UNAH
UNIVERSIDAD NACIONAL
AUTÓNOMA DE HONDURAS

IS912 – Sistemas Expertos

II Periodo 2025

Proyecto II: Creación de un Pipeline de Datos & API con Caché Inteligente

Nº cuenta	Nombre
20181003916	David Alejandro García Ordoñez

Catedrático: Uayeb Caballero Rodriguez

18/07/25

Informe del Proyecto: Creación de un Pipeline de Datos y API con Caché Inteligente

Contenido

Enlaces:

- Portafolio: https://davidgarcia-portafolio.pages.dev/
- Github: https://github.com/DaroGard/movies.api
- Movies API: https://api-pr2-env.azurewebsites.net/docs

Introducción

Este proyecto aborda la construcción de un pipeline de datos y una API con integración a la nube, monitoreo, caché inteligente y despliegue automatizado.

A continuación, se detallan las fases y componentes principales de la solución implementada.

Fase 1: Infraestructura y Migración de Datos en Azure con Terraform

Se utilizó Terraform para automatizar la provisión de recursos en Azure.

- **main.tf:** Configura el provider de Azure y crea el grupo de recursos.
- **storage.tf:** Provisiona una cuenta de almacenamiento y un contenedor privado para almacenar archivos CSV.
- **db.tf:** Crea un servidor SQL Server y una base de datos para alojar los datos migrados.
- **df.tf:** Implementa Azure Data Factory para orquestar la migración ETL del dataset almacenado en Blob Storage hacia la base de datos SQL.
- **appinsight.tf:** Configura Application Insights para monitorear la API.
- **webapps.tf:** Despliega planes de servicio y aplicaciones web Linux con soporte para imágenes Docker.
- **acr.tf:** Implementa Azure Container Registry para almacenar imágenes Docker privadas.

Evidencias:

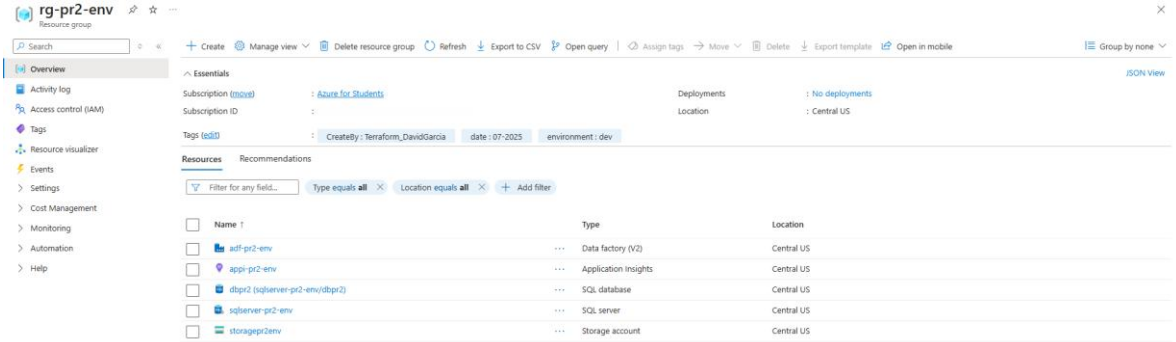
Terraform plan

```
terraform plan -var-file="local.tfvars"

      + network_rules (known after apply)

      + queue_properties (known after apply)

      + routing (known after apply)

      + share_properties (known after apply)

      + static_website (known after apply)
    }

  # azurerm_storage_container.csv will be created
  + resource "azurerm_storage_container" "csv" {
      + container_access_type            = "private"
      + default_encryption_scope         = (known after apply)
      + encryption_scope_override_enabled = true
      + has_immutability_policy          = (known after apply)
      + has_legal_hold                   = (known after apply)
      + id                               = (known after apply)
      + metadata                         = (known after apply)
      + name                             = "csv-content"
      + resource_manager_id              = (known after apply)
      + storage_account_id               = (known after apply)
    }

Plan: 7 to add, 0 to change, 0 to destroy.
```

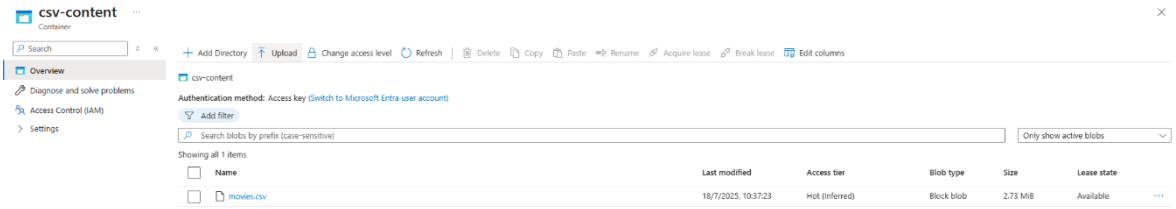
Terraform apply

```
terraform apply -var-file="local.tfvars"
azurerm_storage_account.storage: Creation complete after 1m9s [id=/subscriptions/3d7a4176-0edb-47e6-8b4d-756e5a78a211/resourceGroups/rg-pr2-env/providers/Microsoft.Storage/storageAccounts/storagepr2env]
azurerm_mssql_server.sqlserver: Still creating... [01m10s elapsed]
azurerm_storage_container.csv: Creating...
azurerm_storage_container.csv: Creation complete after 2s [id=/subscriptions/3d7a4176-0edb-47e6-8b4d-756e5a78a211/resourceGroups/rg-pr2-env/providers/Microsoft.Storage/storageAccounts/storagepr2env/blobServices/default/containers/csv-content]
azurerm_mssql_server.sqlserver: Still creating... [01m20s elapsed]
azurerm_mssql_server.sqlserver: Still creating... [01m30s elapsed]
azurerm_mssql_server.sqlserver: Still creating... [01m40s elapsed]
azurerm_mssql_server.sqlserver: Still creating... [01m50s elapsed]
azurerm_mssql_server.sqlserver: Creation complete after 1m55s [id=/subscriptions/3d7a4176-0edb-47e6-8b4d-756e5a78a211/resourceGroups/rg-pr2-env/providers/Microsoft.Sql/servers/sqlserver-pr2-env]
azurerm_mssql_database.moviesdb: Creating...
azurerm_mssql_database.moviesdb: Still creating... [00m10s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [00m20s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [00m30s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [00m40s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [00m50s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m00s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m10s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m20s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m30s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m40s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [01m50s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [02m00s elapsed]
azurerm_mssql_database.moviesdb: Still creating... [02m10s elapsed]
azurerm_mssql_database.moviesdb: Creation complete after 2m11s [id=/subscriptions/3d7a4176-0edb-47e6-8b4d-756e5a78a211/resourceGroups/rg-pr2-env/providers/Microsoft.Sql/servers/sqlserver-pr2-env/databases/dbpr2]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

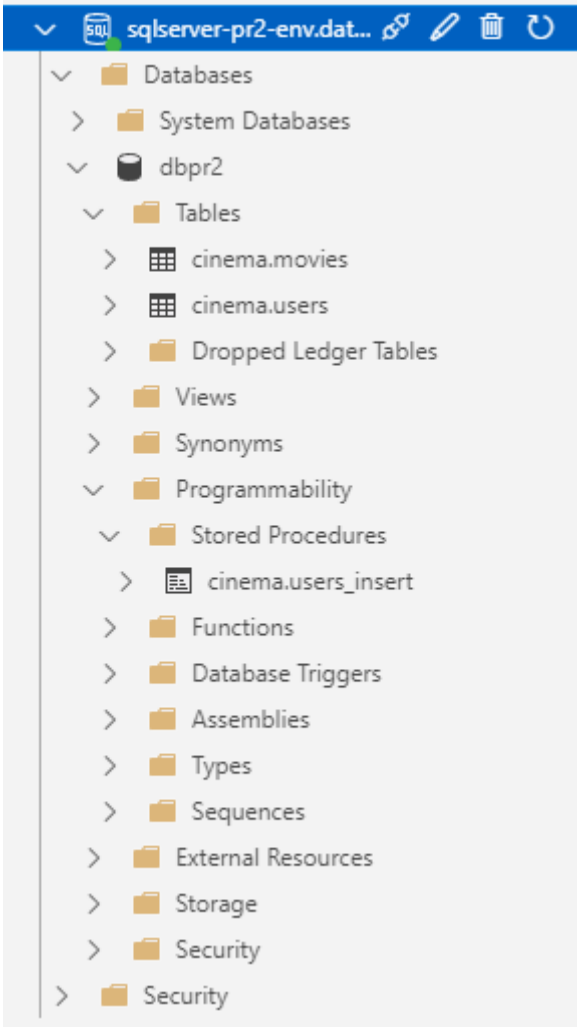
Azure Resource Group



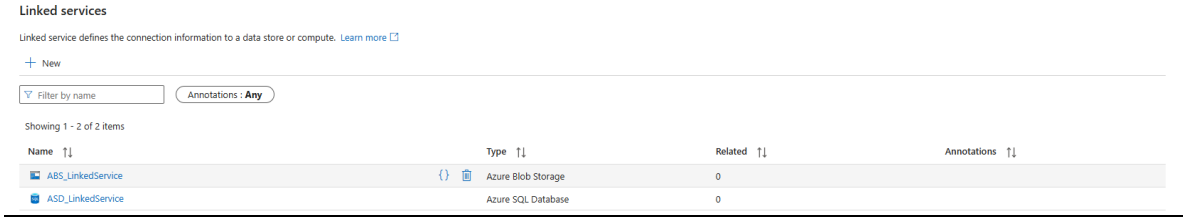
Almacenamiento



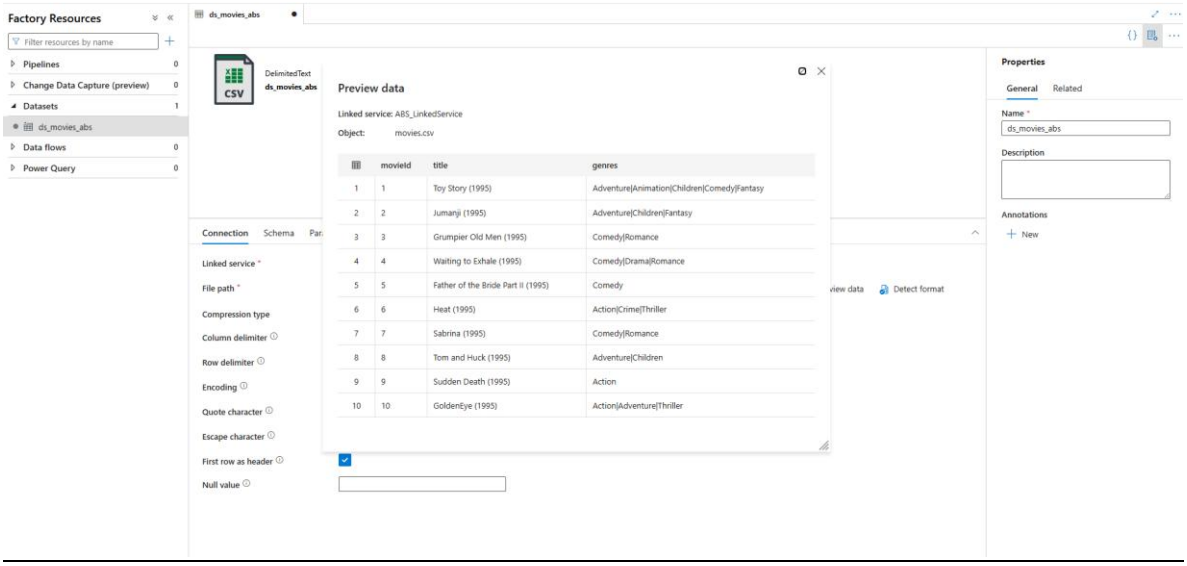
Base de datos

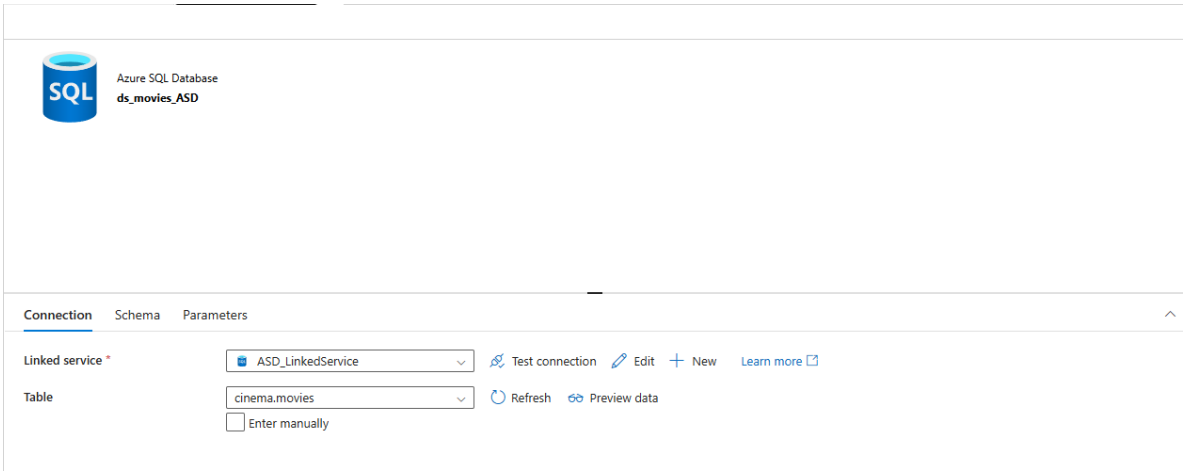


Linked Service

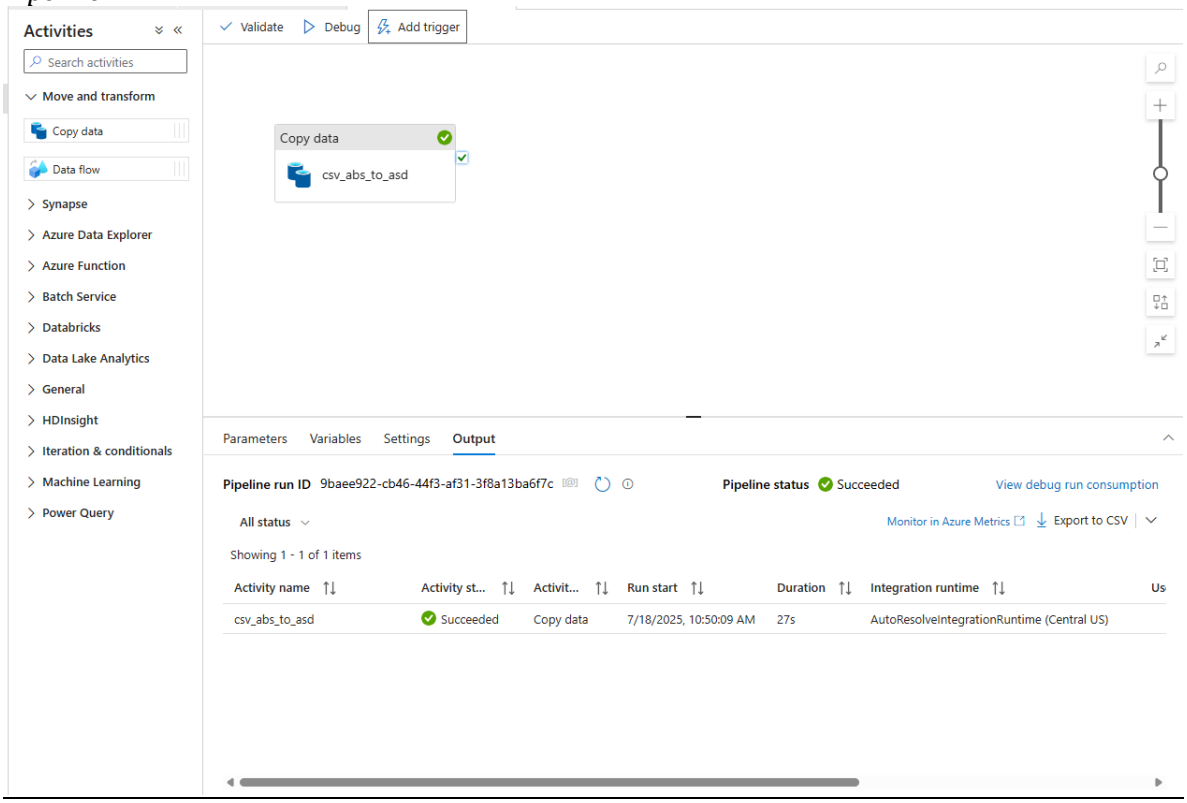


Datasets

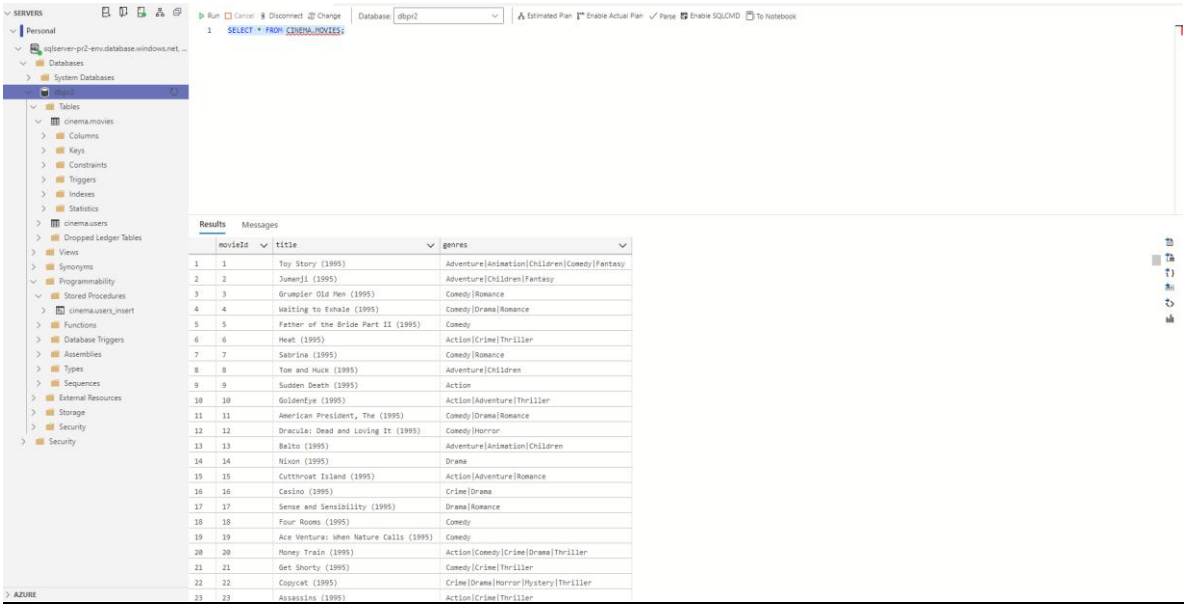




Pipeline



Datos



Fases 2 a 5: Desarrollo de API, Autenticación, Monitoreo y Caché Inteligente

Autenticación con Firebase (controllers/firebase.py):

- ✓ Registro y login de usuarios gestionados con Firebase Authentication.
- ✓ Usuarios almacenados también en la base de datos SQL con campos is_admin e is_active para permisos y estado.
- ✓ Emisión de tokens JWT personalizados para autorización interna.

Catálogo de películas (controllers/moviescatalog.py):

- ✓ Endpoint GET /catalog con soporte para filtros dinámicos (category), usando caché Redis para acelerar respuestas.
- ✓ Endpoint POST /catalog para agregar películas, protegido con validación de administrador, que invalida caché relacionada automáticamente tras inserción.

Modelos (carpeta models)

- ✓ Definición de modelos con validaciones estrictas para usuarios y películas, incluyendo validaciones de formato y complejidad de contraseñas.

Utilidades (carpeta utils)

- ➢ **database.py:** Manejo asíncrono de conexión a SQL Server con pyodbc, ejecución de queries con manejo de errores y conversión a JSON.
- ➢ **redis_cache.py:** Gestión robusta de conexión a Redis, obtención, almacenamiento y eliminación de caché con manejo de errores y logging.
- ➢ **security.py**: Creación y validación de tokens JWT, decoradores para autorización de usuarios normales y administradores.
- ➢ **telemetry.py:** Configuración y habilitación de Application Insights para telemetría completa de la API usando OpenTelemetry.

Esquema de base de datos (DDL.sql)

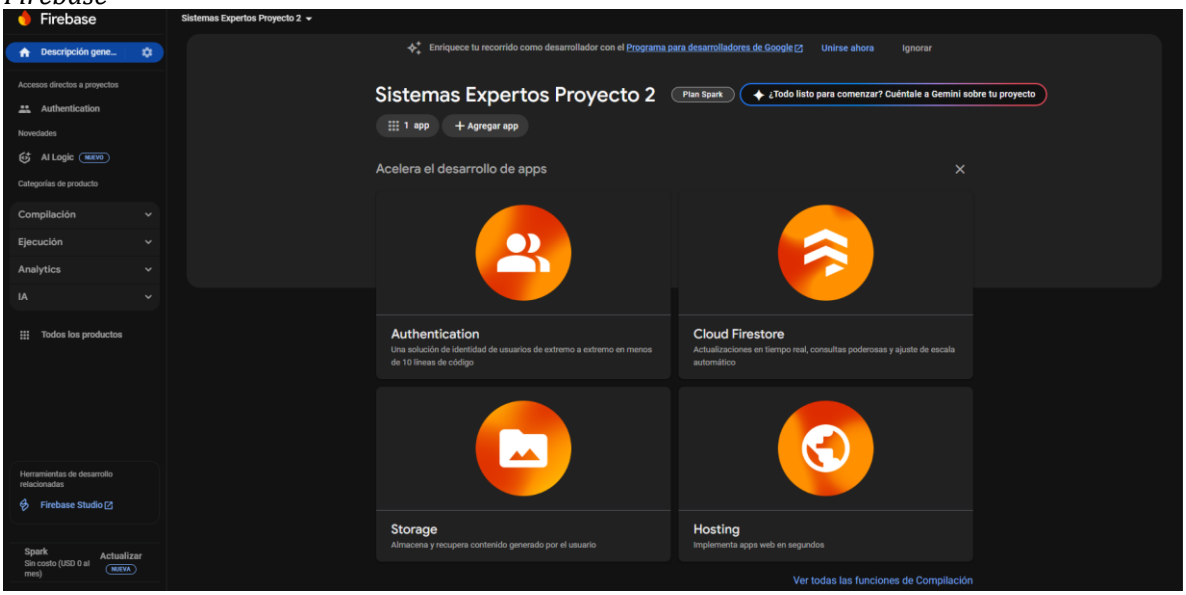
Definición de esquema y tablas para películas y usuarios, con procedimiento almacenado para inserción segura de usuarios.

API principal (main.py)

Inicialización de FastAPI con integración a telemetry y definición de endpoints.
Uso de dependencias para seguridad y autorización con JWT.

Evidencias:

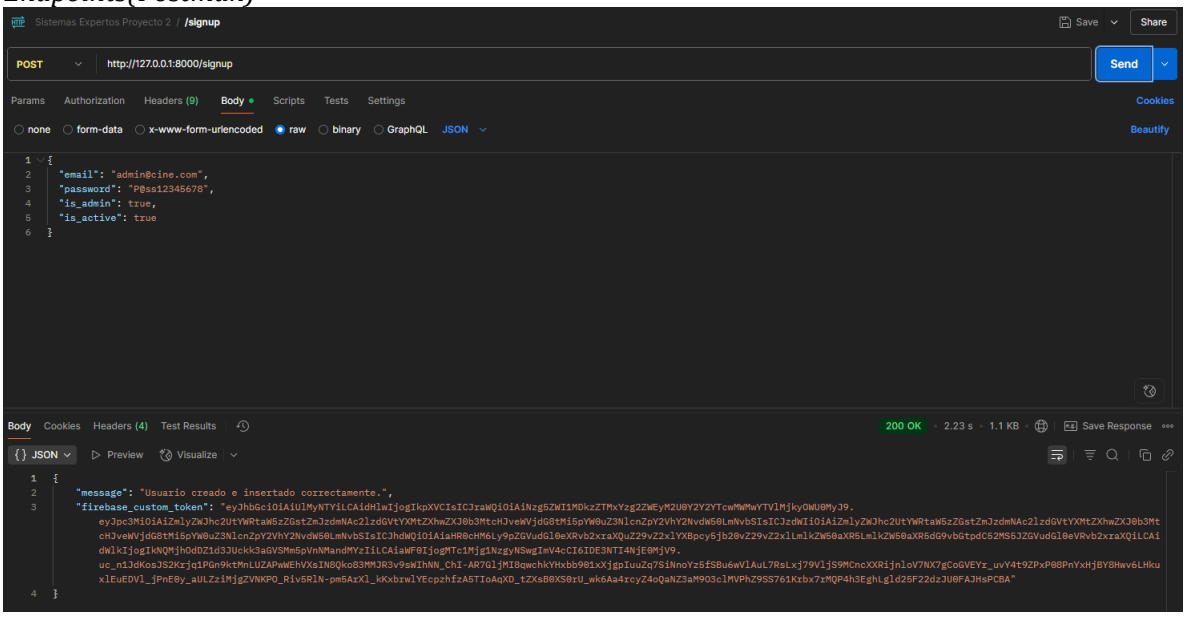
Firebase

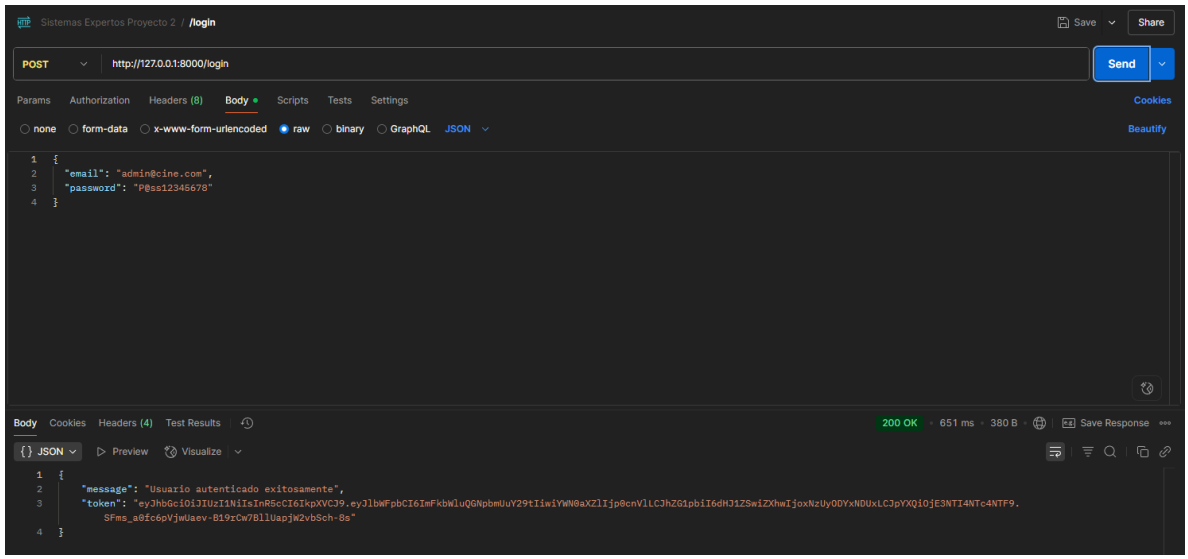


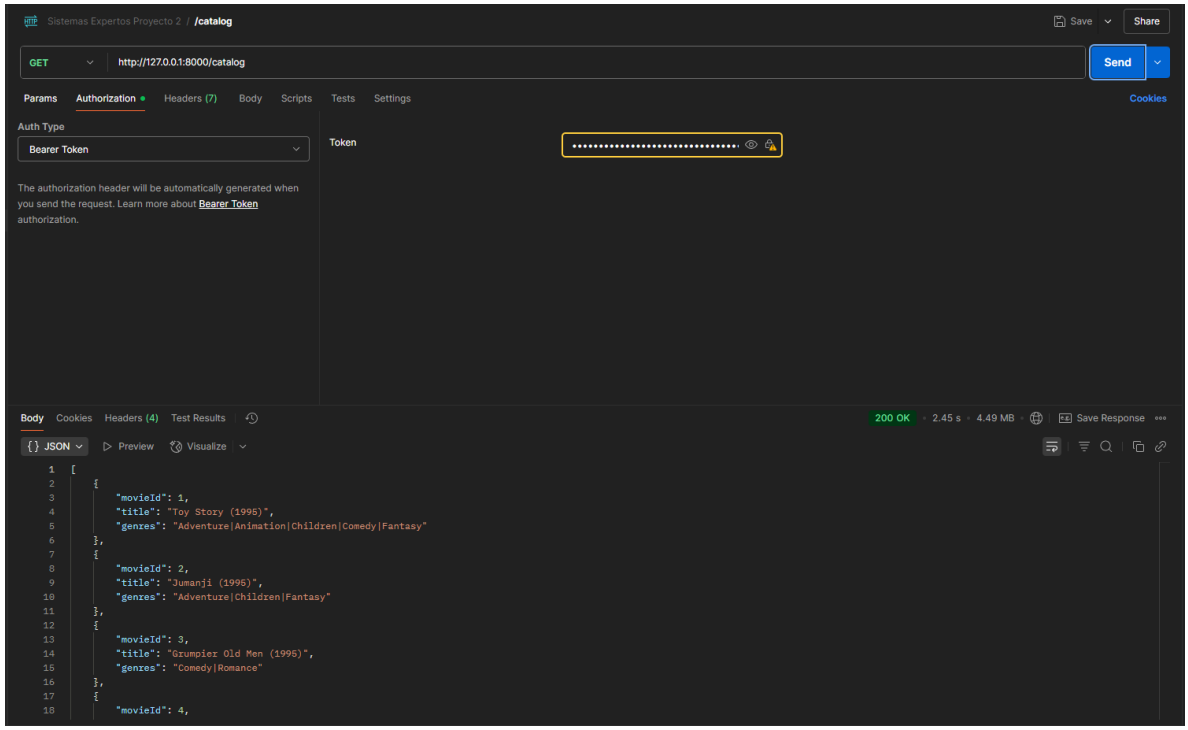
Uvicorn

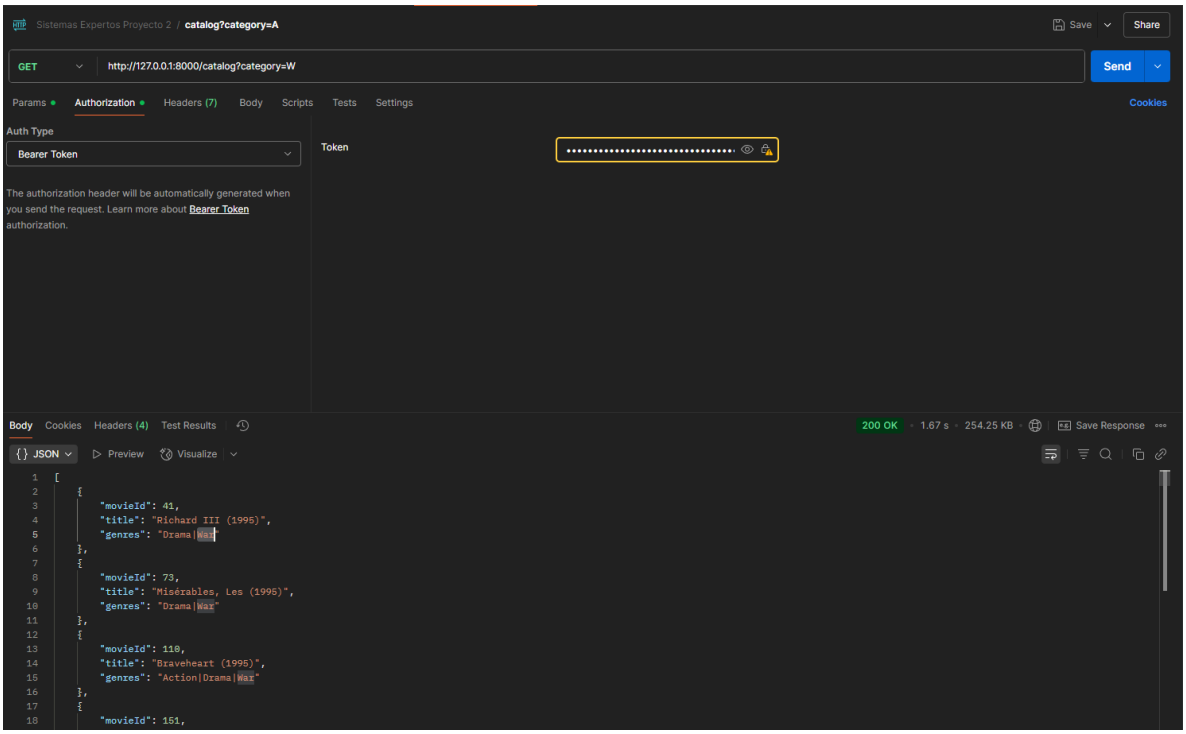
```
python -m uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['D:\\UNAH\\Clases Actuales\\Sistemas Expertos\\Proyecto\\2\\Proyecto\\api']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [19196] using StatReload
2025-07-18 10:55:01,754 - controllers.firebase - INFO - Firebase inicializado correctamente.
2025-07-18 10:55:03,028 - azure.core.pipeline.policies.http_logging_policy - INFO - Request URL: 'https://westus-0.in.applicationinsights.azure.com//v2.1/track'
```

Endpoints(Postman)

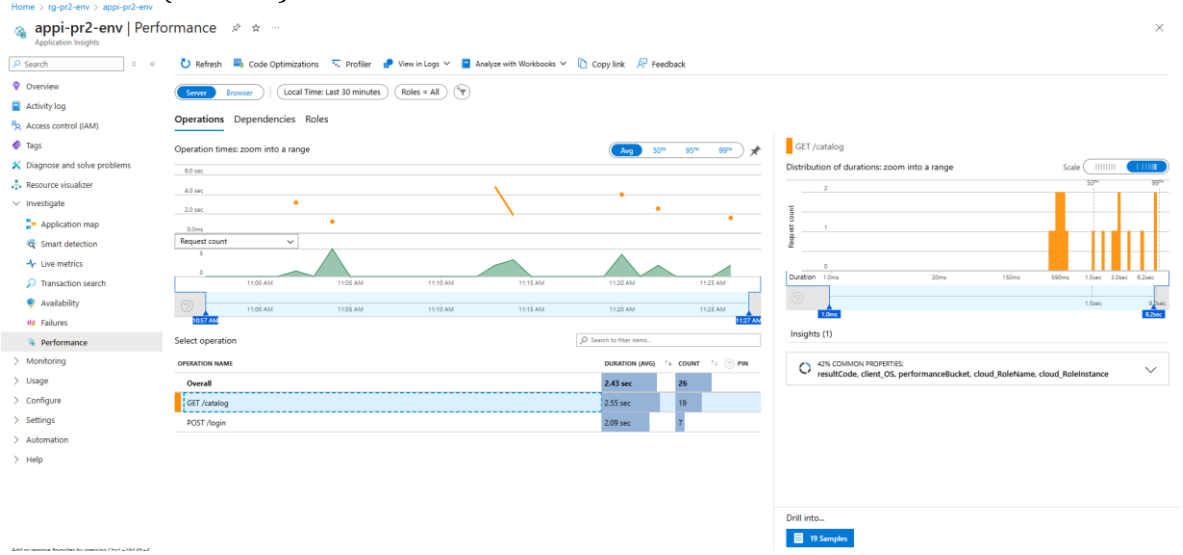


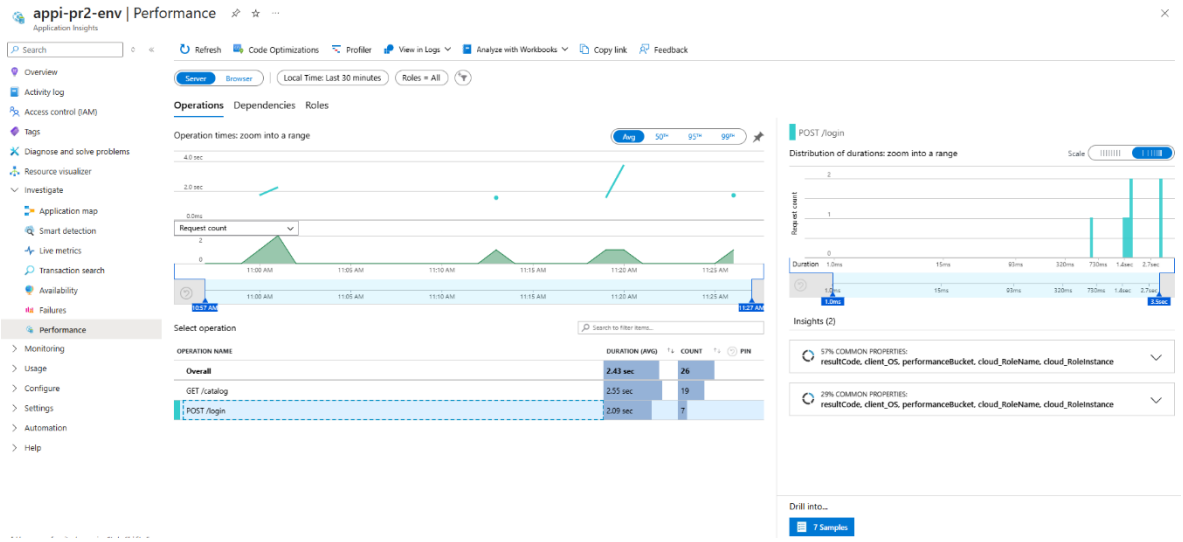

Sistemas Expertos Proyecto 2 / /login
Save
Share
POST
http://127.0.0.1:8000/login
Send
Params Authorization Headers (8) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"email": "admin@cine.com",
"password": "P@ss12345678"
}
Body Cookies Headers (4) Test Results
200 OK 651 ms 380 B
Save Response
{} JSON Preview Visualize
{
"message": "Usuario autenticado exitosamente",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFkbWluQGNpbmUuY29tIiwiYWN0aXZlIjp0cnVlLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzUyODYxNDUxLCJpYXQiOjE3NTI4NTc4NTF9.
SFms_a0fc6pVjwUaev-B19rCw7BllUapjW2vbSch-8s"
}


Sistemas Expertos Proyecto 2 / /catalog
Save
Share
GET
http://127.0.0.1:8000/catalog
Send
Params Authorization Headers (7) Body Scripts Tests Settings
Cookies
Auth Type
Bearer Token
Token
The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.
Body Cookies Headers (4) Test Results
200 OK 2.45 s 4.49 MB
Save Response
{} JSON Preview Visualize
[
{
"movieId": 1,
"title": "Toy Story (1995)",
"genres": "Adventure|Animation|Children|Comedy|Fantasy"
},
{
"movieId": 2,
"title": "Jumanji (1995)",
"genres": "Adventure|Children|Fantasy"
},
{
"movieId": 3,
"title": "Grumpier Old Men (1995)",
"genres": "Comedy|Romance"
},
{
"movieId": 4,


Sistemas Expertos Proyecto 2 / catalog?category=A
Save
Share
GET
http://127.0.0.1:8000/catalog?category=W
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Auth Type
Bearer Token
Token
The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.
Body
Cookies
Headers (4)
Test Results
200 OK
1.67 s
254.25 KB
Save Response
JSON
Preview
Visualize
[
{
"movieId": 41,
"title": "Richard III (1995)",
"genres": "Drama|War"
},
{
"movieId": 73,
"title": "Misérables, Les (1995)",
"genres": "Drama|War"
},
{
"movieId": 110,
"title": "Braveheart (1995)",
"genres": "Action|Drama|War"
},
{
"movieId": 151,

Rendimiento (sin Redis)



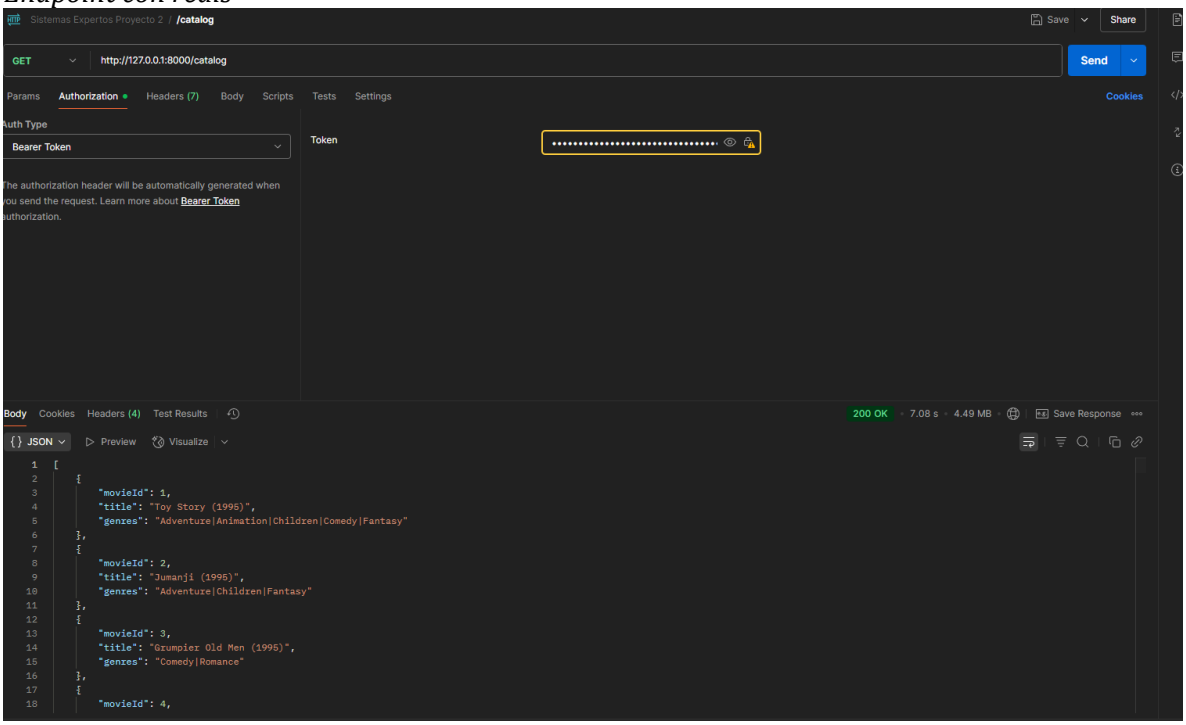


Redis

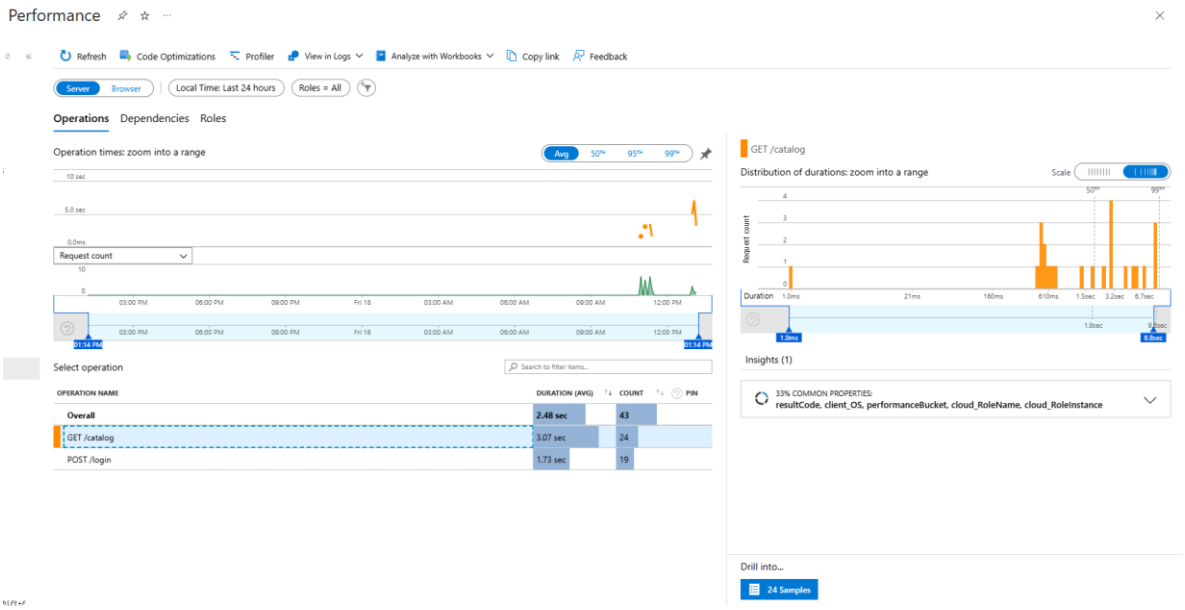
```
2025-07-18 13:07:35,990 - utils.redis_cache - INFO - ✅ Conectado exitosamente a Redis usando Connection String
2025-07-18 13:07:36,077 - controllers.moviescatalog - INFO - Consultando todo el catálogo de películas.
2025-07-18 13:07:36,077 - utils.database - INFO - Intentando conectar a la base de datos...
2025-07-18 13:07:36,658 - azure.core.pipeline.policies.http_logging_policy - INFO - Request URL: 'https://centralus-2.in.applicationinsights.az
track'
```

```
2025-07-18 13:07:41,646 - utils.redis_cache - INFO - 📦 Datos almacenados en caché con clave 'movies:catalog:all' por 1800 segundos
2025-07-18 13:07:41,650 - controllers.moviescatalog - INFO - Películas almacenadas en caché con clave: movies:catalog:all
2025-07-18 13:07:41,681 - azure.core.pipeline.policies.http_logging_policy - INFO - Request URL: 'https://centralus-2.in.applicationinsights.azure.com//v2.1/
```

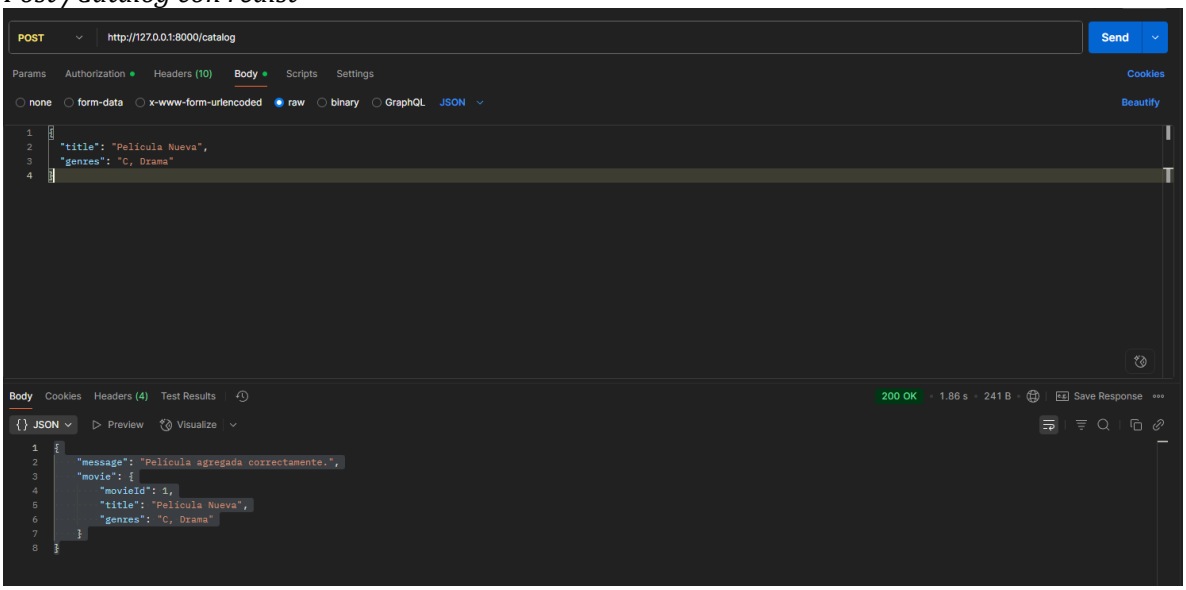
Endpoint con redis



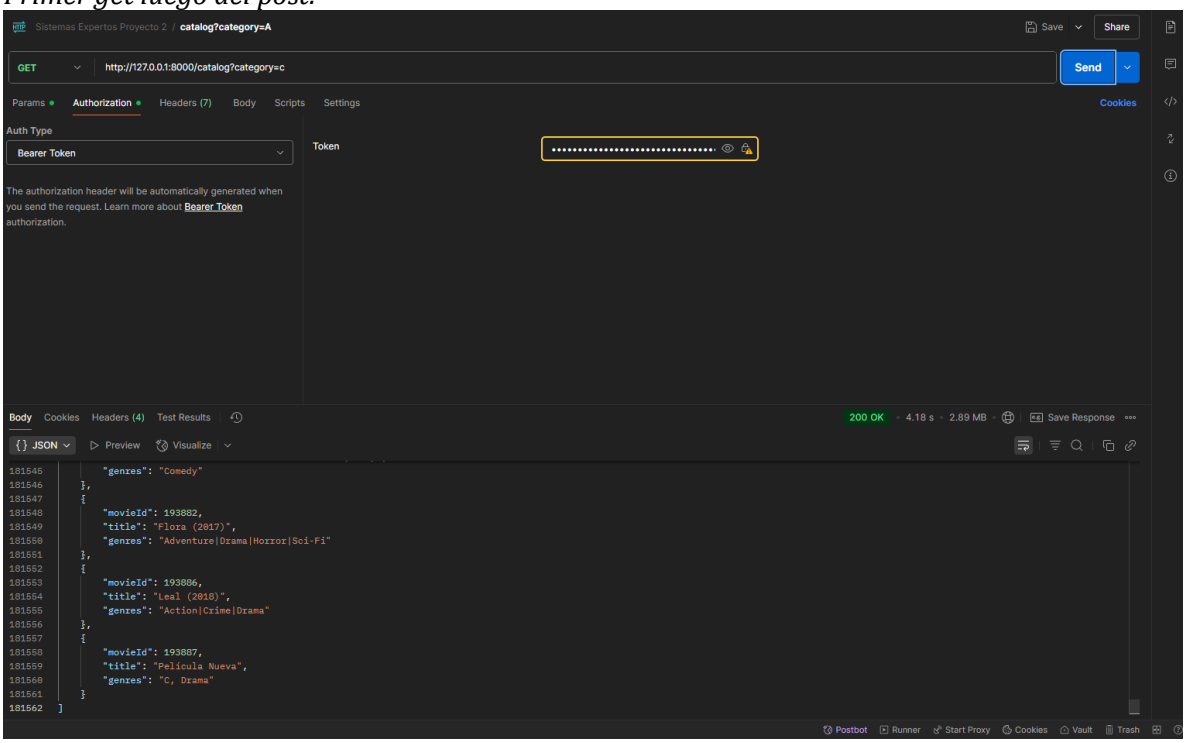
Rendimiento con redis



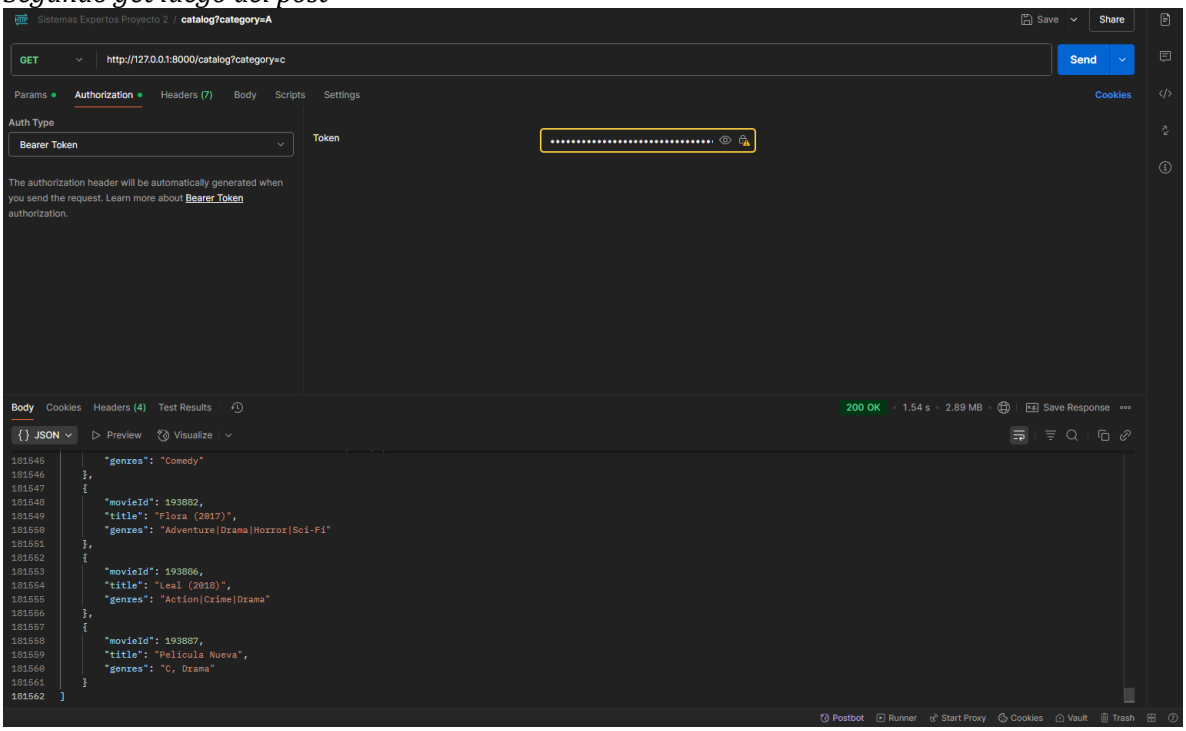
Post /Catalog con redist



Primer get luego del post:



Segundo get luego del post



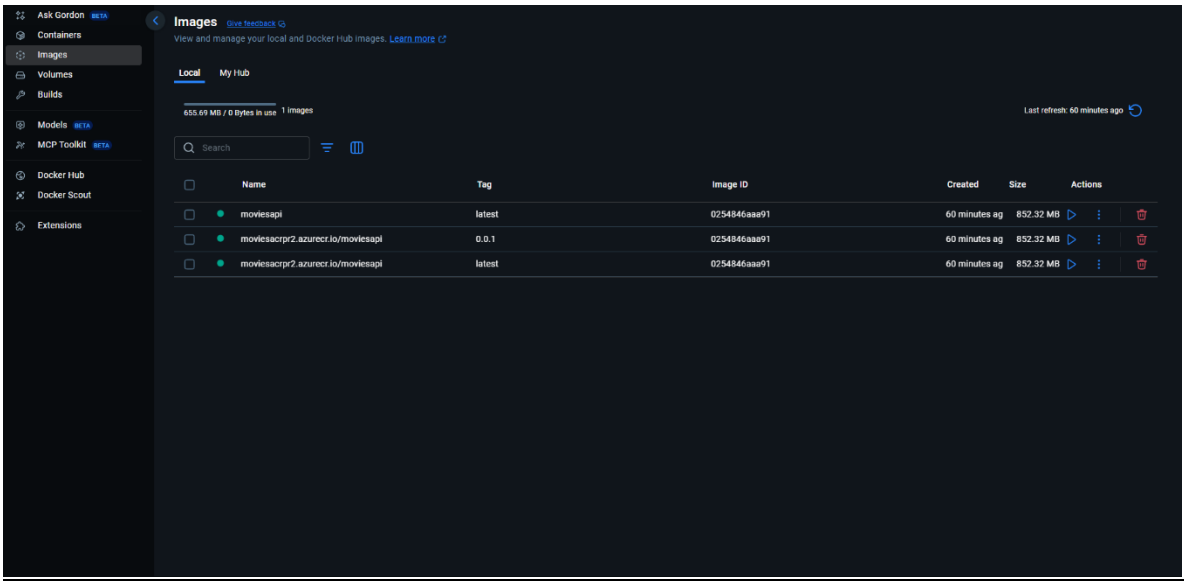
Fase 6: Despliegue con Docker y Azure

Se creó una imagen Docker para la API, almacenada en Azure Container Registry (ACR).

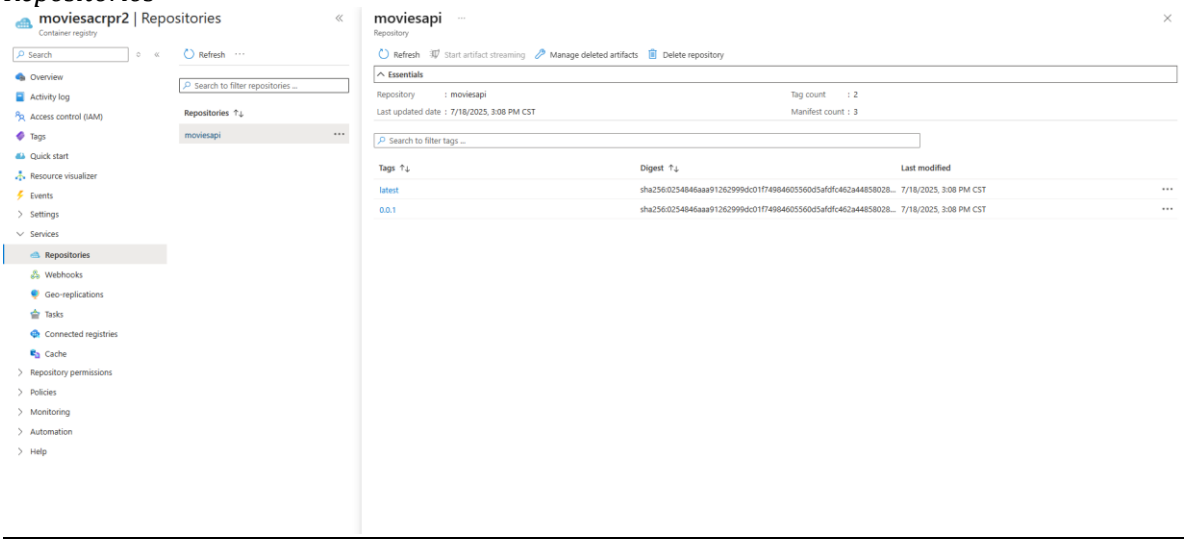
La API y UI se desplegaron en Azure Web Apps Linux con planes de servicio configurados, integrando las imágenes Docker desde ACR.

Evidencias

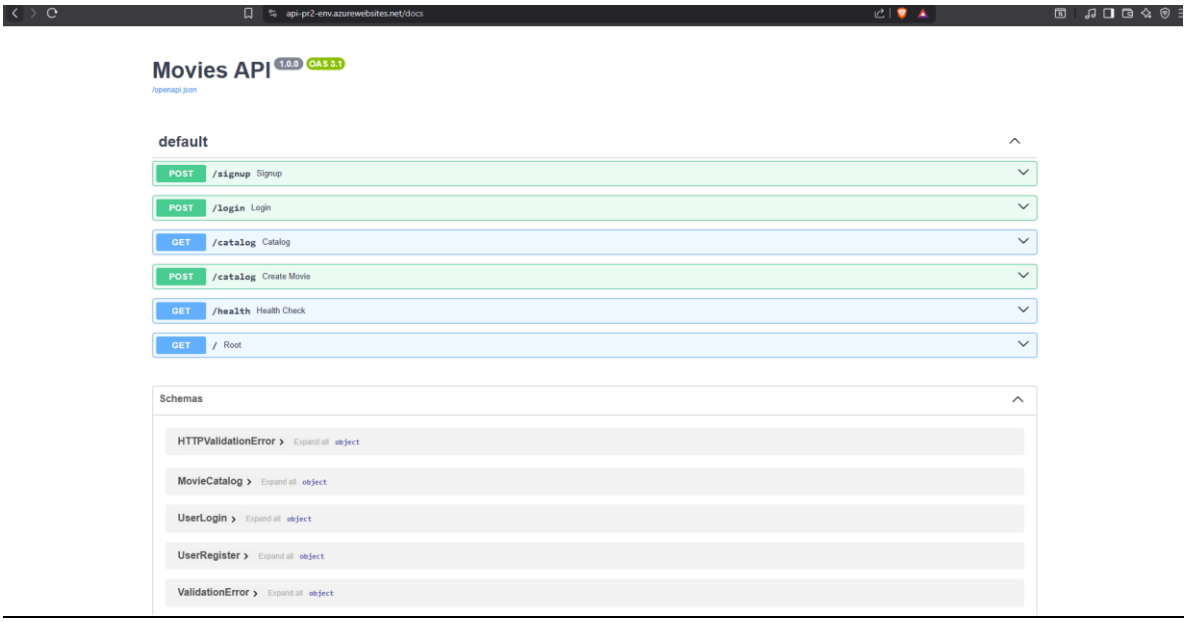
Docker



Repositorios



Lanzamiento



Conclusión

Este proyecto fue una experiencia clave para integrar y aplicar conocimientos sobre desarrollo backend y servicios en la nube, a través de herramientas como Azure Data Factory, FastAPI, Firebase, Redis, Terraform y Docker, se construyó una solución completa; desde la migración de datos hasta el despliegue en un entorno productivo.

Más allá de lo técnico, el trabajo permitió comprender cómo cada tecnología contribuye al rendimiento, seguridad y escalabilidad del sistema, como también, reforzó la importancia de la automatización y el monitoreo en tiempo real para mantener servicios confiables.

En definitiva, este proyecto no solo fortaleció mis habilidades prácticas, sino que me ofreció una visión clara de lo que implica diseñar y operar soluciones modernas en la vida profesional.