

MEMORIA PROYECTO FIN DE MASTER

PROGRAMACIÓN EN JAVA



NOMBRE: DIEGO DOMÍNGUEZ FERNÁNDEZ

DNI: 52875389E

TELEFONO: 658333796

Índice

1) Objetivo y Alcance	pag.3
2) Modelo de Datos y Base de Datos	pag.4
3) Arquitectura	pag.6
4) API Rest definitions	pag.7
5) Funcionamiento	pag.11
6) Peticiones Post-Man	pag.17
7) Mejoras en Póximas Releases	pag.20

Objetivo y Alcance

Como proyecto fin de master se desarrollará una aplicación web que servirá para que un criador de perros pueda gestionar sus perros adultos, camadas, sus clientes y sus ventas. Esta aplicación inicialmente estará enfocada a la cría de perros, pero el diseño se planteará de tal manera que si el criador decide ampliar su actividad empresarial a otros tipos de animales no sea necesario plantear ningún cambio importante en el diseño.

Esta fuera del alcance de esta aplicación la gestión de las facturas, contabilidad y pago de impuestos derivados de la cría de animales. Solamente existirá el concepto de venta que permitirá al criador identificar sus ingresos brutos.

El desarrollo de esta aplicación está orientado a la parte back, por lo cual el diseño de la parte Front queda pendiente para futuras releases de la aplicación.

La aplicación estará securizada de una manera básica para que no sean accesibles las urls si no hay un usuario logeado correctamente. Además, está fuera del alcance de esta primera versión la creación de funcionalidades que permiten administrar a los usuarios mediante perfiles.

Modelo de Datos y Base de Datos

Como base de datos se utilizará MySQL que correrá bajo MAMP, el nombre de la base de datos será *“feederddb”* y estará disponible en la siguiente url *“//localhost:3306/feederddb”*. Esta base de datos será necesario crearla de manera manual al no estar incluida en el script de creación de tablas.

Se ha decidido utilizar el idioma inglés para dar nombre a las tablas/entidades de la aplicación.

La primera entidad del modelo de datos será *“User”* que tendrá los siguientes campos: username, password, email, name, surnames y role. Esta entidad no tendrá relación con ninguna otra.

La siguiente entidad del modelo de datos será *“Animal”* que tendrá los siguientes campos: idAnimal, name, born_date, sex, weight, status,color, father, mother, death_date, idBreed, idClient e idSale.

La siguiente entidad del modelo de datos será *“Breed”* que reflejará las razas de los animales. Los campos de esta entidad serán: idBreed, name, junior_price, senior_price y specy_id.

La siguiente entidad será *“Specy”* que reflejará las especies, es decir, una misma especie podrá tener diferentes razas. Los campos que tendrá esta entidad serán: idSpecy, name y type.

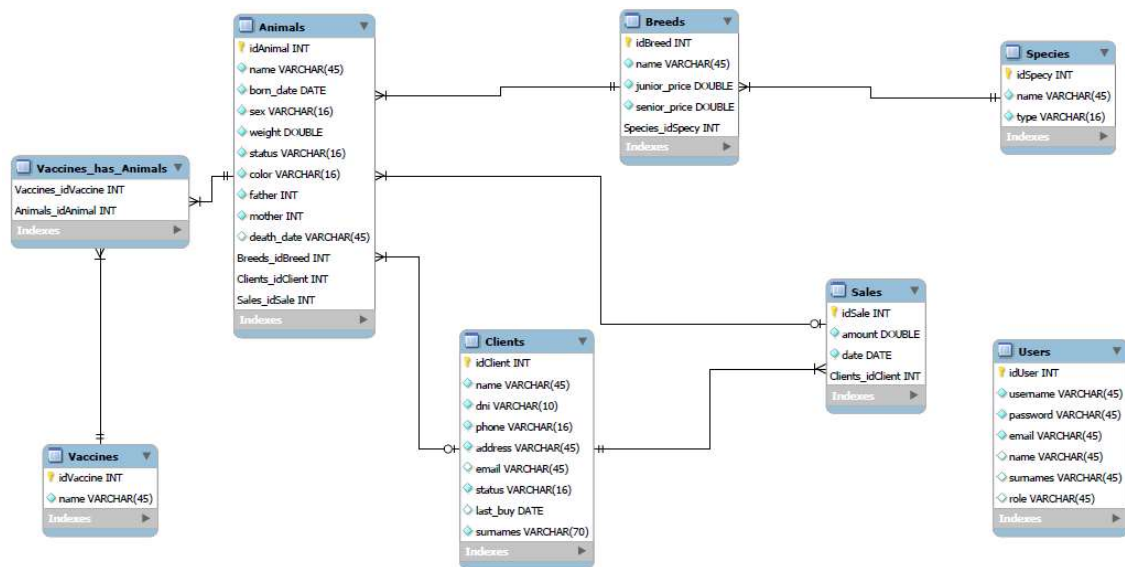
La siguiente entidad será *“Client”* y tendrá los siguientes campos: idClient, name, surnames, dni, pone, address, email, status y last_buy.

La siguiente entidad será *“Sale”* y tendrá los siguientes campos: idSale, amount, date e idClient.

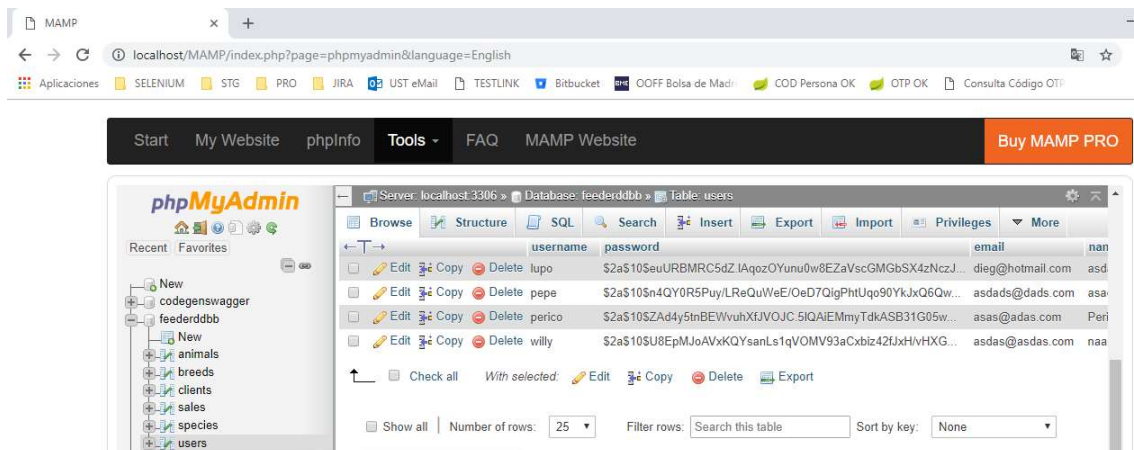
En el modelo de datos existe otra entidad con el nombre de *“Vaccine”* que pretende reflejar las vacunas que se aplican a cada animal pero que no se utilizará en esta primera versión de la aplicación.

En el fichero *“Scripts_ddbb”*, se encuentran todos los scripts de creación de las tablas junto con algunos datos de prueba. La tabla *“Users”* inicialmente estará vacía y será necesario crear a través de la GUI un usuario con el que poder acceder a la aplicación.

A continuación aparece el esquema de las relaciones entre las tablas que definen el modelo de datos de la aplicación:



Al utilizar MySQL se ha detectado un problema a la hora de la nomenclatura de los campos que son claves de otras tablas. El problema consiste en que para cada una de las tablas solo puede haber un campo que contenga el literal “id” (el problema está en la dependencia de MySQL de Spring Boot), por lo cual se utiliza “id” como identificador de la entidad y para el resto de campos que tenían “id” en el nombre, es necesario borrarlo. Por ejemplo, en la tabla “Animals” el campo “id” hace referencia a “idAnimal” y los campos “id_Client”, “id_Sale” e “id_Breed” se han tenido que renombrar a “client”, “sale” y “breed”.



Arquitectura

Se ha decidido utilizar para el desarrollo de la aplicación una arquitectura de microservicios bajo el ecosistema de Spring Boot en Java 8. Todos los microservicios compartirán una única base de datos. El ecosistema de Spring Boot contará con los siguientes microservicios:

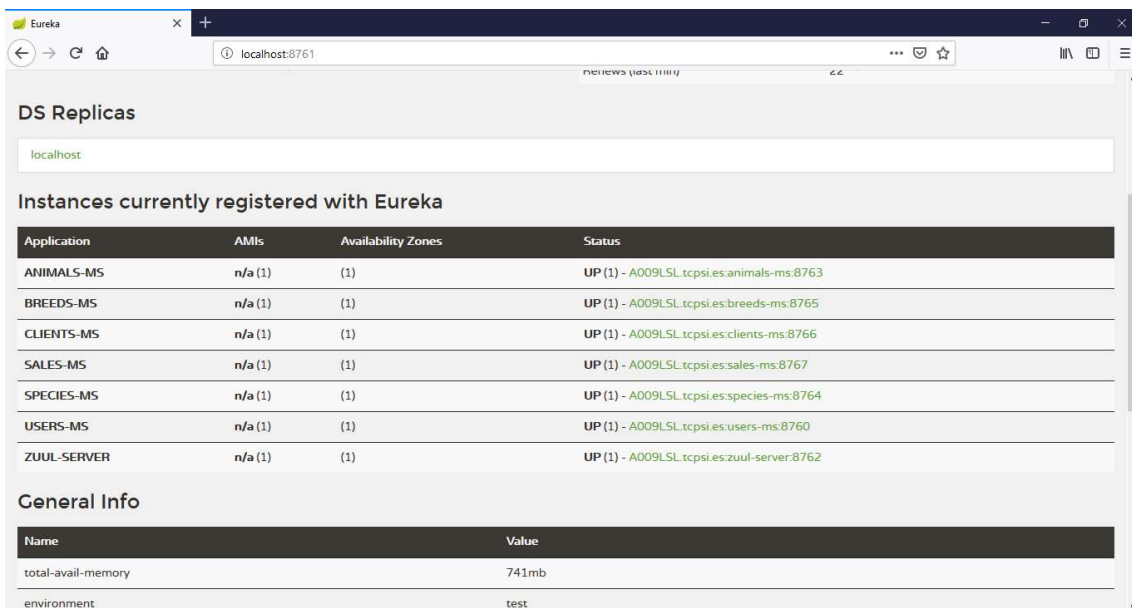
- 1) Config Server – Este microservicio correrá en el puerto 8888 en localhost y la uri donde estarán todos los ficheros properties del resto de microservicios será <https://github.com/ddfdesign2018/SpringConfig>. El nombre del proyecto para este microservicio será ms-ConfigServer y el nombre del microservicio será *"config-server"*.
- 2) Eureka – Este microservicio correrá en el puerto 8761 en localhost. El nombre del proyecto para este microservicio será ms-EurekaServer y el nombre del microservicio será *"eureka-server"*.
- 3) Zuul – Este microservicio correrá en el puerto 8762 en localhost. El nombre del proyecto para este microservicio será ms-Zuul y el nombre del microservicio será *"zuul-server"*.
- 4) Users – Este microservicio correrá en el puerto 8760 en localhost. El nombre del proyecto para este microservicio será ms-users y el nombre del microservicio será *"users-ms"*.
- 5) Animals – Este microservicio correrá en el puerto 8763 en localhost. El nombre del proyecto para este microservicio será ms-animals y el nombre del microservicio será *"animals-ms"*.
- 6) Species – Este microservicio correrá en el puerto 8764 en localhost. El nombre del proyecto para este microservicio será ms-species y el nombre del microservicio será *"species-ms"*.
- 7) Breeds – Este microservicio correrá en el puerto 8765 en localhost. El nombre del proyecto para este microservicio será ms-breeds y el nombre del microservicio será *"breeds-ms"*.
- 8) Clients – Este microservicio correrá en el puerto 8766 en localhost. El nombre del proyecto para este microservicio será ms-clients y el nombre del microservicio será *"clients-ms"*.
- 9) Sales – Este microservicio correrá en el puerto 8767 en localhost. El nombre del proyecto para este microservicio será ms-sales y el nombre del microservicio será *"sales-ms"*.

La secuencia de arranque de los microservicios debe ser la siguiente:

- 1) Arrancar Config Server (la base de datos debe estar creada previamente y disponible en MAMP).

- 2) Arrancar Eureka.
- 3) Arrancar Zuul que debe registrarse en Eureka y cargar configuración desde Config Server.
- 4) Arrancar Users que debe registrarse en Eureka y cargar configuración desde Config Server.
- 5) Arrancar Animals que debe registrarse en Eureka y cargar configuración desde Config Server.
- 6) Arrancar Breeds que debe registrarse en Eureka y cargar configuración desde Config Server.
- 7) Arrancar Species que debe registrarse en Eureka y cargar configuración desde Config Server.
- 8) Arrancar Clients que debe registrarse en Eureka y cargar configuración desde Config Server.
- 9) Arrancar Sales que debe registrarse en Eureka y cargar configuración desde Config Server.

Al consultar la consola de Eureka deben aparecer registrados todos los microservicios excepto el propio Eureka y Config Server.



The screenshot shows the Eureka web console interface. At the top, there's a search bar with 'localhost' entered. Below it, the section 'Instances currently registered with Eureka' displays a table of registered services. The table has four columns: Application, AMIs, Availability Zones, and Status. The registered services include ANIMALS-MS, BREEDS-MS, CLIENTS-MS, SALES-MS, SPECIES-MS, USERS-MS, and ZUUL-SERVER, all with a status of 'UP'. Below the table, the 'General Info' section shows two entries: 'total-avail-memory' with a value of '741mb' and 'environment' with a value of 'test'.

Application	AMIs	Availability Zones	Status
ANIMALS-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:animals-ms:8763
BREEDS-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:breeds-ms:8765
CLIENTS-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:clients-ms:8766
SALES-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:sales-ms:8767
SPECIES-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:species-ms:8764
USERS-MS	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:users-ms:8760
ZUUL-SERVER	n/a (1)	(1)	UP (1) - A009LSL.tcpsi.es:zuul-server:8762

Name	Value
total-avail-memory	741mb
environment	test

Se configura Zuul para que todos los microservicios sean accesibles desde un único punto de acceso. En la sección de funcionamiento se mostrará un ejemplo de como se puede acceder a los microservicios directamente desde su puerto o desde Zuul.

API Rest definitions

Se ha utilizado swaggerhub para la definición de todas las API rest de los microservicios que componen el ecosistema Spring Boot.

Si se accede a la url que figura a continuación se puede revisar el contrato de cada una de las APIs: <https://app.swaggerhub.com/apis/ddfdesign2018/APIBREEDER/1.0.0>

A modo de ejemplo se adjuntan la definición del API rest de Animals:

The screenshot shows the SwaggerHub web interface for the APIBREEDER/1.0.0 API. The left sidebar lists the API endpoints grouped by category:

- ANIMAL**
 - GET /animal
 - POST /animal
 - GET /animal/{idAnimal}
 - PUT /animal/{idAnimal}
 - DELETE /animal/{idAnimal}
 - GET /animal/name/{name}
 - GET /animal/breed/{idBreed}
 - GET /animal/status/{status}
 - GET /animal/father/{mother}
 - GET /animal/client/{idClient}
- BREED**
 - GET /breed
 - POST /breed
 - GET /breed/{idBreed}
 - PUT /breed/{idBreed}
 - DELETE /breed/{idBreed}
- SPECY**
 - GET /specy
 - POST /specy
 - GET /specy/{idSpecy}
 - PUT /specy/{idSpecy}
 - DELETE /specy/{idSpecy}

The main area displays the OpenAPI specification in JSON format. The visible JSON snippet includes the following details:

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "APIBREEDER",
    "version": "1.0.0"
  },
  "paths": {
    "/animal": {
      "get": {
        "tags": [
          "animal"
        ],
        "summary": "Find all animals from database",
        "parameters": [],
        "responses": {
          "200": {
            "description": "OK successful operation",
            "schema": {
              "$ref": "#/definitions/Animal"
            }
          }
        }
      },
      "post": {
        "tags": [
          "animal"
        ],
        "summary": "Add one animal into database",
        "parameters": [
          {
            "in": "body",
            "name": "Animal",
            "description": "Animal object",
            "required": true,
            "schema": {
              "$ref": "#/definitions/Animal"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Animal created correctly"
          },
          "405": {
            "description": "Invalid input"
          }
        }
      }
    },
    "/animal/{idAnimal}": {
      "get": {
        "tags": [
          "animal"
        ],
        "summary": "Get information about one animal by id",
        "parameters": [
          {
            "in": "path",
            "name": "idAnimal",
            "required": true
          }
        ]
      }
    }
  },
  "definitions": {
    "Animal": {
      "type": "object",
      "properties": {
        "name": "string",
        "status": "string",
        "breed": "string",
        "father": "string",
        "client": "string"
      }
    }
  }
}
```

The interface also shows a "Last Saved" timestamp of 12:48:51 pm - Oct 30, 2018 and a "VALID" status indicator.

animal Everything about animals		
GET	/animal	Finds all animals from database
POST	/animal	Add one animal into database
GET	/animal/{idAnimal}	Get information about one animal by id
PUT	/animal/{idAnimal}	Update an existing animal
DELETE	/animal/{idAnimal}	Delete an existing animal
GET	/animal/name/{name}	get idAnimal from name
GET	/animal/breed/{idBreed}	List of Animals filter by idBreed
GET	/animal/status/{status}	List of Animal filter by idStatus
GET	/animal/{father}/{mother}	Get children from two idAnimals

animal Everything about animals

GET /animal Finds all animals from database

Parameters

Try it out

No parameters

Response

Responsescontent type

application/json

Code

Description

200

OK successful operation

Example Value | Model

```
{
  "idAnimal": 1111111,
  "name": "Vako",
  "born_date": {},
  "sex": "male",
  "weight": 40.5,
  "status": "adult_stud",
  "color": "light brown",
  "father": 2222222,
  "mother": 3333333,
  "death_date": {},
  "idBreed": 4444444,
  "idClient": 6666666,
  "idSale": 7777777
}
```

PUT

/animal/{idAnimal}

Update an existing animal

←

Parameters

Try it out

Name	Description
idAnimal * required	
string	
(path)	

Response

Responsescontent type

application/json

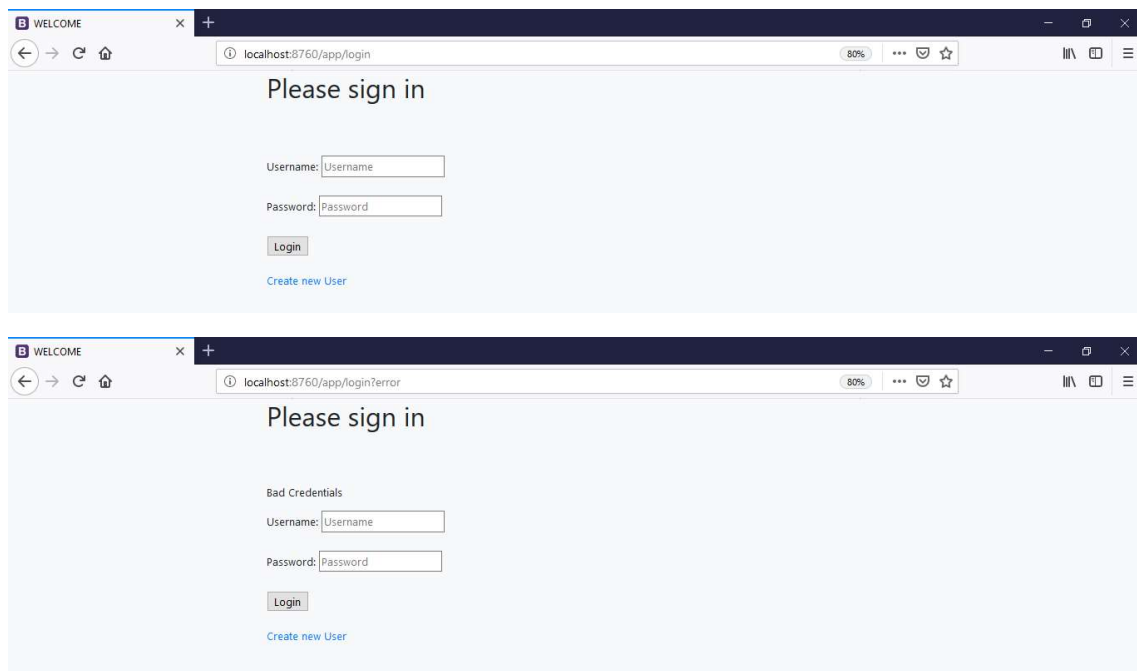
▼

Code	Description
200	Animal updated correctly into database
400	Invalid idAnimal supplied
404	Animal not found
405	Validation exception

Funcionamiento

Una vez que estén arrancados todos los microservicios y se hayan registrado en Eureka la página de arranque de la aplicación es <http://localhost:8760/app/login>. Además, la base de datos deberá estar operativa según se ha indicado en el apartado de modelo de datos.

En esta pantalla el usuario podrá hacer login o crear un nuevo usuario. Inicialmente la base de datos no tendrá ningún usuario por lo que se deberá crear uno nuevo pulsando el enlace de “Create new User”. Si el usuario trata de introducir un username y password no válidos, la aplicación devolverá el mensaje de “Bad Credentials”.



Si se rellena el formulario de creación de usuarios correctamente, se deberá poder acceder a la aplicación con el username y la contraseña.

USER CREATION

localhost:8760/app/infouser

User Creation Form

Username
user0

Password
password0

Email
you@example.com

Name
Name0

Surnames
Surname0

Submit Reset

[Welcome page](#)

Las urls que dependen del microservicio ms-users estarán securizadas por lo que solamente se podrán acceder a ellas si se ha hecho login con un usuario válido.

Una vez que se ha hecho login, la página que aparece permitirá al usuario realizar diferentes operativas básicas. La primera de ellas será la de poder hacer logout.

El siguiente botón que aparece “Animals con Zuul” permitirá sacar por pantalla una tabla con todos los animales que están disponibles en la base de datos. Esta petición se hará a través de Zuul.

El siguiente botón que aparece “Animals sin Zuul” realiza la misma operación que el botón anterior pero se realiza la petición directamente al microservicio ms-animals. Se puede observar que las urls que aparecen en la barra de navegación son diferentes si se consulta el servicio por Zuul o no.

LOGGED

localhost:8760/app/secure/logged

LOGGED SUCCESSFULLY

Logged in user: **julio**

Logout

Animals con Zuul

Animals sin Zuul

Id:

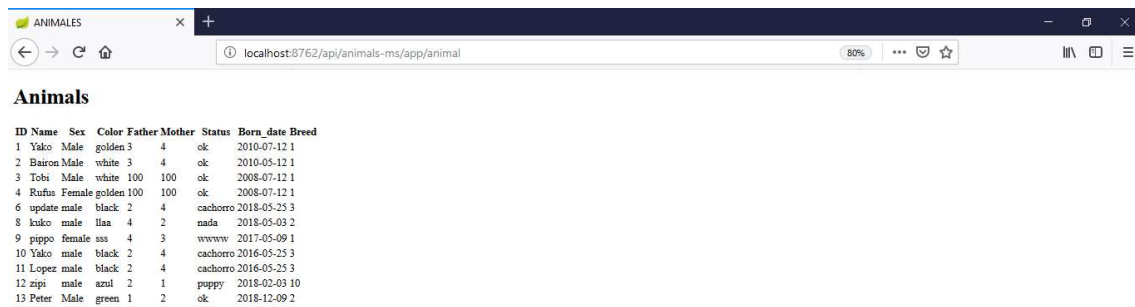
Animal by ID sin Zuul

Create animal

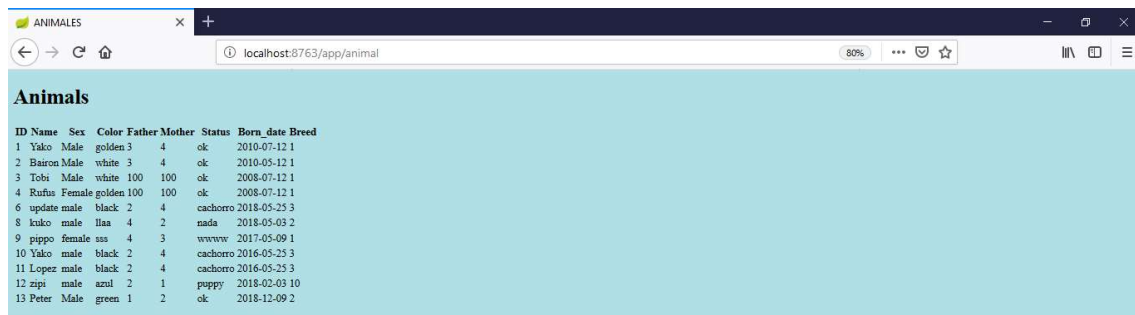
Id:

Animal by Breed sin Zuul

All desde animals sin Zuul

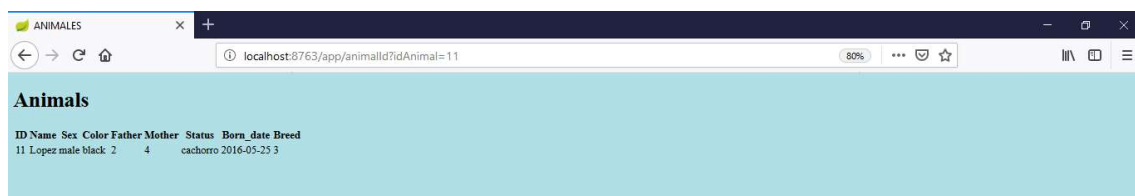


ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
1	Yako	Male	golden	3	4	ok	2010-07-12	1
2	Bairon	Male	white	3	4	ok	2010-05-12	1
3	Tobi	Male	white	100	100	ok	2008-07-12	1
4	Rufus	Female	golden	100	100	ok	2008-07-12	1
6	update	male	black	2	4	cachorro	2018-05-25	3
8	kuiko	male	lila	4	2	nada	2018-05-03	2
9	pippo	female	sss	4	3	wvvvvv	2017-05-09	1
10	Yako	male	black	2	4	cachorro	2016-05-25	3
11	Lopez	male	black	2	4	cachorro	2016-05-25	3
12	zipi	male	azul	2	1	puppy	2018-02-03	10
13	Peter	Male	green	1	2	ok	2018-12-09	2



ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
1	Yako	Male	golden	3	4	ok	2010-07-12	1
2	Bairon	Male	white	3	4	ok	2010-05-12	1
3	Tobi	Male	white	100	100	ok	2008-07-12	1
4	Rufus	Female	golden	100	100	ok	2008-07-12	1
6	update	male	black	2	4	cachorro	2018-05-25	3
8	kuiko	male	lila	4	2	nada	2018-05-03	2
9	pippo	female	sss	4	3	wvvvvv	2017-05-09	1
10	Yako	male	black	2	4	cachorro	2016-05-25	3
11	Lopez	male	black	2	4	cachorro	2016-05-25	3
12	zipi	male	azul	2	1	puppy	2018-02-03	10
13	Peter	Male	green	1	2	ok	2018-12-09	2

La siguiente funcionalidad que está disponible desde la aplicación es la capacidad de mostrar por pantalla la información de un Animal introduciendo un id. Si se introduce un id válido se mostrarán algunos campos del Animal.



ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
11	Lopez	male	black	2	4	cachorro	2016-05-25	3

El siguiente link que se muestra permite crear un Animal, aparecerá un formulario que se debe rellenar con valores válidos y al pulsar sobre el botón de crear se mostrará el listado de Animales actualizado, el último valor de la tabla debe ser el Animal que se acaba de crear.

Create Animal

Name: Zarco

Sex: male

Color: black

Mother: 1

Father: 2

Status: puppy

Born date: 2018-10-30

Breed: 2

Weight: 4.56

Animal Creation by ID sin Zuul

Animals

ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
1	Yako	Male	golden	3	4	ok	2010-07-12	1
2	Bairon	Male	white	3	4	ok	2010-05-12	1
3	Tobi	Male	white	100	100	ok	2008-07-12	1
4	Rufus	Female	golden	100	100	ok	2008-07-12	1
6	update	male	black	2	4	cachorro	2018-05-25	3
8	kuiko	male	lila	4	2	nada	2018-05-03	2
9	pippo	female	sss	4	3	www	2017-05-09	1
10	Yako	male	black	2	4	cachorro	2016-05-25	3
11	Lopez	male	black	2	4	cachorro	2016-05-25	3
12	zipi	male	azul	2	1	puppy	2018-02-03	10
13	Peter	Male	green	1	2	ok	2018-12-09	2
14	Zarco	male	black	2	1	puppy	2018-10-30	2

El siguiente formulario permite realizar una búsqueda por id de Breed y se mostrarán todos los Animales que pertenecen a una misma Breed.

Animals

ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
1	Yako	Male	golden	3	4	ok	2010-07-12	1
2	Bairon	Male	white	3	4	ok	2010-05-12	1
3	Tobi	Male	white	100	100	ok	2008-07-12	1
4	Rufus	Female	golden	100	100	ok	2008-07-12	1
9	pippo	female	sss	4	3	www	2017-05-09	1

El último botón permite mostrar por pantalla el contenido de las tablas de Animals, Breeds, Species, Clients y Sales.

Eureka

ANI-BRE-SPE

localhost:8763/app/all

Animals-Breeds-Species

Lista de Animals

ID	Name	Sex	Color	Father	Mother	Status	Born_date	Breed
1	Yako	Male	golden	3	4	ok	2010-07-12	1
2	Bairon	Male	white	3	4	ok	2010-05-12	1
3	Tobi	Male	white	100	100	ok	2008-07-12	1
4	Rufus	Female	golden	100	100	ok	2008-07-12	1
6	update	male	black	2	4	cachorro	2018-05-25	3
8	kuko	male	llaa	4	2	nada	2018-05-03	2
9	pippo	female	sss	4	3	www	2017-05-09	1
10	Yako	male	black	2	4	cachorro	2016-05-25	3
11	Lopez	male	black	2	4	cachorro	2016-05-25	3
12	zipi	male	azul	2	1	puppy	2018-02-03	10

Lista de Species

ID	Name	Type
1	Dog	mammal
2	Cat	mammal
3	Canary	bird
5	salmon	fish
6	asdadassalmon	fish

Lista de Breeds

ID	Name	Junior Price	Adult Price	Specy
----	------	--------------	-------------	-------

Eureka

ANI-BRE-SPE

localhost:8763/app/all

...

5

salmon

fish

6

asdadassalmon

fish

Lista de Breeds

ID	Name	Junior Price	Adult Price	Specy
1	Golden	300.0	250.0	1
2	Boxer	350.0	150.0	1
3	Cardenalito	50.0	450.0	3
4	Montes	50.0	55.0	2
5	pastor aleman	200.98	150.87	1

Lista de Clients

ID	Name	Surnames	DNI	Phone	Address	eMail	Status	Last Buy
4	updated	Gar Garcia	2222222T	34917651212	Calle Alcala 90	Calle Alcala 90	vip	1970-08-08
5	Ernesto	Gomez Gomez	2111111X	911111112	Calle Alcala 2	Calle Alcala 2	standard	2018-06-24
7	Vicente	Garcia Garcia	2222222T	34917651212	Calle Alcala 90	Calle Alcala 90	vip	1970-08-08
8	Balbi	Garces	2222223T	3492651212	Calle Alcala 91	Calle Alcala 91	vip	1970-08-08

Lista de Sales

ID	Amount	Date	Client ID
1	200.54	2018-10-21	1
2	300.54	2018-10-22	2
3	400.54	2018-10-22	1
4	100.56	2018-10-23	3
5	90.0	2018-10-20	4
6	666.0	2018-10-12	3

The screenshot shows a web browser window with two tabs: 'Eureka' and 'ANI-BRE-SPE'. The address bar shows 'localhost:8763/app/all'. The page content is divided into two sections: 'Lista de Clientes' and 'Lista de Sales'.

Lista de Clientes

ID	Name	Surnames	DNI	Phone	Address	eMail	Status	Last Buy
4	updated	Gar Garcia	2222222T	34917651212	Calle Alcalá 90	Calle Alcalá 90	vip	1970-08-08
5	Ernesto	Gomez Gomez	2111111X	911111112	Calle Alcalá 2	Calle Alcalá 2	standard	2018-06-24
7	Vicente	Garcia Garcia	2222222T	34917651212	Calle Alcalá 90	Calle Alcalá 90	vip	1970-08-08
8	Balbi	Garces	2222223T	3492651212	Calle Alcalá 91	Calle Alcalá 91	vip	1970-08-08

Lista de Sales

ID	Amount	Date	Client ID
1	200.54	2018-10-21	1
2	300.54	2018-10-22	2
3	400.54	2018-10-22	1
4	100.56	2018-10-23	3
5	90.0	2018-10-20	4
6	666.0	2018-10-12	3
8	23.98	2018-10-27	4
9	23.98	2018-10-27	20
10	23.98	2018-10-27	20
11	28.98	2018-10-27	6
12	23.98	2018-10-28	3
13	43.98	2018-09-28	5
14	46.98	2018-08-28	6
15	166.98	2018-08-20	5
16	176.98	2018-07-20	5
17	176.98	2018-07-20	6
18	777.98	2018-06-23	5
19	888.98	2018-06-24	5

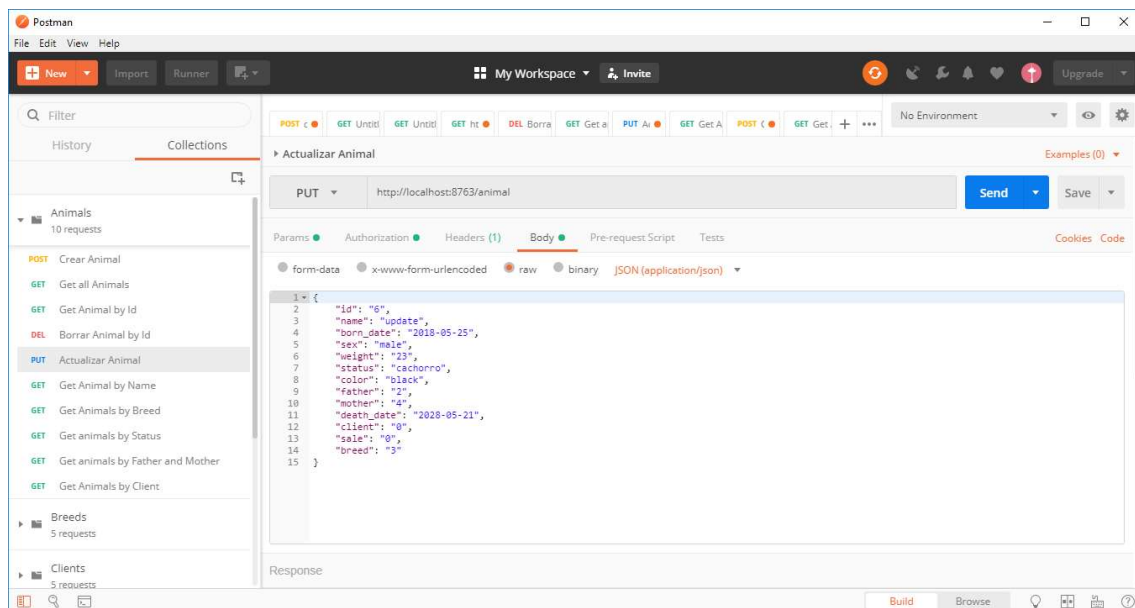
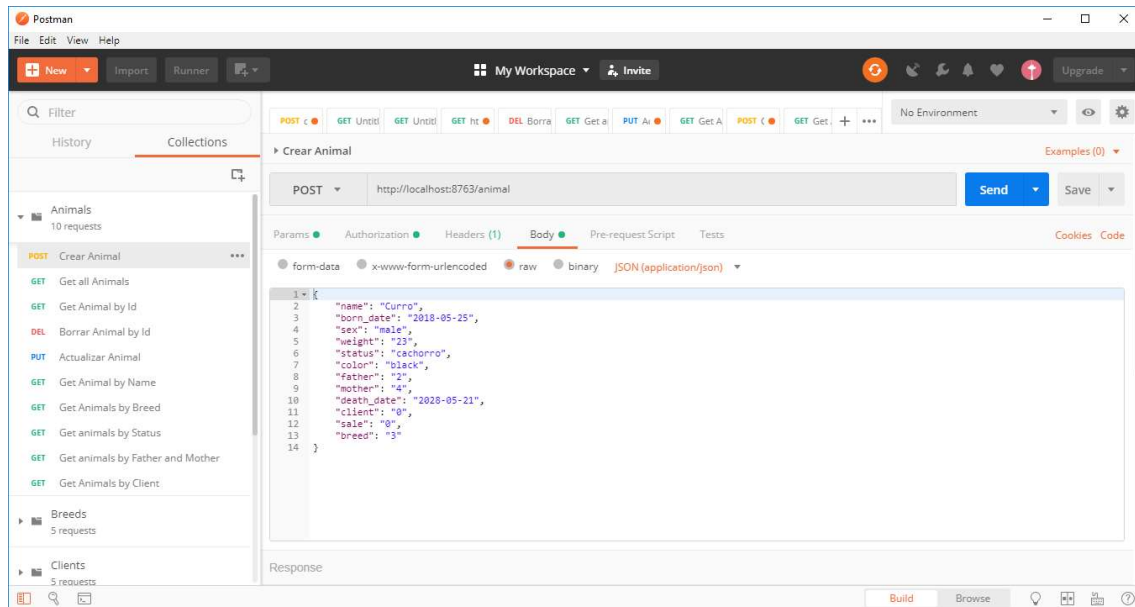
A parte de las funcionalidades que se pueden hacer por GUI, si se utiliza Postman se pueden realizar todas las peticiones que están definidas en Swagger.

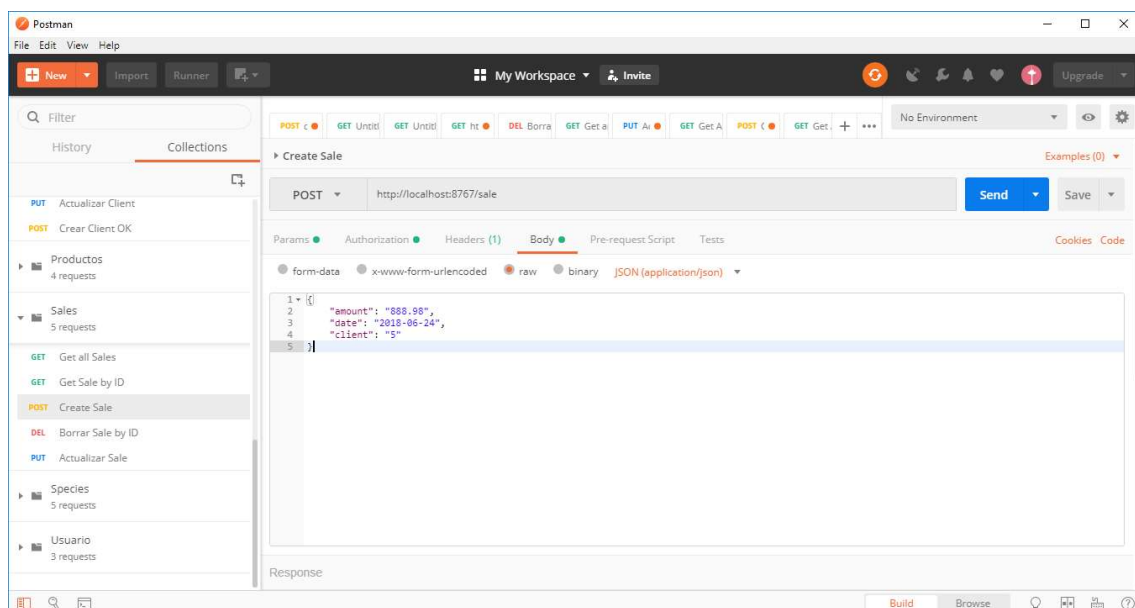
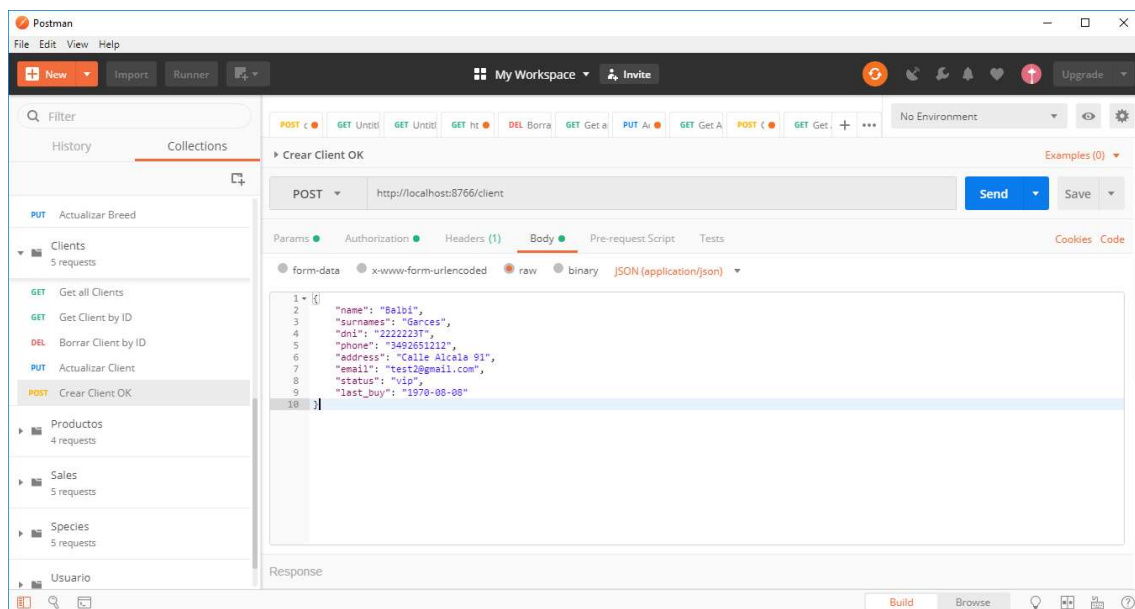
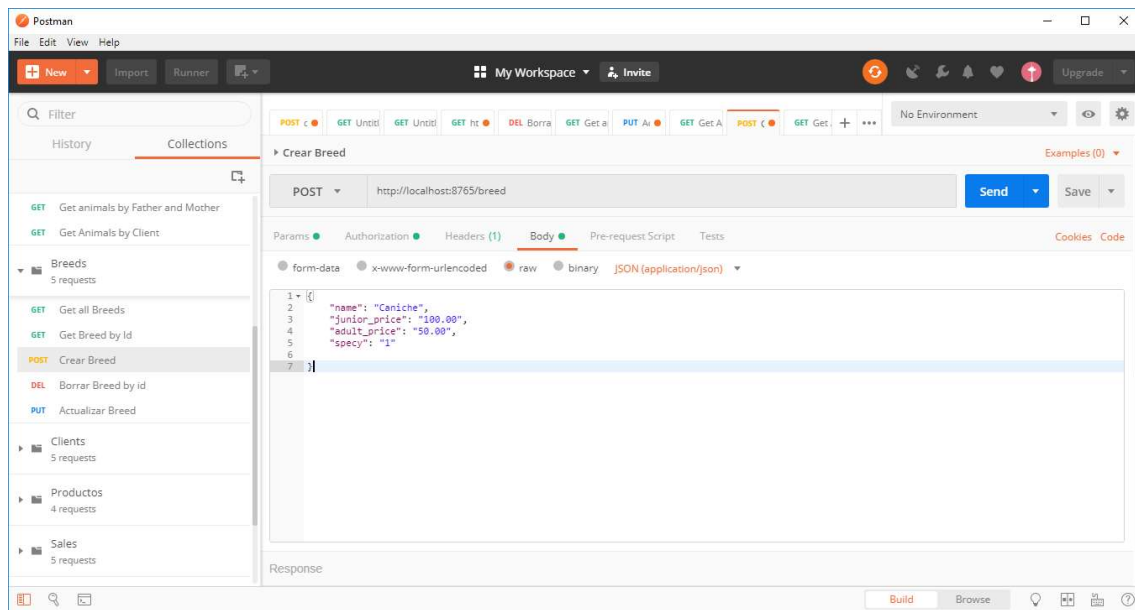
Son especialmente interesantes las peticiones de creación de entidades que implican incluir un id de otra entidad. Por ejemplo, si se desea crear un Animal, el Breed_id que se introduce debe pertenecer a un Breed que ya se encuentre en disponible en base de datos.

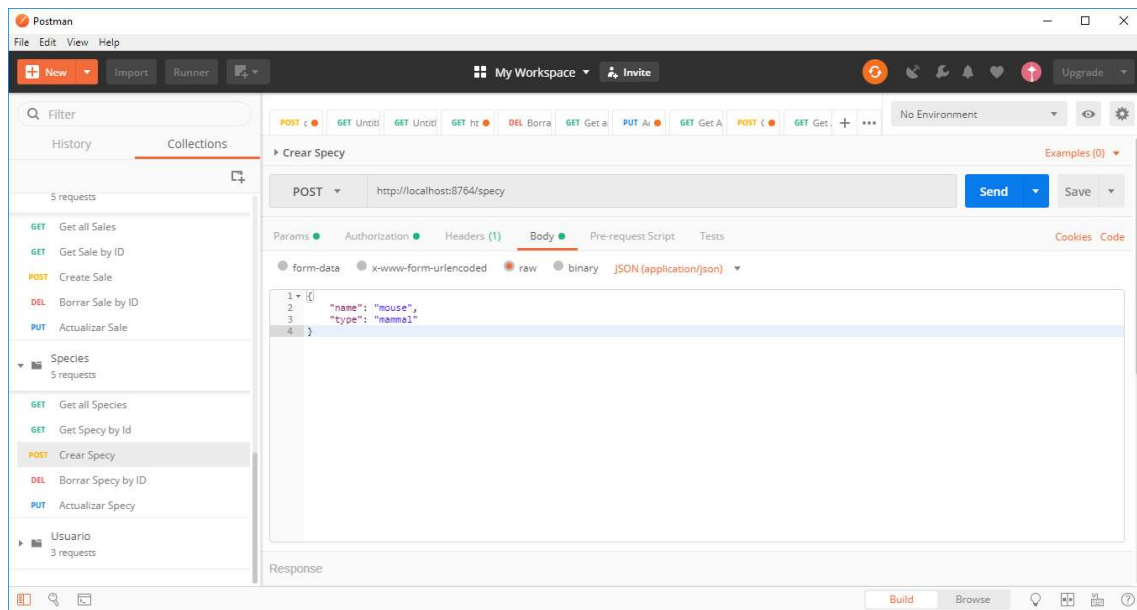
Otra petición que también resulta relevante es la que se hace cuando se crea un Sale. Al realizar un petición Post de creación Sale el microservicio ms-sales, a parte de comprobar que el Client existe en base de datos, realizará una actualización del campo last_buy de Client vinculado a la Sale a través de una petición por Feign al microservicio ms-clients. Quedaría pendiente por implementar una llamada al microservicio ms-animals para actualizar el campo Client_id de la tabla Animal vinculado al Sale.

Peticiones Post-Man

A continuación se muestran algunos pantallazos de las peticiones que se han creado para probar los API rest definidos en Swagger:







Mejoras en Próximas Releases

Para próximas versiones se puede plantear la creación de varios microservicios que permitan gestionar todos los costes derivados de la cría de animales. También se podrán crear más microservicios que permitan gestionar las vacunas y tratamientos veterinarios de los animales.

La aplicación podrá ser revisada para que pueda ser utilizada en otros idiomas a parte del inglés.

Se podrá realizar un re-styling a nivel Front de la aplicación para que sea más atractiva e intuitiva a los usuarios.

A nivel de seguridad, solamente están securizadas las url que gestiona el microservicio ms-users. Se podría extender el nivel de seguridad a todas las urls de todos los microservicios.

Se puede implementar la funcionalidad para gestionar los perfiles de los usuarios y segregar el acceso a funcionalidades dependiendo del perfil.

No se han implementado logs para cada uno de los microservicios, quedaría este punto pendiente para futuras releases. También quedaría pendiente para próximas versiones el control de errores de todos los microservicios que se han creado.

Queda pendiente para próximos releases la creación de formularios para poder crear las siguientes entidades: breed, specy, client y sale. Ahora mismo solamente se pueden crear por interfaz users y animals. Sin embargo, de los API rest de cada microservicio sí está disponible el CRUD de todas las entidades.