



UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS DE INFORMACIÓN

PACIENTE DE CLÍNICA – GRUPO 04

Desarrollo para Sistemas Distribuidos

Integrantes:

- Christian Arturo Cabrera Pajares- U202020431
- Oscar David Hospinal Roman - U202021214
- Edgard Daniel Castañeda De La Cruz- U201724395
 - Juan Pablo Cánepa Alvarez- U201816287
 - Jose Daniel Orosco Figueroa- U201501207
 - Lucas Daniel Mosquera Llosa – U202020951

PROFESOR

Flores Orihuela Carlos Alberto

Índice

1. Introducción.....	5
2. Crud con Post Man - método Post.	6
3. Resultados Método Post - Base Datos Mysql.	7
4. Crud con Post Man - método Get.	9
5. Crud con Post Man - método Delete.....	10
6. Resultados Método Delete - Base Datos Mysql.....	11
7. Crud con Post Man - método Put.....	13
8. Resultados Método Put - Base Datos Mysql.....	21
9. Manejo de Excepciones por métodos.....	23

Introducción

A continuación el presente trabajo presentará la creación de un “CRUD”, utilizando métodos en HTTP (PostMan), para “Paciente de clínica”, utilizando el lenguaje de programación Java, usando la herramienta IntelliJ Idea. Esta a su vez utilizará una base de datos (Mysql) donde se guardará la data en general.

En pocas palabras, CRUD resume las funciones requeridas por un usuario para crear y gestionar datos. Varios procesos de gestión de datos están basados en CRUD, en los que dichas operaciones están específicamente adaptadas a los requisitos del sistema y de usuario, ya sea para la gestión de bases de datos o para el uso de aplicaciones. Para los expertos, las operaciones son las herramientas de acceso típicas e indispensables para comprobar, por ejemplo, los problemas de la base de datos, mientras que para los usuarios, CRUD significa crear una cuenta (create) y utilizarla (read), actualizarla (update) o borrarla (delete) en cualquier momento. Dependiendo de la configuración regional, las operaciones CRUD pueden implementarse de diferentes maneras, como lo muestra la siguiente tabla:

CRUD-Operation	SQL	RESTful HTTP	XQuery
Create	INSERT	POST, PUT	insert
Read	SELECT	GET, HEAD	copy/modify/return
Update	UPDATE	PUT, PATCH	replace, rename
Delete	DELETE	DELETE	delete

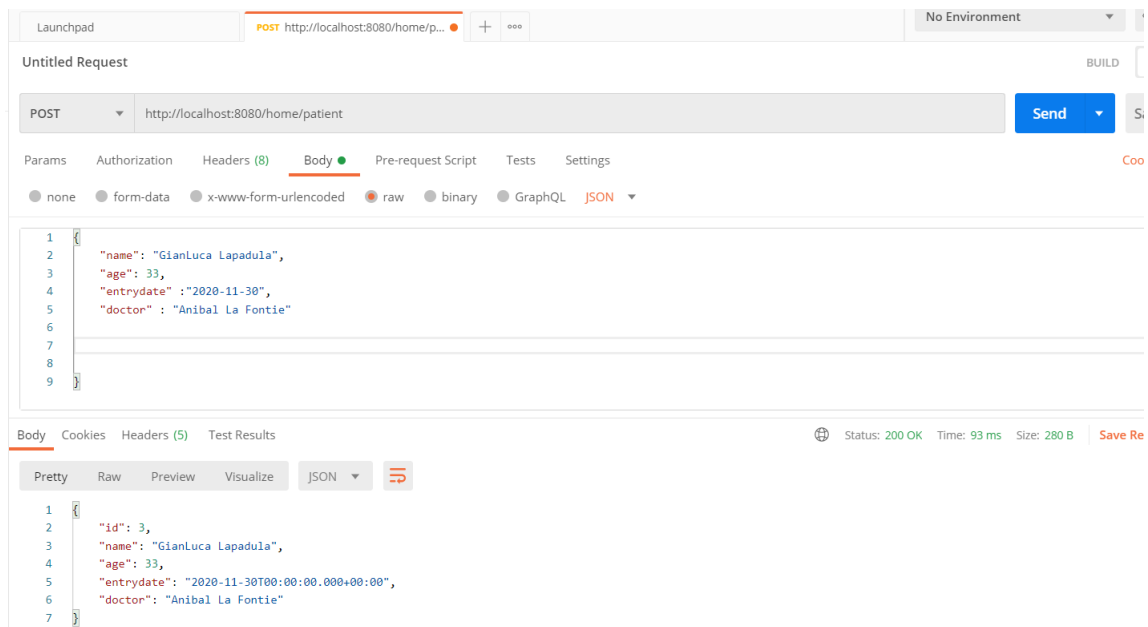
Crud con Post Man - método Post

A continuación se muestra la data ingresada al Postman, utilizando el método post, luego llamando a la conexión :” <http://localhost:8080/home/patient>” y en el Body insertando la data con los atributos creados en la clase “Patient”. La cual arroja como resultado la misma data ingresada.

Ingreso 01:

```
"name": "GianLuca Lapadula",
"age": 33,
"entrydate" : "2020-11-30",
"doctor" : "Anibal La Fontie"
```

Resultado 01:



Ingreso 02:

```
"name": "Renato Tapia",
"age": 26,
"entrydate" : "2020-12-05",
"doctor" : "Alonso la Valle"
```

Resultado 02:

```
{
  "id": 4,
  "name": "Renato Tapia",
  "age": 26,
  "entrydate": "2020-12-05T00:00:00.000+00:00",
  "doctor": "Alonso la Valle"
}
```

Launchpad POST http://localhost:8080/home/p... + ...

Untitled Request

POST http://localhost:8080/home/patient

Params Authorization Headers (8) Body Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

```
1 {
2   "name": "Renato Tapia",
3   "age": 26,
4   "entrydate": "2020-12-05",
5   "doctor": "Alonso la Valle"
6 }
7
8
9 }
```

Body Cookies Headers (5) Test Results Status: 200

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "id": 4,
3   "name": "Renato Tapia",
4   "age": 26,
5   "entrydate": "2020-12-05T00:00:00.000+00:00",
6   "doctor": "Alonso la Valle"
7 }
```

Ingreso 03:

```
{
  "name": "Paolo Guerrero",
  "age": 38,
  "entrydate": "2021-07-13",
  "doctor": "Jose la Marque Ildenfonso"
}
```

Resultado 03:

```
{
  "id": 5,
  "name": "Paolo Guerrero",
  "age": 38,
  "entrydate": "2021-07-13T00:00:00.000+00:00",
  "doctor": "Jose la Marque Ildenfonso"
}
```

POST http://localhost:8080/home/patient

Params Authorization Headers (8) Body Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```
1 {  
2   "name": "Paolo Guerrero",  
3   "age": 38,  
4   "entrydate": "2021-07-13",  
5   "doctor": "Jose la Marque Ildenfonso"  
6 }  
7
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize

JSON ▼



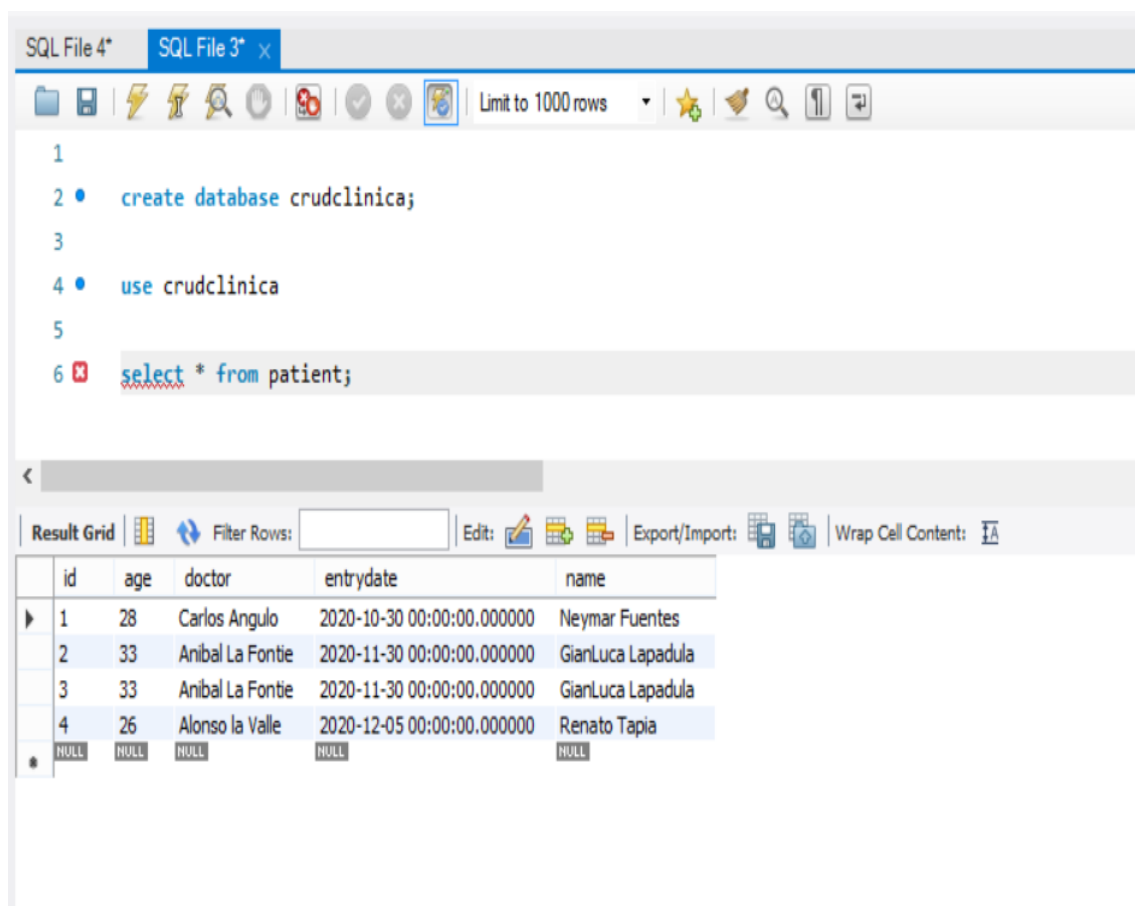
```
1 {  
2   "id": 5,  
3   "name": "Paolo Guerrero",  
4   "age": 38,  
5   "entrydate": "2021-07-13T00:00:00.000+00:00",  
6   "doctor": "Jose la Marque Ildenfonso"  
7 }
```

Resultados Método Post - Base Datos Mysql

A continuación se muestra en el Mysql , la actualización de de la base de datos “crudclinica” y el ingreso de la data en la tabla “patient”.

Para ello previamente en application.properties del IntelliJ se hizo la conexión.

```
## Spring DATASOURCE
#Cadena de conexion jdbc a la base de  datos
spring.datasource.url=jdbc:mysql://localhost:3306/crudclinica?serverTi
mezone=UTC
#usuario de la base de datos
spring.datasource.username=root
#clave de la base de datos
spring.datasource.password=mysql
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen
database
#tipo o nombre de la base de datos
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Di
alect
# Hibernate ddl auto (create, create-drop, validate, update)
#update sino existe las tablas Entidad las creará, sino las
actualizará
spring.jpa.hibernate.ddl-auto=update
#Desactiva el atributo trace de la Excepciones tipo
ResponseStatusException en los servicios REST
server.error.include-stacktrace=on trace param
```



The screenshot shows an IDE window with two tabs: "SQL File 4*" and "SQL File 3* x". The "SQL File 3* x" tab is active, displaying the following SQL queries:

```
1
2 • create database crudclinica;
3
4 • use crudclinica
5
6 ✖ select * from patient;
```

Below the queries, the "Result Grid" is visible, showing the results of the "select * from patient;" query. The grid has columns: id, age, doctor, entrydate, and name. The results are as follows:

	id	age	doctor	entrydate	name
1	1	28	Carlos Angulo	2020-10-30 00:00:00.000000	Neymar Fuentes
2	2	33	Anibal La Fontie	2020-11-30 00:00:00.000000	GianLuca Lapadula
3	3	33	Anibal La Fontie	2020-11-30 00:00:00.000000	GianLuca Lapadula
4	4	26	Alonso la Valle	2020-12-05 00:00:00.000000	Renato Tapia
*	NULL	NULL	NULL	NULL	NULL

Crud con Post Man - método Get

A continuación se muestra la consulta , utilizando el método get, luego llamando a la conexión :” <http://localhost:8080/home/patients>”, la cual al final me devuelve un arreglo de Json.

Por otro lado también podemos listar llamando por código:” <http://localhost:8080/home/patient/5>”, la cual al final me devolverá la data con el código consultado.

Resultado 01 – Listando:

```
[
  {
    "id": 1,
    "name": "Neymar Fuentes",
    "age": 28,
    "entrydate": "2020-10-30T00:00:00.000+00:00",
    "doctor": "Carlos Angulo"
  },
  {
    "id": 2,
    "name": "GianLuca Lapadula",
    "age": 33,
    "entrydate": "2020-11-30T00:00:00.000+00:00",
    "doctor": "Anibal La Fontie"
  },
  {
    "id": 3,
    "name": "GianLuca Lapadula",
    "age": 33,
    "entrydate": "2020-11-30T00:00:00.000+00:00",
    "doctor": "Anibal La Fontie"
  },
  {
    "id": 4,
    "name": "Renato Tapia",
    "age": 26,
    "entrydate": "2020-12-05T00:00:00.000+00:00",
    "doctor": "Alonso la Valle"
  },
  {
    "id": 5,
    "name": "Paolo Guerrero",
    "age": 38,
    "entrydate": "2021-07-13T00:00:00.000+00:00",
    "doctor": "Jose la Marque Ildenfonso"
  }
]
```


GET http://localhost:8080/home/patients Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1773 ms Size: 744 B

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "id": 1,
4     "name": "Neymar Fuentes",
5     "age": 28,
6     "entrydate": "2020-10-30T00:00:00.000+00:00",
7     "doctor": "Carlos Angulo"
8   },
9   {
10    "id": 2,
11    "name": "Gianluca Lapadula",
12    "age": 33,
13    "entrydate": "2020-11-30T00:00:00.000+00:00",
14    "doctor": "Anibal La Fontie"
  }
]

```

Resultado 02 – Listando por código:

```

{
  "id": 5,
  "name": "Paolo Guerrero",
  "age": 38,
  "entrydate": "2021-07-13T00:00:00.000+00:00",
  "doctor": "Jose la Marque Ildenfonso"
}

```

Untitled Request

GET http://localhost:8080/home/patient/5

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 5,
3   "name": "Paolo Guerrero",
4   "age": 38,
5   "entrydate": "2021-07-13T00:00:00.000+00:00",
6   "doctor": "Jose la Marque Ildenfonso"
7 }

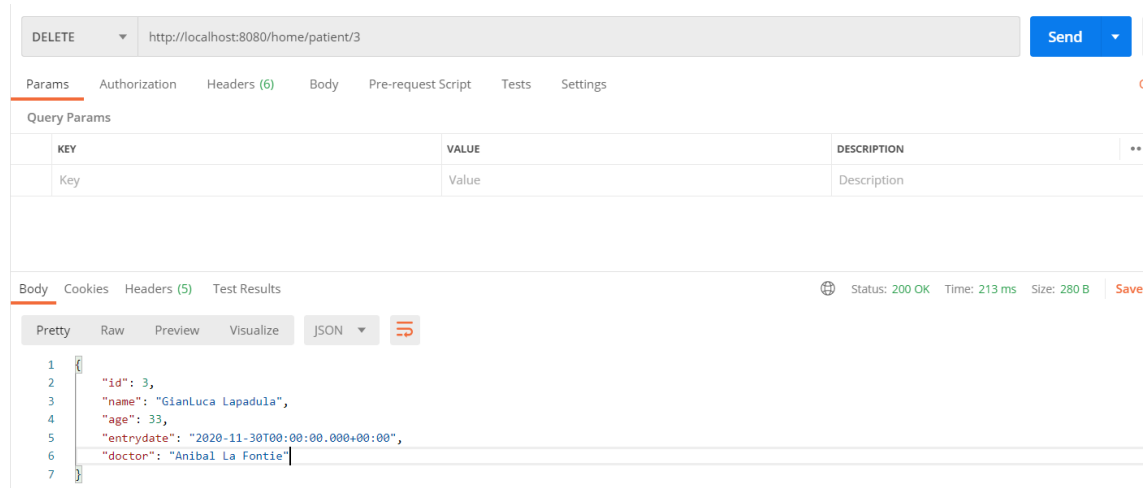
```

Crud con Post Man - método Delete

A continuación se muestra el método delete la cual se ingresa un código para eliminar la data de esa fila de la tabla en nuestra base de datos mysql, utilizando el método delete en el Post, luego llamando a la conexión:”
`http://localhost:8080/home/patient/3`”.

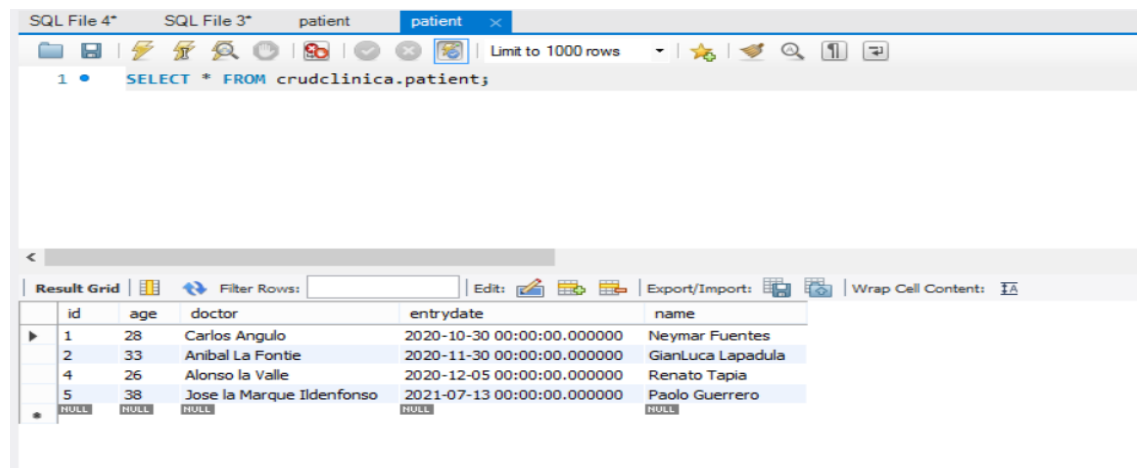
Resultado 01 – Eliminando por código:

```
{
  "id": 3,
  "name": "GianLuca Lapadula",
  "age": 33,
  "entrydate": "2020-11-30T00:00:00.000+00:00",
  "doctor": "Anibal La Fontie"
}
```



Resultados Método Delete - Base Datos Mysql

A continuación se muestra en el Mysql , la eliminación de la tabla por el código solicitado en la base de datos “crudclinica”.



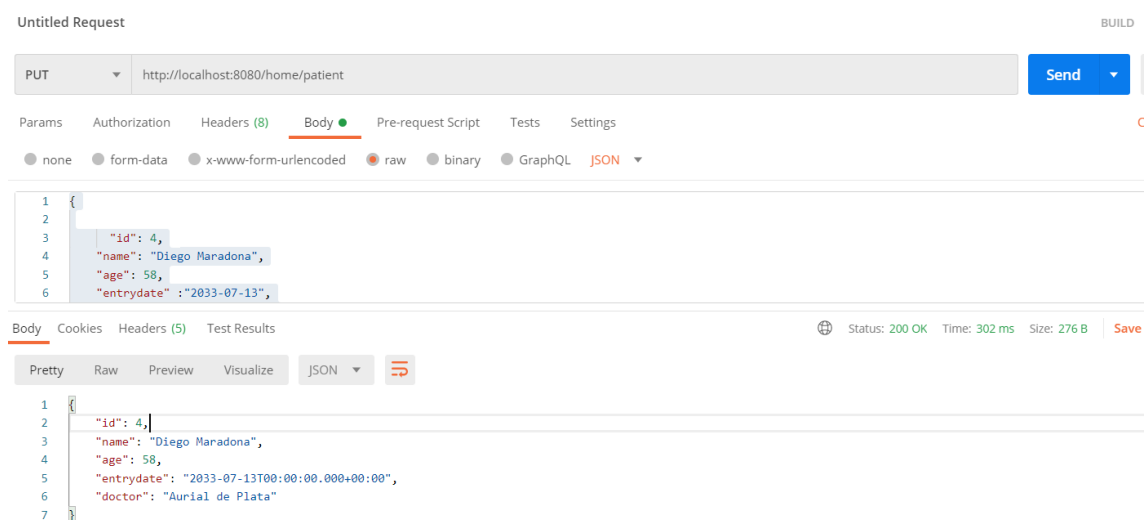
Crud con Post Man - método Put

A continuación se muestra el método put la cual se ingresa un código para actualizar la data de esa fila de la tabla en nuestra base de datos mysql, utilizando el método put en el Post, luego llamando a la conexión:"

<http://localhost:8080/home/patient>".

Resultado 01 – Actualizar por código:

```
{  
  
  "id": 4,  
  "name": "Diego Maradona",  
  "age": 58,  
  "entrydate" : "2033-07-13",  
  "doctor" : "Aurial de Plata"  
}
```



Resultados Método Put - Base Datos Mysql

A continuación se muestra en el Mysql , la actualización de la tabla por el código solicitado en la base de datos "crudclinica".

SQL File 4*SQL File 3*patientpatientpatient

Limit to 1000 rows

1

<

Manejo de Excepciones por métodos

En ello utilizaremos excepciones a los métodos usados, para poder controlar cualquier tipo de error.

Es una técnica de programación que permite al programador controlar los errores ocasionados durante la ejecución de un programa, este caso aplicados al java.

Resultado 01 –Manejo de excepciones en Java- Método put:

```
@PutMapping("/patient")
public Patient updatePatient(@RequestBody Patient patient) {
    Patient p = null;
    try {
        Patient a = servicesPatient.obtainPatientId(patient.getId());
        p = servicesPatient.updatePatient(patient);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "No se puede actualizar id inexistente");
    }
    return p;
}
```

PUT http://localhost:8080/home/patient

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

```
1 {
2
3   "id": 8,
4   "name": "Diego Maradona",
5   "age": 58,
6   "entrydate": "2033-07-13",
7 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-30T08:31:57.664+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No se puede actualizar id inexistente",
6   "path": "/home/patient"
7 }
```

Resultado 02 –Manejo de excepciones en Java- Método delete:

```
@DeleteMapping("/patient/{id}")
public Patient deletePatient(@PathVariable(value = "id") Long id) {
    Patient p = null;
    try {
        Patient a = servicesPatient.obtainPatientId(id);
        p = servicesPatient.deletePatient(id);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "No se puede eliminar id inexistente");
    }
    return p;
}
```

DELETE http://localhost:8080/home/patient/10

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-30T08:43:37.440+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No se puede eliminar id inexistente",
6   "path": "/home/patient/10"
7 }
```

Resultado 03 –Manejo de excepciones en Java- Método get:

```
@GetMapping("/patients")
public List<Patient> obtainPatient() {
    List<Patient> p = null;
    try {
        p = servicesPatient.obtainPatient();
        if(!p.isEmpty()){
            p = servicesPatient.obtainPatient();
        } else {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "No hay datos para mostrar");
        }
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Error al listar");
    }
    return p;
}
```

GET http://localhost:8080/home/patient/10

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-30T08:44:30.194+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "message": "Id inexistente",
6   "path": "/home/patient/10"
7 }
```

Resultado 04 –Manejo de excepciones en Java- Método post:

```
@PostMapping("/patient")
public Patient createPatient(@RequestBody Patient patient) {
    Patient p = null;
    try {
        p = servicesPatient.createPatient(patient);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "No se puede crear");
    }
    return p;
}
```

```
1 {
2   "name": "Paulo",
3   "age": "Hello",
4   "entrydate": "2021-07-13",
5   "doctor": "Jose la Marque Ildenfonso"
6 }
7
```

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "timestamp": "2020-10-30T08:43:37.440+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No se puede crear",
6   "path": "/home/patient"
7 }
```