Creación del Front-End desde Python.

Pasos:

1. Partimos de que ya tenemos en el **backend** funcionando las siguientes rutas:
    a. **GET /genero/:** lista todos los géneros de las películas
    b. **POST /genero/:** agregar un genero.
    c. **PUT /genero:** actualizar un genero
    d. **DELETE /genero/:** eliminar un genero
    e. **GET /pelicula/:** lista todos películas
    f. **POST /pelicula/:** agregar una película.
    g. **PUT /película/:** actualizar una película
    h. **DELETE /pelicula/:** eliminar una película

2. La conexión a la base de datos se debe hacer a mongo atlas.
3. Crear los htmls utilizando plantillas de jinja en Python
4. Crear la hoja de estilos en un archivo llamado **app.css** que debe estar dentro de la carpeta **static** y dentro de ella en una carpeta llamada **css**.
5. Crear el documento o plantilla principal. En el ejemplo se llama **index.html**. En este archivo incluimos las librerías que estamos seguros que las requerimos en todas los documentos html. También incluimos nuestra hoja de estilos.

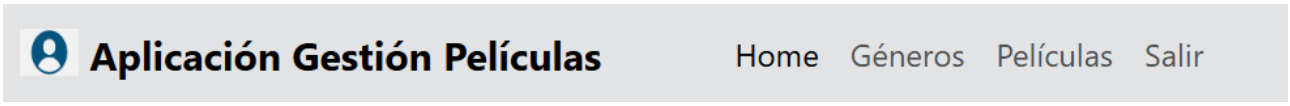
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!--bootstrap-->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
    rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
    <!--sweet alert-->
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
    <!-- fontawesome-->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <!--estilos propios-->
    <link rel="stylesheet" href="../static/css/app.css">
    <title>GESTIÓN PELÍCULAS</title>
</head>
```

```
<body>
    <div class="container-fluid">
        <header>
            {%block encabezado %}
            {%endblock  %}
        </header>
        <nav>
            {%block menu %}
            {%endblock  %}
        </nav>
        <section>
            {%block contenido %}
            {%endblock  %}
        </section>
        <footer>
            {%block piePagina %}
            {%endblock  %}
        </footer>
    </div>
</body>
</html>
```

6. Crear un documento html para el **encabezado**: Aquí colocar de acuerdo a su propuesta de diseño

7. Crear un documento html para el **pie de pagina**: Aquí colocar su propuesta de diseño.

8. Crear un documento html con un **menú** que contenga las siguientes opciones:
    a. **Home**: Muestra la página principal
    b. **Géneros**: Muestra la lista de generos
    c. **Películas**: Muestra la lista de peliculas
    d. **Salir**: salir de la aplicación.



9. Crear documento html que se muestre al iniciar la aplicación. En el ejemplo se va a crear un archivo llamado **contenido.html** así:

```
es > <> contenido.html
{% extends "index.html" %}
{% block menu %}
    {% include "menu.html" %}
{%endblock %}
{% block contenido %}

    <h3 class="text-center fw-bold">Funcionamiento Aplicación</h3>
    <p>
        Aplicación desarrollada en Python con Flask y como motor de bases de Datos MongoDB.
        Realiza los procesos de l CRUD a una base de datos.
    </p>
    <p class="fw-bold">Funcionalidades:</p>
    <ul>
        <li>Add Género</li>
        <li>Update Género</li>
        <li>Delte Género</li>
        <li>List Géneros</li>
        <li>Add Película</li>
        <li>Update Película</li>
        <li>Delte Película</li>
        <li>List Películas</li>
    </ul>

{%endblock %}
{% block piePagina %}
    {% include "piePagina.html" %}
{%endblock %}
```

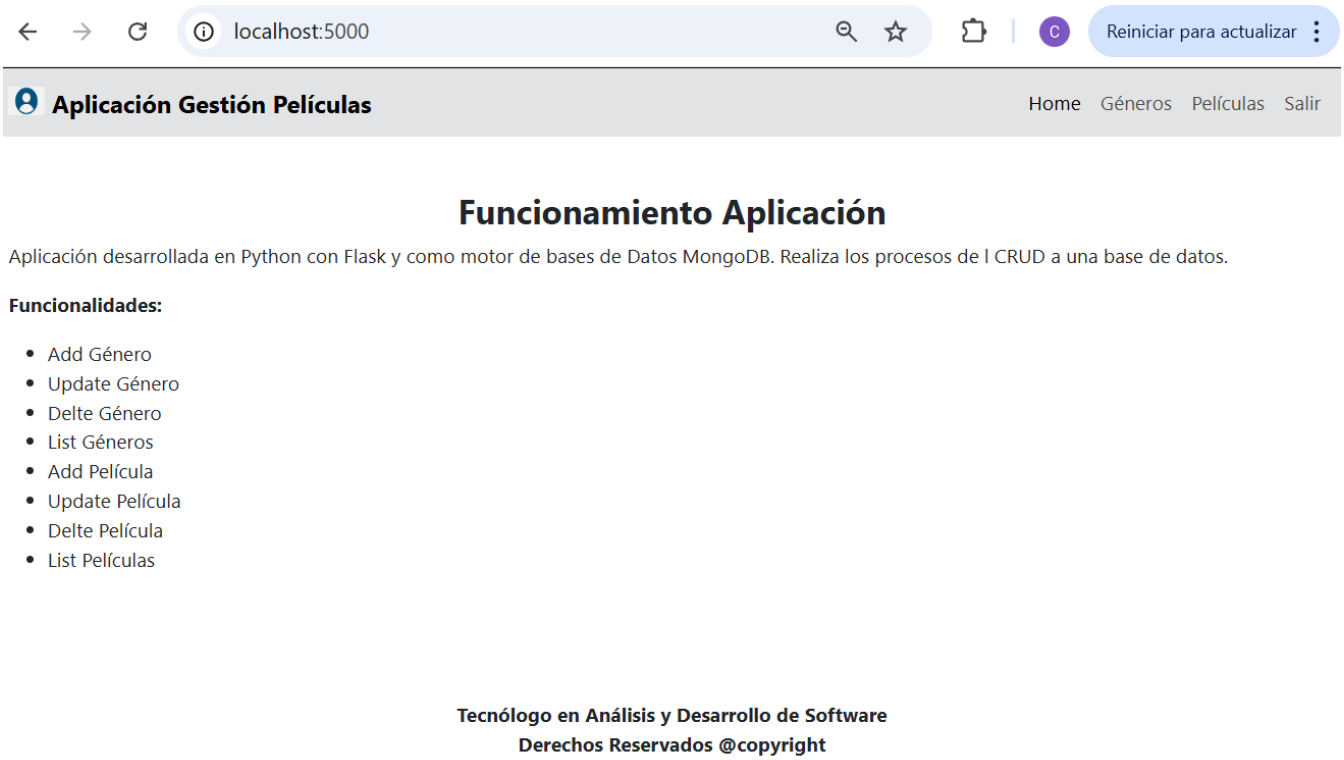
De acuerdo al uso de plantillas con **jinja** en **Python**, debemos primero extender el documento principal que en nuestro caso se llama **index.html**. También se pueden usar los bloques existentes en el documento principal y dentro de cada uno de ellos colocar código, o si ya existe en un archivo se debe incluir el archivo. En nuestro ejemplo estamos incluyendo el archivo **menú.html**, y **piePagina.html**. En el bloque llamado **contenido** estamos incluyendo el html de la página principal.

De la misma forma como se realiza este documento así mismo hacemos para todos los documentos html que cambian el bloque contenido.

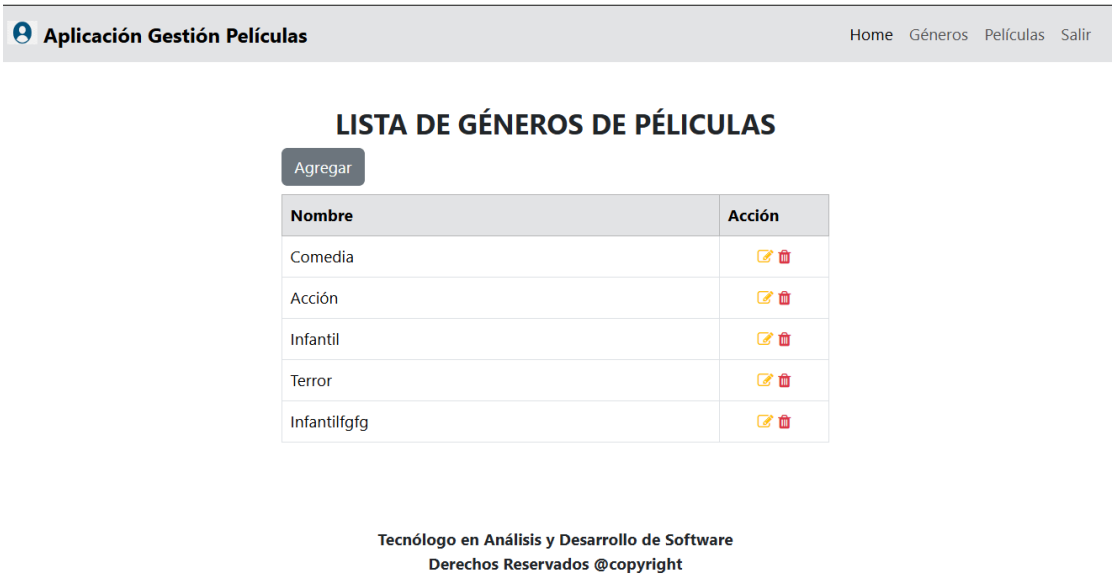
10. Crear una ruta en Python para la raíz inicial de la aplicación, la cual debe renderizar el archivo contenido.html. En nuestro ejemplo la ruta la estamos creando en el archivo **app.py**

```
Tabnine | Edit | Test | Explain | Document
@app.route("/")
def inicio():
    return render_template("contenido.html")
```

11. Al arrancar la aplicación se muestra la siguiente interfaz:



12. Propuesta de interfaz para listar los Géneros



13. Código html del archivo que lista los géneros

templates > listarGeneros.html

```
{% extends "index.html" %}
{% block menu %}
    {% include "menu.html"%}
{%endblock%}
{% block contenido %}
    <!--html para lstar-->
    <div class="w-50" style="margin: 0 auto">
        <h3 class="text-center fw-bold">LISTA DE GÉNEROS DE PÉLICULAS</h3>
        <div>
            <a href="/vistaGenero/">
                <button class="btn btn-secondary">Agregar</button>
            </a>
        </div>
        <table class="table table-bordered mt-2">
            <thead class="table-secondary">
                <tr>
                    <th>Nombre</th>
                    <th>Acción</th>
                </tr>
            </thead>
            <tbody>
                {% for genero in generos %}
                    <tr>
                        <td width="80%">{{genero.nombre}}</td>
                        <td class="text-center">
                            <i class="fa fa-edit text-warning"></i>
                            <i class="fa fa-trash text-danger"></i>
                        </td>
                    </tr>
                {% endfor%}
            </tbody>
        </table>
    </div>
{%endblock%}
{%block piePagina%}
    {% include "piePagina.html" %}
{%endblock%}
```

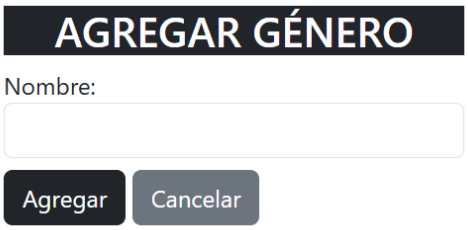
14. Crear una ruta en el **backend** para mostrar la anterior vista así:

```python
Tabnine | Edit | Test | Explain | Document
@app.route("/generos/", methods=['GET'])
def listarGeneros():
    try:
        mensaje=""
        generos=Genero.objects()
    except Exception as error:
        mensaje=str(error)

    return render_template("listarGeneros.html",
                           generos=generos,mensaje=mensaje)
```

La ruta debe ser creada en el archivo **genero.py** que está en la carpeta **routes**.

15. Crear documento html para agregar un genero.



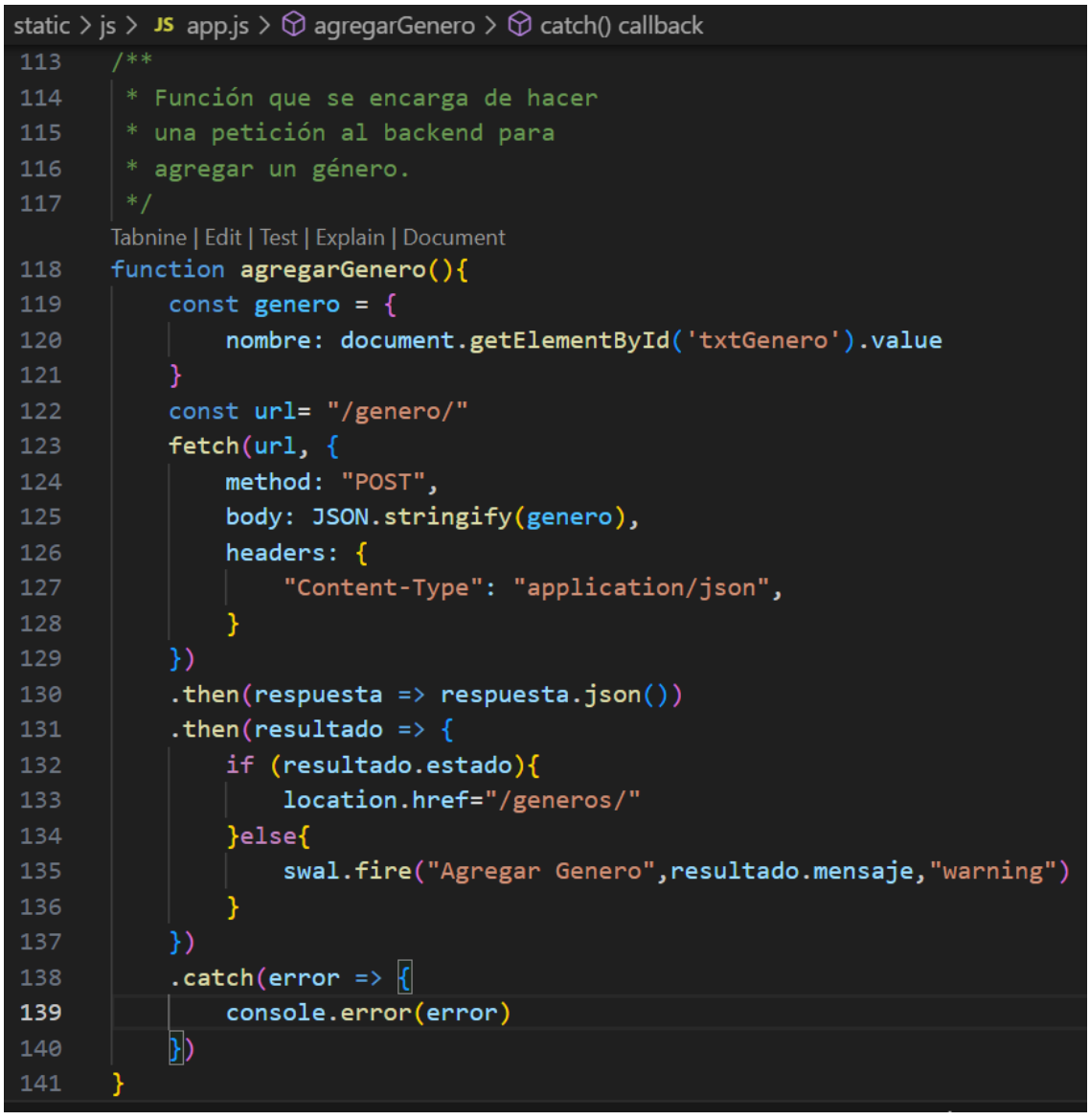
16. Código hrml del formulario para agregar un genero

templates > <> frmAgregarGenero.html > ...

```html
{% extends "index.html" %}
{% block menu %}
    {% include "menu.html" %}
{%endblock %}
{% block contenido %}
<script src="../static/js/app.js"></script>
<div class="w-25" style="margin:0 auto">
    <form id="frmAgregarGenero">
        <h3 class="text-center text-white bg-dark">AGREGAR GÉNERO</h3>
        <div class="mb-2">
            <label for="txtGenero">Nombre:</label>
            <input type="text" name="txtGenero" id="txtGenero"
            class="form-control" required>
        </div>
        <div class="mb-2">
            <button type="button" class="btn btn-dark" onclick="agregarGenero()">Agregar</button>
            <a href="/generos/">
                <button type="button" class="btn btn-secondary">Cancelar</button>
            </a>
        </div>
    </form>
</div>
{%endblock %}
{% block piePagina %}
    {% include "piePagina.html" %}
{%endblock %}
```

17. El formulario anterior llama a un archivo javascript llamado **app.js** que contiene la función que realiza la petición al servidor para agregar el genero. La función es invocada al dar clic en el **botón** que dice **agregar** del formulario.

18. Función **agregarGenero()** del archivo **app.js**



La función anterior realiza una petición al servidor mediante la api **fetch** a la ruta **/genero/** de tipo post.

19. Código de la ruta **/genero/** del **backend** que permite agregar el genero a la base de datos

```
Tabnine | Edit | Test | Explain | Document
@app.route("/genero/", methods=['POST'])
def addGenero():
    try:
        mensaje=None
        estado=False
        if request.method=='POST':
            datos= request.get_json(force=True)
            genero = Genero(**datos)
            genero.save()
            estado=True
            mensaje="Genero agregado correctamente"
        else:
            mensaje="No permitido"
    except Exception as error:
        mensaje=str(error)

    return {"estado":estado, "mensaje":mensaje}
```

20. A continuación vamos a mostrar el proceso de **listar las películas**. Para listar las películas se propone la siguiente interfaz:

21. Crear una ruta en el archivo **película.py** que renderice el html donde se van a listar las películas.

A continuación se muestra el código:

```
Tabnine | Edit | Test | Explain | Document
@app.route("/peliculas/", methods=['GET'])
def listarPeliculas():
    peliculas = Pelicula.objects()
    generos = Genero.objects()
    print(generos)
    return render_template("listarPeliculas.html",
                           peliculas=peliculas,
                           generos=generos)
```

El código anterior responde a la petición **/películas/** de tipo **GET** la cual consulta las películas y géneros existentes, y los retorna a la vista **listarPeliculas.html.**

22. Código html del archivo **listarPeliculas.html**

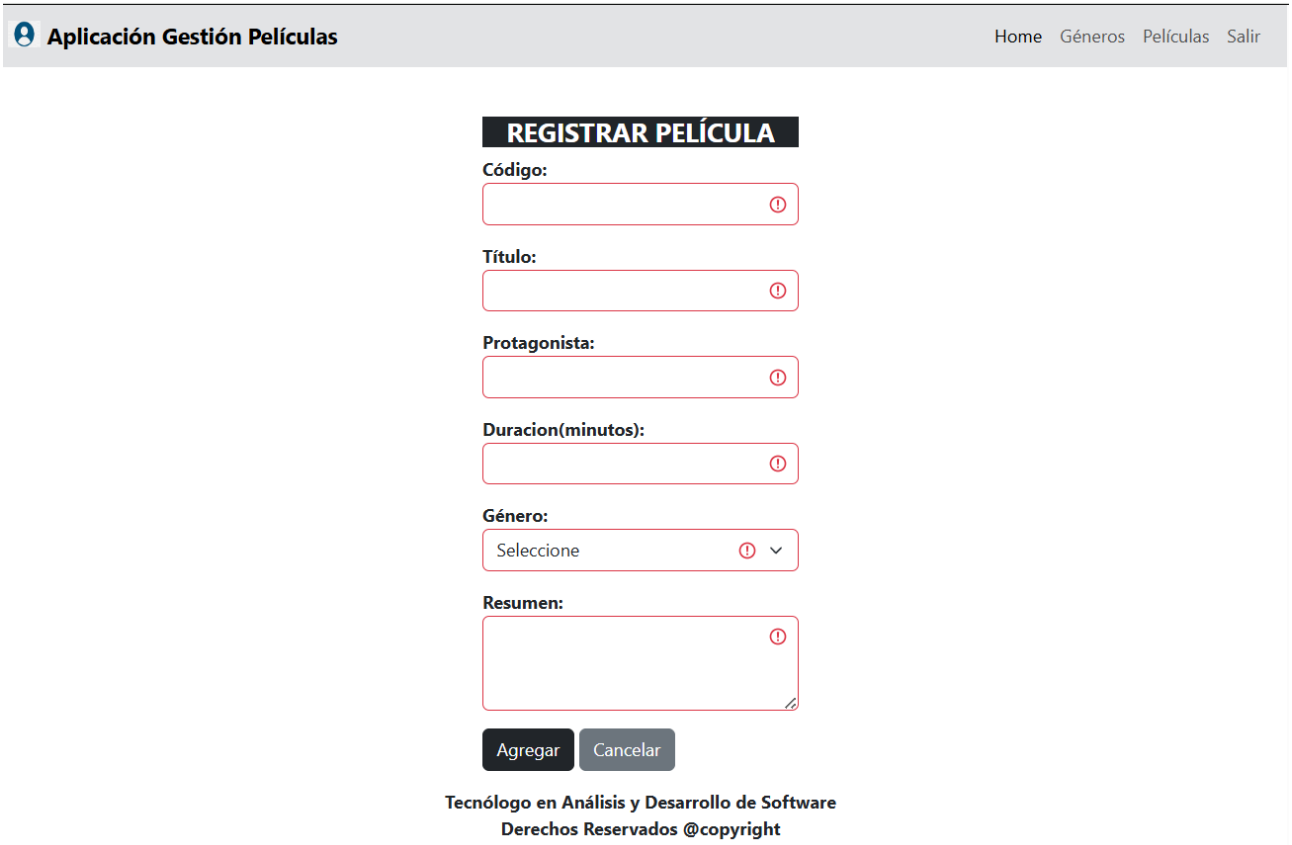
```
templates > <> listarPeliculas.html > ...
1    {% extends "index.html" %}
2    {% block menu %}
3        {% include "menu.html" %}
4    {%endblock %}
5    {% block contenido %}
6    >   <div class="w-75" style="margin: 0 auto">···
48       </div>
49   {%endblock %}
50   {% block piePagina %}
51       {% include "piePagina.html" %}
52   {%endblock %}
```

Ahora el código que está dentro del block contenido

```
{% block contenido %}
    <div class="w-75" style="margin: 0 auto">
        <h3 class="text-center fw-bold">LISTADO DE PELÍCULAS</h3>
        <div>
            <a href="/vistaAgregarPelicula/"><button class="btn btn-secondary">Agregar</button></a>
        </div>
        <DIV>
            <table id="tbPeliculas" class="table table-bordered mt-2" >
                <thead class="table-secondary text-center">
                    <tr>
                        <th>Código</th>
                        <th>Titulo</th>
                        <th>Duración</th>
                        <th>Protagonista</th>
                        <th>Género</th>
                        <th>Acción</th>
                    </tr>
                </thead>
```

```
                <tbody id="datosPeliculas">
                    {%for p in peliculas%}
                        <tr>
                            <td>{{p.codigo}}</td>
                            <td>{{p.titulo}}</td>
                            <td>{{p.duracion}}</td>
                            <td>{{p.protagonista}}</td>
                            <!--ontener nombre de genero a partir del id-->
                            {% for g in generos %}
                                {% if g.id == p.genero.id %}
                                    <td>{{g.nombre}}</td>
                                {%endif%}
                            {%endfor%}
                            <td class="text-center" style="font-size:3vh">
                                <i class="fa fa-edit text-warning" title="Editar"></i>
                                <i class="fa fa-trash text-danger" title="Eliminar"></i>
                            </td>
                        </tr>
                    {%endfor%}
                </tbody>
            </table>
        </DIV>
    </div>
{%endblock %}
```

23. Crear la funcionalidad de **agregar una película**. Se hace la siguiente propuesta de interfaz:



24. Se debe crear una ruta en el **backend** para que muestre el formulario. La ruta se crea en el archivo **película.py** de la carpeta **routes**.

```
Tabnine | Edit | Test | Explain | Document
@app.route("/vistaAgregarPelicula/", methods=['GET'])
def vistaAgregarPelicula():
    generos = Genero.objects()
    return render_template("frmAgregarPelicula.html", generos=generos)
```

Está ruta renderiza la vista **frmAgregarPelicula.html** y retorna una lista con los **géneros** existentes en la base de datos. Esa lista se retorna para que en el formulario en la parte del genero de la película lo seleccione de acuerdo a los ya exsitentes.

25. Código html del formulario

```
{% extends "index.html" %}
{% block menu %}
    {% include "menu.html" %}
{%endblock %}
{% block contenido %}
    <script src="../static/js/app.js"></script>
    <div class="w-25" style="margin: 0 auto">···
    </div>
{%endblock %}
{% block piePagina %}
    {% include "piePagina.html" %}
{%endblock %}
```

Y el código del **block contenido** es:

```
{% block contenido %}
    <script src="../static/js/app.js"></script>
    <div class="w-25" style="margin: 0 auto">
        <form id="frmPelicula" class="was-validated" enctype="multipart/form-data">
            <div>
                <h3 class="text-center fw-bold bg-dark text-white" style="font-size:2lvw">REGISTRAR PELÍCULA</h3>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="txtCodigo">Código:</label>
                <input type="number" name="txtCodigo" id="txtCodigo" class="form-control"  required>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="txtNombre">Título:</label>
                <input type="text" name="txtTitulo" id="txtTitulo" class="form-control"  required>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="txtProtagonista">Protagonista:</label>
                <input type="text" name="txtProtagonista" id="txtProtagonista" class="form-control"  required>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="txtDuracion">Duracion(minutos):</label>
                <input type="number" name="txtDuracion" id="txtDuracion" min="30" max="200" class="form-control"  required>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="cbGenero">Género:</label>
                <select name="cbGenero" id="cbGenero" class="form-select" required>
                    <option value="" selected>Seleccione</option>
                    {%for g in generos %}
                        <option value="{{g.id}}">{{g.nombre}}</option>
                    {%endfor%}
                </select>
            </div>
            <div class="mb-3">
                <label class="fw-bold" for="txtresumen">Resumen:</label>
                <textarea name="txtResumen" id="txtResumen" cols="30" rows="3" class="form-control" required></textarea>
            </div>
            <div>
                <button type="button" class="btn btn-dark" onclick="agregarPelicula()">Agregar</button>
                <a href="/peliculas/"><button type="button"  class="btn btn-secondary">Cancelar</button></a>
            </div>
        </form>
    </div>
{%endblock %}
```

26. Al dar clic en el **botón Agregar** llama a una función en **Javascript** que se encarga de hacer la petición al **backend** para agregar.

```
/**
 * Función que se encarga de hacer
 * una petición al backend para
 * agregar una película.
 */
Tabnine | Edit | Test | Explain | Document
function agregarPelicula(){
    url = "/pelicula/"
    const pelicula={
        codigo: document.getElementById('txtCodigo').value,
        titulo: document.getElementById('txtTitulo').value,
        protagonista: document.getElementById('txtProtagonista').value,
        duracion: document.getElementById('txtDuracion').value,
        resumen: document.getElementById('txtResumen').value,
        genero: document.getElementById('cbGenero').value,
        foto:''
    }
    fetch(url, {
        method: "POST",
        body: JSON.stringify(pelicula),
        headers: {
            "Content-Type": "application/json",
        }
    })
    .then(respuesta => respuesta.json())
    .then(resultado => {
        if (resultado.estado) {
           location.href = "/peliculas/"
        }else{
            swal.fire("Agregar Pelicula", resultado.mensaje, "warning")
        }
    })
        .catch(error => {
            console.error(error)
    })
}
```

27. Código Python del **backend** de la ruta **/película/** de tipo **post** que permite agregar una película.

```
Tabnine | Edit | Test | Explain | Document
@app.route("/pelicula/", methods=['POST'])
def addPelicula():
    try:
        mensaje=None
        estado=False
        if request.method=="POST":
            datos= request.get_json(force=True)
            genero= Genero.objects(id=datos['genero']).first()
            if genero is None:
                mensaje="Genero no existe no se puede crear la pelicula"
            else:
                datos['genero'] = genero
                pelicula = Pelicula(**datos)
                pelicula.save()
                estado=True
                mensaje="Pelicula agregada correctamente"
        else:
            mensaje="No permitido"
    except Exception as error:
        mensaje=str(error)

    return {"estado":estado, "mensaje":mensaje}
```

Como ejercicio se deben implementar las siguientes funcionalidades:

- **Editar y eliminar un genero**: Se deben hacer mediante peticiones usando fetch.
- **Editar y eliminar una película**: Se deben hacer mediante peticiones usando fetch.
- Agregar una colección **usuario** con los siguientes atributos:
    - **Usuario**: string, único
    - **Password**: string
    - **Nombre completo**: string
    - **Correo electrónico**: String de tipo correo electrónico
- Crear un módelo para esa colección
- Agregar manualmente unos dos usuarios.
- Crear un archivo **usuario.py** en la carpeta **routes**
- Implementar el inicio de sesión. Crear las rutas que requiera en el archivo **usuario.py** de la carpeta **routes**.
- No se puede realizar ninguna tarea sino han iniciado sesión.
- Publicar en repositorio Github.
- Debe ser sustentado el día miércoles 2 de abril en el ambiente de forma individual.

Fecha de Entrega: Julio 9 de 2025.

CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	César Marino Cuéllar Chacón	Instructor	CTPI-CAUCA	26-06-2025