DESARROLLO Y PRUEBAS			
Área Desarrollo:		No. FSP	
Autor del Desarrollo:	**Melquis Jair Peralta Vega**		
Proyecto:			
Proveedor:			
Fecha:	28/05/2024		

Hoja de Estado de Documento

Versión	Fecha	Responsable	Descripción
1.0	28/05/2024	Melquis Peralta	

1. Introducción

Este documento tiene como objetivo describir las especificaciones para la para la creaciond una API RESTful para el registro de usuarios.

2. Aspectos Generales de la Aplicación.

Todos los endpoints deben aceptar y retornar solamente JSON, inclusive los mensajes de error.

Todos los mensajes deben seguir el formato:

{"mensaje": "mensaje de error"}

Registro

● Ese endpoint deberá recibir un usuario con los campos "nombre", "correo", "contraseña", más un listado de objetos "teléfono", respetando el siguiente formato:

```
{
"name": "Juan Rodriguez",
"email": "juan@rodriguez.org",
"password": "hunter2",
"phones": [
```

{
"number": "1234567",
"citycode": "1",
"contrycode": "57"
}
]
}
● Responder el código de status HTTP adecuado
● En caso de éxito, retorne el usuario y los siguientes campos:

- id: id del usuario (puede ser lo que se genera por el banco de datos, pero sería
- más deseable un UUID)
- created: fecha de creación del usuario
- modified: fecha de la última actualización de usuario
- last_login: del último ingreso (en caso de nuevo usuario, va a coincidir con la fecha de creación).
- token: token de acceso de la API (puede ser UUID o JWT)
- isactive: Indica si el usuario sigue habilitado dentro del sistema.

● Si caso el correo conste en la base de datos, deberá retornar un error "El correo ya registrado".
● El correo debe seguir una expresión regular para validar que formato sea el correcto (aaaaaaa@dominio.cl).
● La clave debe seguir una expresión regular para validar que formato sea el correcto. (El valor de la expresión regular debe ser configurable).
● El token deberá ser persistido junto con el usuario.

Especificaciones

● Banco de datos en memoria. H2DATABASE
● Proceso de build Maven.
● Persistencia con JPA. Hibernate
● Framework SpringBoot Intellij IDEA 2022
● Java 17
● Entrega en un repositorio publico https://github.com/mejapeve/Poryecto-Creasion-de-Usuarios-Spring-Boot-.git
● Readme explicando cómo probarlo.
● Diagrama de la solución.

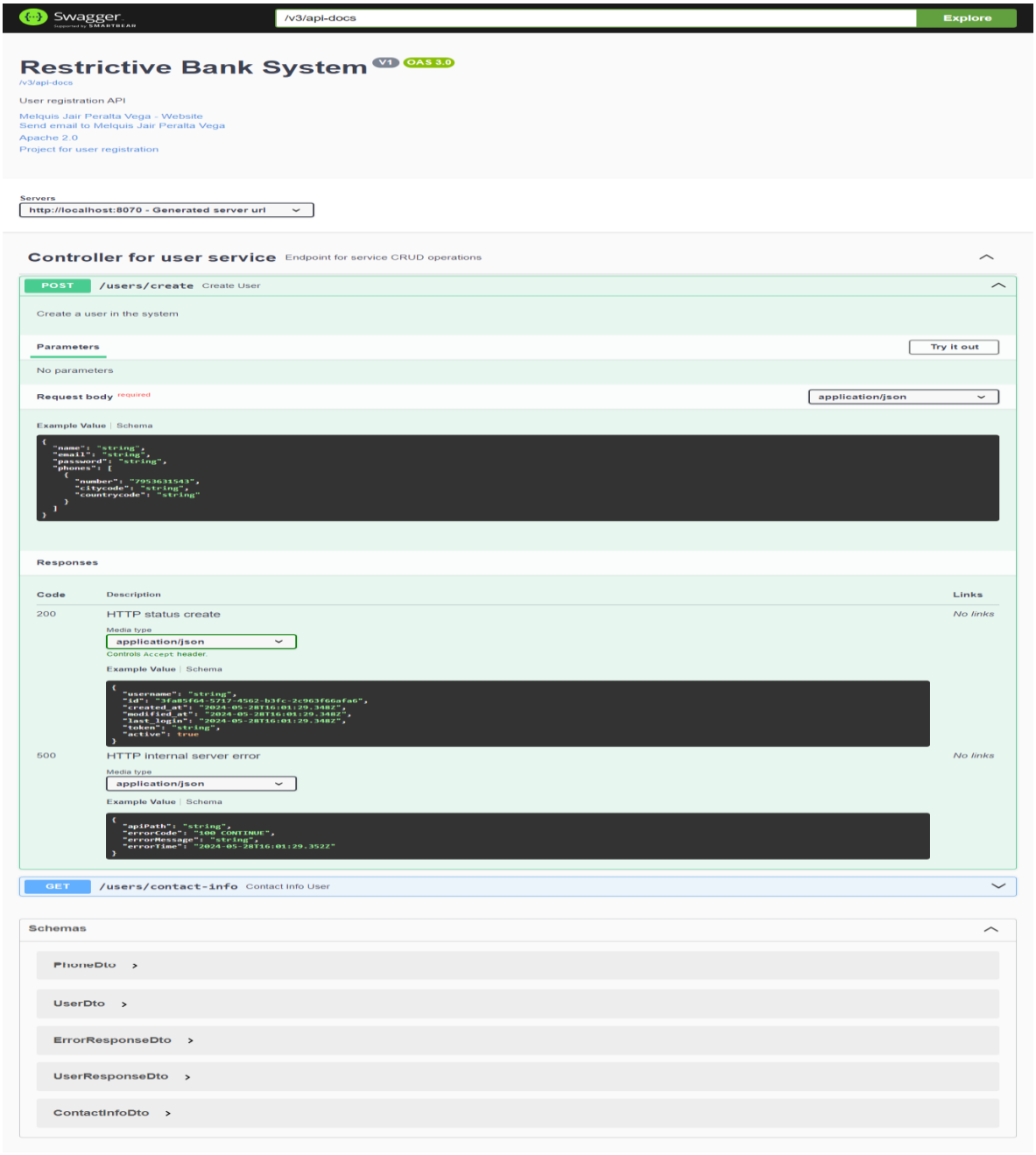
Requisitos opcionales

● JWT como token
● Pruebas unitarias

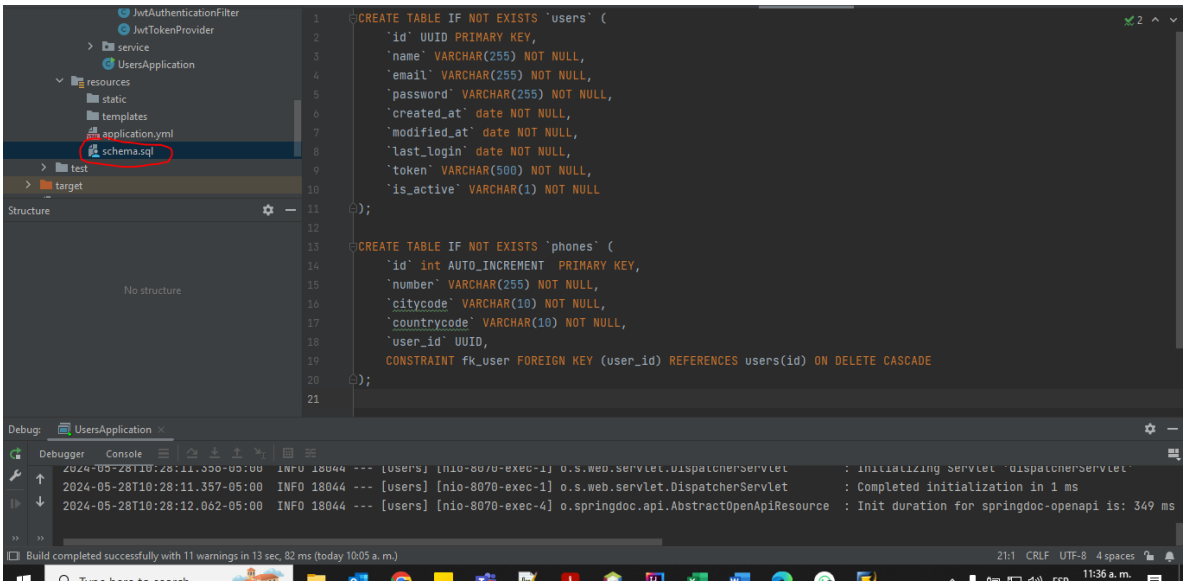
● Swagger

3. Resultados de las Pruebas:

La API es accesible a través de Swagger http://localhost:8070/swagger-ui/index.html#/



La creacion de las tablas se realiza por persistencia:

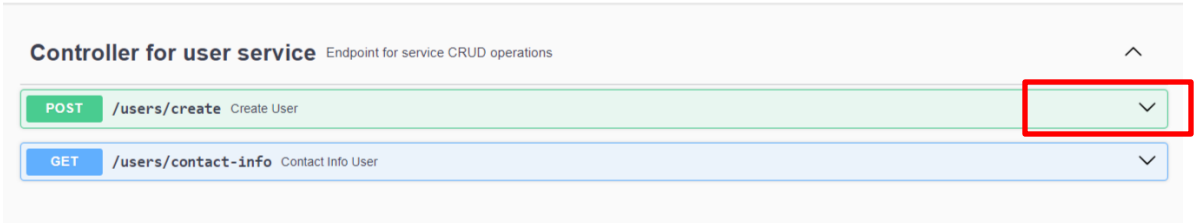


Los scrips para la creacion de dichas tablas son los siguientes:

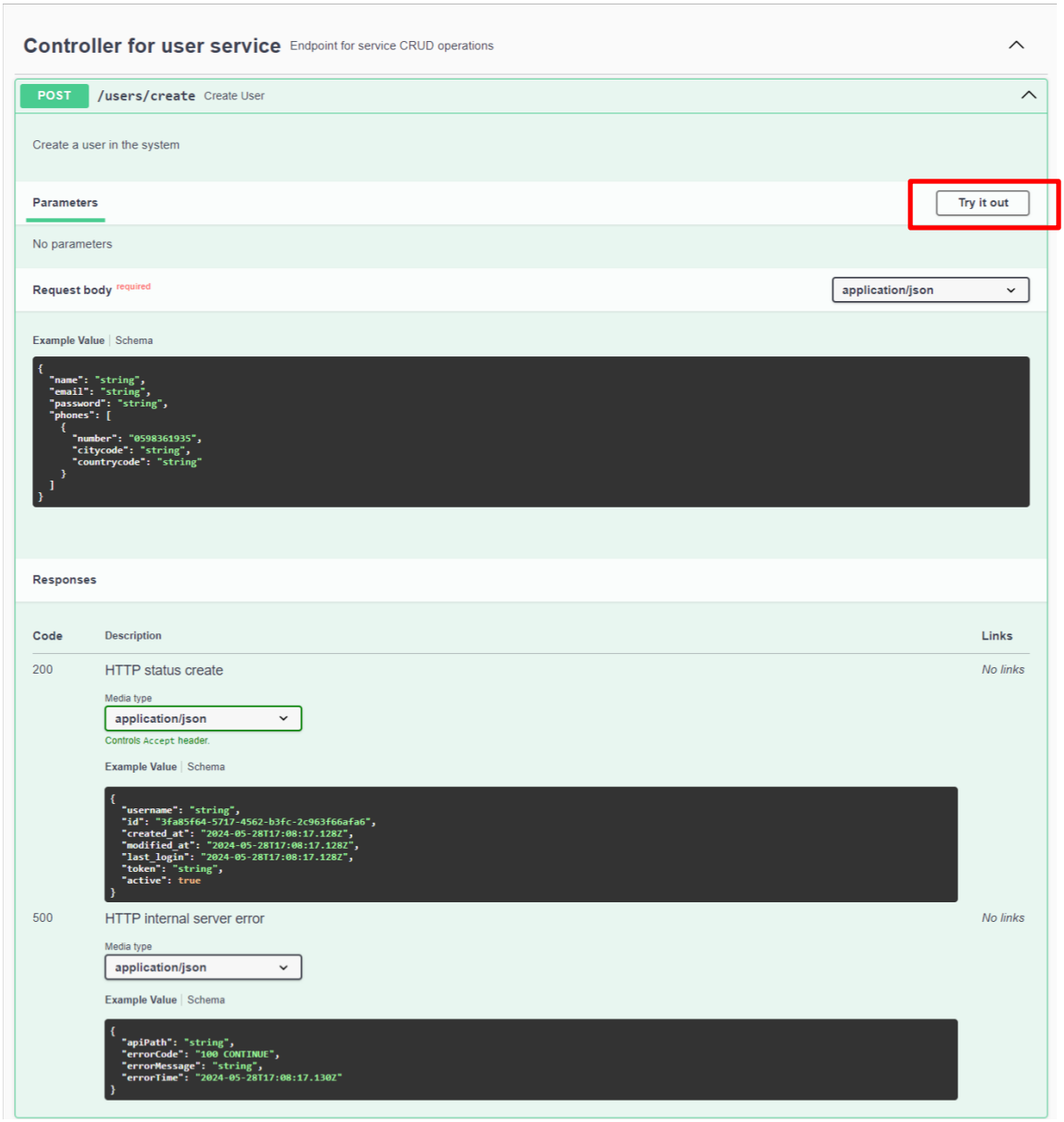
```
CREATE TABLE IF NOT EXISTS `users` (
    `id` UUID PRIMARY KEY,
    `name` VARCHAR(255) NOT NULL,
    `email` VARCHAR(255) NOT NULL,
    `password` VARCHAR(255) NOT NULL,
    `created_at` date NOT NULL,
    `modified_at` date NOT NULL,
    `last_login` date NOT NULL,
    `token` VARCHAR(500) NOT NULL,
    `is_active` VARCHAR(1) NOT NULL
);

CREATE TABLE IF NOT EXISTS `phones` (
    `id` int AUTO_INCREMENT  PRIMARY KEY,
    `number` VARCHAR(255) NOT NULL,
    `citycode` VARCHAR(10) NOT NULL,
    `countrycode` VARCHAR(10) NOT NULL,
    `user_id` UUID,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users(id) ON
DELETE CASCADE
);
```

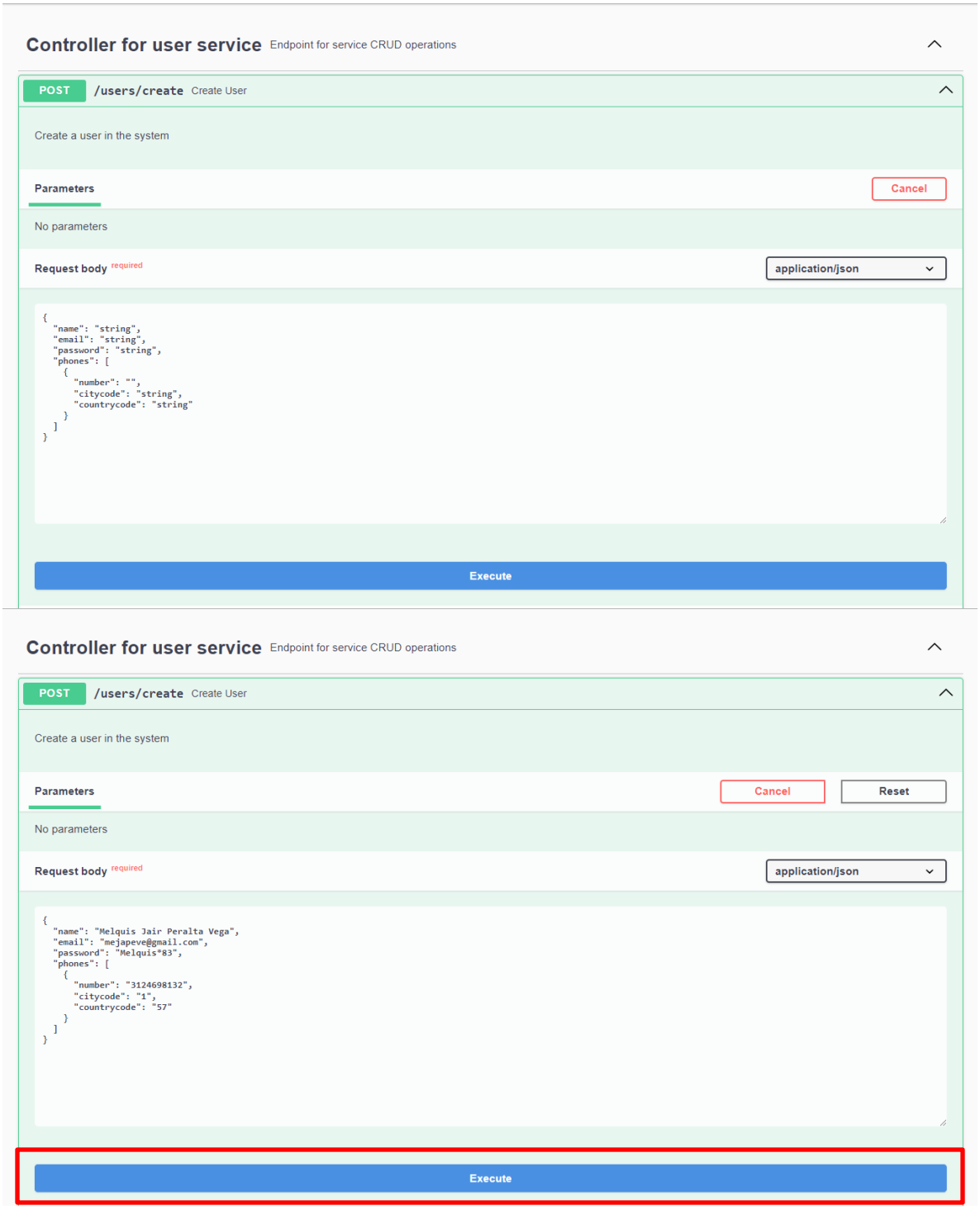
Ingresamos a Swagger y escogemos el primer servicio



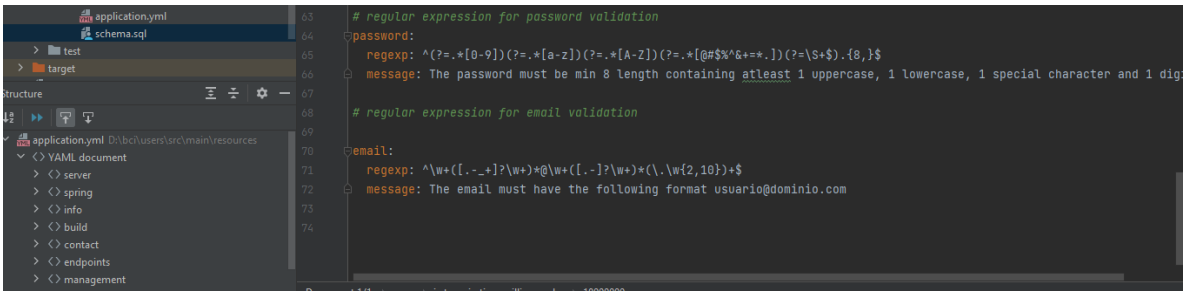
Al ingresar seleccionamos try it out en seccion de parametros



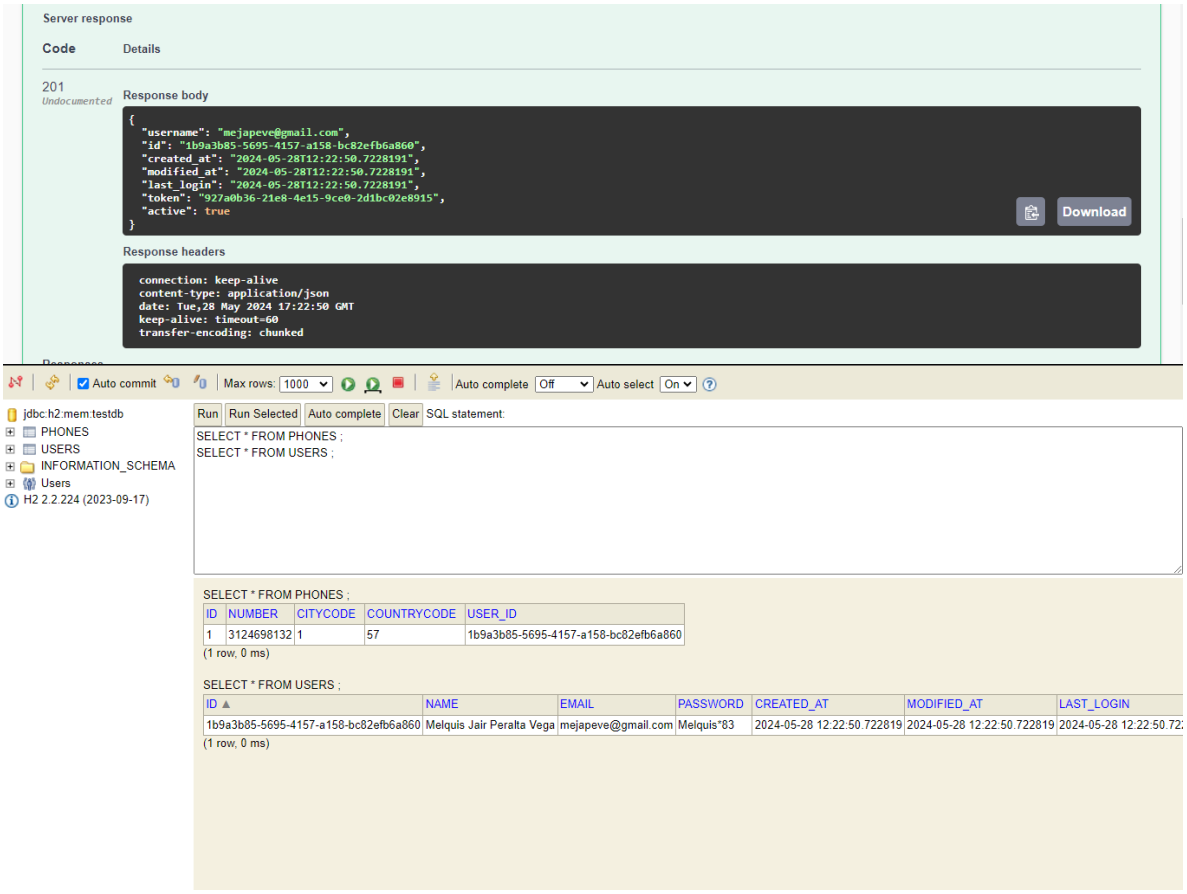
Aquí podemos validar el json de entrada del registro de usuario de acuerdo a la solicitud. Digitamos los datos de entrada y hacemos click en Execute.



De acurdo a la solictud el servicio debe validar el patron del correo y la contraseña y estos patrones deben ser configurables.

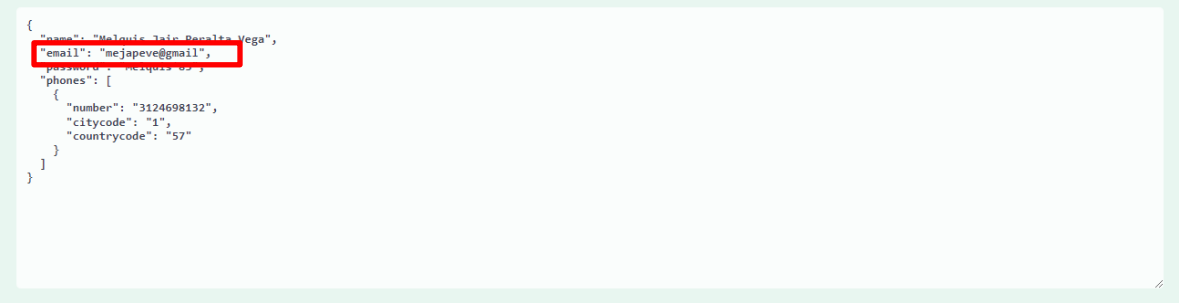


Se realiza una primera prueba con los datos correctos y el resultado es el siguinte:

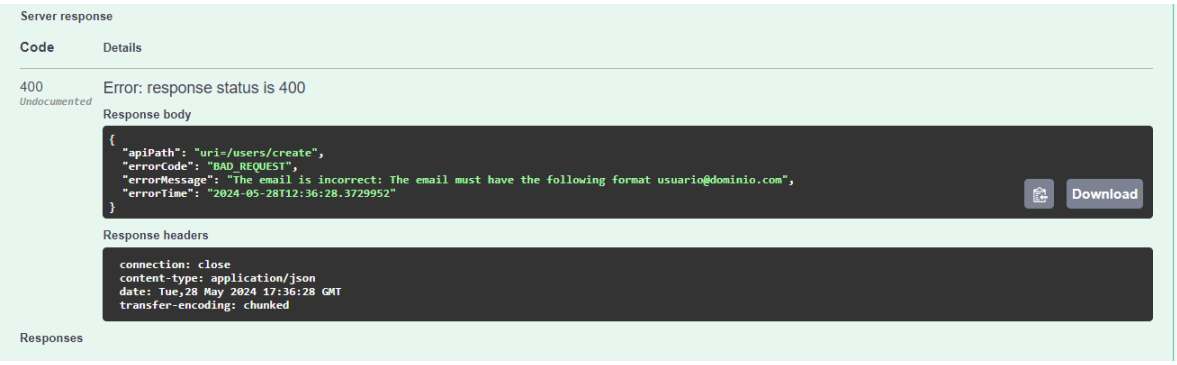


	LAST_LOGIN	TOKEN	IS_ACTIVE
19	2024-05-28 12:22:50.722819	927a0b36-21e8-4e15-9ce0-2d1bc02e8915	TRUE

La segunda pruba se reliza con el escenario de un correo sin el formato adecuado



El resultado es el siguinete:



La tercera prueba se realiza ingresando la contraseña sin el formato adeacudo (The password must be min 8 length containing atleast 1 uppercase, 1 lowercase, 1 special character and 1 digit).

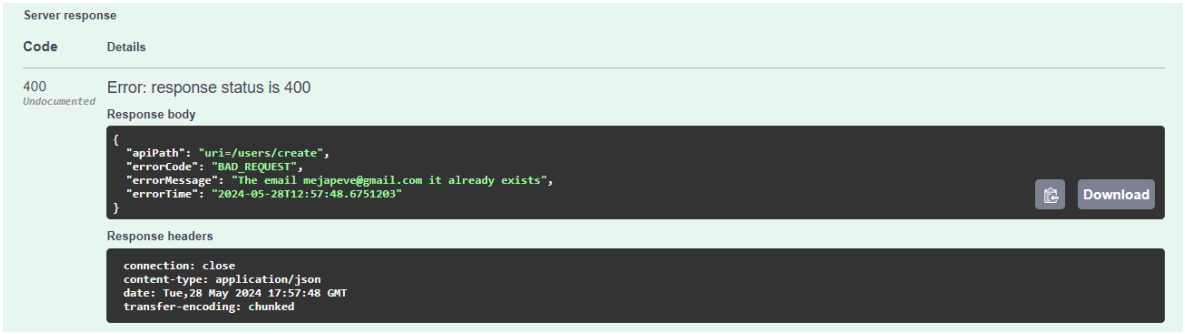
```
{
  "name": "Melquis Jair Peralta Vega",
  "email": "mejapeve@gmail.com",
  "password": "melquis*83",
  "phones": [
    {
      "number": "3124698132",
      "citycode": "1",
      "countrycode": "57"
    }
  ]
}
```


El resultado es el siguiente:

La cuarta prueba se reliza tratando de ingresar el mismo correo por segunda vez


```
{
  "name": "Melquis Jair Peralta Vega",
  "email": "mejapeve@gmail.com",
  "password": "Melquis*83",
  "phones": [
    {
      "number": "3124698132",
      "citycode": "1",
      "countrycode": "57"
    }
  ]
}
```

El resultado es el siguiente:



Para la quinta prueba se envia un request sin uno de los datos(“name”).

El resultado es el siguiente:

Para la sexta prueba ingresamos un usuario con un correo diferente.

```
{
  "name": "Melquis Jair Peralta Vega",
  "email": "mejapeve@hotmail.com",
  "password": "Melquis*83",
  "phones": [
    {
      "number": "3124698132",
      "citycode": "1",
      "countrycode": "57"
    }
  ]
}
```

El resultado es el siguiente:

SELECT * FROM PHONES;

ID	NUMBER	CITYCODE	COUNTRYCODE	USER_ID
1	3124698132	1	57	1b9a3b85-5695-4157-a158-bc82efb6a860
2	3124698132	1	57	2f569a0e-753c-430b-af67-ae05b41c8e56

(2 rows, 0 ms)

SELECT * FROM USERS;

ID	NAME	EMAIL	PASSWORD	CREATED_AT	MODIFIED_AT	LAST_LOGIN
1b9a3b85-5695-4157-a158-bc82efb6a860	Melquis Jair Peralta Vega	mejapeve@gmail.com	Melquis*83	2024-05-28 12:22:50.722819	2024-05-28 12:22:50.722819	2024-05-28 12:22:50.7
2f569a0e-753c-430b-af67-ae05b41c8e56	Melquis Jair Peralta Vega	mejapeve@hotmail.com	Melquis*83	2024-05-28 13:04:49.917459	2024-05-28 13:04:49.917459	2024-05-28 13:04:49.9

(2 rows, 1 ms)