PROYECTO BIG DATA
MP14_UF1_PROJ3

WEB POTTERHEAD

Natalia Soria

Natalia García

INDEX

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:

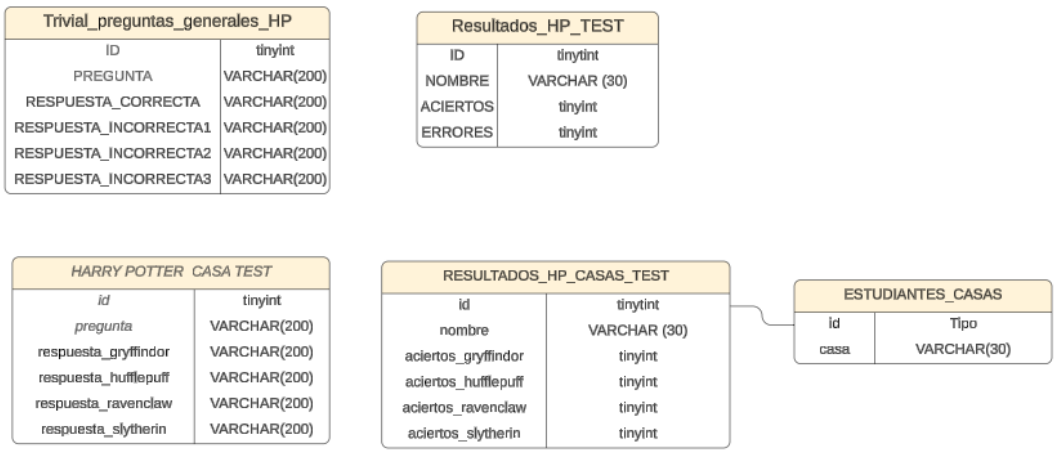
- Uso de arquitectura Flask
- Uso de base de datos
- Uso de matplotlib

DISEÑO

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

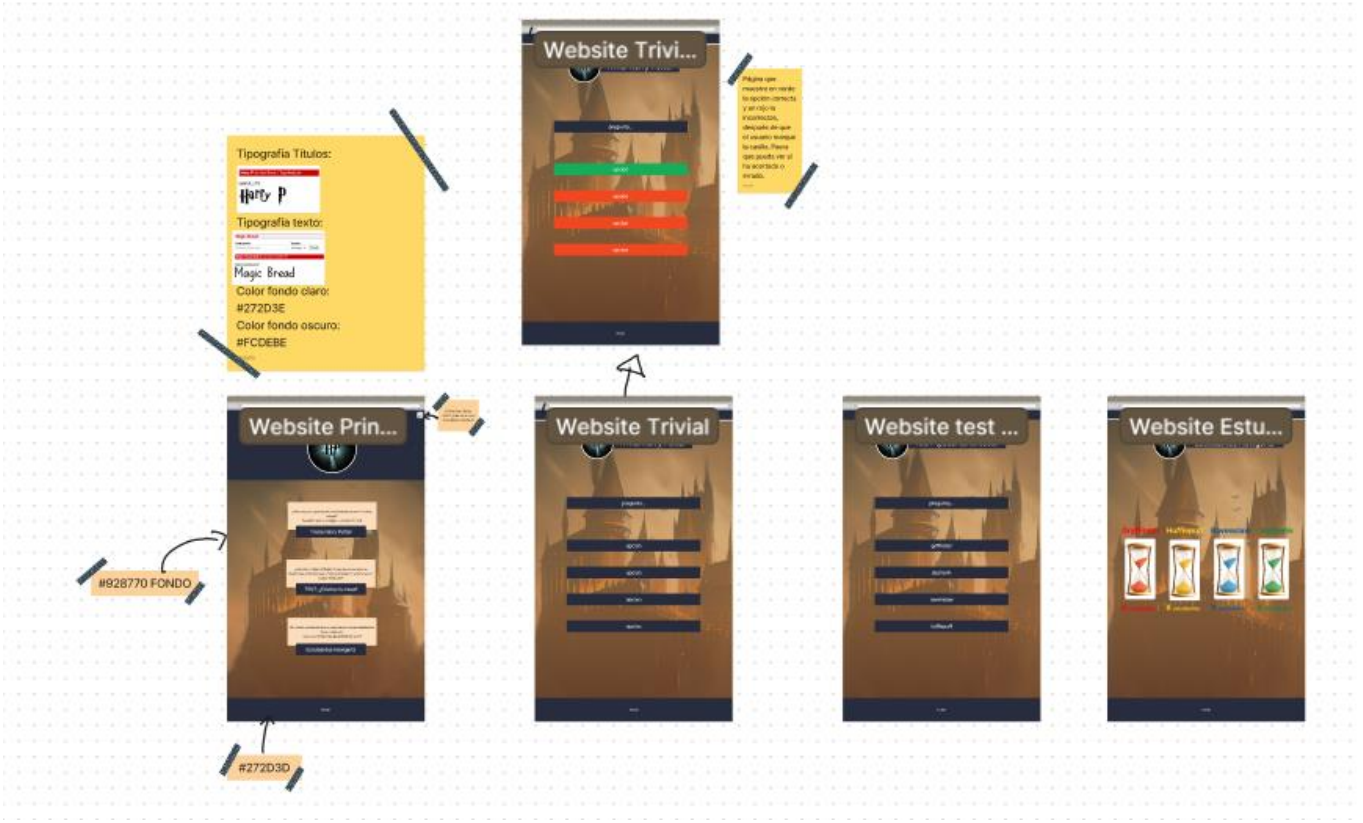
En una primera instancia queríamos utilizar alguna API con quiz de Harry Potter ya preparados para poder hacer llamados a ella. Pero lamentablemente no encontramos ninguna que estuviera en castellano y que nos fuera de utilidad, así que decidimos hacer nosotras mismas las preguntas y guardar todo en 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 qué casa pertenecen para así poder llevar un conteo y mostrar, a cualquier usuario que quisiera, las gráficas 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:



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….

Hicimos un diseño previo con Figma para que ambas tuviéramos siempre la referencia visual de cómo 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 está 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=obtenerP
reguntaTrivialhp(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 páginas 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 está 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?

Para la parte de tipo test empezamos creando una función:

```
def pedir_preguntas_test(numero_preguntas=10):
    bd = mysql.connect(user="root", password="", host="127.0.0.1", database="trivialhp")
    cursor = bd.cursor(dictionary=True)
    cursor.execute("SELECT * FROM TEST_Casas ORDER BY RAND() LIMIT %s", (numero_preguntas,))
    lista_preguntas = cursor.fetchall()
    cursor.close()
    bd.close()
    return lista_preguntas
```

Con esta función, la idea era pedir 10 preguntas de la tabla “testCasas”. Pero, además, queríamos guardar el tipo de opción seleccionada por cada pregunta en otra tabla (opción gryffindor, hufflepuff, ravenclaw, slytherin) y que las preguntas fueran de forma aleatoria. El “ORDER BY RAND()” es una opción para hacer random las preguntas, pero decidimos guardarla en una función aparte.

Por eso acudimos a realizar otras funciones:

```
def obtener_pregunta_test(preguntas_random):
    pregunta = preguntas_random[0]
    opciones_respuestas = preguntas_random[1:]
    random.shuffle(opciones_respuestas)
    return pregunta, opciones_respuestas
```

Esta función es la encargada de obtener preguntas aleatorias. Se fija desde el punto 0 hasta el final de las preguntas, la variable “opciones_respuestas” hace referencia a que las opciones también se encuentren en el random. De esta forma, con cada actualización en la página cambian las preguntas.


```python
def guardar_respuesta_test(pregunta_id, opcion_seleccionada):
    bd = mysql.connect(user="root", password="", host="127.0.0.1", database="trivialhp")
    cursor = bd.cursor()
    query = "INSERT INTO respuestas_casas (pregunta_id, opcion_seleccionada) VALUES (%s, %s)"
    cursor.execute(query, (pregunta_id, opcion_seleccionada))
    bd.commit()
    cursor.close()
    bd.close()
```

Con esta otra función, las opciones seleccionadas por cada pregunta se guardan en la tabla de “respuestas_casas”. Decidimos que, de las 4 opciones, la primera fuera para Gryffindor, la segunda para Hufflepuff, la tercera para ravenclaw y la última para slytherin. De esta manera, dependiendo de cual eligiera el usuario, se guardaría el nombre de la casa en dicha tabla (pero las respuestas seleccionadas no serían visibles para el jugador).

Ahora pasamos a la sección “jugar_test”:

Empezamos realizando una base para ver cómo se iba ejecutando.

```python
@app.route('/testCasas', methods=["GET", "POST"])
def jugar_test():
    if request.method == "GET":
        preguntas = obtener_preguntas_test()
        return render_template('testCasas.html', preguntas=preguntas)
    elif request.method == "POST":
        pregunta_id = request.form.get("pregunta_id")
        respuesta_seleccionada = request.form.get("respuesta")
        actualizar_resultado(pregunta_id, respuesta_seleccionada)
        preguntas = obtener_preguntas_test()
        return render_template('testCasas.html', preguntas=preguntas)
```

Pero estaba incompleta. Faltaba pedir la cantidad límite de preguntas, que fuesen de forma random y que haya una pregunta por página. El mayor problema que tuvimos fue que al ejecutarse la primera pregunta, y al querer pasar a la siguiente, no nos redireccionaba a la misma función para que se muestren las otras preguntas. Tuvimos que crear un índex, que cuente como contador, y así especificar

que mientras el índex sea menor al límite de preguntas requerido, seguirá tomando preguntas y de esta manera se ejecutaran hasta terminar.

función definitiva:

```python
@app.route('/testCasas', methods=["GET", "POST"])
def jugar_test():
    if request.method == "GET":
        listaPreguntas = pedir_preguntas_test()
        session['preguntas'] = listaPreguntas
        session['index'] = 0
        return mostrar_pregunta(listaPreguntas)

    elif request.method == "POST":
        pregunta_actual = session.get('pregunta_actual')
        pregunta_id = pregunta_actual['id']
        respuesta_seleccionada = request.form.get("respuesta")
        guardar_respuesta_test(pregunta_id, respuesta_seleccionada)

        listaPreguntas = session.get('preguntas')
        indexPreguntas = session.get('index')

        if indexPreguntas < len(listaPreguntas):
            return mostrar_pregunta(listaPreguntas)
        else:
            return redirect(url_for('porcentajeCasas'))
```

Esta función quedo como definitiva, aunque al principio tuvimos algunas complicaciones, termino ejecutándose como queríamos. Tomamos la función de pedir_preguntas_test para seleccionar de entre las 30 solo 10, y establecemos el index = 0 para que nos siga redirigiendo mientras el len(listaPreguntas) sea menor al establecido. A su vez guardamos las opciones seleccionadas en la tabla respuestas_casas tomando la función guardar_respuesta_test y tomando como parámetro la opción que se seleccionó y el id de la pregunta. Una vez terminadas las preguntas se moverá a la página de porcentajeCasas.


```python
def mostrar_pregunta(listaPreguntas):
    listaPreguntas = session.get('preguntas')
    pregunta_actual = listaPreguntas[session['index']]
    opciones_respuestas = [
        pregunta_actual['respuesta_griffindor'],
        pregunta_actual['respuesta_hufflepuff'],
        pregunta_actual['respuesta_ravenclaw'],
        pregunta_actual['respuesta_slytherin']
    ]
    random.shuffle(opciones_respuestas)
    session['pregunta_actual'] = pregunta_actual
    session['index'] += 1
    return render_template('testCasas.html', pregunta=pregunta_actual['pregunta'], opciones_respuestas=opciones_respuestas)
```

Con esta función nos mostraba en la página testCasas lo que fuimos implementando en la función de jugar_test. Y además establecimos que las opciones sean gryffindor, hufflepuff, etc. Las opciones también se mezclan para que no estén en un orden establecido.

Porcentajes:

```python
@app.route('/porcentajeCasas', methods=["GET", "POST"])
def porcentajeCasas():
    bd = mysql.connect(user="root", password="", host="127.0.0.1", database="trivialhp")
    cursor = bd.cursor()

    cursor.execute("SELECT COUNT(opcion_seleccionada) FROM respuestas_casas WHERE opcion_seleccionada='Griffindor'")
    griffindor = cursor.fetchone()[0]
    cursor.execute("SELECT COUNT(opcion_seleccionada) FROM respuestas_casas WHERE opcion_seleccionada='Hufflepuff'")
    hufflepuff = cursor.fetchone()[0]
    cursor.execute("SELECT COUNT(opcion_seleccionada) FROM respuestas_casas WHERE opcion_seleccionada='Ravenclaw'")
    ravenclaw = cursor.fetchone()[0]
    cursor.execute("SELECT COUNT(opcion_seleccionada) FROM respuestas_casas WHERE opcion_seleccionada='Slytherin'")
    slytherin = cursor.fetchone()[0]

    total_respuestas = griffindor + hufflepuff + ravenclaw + slytherin

    porcentaje_griffindor = (griffindor / total_respuestas) * 100 if total_respuestas != 0 else 0
    porcentaje_hufflepuff = (hufflepuff / total_respuestas) * 100 if total_respuestas != 0 else 0
    porcentaje_ravenclaw = (ravenclaw / total_respuestas) * 100 if total_respuestas != 0 else 0
    porcentaje_slytherin = (slytherin / total_respuestas) * 100 if total_respuestas != 0 else 0
```

Al finalizar el test, los porcentajes determinan a que casa perteneces. Es decir, si la mayoría de las opciones elegidas corresponden a Gryffindor, tu casa será Gryffindor, y lo mismo pasará con las otras casas. En caso de que haya igualdad en los porcentajes, no perteneces a una sola casa, sino a 2 o más.

En el html establecimos esta condición para saber cuál es mayor.

```
{% if porcentaje_griffindor > porcentaje_hufflepuff and porcentaje_griffindor > porcentaje_ravenclaw and porcentaje_griffindor > porcentaje_slytherin
    <h1 style="color:rgb(187, 0, 0);">"GRYFFINDOR"</h1>
    <img src="{{ url_for('static', filename='assets/griffindor.png') }}" height="250px" width="250px" >
```

También queríamos guardar los resultados en estudiantes, sumando un punto a la casa ganadora.

```
mayorPuntuacion=0
if griffindor>mayorPuntuacion:
    mayorPuntuacion=griffindor
    casaGanadora='Gryffindor'
if hufflepuff>mayorPuntuacion:
    mayorPuntuacion=hufflepuff
    casaGanadora='Hufflepuff'
if ravenclaw>mayorPuntuacion:
    mayorPuntuacion=ravenclaw
    casaGanadora='Ravenclaw'
if slytherin>mayorPuntuacion:
    casaGanadora='Slytherin'

query=f"UPDATE `estudiantes_casas` SET `numEstudiantes`=`numEstudiantes`+1 WHERE `casa`='{casaGanadora}';"
cursor.execute(query)
bd.commit()
bd.close()

return render_template('porcentajeCasas.html', porcentaje_griffindor=porcentaje_griffindor, porcentaje_hufflepuff=porcentaje_hufflepuff, por
```

Acá finalizamos con la parte del “TEST ¿A qué casa perteneces?” realizando las funciones necesarias para establecer una buena ejecución en el modo de juego.

1. pedir_preguntas_test(numero_preguntas=10)
2. obtener_pregunta_test(preguntas_random)
3. guardar_respuesta_test(pregunta_id, opcion_seleccionada)
4. jugar_test()
5. mostrar_pregunta(listaPreguntas)
6. porcentajeCasas()

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 página 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)
  Cómo hacer Efecto Borroso o Desenfocado (Blur) en CSS (youtube.com)
- Incluir fuentes en CSS con @font-face | Kodetop
  Incluir fuentes en CSS con @font-face | Kodetop
  https://www.youtube.com/watch?v=pN3Ics48Hg8
- Cómo procesar los datos de solicitud entrantes en Flask | DigitalOcean
- https://fonts.cdnfonts.com ur de fuente externa para pasar a css - Buscar con Google
- 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)
- Examples — Matplotlib 3.8.4 documentation
- Decorando gráficos en Matplotlib | E. J. Khatib (emilkhatib.es)
- Códigos de Colores HTML (htmlcolorcodes.com)
- flask: Plantillas con jinja2 (4ª parte) - PLEDIN 3.0 (josedomingo.org)
- Gráficas de pastel con Matplotlib | Numython
- CHATGPT:
  - Session para almacenar información sobre el usuario
  - Bucle for con índex para las preguntas trivial (tanto test como casas)
  - Random.Shuffle (para mezclar toda la lista de datos)
  - Aprender sobre el uso de backend para usar matplotlib
  - Ayuda con diseño en CSS
  - Uso de os.path.join para ubicar la dirección del directorio del root