Sistemas de Información

Práctica 3: Concepción, desarrollo y puesta en obra de un sistema de información web

Jose Secadura Del Olmo

Chloé Bolle-Reddat

Sergio Saura Oliva

Facultad de Ingeniería y Arquitectura. Campus de Zaragoza.

2025

Índice

Diseño del modelo de datos

Modelo E/R

Primero mencionar que todas las imágenes estarán subidas en el repositorio de GitHub por si no se vieran bien en el documento.

Para el modelo E/R se han creado un total de 8 tablas necesarias para la creación del programa. Estas tablas son: usuarios, comentario, ranking, item_ranking, película, lista, encuesta, actividades y api_key.

Las tablas que más confusión pueden dar son la de actividades, item_ranking o api_key.

La tabla actividades se utiliza para saber los gustos del usuario, se utiliza para conocer que tipos de género visita más o que actor y poder darle recomendaciones para eso.

Para la tabla item_ranking es para poder relacionar la película con el ranking, ya que puede haber varios rankings y la película puede aparecer en varios de estos.

Por último la tabla api_key como se puede apreciar en la imagen no está relacionada ya que es una tabla aparte que sirve para guardar las API Keys de las APIs a la que se van a acceder para mostrar los datos al usuario.

Link Anexo Imagen E/R

Modelo lógico relacional

Para el modelo lógico se ha utilizado la herramienta dbdiagram, aquí se han creado 3 tablas más para relacionar ciertas tablas:

- Tabla lista_peliculas: Esta tabla sirve para relacionar una película con varias listas ya que una misma peli puede estar en varias listas del usuario.
- Tabla usuario_listas: Sirve para relacionar el usuario con sus listas, un mismo usuario puede tener muchas listas o incluso compartir listas con otros usuarios, de ahí que sea necesario la creación de esta tabla.
- Tabla usuario_seguidor: Los usuarios pueden seguirse entre ellos, para ello es necesario una tabla que guarde los identificadores de los usuarios que se siguen para poder relacionarlos.

Link Anexo Imagen Modelo Relacional

Diseño de la capa de persistencia de datos

Se ha implementado una biblioteca de clases en .NET en la que se han introducido el contexto de la BD, se ha comprobado la conexión a esta a través de Azure Data Studio. El entorno que se utilizó fue Visual Studio.

Para conectarse a la base de datos con .NET se ha instalado un paquete para poder gestionar la conexión introduciéndole la “Connection String” en la clase ServiceExtension.cs.

Se ha creado una clase para cada modelo DAO para poder realizar las diferentes configuraciones para cada tabla que hay en la base de datos y comunicarse con esta. Aquí un ejemplo de la clase Usuarios:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Fylt.Infrastructure.DAOs
{
    public class Usuario
    {
        public int Id { get; set; }
        public string? RealName { get; set; }
        public string Username { get; set; } = null!;
        public string? Descripcion { get; set; }
        public int Seguidores { get; set; } = 0;
        public int Seguidos { get; set; } = 0;
        public string? Foto { get; set; }
        public string Password { get; set; } = null!;
        public bool BoolAdmin { get; set; } = false;

        // Relaciones
        public List<Encuesta>? EncuestasCreadas { get; set; }
        public Actividades? Actividades { get; set; }
        public List<UsuarioLista>? UsuarioListas { get; set; }
        public List<UsuarioSeguidor>? SeguidoresUsuarios { get; set; }
    }
}
```

Se han creado las diferentes clases VO para mapear los datos que obtienen desde las tablas de la base de datos y poder así gestionarlos. Aquí un ejemplo para la creación de usuarios:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Fylt.Domain.VOs.UsuariosVOs
{
    public class CreateUsuarioVO
    {
        public string? RealName { get; set; }
        public string Username { get; set; } = null!;
        public string? Descripcion { get; set; }
        public int Seguidores { get; set; } = 0;
```

```
        public int Seguidos { get; set; } = 0;
        public string? Foto { get; set; }
        public string Password { get; set; }
        public bool BoolAdmin { get; set; } = false;
    }
}
```

Se han realizado pruebas CRUD para probar la conexión a la base de datos, para ello se ha creado una API que hace llamadas a los servicios que sirven para obtener datos de la base de datos. Aquí un vínculo a los anexos donde se muestran las capturas:

Capturas pruebas

Metodologías

Para el diseño de la base de datos se ha utilizado draw.io para la creación del diagrama de entidad relación ya que es una herramienta muy sencilla para la elaboración de estos diagramas.

Para el diseño del modelo relacional se ha utilizado la herramienta dbdiagram. Esta permite con un código bastante simple crear tablas y relacionarlas, incluso tiene la opción de hacer el ddl de las tablas con PostgreSQL a raíz del modelo relacional. Cosa que se ha utilizado, aunque hay varias líneas que se han modificado a nuestro gusto para que sea más legible el script y se creen las tablas como se quería.

Para la base de datos se utiliza el motor de PostgreSQL, aunque integrado dentro de neon.com. En un principio se iba a utilizar Supabase pero había problemas para obtener los datos ya que Supabase actuaba como una API para conectarse a la base de datos y Neon permite una conexión directa a la base de datos, cosa que facilita mucho para la implementación del programa y las dos ofrecen 0.5GB gratuitos de almacenamiento.

El lenguaje y el entorno que se ha decidido para implementar el proyecto es .NET con Visual Studio en lugar de Java porque facilitaba la conexión a la base de datos con PostgreSQL a través de bibliotecas que proporcionaba Visual Studio. También el grupo ya tenía una experiencia previa con C#. También se utilizo la aplicación Azure Data Studio para visualizar la base de datos y probar su correcta conexión.

La distribución del trabajo ha sido tal que Sergio ha liderado el grupo y ha estado ayudando en todos los entornos de este trabajo. José junto con Sergio se han encargado de la creación, búsqueda y pruebas de la base de datos, así como también de la realización de los scripts.

Chloé y Sergio se encargaron del VO y DAO y de la implementación de estos en el entorno de .NET, así como las pruebas de estos donde José también ayudó.

Los 3 se han encargado juntos del diseño de la base de datos y del razonamiento de este.

Costes

Tareas	Horas
Búsqueda de diferentes entornos para la BD	3h
Diseño BD	3h
Creación e implementación de scripts para la BD	4h
Instalación del entorno en .NET	4h
DAO/VO	5h
Documentación	2h

Dificultades encontradas

Se han tenido dificultades prácticamente en todas las tareas. Para el diseño de base de datos se gasto mucho tiempo en discurrir como diseñarla y que tablas eran necesarias para la realización de ciertas tareas.

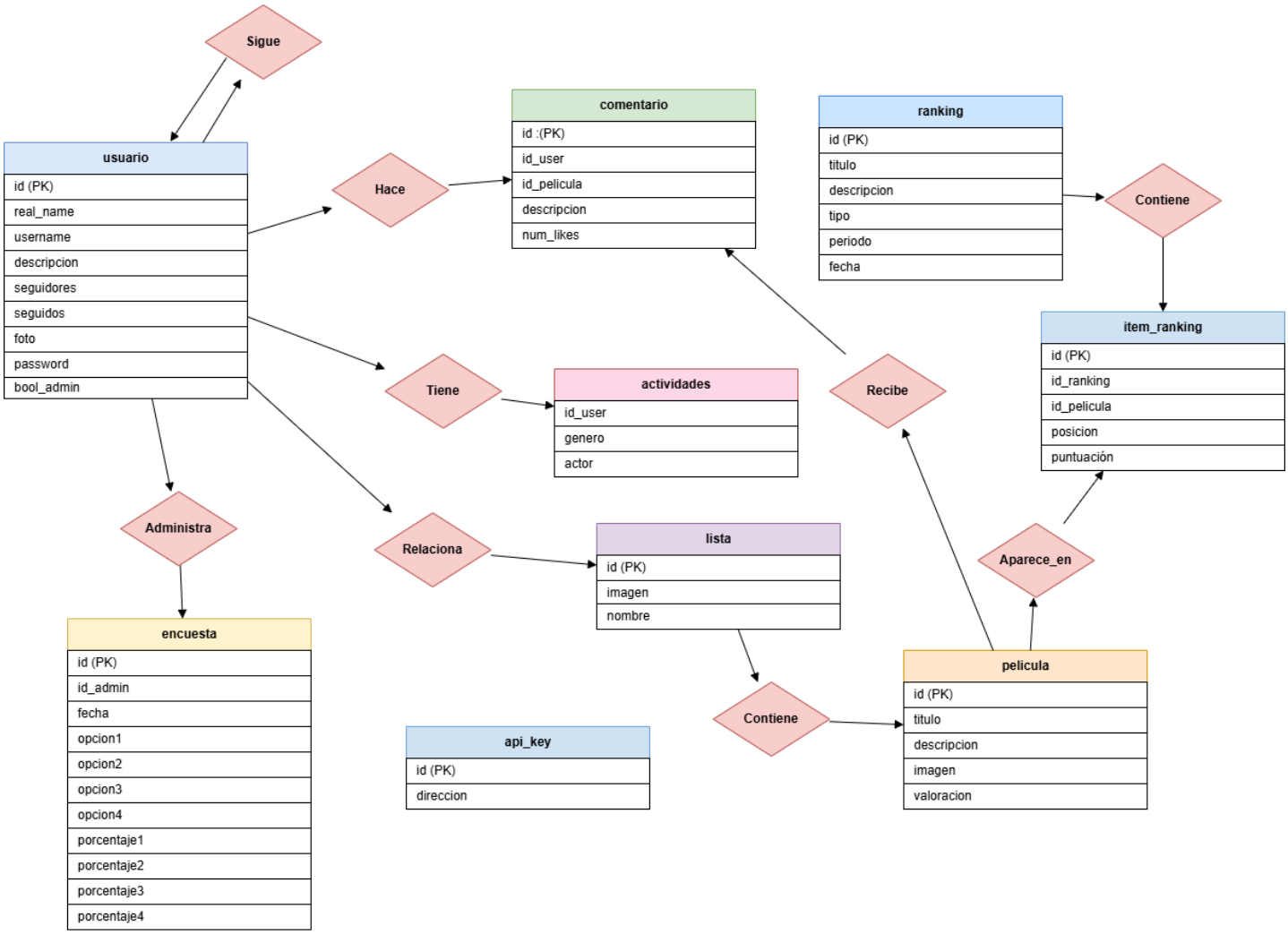
Para la búsqueda de dónde hacer la BD al principio se decidió de Supabase hasta había que enfrentarse al problema que antes se ha mencionado por lo que se decidió cambiar a Neon después de una larga búsqueda y de probar otras opciones.

Para los scripts fue donde menos problemas hubo ya que era bastante repetitivo y sencillo hacerlo, aunque algún error a la hora de ejecutarlo se tuvo, pero no invirtió mucho tiempo arreglarlo.

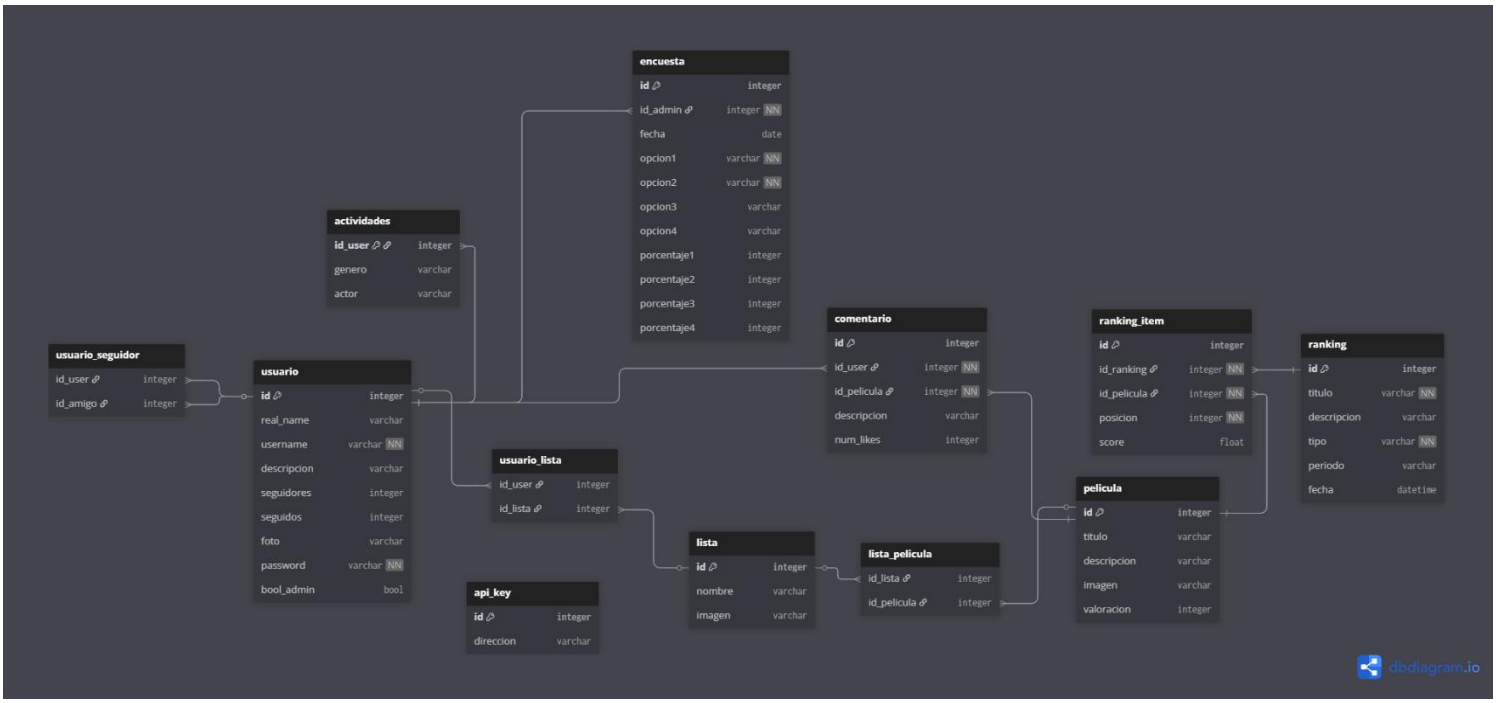
En la instalación para usar el entorno y utilizar un backend con .NET si que hubo muchas dificultades en la creación del proyecto, en buscar las librerías exactas para conectarse la base de datos y en la realización de la parte de DAO/VO, se invirtieron muchas horas en buscar librerías y en crear un proyecto acorde con lo que se necesitaba.

Anexos

Modelo E/R



Modelo lógico relacional



Código de dbdiagram:

```
// Use DBML to define your database structure
// Docs: https://dbml.dbdiagram.io/docs

Table usuario{
  id integer [pk, increment]
  real_name varchar
  username varchar [unique, not null]
  descripcion varchar
  seguidores integer [default: 0]
  seguidos integer [default: 0]
  foto varchar
  password varchar [not null]
  bool_admin bool [default: false]
}

Table lista{
  id integer [pk, increment]
  nombre varchar
  imagen varchar
}


Table api_key{
```

```
  id integer [pk, increment]
  direccion varchar
}

Table encuesta{
  id integer [pk, increment]
  id_admin integer [not null]
  fecha date
  opcion1 varchar [not null]
  opcion2 varchar [not null]
  opcion3 varchar
  opcion4 varchar
  porcentaje1 integer [default: 0]
  porcentaje2 integer [default: 0]
  porcentaje3 integer [default: 0]
  porcentaje4 integer [default: 0]
}

Table usuario_lista {
  id_user integer
  id_lista integer

  Indexes {
    (id_user, id_lista) [unique]  // un usuario no se asocia dos veces a la misma lista
    id_user
    id_lista
  }
}

Table lista_pelicula{
  id_lista integer
  id_pelicula integer

  Indexes {
    (id_lista, id_pelicula) [unique] // una peli no puede repetirse en la misma lista
    id_lista
    id_pelicula
  }
}
Table comentario{
  id integer [pk, increment]
  id_user integer  [not null]
  id_pelicula integer [not null]
  descripcion varchar
```

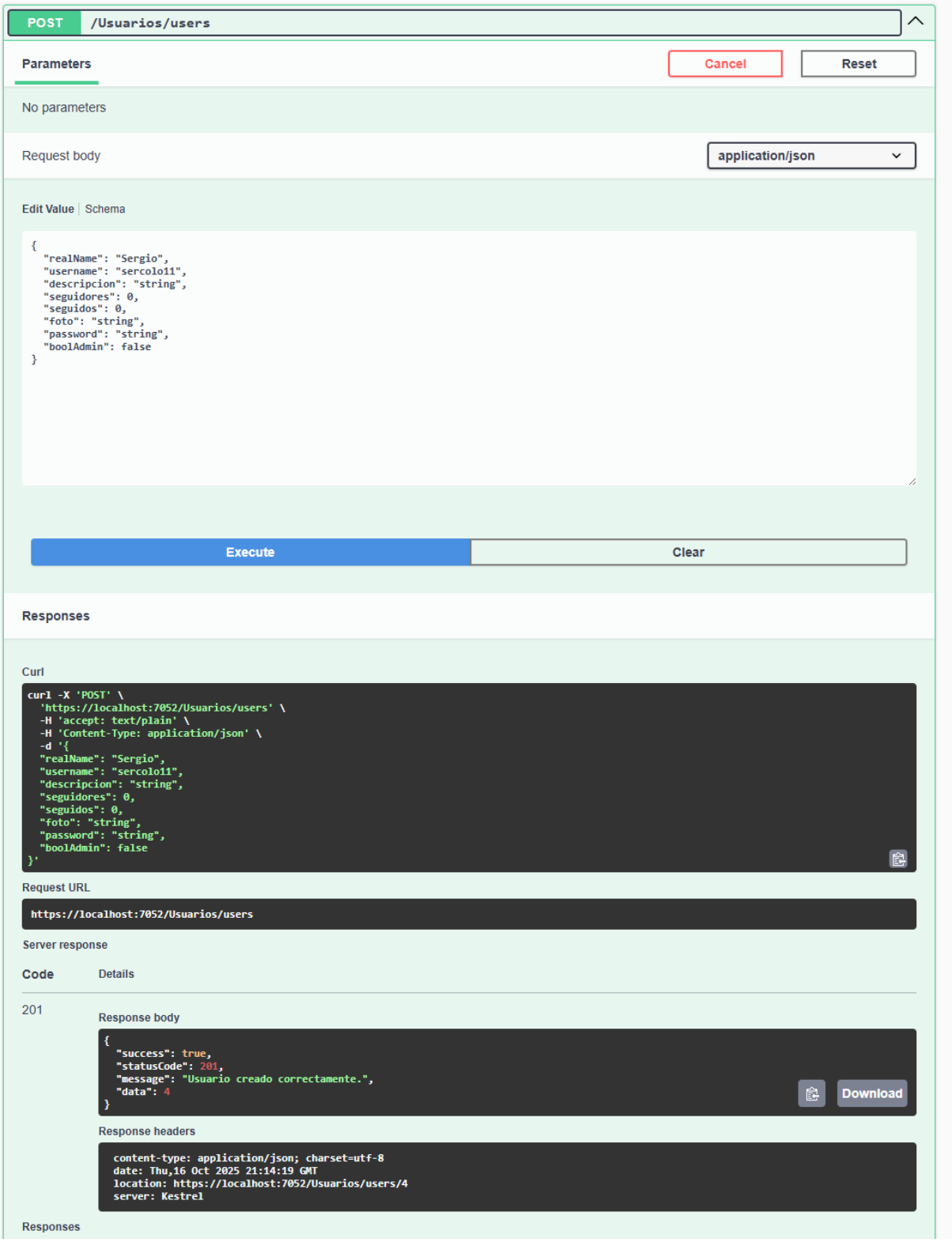
```
  num_likes integer
}

Table pelicula{
  id integer [pk, increment]
  titulo varchar
  descripcion varchar
  imagen varchar
  valoracion integer
}

Table usuario_seguidor{
  id_user integer
  id_amigo integer
}

Table actividades{
  id_user integer [pk, increment]
  genero varchar // Para sugerencias de genero
  actor varchar  // Para sugerencias con actores
}

Table ranking {
  id integer [pk, increment]
  titulo varchar [not null]
  descripcion varchar
  tipo varchar [not null]  //popularidad, valoracion, estrenos, genero
  periodo varchar   //semanal, mensual, global
  fecha datetime
}

Table ranking_item {
  id          integer [pk, increment]
  id_ranking   integer [not null]
  id_pelicula  integer [not null]
  posicion     integer [not null]
  score        float

  Indexes {
    (id_ranking, id_pelicula) [unique]  // evitar repetir la misma peli en el mismo
ranking
    (id_ranking, posicion)    [unique]  // evitar duplicar posición dentro del ranking
    id_ranking
  }
```

}

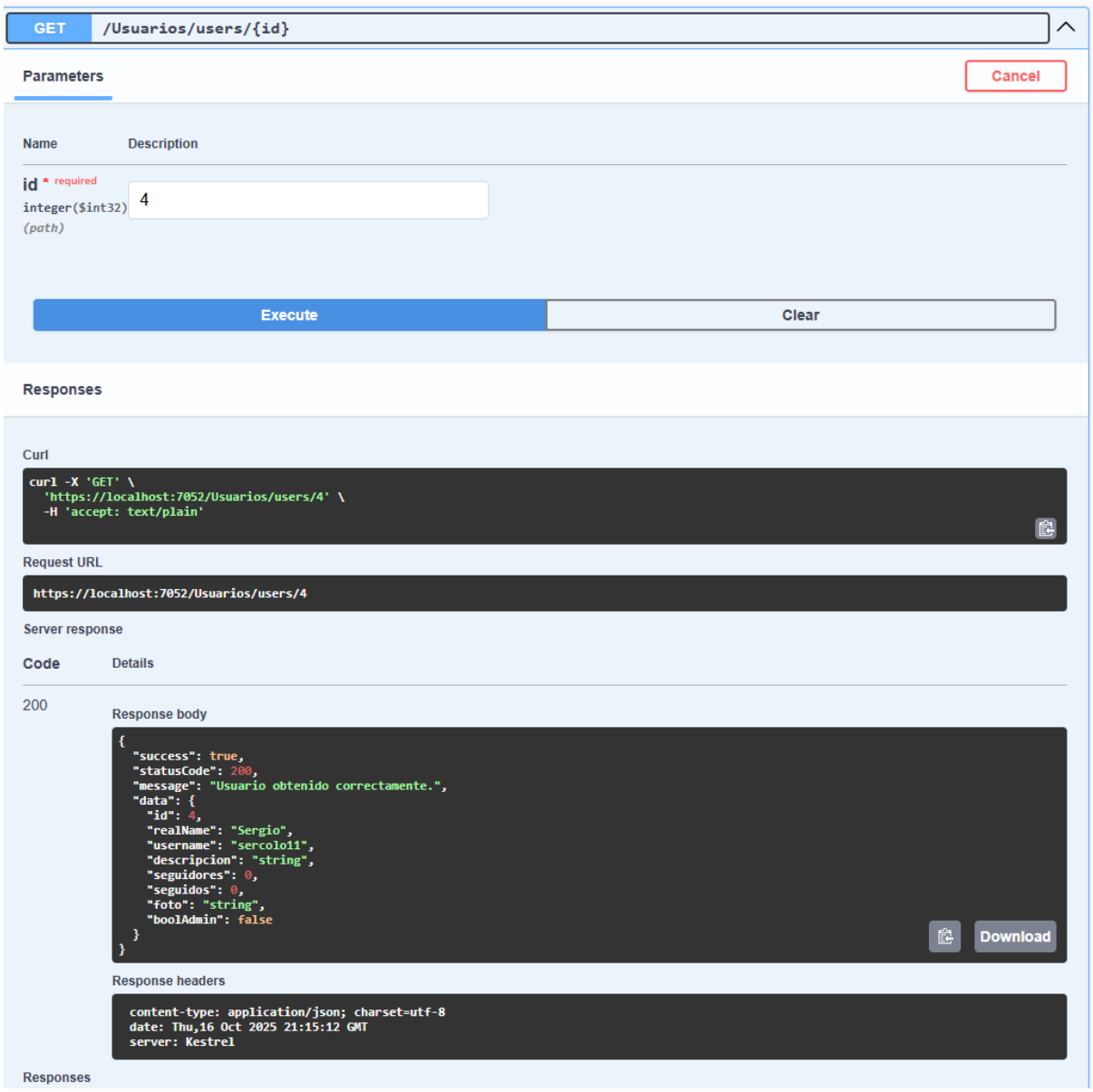
ref: comentario.id_user > usuario.id
ref: comentario.id_pelicula > pelicula.id
ref: encuesta.id_admin > usuario.id
ref: usuario_lista.id_user > usuario.id
ref: usuario_lista.id_lista > lista.id
ref: lista_pelicula.id_pelicula > pelicula.id
ref: lista_pelicula.id_lista > lista.id
ref: usuario_seguidor.id_user > usuario.id
ref: usuario_seguidor.id_amigo > usuario.id
ref: actividades.id_user > usuario.id
ref: ranking_item.id_ranking > ranking.id
ref: ranking_item.id_pelicula > pelicula.id

Capturas Test

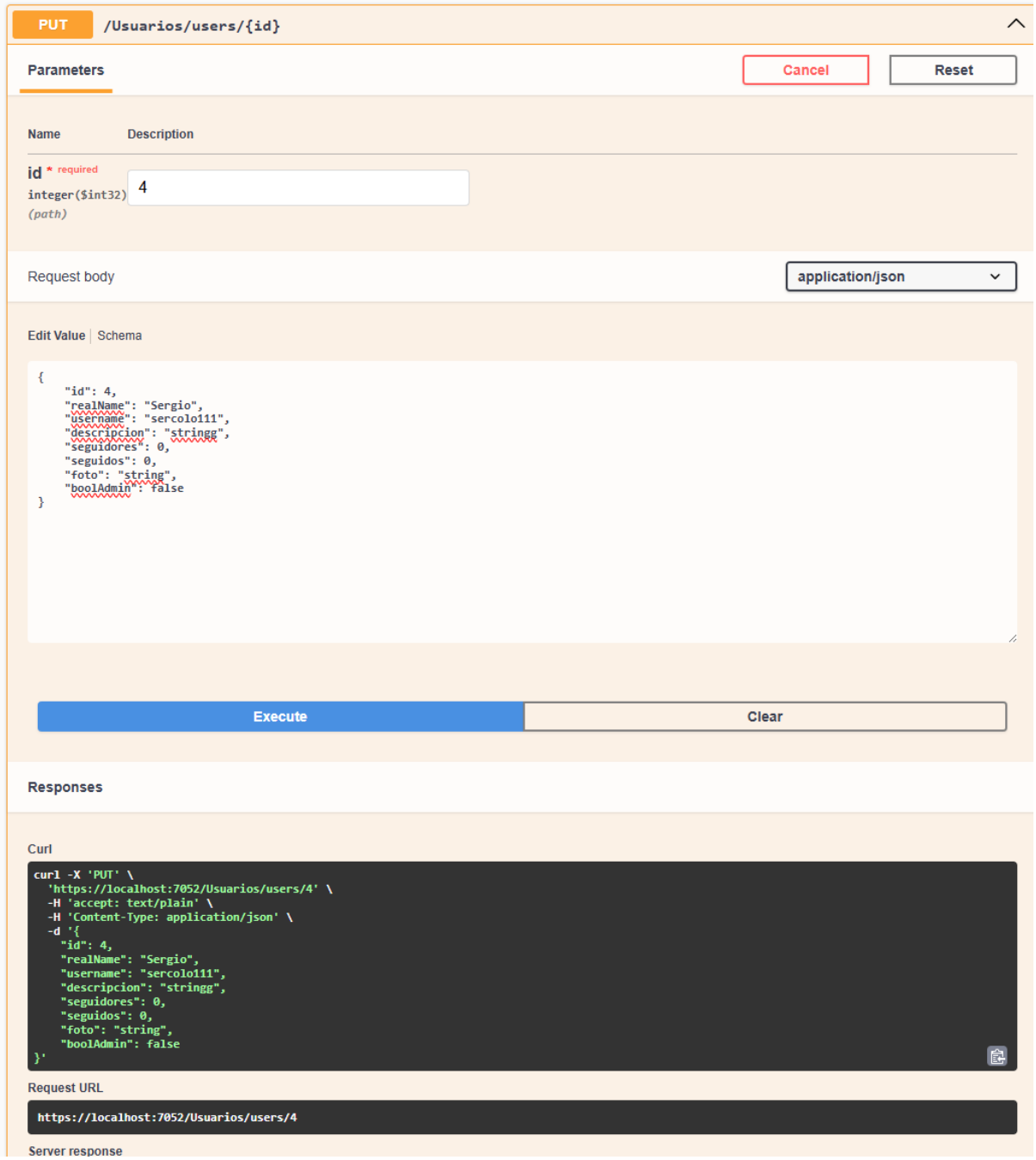
Si sale code 200 significa que lo realizó correctamente.
Creación usuarios:



Obtención del Usuario:



Actualización de Usuario:



Eliminar Usuario: