Universidad ORT Uruguay Facultad de Ingeniería

Obligatorio 3

173630 - Fabio Orlinski.
234912 - Federico Barrios.
154232 - Juan Viscardi.

Docentes: Andrés Soria y Gabriel Bentos

2023

N° Versión: 1

1. INTRODUCCIÓN

El presente documento describe los aspectos y decisiones más relevantes en el desarrollo de la plataforma de comunicación entre clientes y servidores. Entre los cuales pueden comunicarse entre sí de forma remota mediante web APIs, o por TCP. En ambos casos el servidor que resuelve las solicitudes genera logs de forma asincrónica enviandolos a RabbitMQ de forma de no afectar la performance de la aplicación. Además, se creó un servidor para los Logs que guarda en memoria los logs enviados por Rabbit a la cola de mensajes utilizada, y puede ser consultado de forma remota por otro servidor. Entre los endpoints de la lógica de negocio solicitados y el servidor donde se resuelve la petición, se utilizó comunicación GRPC.

Antes de utilizar la aplicación hay que instalar en la máquina donde se van a prender los servicios RabbitMQ y Erlang, en nuestro caso instalamos las siguientes versiones y las dejamos como referencia:

- RabbitMQ v13.11.17
- Erlang v25.3.2.1
- .NET 6

Se deja respaldo del código adicional aqui.

2. DESCRIPCIÓN DE LA ARQUITECTURA

Se realiza el siguiente diagrama para mostrar la arquitectura utilizada, donde el cliente se representa con uno de los posibles, en el siguiente dibujo “Postman” y los servidores que escuchan solicitudes HTTPs son AdminServer y LogServer, cada uno independiente del otro.
Para conectarse mediante TCP creamos el cliente ClientProgram.
En el caso de las solicitudes TCP del ClientProgram y las HTTPs de AdminServer van a ser resueltas por el servidor GrpcMainServer dependiendo del canal de comunicación, cuyas acciones enviará de forma asincrónica a RabbitMQ a la cola *“logs”* y serán escuchadas por LogServer y almacenadas en memoria para que poder ser consultadas de forma remota con filtros personalizados.

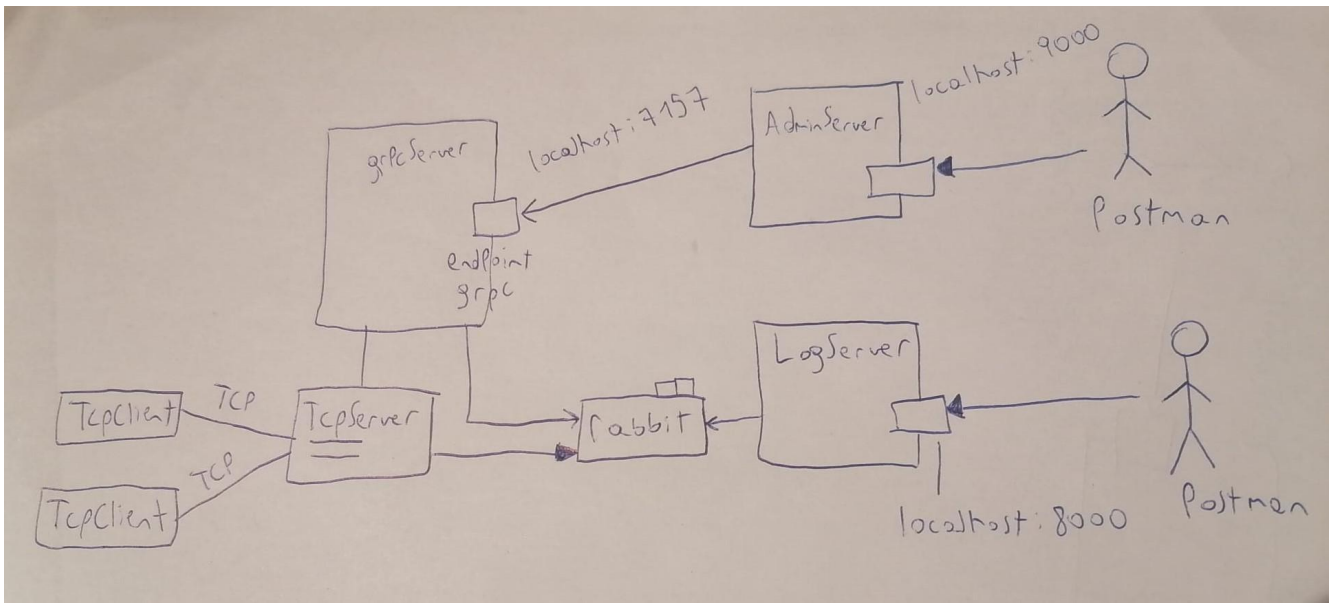
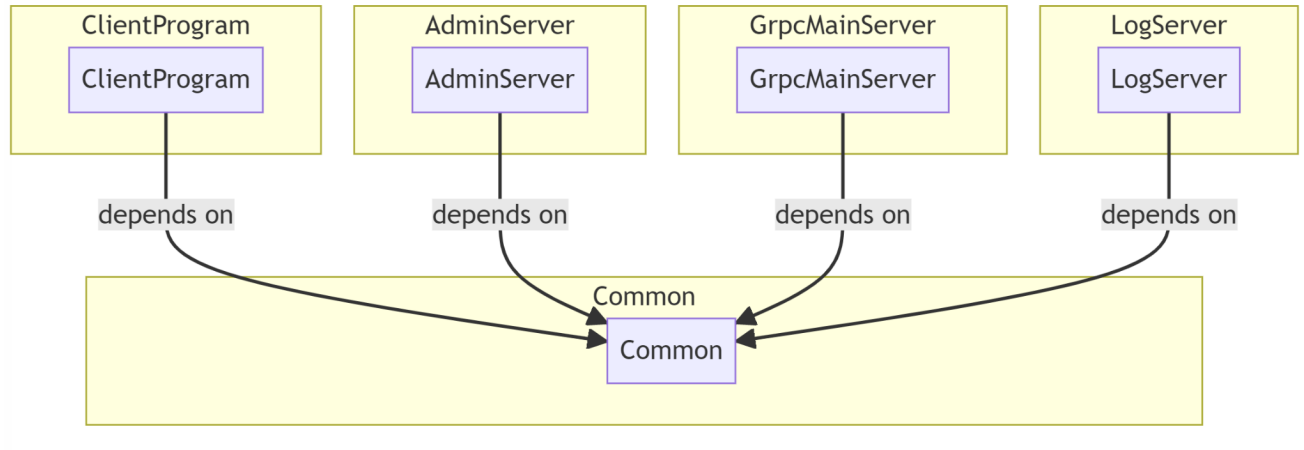
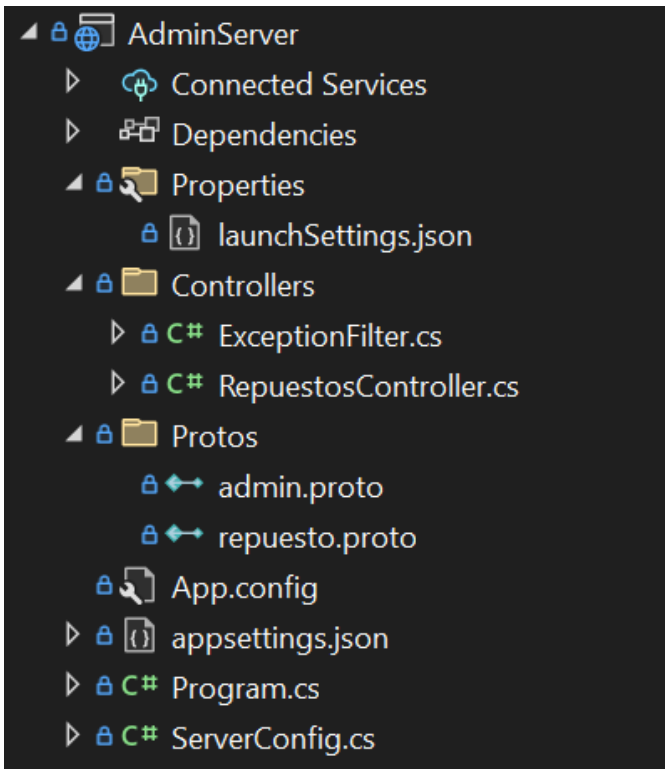


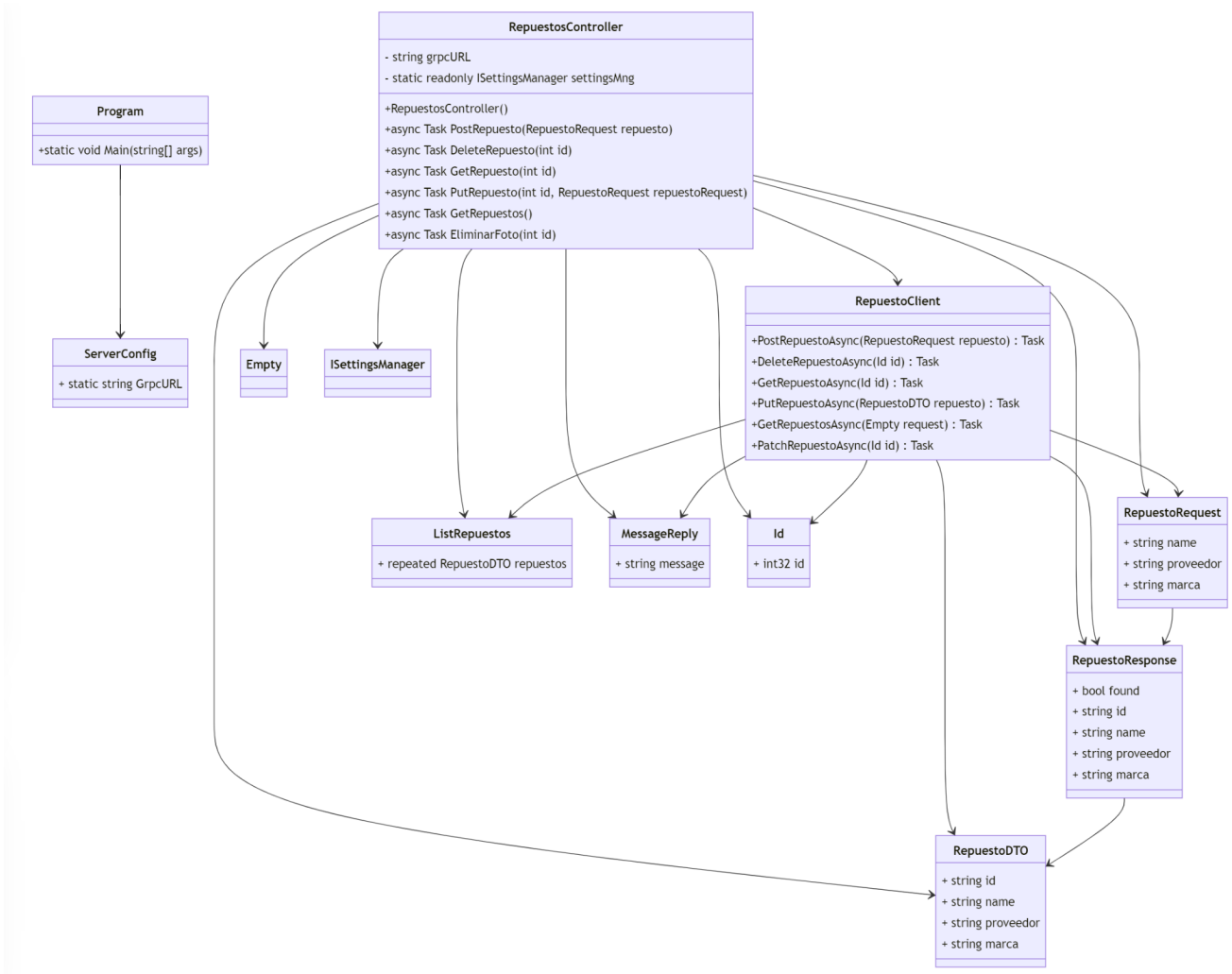
Diagrama de paquetes:



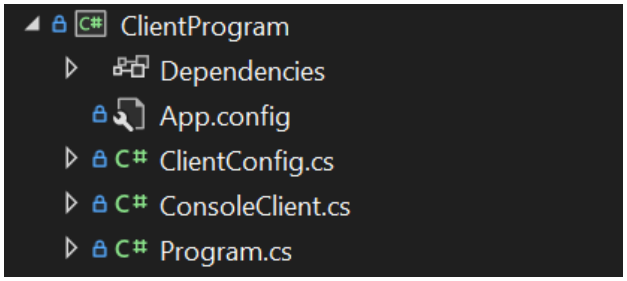
3. DOCUMENTACIÓN DE DISEÑO DETALLADA DE CADA COMPONENTE

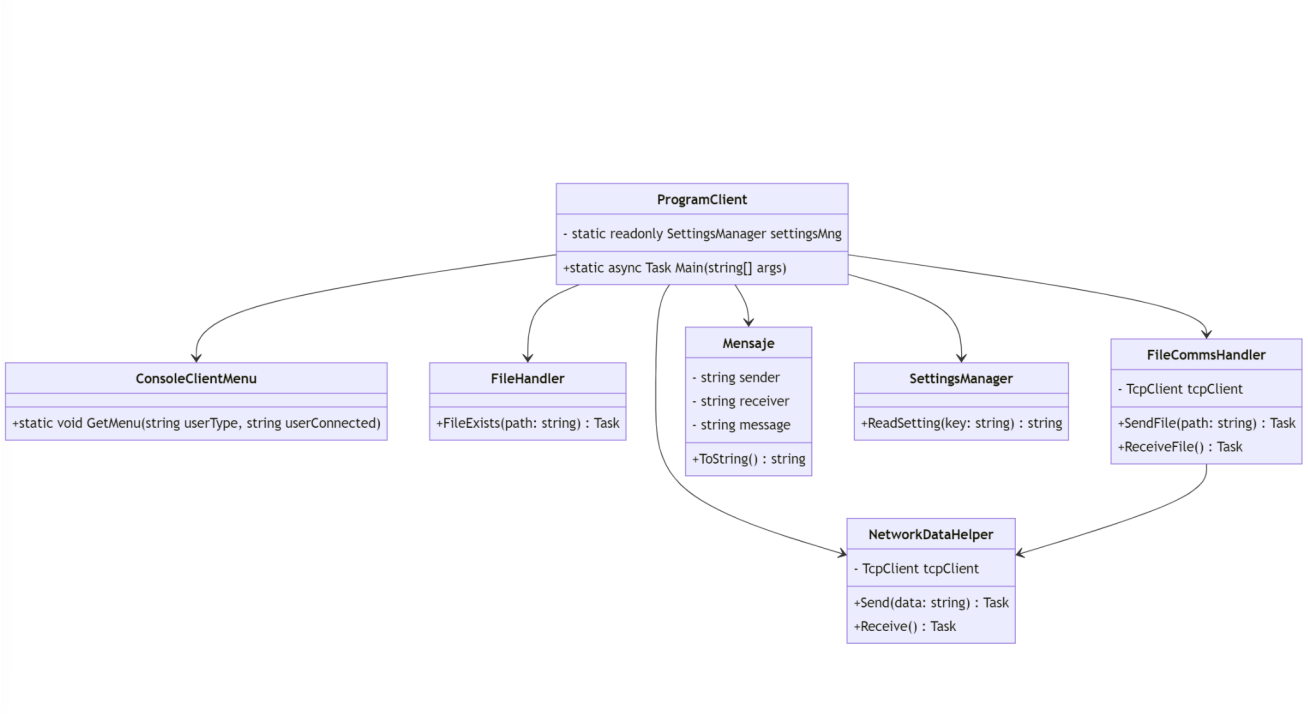
1) AdminServer


AdminServer
Connected Services
Dependencies
Properties
launchSettings.json
Controllers
ExceptionFilter.cs
RepuestosController.cs
Protos
admin.proto
repuesto.proto
App.config
appsettings.json
Program.cs
ServerConfig.cs


RepuestosController
- string grpcURL
- static readonly ISettingsManager settingsMng
+RepuestosController()
+async Task PostRepuesto(RepuestoRequest repuesto)
+async Task DeleteRepuesto(int id)
+async Task GetRepuesto(int id)
+async Task PutRepuesto(int id, RepuestoRequest repuestoRequest)
+async Task GetRepuestos()
+async Task EliminarFoto(int id)
Program
+static void Main(string[] args)
ServerConfig
+ static string GrpcURL
Empty
ISettingsManager
RepuestoClient
+PostRepuestoAsync(RepuestoRequest repuesto) : Task
+DeleteRepuestoAsync(Id id) : Task
+GetRepuestoAsync(Id id) : Task
+PutRepuestoAsync(RepuestoDTO repuesto) : Task
+GetRepuestosAsync(Empty request) : Task
+PatchRepuestoAsync(Id id) : Task
ListRepuestos
+ repeated RepuestoDTO repuestos
MessageReply
+ string message
Id
+ int32 id
RepuestoRequest
+ string name
+ string proveedor
+ string marca
RepuestoResponse
+ bool found
+ string id
+ string name
+ string proveedor
+ string marca
RepuestoDTO
+ string id
+ string name
+ string proveedor
+ string marca

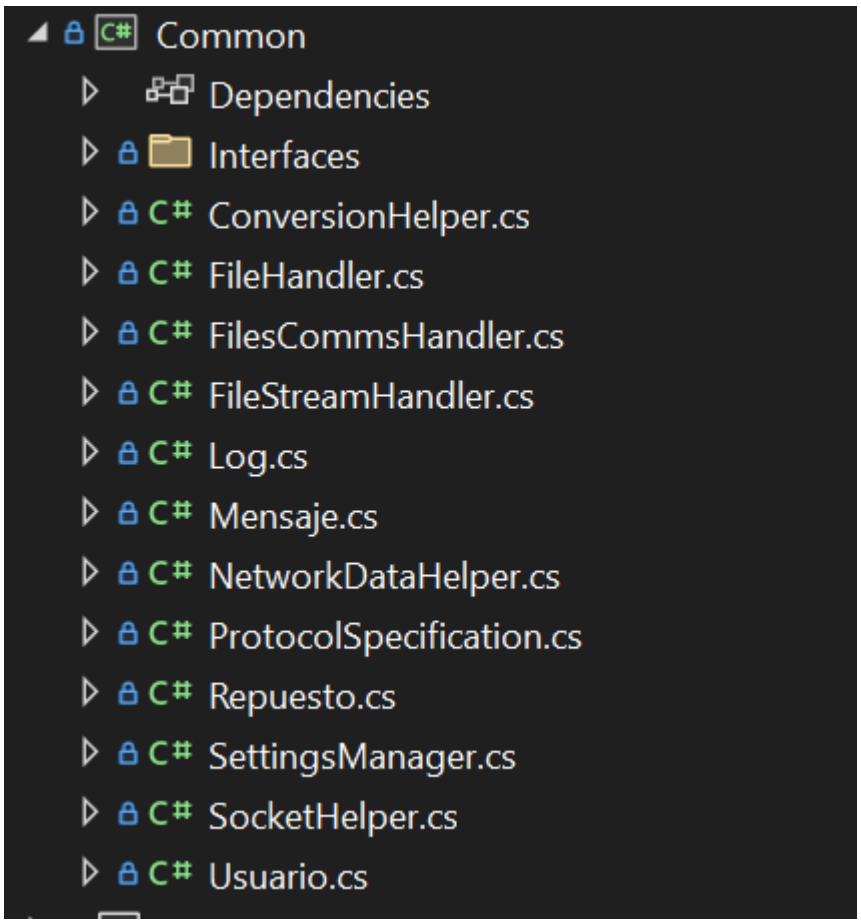
2) ClientProgram



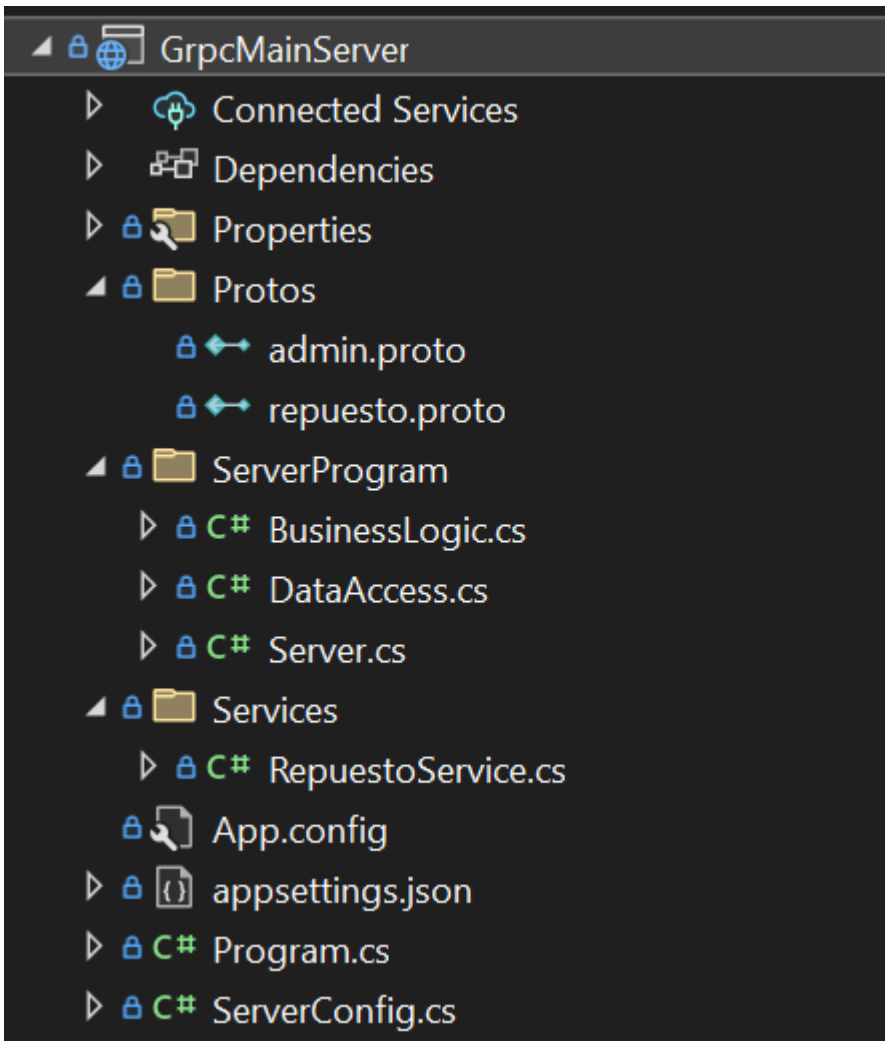


3) Common

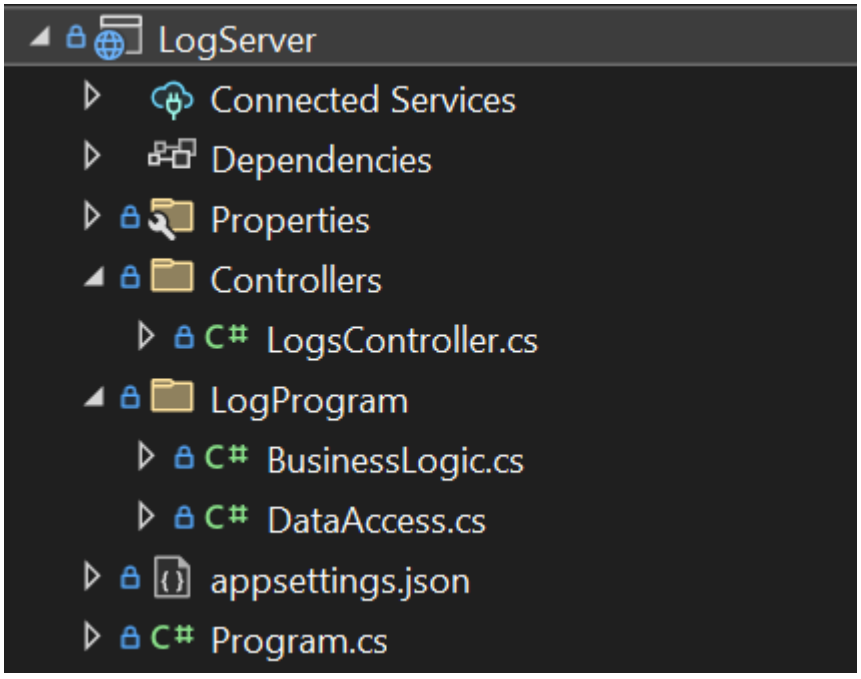
Aquí se implementan las clases que serán utilizadas en los demás paquetes.



4) GrpcMainServer


GrpcMainServer
Connected Services
Dependencies
Properties
Protos
admin.proto
repuesto.proto
ServerProgram
BusinessLogic.cs
DataAccess.cs
Server.cs
Services
RepuestoService.cs
App.config
appsettings.json
Program.cs
ServerConfig.cs

5) LogServer


LogServer
Connected Services
Dependencies
Properties
Controllers
LogsController.cs
LogProgram
BusinessLogic.cs
DataAccess.cs
appsettings.json
Program.cs

4. MECANISMOS DE COMUNICACIÓN DE LOS COMPONENTES EN NUESTRA SOLUCIÓN

4.1. Servidor de logs. (tecnología RabbitMQ)

Se usa utiliza este camino, rabbitMQ, ya que se espera que el manejo de logs sea asincrónico.

- **Implementa SLRF1**. Recepción de logs desde el servidor principal.
- **Implementa SLRF2.** Filtrado de logs. El filtrado puede ser: Tipo de evento u Hora
- **Implementa SLRF3.** Acceso remoto. El servidor de logs debe exponer un servicio que permita acceder a los logs filtrados de manera remota. El log podrá ser accedido mediante conexión rabbitMQ ya que así se implementó siendo asincrónica.

4.2. Servidor administrativo (tecnología gRPC)

Se utiliza esta tecnología, gRPC, ya que la administración de repuesto debe ser inmediata, sincrónica.

- **Implementa SARF1**. ABM de Repuesto. Se deberá dar de alta, baja o modificación de repuestos tomando en cuenta las restricciones ya existentes para los obligatorios anteriores.
- **Implementa SARF2**. Eliminar una foto. Se deberá poder eliminar la foto de un repuesto dado.
- **Implementa SARF3**. Acceso remoto. El servidor administrativo expone servicio que permite acceder a las funcionalidades administrativas anteriores. El servidor podrá ser accedido mediante conexión gRPC ya que así se implementó siendo sincronica.

4.3. Servidor GrpcMainServer conteniendo:

- Solución tcp servidor
- Solución tcp cliente
- Provee conexión al servidor Administrativo mediante gRPC
- Provee conexión al servidor logs mediante rabbitMQ

La elección de utilizar GRPC con protos para la comunicación entre el paquete de .NET y el servidor tiene varias ventajas significativas. En primer lugar, GRPC es un framework de comunicación de alto rendimiento que permite la transferencia eficiente de datos entre sistemas distribuidos. Al utilizar GRPC, podemos aprovechar su enfoque basado en ***protocol buffers*** para definir los mensajes y servicios que se intercambiarán entre el cliente y el servidor. Esto proporciona una forma sencilla y legible de definir la estructura de los datos y las operaciones que se pueden realizar.

Además, GRPC ofrece soporte para varios lenguajes de programación, lo que nos brinda flexibilidad para desarrollar componentes en diferentes tecnologías, manteniendo una comunicación consistente y eficiente entre ellos. En este caso, al utilizar .NET en el cliente y el servidor, podemos aprovechar las bibliotecas y herramientas de GRPC específicas para este lenguaje, lo que facilita la implementación y el mantenimiento del sistema.

Por otro lado, la integración con RabbitMQ para el envío de logs desde el servidor a un LogServer tiene sus propias ventajas. RabbitMQ es un sistema de mensajería robusto y escalable que se basa en el protocolo de ***mensajería AMQP (Advanced Message Queuing Protocol)***. Al utilizar RabbitMQ, podemos separar la generación de logs en el servidor de su procesamiento y almacenamiento en el LogServer. Esto nos permite desacoplar estos componentes y escalarlos de forma independiente

según sea necesario. Además, RabbitMQ proporciona características como la durabilidad de los mensajes y la capacidad de encolar y distribuir los logs de manera eficiente, lo que garantiza una entrega confiable y la posibilidad de procesar los logs en paralelo.

El uso de un controller remoto para acceder a los logs desde el LogServer también es beneficioso. Al exponer un controller remoto, podemos acceder a los logs de forma segura y conveniente desde cualquier lugar, lo que facilita la supervisión y el análisis de los registros del sistema. Además, al utilizar GRPC para esta comunicación, podemos aprovechar sus capacidades de serialización eficiente y su soporte para transmisiones de datos en tiempo real, lo que nos permite acceder a los logs en tiempo real y realizar operaciones como filtrado y búsqueda de forma eficiente.

En resumen, la elección de utilizar GRPC con protos para la comunicación entre el paquete de .NET y el servidor, junto con la integración de RabbitMQ para el envío de logs y el uso de un controller remoto para acceder a los logs, proporciona un enfoque escalable, eficiente y flexible para el intercambio de datos y la supervisión del sistema. Estas tecnologías y métodos de comunicación nos permiten construir un sistema distribuido robusto, modular y fácil de mantener.

5. ARCHIVOS DE CONFIGURACIÓN

5.1 Archivos de configuración

A continuación detallaremos los archivos de configuración que deben ser configurados para utilizar la configuración, por defecto dejamos los valores que presentaremos a continuación. Cabe mencionar que los valores almacenados en archivos *App.Config* pueden ser alterados en tiempo de ejecución mientras que los *launchSettings.json* deben ser configurados antes de iniciarla.

5.1.1. Paquete AdminServer:

App.Config:

Aquí se guarda el url del servidor GRPC.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="GrpcURL" value="https://localhost:7157"/>
        </appSettings>
</configuration>
```

launchSettings.json:

Aquí se guarda el url donde se van a exponer las APIs.

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
```

```
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:17819",
      "sslPort": 44374
    }
  },
  "profiles": {
    "AdminServer": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:9000;http://localhost:9001",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

5.1.2 Paquete ClientProgram:

Aquí se guarda el IP y Puerto del servidor donde se harán solicitudes TCP para que el cliente pueda acceder.

App.Config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="ServerAddress" value="127.0.0.1"/>
                <add key="ServerPort" value="20000"/>
                <add key="ClientAddress" value="127.0.0.1"/>
                <add key="ClientPort" value="0"/>
        </appSettings>
</configuration>
```

5.1.3. Paquete GrpcMainServer:

Aquí se guarda el IP y Puerto del servidor donde se harán solicitudes TCP para que todos los clientes puedan acceder. Además se fija un usuario y contraseña para que se pueda ingresar como

administrador a la aplicación y crear mecánicos.

App.Config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="ServerAddress" value="127.0.0.1"/>
                <add key="ServerPort" value="20000"/>
                <add key="UsernameConfigKey" value="admin"/>
                <add key="PasswordConfigKey" value="admin123"/>
        </appSettings>
</configuration>
```

launchSettings.json:

Aquí se deja el url para que se pueda acceder por GRPC.

```
{
  "profiles": {
    "GrpcMainServer": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5240;https://localhost:7157",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

5.1.4. Paquete LogServer:

Aquí se fija el URL para poder recibir APIs al servidor de Logs.

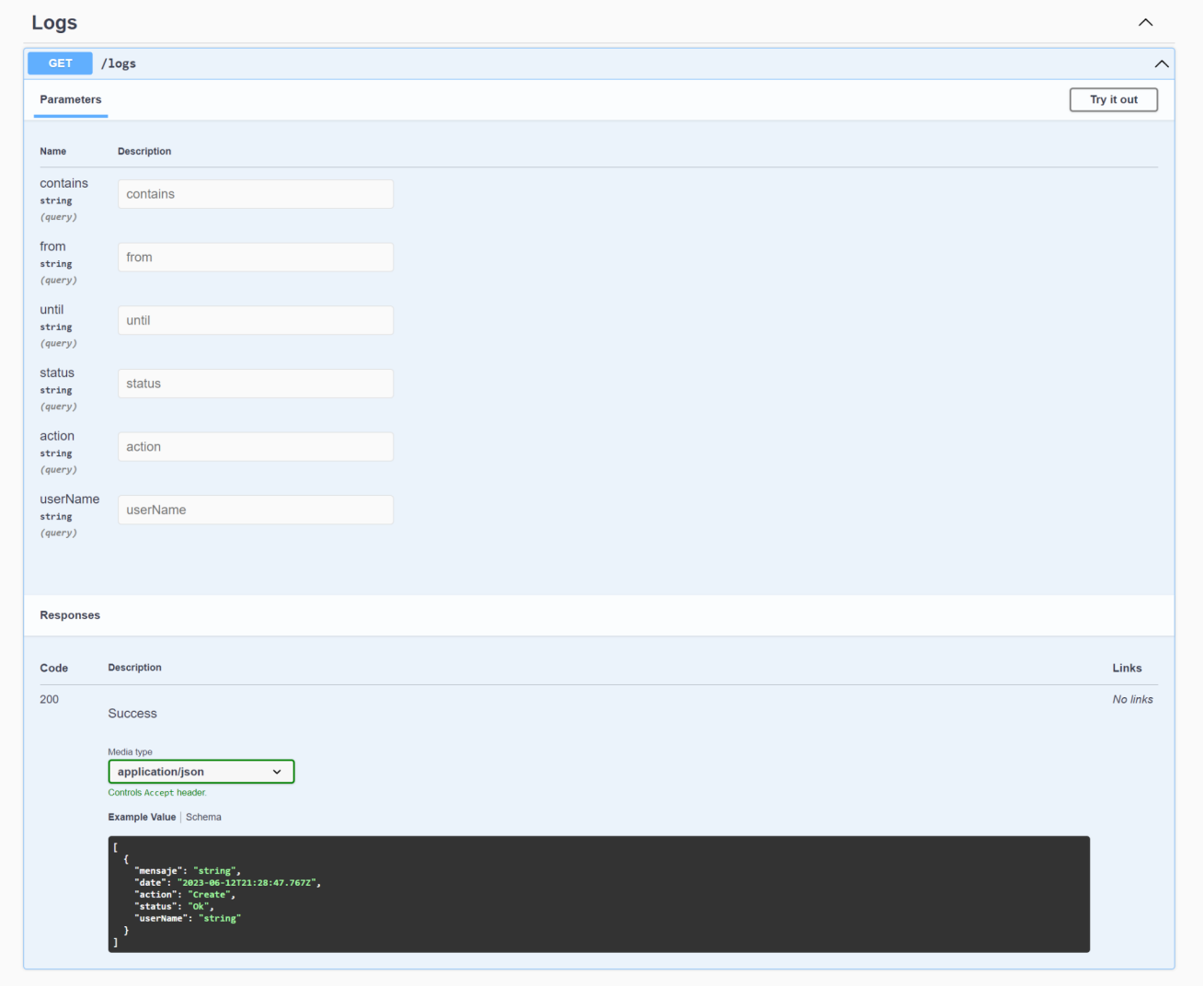
launchSettings.json:

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:15858",
      "sslPort": 44337
    }
  },
  "profiles": {
    "LogServer": {
      "commandName": "Project",
```

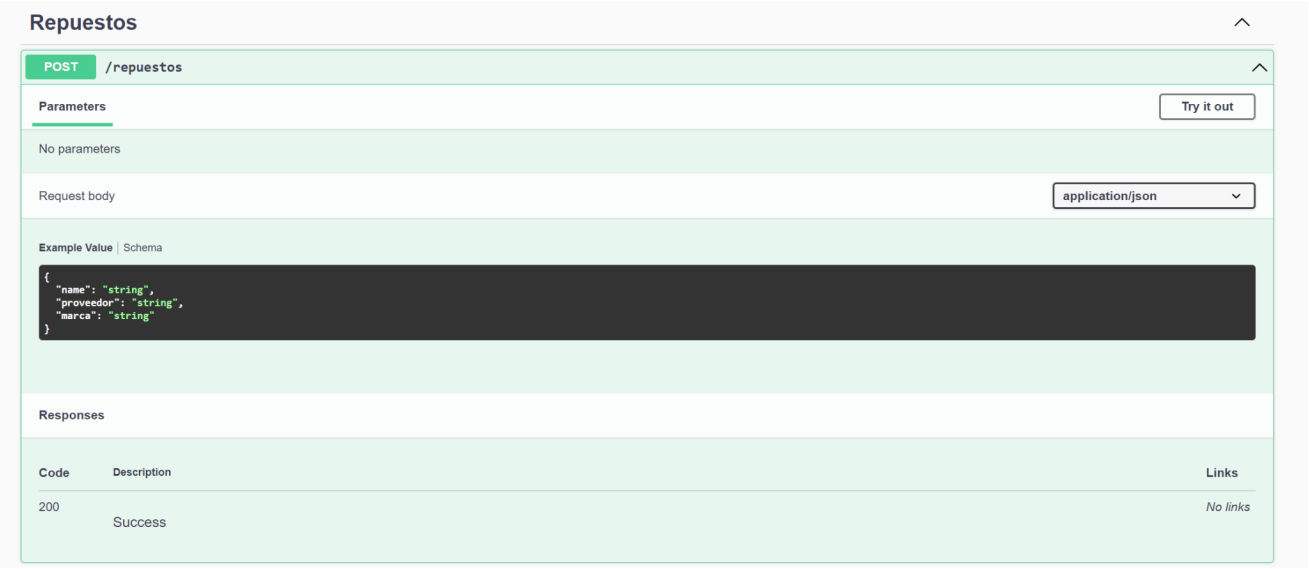
```
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:8000;http://localhost:8001",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

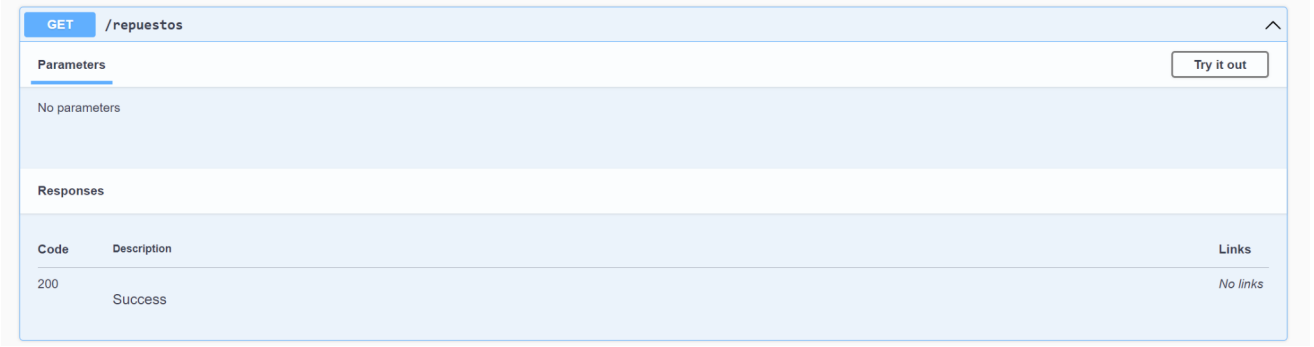
5.2 Endpoints

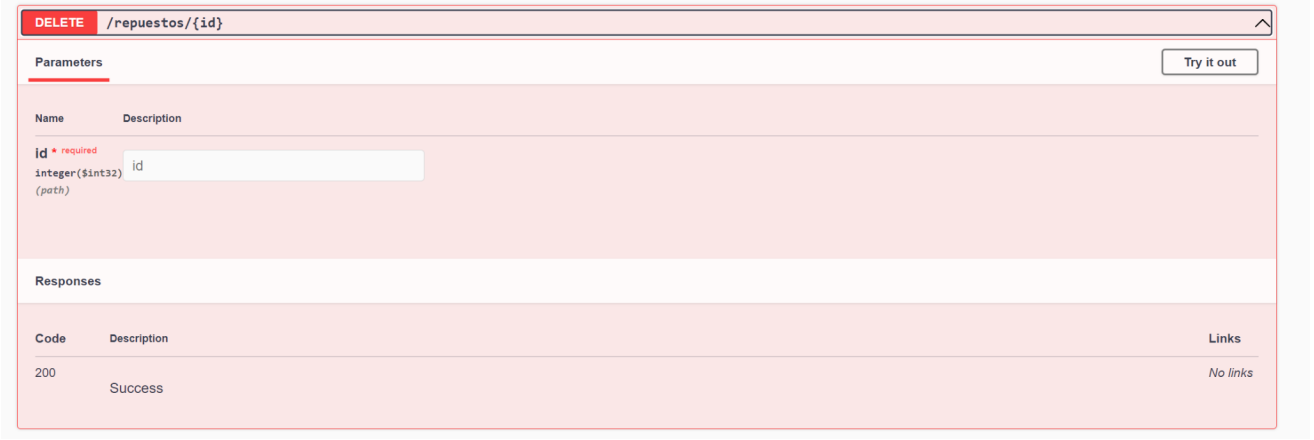
5.2.1 LogServer

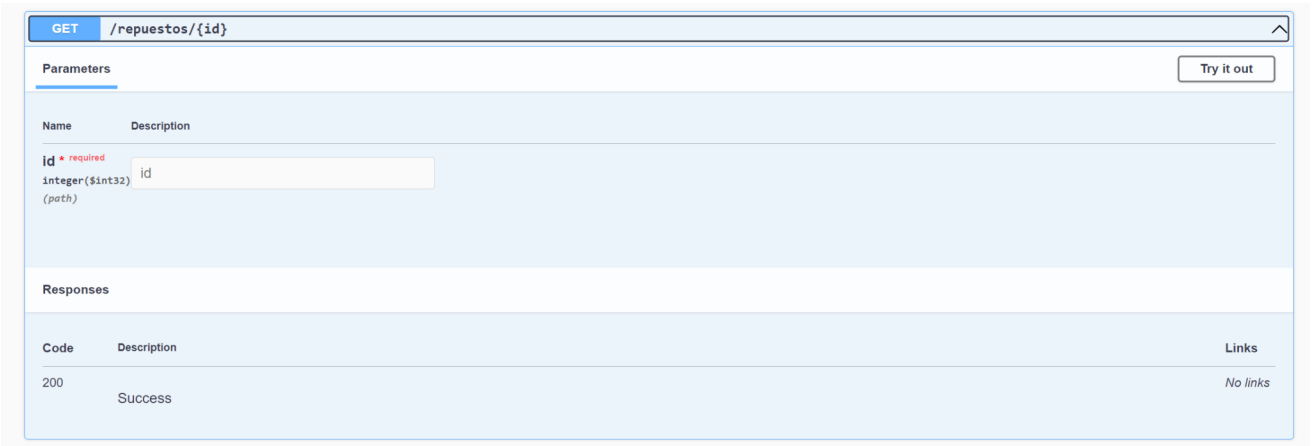


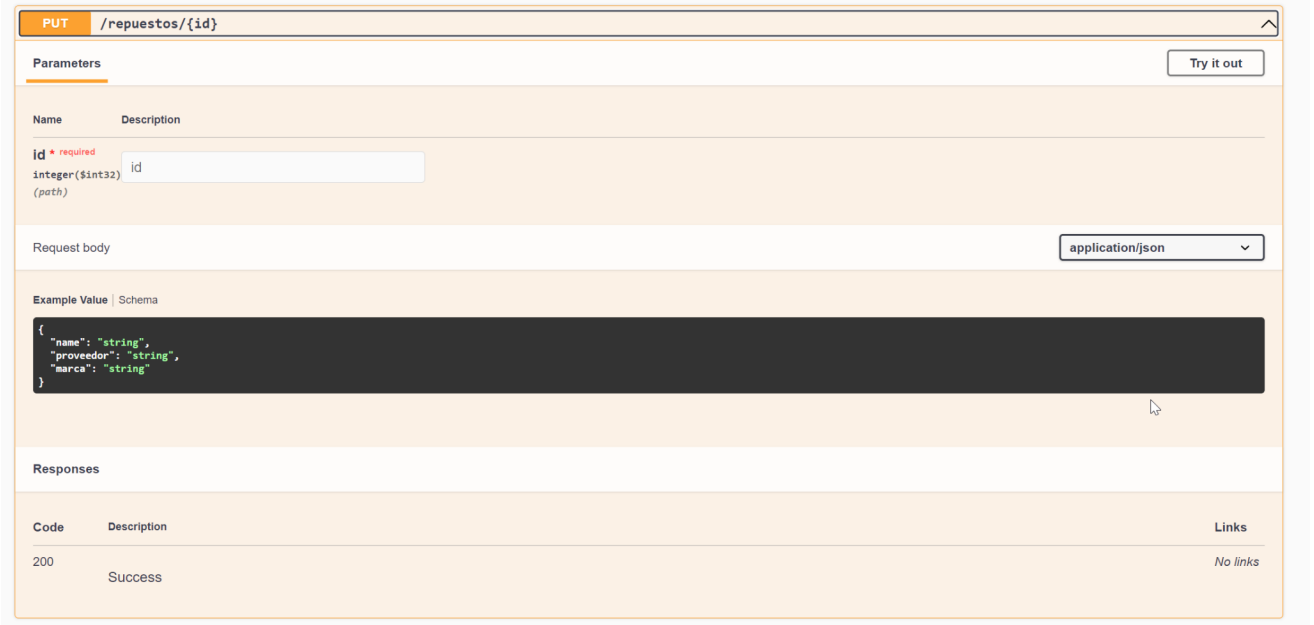
5.2.2. AdminServer

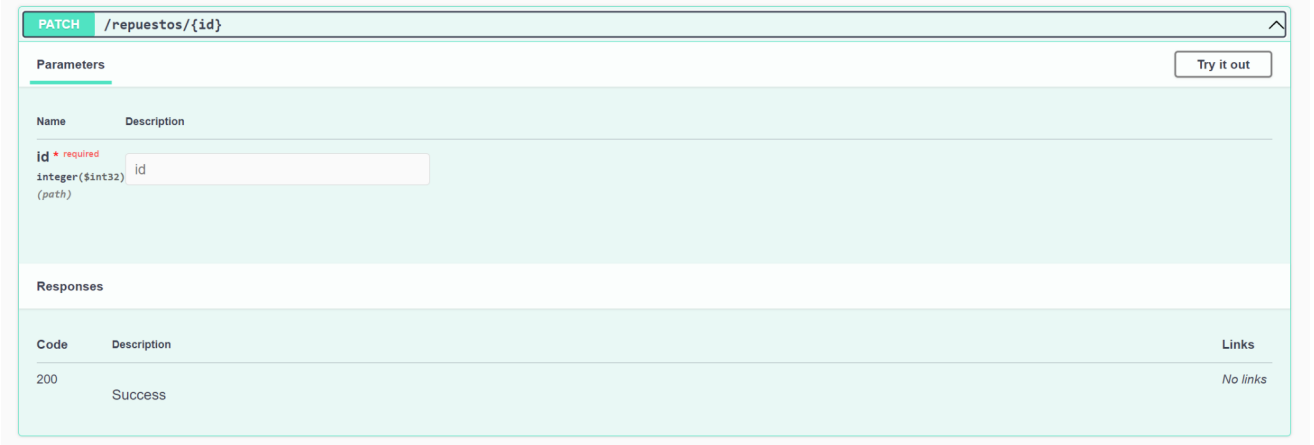



GET
/repuestos
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Success
No links


DELETE
/repuestos/{id}
Parameters
Try it out
Name
Description
id
* required
integer($int32)
(path)
id
Responses
Code
Description
Links
200
Success
No links


GET
/repuestos/{id}
Parameters
Try it out
Name
Description
id
* required
integer($int32)
(path)
id
Responses
Code
Description
Links
200
Success
No links





Nota: Todos estas APIs pueden también devolver Status code 404 Not Found cuando corresponda.

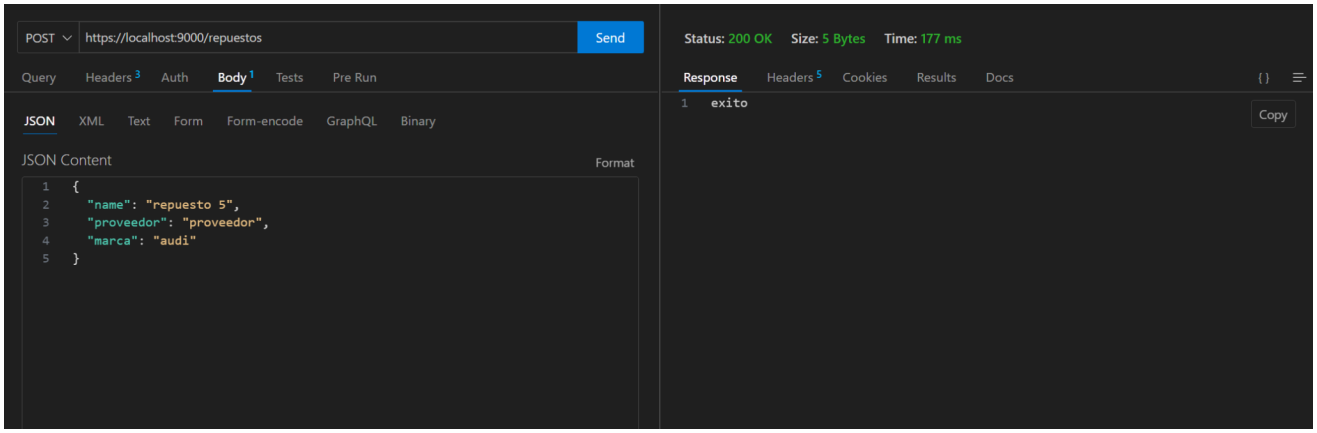
6. SUPUESTOS

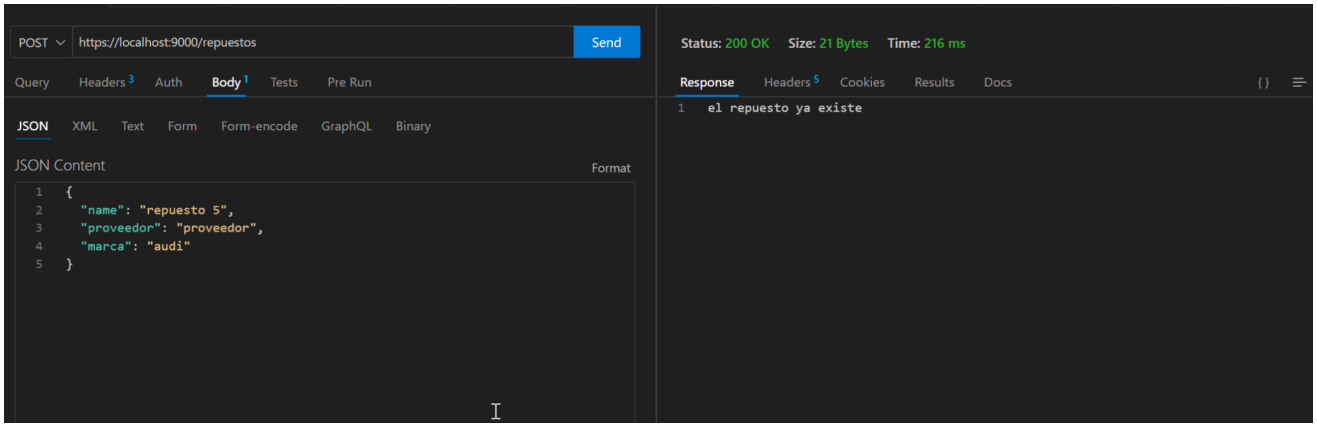
Eliminamos un supuesto anteriormente asumido y restrictivo agregando ID a repuesto y de esta manera aseguramos el correcto funcionamiento del ABM repuesto

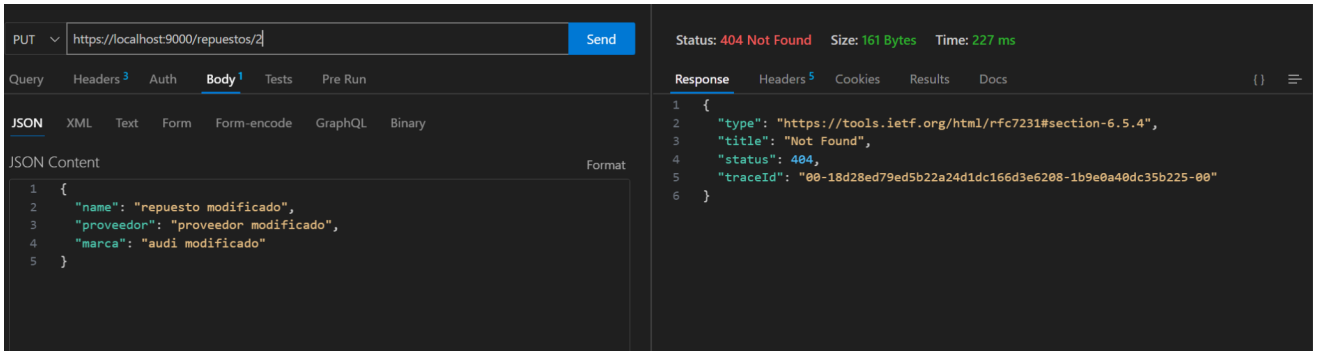
7. POSIBLES MEJORAS

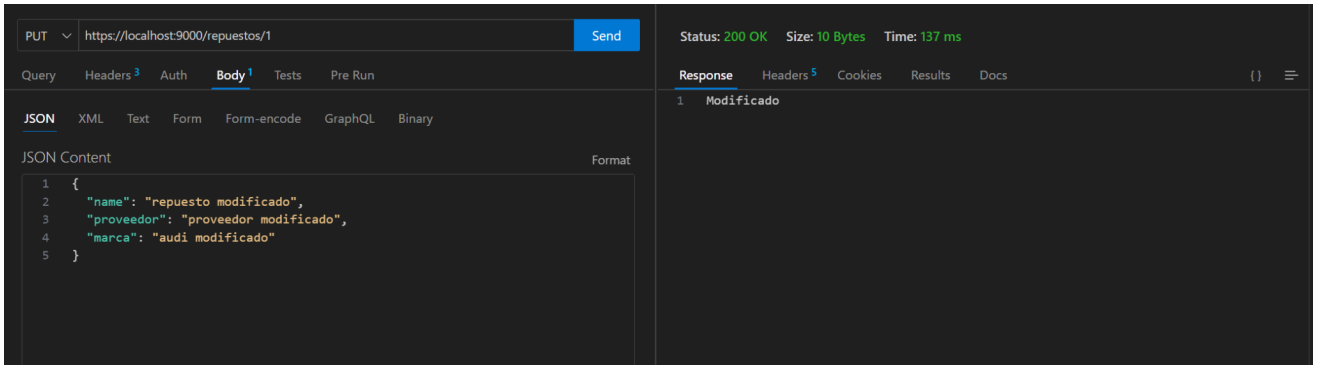
- Devolver status code más detallados. Por ejemplo para la creación de nuevos repuestos con el method POST se devuelve 200 en vez de 201 cómo sería ideal.

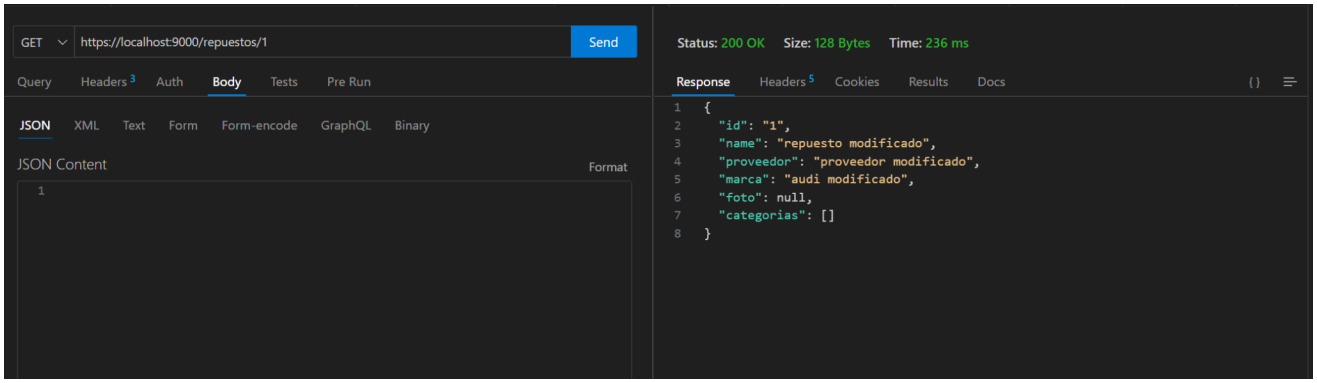
8. EJEMPLOS DE USO

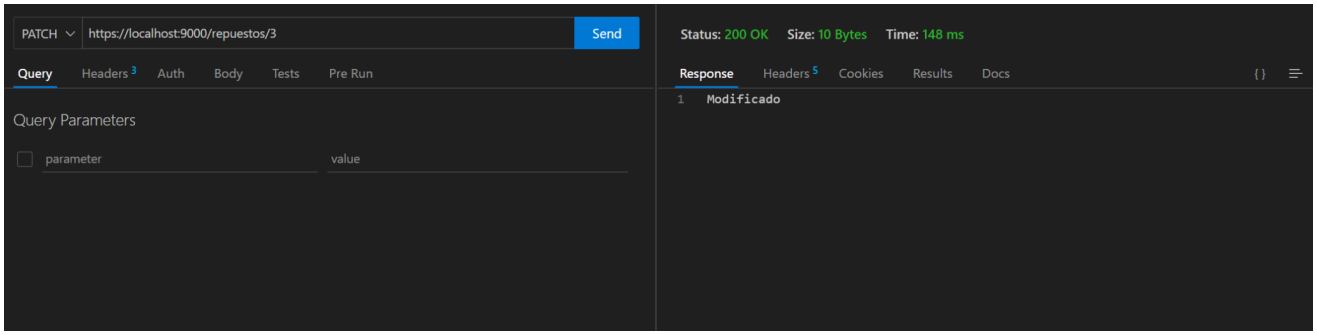


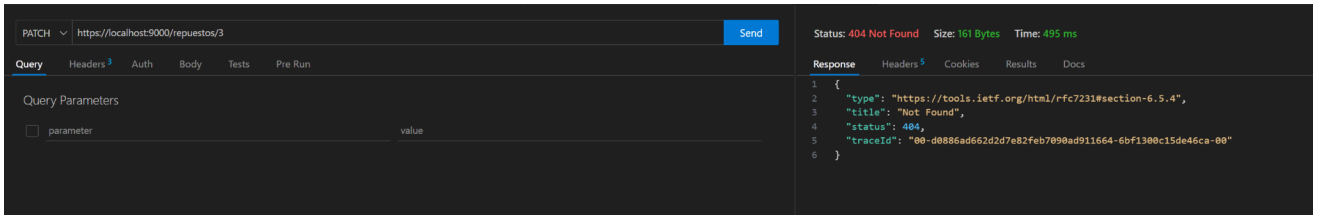


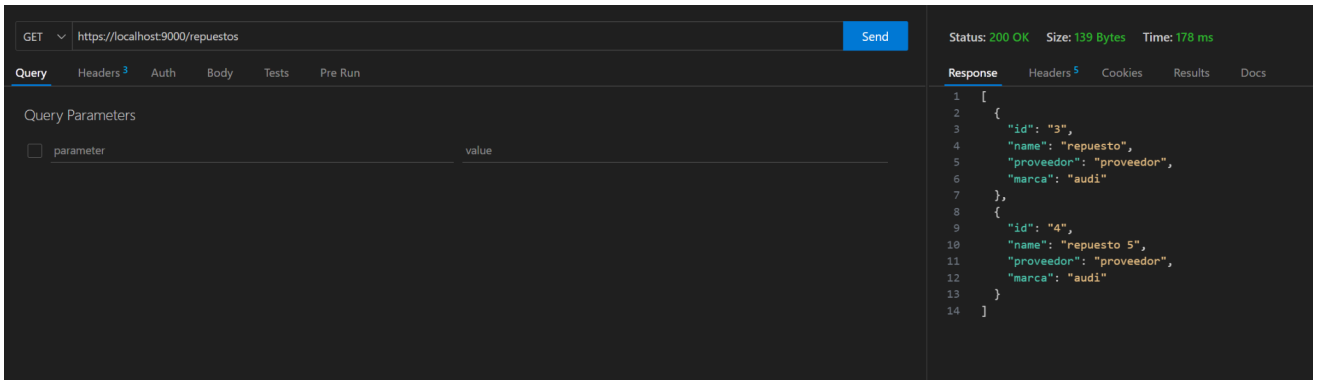


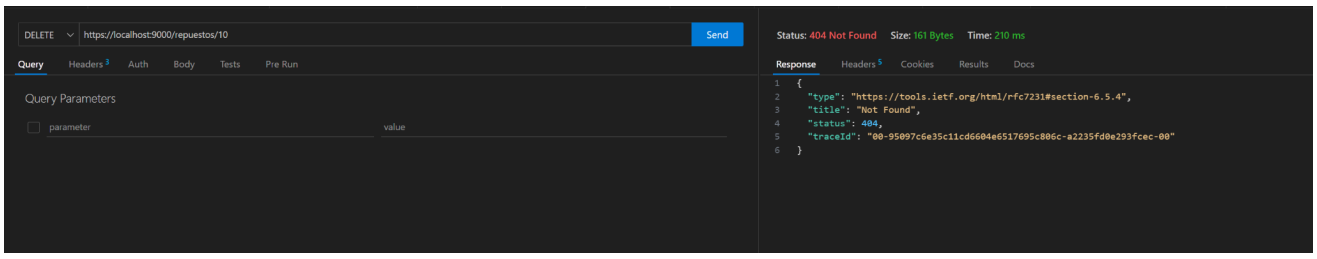


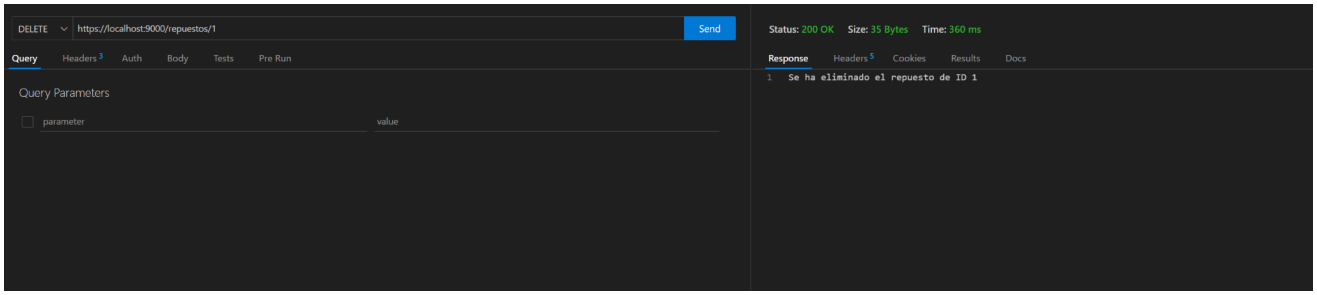

GET
https://localhost:9000/repuestos/1
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 128 Bytes
Time: 236 ms
Response
Headers 5
Cookies
Results
Docs
{
"id": "1",
"name": "repuesto modificado",
"proveedor": "proveedor modificado",
"marca": "audi modificado",
"foto": null,
"categorias": []
}

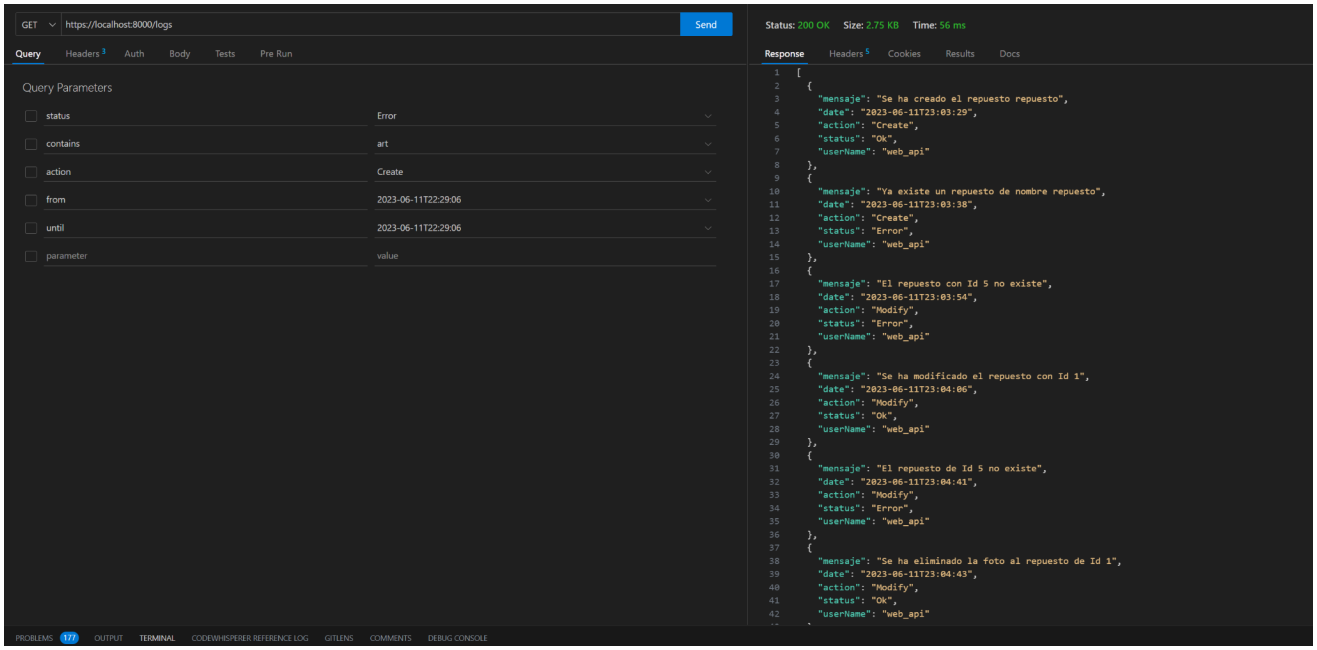

PATCH
https://localhost:9000/repuestos/3
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
parameter
value
Status: 200 OK
Size: 10 Bytes
Time: 148 ms
Response
Headers 5
Cookies
Results
Docs
Modificado


PATCH
https://localhost:9000/repuestos/3
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
parameter
value
Status: 404 Not Found
Size: 161 Bytes
Time: 495 ms
Response
Headers 5
Cookies
Results
Docs
{
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
"title": "Not Found",
"status": 404,
"traceId": "00-d0886ad662d2d7e82feb7090ad911664-6bf1300c15de46ca-00"
}


GET
https://localhost:9000/repuestos
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
parameter
value
Status: 200 OK
Size: 139 Bytes
Time: 178 ms
Response
Headers 5
Cookies
Results
Docs
[
{
"id": "3",
"name": "repuesto",
"proveedor": "proveedor",
"marca": "audi"
},
{
"id": "4",
"name": "repuesto 5",
"proveedor": "proveedor",
"marca": "audi"
}
]


DELETE
https://localhost:9000/repuestos/10
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
parameter
value
Status: 404 Not Found
Size: 161 Bytes
Time: 210 ms
Response
Headers 5
Cookies
Results
Docs
{
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
"title": "Not Found",
"status": 404,
"traceId": "00-95097c6e35c11cd6604e6517695c806c-a2235fd0e293fcec-00"
}


DELETE
https://localhost:9000/repuestos/1
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
parameter
value
Status: 200 OK
Size: 35 Bytes
Time: 360 ms
Response
Headers 5
Cookies
Results
Docs
1 Se ha eliminado el repuesto de ID 1


GET
https://localhost:8000/logs
Send
Query
Headers 3
Auth
Body
Tests
Pre Run
Query Parameters
status
Error
contains
art
action
Create
from
2023-06-11T22:29:06
until
2023-06-11T22:29:06
parameter
value
Status: 200 OK
Size: 2.75 KB
Time: 56 ms
Response
Headers 5
Cookies
Results
Docs
[
{
"mensaje": "Se ha creado el repuesto repuesto",
"date": "2023-06-11T23:03:29",
"action": "Create",
"status": "Ok",
"userName": "web_api"
},
{
"mensaje": "Ya existe un repuesto de nombre repuesto",
"date": "2023-06-11T23:03:38",
"action": "Create",
"status": "Error",
"userName": "web_api"
},
{
"mensaje": "El repuesto con Id 5 no existe",
"date": "2023-06-11T23:03:54",
"action": "Modify",
"status": "Error",
"userName": "web_api"
},
{
"mensaje": "Se ha modificado el repuesto con Id 1",
"date": "2023-06-11T23:04:06",
"action": "Modify",
"status": "Ok",
"userName": "web_api"
},
{
"mensaje": "El repuesto de Id 5 no existe",
"date": "2023-06-11T23:04:41",
"action": "Modify",
"status": "Error",
"userName": "web_api"
},
{
"mensaje": "Se ha eliminado la foto al repuesto de Id 1",
"date": "2023-06-11T23:04:43",
"action": "Modify",
"status": "Ok",
"userName": "web_api"
PROBLEMS 177
OUTPUT
TERMINAL
CODEWHISPERER REFERENCE LOG
GITLENS
COMMENTS
DEBUG CONSOLE