Meddra Gateway

1. Código fuente

El código fuente del Gateway está en su totalidad escrito con javascript sobre nodejs, para que se pueda integrar en DHIS2 es necesario utilizar un formulario personalizado, debido a que DHIS2 no cuenta con un proceso nativo para integrarse a APIS externas.

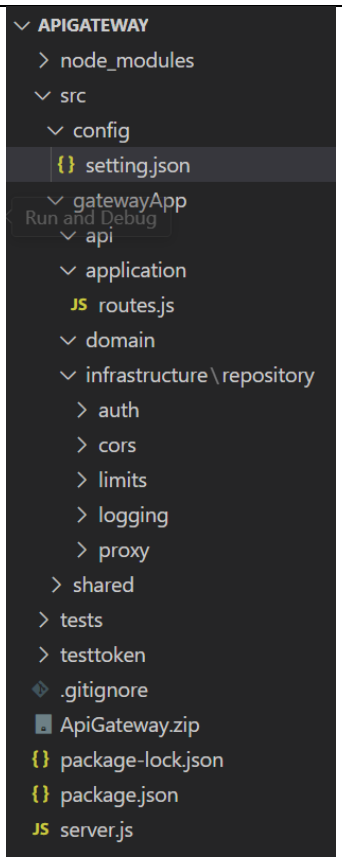
Para la construcción del Gateway se utilizó los lenguajes a continuación.

- Javascript sobre nodejs (versión 16.15), dependencias.
    - "axios": "^1.3.4"
    - "cors": "^2.8.5"
    - "express": "^4.18.2"
    - "express-rate-limit": "^5.2.6"
    - "express-session": "^1.17.3"
    - "http-proxy-middleware": "^1.3.1"
    - "keycloak-connect": "^12.0.4"
    - "morgan": "^1.10.0"

La estructura del proyecto Gateway se conforma de la siguiente manera:

Todo el código se encuentra en la carpeta src, en ella se las carpetas más importantes son:config, application, intrastructure.

1. Config, contiene el archivo de configuración setting.json, en él se define el servidor de DHIS2 y el token de autenticación, el servicio de autenticación con meddra y las políticas de acceso CORS al servicio.
2. Application, contiene las rutas del API construida en el archivo routers.js, se define la url que se asignará, si requiere autenticación para su consumo y el método HTTP junto al manejaron que se expondrá. El método HTTP y el manejado se encuentran implementador en el middleware proxy dentro de infraestructure.
3. Infraestructure contiene todos los middlewares express que constituyen la apiGateway que son:
    a. Auth, el servicio de autenticación en este caso se utiliza DHIS2 para autenticarse
    b. Cors, contiene las políticas de acceso a la apiGateway, o sea que dirección ip puede utilizar el servicio.
    c. Limits, maneja la cantidad de conexiones concurrentes que se pueden tener al servicio
    d. Logging, maneja la generación del logs utilizando Morgan
    e. Proxy, se encarga de redireccionar y ocultar los mecanismos de acceso y autenticación de meddra, de forma que hacia el cliente solo observará la url definida en router.



2. Configuración

La configuración se hace en el archivo setting.json, el archivo tiene la siguiente estructura

```
{
    "meddra":{
        "url":"https://mapisbx.meddra.org/api/search",
        "url_token":"https://mid.meddra.org/connect/token",
        "client_id":"mcpclient",
        "username":"23670",
        "app_secret":"278Xp345fAFTdsqwrrlkz90",
        "scope":"meddraapi",
        "token":""
    },
    "dhis2":{
        "url":"https://datos.hispcolombia.org/dhis",
        "url_token":"https://datos.hispcolombia.org/dhis/uaa/oauth/t
oken",
        "name":"helder",
        "client_id":"helder",
        "client_secret":"47e09946a-d503-06ae-6690-dx32f30c91c"
    },
    "cgip":{
        "url":"www.cgip",
        "user":"abc",
        "password":"xyz"
    },
    "allowedOrigins":["http://127.0.0.1:5500","http://localhost:5500
","https://datos.hispcolombia.org"]
}
```

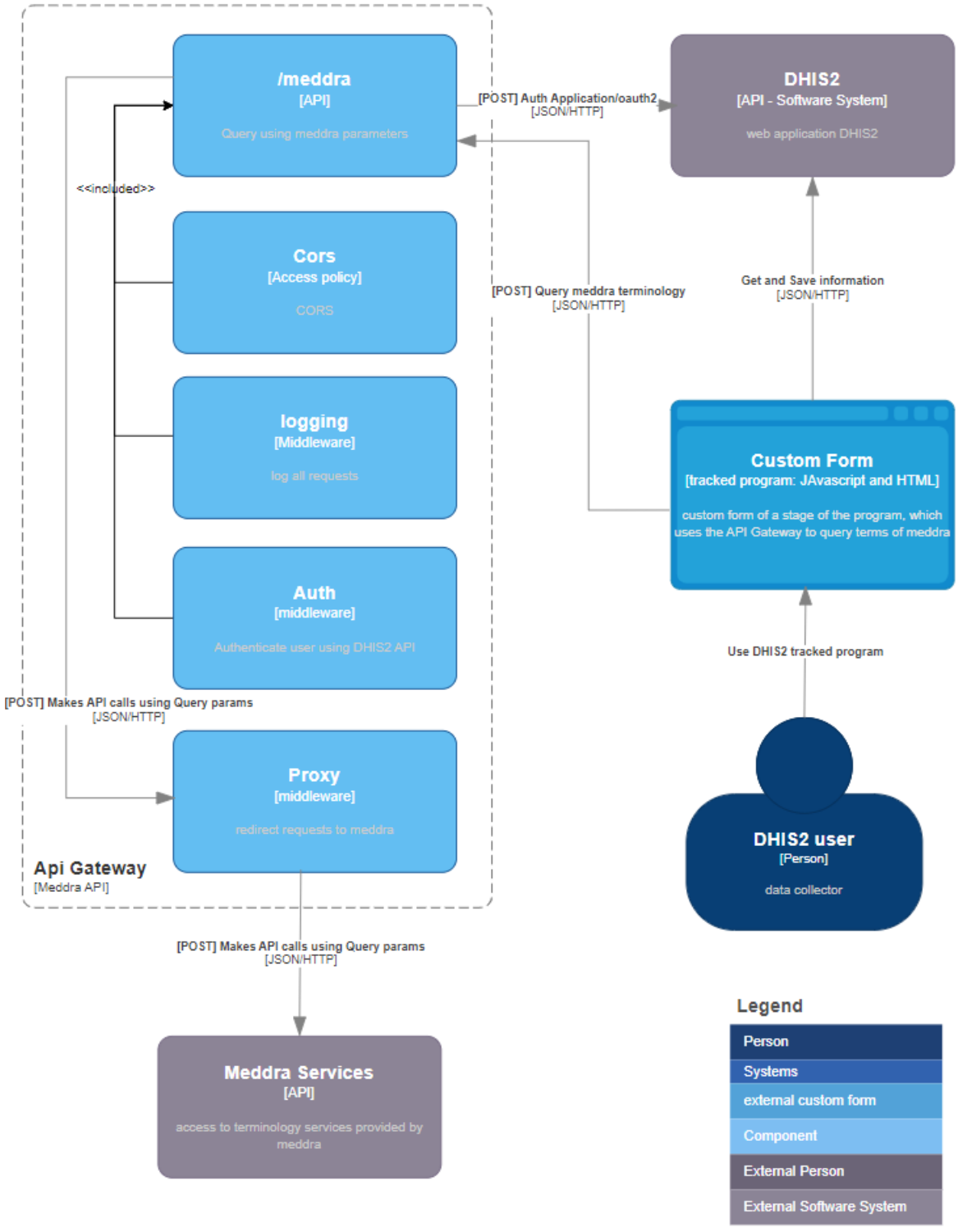
Para esto es necesario tener una licencia de acceso válida y vigente de meddra, quien provee el client_id, app_secret y el username. Esto se configura en la key meddra. Por otro lado, se debe generar un cliente oauth2 en dhis2, esto lo puedes hacer en el menú de configuración del en la opción Oauth2 clients. el usuario creado debe ser configurado en la key dhis2 con el name, client_id, client_secret, también es necesario configurar la url del token y la url path del servidor DHIS2 con el cual se integrará.

Se debe definir los servidores que tendrán comunicación con la apiGateway, estas deben ser registradas en el array de la key allowedOrigins, incluyendo el protocolo y el puerto de ser necesario.

El mismo servicio de autenticación se encarga generar el token de autenticación con meddra, y renovarlo cuando sea necesario. Es importante garantizar que se tenga una licencia vigente con meddra y que está adecuadamente configurada para que el token se genere correctamente.

3. Arquitectura

LA API Gateway tiene un endpoint en /meddra con protocolo http y método POST, la api primero valida que el cliente está dentro de las políticas cors definidas, además verifica el usuario que hace la solicitud con DHIS2, toda transacción es registrada en los logs por el componente logging y quien hace peticiones al servicio de meddra es el componente proxy. A continuación, se describe gráficamente como está construida el api Gateway



4. Peticiones http el api Gateway

Método HTTP	POST
URL	/meddra
autenticación	Bearer token

Token	DHIS2 token oauth2

Body (ejemplo, para entender como solicitar información a Meddra remitirse a la documentación de la misma)	{
	"bview": "SOC",
	"rsview": "release",
	"language": "Spanish",
	"stype": 1,
	"addlangs": [],
	"filters": [],
	"version": 21.1,
	"searchterms": [
	{
	"searchtype": 0,
	"searchterm": "insuficiencia hepática",
	"searchlogic": 0
	}
	],
	"kana": true,
	"idiacritical": true,
	"synonym": true,
	"contains": true,
	"soc": true,
	"hlgt": true,
	"hlt": true,
	"pt": true,
	"llt": true,
	"smq": true,
	"skip": 0,
	"take": 10,
	"separator": 2
	}

5. Despliegue

Para el despliegue se debe tener un servidor con nodejs[1] versión 16 o superior, y pm2[2]. A continuación, se describe cada paso para avanzar en el despliegue del apiGateway

- Copiar el contenido al servidor, esto se puede hacer de varias formas, si se tiene acceso por ssh se puede usar CSP para copiar los archivos al servidor, si se copia el código en un repositorio git (github, gitlab,azure, etc.) se puede clonar el repositorio directamente desde el servidor, para este procedimiento requeriría tener instalado git en el servidor.
- Iniciar el servicio con pm2, asegurarse en donde se encuentra ubicado el servicio, (server.js)

[1] Node.js (nodejs.org)
[2] PM2 - Quick Start (keymetrics.io)

```
medra@dev:/home/medra/apigateway$ ls
ApiGateway.zip  package.json       server.js  tests
node_modules    package-lock.json  src        testtoken
medra@dev:/home/medra/apigateway$ pm2 start server.js
[PM2][ERROR] Script already launched, add -f option to force re-execution
medra@dev:/home/medra/apigateway$ pm2 start server.js
```

- Para asegurarse que el servicio inicie automáticamente se puede ejecutar el siguiente comando, esto permitirá que los servidores iniciados y configurados en pm2 inicien con el servidor y que en fallas se inicie nuevamente.

  sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u medra --hp /opt/medra

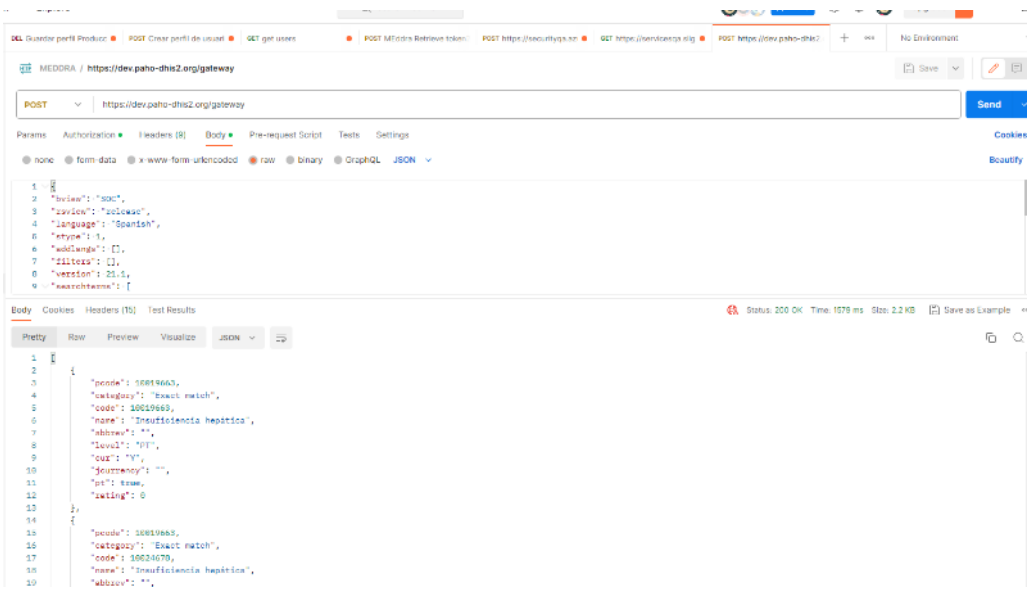
- Con el comando **pm2 list** se puede ver el servidor creado y el estado

```
medra@dev:/home/medra/apigateway$ pm2 list
```

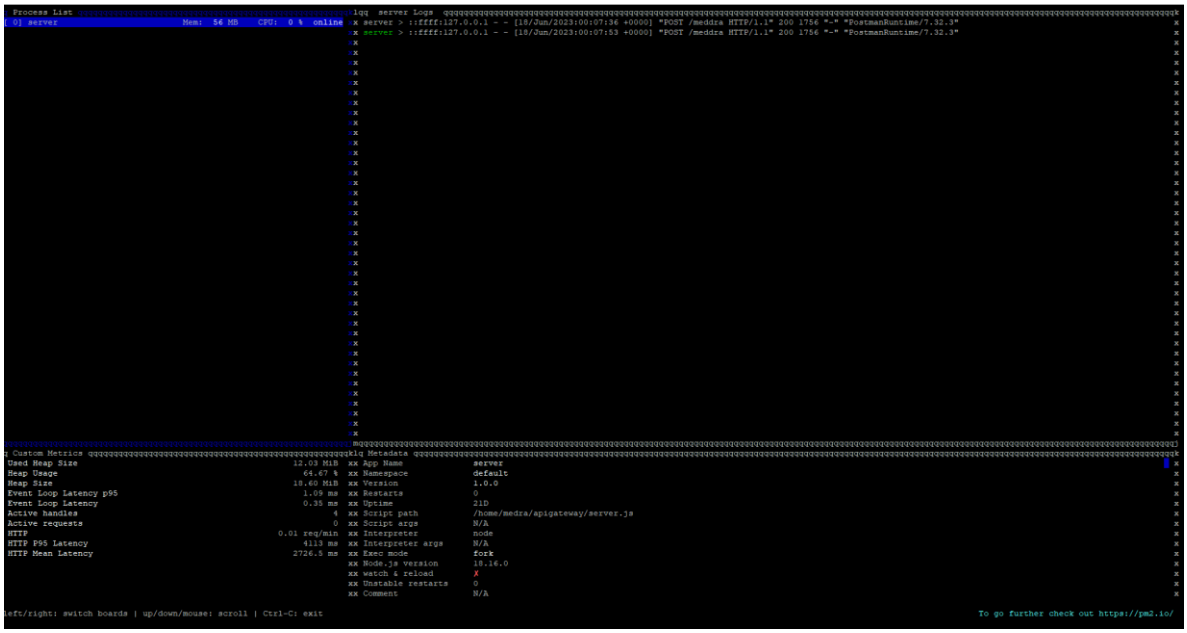
id	name	mode	↺	status	cpu	memory
0	server	fork	0	online	0%	40.9mb

```
medra@dev:/home/medra/apigateway$
```

- Otro comando importante es ver los logs del servicio, es útil para ver la ejecución y las peticiones que recibe **mp2 monit,** como ejemplo, usando postman se hace una solicitud y en el monitor se observa la solicitud recibida



Monitor MP2



6. Custom form ejemplo de uso

Como ejemplo se crea un custom form (ver código en GitHub), que muestra un listado de diagnósticos en una lista desplegable. La petición usando javascript sería el escrito a continuación.

En la cabecera que escribe el token válido definido por DHIS2 y demás datos del header como la url y el método que deberá ser validado en las políticas CORS

```
198         var myHeaders = new Headers();
199         myHeaders.append("Authorization", "Bearer SO4RkirtsFFeoxqwIPJq_EokaCo");
200         myHeaders.append("Content-Type", "application/json");
201         myHeaders.append("Access-Control-Allow-Origin","https://dev.paho-dhis2.org");
202         myHeaders.append("Access-Control-Allow-Methods","POST");
```

El body tendrá la misma estructura de la petición meddra, como lo indica la documentación, en este caso los datos que ingresan en la búsqueda como es el termino de a buscar se ubican el el campo serchterm (línea 214).

```
var raw = JSON.stringify({
    "bview": "SOC",
    "rsview": "release",
    "language": "Spanish",
    "stype": 1,
    "addlangs": [],
    "filters": [],
    "version": 21.1,
    "searchterms": [
        {
            "searchtype": 0,
            "searchterm": value,
            "searchlogic": 0
        }
    ],
    "kana": true,
    "idiacritical": true,
    "synonym": true,
    "contains": true,
    "soc": true,
    "hlgt": true,
    "hlt": true,
    "pt": true,
    "llt": true,
    "smq": true,
    "skip": 0,
    "take": 10,
    "separator": 2
});
```

Por último, se hace la llamada al endpoint creado en la API Gateway y desplegado en el servidor, requestOption tendrá el body en formato json como el ejemplo anterior y en este caso particular los resultados se adicionan a una lista que se maneja con el método addtoDataList,

```
var requestOptions = {
    method: 'POST',
    headers: myHeaders,
    body: raw,
    redirect: 'follow'
};
const resp= await fetch("https://dev.paho-dhis2.org/gateway", requestOptions);
const result=await resp.json();
return await addToDataList(result)
}
autocomplete(document.getElementsByName("autocomplete")[0]);
```

El código completo está en : [apiGateway/FormDemo/demoHISPColombia.html at main · HISPColombia/apiGateway · GitHub]

El código completo se encuentra en : [apiGateway/ApiGateway.zip at main · HISPColombia/apiGateway (github.com)]