

Informe de Django

Tema: Django

| Nota |
|------|
| |

| Estudiante | Escuela | Asignatura |
|--------------------------------------------------------------|--------------------------------------------------|--------------------------------------------------------|
| Victor Gonzalo Maldonado Vilca vmaldonadov@unsa.edu.pe | Escuela Profesional de Ingeniería de Sistemas | Programación Web 2 Semestre: III Código: 1702122 |

| Tarea | Tema | Duración |
|-------|--------|----------|
| 11 | Django | 2 horas |

| Semestre académico | Fecha de inicio | Fecha de entrega |
|--------------------|------------------------|------------------------|
| 2024 - A | Del 23 de mayo de 2024 | Al 15 de julio de 2024 |

1. Introducción

Django combina Django en el backend y Angular en el frontend para crear aplicaciones web modernas y escalables. Django maneja la lógica del servidor y la base de datos, mientras que Angular se encarga de la interfaz de usuario dinámica y interactiva en el navegador. Esta combinación ofrece una estructura poderosa y modular para desarrollar aplicaciones web de alto rendimiento.

2. Objetivos

- Comprender el funcionamiento de Django como BackEnd y Angular como FrontEnd.
- Construir aplicaciones escalables separando claramente las responsabilidades entre el backend y el frontend.
- Aprovechar las características de seguridad y rendimiento de Django en el servidor para garantizar aplicaciones robustas y rápidas.

3. Tarea

- Crear un proyecto que combine Angular con Django, repitiendo la experiencia de las clases teóricas y haciendo commit cada avance.
- Compartirlo con el docente (CarloCorrales010).

4. Entregables

- Informe hecho en Latex.
- URL: Repositorio GitHub.

5. Equipos, materiales y temas utilizados

- Django
- Angular
- RestFrameWork
- Api
- JSON

6. URL de Repositorio Github

- Link de Repositorio GitHub.
- <https://github.com/Victor-Gonzalo-Maldonado-Vilca/Djangular.git>

7. Desarrollo del trabajo

7.1. Capturas de la Actividad Realizada

7.1.1. Django

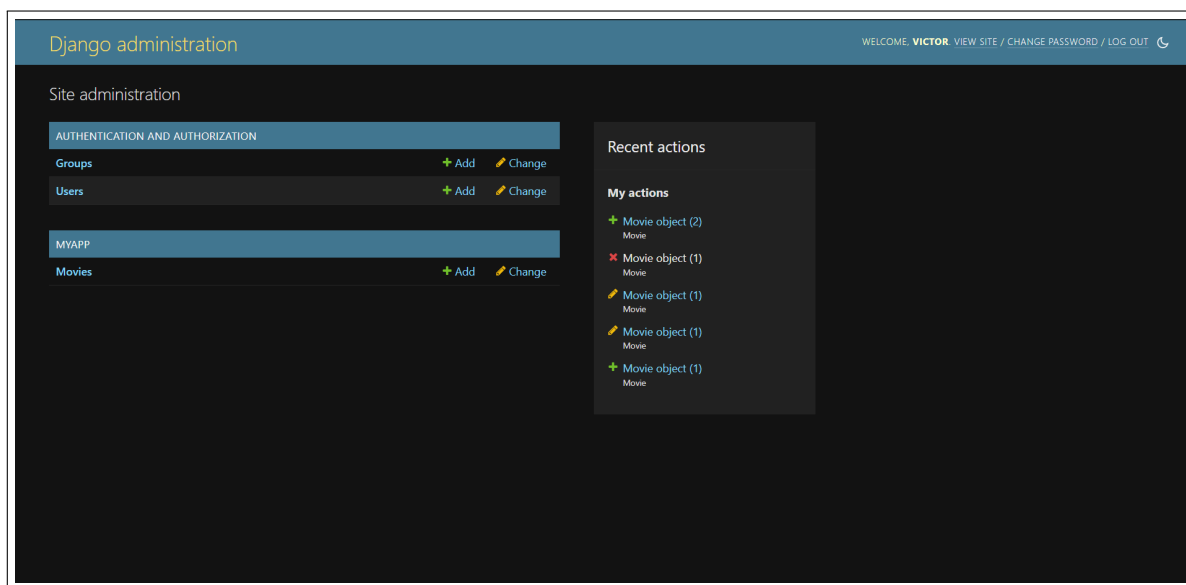
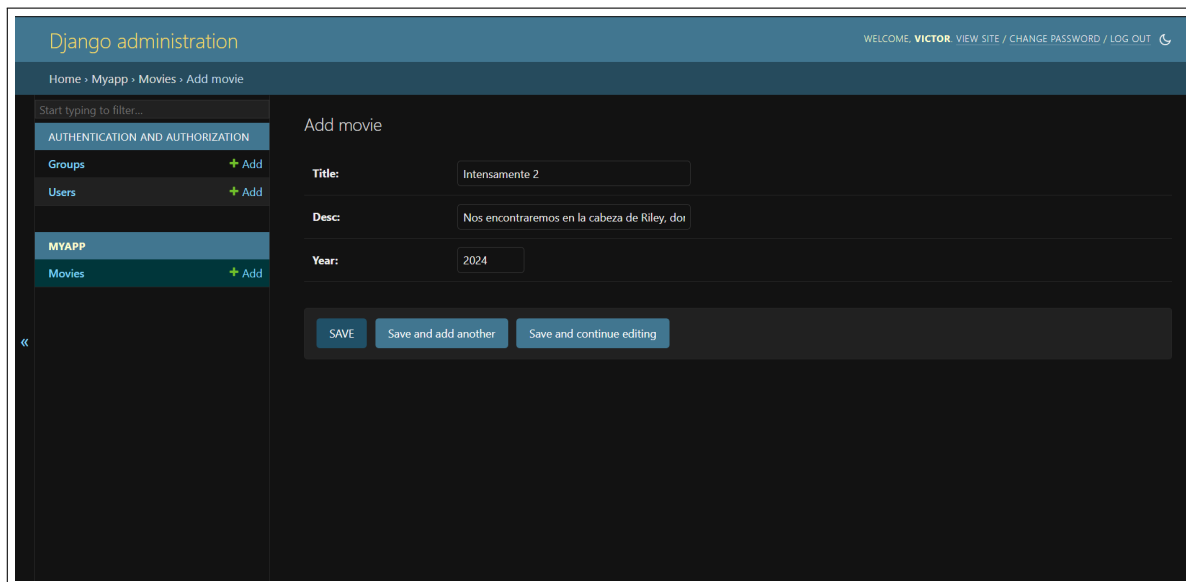
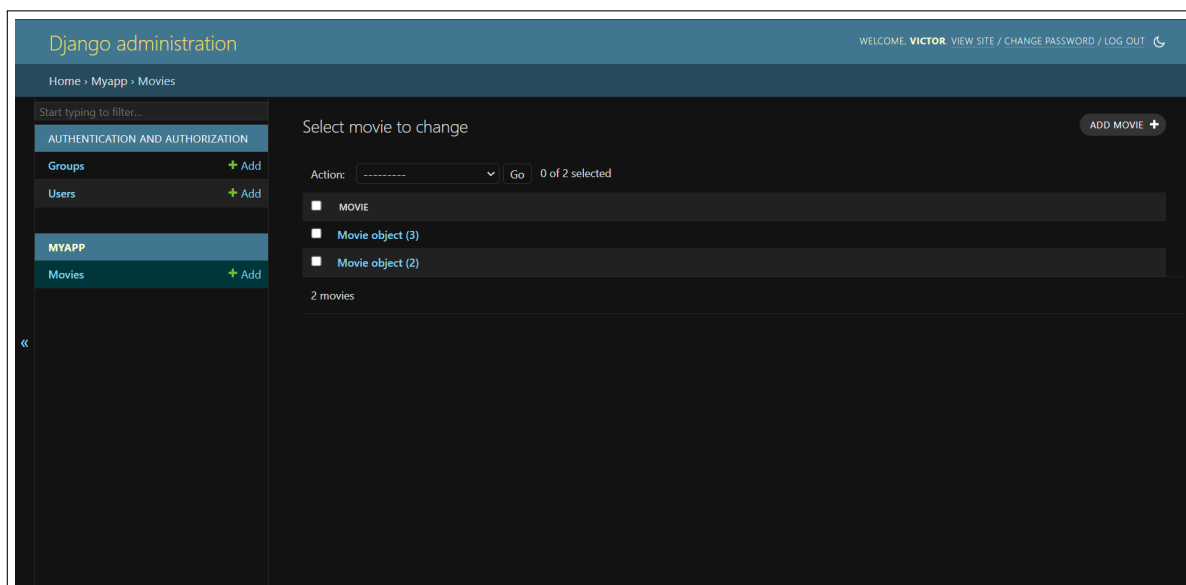


Figura 1: Admin Django



The screenshot shows the Django administration interface for adding a new movie. The breadcrumb trail is 'Home > Myapp > Movies > Add movie'. The left sidebar contains a search bar and two main sections: 'AUTHENTICATION AND AUTHORIZATION' with links for 'Groups' and 'Users' (each with a '+ Add' button), and 'MYAPP' with a link for 'Movies' (with a '+ Add' button). The main content area is titled 'Add movie' and contains three input fields: 'Title' with the value 'Intensamente 2', 'Desc' with the value 'Nos encontraremos en la cabeza de Riley, do', and 'Year' with the value '2024'. At the bottom of the form are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

Figura 2: Agregando un objeto Movie desde el Admin de Django



The screenshot shows the Django administration interface for selecting an existing movie to edit. The breadcrumb trail is 'Home > Myapp > Movies'. The left sidebar is identical to the previous screenshot. The main content area is titled 'Select movie to change' and includes an 'ADD MOVIE +' button in the top right. Below the title, there is an 'Action:' dropdown menu, a 'Go' button, and a status '0 of 2 selected'. A list of objects is displayed, each with a checkbox and a label: 'MOVIE', 'Movie object (3)', and 'Movie object (2)'. Below the list, it says '2 movies'.

Figura 3: Visualización de los objetos Movie Creados

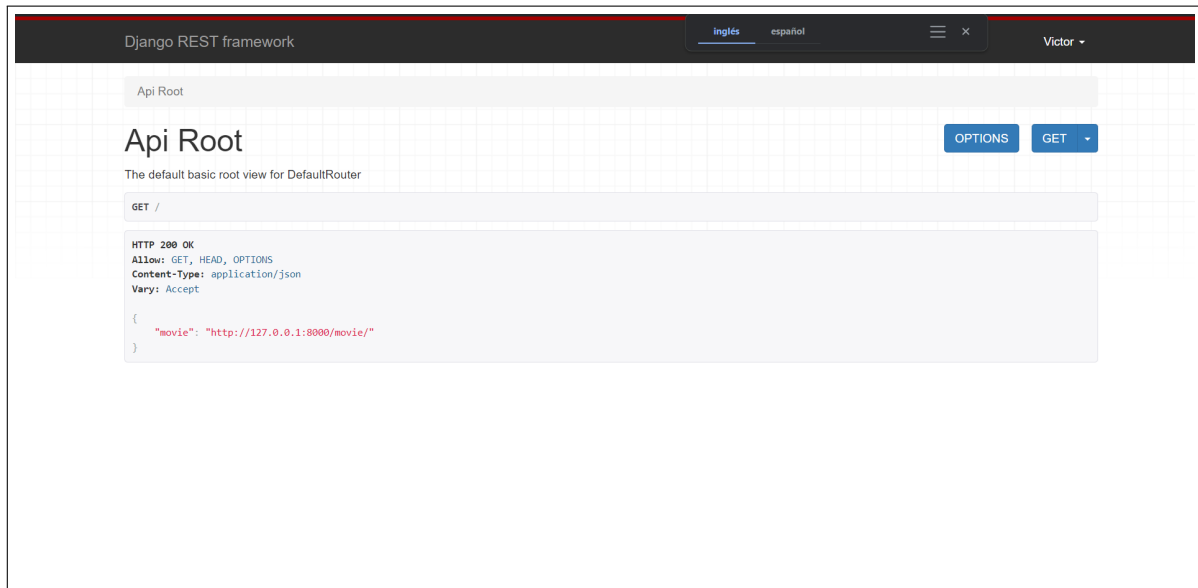


Figura 4: Api Root

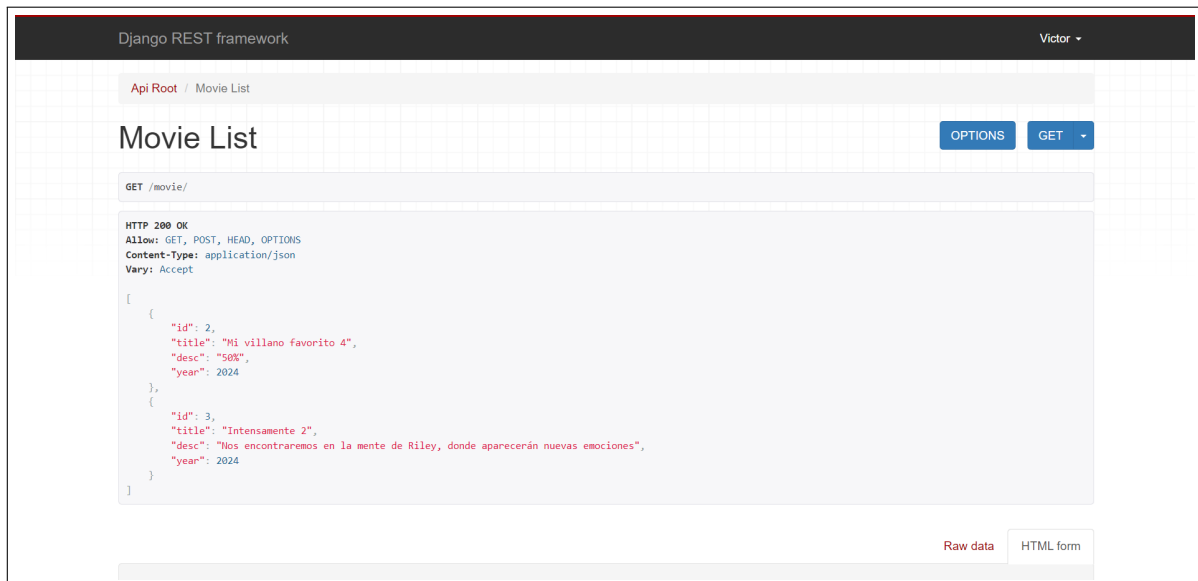


Figura 5: Api Movie List

7.1.2. Angular

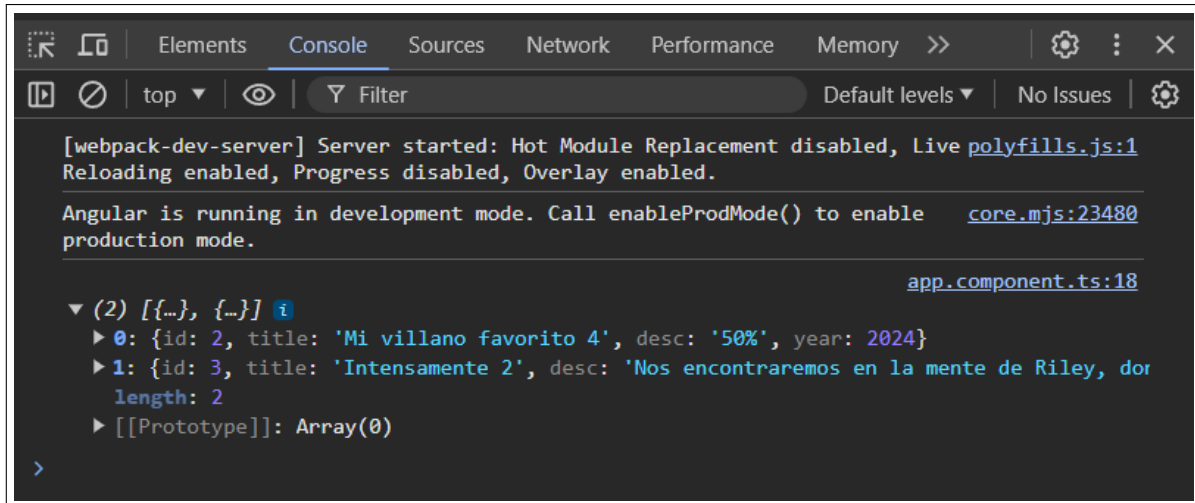


Figura 6: Consola – Obtener datos de la API

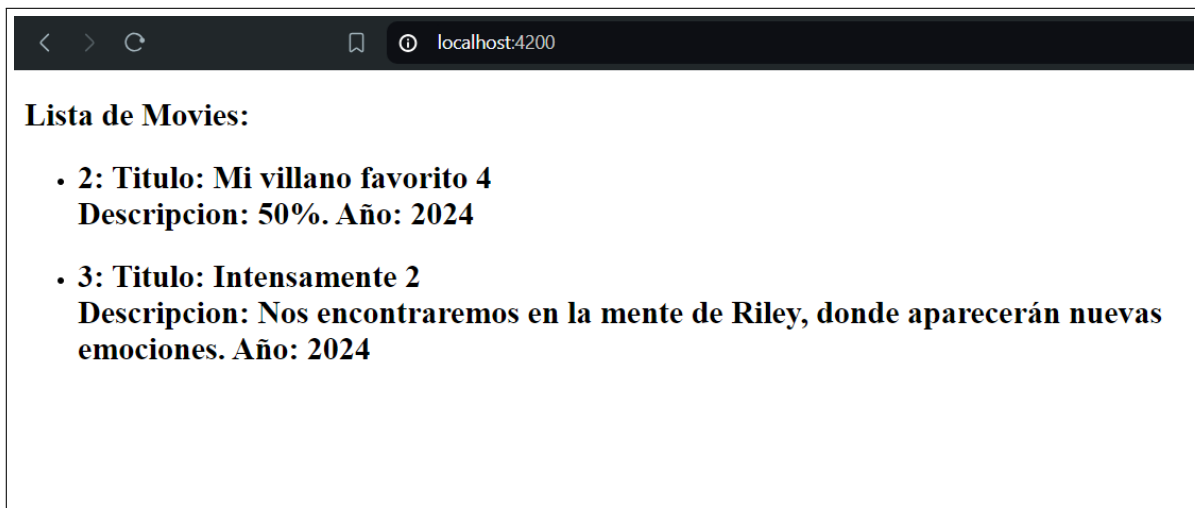


Figura 7: Ejecución – Angular

7.2. Django

7.2.1. Proyecto djangocrud 'settings.py'

- **INSTALLED_APPS:** Lista las aplicaciones instaladas en Django, incluyendo api, rest_framework, myapp, y corsheaders. api y myapp son aplicaciones personalizadas, rest_framework proporciona herramientas para construir APIs web, y corsheaders permite configurar políticas de CORS en la aplicación Django.

```
33 INSTALLED_APPS = [
34     ...
35     'api',
```

```
36     'rest_framework',
37     'myapp',
38     'corsheaders',
39 ]
```

- **MIDDLEWARE:** Incluye middleware como CorsMiddleware y CommonMiddleware. El CorsMiddleware facilita el manejo de solicitudes de recursos cruzados (CORS), mientras que CommonMiddleware proporciona funcionalidades comunes para el procesamiento de solicitudes HTTP.

```
46 MIDDLEWARE = [
47     ...
48     'corsheaders.middleware.CorsMiddleware',
49     'django.middleware.common.CommonMiddleware',
50 ]
```

- **CORS_ALLOWED_ORIGINS:** Define los orígenes permitidos para las solicitudes CORS, limitando las solicitudes a `http://localhost:4200`. Esto asegura que la aplicación Django pueda recibir solicitudes AJAX desde el servidor que corre en `localhost:4200`, típicamente utilizado para desarrollo frontend con frameworks como Angular.

```
131 CORS_ALLOWED_ORIGINS = [
132     "http://localhost:4200"
133 ]
```

7.2.2. Proyecto `django`crud 'urls.py'

- **Descripción:** Este código configura las rutas esenciales de la aplicación Django con Django REST Framework. Incluye un enrutador (`DefaultRouter`) para gestionar las URLs de la API, registra `MovieViewSet` para operaciones CRUD bajo la ruta `r'movie'`, y define rutas para el panel de administración (`admin/`), la API principal (`''`), y la autenticación de la API (`api-auth/`).

- **Código:**

```
1 from django.contrib import admin
2 from django.urls import path, include
3 from rest_framework import routers
4 from myapp import views
5 router = routers.DefaultRouter()
6 router.register(r'movie', views.MovieViewSet)
7 urlpatterns = [
8     path('admin/', admin.site.urls),
9     path('', include(router.urls)),
10    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
11 ]
```

7.2.3. Aplicación `myapp` - api 'models.py'

- **Descripción:** El modelo `Movie` en Django está diseñado para almacenar información básica sobre películas: El campo `title` es de tipo `CharField` y puede contener hasta 32 caracteres, representando el título de la película. El campo `desc` es también de tipo `CharField`, con una capacidad máxima de 256 caracteres, utilizado para almacenar una breve descripción de la película. El campo `year` es de tipo `IntegerField` y se utiliza para almacenar el año de lanzamiento de la película.

■ Código:

```
1 from django.db import models
2 # Create your models here.
3 class Movie(models.Model):
4     title = models.CharField(max_length=32)
5     desc = models.CharField(max_length=256)
6     year = models.IntegerField()
```

7.2.4. Aplicación myapp - api 'admin.py'

- **Descripción:** `admin.site.register(Movie)` en Django registra el modelo `Movie` en el panel de administración de Django, permitiendo a los administradores gestionar las películas directamente desde una interfaz de usuario administrativa.

■ Código:

```
1 from django.contrib import admin
2 from .models import Movie
3 # Register your models here.
4 admin.site.register(Movie)
```

7.2.5. Aplicación myapp - api 'serializer.py'

- **Descripción:** El `MovieSerializer` define cómo los objetos de la clase `Movie` deben ser serializados y deserializados. Utiliza el modelo `Movie` como base y especifica los campos `id`, `title`, `desc`, y `year` que serán incluidos en la representación serializada del objeto. Este serializer es esencial para manejar datos de películas de manera estructurada y eficiente dentro de una API Django, permitiendo la manipulación y visualización coherente de información relacionada con películas a través de solicitudes HTTP.

■ Código:

```
1 from django.contrib.auth.models import User, Group
2 from rest_framework import serializers
3 from .models import Movie
4
5 class MovieSerializer(serializers.ModelSerializer):
6     class Meta:
7         model = Movie
8         fields = ('id', 'title', 'desc', 'year')
```

7.2.6. Aplicación myapp - api 'views.py'

- **Descripción:** El `MovieViewSet` define un conjunto de vistas que manejan las operaciones CRUD para el modelo `Movie`. Utiliza el queryset `Movie.objects.all()` para obtener todas las instancias de `Movie` desde la base de datos. El `serializer_class` especifica que el serializador `MovieSerializer` se utilizará para convertir los objetos `Movie` en formatos como JSON, permitiendo así que los datos de las películas se manipulen de manera estructurada y consistente a través de solicitudes HTTP dentro de una API Django.

■ Código:

```
1 from django.shortcuts import render
2 from django.contrib.auth.models import User, Group
3 from rest_framework import viewsets
4 from .serializer import MovieSerializer
5 from .models import Movie
6
7 class MovieViewSet(viewsets.ModelViewSet):
8     queryset = Movie.objects.all()
9     serializer_class = MovieSerializer
```

7.3. Angular

7.3.1. Servicio api.service.ts

- **Descripción:** Después de importar los modulos necesarios, tenemos la clase ApiService. Esta clase se encarga de comunicarse con una API RESTful: La propiedad baseUrl establece la dirección base de la API en http://127.0.0.1:8000. httpHeaders define los encabezados HTTP necesarios para las solicitudes, con Content-Type configurado como application/json. El constructor de la clase inyecta el servicio HttpClient, permitiendo realizar solicitudes HTTP dentro de los métodos de la clase. El método getAllMovies() utiliza el servicio HttpClient para realizar una solicitud GET a la URL completa de la API concatenada con /movie/, incluyendo los encabezados definidos anteriormente.

■ Código:

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6     providedIn: 'root'
7 })
8 export class ApiService {
9
10     baseUrl = "http://127.0.0.1:8000";
11     httpHeaders = new HttpHeaders({'Content-Type': 'application/json'});
12     constructor(private http: HttpClient) { }
13     getAllMovies(): Observable<any>{
14         return this.http.get(this.baseUrl+'movie/',
15             {headers: this.httpHeaders});
16     }
17 }
```

7.3.2. Componente app.component.ts

- **Descripción:** Luego de hacer las importaciones correspondientes se define la clase AppComponent. La propiedad movies es un arreglo de objetos de películas inicializado localmente para demostración. El constructor inyecta el servicio ApiService y llama al método getMovies() al inicializar el componente. El método getMovies() utiliza el servicio ApiService para obtener películas desde la API. Los datos devueltos por getAllMovies() se registran en la consola.

■ Código:


```
10 export class AppComponent {
11   movies = [{id:1,title:'peli1',desc:"50%",year:2021},{id:2,title:'peli2',
12     desc:'50%',year:2022}];
13   constructor(private api:ApiService) {
14     this.getMovies();
15   }
16   getMovies = () => {
17     this.api.getAllMovies().subscribe (
18       data => {
19         console.log(data);
20         this.movies = data; //data.results;
21       },
22       error => {
23         console.log(error);
24       })
25   }
26 }
```

7.3.3. html app.component.html

- **Descripción:** Se muestra una lista de películas. Utiliza `router-outlet` como un marcador de posición para renderizar componentes según las rutas definidas en la aplicación Angular. A continuación, un encabezado (`<h2>`) dice "Lista de Movies:". Luego, se muestra una lista desordenada (``) que itera sobre el array `movies` utilizando `*ngFor`. Para cada película en el array, se crea un elemento de lista (``) que muestra el ID, título, descripción y año de la película dentro de un encabezado de segundo nivel (`<h2>`).

- **Código:**

```
1 <router-outlet></router-outlet>
2 <!--Mi lista aqui va -->
3 <h2>Lista de Movies:</h2>
4 <ul>
5   <li *ngFor="let movie of movies">
6     <h2> {{movie.id }}: Titulo: {{ movie.title }} <br>
7     Descripción: {{ movie.desc }}. year: {{ movie.year }}</h2>
8   </li>
9 </ul>
```

8. Conclusiones

- La integración de Django y Angular permite una aplicación web robusta que utiliza Django como backend y Angular como frontend.
- Django proporciona una API RESTful que Angular consume para mostrar datos dinámicamente.
- Las listas dinámicas en Angular, como la lista de películas, demuestran la capacidad de Angular para manejar y mostrar datos obtenidos de la API de Django.
- Esta combinación de tecnologías permite una separación clara entre el backend y el frontend, facilitando el desarrollo y mantenimiento de la aplicación.

8.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

| | Nivel | | | |
|--------|----------------------|-----------------|--------------------|---------------------|
| Puntos | Insatisfactorio 25 % | En Proceso 50 % | Satisfactorio 75 % | Sobresaliente 100 % |
| 2.0 | 0.5 | 1.0 | 1.5 | 2.0 |
| 4.0 | 1.0 | 2.0 | 3.0 | 4.0 |

Tabla 2: Rúbrica para contenido del Informe y demostración

| | Contenido y demostración | Puntos | Checklist | Estudiante | Profesor |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------|------------|----------|
| 1. GitHub | Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar. | 2 | X | 2 | |
| 2. Commits | Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación). | 4 | X | 4 | |
| 3. Código fuente | Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones. | 2 | X | 2 | |
| 4. Ejecución | Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente. | 2 | X | 2 | |
| 5. Pregunta | Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación). | 2 | X | 2 | |
| 6. Fechas | Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos. | 2 | X | 2 | |
| 7. Ortografía | El documento no muestra errores ortográficos. | 2 | X | 2 | |
| 8. Madurez | El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación). | 4 | X | 4 | |
| Total | | 20 | | 20 | |

9. Referencias

- <https://www.djangoproject.com/>
- <https://v17.angular.io/guide/architecture>