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

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 que casa pertenecían.

Con la temática clara y un planteamiento previo decidimos que 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. Necesitamos 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 generar 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 que cada estaba cada uno.

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

Diagrama

Descripción generada automáticamente

A medida que íbamos avanzando con el código nos dimos cuenta de que faltaban tablas para agregar a la base de datos, así que las fuimos agregando despues.

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 Paginas queríamos que tuviera….

Imagen que contiene Diagrama

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

Con todo esto claro, procedimos a la aplicación 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. Por esta razón utilizamos el usuario root para poder acceder a los datos de la web.  
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

(…explicación..)

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

app.secret\_key='1234'

(…explicación..)

@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))

@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]

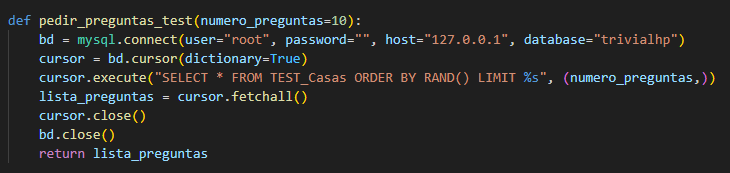
    generarGraficoTestHP(usuario)

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

(…explicación..)

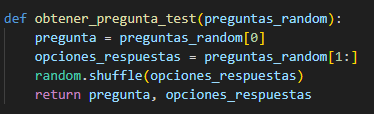
**TEST: ¿A QUÉ CASA PERTENECES?**

Para la parte de tipo test empezamos creando una función:

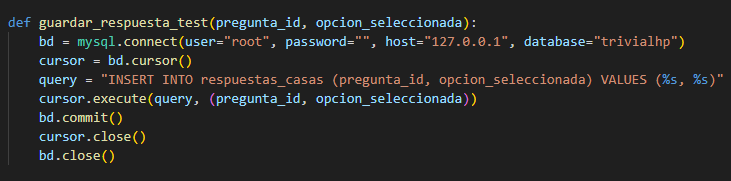


Con esta función, la idea era pedir 10 preguntas de la tabla “testCasas”. Pero además, queríamos guardar el tipo de opción seleccionada por cada pregunta en otra tabla (opcion gryffindor, hufflepuff, ravenclaw, slytherin) y que las preguntas fueran de forma aleatoria. El “ORDER BY RAND()” es una opción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

Por eso acudimos a realizar otras funciones:



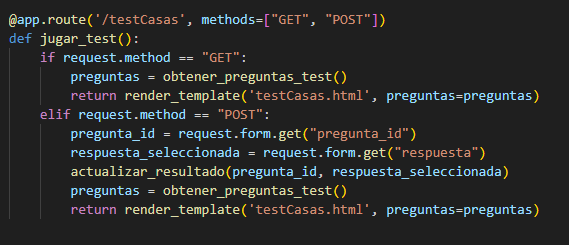
Esta función es la encargada de obtener preguntas aleatorias. Se fija desde el punto 0 hasta el final de las preguntas, la variable “opciones\_respuestas” hace referencia a que las opciones tambien se encuentren en el random. De esta forma, con cada actualización en la página cambian las preguntas.



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 Griffindor, 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).

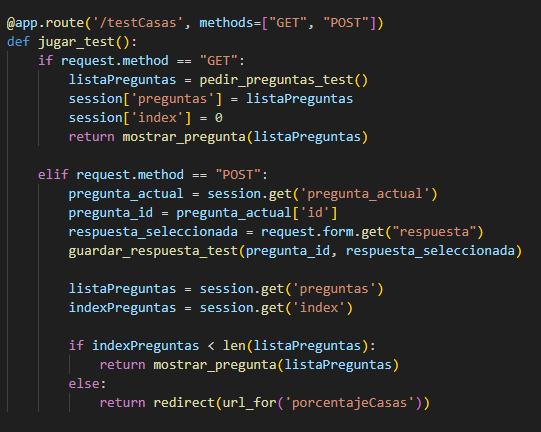
Ahora pasamos a la sección “jugar\_test”:

Empezamos realizando una base para ver cómo se iba ejecutando.

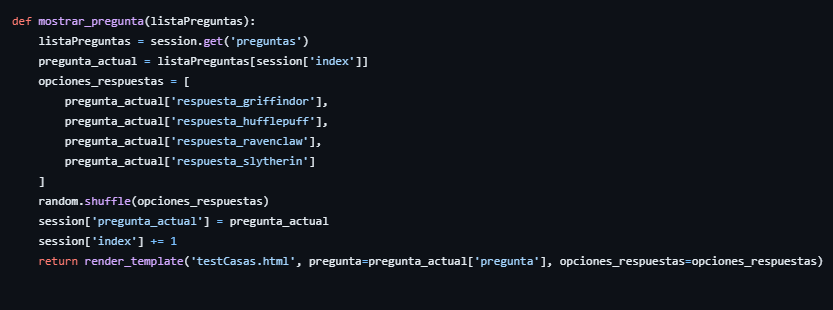


Pero estaba incompleta. Faltaba pedir la cantidad límite de preguntas, que fuesen de forma random y que haya una pregunta por página. El mayor problema que tuvimos fue que al ejecutarse la primera pregunta, y al querer pasar a la siguiente, no nos redireccionaba a la misma función para que se muestren las otras preguntas. Tuvimos que crear un index, que cuente como contador, y asi especificar que mientras el index sea menor al límite de preguntas requerido, seguirá tomando preguntas y de esta manera se ejecutaran hasta terminar.

Funcion definitiva:

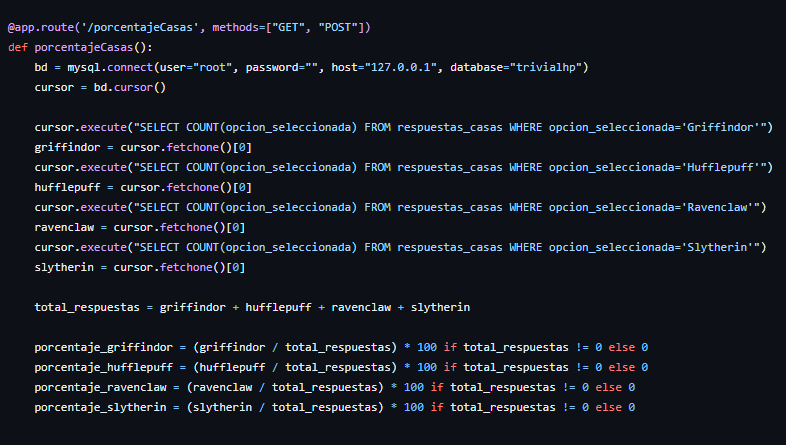


Esta función quedo como definitiva, aunque al principio tuvimos algunas complicaciones, termino ejecutándose como queríamos. Tomamos la funcion de pedir\_preguntas\_test para seleccionar de entre las 30 solo 10, y establecemos el index = 0 para que nos siga redirigiendo mientras el len(listaPreguntas) sea menor al establecido. A su vez guardamos las opciones seleccionadas en la tabla respuestas\_casas tomando la funcion guardar\_respuesta\_test y tomando como parametro la opcion que se selecciono y el id de la pregrunta. Una vez terminadas las preguntas se movera a la pagina de porcentajeCasas.



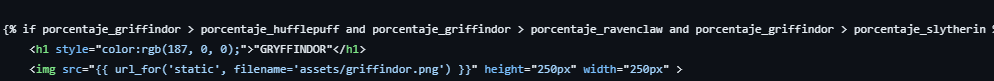
Con esta funcion nos mostraba en la pagina testCasas lo que fuimos implementando en la funcion de jugar\_test. Y ademas establecimos que las opciones sean griffindor, hufflepuff, etc. Las opciones tambien se mezclan para que no esten en un orden establecido.

Porcentajes:



Al finalizar el test, los porcentajes determinan a que casa perteneces. Es decir, si la mayoria de opciones elegidas correspondian a Griffindor, tu casa sera griffindor, y lo mismo pasara con las otras casas. En caso de que haya igualdad en los porcentajes, no perteneceras a una sola casa, sino a 2 o mas.

En el html establecimos esta condicion para saber cual es mayor.



Tambien queriamos guardar los resultados en estudiantes, sumando un punto a la casa ganadora.



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")

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 en fondo del gráfico y que solo apareciera el pie o las columnas en el caso de los estudiantes de las casa, 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, ya que al ser paguinas rediriguidas nos daba error al intentar crear los gráficos de respuestas test

Tambien podriamos implementar la creacion de la base de dato en el mismo python (SqlAlchemy), para que se maneje todo desde ahi y de esta manera no tener problemas de coneccion.

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/)
* <https://aulacm.com/codigos-web-css-y-html-wordpress/>
* <https://developer.mozilla.org/es/docs/Learn/HTML/Multimedia_and_embedding/Images_in_HTML>
* <https://www.digitalocean.com/community/tutorials/processing-incoming-request-data-in-flask-es>
* <https://developer.mozilla.org/es/docs/Web/CSS/CSS_flexible_box_layout/Aligning_items_in_a_flex_container>
* <https://developer.mozilla.org/es/docs/Web/CSS/@import>