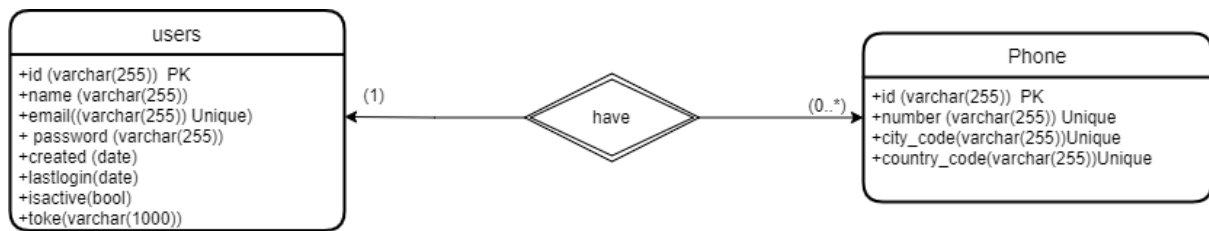


1.Base de datos

Se utilizó la base de datos que viene integrada H2.

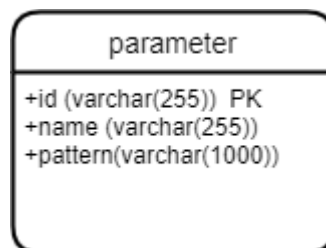
1.1 Tablas principales

La relación entre las tablas: Un usuario puede o no tener muchos teléfonos, un teléfono debe tener un usuario



1.2 Tabla secundaria

Esta tabla almacena los parámetros que serán configurables desde la aplicación.(Para esta versión solo se almacenarán parámetros para validación RegExr)



2. Funcionalidades Principales para la primera versión

2.1 Registro de un usuario y sus teléfonos

Para acceder a esta funcionalidad tendremos que ejecutar el proyecto si estamos en un ambiente local la url será esta:

es un servicio post <http://localhost:8080/user>

ejemplo de curl

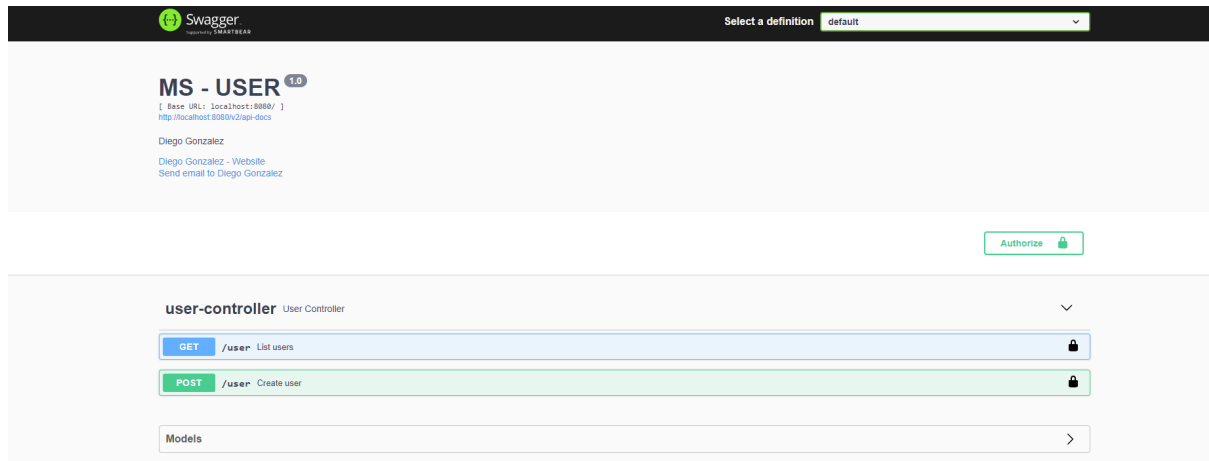
```
curl -X POST "http://localhost:8080/user" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"email\": \"diegoaliriogm@gmail.com\", \"name\": \"string\", \"password\": \"A!$12#\", \"phones\": [ { \"cityCode\": \"string\", \"contryCode\": \"string\", \"number\": \"string\" } ]}"
```

<http://localhost:8080/swagger-ui/index.html#/>

Si estamos en un ambiente desplegado en el servidor

<http://msusers-env.eba-eqkm3rfj.us-east-1.elasticbeanstalk.com/swagger-ui/index.html#/>

Se nos mostrara la pagina de swagger



debemos ir a post Create user que nos pide el siguiente request:

```
{
  "email": "diegoaliriogm@gmail.com",
  "name": "string",
  "password": "A!qweqeq$12#",
  "phones": [
    {
      "cityCode": "string",
      "contryCode": "string",
      "number": "string"
    }
  ]
}
```

existen algunas **validaciones** principales:

- La del email que tiene que ser el formato específico de un correo electrónico:

```
@Pattern(regex = REGEXEMAIL, message = "No es un formato valido para un email")
private String email;
```

- La de la contraseña que utiliza el siguiente RegExr

```
^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#&()-[{]}:; ',?/*~$
^+=<>]).{8,20}$
```

que traduce :

La contraseña debe contener al menos un carácter en minúscula, un carácter en mayúscula, un dígito, un carácter especial y una longitud entre 8 y 20.

- Para que no existan teléfonos repetidos se creó un constraint para evitar eso.

```
@Entity
@Table(name = "phone", uniqueConstraints=
@UniqueConstraint(columnNames={"number", "city_code", "country_code"}))
```

Ejemplo de este endpoint:

POST

/user

Create user

Parameters

Name

Description

createUserDto required

object

(body)

createUserDto

Edit Value | Model

```
{
  "email": "diegoalirio@gmail.com",
  "name": "string",
  "password": "A!oeaeqdz#",
  "phones": [
    {
      "cityCode": "string",
      "countryCode": "string",
      "number": "string"
    }
  ]
}
```

Al ejecutarlo nos traerá este resultado

```
{
  "id": "7787d69a-939d-493a-a561-1295db72cb38",
  "name": "string",
  "email": "digoosilrpg@gmail.com",
  "password": "92z5lI$9v/RBVYThw?ZnoTyzBtGru76NuR85SMXyCstFoBu7/Qhaio8qIry",
  "phones": [
    {
      "number": "string",
      "cityCode": "string",
      "contryCode": "string"
    }
  ],
  "created": [
    2022,
    12,
    17
  ],
  "lastlogin": [
    2022,
    12,
    17
  ],
  "token": "Bearer eyJhbGciOiJIU2wiLWVudDkiLCJkaWVudDIkbnVmbS4sZm9kdjdtQ2dtYmlslmNvbSI6InR1eWlnbmN0cmVzYSIsImF1dGhwcm9uZWUiOiJpbGlJTTEVFVWFVFIjClcjdCYXQ1OjE2ZnEyODksImVudCI6IGFiCiIWI3MTI0M2xkbXB0LDRoIjVlIN30ZPbcvNbUz8UgOOD_sIkCSsfetgiuDbYV7oyJoklr15CrwzrGaobHqdms4Kcrd9G5qeCaKr",
  "active": true
}
```

Al ejecutarlo y si no pasa la validación de columnas únicas nos traerá este resultado

Server response	
Code	Details
400 <i>undocumented</i>	Error. Response body <pre>{ "message": "No se respetó un valor que debe ser unico, por favor revisar hay dos opciones(email, telefono).-" }</pre> Download Response headers

Al ejecutarlo y si no cumple la validación del email:

```
{
  "email": "diegoaliriogail.com",
  "name": "string",
  "password": "A!qweqeq$12#",
  "phones": [
    {
      "cityCode": "string",
      "contryCode": "string",
      "number": "string"
    }
  ]
}
```

Code Details

400 Undocumented Error:

Response body

```
{
  "message": "No es un formato valido para un email"
}
```

Download

y si no cumple el valor de la contraseña:

```
{
  "email": "diegoaliriogm@gmail.com",
  "name": "string",
  "password": "A!$12#",
  "phones": [
    {
      "cityCode": "string",
      "contryCode": "string",
      "number": "string"
    }
  ]
}
```

400 Undocumented Error:

Response body

```
{
  "message": "La contraseña no cumple con el parametro establecido"
}
```

Download

Nota: para poder crear un usuario no debemos estar logueados.

2.2 Listado de los usuarios y sus datos:

Esto nos permitirá saber que usuarios están registrados en el sistema, debemos ir a ejecutar:

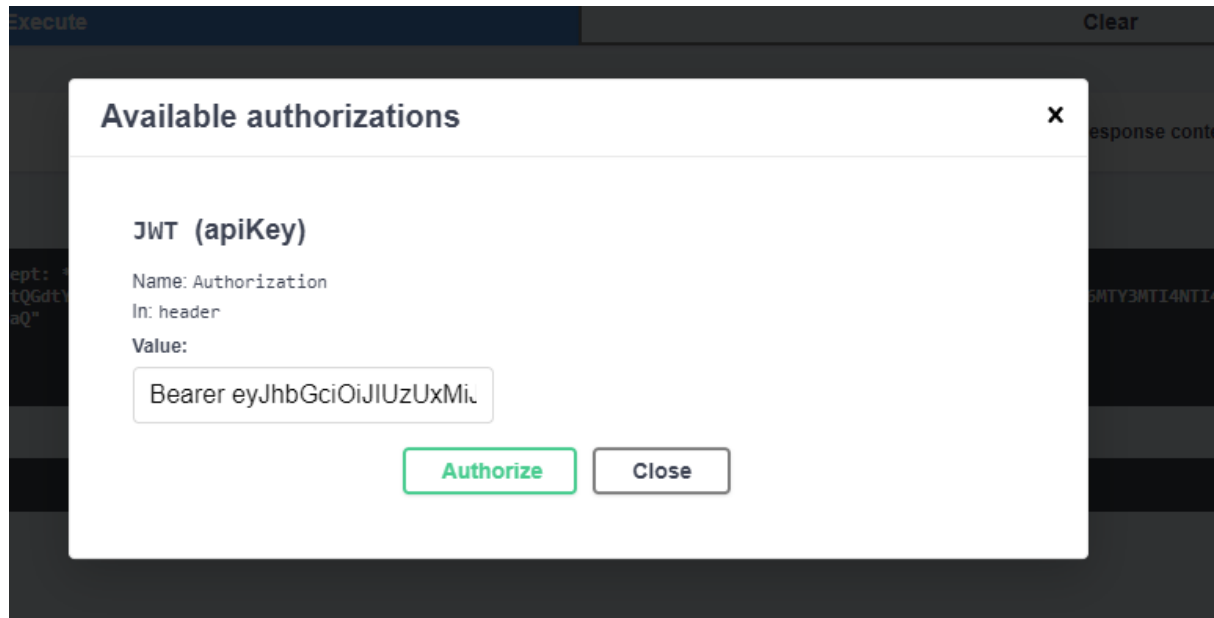
<http://localhost:8080/swagger-ui/index.html#/>

Este es un ejemplo de curl

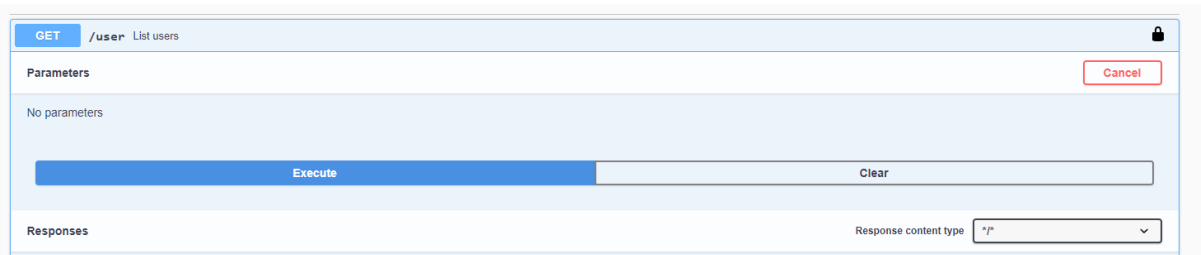
```
curl -X GET "http://localhost:8080/user" -H "accept: */*" -H "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJkaWVnb2FsaXJpb2dtQGdtYWlsLmNvbSIsInN1"
```

Yil6lnN0cmluZyIsImF1dGhvcml0aWVzIjpbIlJPTEVfVVNFUiJdLCJpYXQiOjE2NzEyODQ2ODMsImV4cCI6MTY3MTI4NTI4M30u0hfPuG0EexnZBrz04rUZZ1dhpBmZqsWaS3_3kQEp6uEOf0UmFrvsyYs0m5Je_pxApzUmdycOG6b9-AclvINOaQ"

Para ejecutar tendremos que enviar el token que nos da como resultado al registrar un usuario



le damos Authorize.



le damos Execute y nos trae el siguiente resultado.

```
[
  {
    "id": "b6186628-bb3b-43bf-a8d7-6c00bb32ef72",
    "name": "string",
    "email": "diegoaliriogm@gmail.com",
    "password":
"$2a$10$pQJwVFfwQ38hirobERNk.mBzbAT0Elc6d5imrnMPIWEnrbvFYsLu",
    "phones": [
      {
        "number": "string",
        "cityCode": "string",
        "contryCode": "string"
      }
    ],
    "created": [
```

```

2022,
12,
17
],
"lastLogin": [
2022,
12,
17
],
"token": "Bearer
eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJkaWVnb2FsaXJpb2dtQGdtYWlsLmNvbSIsInN1Yil6InN0cmduZyIsImF1dGhvcml0aWVzIjpblJPTEVfVVNFUiJdLCJpYXQiOiJE2NzEyODk5MzMslmV4cCI6MTY3MTI5MDUzM30.pr-BHezB2eq4nhMqsPms1et3OGQ9i2hSLXG551RDbzOrZTWM
BANv4SkMgGTN2qx58iHFaMLvGM2-7NICetwtg",
"active": true
}
]

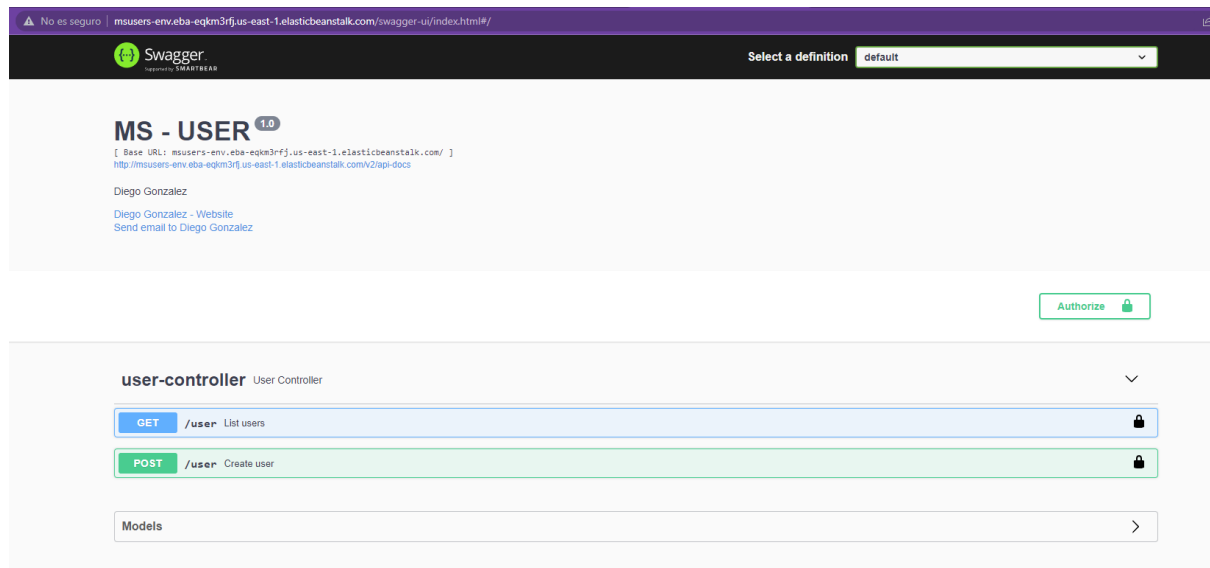
```

Nota: para poder crear un usuario no debemos estar logueados.

3 Despliegue en AWS

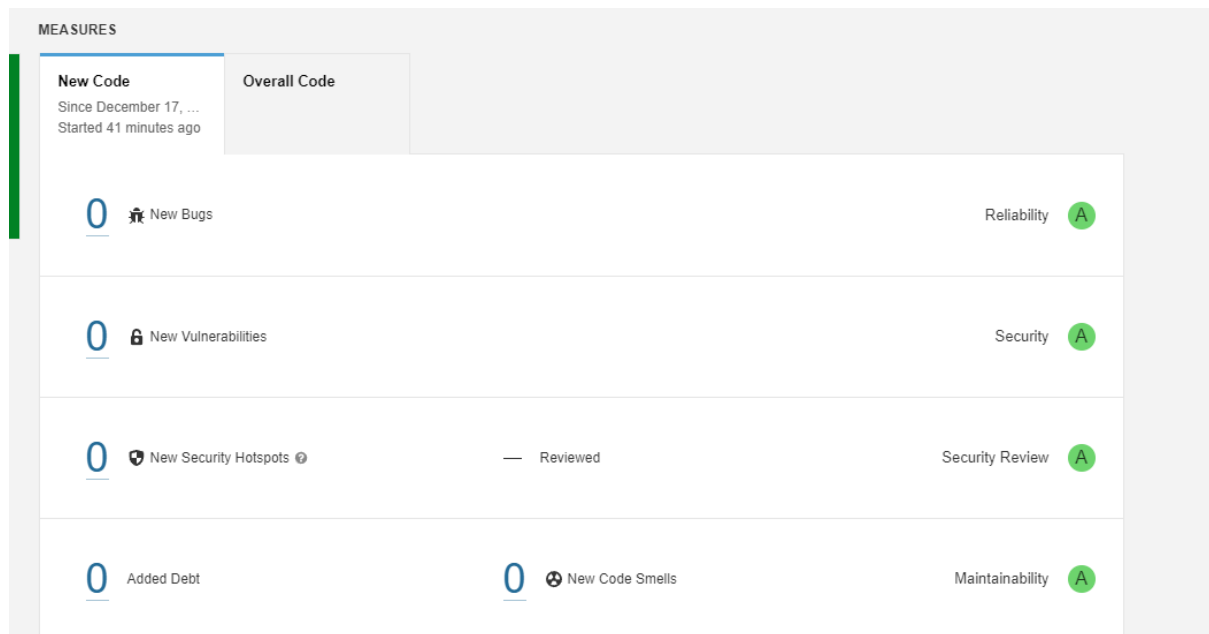
Este proyecto se desplegó en la nube de aws específicamente en el servicio de Elastic Beanstalk, la url del proyecto la podremos ver en :

<http://msusers-env.eba-eqkm3rfj.us-east-1.elasticbeanstalk.com/swagger-ui/index.html#/user-controller/getAllUsingGET>



4 Métricas SonarQube

Se realizó un pequeño análisis de código con esta herramienta arrojandolos el siguiente resultado.



5 Cobertura de código:

Se realizaron pruebas unitarias y nos arrojó la siguiente cobertura.

com	100% (19/19)	90% (67/74)	92% (161/175)
msusers	100% (19/19)	90% (67/74)	92% (161/175)
diego	100% (19/19)	90% (67/74)	92% (161/175)
configuration	100% (1/1)	100% (2/2)	100% (9/9)
controller	100% (2/2)	87% (7/8)	88% (15/17)
dto	100% (5/5)	95% (22/23)	95% (22/23)
entities	100% (3/3)	80% (16/20)	80% (17/21)
exception	100% (1/1)	100% (1/1)	100% (1/1)
mapper	100% (4/4)	100% (8/8)	88% (47/53)
repository	100% (0/0)	100% (0/0)	100% (0/0)
service	100% (1/1)	100% (9/9)	100% (45/45)
utils	100% (1/1)	100% (2/2)	100% (4/4)

6. Mejoras para siguientes versiones

- Personalizar mejor la excepción causada por email y teléfono para que den un mensaje más específico.
- Implementar todos los métodos de autorización(Login, logout,refresh token).
- Implementar más métodos para usuarios (eliminar,activar, desactivar,actualizar)
- Implementar base de datos externa al proyecto