Práctica 3.2: Visualización de productos: galería sin venta con base de datos.

Objetivo

Crear una galería de 10 películas (con nombre, sinopsis, duración, género y calificación) visible únicamente después del inicio de sesión. Se utilizará React para el frontend y Django para el backend, con una interfaz moderna, responsive y con animaciones.

- **Estructura de carpetas esperada**

```
/netflix-auth-project/
|
├── backend/
|   ├── manage.py
|   ├── backend/
|   |   ├── settings.py
|   ├── movies/              ← NUEVA APP
|   |   ├── models.py
|   |   ├── serializers.py
|   |   ├── views.py
|   |   ├── urls.py
|
├── frontend/
|   ├── src/
|   |   ├── Login.js
|   |   ├── MovieGallery.js   ← NUEVO COMPONENTE
|   |   ├── App.js
|   |   ├── App.css
|
├── .git/
```

PASOS

Paso 1: Crear tareas y dividir responsabilidades.

- **Backend:**
  - Crear modelo Movie con campos: title, synopsis, duration, genre, rating.
  - Crear 10 registros de películas en la base de datos.
  - Crear API para enviar las películas (solo si el usuario está autenticado).

- Asegurar las rutas con permisos (solo usuarios logueados).

- **Frontend:**
  - Reutilizar Login.js.
  - Crear componente MovieGallery.js que:
    - Obtiene las películas de la API.
    - Muestra la información en tarjetas modernas y responsivas.
    - Agrega animaciones y estilos con CSS.

- Desde la actividad 3.1 ya sabemos quién es el compañero con el que trabajaremos. Uno de los dos crea el repositorio con las ramas frontend y backend.

Paso 2: Verifica que ya funcione el login (React + Django).

- Este paso es de repaso. Antes de continuar, asegúrate de que el login ya funcione y recibas el token JWT del backend. Si no, revisa la práctica 3.1.

Paso 3: Backend - Crear app de películas en Django.

- **¿Qué es una app en Django?**
- Una app es un módulo independiente dentro de tu proyecto. En este caso, la app "movies" será la encargada de manejar la información de las películas, crearemos dentro de nuestra carpeta de backend.
- **Comando para crear app fuera de la carpeta de backend:**

```
python manage.py startapp movies
```

- Agrega "movies" al INSTALLED_APPS de settings.py.

Paso 4: Crear el modelo Movie.

- **¿Qué es un modelo?**
- Es una representación de una tabla en la base de datos. En este caso, Movie será una tabla con columnas como nombre, duración, etc. Buscamos el archivo models.py dentro de la carpeta movies

```python
# movies/models.py
from django.db import models

class Movie(models.Model):
    title = models.CharField(max_length=100)
    synopsis = models.TextField()
    duration = models.IntegerField(help_text="Duración en minutos")
    genre = models.CharField(max_length=50)
    rating = models.DecimalField(max_digits=3, decimal_places=1)

    def __str__(self):
        return self.title
```

- CharField es para texto corto.
- TextField es para texto largo (como sinopsis).
- IntegerField es para números enteros.
- DecimalField es para calificaciones como 7.5 o 9.2.

Paso 5: Migrar la base de datos de terminal de VS Code.

- Nos colocamos en la carpeta /netflix-auth-app y ejecutamos los comandos:

```
python manage.py makemigrations
python manage.py migrate
```

- Esto crea físicamente la tabla de películas.

Paso 6: Agregar 10 películas desde el admin de Django.

Habilita el acceso al panel de administración

- Primero, asegúrate de haber registrado el modelo en el admin.
- Abre el archivo: backend/movies/admin.py
- Agrega lo siguiente:

```python
from django.contrib import admin
from .models import Movie

admin.site.register(Movie)
```

- Esto le dice a Django que el modelo Movie estará disponible en el panel de administración.
- Abre este archivo: backend/backend/settings.py
- Busca la sección que dice INSTALLED_APPS y asegúrate de que incluya:

```
INSTALLED_APPS = [
    ...
    'movies',
    'rest_framework',  # ← si estás usando DRF
]
```

Paso 7: Crear el serializer.

- Convierte los objetos de Django en formato JSON para que puedan ser enviados al frontend. Creamos dentro de la carpeta de movies el archivo serializers.py y vamos a escribir el siguiente código:

```
# movies/serializers.py
from rest_framework import serializers
from .models import Movie

class MovieSerializer(serializers.ModelSerializer):
    class Meta:
        model = Movie
        fields = '__all__'
```

Paso 8: Crear vista y URL de la API.

- Aquí vamos a construir la capa intermedia que permite que el frontend (React) pueda solicitar la lista de películas al backend (Django). Esta capa se conoce como API (Interfaz de Programación de Aplicaciones). Usamos Django REST Framework porque facilita crear APIs modernas, seguras y fáciles de consumir.

- Crear la vista (APIView) en movies/views.py

```python
# movies/views.py
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import Movie
from .serializers import MovieSerializer

class MovieListView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        movies = Movie.objects.all()  # obtiene todas las películas
        serializer = MovieSerializer(movies, many=True)  # las convierte a JSON
        return Response(serializer.data)
```

- **MovieListView:** Esta es la vista que vamos a conectar con una URL.
- **permission_classes:** Solo deja pasar a los usuarios logueados.
- **get(self, request):** Este método se activa cuando alguien hace una petición GET (como abrir una página).
- **serializer = MovieSerializer(movies, many=True):** Toma todas las películas y las transforma en datos legibles por React.
- **return Response(...):** Devuelve esos datos a quien los pidió (en este caso, React).

Crear la URL que conecta esa vista con una ruta

- Dentro de la carpeta movies vamos a crear el archivo urls.py
- Archivo: movies/urls.py

```python
from django.urls import path
from .views import MovieListView

urlpatterns = [
    path('movies/', MovieListView.as_view()),  # localhost:8000/api/movies/
]
```

- Objetivo: decirle a Django que, cuando reciba una petición GET /api/movies/, debe ejecutar la lógica del MovieListView.

- Conectar esa ruta con el proyecto principal

- En backend/urls.py debes agregar:

```
from django.urls import path, include

urlpatterns = [
    path('api/', include('movies.urls')),  # Esto activa todas las rutas de la app movies
]
```

  - ➡ path('api/', include('movies.urls')): añade todas las rutas definidas en movies/urls.py bajo el prefijo /api/.

  - ➡ En consecuencia, GET /api/movies/ llama a MovieListView.

- Con esto, ya tienes una API protegida en http://localhost:8000/api/movies/, y solo funcionará si se envía un token válido (que se consigue al hacer login).

Paso 9: Frontend - Crear MovieGallery.js

- Vamos a crear un nuevo componente en React que:

  - ➡ Se conecta con la API de Django.

  - ➡ Muestra las películas en forma de tarjetas.

  - ➡ Aplica diseño moderno con CSS.

  - ➡ Solo funciona si el usuario ya está autenticado (tiene token JWT).

- Crear el archivo

- En frontend/src/, crea un archivo llamado MovieGallery.js.

```jsx
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import './App.css';

function MovieGallery({ token }) {
  const [movies, setMovies] = useState([]);

  useEffect(() => {
    axios.get('http://localhost:8000/api/movies/', {
      headers: {
        Authorization: `Bearer ${token}`
      }
    })
    .then(response => {
      setMovies(response.data); // guarda las películas en el estado
    })
    .catch(error => {
      console.error('Error al cargar películas:', error);
    });
  }, [token]);

  return (
    <div className="gallery">
      {movies.map(movie => (
        <div className="card" key={movie.id}>
          <h3>{movie.title}</h3>
          <p><strong>Género:</strong> {movie.genre}</p>
          <p><strong>Duración:</strong> {movie.duration} min</p>
          <p><strong>Calificación:</strong> {movie.rating}/10</p>
          <p>{movie.synopsis}</p>
        </div>
      ))}
    </div>
  );
}

export default MovieGallery;
```

- **useEffect:** Este hook se ejecuta una sola vez cuando se carga el componente.
- **axios.get:** Hace la petición GET a Django y le manda el token.
- **headers: { Authorization: 'Bearer ' + token }:** Aquí se manda el token JWT que prueba que el usuario está logueado.

- **setMovies:** Guarda las películas en un estado para poder mostrarlas.
- **movies.map(...):** Recorre el arreglo de películas y genera una tarjeta para cada una.

Paso 10: Agregar estilos con diseño moderno y animaciones.

- Abre el archivo App.css y agrega al final del código lo siguiente:

```css
.gallery {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 20px;
  padding: 20px;
  background-color: #121212;
  margin-top: 300px;
}

.card {
  background-color: #1f1f1f;
  padding: 20px;
  border-radius: 12px;
  color: #fff;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  box-shadow: 0px 2px 10px rgba(255, 255, 255, 0.1);
}

.card:hover {
  transform: scale(1.05);
  box-shadow: 0 0 20px #e50914;
}

h3 {
  color: #e50914;
}

@media (max-width: 600px) {
  .gallery {
    grid-template-columns: 1fr;
  }
}
```

- **display: grid:** Distribuye las tarjetas en columnas.
- **auto-fill:** Las acomoda automáticamente dependiendo del tamaño de pantalla.
- **@media:** Hace que se vea bien en celular (responsive).
- **transform + box-shadow:** Crea la animación cuando el usuario pasa el mouse.

Paso 11: Mostrar la galería solo si el usuario ha iniciado sesión.

- Edita el archivo App.js Para agregar los cambios realizados:
- Código completo actualizado con las modificaciones:

```
import React, { useState } from 'react';
import Login from './Login';
import MovieGallery from './MovieGallery';

function App() {
  const [token, setToken] = useState(null);

  return (
    <div className="App">
      {!token ? (
        <Login setToken={setToken} />
      ) : (
        <MovieGallery token={token} />
      )}
    </div>
  );
}

export default App;
```

- Modifica frontend/Login.js

```
// src/Login.js
import React, { useState } from 'react';
import axios from 'axios';

const Login = ({ setToken }) => {
  const [isRegistering, setIsRegistering] = useState(false);
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
```

```
  const handleSubmit = async (e) => {
    e.preventDefault();
    const url = isRegistering
      ? 'http://localhost:8000/api/register/' // endpoint para registro
      : 'http://localhost:8000/api/login/';   // endpoint para login

    try {
      const response = await axios.post(url, {
        username,
        password
      });

      if (isRegistering) {
        setMessage('Registro exitoso. Ahora puedes iniciar sesión.');
      } else {
        setMessage('Inicio de sesión exitoso.');
        setToken(response.data.access);

      }
    } catch (error) {
      setMessage('Hubo un error. Verifica tus datos.');
    }
  };

  return (
    <div>
      <h1 className="cecyflix-logo">CECYFLIX</h1>
      <h2>{isRegistering ? 'Crear cuenta' : 'Iniciar sesión'}</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Usuario"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Contraseña"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button type="submit">
          {isRegistering ? 'Registrarse' : 'Ingresar'}
        </button>
      </form>
      <p>{message}</p>
      <p>
```

```
        {isRegistering ? '¿Ya tienes cuenta?' : '¿No tienes cuenta?'}{' '}
        <span
          onClick={() => setIsRegistering(!isRegistering)}
          style={{ color: '#e50914', cursor: 'pointer' }}
        >
          {isRegistering ? 'Inicia sesión' : 'Regístrate'}
        </span>
      </p>
    </div>
  );
};

export default Login;
```

- Guardar y subir tus avances al repositorio
- Si eres backend y ya terminaste tu parte:

```
git add .
git commit -m "Terminé la integración de peliculas en Django"
git push origin backend
```

- Si eres frontend y terminaste tu parte:

```
git add .
git commit -m "Terminé el diseño del formulario y galería"
git push origin frontend
```

- Traer los cambios del compañero
- Esto lo debe hacer uno de los dos (de preferencia quien vaya a ejecutar toda la app).

➡ Cambia a la rama principal (`main`) o crea una nueva rama combinada:

```
git checkout main
```

➡ Trae lo más reciente de ambas ramas:

```
git pull origin backend
git pull origin frontend
```

- Esto no sobreescribe, pero es importante que resuelvas conflictos si aparecen (Git te dirá qué archivos están en conflicto y debes corregirlos antes de continuar).

- Después realiza un push sobre la rama main para subir los archivos actualizados a github.

Paso 12: Ejecutar el backend y Frontend.

- Abre una terminal en VS Code dentro de la carpeta netflix-auth-app:

```
cd backend
```

- Activa el entorno virtual (si no lo hiciste antes):

```
source venv/bin/activate    # En Linux o macOS
venv\Scripts\activate       # En Windows
```

- IMPORTANTE: Debemos estar en la carpeta principal "netflix-auth-app" e instalar django en la rama main y librerías que se instalaron por separado.
- Antes de ejecutar:
  1. Asegúrate de que tu entorno virtual esté activado ((venv) debe aparecer en la terminal). Ve a:
  2. Ejecuta este comando en la terminal:

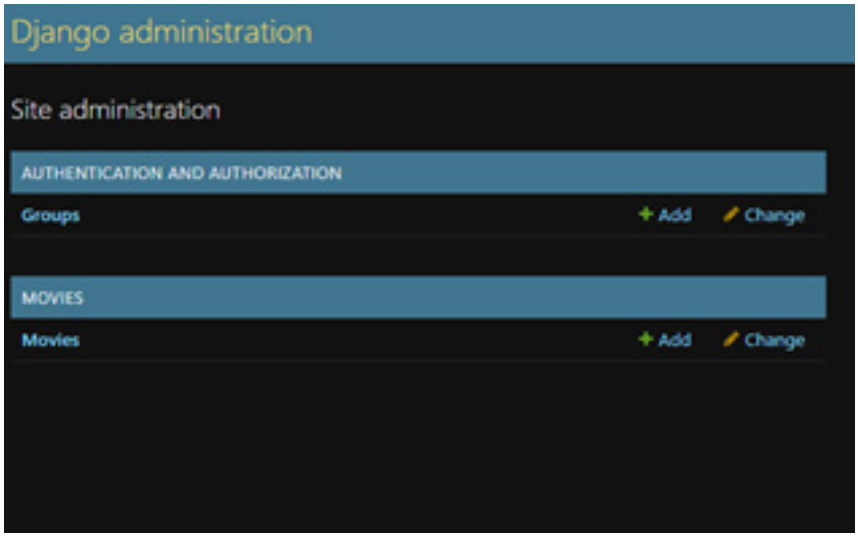
```
pip install djangorestframework-simplejwt
```

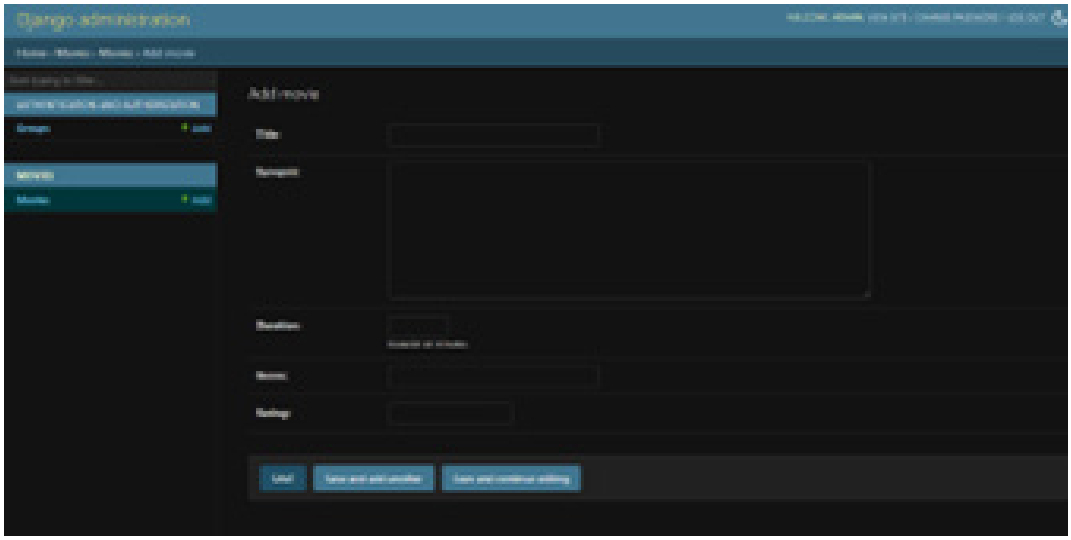
- Ejecuta el servidor de Django:

```
python manage.py runserver
```

- Ingresamos al panel para agregar las peliculas:

```
http://localhost:8000/admin
```





Paso 13: Ejecutar el frontend.

- Abre otra terminal nueva, y entra a la carpeta `frontend`:

```
cd frontend
```

- Inicia el servidor React:

```
npm start
```

NOTA: deben de existir 2 terminales de VS Code simultáneamente donde una se esté ejecutando django y la otra React.