M6_01_Sebastian_Cabrera_

1.Mejoras la interfaz gráfica

Implemente una base uniforme para todos los navegadores

main.css

```css
/*  Estilos generales   */

*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body{
  background: #1A2980;  /* fallback for old browsers */
  background: -webkit-linear-gradient(to right, #26D0CE, #1A2980);  /* Chrome 10-25, Safari 5.1-6 */
  background: linear-gradient(to right, #26D0CE, #1A2980); /* W3C, IE 10+/ Edge, Firefox 16+, Chrome 26+, Opera 12+, Safari 7+ */
  font-family: "Permanent Marker", cursive;
}
```

La tarjeta tiene un color personalizado sombras para resaltar y los botones tipo radio cambian de color cuando son chequeados

```css
/*  Estilos card*/

.card{
  background-color: moccasin;
}

/*  Estilos card-header */

.form-control{
  margin-bottom: 30px;
  box-shadow: 4px 4px 4px 4px rgba(0, 0, 0, 0.2);
}
.botones_radio{
  display: flex;
  align-items: center;
  align-content: center;
  justify-content: space-around;
}
.botones_radio input[type="radio"]:checked + label{
  background:#13EAB6;
  color: #000000;
}
.form-fecha{
  display: flex;
  align-items: flex-start;
  justify-content: space-evenly;
  margin: 15px;
}
h4{
  font-family: "Permanent Marker", cursive;
}
label{
  text-shadow: 2px 2px 4px #000000;
```

En lo que respecta a la lista de tareas implemente cambios asociados al uso de flex-box y el uso de word-wrap para que evitar que el contenido de la tarea exceda el limite del contenedor. También modifique márgenes y colores de los iconos

main.css

```css
/*  Estilos card-body   */

ul .lista{
  border-radius: 15px;
  border: 2px solid #000000;
}
.list-group-item {
  display: flex;
  justify-content: space-between
  margin-bottom: 1rem; /* Añadido para el espacio entre elementos */
  padding: 1rem;
}
.hecha {
  text-decoration: line-through;
  color: #cfcfcf;
  text-overflow:ellipsis;
}
.tarea-contenido {
  word-wrap: break-word;
  white-space: normal;
  margin-bottom: 0.5rem;
  width: 100%;
}
.bi-archive path {
  fill: #00ff00;
}
.bi-trash3 path {
  fill: #FF0000;
}
.btn-edicion{
  border: none;
}
.bi-pencil-fill{
  fill: #A020F0;
```

Di estilo a las implementaciones de footer y alert.

main.css

```css
/* estilos del footer */

footer{
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 0px 0px 20px 0px;
}
.btn-footer{
  color:#ffffff;
}

/* estilos del alert */

#alert {
  display: none;
  transition: opacity 5s ease;
  opacity: 0;
}
.alert-activo{
  display: flex;
  align-content: center;
  align-items: center;
  justify-content: center;
  margin: 100px;
  padding: 20px;
  opacity: 1;
}
```

Realice unas pequeñas modificaciones al estilo de las tablas implementadas para: hogar.html, estudio.html, trabajo.html.

```css
/* estilos de tablas */

.table{
  color: blanchedalmond;
}
th, td{
  text-align: center;
}
```

El estilo de tablas original es de bootstrap

```html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sandstone/bootstrap.min.css"
        integrity="sha384-zEpdAL7W11eTKeoBJK1g79kgl9qjP7g84KfK3AZsuonx38n8ad+f5ZgXtoSDxPOh" crossorigin="anonymous">
  <link rel="stylesheet" href="{{url_for('static',filename = 'main.css')}}">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
```

2.Añadir un campo Categoría para la tarea

Implemente una clase categoría y sume un atributo a la clase tarea

```python
class Categorias():
    """Categorias:
        Es una clase para ser usada en Tarea()
        instanciada en main.py home()
            args
            trabajo : str trabajo
            hogar : str hogar
            estudio : str estudio"""

    def __init__(self, trabajo, hogar, estudio):
        self.trabajo = trabajo
        self.hogar = hogar
        self.estudio = estudio
        print("Categorias creadas con éxito")


    def __str__(self):
        return "Categorias: {}-- {}-- {}".format( *args: self.trabajo, self.hogar, self.estudio)
```

```python
class Tarea(db.Base):
    """Tarea:
       Esta clase representa tareas en general
            args
            id : autoincremental (PK)
            contenido : str texto de tarea
            categoria : str son posibles solo tres
            depende de la clase Categoria()
                . trabajo
                . hogar
                . estudio
            hecha : bool tarea lista o pendiente
            fecha : DateTime de sqlAlchemy"""
    __tablename__ = "tarea"
    __table_args__ = {'sqlite_autoincrement': True}
    id = Column(Integer, primary_key=True)
    contenido = Column(String(200), nullable=False)
    categoria = Column(String(200), nullable=True)
    hecha = Column(Boolean)
    fecha = Column(DateTime, default=datetime.utcnow)

    def __init__(self, contenido, hecha, categoria, fecha):
        self.contenido = contenido
        self.hecha = hecha
        self.categoria = categoria
        self.fecha = fecha
        print("Tarea creada con éxito")

    # funcion para representacion por consola
    def __repr__(self):
```

En main se instancia la clase y se utiliza para renderizar index.html

```python
@app.route('/')
def home():
    """def home():
       Obtiene todos los registros de la tabla Tareas de la base de datos,
       y luego renderiza el template principal.
    """
    alert = False
    todas_las_tareas = db.session.query(Tarea).all()
    categorias1 = Categorias( trabajo: "Trabajo",  hogar: "Hogar",  estudio: "Estudio")
    categorias_dic = categorias1.__dict__
    categorias = []
    for key, value in categorias_dic.items():
        categorias.append(value)
    return render_template( template_name_or_list: "index.html",
                           lista_de_tareas=todas_las_tareas,
                           categorias=categorias,
                           datetime=datetime,
                           alert=alert)
```

Para crear un nuevo registro se obtiene con el método request la categoría que fue selecciona para la tarea

```python
ain.py
@app.route( rule: '/crear-tarea', methods=["POST"])
def crear():
    """def crear():
        Crea los registros de la tabla Tareas en la base de datos,
        según el método "POST" que reciba como parámetro.
    """
    try:
        categoria_py = request.form.get("categoria")
        if categoria_py is None:
            raise ValueError("¡¡¡Falta marcar categoria!!!")

        fecha_str = request.form["fecha"]

        if fecha_str is None:
            raise ValueError("¿Falta marcar fecha!")

        print(f"Tipo de fecha_py: {type(fecha_str)}")
        fecha_py = datetime.strptime(fecha_str, __format: '%Y-%m-%d')
        print(f"Tipo de fecha_py: {type(fecha_py)}")

        tarea = Tarea(contenido=request.form["contenido_tarea"].lower().title(),
                      categoria=categoria_py,
                      hecha=False,
                      fecha=fecha_py)
        db.session.add(tarea)
        db.session.commit()
    except ValueError as ve:
        alert = True
        print(f"Error: {ve}")
        db.session.rollback()
    return render_template( template_name_or_list: 'index.html'
```

En index.html se renderizan las categorías y a través de los atributos name y value la categoría chequeada se envía a la función crear()

```
                    autofocus>
            </div>
            <div class="botones_radio">
                {% for categoria in categorias %}
                <div class="btn-group" role="group" aria-label="Basic radio toggle button group">
                    <input type="radio" class="btn-check" id="btncheck{{ categoria }}" name="categoria"
                           value="{{ categoria }}" autocomplete="off">
                    <label class="btn btn-info" for="btncheck{{ categoria }}">{{ categoria }}</label>
                </div>
                {% endfor %}
            </div>
```

Las categorías instanciadas (que son fijas) las utilizo también en el footer de index.html para implementar un filtro personalizado del registro de tareas.

```
<footer>
<form action="/filtro-categoria" method="post">
    <div class="botones_footer">
        {% for categoria in categorias %}
        <div class="btn-group" role="group" aria-label="Default button group" >
            <button type="submit" class="btn btn-outline-primary btn-footer" name="categoria-en-footer" value="{{ categoria }}">
        </div>
        {% endfor %}
    </div>
```

Este filtro de tareas renderiza tres posibles .html con sus respectivas tablas y registros

```python
@app.route( rule: '/filtro-categoria', methods=["POST"])
def filtro_categoria():
    """def filtro_categoria():
        Esta función filtra las tareas por categoria,
        puede redirigir según corresponda hacia:
        hogar.html
        trabajo.html
        estudio.html.
    """
    try:
        todas_las_tareas = db.session.query(Tarea).all()
        tarea_hogar = []
        tarea_trabajo = []
        tarea_estudio = []
        categoria_py = request.form.get("categoria-en-footer")
        print("categoria que llega de footer", categoria_py)
        for tarea in todas_las_tareas:
            if categoria_py == "Hogar" and tarea.categoria == categoria_py and not tarea.hecha:
                tarea_hogar.append(tarea)
                print(tarea_hogar)
            elif categoria_py == "Trabajo" and tarea.categoria == categoria_py and not tarea.hecha:
                tarea_trabajo.append(tarea)
                print(tarea_trabajo)
            elif categoria_py == "Estudio" and tarea.categoria == categoria_py and not tarea.hecha:
                tarea_estudio.append(tarea)
                print(tarea_estudio)

    except Exception as e:
        print(f"Error en filtro_categoria(): {type(e).__name__}")
        return redirect(url_for("home"))

    if categoria_py == "Hogar":
        return render_template( template_name_or_list: "hogar.html", tarea_hogar=tarea_hogar)
```

3.Añadir un campo de Fecha límite para la tarea

Agregue el atributo fecha de tipo Datetime como hice con categoría modifique el constructor de la clase Tarea y en la función crear se uso el atributo strptime para convertirlo en un objeto de tipo datetime

https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior

Conversely, the `datetime.strptime()` class method creates a `datetime` object from a string representing a date and time and a corresponding format string.

En index.html el proceso es inverso. Para poder luego manipularlo el input de tipo date lo transforme con el método strftime

```html
<div class="form-fecha">
    <label for="fecha"><h4> Debe estar completa antes del: </h4></label>
    <input type="date" id="fecha" name="fecha"
           value="{{ datetime.now().strftime('%Y-%m-%d') }}"
           min="2024-01-01"
           max="2030-12-31"/>
```

`datetime.strftime(format)`

Return a string representing the date and time, controlled by an explicit format string. See also strftime() and strptime() Behavior and `datetime.isoformat()`.

El valor del input es siempre la fecha del día en que se ejecuta la aplicación

```
value="{{ datetime.now()
```

4.Posibilidad de Editar la tarea

La ruta editar-contenido permite filtrar la tarea que se desea modificar a través del id del registro y renderiza el template contenido.html

```python
@app.route( rule: "/editar-contenido", methods=["POST"])
def activar_section_contenido():
    """def activar_section_contenido():
        Captura la tarea cuyo contenido se desea modificar,
        y redirige a contenido.html."""
    try:
        tarea_id = request.form["tarea_id"]
        tarea = db.session.query(Tarea).filter_by(id=int(tarea_id)).first()
        if not tarea:
            raise ValueError(f"la tarea con id {tarea_id} no existe en la base de datos")
    except Exception as e:
        db.session.rollback()
        print(f"Error en editar-contenido: {type(e).__name__}")
        return render_template( template_name_or_list: "index.html",
                               datetime=datetime)

    return render_template( template_name_or_list: "contenido.html", tarea=tarea)
```

A ella se accede desde un botón implementado en index.html

```
<span>
    <form action="/editar-contenido" method="post">
        <button class="btn-edicion">
            <input type="hidden" name="tarea_id" value="{{ tarea.id }}">
            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
                 class="bi bi-pencil-fill" viewBox="0 0 16 16">
            <path d="M12.854.146a.5.5 0 0 0-.707 0L10.5 1.793 14.207 5.5l1.647-1.646a.5.5 0 0 0
            </svg>
        </button>
    </form>
</span>
```

En contenido.html muestro un formulario que permite la modificación del contenido y de la fecha del registro en cuestión

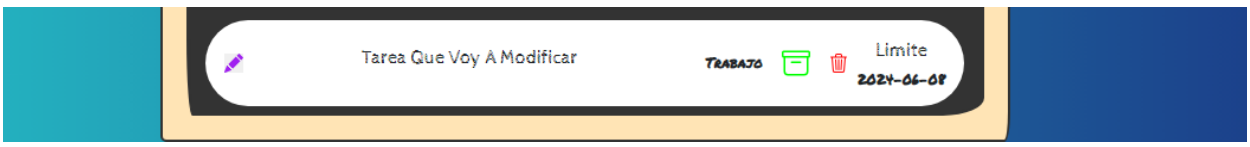
```
    <label for="floatingInput"><h5> nueva tarea </h5></label>
</div>
<form action="/edicion-contenido" method="post">
    <div class="form-floating">
        <input type="hidden" name="tarea_id" value="{{ tarea.id }}">
        <input type="text" name="nuevo_contenido_tarea" class="form-control text-center fw-bold"
               id="floatingPassword"
               placeholder="Ingrese modificación" autofocus>
        <label for="floatingPassword"><h5> ¿nueva fecha? </h5></label>
        <div class="form-floating d-flex justify-content-center mb-4">
        <input type="date" id="fecha" name="fecha" value="{{ tarea.fecha.strftime('%Y-%m-%d') }}"
               min="2024-01-01"
               max="2030-12-31" />
        </div>
    </div>
    <div>
        <button type="submit" class="btn btn-block guardar "><h4> Guardar nueva tarea </h4></button>
    </div>
</form>
```

Esta edición es implementada en la ruta edición-contenido que una vez cumplimentada retorna a la pagina inicial

```python
@app.route( rule: '/edicion-contenido', methods=["POST"])
def editar_contenido():
    """def editar_contenido():
        Modifica:  tarea.contenido y/o tarea.fecha,
        luego redirige a home."""
    try:
        tarea_id = request.form["tarea_id"]
        tarea = db.session.query(Tarea).filter_by(id=int(tarea_id)).first()
        tarea.contenido = request.form["nuevo_contenido_tarea"]
        fecha_str = request.form["fecha"]
        fecha_py = datetime.strptime(fecha_str,  __format: '%Y-%m-%d')
        tarea.fecha = fecha_py
        db.session.commit()
    except Exception as e:
        db.session.rollback()
        print(f"Error en editar: {type(e).__name__}")

    return redirect(url_for("home"))
```

Vista en servidor



Vista en DB Browser

id	contenido	categoria	hecha	fecha
12	Aprender Jinja	Estudio	0	2025-05-10 00:00
14	Si La Tearea Es Muy Larga Y Ademas Icluye ...	Trabajo	0	2025-01-01 00:00
15	Lavar La Ropa	Hogar	0	2024-07-06 00:00
17	Aprender Python	Estudio	0	2024-06-08 00:00
18	Tarea Que Voy A Modificar	Trabajo	0	2024-06-08 00:00

```
xecution finished without errors.
esult: 5 rows returned in 9ms
: line 1:
ELECT * FROM tarea
```

Vista en servidor

Vista en DB Browser

id	contenido	categoria	hecha	fecha
12	Aprender Jinja	Estudio	0	2025-05-10 ...
14	Si La Tearea Es Muy Larga Y Ademas Icluye Parece Que Supera El Contenido Posoble Del Reglon	Trabajo	0	2025-01-01 ...
15	Lavar La Ropa	Hogar	0	2024-07-06 ...
17	Aprender Python	Estudio	0	2024-06-08 ...
18	Tarea modificada	Trabajo	0	2025-07-10 ...

```
ıtion finished without errors.
.t: 5 rows returned in 9ms
.ne 1:
:T * FROM tarea
```

5.Otras implementaciones

Implemente notas en todas las funciones y clases para facilitar la documentación y mostrar los propósitos de cada una de ellas

El manejo de errores que puedan urgir durante la ejecución lo hice a través de implementar try/except, mostrando mensajes de error al usuario. Ademas implemente el método rollback() de sqlalchemy para revertir una transacción que esta sujeta a un posible error.

Utilice un alert que muestra un mensaje de error que puede ser generado al momento de crear una tarea. Este mensaje se mantiene oculto en tanto la variable mantenga su condición de falsa.

Implemente un filtro de tareas por categorías al que se puede acceder desde botones ubicados al pie de página.