Vista de una ciudad desde lo alto de una torre

Descripción generada automáticamente

**PROYECTO BIG DATA**

**MP14\_UF1\_PROJ3**

**WEB POTTERHEAD**

Natalia Soria

Natalia García

INDEX

INTRODUCCIÓN…………………………………………………pag. 1

DISEÑO…………………………………………………...……..pag.3

PROGRAMACIÓN APPLICACIÓN……………………………..…….pag.5

TEST: TRIVIAL HARRY POTTER………………………….……...pag.6

TEST: ¿A QUE CASA PERTENECES?……………………………..pag.11

NUESTROS ESTUDIANTES…………..……………………………pag.

SUGERENCIAS DE MEJORAS…………………………………..pag.

FUENTES…………………..………………………………..pag.

INTRODUCCIÓN

Desde el principio tuvimos claro que la temática principal de nuestro proyecto sería la famosa saga de libros y películas: Harry Potter ya que ambas somos fans de la saga y nos gustaba la idea de trabajar sobre ella para poder realizar algo que a nosotras mismas nos gustaría encontrar por internet. ¿Pero qué podíamos hacer y que requisitos podríamos implementar para llevar a cabo nuestra idea?

Con esta premisa decimos hacer una web de Quiz o cuestionarios, donde los fans de saga pudieran poner a prueba sus conocimientos o averiguar a qué casa pertenecían.

Con la temática clara y un planteamiento previo, decidimos qué las funcionalidades que intentaríamos incorporar serían:

* Uso de arquitectura Flask
* Uso de base de datos
* Uso de matplotlib

DISEÑO

Empezamos a trabajar bien en el diseño de nuestra aplicación. Es necesario tener una buena base para poder trabajar bien sobre ella después.

En una primera instancia queríamos utilizar alguna API con quiz de Harry Potter ya preparados para poder hacer llamados a ella. Pero lamentablemente no encontramos ninguna que estuviera en castellano y que nos fuera de utilidad, así que decidimos hacer nosotras mismas las preguntas y guardar todo en una base de datos, para poder acceder a ellas desde nuestra aplicación Flask.   
Además de generas algunas otras tablas para guardar las puntuaciones de los usuarios y poder mostrarlas al finalizar el test y poder implementar una nueva función: guardar todos los usuarios que hicieran el test acerca de a que casa pertenecen para así poder llevar un conteo y mostrar, a cualquier usuario que quisiera, las graficas de cada casa, cuantos estudiantes habían participado y en qué casa estaba cada uno.

Para poder organizar bien la base de datos hicimos un diagrama:

Diagrama

Descripción generada automáticamente

Ahora que teníamos claro como queríamos que se estructurara nuestra base de datos nos encargamos de diseñar como queríamos que fuera nuestra web. Aspecto, paleta de colores, tipografía, cuantas páginas queríamos que tuviera….

Imagen que contiene Diagrama

Descripción generada automáticamenteHicimos un diseño previo con Figma para que ambas tuviéramos siempre la referencia visual de como debía de ser nuestra web:

Con todo esto claro, procedimos al desarrollo de nuestra web.

PROGRAMACIÓN APLICACIÓN

Lo primero que hicimos fue redactar todos los registros de nuestra base de datos: las preguntas del quiz test Harry Potter y las preguntas quiz ¿A qué casa perteneces? Junto con toda la información de creación de base de datos y tablas.

Uno de los problemas que nos encontramos fue no poder subir la base de datos en la nube o crear usuarios con credenciales universales para cualquier IP. Así que para poder usar nuestra aplicación había que, previamente, importar la base de datos trivialhp a nuestro servidor SQL.

Con la base de datos ya creada y subida a nuestro MYSQL empezamos a trabajar en la aplicación:

from flask import Flask,render\_template,

app= Flask(\_\_name\_\_)

@app.route('/')

def root():

    return render\_template('index.html')

Empezamos creando nuestra aplicación Flask, importando los recursos necesarios de flask y creando una ruta raíz donde nos dirige al index. Desde el html del index el usuario podrá elegir que función de nuestra aplicación quiere realizar:

* Test Trivial Harry Potter
* Test ¿A qué casa perteneces?
* Nuestros estudiantes

**TEST: TRIVIAL HARRY POTTER**

from flask import Flask,render\_template,request,redirect,url\_for, session, flash

def pedirPreguntasTrivialhp(numero\_preguntas=10):

    bd=mysql.connect(user="root",password="",host="127.0.0.1",

                     database="trivialhp")

    cursor=bd.cursor()

    cursor.execute("SELECT `pregunta`, `respuesta\_correcta`, `respuesta\_incorrecta1`,` respuesta\_incorrecta2`,` respuesta\_incorrecta3` FROM `trivial\_preguntas\_generales\_hp` ORDER BY RAND() LIMIT %s;", (numero\_preguntas,))

    listaPreguntas = cursor.fetchall()

    cursor.close()

    bd.close()

    return listaPreguntas

def obtenerPreguntaTrivialhp(preguntasRandom):

        pregunta=preguntasRandom[0]

        respuestas= [preguntasRandom[1],preguntasRandom[2],preguntasRandom[3],preguntasRandom[4]]

        random.shuffle(respuestas)

        respuesta1= respuestas[0]

        respuesta2=respuestas[1]

        respuesta3=respuestas[2]

        respuesta4=respuestas[3]

        return pregunta,respuesta1,respuesta2,respuesta3,respuesta4

def comprobarResultadoTrivialhp(pregunta,respuestaUsuario):

    bd=mysql.connect(user="root",password="",host="127.0.0.1",

                     database="trivialhp")

    cursor=bd.cursor()

    query=f"SELECT `respuesta\_correcta` FROM `trivial\_preguntas\_generales\_hp` where `pregunta`='{pregunta}';"

    cursor.execute(query)

    respuesta=cursor.fetchall()

    bd.close()

    respuestaOK=respuesta[0][0]

    if respuestaUsuario==respuestaOK:

        return True

    else:

        return False

Primero de todo definimos las funciones de búsquedas de preguntar trivial, escogiendo 10 preguntas aleatorias y, después, a la hora de mostrar, mostrar las respuestas también desordenadas, ya que si no la primera respuesta sería la correcta. Y una última función para comprobar si la respuesta dada por el usuario era correcta o no. Hasta este punto pudimos trabajar sin problemas ni fallos en el programa.

Nuestro principal problema fue a la hora de querer mostrar las preguntas y respuestas y registrar las respuestas una a una. No conseguíamos que el programa parara al hacer 10 preguntas o qué pasara de la primera pregunta de la lista preguntas obtenida con la función pedirPreguntaTrivialHP(). Después de mucho probar con bucles while y for y mirar mucho por diversos blogs de ayuda y un poco de ayuda de chatgpt conseguimos dar con la solución.  
Primero de todo, al querer mantener el juego sobre el mismo usuario y con la misma lista de preguntas se debía de guardar tanto las preguntas como el índex de la pregunta por la que va jugando el usuario con session, de esta forma se asocia tanto el listado de preguntas como el índex a el usuario que esta jugando en ese momento. Y sumando uno al índex cada vez que se mostraba una plantilla trivialHarryPotter.html para una pregunta pudimos recorrer toda la lista sin problemas.   
Y por último, a la hora de querer mostrar los resultados de ese usuario debíamos hacer un redirect a la función mostrarResultados(), no a la url.

Así enviamos toda la información guardada de la sesión de ese usuario.

Para poder usar session previamente había que dejar declarado en el programa una secret\_key:

app= Flask(\_\_name\_\_)

app.secret\_key='1234'

Conseguimos obtener el siguiente código funcional para esas funciones:

@app.route('/jugarTrivialHP/<usuario>', methods=["GET","POST"])

def jugarTrivialHP(usuario):

    listaPreguntas=session.get('preguntas')

    indexPreguntas=session.get('index')

    # comprobación que los datos han sido dados de forma correcta

    print(listaPreguntas)

    print(indexPreguntas)

    if request.method=="GET":

        if indexPreguntas< int(len(listaPreguntas)):

            pregunta,respuesta1,respuesta2,respuesta3,respuesta4=obtenerPreguntaTrivialhp(listaPreguntas[indexPreguntas])

            session['index']=indexPreguntas + 1

            print(indexPreguntas)

            return render\_template('trivialHarryPotter.html', preguntaHtml=pregunta,respuesta1Html=respuesta1,respuesta2Html=respuesta2,respuesta3Html=respuesta3,respuesta4Html=respuesta4,usuario=usuario)

        else:

            return  redirect(url\_for('mostrarResultados', usuario=usuario))

    elif request.method=="POST":

        bd=mysql.connect(user="root",password="",host="127.0.0.1",

                    database="trivialhp")

        pregunta=request.form.get("pregunta")

        respuestaUsuario=request.form.get("respuesta")

        resultado=comprobarResultadoTrivialhp(pregunta,respuestaUsuario)

        cursor=bd.cursor()

        if resultado==True:

            query=f"UPDATE `resultados\_hp\_test` SET `aciertos`=`aciertos`+1 WHERE `nombre`='{usuario}';"

        else:

            query=f"UPDATE `resultados\_hp\_test` SET `errores`=`errores`+1 WHERE `nombre`='{usuario}';"

        cursor.execute(query)

        bd.commit()

        bd.close()

        return redirect(url\_for('jugarTrivialHP',usuario=usuario))

Otra gran complicación que tuvimos fue a la hora de mostrar el grafico pie para el resultado del usuario actual. Cuando intentábamos realizar más de dos test el programa daba error. No permitía seguir porque informaba que el programa necesitaba trabajar sobre el main y no lo estaba haciendo.

Este problema es debido al uso de matplotlib, no trabaja bien cuando se necesita hacer más de un gráfico ni cuando se hace en paginas redirigidas.

@app.route ('/resultadoTrivial/<usuario>',methods=["GET","POST"])

def mostrarResultados(usuario):

    bd=mysql.connect(user="root",password="",host="127.0.0.1",

                     database="trivialhp")

    cursor=bd.cursor()

    query=f"SELECT `aciertos` FROM `resultados\_hp\_test` where `nombre`='{usuario}';"

    cursor.execute(query)

    aciertos=cursor.fetchone()

    puntuacion=aciertos[0]

    generarGraficoTestHP(usuario)

    return render\_template('resultadoTrivial.html',puntuacion=puntuacion)

Por eso lo primero que hicimos fue sacar la extracción de grafico en una función externa al flujo de la aplicación Flask, y, además, dejamos declarado que queríamos que matplotlib trabajara en el backend ‘Agg’, no en el programa. Así evitamos que nos diera el error

matplotlib.use('Agg')

def generarGraficoTestHP(usuario):

    bd=mysql.connect(user="root",password="",host="127.0.0.1",

                     database="trivialhp")

    cursor=bd.cursor()

    query=f"SELECT `aciertos`,`errores` FROM `resultados\_hp\_test` where `nombre`='{usuario}';"

    cursor.execute(query)

    data=cursor.fetchone()

    bd.close()

    plt.pie(data,labels=['ACIERTOS', 'ERRORES'],autopct='%0.1f%%',colors=colores)

    plt.axis("equal")

    static\_folder = os.path.join(app.root\_path, 'static')

    save\_path= os.path.join(static\_folder, 'assets','resultado.jpg')

    if os.path.exists(save\_path):

        os.remove(save\_path)

    plt.savefig(save\_path)

    plt.close()

También pusimos la condición de que si existía un archivo con ese nombre lo reescribiera para evitar posibles errores.

Como necesitamos que el programa guarde la imágenes dentro de la carpeta static/assets de nuestro proyecto, pero necesitamos que funcione desde cualquier ordenador, utilizamos el comando os.path.join que nos ayuda a localizar el path del root o usuario que esta utilizando la aplicación, y lo guarda en ese path en concreto. Si no el programa no podría encontrar la ruta ya que la indicada era la de nuestro ordenador, no una ‘’universal’’.

**TEST: ¿A QUÉ CASA PERTENECES?**

**NUESTROS ESTUDIANTES**

@app.route('/estudiantesCasas',methods=["GET","POST"])

def calcularEstudiantes():

    bd=mysql.connect(user="root",password="",host="127.0.0.1",

                     database="trivialhp")

    cursor=bd.cursor()

    query="SELECT `numEstudiantes`  FROM `estudiantes\_casas`;"

    cursor.execute(query)

    data=cursor.fetchall()

    bd.close()

    fig,ax= plt.subplots(figsize=(10,8))

    x=['Griffindor','Hufflepuff','Ravenclaw','Slytherin']

    y=[data[0][0],data[1][0],data[2][0],data[3][0]]

    colores=['#C70039','#ECCB25','#1511C6','#047134']

    ax.bar(x,y, color=colores)

    fig.set\_facecolor('#FcDEBE')

    ax.set\_facecolor('#FcDEBE')

    static\_folder = os.path.join(app.root\_path, 'static')

    save\_path= os.path.join(static\_folder, 'assets','graficoEstudiantesCasas.jpg')

    if os.path.exists(save\_path):

        os.remove(save\_path)

    plt.savefig(save\_path)

    plt.close()

    bd.close()

    return render\_template("estudiantesCasas.html")

Para esta función reutilizamos el código para crear el gráfico del pie resultado test, adaptándolo para extraer la información de los estudiantes de cada casa y, en este caso, conseguimos modificar el color de fondo del grafico además de los colores de las barras.

SUGERENCIAS DE MEJORAS

La base de datos podría intentar hacerse con mongoDB, para poder acceder desde la nube a ella y que el usuario que quisiera testearla no tuviera que importar nuestra base de datos a su BBDD local. Creando un usuario con contraseña que pudiera conectarse desde cualquier dirección IP.

También conseguir otro método de representación de gráficos, ya que matplotlib no nos ha permitido quitar el fondo del gráfico y que solo apareciera el pie o las columnas en el caso de los estudiantes de la casa, ni cambiar la tipografía del gráfico para que fuera en consonancia con el resto de la web…

Además de que nos ha dificultado mucho el trabajo ya que creaba conflicto con el flujo de la aplicación Flask y tuvimos que aplicar matplotlib.use(‘Agg’) para que el matplotlib no interactuara sobre el hilo del programa, si no sobre este backup, por que al ser páginas redirigidas nos daba error al intentar crear los gráficos de respuestas test.

También nos gustaría poder hacer algún test más como: ¿cuál es tu patronus?, una pagina sobre curiosidades sobre la saga, actualización de novedades (cómo la información acerca de la serie que están produciendo) y algún tipo de foro donde se pudieran registrar los usuarios y guardar su información acerca de los resultados de los test y hablar entre ellos.

FUENTES

* [Fondo transparente: Opacidad de la imagen con CSS y HTML (freecodecamp.org)](https://www.freecodecamp.org/espanol/news/fondo-transparente-opacidad-de-la-imagen-con-css-y-html/)[Cómo hacer Efecto Borroso o Desenfocado (Blur) en CSS (youtube.com)](https://www.youtube.com/watch?v=9nta9-bsMiw)
* [Incluir fuentes en CSS con @font-face | Kodetop  
  Incluir fuentes en CSS con @font-face | Kodetop](https://www.kodetop.com/incluir-fuentes-en-css-con-font-face/)<https://www.youtube.com/watch?v=pN3Ics48Hg8>
* [Cómo procesar los datos de solicitud entrantes en Flask | DigitalOcean](https://www.digitalocean.com/community/tutorials/processing-incoming-request-data-in-flask-es)
* [https://fonts.cdnfonts.com ur de fuente externa para pasar a css - Buscar con Google](https://www.google.com/search?q=https%3A%2F%2Ffonts.cdnfonts.com+ur+de+fuente+externa+para+pasar+a+css&sca_esv=e7ecf005577abd29&sca_upv=1&ei=T1U7Zv6_Lfjsi-gPuaaHqAc&ved=0ahUKEwi-o4DU7f2FAxV49gIHHTnTAXUQ4dUDCBA&uact=5&oq=https%3A%2F%2Ffonts.cdnfonts.com+ur+de+fuente+externa+para+pasar+a+css&gs_lp=Egxnd3Mtd2l6LXNlcnAiQGh0dHBzOi8vZm9udHMuY2RuZm9udHMuY29tIHVyIGRlIGZ1ZW50ZSBleHRlcm5hIHBhcmEgcGFzYXIgYSBjc3MyBBAhGBVIxVJQowVYylFwBHgAkAEAmAGrAaABlBaqAQQwLjI0uAEDyAEA-AEBmAIboALGFsICBhAAGBYYHsICCBAAGIAEGKIEwgIFECEYoAHCAgcQIRigARgKmAMAiAYBkgcEMy4yNKAHqUI&sclient=gws-wiz-serp)
* <https://www.cdnfonts.com/harry-p.font>
* [(python) Ejemplo de publicación en flask de una gráfica generada siguiendo el mismo proceso que se seguiríamos para crearla en un jupyter notebook. (github.com)](https://gist.github.com/1N0T/feafc42b5aebc58c9c16d23b0e03525e)
* [Examples — Matplotlib 3.8.4 documentation](https://matplotlib.org/stable/gallery/index.html)
* [Decorando gráficos en Matplotlib | E. J. Khatib (emilkhatib.es)](https://www.emilkhatib.es/decorando-graficos-en-matplotlib/)
* [Códigos de Colores HTML (htmlcolorcodes.com)](https://htmlcolorcodes.com/es/)
* [flask: Plantillas con jinja2 (4ª parte) - PLEDIN 3.0 (josedomingo.org)](https://www.josedomingo.org/pledin/2018/04/flask-plantillas-con-jinja2-4a-parte/)
* [Gráficas de pastel con Matplotlib | Numython](https://numython.github.io/posts/graficas-de-pastel-con-matplotlib/)
* CHATGPT