

Documentación Programación de Redes

Obligatorio 2

Diego Franggi 210434
Javier de Mattos 222439

<https://github.com/diale13/obligatorioRedes/>

2 de julio de 2020

Índice general

1.	Introducción	3
2.	Mecanismos generales	3
2.1.	Arquitectura general	3
2.2.	Paquetes	3
2.3.	Asincronía	4
2.4.	Semáforos	5
2.5.	Remoting	5
2.6.	Message Queue	7
2.7.	Filtros	8
2.8.	Configuraciones	9
3.	Endpoints	9
3.1.	User	9
3.2.	User/{username}	10
3.3.	User/{username}/{favoritemovies}	10
3.4.	Token	10
3.5.	movie	10
3.6.	movie/{movieName}/	10
3.7.	movie/{movieName}/rating	10
3.8.	Log/	10
3.9.	Log/{filter}/	11
4.	Casos de prueba usuarios	11
4.1.	Dar de alta usuario	11
4.2.	Modificar usuario / Traer usuario modificado	11
4.3.	Usuario único	12
4.4.	Borrar usuario (luego de login)	13
4.5.	Validación de modelos	13
5.	Casos de prueba login	14
5.1.	Filtro de autorización	14
5.2.	Login correcto	15
5.3.	Login incorrecto	15
6.	Pruebas sobre películas	16
6.1.	Ver película	16
6.2.	Agregar película favoritas	17
6.3.	Remover película favorita	18
6.4.	Obtener películas favoritas de usuario	18
6.5.	Agregar películas favoritas a otro usuario	19
6.6.	Agregar películas favoritas que no existen	19

7.	Calificaciones	20
7.1.	Voto correcto	20
7.2.	Traer resultado	20
7.3.	Traer resultado luego de dos votos (promedio)	21
7.4.	Voto fuera de rango	22
7.5.	Votar por otro	23

1. Introducción

Se decide expandir el sistema construido previamente exponiendo nuevas funcionalidades en un nuevo servidor que permite conexiones mediante http. Este servidor (llamado administrativo) permite el manejo de usuarios, su administración y modificación de selecciones de películas favoritas. De igual modo estos usuarios pueden acceder a la lista de películas del servidor realizado en la primer entrega y la calificación de estas por parte de los usuarios. Asimismo se convierten las tecnologías sincrónicas utilizadas en la primer entrega a una versión asincrónica. Se implementa también un sistema de *log* de eventos usando las tecnologías de *Microsoft message queue*.

2. Mecanismos generales

2.1. Arquitectura general

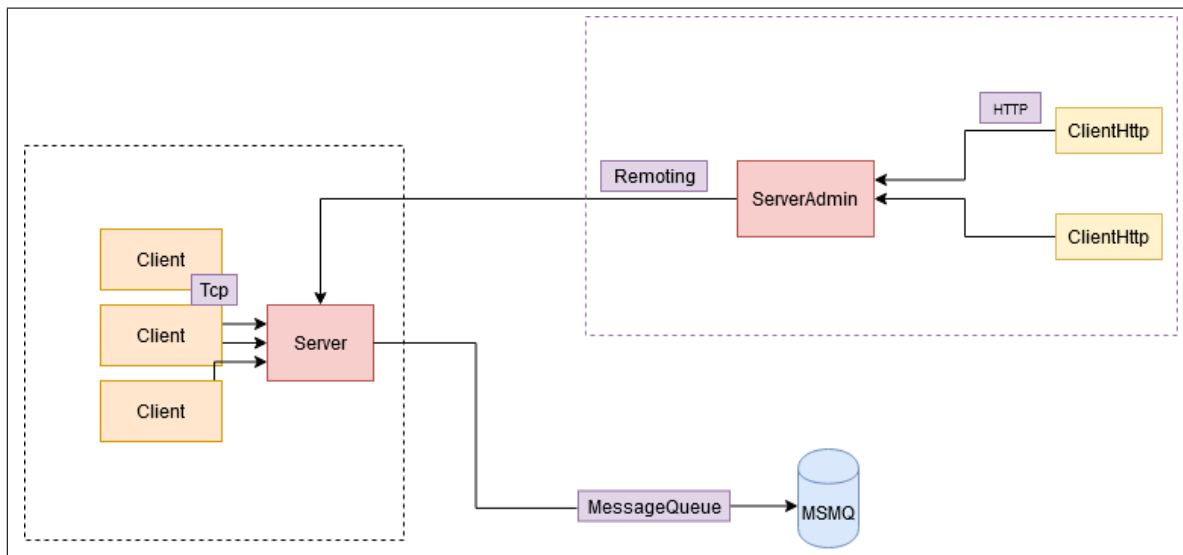


Figura 1: Diagrama

Con esto vemos un pantallazo de la estructura de la aplicación. La ubicada más a la izquierda es la vista en el obligatorio uno. Esta fue actualizada para realizar logs de sus eventos y los procesados de la capa http mediante una conexión a la cola de mensajes. ServerAdmin refiere a aquellos componentes de webapi, esta se hace apoyo en los servicios expuestos por el servidor mediante remoting.

2.2. Paquetes

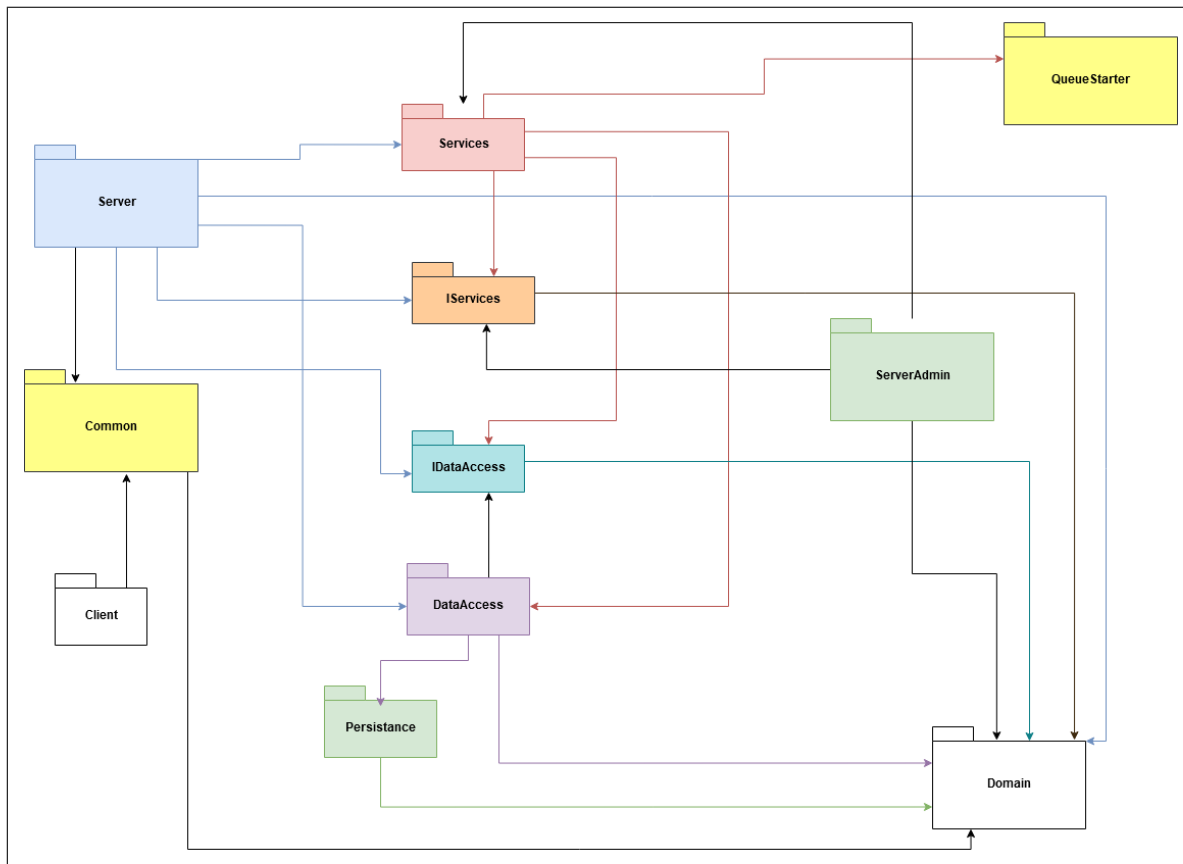


Figura 2: Diagrama de paquetes

2.3. Asincronía

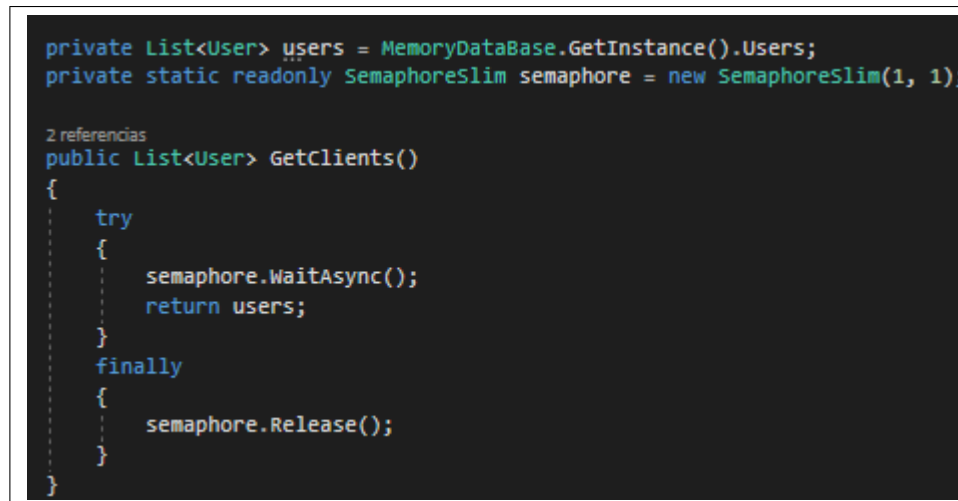
Como solicitado se convierten los métodos previamente bloqueantes por su uso de threads a tareas (task) usando patrón async await. Para esto se eliminaron completamente el uso de threads, además de sustituir métodos como por ejemplo *AcceptTcpClient* a su versión asíncrona, *AcceptTcpClientAsync*. Se enfatiza que la *web api* también cumple este patrón, sin embargo se ha de usar como asistencia el metodo *await Task().Yield* para convertir sus métodos.

```
try
{
    Task serverFunctionsTask = Task.Run(() => ServerFunctions());
    while (isServerUp)
    {
        var tcpClientSocket = await tcpListener.AcceptTcpClientAsync().ConfigureAwait(false);
        var handleClientTask = Task.Run(async () => await HandleClient(tcpClientSocket).ConfigureAwait(false));
    }
}
```

Figura 3: Uso de async-await

2.4. Semáforos

Para continuar con el pasaje a asincronía debemos eliminar los lock que utilizábamos previamente para garantizar una solución al problema de la mutua exclusión entre los recursos de persistencia. Para esto hacemos uso de la clase *SemaphoreSlim*, inicializándolos con una cantidad máxima de uno para simular los lock de una forma no bloqueante. Luego cuidamos el liberar el semáforo mediante el uso de la sentencia *finally*.

A screenshot of a code editor showing C# code. The code defines a static semaphore and a method to get clients asynchronously, using try-finally blocks to ensure the semaphore is released.

```
private List<User> users = MemoryDataBase.GetInstance().Users;
private static readonly SemaphoreSlim semaphore = new SemaphoreSlim(1, 1);

2 referencias
public List<User> Getclients()
{
    try
    {
        semaphore.WaitAsync();
        return users;
    }
    finally
    {
        semaphore.Release();
    }
}
```

Figura 4: Uso de semáforo

2.5. Remoting

Para que la web api realice correctamente sus funcionalidades hace uso de **servicios** que expone-mos en el servidor. La api solamente conoce las interfaces que requiere y obtiene con esta tecnología acceso a ellas. La ventaja es que el servidor solo necesita conocer el paquete de interfaces y no el del servidor al que se conecta eliminando dependencias innecesarias. Además estos servicios expuestos pueden acceder fácilmente a la base de datos en memoria. Para la inicialización de estos servicios en el servidor nos hicimos de ayuda de una clase llamada **RemotingManager** el que es encargado de iniciar los servicios. Estos son inicializados en el canal de tcp cada uno con su puerto **y nombre** puntuales (se usa el metodo getchannel para esto). El servidor cuida cerrar estos canales luego de atender las peticiones.

```

1 referencia
public static TcpChannel InitiateLogService()
{
    var port = Int32.Parse(SettingsMgr.ReadSetting(ServerConfig.LogServicePort));
    var logServiceChannel = (TcpChannel)GetChannel("logServiceChannel", port, false);
    ChannelServices.RegisterChannel(logServiceChannel, false);
    RemotingConfiguration.RegisterWellKnownServiceType(
        typeof(logService),
        "logService",
        WellKnownObjectMode.Singleton);
    return logServiceChannel;
}

4 referencias
public static IChannel GetChannel(string name, int tcpPort, bool isSecure)
{
    BinaryServerFormatterSinkProvider serverProv =
        new BinaryServerFormattersSinkProvider
        {
            TypeFilterLevel = TypeFilterLevel.Full
        };
    IDictionary propBag = new Hashtable
    {
        ["port"] = tcpPort,
        ["typeFilterLevel"] = TypeFilterLevel.Full,
        ["name"] = name
    };
    if (isSecure)
    {
        propBag["secure"] = isSecure;
        propBag["impersonate"] = false;
    }
    return new TcpChannel(
        propBag, null, serverProv);
}

```

Figura 5: RemotingManager

```

var sessionServiceChannel = RemotingManager.InitiateRemotingSessionService();
var apiUserServiceChannel = RemotingManager.InitiateRemotingApiUserService();
var movieServiceChannel = RemotingManager.InitiateRemotingMovieService();
var logServiceChannel = RemotingManager.InitiateLogService();

try
{
    Task serverFunctionsTask = Task.Run(() => ServerFunctions());
    while (isServerUp)
    {
        var tcpClientSocket = await tcpListener.AcceptTcpClientAsync().ConfigureAwait(false);
        var handleClientTask = Task.Run(async () => await HandleClient(tcpClientSocket).ConfigureAwait(false));
    }
}
finally
{
    ChannelServices.UnregisterChannel(sessionServiceChannel);
    ChannelServices.UnregisterChannel(apiUserServiceChannel);
    ChannelServices.UnregisterChannel(movieServiceChannel);
    ChannelServices.UnregisterChannel(logServiceChannel);
}

```

Figura 6: Inicialización de servicios

```

public class LogController : ApiController
{
    private ILogService logService;
    ...
    0 referencias
    public LogController()
    {
        logService = (ILogService)Activator.GetObject(
            typeof(ILogService), ApiConfig.LogServiceIP);
    }
}

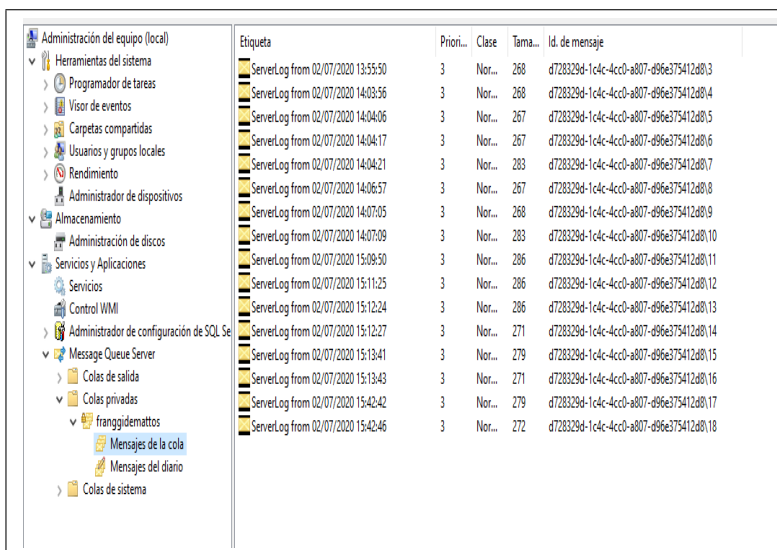
```

Figura 7: Código en controller

2.6. Message Queue

Para incluir las colas de mensajes primero se deberá iniciar una instancia desde el proyecto *QueueStarter*, este verificará la existencia de la cola y de no existir la creará. Luego de esto nuestro sistema ya comienza a estar en funcionamiento. Se pueden ver directo desde la cola de windows o solicitándolas vía web api con o sin filtro.

Ejemplos:



Etiqueta	Prioridad	Clase	Tamaño	Id. de mensaje
ServerLog from 02/07/2020 13:55:50	3	Nor...	268	d728329d-1c4c-4cc0-a807-d96e375412d8.3
ServerLog from 02/07/2020 14:03:56	3	Nor...	268	d728329d-1c4c-4cc0-a807-d96e375412d8.4
ServerLog from 02/07/2020 14:04:06	3	Nor...	267	d728329d-1c4c-4cc0-a807-d96e375412d8.5
ServerLog from 02/07/2020 14:04:17	3	Nor...	267	d728329d-1c4c-4cc0-a807-d96e375412d8.6
ServerLog from 02/07/2020 14:04:21	3	Nor...	283	d728329d-1c4c-4cc0-a807-d96e375412d8.7
ServerLog from 02/07/2020 14:06:57	3	Nor...	267	d728329d-1c4c-4cc0-a807-d96e375412d8.8
ServerLog from 02/07/2020 14:07:05	3	Nor...	268	d728329d-1c4c-4cc0-a807-d96e375412d8.9
ServerLog from 02/07/2020 14:07:09	3	Nor...	283	d728329d-1c4c-4cc0-a807-d96e375412d8.10
ServerLog from 02/07/2020 15:09:50	3	Nor...	286	d728329d-1c4c-4cc0-a807-d96e375412d8.11
ServerLog from 02/07/2020 15:11:25	3	Nor...	286	d728329d-1c4c-4cc0-a807-d96e375412d8.12
ServerLog from 02/07/2020 15:12:24	3	Nor...	286	d728329d-1c4c-4cc0-a807-d96e375412d8.13
ServerLog from 02/07/2020 15:12:27	3	Nor...	271	d728329d-1c4c-4cc0-a807-d96e375412d8.14
ServerLog from 02/07/2020 15:13:41	3	Nor...	279	d728329d-1c4c-4cc0-a807-d96e375412d8.15
ServerLog from 02/07/2020 15:13:43	3	Nor...	271	d728329d-1c4c-4cc0-a807-d96e375412d8.16
ServerLog from 02/07/2020 15:42:42	3	Nor...	279	d728329d-1c4c-4cc0-a807-d96e375412d8.17
ServerLog from 02/07/2020 15:42:46	3	Nor...	272	d728329d-1c4c-4cc0-a807-d96e375412d8.18

Figura 8: Vista de logs en message queue

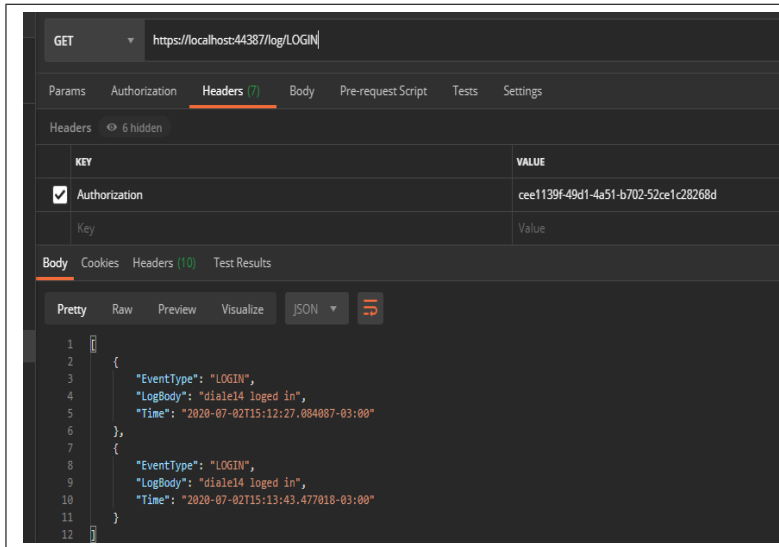


Figura 9: Get all

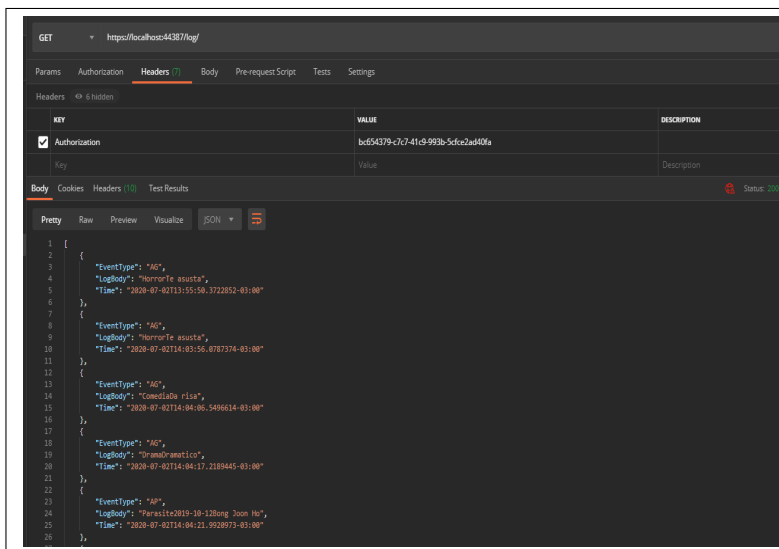


Figura 10: Get by filter

2.7. Filtros

- Borrar genero (BG)
- Modificar genero (MG)
- Alta de pelicula (AP)

- Baja de pelicula (BP)
- Modificar pelicula (MP)
- Asociar pelicula a genero (AS)
- Desasociar pelicula a genero (DS)
- Alta de director (AD)
- Baja de director (BD)
- Asociar director y pelicula (DM)
- Desasociar director y pelicula (DD)
- Modificar director (MD)
- Subir archivos de pelicula (SA)
- Usuario de api creado (POSTUSER)
- Usuario de api borrado (DEUSER)
- Usuario de api actualizado (PUTUSER)
- Usuario de api obtenido (GETUSER)
- Pelicula favorita agregada (ADDFAV)
- Pelicula favorita removida (DELFAY)
- Pelicula favorita obtenida (GETFAV)
- Calificacion borrada (DELRATING)
- Calificacion agregada (ADDRATING)
- Usuario logeado (LOGIN)

2.8. Configuraciones

No se presentan Ip's ni puertos harcodeados, estos han de ser modificados de las clases app.config o ApiConfig.cs en el servidor administrativo.

3. Endpoints

3.1. User

- **Post:** al hacer post con un cuerpo de request correspondiente a un modelo de registro de usuario se agrega un usuario al sistema
- **Put:** actualiza un usuario (no es posible actualizar un nickname)
- **Delete:** se ha de brindar usuario y contraseña como confirmación de que se desea borrar el usuario además de estar logeado.

3.2. User/{username}

- **Get:** se obtiene información del usuario solicitado

3.3. User/{username}/{favoritemovies}

Suponiendo usuario está logeado:

- **Get:** se obtiene lista de películas favoritas del usuario
- **Post:** se agrega película favorita al usuario si esta existe
- **Delete:** se remueve película favorita del usuario.

No es posible postear ni borrar películas de un usuario cuyo token de autorización no coincide con su nombre.

3.4. Token

Endpoint de log in

- **Post:** se envía usuario y contraseña, retorna token que ha de ser usado como header de autorización.
- **Delete:** se hace logout del usuario, removiendo su sesión.

3.5. movie

Todos las siguientes entradas requieren autorización o retornarán código http 403

- **Get:** retorna películas registradas por usuarios interactuando con el servidor del primer obligatorio.

3.6. movie/{movieName}/

- **Get** Retorna la película cuyo nombre coincide con el parámetro.

3.7. movie/{movieName}/rating

- **Get** trae la puntuación de esa película (total).
- **Post** agrega una calificación específica para el usuario.
- **Put** cambia la calificación que se había dado previamente o la inserta de no existir.
- **Delete** borra la calificación que dio ese usuario.

3.8. Log/

- **Get :** se traen todos los *log* del sistema

3.9. Log/{filter}/

- **Get** : se traen los *log* del sistema filtrados acorde al parámetro.

4. Casos de prueba usuarios

4.1. Dar de alta usuario

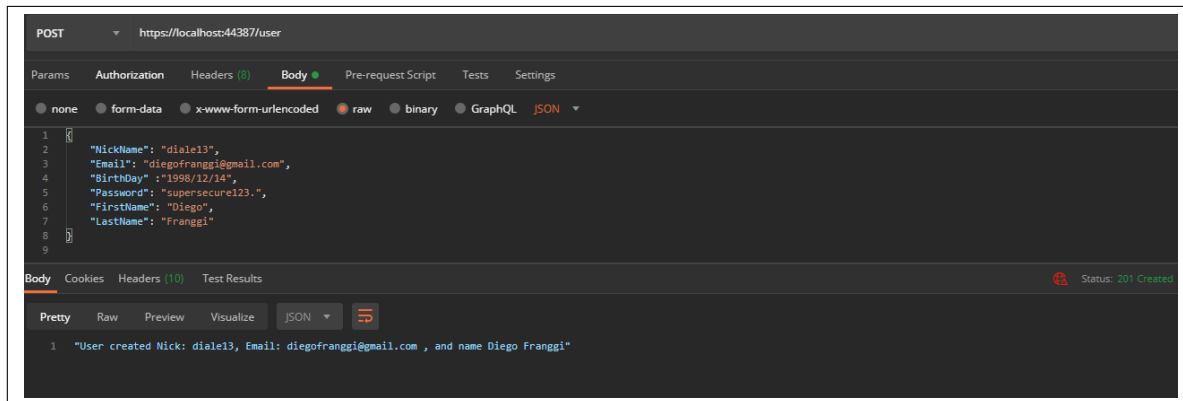


Figura 11: Alta de usuario

POST A USER

```
{
  "NickName": "diale13",
  "Email": "diegofranggi@gmail.com",
  "BirthDay" : "1998/12/14",
  "Password": "supersecure123.",
  "FirstName": "Diego",
  "LastName": "Franggi"
}
```

4.2. Modificar usuario / Traer usuario modificado

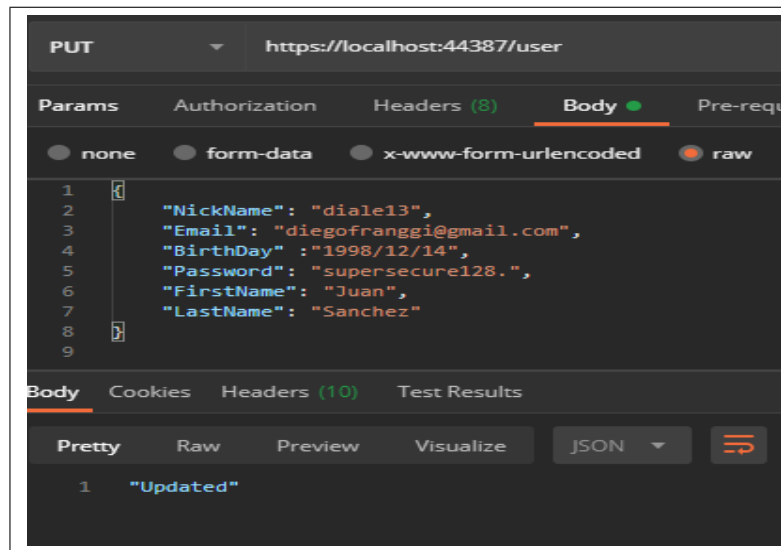


Figura 12: Actualizamos el usuario

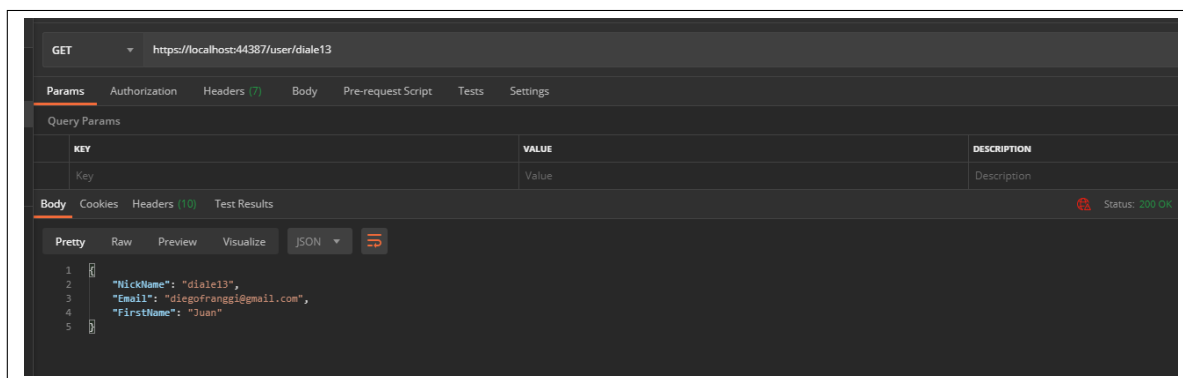


Figura 13: Lo obtenemos

```

{
  "NickName": "diale13",
  "Email": "diegofranggi@gmail.com",
  "BirthDay": "1998/12/14",
  "Password": "supersecure128.",
  "FirstName": "Juan",
  "LastName": "Sanchez"
}

```

4.3. Usuario único

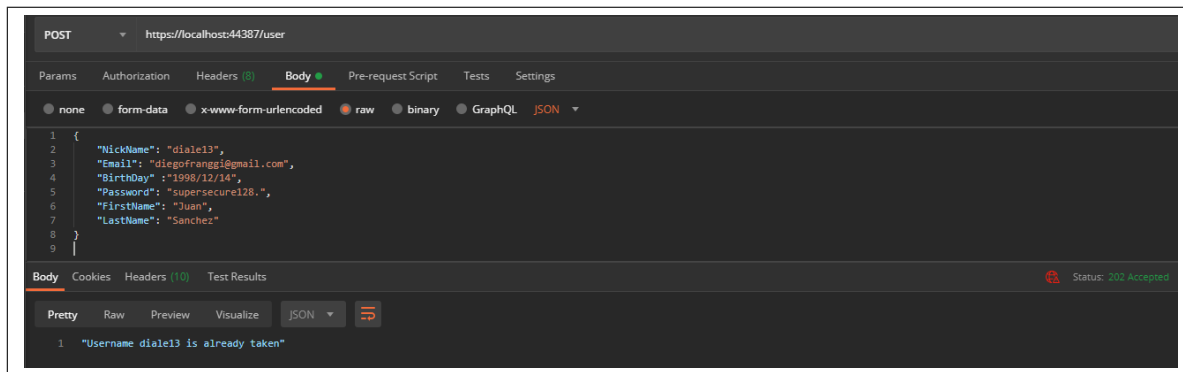


Figura 14: Lo obtenemos

Repetimos el post

```

{
  "NickName": "diale13",
  "Email": "diegofranggi@gmail.com",
  "BirthDay": "1998/12/14",
  "Password": "supersecure128.",
  "FirstName": "Juan",
  "LastName": "Sanchez"
}

```

4.4. Borrar usuario (luego de login)

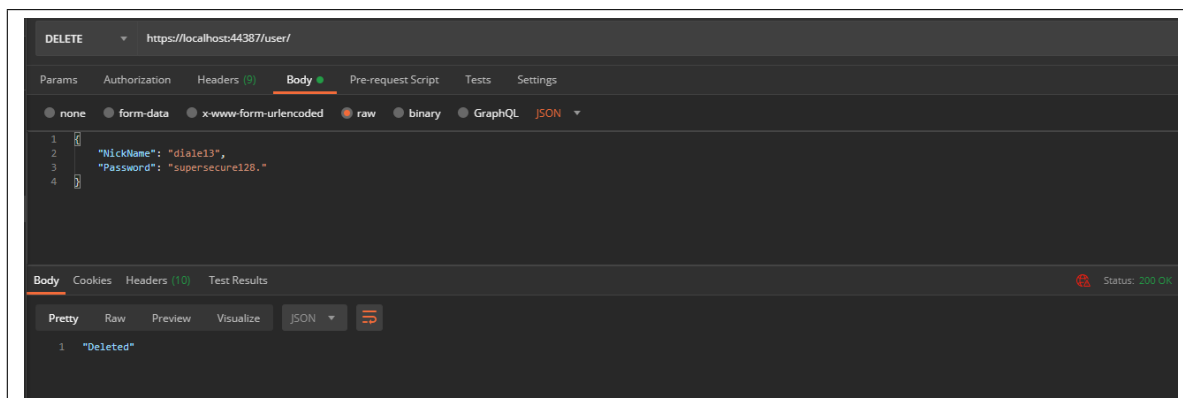


Figura 15: Borrado (si se proporcionó token de login)

4.5. Validación de modelos

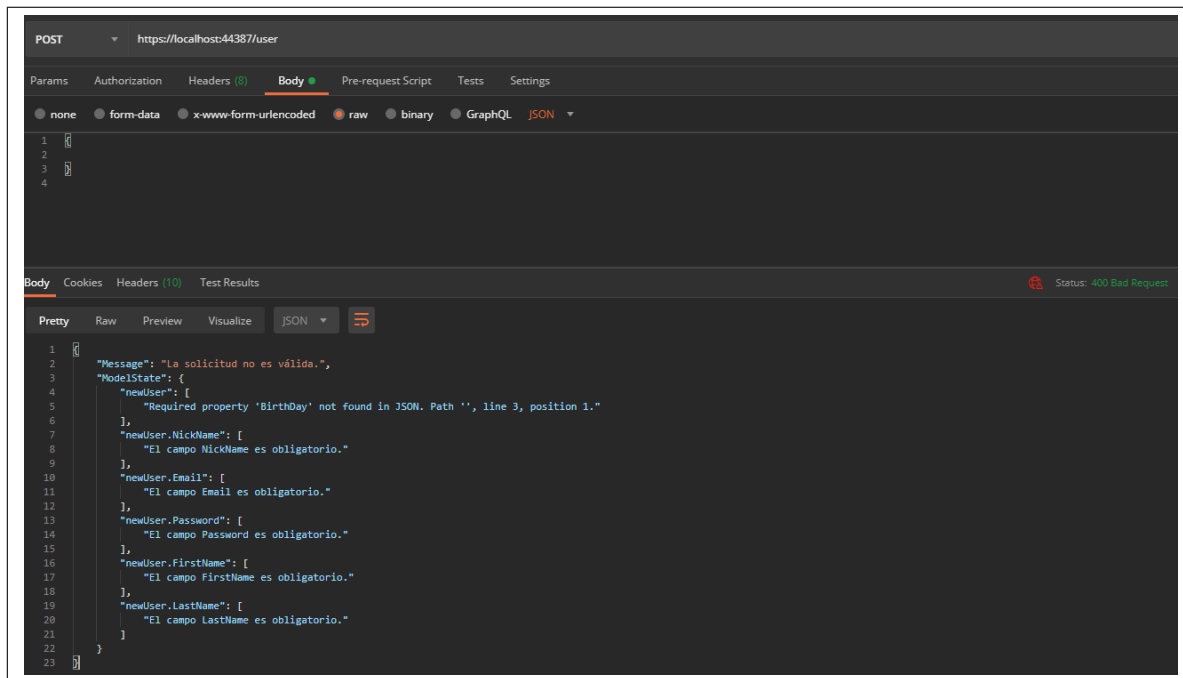


Figura 16: Validaci3n de modelo de registro. Adem\u00e1s se valida contrase\u00f1a compleja y tipo de correo electr3nico

5. Casos de prueba login

5.1. Filtro de autorizaci3n

Ejemplo de filtro de autorizaci3n en funcionamiento:

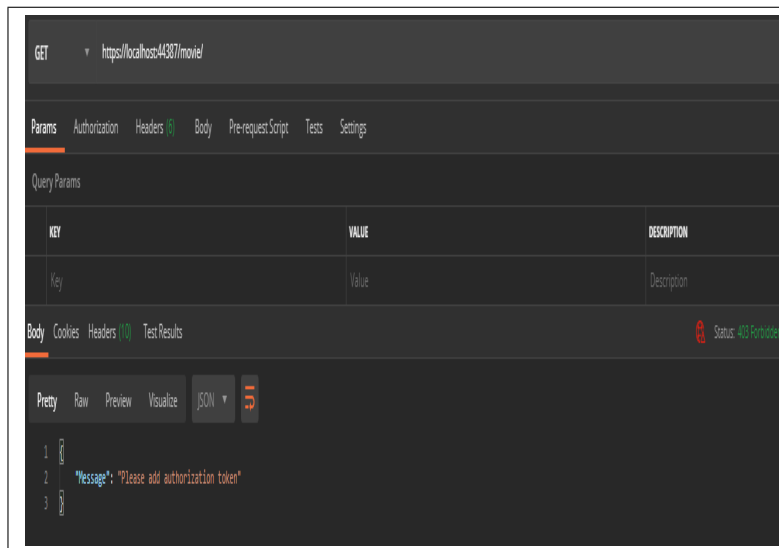


Figura 17: Filtro para obtener todas las películas requiere autorización

Mismo filtro para la selección de películas favoritas y para calificar películas

5.2. Login correcto

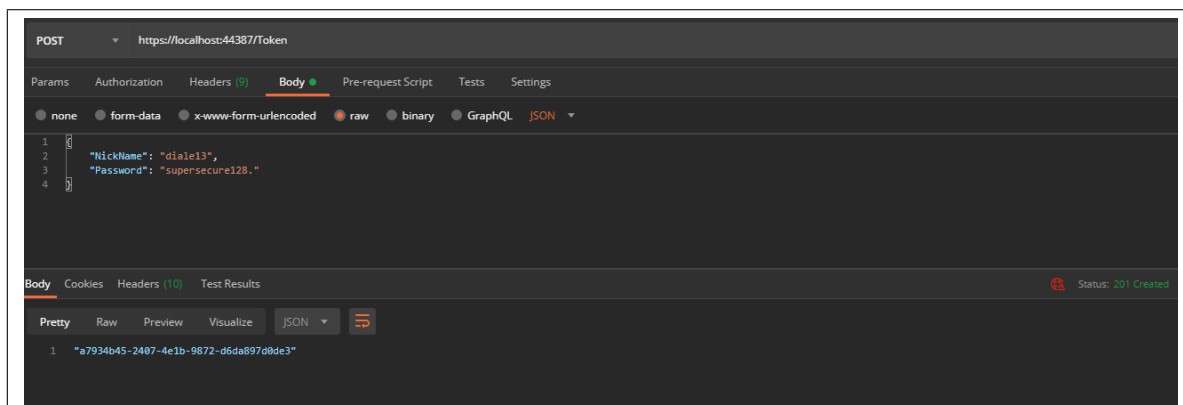


Figura 18: El token brindado sera nuestro header Authorization

5.3. Login incorrecto

Un **login incorrecto** devuelve una lista vacía, es decir no habrá tal token usable.

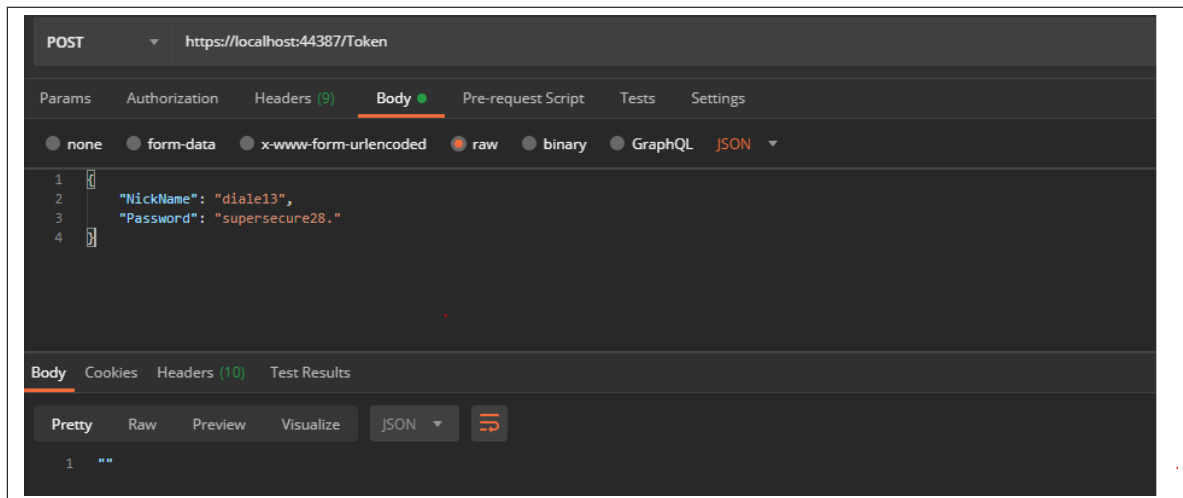


Figura 19: Login erróneo

6. Pruebas sobre películas

Para esto han de existir películas dadas de alta en el servidor viejo por lo que se brinda el mismo código que dimos para la primera instancia:

```
REQAG9999Drama@Dramatico
REQAG9999Horror@Te asusta
REQAG9999Comedia@Da risa
```

```
REQAP9999Parasite@2019-10-12@Bong Joon Ho@Drama
REQAP9999Mi vida@1998-09-12@Diego Franggi@Horror#Drama#Comedia
REQAP9999Killer Balloon@2020-04-12@Javier de Mattos@Drama#Horror
```

6.1. Ver película

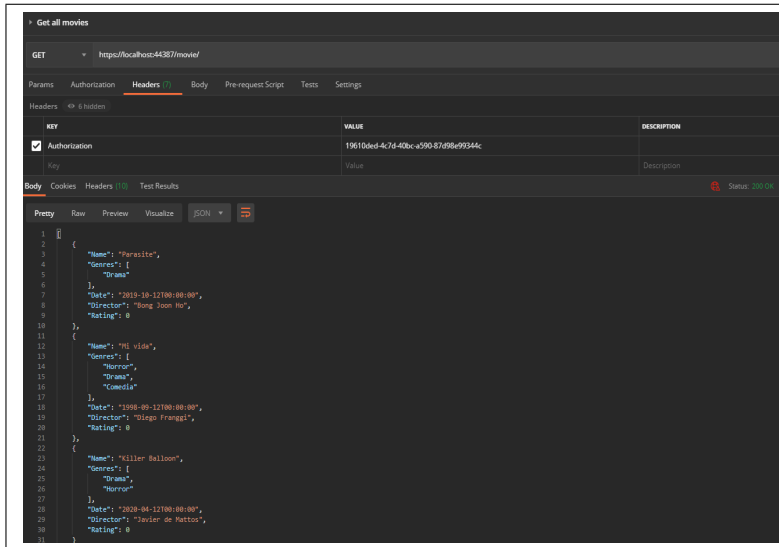


Figura 20: Obtener todas las películas

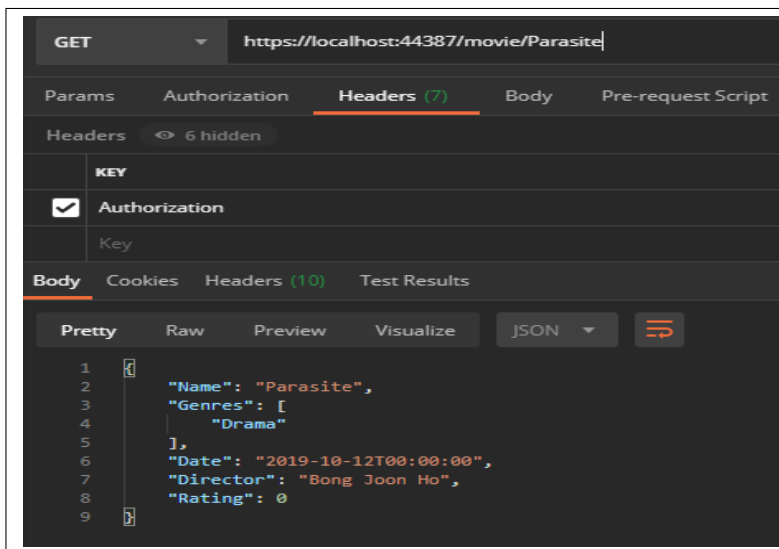


Figura 21: Obtener una película

6.2. Agregar película favoritas

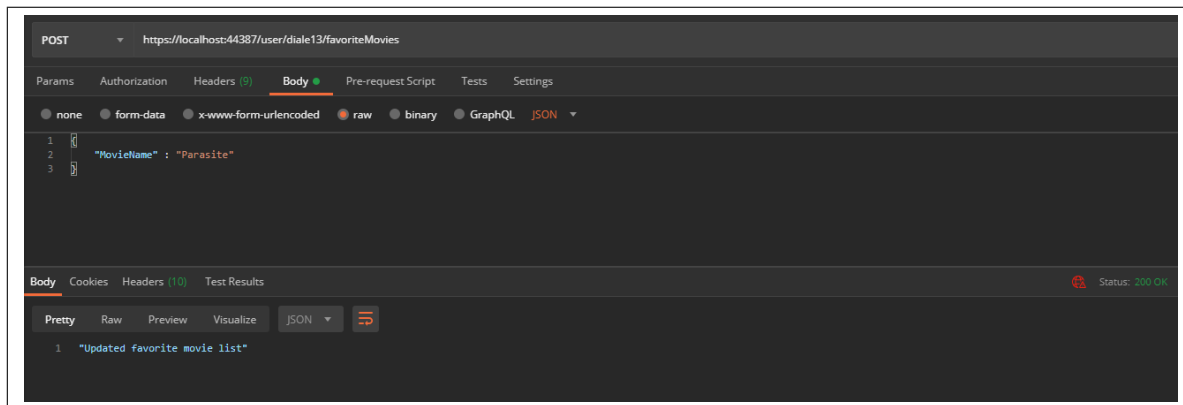


Figura 22: Agrega película a favoritas

6.3. Remover película favorita

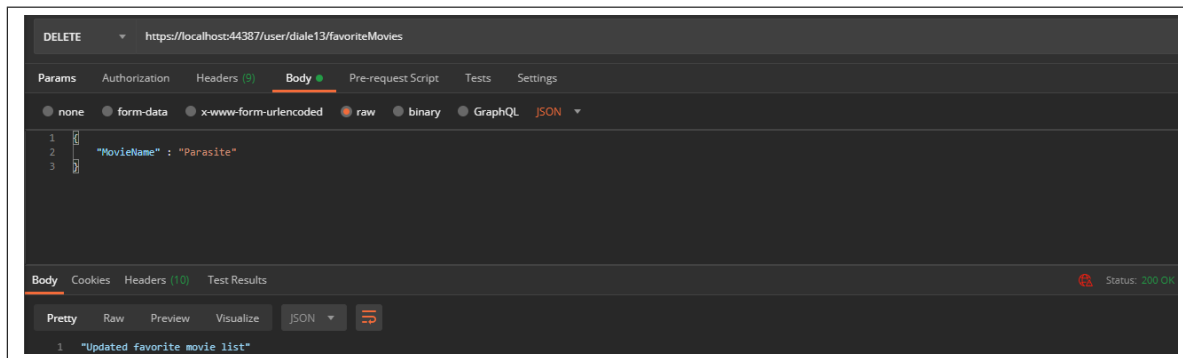


Figura 23: Ejemplo de caso

6.4. Obtener películas favoritas de usuario

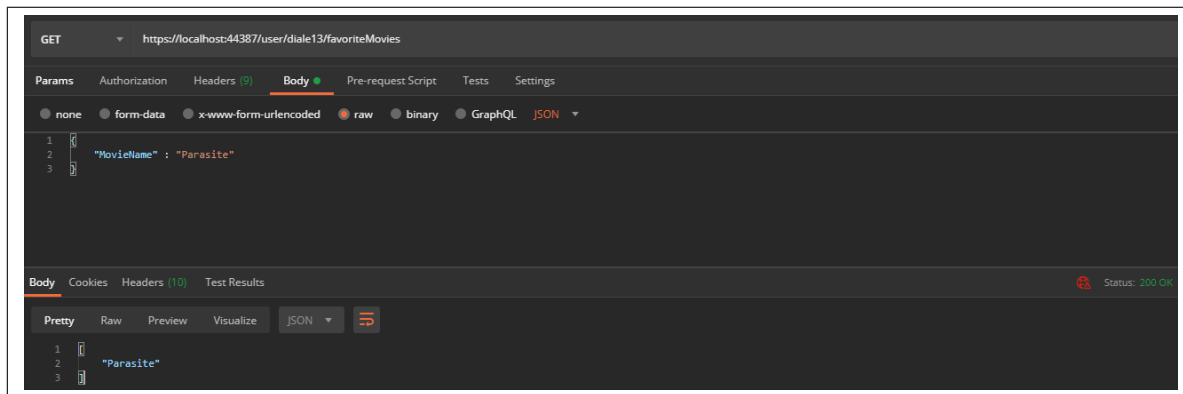


Figura 24: Ejemplo de caso

6.5. Agregar películas favoritas a otro usuario

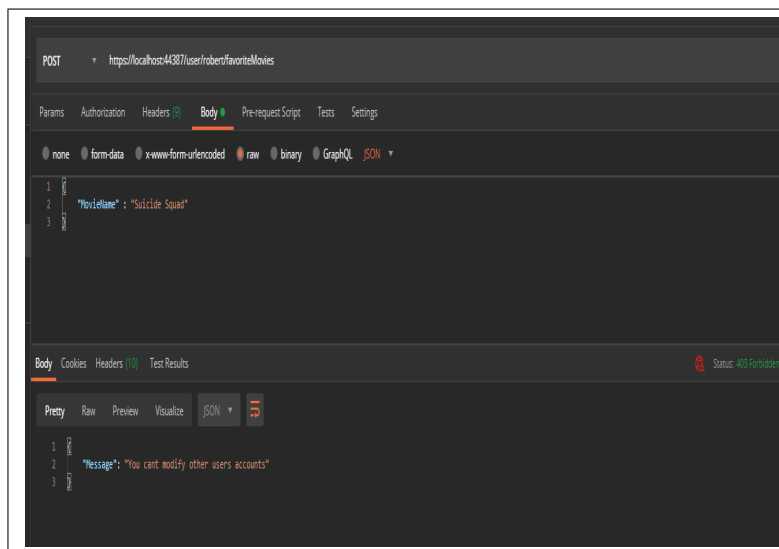


Figura 25: Retorna prohibido y mensaje acorde

6.6. Agregar películas favoritas que no existen

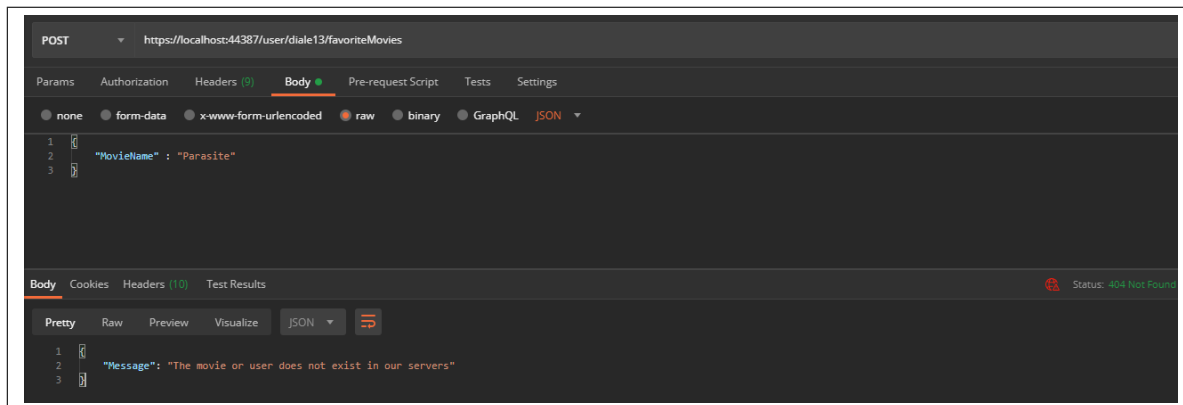


Figura 26: Retorna 404 y mensaje

7. Calificaciones

7.1. Voto correcto

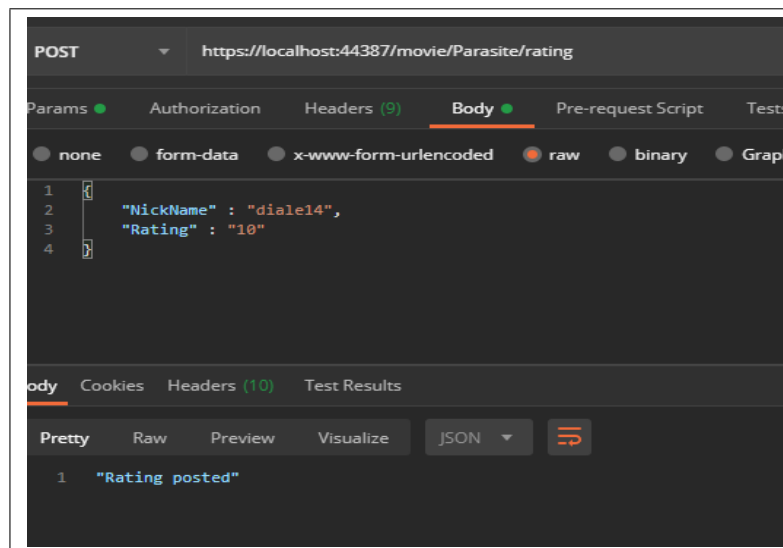


Figura 27: Retorna 404 y mensaje

7.2. Traer resultado

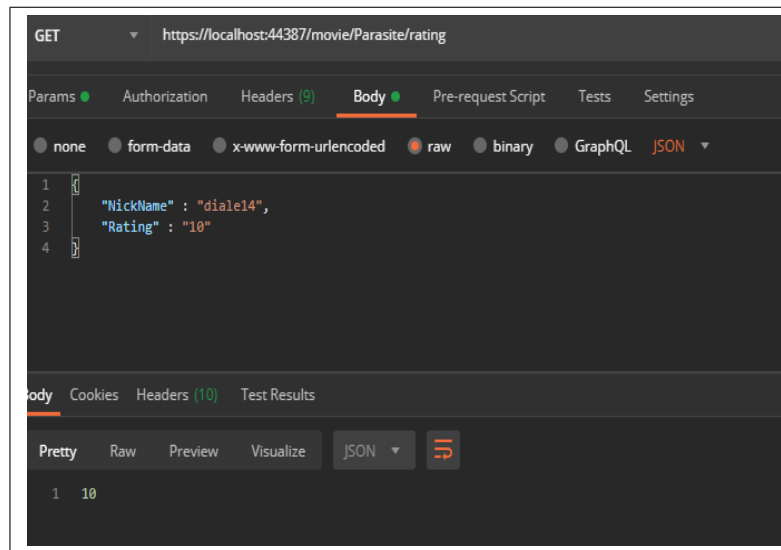


Figura 28: Retorna 404 y mensaje

7.3. Traer resultado luego de dos votos (promedio)

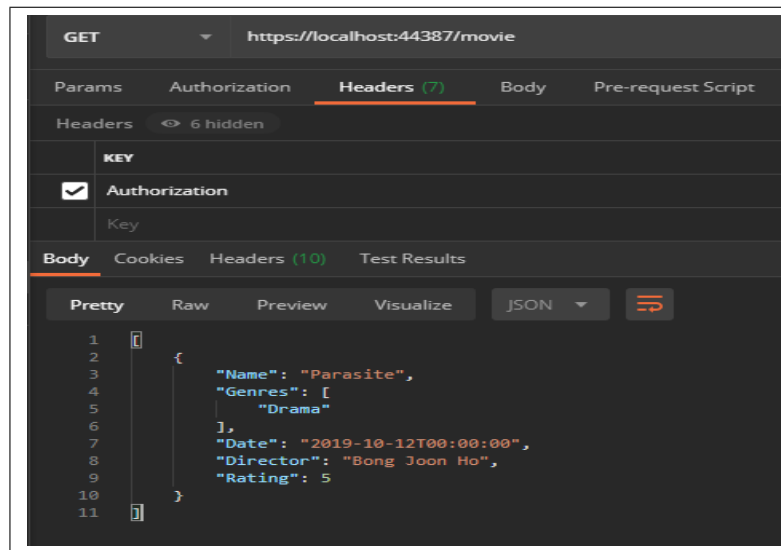


Figura 29: Actualiza correctamente el promedio

7.4. Voto fuera de rango

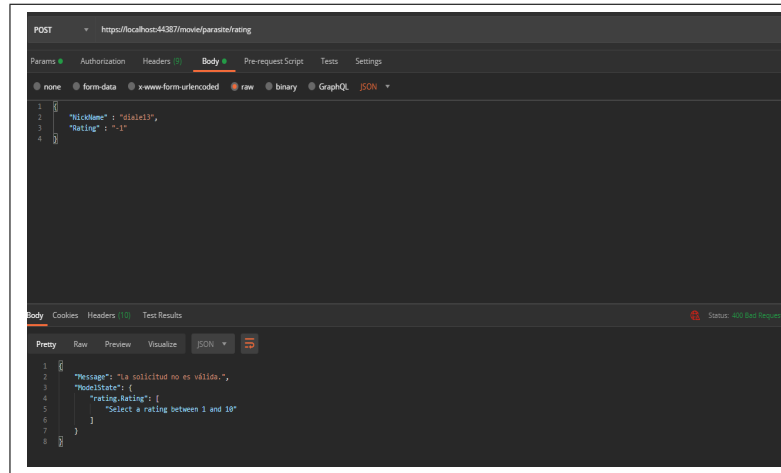


Figura 30: Retorna error en los modelos

7.5. Votar por otro

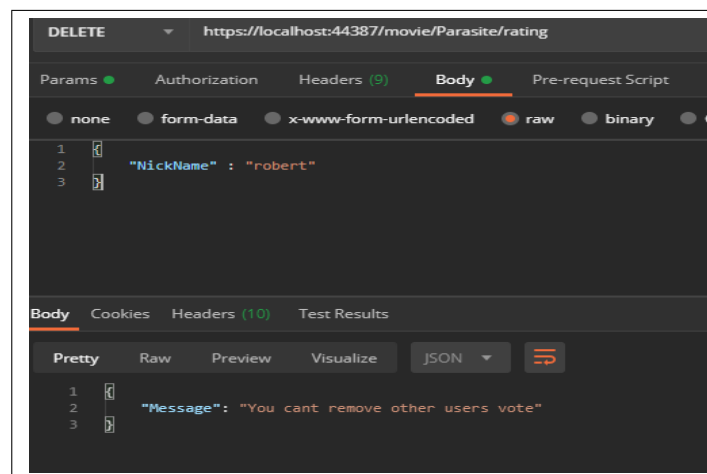


Figura 31: Retorna Forbbiden y mensaje