for("imagen2"))
    elif pagina == 2:
        return redirect(url_for("imagen3"))
    elif pagina == 3:
        return redirect(url_for("imagen4"))
    elif pagina == 4:
        return redirect(url_for("resultado"))


return render_template(f"encuesta{pagina}.html", preguntas=preguntas_pagina, pagina=pagina, total_paginas=4, respuestas_previas=respuestas_previas)
```

11. Clase NotaPredictor

En esta clase como ya habíamos comentado en el diagrama de clases, tiene un método constructor y 2 métodos.

```python
class NotaPredictor:
    def __init__(self, model_path=MODEL_PATH):
        if os.path.exists(model_path):
            try:
                loaded_data = joblib.load(model_path)
                if isinstance(loaded_data, tuple) and len(loaded_data) == 2:
                    self.model, self.label_encoder = loaded_data
                else:
                    raise ValueError("El archivo del modelo no contiene los datos esperados.")
            except Exception as e:
                print(f"⚠ Error al cargar el modelo: {e}. Se reentrenará desde cero.")
                self.model = None
                self.label_encoder = LabelEncoder()
                self.entrenar_modelo()
        else:
            self.model = None
            self.label_encoder = LabelEncoder()
            self.entrenar_modelo()
```

```python
def entrenar_modelo(self):
    # Conectar a la base de datos SQLite y obtener datos actualizados
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT estilo, Apps_usadas, ciclo_1, ciclo_2, ciclo_3 FROM usuarios WHERE estilo IS NOT NULL AND Apps_usadas IS NOT NULL")
    usuarios_data = cursor.fetchall()
    release_db_connection(conn)

    if not usuarios_data:
        print("⚠ No hay datos suficientes para entrenar el modelo.")
        return

    df = pd.DataFrame(usuarios_data, columns=['estilo', 'Apps_usadas', 'ciclo_1', 'ciclo_2', 'ciclo_3'])

    # Codificar 'estilo' en valores numéricos
    self.label_encoder = LabelEncoder()
    df['estilo'] = self.label_encoder.fit_transform(df['estilo'].astype(str))

    # Convertir 'Apps_usadas' a la cantidad de aplicaciones usadas
    df['Apps_usadas'] = df['Apps_usadas'].astype(str).apply(lambda x: len(x.split(',')) if x else 0)

    # Calcular la nota promedio de los ciclos
    df['nota_promedio'] = df[['ciclo_1', 'ciclo_2', 'ciclo_3']].mean(axis=1)

    # Definir las variables de entrada y salida
    X = df[['estilo', 'Apps_usadas']]
    y = df['nota_promedio']

    if len(df) < 5:
        print(" Datos insuficientes para entrenar el modelo de predicción.")
        return

    # Entrenar el modelo
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Guardar el modelo entrenado y el label encoder correctamente
    joblib.dump((model, self.label_encoder), MODEL_PATH)
    self.model = model

    print(" Modelo entrenado correctamente con datos actualizados.")
```

```python
def predecir_nota(self, estilo, apps_usadas):
    if self.model is None:
        return "Modelo no entrenado"

    # Verificar si el estilo está en los datos entrenados
    if estilo not in self.label_encoder.classes_:
        print(f"⚠ Estilo '{estilo}' no reconocido, asignando 'Activo' por defecto.")
        estilo = 'Activo'

    estilo_codificado = self.label_encoder.transform([estilo])[0]

    # Contar la cantidad de apps usadas
    apps_cantidad = len(apps_usadas.split(',')) if apps_usadas else 0

    # Realizar la predicción
    entrada = np.array([[estilo_codificado, apps_cantidad]])
    nota_predicha = self.model.predict(entrada)[0]

    return round(nota_predicha, 2)
```

12. Método de `prediccion_nota`

```python
@app.route('/prediccion_nota', methods=['GET'])
def prediccion_nota():
    if "usuario_id" not in session:
        return redirect(url_for("login"))

    usuario_id = session["usuario_id"]

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT estilo, Apps_usadas FROM usuarios WHERE id_usuario = %s", (usuario_id,))
    usuario_data = cursor.fetchone()
    release_db_connection(conn)

    if not usuario_data:
        return "No se encontraron datos del usuario", 404

    estilo_aprendizaje, apps_usadas = usuario_data

    predictor = NotaPredictor()
    nota_predicha = predictor.predecir_nota(estilo_aprendizaje, apps_usadas)

    return render_template("prediccion_nota.html", nota_predicha=nota_predicha)
```

13. Ruta del resultado de la encuesta

Aquí tenemos métodos los cuales verificaran cuantas respuestas ha respondido el usuario, si la persona ha completado todas las preguntas entonces saldrá el resultado que le mostrara que estilo tiene y su promedio de los 3 últimos ciclos.

```python
@app.route('/resultado', methods=['GET', 'POST'])
def resultado():
    if "usuario_id" not in session:
        return redirect(url_for("login"))

    usuario_id = session["usuario_id"]
    nombre = session["nombre"]
    apellido = session["apellido"]

    conn = get_db_connection()
    cursor = conn.cursor()

    # Verificar cuántas respuestas ha respondido el usuario
    cursor.execute("SELECT COUNT(*) FROM respuestas WHERE id_usuario = %s", (usuario_id,))
    num_respuestas = cursor.fetchone()[0]

    # Número total de preguntas
    total_preguntas = len(preguntas)  # Asegurarse de que `preguntas` contiene todas las preguntas

    # Si el usuario no ha respondido todas las preguntas, redirigir a progreso
    if num_respuestas < total_preguntas:
        release_db_connection(conn)
        return render_template("progreso.html", error="Aún no has terminado la encuesta. Responde todas las preguntas para ver tu estilo de aprendizaje.")

    # Obtener respuestas del usuario
    cursor.execute("SELECT pregunta, respuesta FROM respuestas WHERE id_usuario = %s", (usuario_id,))
    respuestas = dict(cursor.fetchall())
    release_db_connection(conn)
```

```python
estilos = {"Activo": 0, "Reflexivo": 0, "Teórico": 0, "Pragmático": 0}

for pregunta in preguntas:
    respuesta = respuestas.get(pregunta["texto"])
    if respuesta == '+':
        estilos[pregunta["estilo"]] += 1

estilo_predominante = max(estilos, key=estilos.get)

# Guardar el estilo de aprendizaje en la base de datos
conn = get_db_connection()
cursor = conn.cursor()
cursor.execute("UPDATE usuarios SET estilo = %s WHERE id_usuario = %s", (estilo_predominante, usuario_id))
conn.commit()
release_db_connection(conn)

rendimiento = CalculoDeRendimiento.obtener_rendimiento(nombre, apellido)
promedio_rendimiento = rendimiento["promedio"]
tipo_rendimiento = rendimiento["tipo_rendimiento"]

# Obtener estilo y apps usadas para predecir la nota
conn = get_db_connection()
cursor = conn.cursor()
cursor.execute("SELECT estilo, Apps_usadas FROM usuarios WHERE id_usuario = %s", (usuario_id,))
usuario_data = cursor.fetchone()
release_db_connection(conn)

if usuario_data:
    estilo_aprendizaje, apps_usadas = usuario_data
    predictor = NotaPredictor()
    nota_predicha = predictor.predecir_nota(estilo_aprendizaje, apps_usadas)
else:
    nota_predicha = "N/A"
```

```
# Determinar la última página respondida
ultima_pagina = 1
if num_respuestas > 60:
    ultima_pagina = 4
elif num_respuestas > 40:
    ultima_pagina = 3
elif num_respuestas > 20:
    ultima_pagina = 2

return render_template('resultado.html',
                       nombre=nombre, apellido=apellido,
                       estilo=estilo_predominante,
                       promedio_rendimiento=promedio_rendimiento,
                       tipo_rendimiento=tipo_rendimiento,
                       ultima_pagina=ultima_pagina,
                       nota_predicha=nota_predicha)
```

14. Método recomendaciones

En esta parte, se buscará el resultado de la encuesta en donde saldra que estilo de aprendizaje es el usuario para así proceder con la recomendación.

```
@app.route('/recomendaciones', methods=['GET'])
def recomendaciones():
    usuario_id = session.get('usuario_id')
    if not usuario_id:
        return redirect(url_for('login'))

    conn = None
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute("SELECT estilo FROM usuarios WHERE id_usuario = %s", (usuario_id,))
        resultado = cursor.fetchone()

        if not resultado:
            return "No se encontró el estilo de aprendizaje.", 404

        Estilos = resultado[0]
        print(f"🔍 Estilo recuperado: {Estilos}")

        recomendaciones = HerramientasEducativas.obtener_herramientas_recomendadas(Estilos)

        print(f"Recomendaciones encontradas para estilo {Estilos}: {recomendaciones}")

        return render_template('recomendaciones.html', recomendaciones=recomendaciones, Estilos=Estilos)

    except Exception as e:
        print(f"⚠️ Error en recomendaciones: {e}")
        return "Ocurrió un error al obtener recomendaciones.", 500

    finally:
        if conn:
            release_db_connection(conn)  # Ahora se libera correctamente
```

15. Método de ver progreso

En esta parte el usuario podrá ver sus avances de la encuesta, es decir la cantidad de preguntas que respondió.

```python
@app.route("/ver_progreso")
def ver_progreso():
    if "usuario_id" not in session:
        return redirect(url_for("login"))  # Redirige a login si el usuario no ha iniciado sesión

    usuario_id = session["usuario_id"]
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT pregunta, respuesta FROM respuestas WHERE id_usuario = %s", (usuario_id,))
    respuestas = cursor.fetchall()

    release_db_connection(conn)
    respuestas = list(dict.fromkeys(respuestas)) #eliminara duplicados al ver el progreso de las respuestas

    return render_template("progreso.html", respuestas=respuestas)
```

16. Método `guardar_respuestas()`

Este método se encarga de guardar las respuestas marcadas por el usuario.

```python
@app.route('/guardar_respuestas', methods=['POST'])
def guardar_respuestas():
    if "usuario_id" not in session:
        return redirect(url_for("login"))

    usuario_id = session["usuario_id"]
    conn = get_db_connection()
    cursor = conn.cursor()

    for i, pregunta in enumerate(preguntas):
        respuesta = request.form.get(f'pregunta{i+1}')  # Se asegura de capturar correctamente la respuesta

        if respuesta:  # Solo guarda respuestas marcadas
            cursor.execute("""
                INSERT OR REPLACE INTO respuestas (id_usuario, pregunta, respuesta)
                VALUES (%s, %s, %s)
            """, (usuario_id, pregunta["texto"], respuesta))

    conn.commit()
    release_db_connection(conn)

    # Verificar si ya respondió todas las preguntas
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT COUNT(*) FROM respuestas WHERE id_usuario = %s", (usuario_id,))
    num_respuestas = cursor.fetchone()[0]
    release_db_connection(conn)

    # Redirigir al usuario a la siguiente página de la encuesta
    if num_respuestas < len(preguntas):
        return redirect(url_for("encuesta", pagina=(num_respuestas // 20) + 1))
    else:
        return redirect(url_for("resultado"))  # Si termina, ir a resultados
```

17. Ruta para cerra sesión

En esta parte vemos la parte de cerrar sesión y como se hace los llamados al HTML, además del inicio del programa con el puerto para que conectarse a una red local.

```python
# Ruta para cerrar sesión
@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))

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

verificar_base_datos()

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```