

UNIVERSIDAD PRIVADA UNIFRANZ

INGENIERIA DE SISTEMAS



## **Tarea de fin de Hito 3**

### **Manejo de Spring Framework (Rest API)**

NOMBRE: JOEL CRESPO JIMENEZ

MATERIA: SISTEMAS DIGITALES

SEMESTRE: 4<sup>TO</sup> SEMESTRE

HITO: 3

DOCENTE: ING. WILLIAM BARRA

URL: [github.com/JCJ20/prograiii/tree/master/hito\\_iii/Tareas\\_Hito3/Corona\\_Virus\\_Tarea\\_H3/src/main/java/com/example/Corona\\_Virus](https://github.com/JCJ20/prograiii/tree/master/hito_iii/Tareas_Hito3/Corona_Virus_Tarea_H3/src/main/java/com/example/Corona_Virus)

CBBA-BOLIVIA

2020

## Visión general

Se tiene como visión general a la capacidad de poder implementar y generar un servicio REST desarrollado en java bajo el framework Spring accesible desde cualquier cliente web.

## Objetivos

1. Manejo de conceptos(REST).
2. Diseño y manejo de Endpoints(POST, PUT, DELETE y GET).
3. Diseño y manejo de errores y códigos HTTP(TRY, CATCH, 200, 500, 404, etc).

## Especificaciones

Debe de generar un documento (.pdf), este documento también debe estar disponible en **github**, en la carpeta **Hito3** debe crear un nuevo directorio de nombre **Tareas** y ahi debe estar el documento pdf adicionalmente todo el código debe estar disponible aquí.

La generación de este documento debe tener una **carátula**, **preguntas teóricas** y **preguntas prácticas**. Cada pregunta debe tener como respuesta una imagen(**capturas del EDITOR y POSTMAN**) del correcto funcionamiento y el **código** que resuelve el problema, adicionalmente una breve explicación de la solución de la pregunta.

## Parte Teórica.

### 1. Preguntas.

Responda de manera breve, clara y concisa posible.

- Defina y muestre ejemplo de un servicio REST.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

```
@Service
public class CoronaService implements CoronaServiceInterface {
    @Autowired
    private CoronaRepo coronaRepo;

    @Override
    public CoronaModel guardar(CoronaModel cModel) {
        return coronaRepo.save(cModel);
    }

    @PostMapping("/corona")
    public ResponseEntity save(@RequestBody CoronaModel corona) {
        try {
            return new ResponseEntity<>(coronaService.guardar(corona), HttpStatus.CREATED);
        } catch (Exception exception) {
            return new ResponseEntity<>(headers: null, HttpStatus.EXPECTATION_FAILED);
        }
    }
}
```

► POST Corona

POST ▼ http://localhost:8083/api/v1/corona

Params Authorization Headers (9) **Body ●** Pre-request Script Tests Settings

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

```

1 {
2   "idDep" : 2,
3   "nombreDepartamento" : "Oruro",
4   "casosSospechosos" : 20,
5   "casosConfirmados" : 5,
6   "casosRecuperados" : 10
7 }

```

- Que es JPA y como configurar en un entorno Spring.

JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).

Se crea un proyecto en spring inicializr:



**Proyecto** ☒ Proyecto Maven ☐ Proyecto Gradle

**Idioma** ☒ Java ☐ Kotlin ☐ Maravilloso

**Bota de primavera**

☐ 2.3.0 RC1 ☐ 2.3.0 (INSTANTÁNEA) ☐ 2.2.8 (INSTANTÁNEA) ☒ 2.2.7

☐ 2.1.15 (INSTANTÁNEA) ☐ 2.1.14

**Metadatos del proyecto**

Grupo

Artefacto

Nombre

Descripción

Nombre del paquete

embalaje ☒ Tarro ☐ Guerra

Java ☐ 14 ☐ 11 ☒ 8

Se añade dependencias al proyecto:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

Se conecta el proyecto a la base de datos:

```
spring.jpa.database=POSTGRESQL
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://ec2-52-201-55-4.compute-1.amazonaws.com:5432/da8dod2sp0mspl
spring.datasource.username=yamlturunfsyfj
spring.datasource.password=691b392a7d4a5b5bfe995a2793bb6bcb31ab9d8b9307156b2e68e2a0a63c41cf
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

server.port=8083
|
```

- Que es MAVEN - POM.

**Maven** es una herramienta de software para la gestión y construcción de proyectos **Java** creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML.

**POM** responde a las siglas de Project Object Model, es un fichero XML, que es la “unidad” principal de un proyecto Maven. Contiene información a cerca del proyecto, fuentes, test, dependencias, plugins, versión.

- Qué son los Spring estereotipos y anotaciones muestre ejemplos.

Spring Stereotypes

En estos momentos existen

únicamente 4

### @Component:

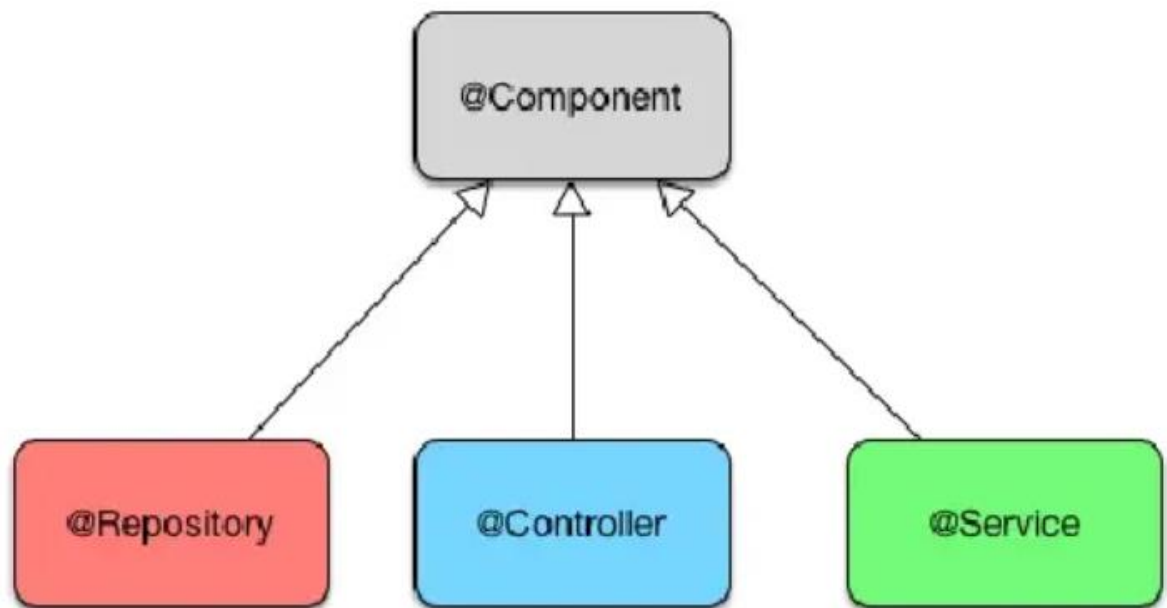
Es el estereotipo general y permite anotar un bean para que Spring lo considere uno de sus objetos.

### @Repository:

Es el estereotipo que se encarga de dar de alta un bean para que implemente el patrón repositorio que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. Al marcar el bean con esta anotación Spring aporta servicios transversales como conversión de tipos de excepciones

### @Service:

Se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea.



- Describa las características principales de REST.

### 1# PROTOCOLO CLIENTE/SERVIDOR SIN ESTADO

Cada petición HTTP contiene toda la información necesaria para ejecutarla, por tanto, esto permite que ni cliente ni servidor necesiten recordar ningún estado previo. No obstante, existen algunas excepciones y hay algunas aplicaciones HTTP que incorporan memoria caché, para que así, el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas.

## 2# CUATRO OPERACIONES MÁS IMPORTANTES

Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro; POST (crear), GET (leer y consultar), PUT (editar) y DELETE (borrar).

## 3# OBJETOS EN REST MANIPULADOS CON URI

La URI es el identificador único de cada recurso de un sistema REST. Esta, nos facilita el acceso a la información, para poder modificarla o borrarla. También para compartir su ubicación exacta a terceros.

## 4# INTERFAZ UNIFORME

Para poder realizar una transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto lo que permite es facilitar la existencia de una interfaz uniforme que sistematiza el proceso con la información.

## 5# SISTEMA DE CAPAS

Su estructura o arquitectura es jerárquica entre sus componentes, y cada una de estas capas, lleva a cabo una funcionalidad dentro del sistema REST.

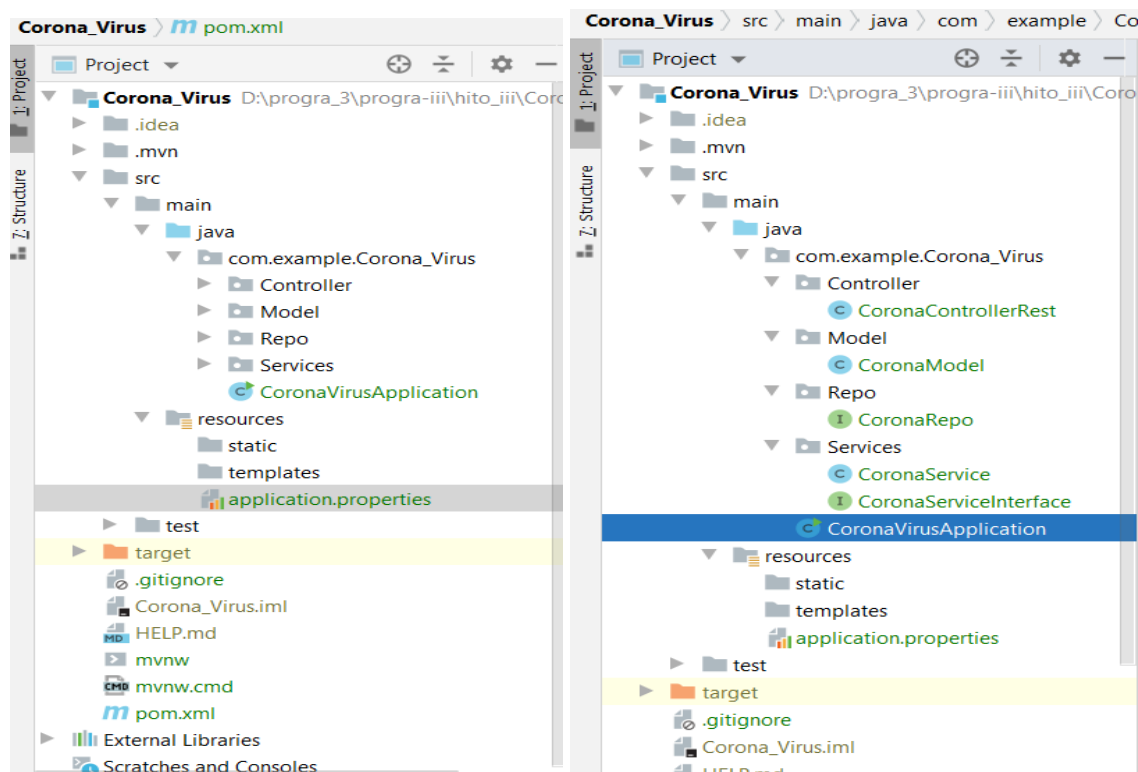
## 6# UTILIZACIÓN DE HIPERMEDIOS

El concepto hipermedio utilizado en los casos de API REST sirve para explicar la capacidad de un interfaz de desarrollo de aplicaciones para proporcionar al cliente y al usuarios los enlaces adecuados, y ejecutar acciones concretas sobre los datos. Debemos tener en cuenta que cualquier API debe disponer de hipermedios, puesto que este principio es el que define que cada vez que se hace una petición al servidor y este devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

## Parte Práctica.

Debe de trabajarlos en un nuevo proyecto mismo que debe estar en la carpeta de github del hito actual.

- a. Crear los PACKAGES necesarios(debe reflejarse el modelo MVC).



- b. Debe de crear los servicios REST para el siguiente escenario.
- i. Actualmente toda la humanidad está pasando por una pandemia conocida como Corona Virus COVID19. En Bolivia se pretende crear una plataforma en tiempo real para mostrar estos datos a cada habitante.

Es decir mostrar casos contagiados, casos sospechosos, casos recuperados, etc.

console [H5] x da8dod2sp0mspl.public.corona\_virus [PostgreSQL - da8dod2sp0mspl@ec2-52-201-55-4.compute-1.amazonaws.com] x

<

>

1 row ▾

>

>|

↺

+

-

Tx: Auto ▾

DB ▾

✓

↺

■

DDL

⚡

Comma...d (CSV) ▾

⬇

⬆

da8dod2sp0mspl.pu

Q: <Filter Criteria>

<div>id_dep ▾</div>	<div>casos_confirmados ▾</div>	<div>casos_recuperados ▾</div>	<div>casos_sospechosos ▾</div>	<div>nombre_departamento ▾</div>
1	17	26	10	70 Cochabamba



- ii. Para este propósito la primera fase de desarrollo es la creación de un servicio rest que pueda crear, modificar y retornar estos datos.

Primero creamos el modelo:

```
package com.example.Corona_Virus.Model;

import javax.persistence.*;

@Entity
@Table(name = "CoronaVirus")
public class CoronaModel {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int idDep;
    @Column(name = "NombreDepartamento", length = 50)
    private String nombreDepartamento;
    @Column(name = "CasosSospechosos")
    private int casosSospechosos;
    @Column(name = "CasosConfirmados")
    private int casosConfirmados;
    @Column(name = "CasosRecuperados")
    private int casosRecuperados;

    public Integer getIdDep() {
        return idDep;
    }

    public void setIdDep(Integer idDep) {
        this.idDep = idDep;
    }

    public String getNombreDepartamento() {
        return nombreDepartamento;
    }

    public void setNombreDepartamento(String nombreDepartamento) {
        this.nombreDepartamento = nombreDepartamento;
    }

    public int getCasosSospechosos() {
        return casosSospechosos;
    }

    public void setCasosSospechosos(int casosSospechosos) {
        this.casosSospechosos = casosSospechosos;
    }

    public int getCasosConfirmados() {
        return casosConfirmados;
    }

    public void setCasosConfirmados(int casosConfirmados) {
        this.casosConfirmados = casosConfirmados;
    }
}
```

```

    public int getCasosRecuperados() {
        return casosRecuperados;
    }

    public void setCasosRecuperados(int casosRecuperados) {
        this.casosRecuperados = casosRecuperados;
    }
}

```

Luego procedemos a crear el servicio del proyecto:

```

package com.example.Corona_Virus.Services;

import com.example.Corona_Virus.Model.CoronaModel;
import com.example.Corona_Virus.Repo.CoronaRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class CoronaService implements CoronaServiceInterface {
    @Autowired
    private CoronaRepo coronaRepo;

    @Override
    public CoronaModel guardar(CoronaModel cModel) {
        return coronaRepo.save(cModel);
    }

    @Override
    public CoronaModel actualizar(Integer IdDep, CoronaModel coronaModel) {
        Optional<CoronaModel> corona = coronaRepo.findById(IdDep);
        CoronaModel depUpdate = null;

        if (corona.isPresent()) {
            depUpdate = corona.get();
            depUpdate.setNombreDepartamento(coronaModel.getNombreDepartamento());
            depUpdate.setCasosSospechosos(coronaModel.getCasosSospechosos());
            depUpdate.setCasosConfirmados(coronaModel.getCasosConfirmados());
            depUpdate.setCasosRecuperados(coronaModel.getCasosRecuperados());

            coronaRepo.save(depUpdate);
        }
        return depUpdate;
    }

    @Override
    public Integer delete(Integer IdDep) {

```

```

        coronaRepo.deleteById(IdDep);
        return 1;
    }

    @Override
    public List<CoronaModel> getAllCoronaDeps() {
        List<CoronaModel> coronaDeps = new ArrayList<CoronaModel>();
        coronaRepo.findAll().forEach(coronaDeps::add);






        return coronaDeps;
    }

    @Override
    public CoronaModel getCoronaById(Integer idPer) {
        Optional<CoronaModel> corona = coronaRepo.findById(idPer);
        CoronaModel cdModel = null;

        if (corona.isPresent()) {
            cdModel = corona.get();
        }
        return cdModel;
    }
}

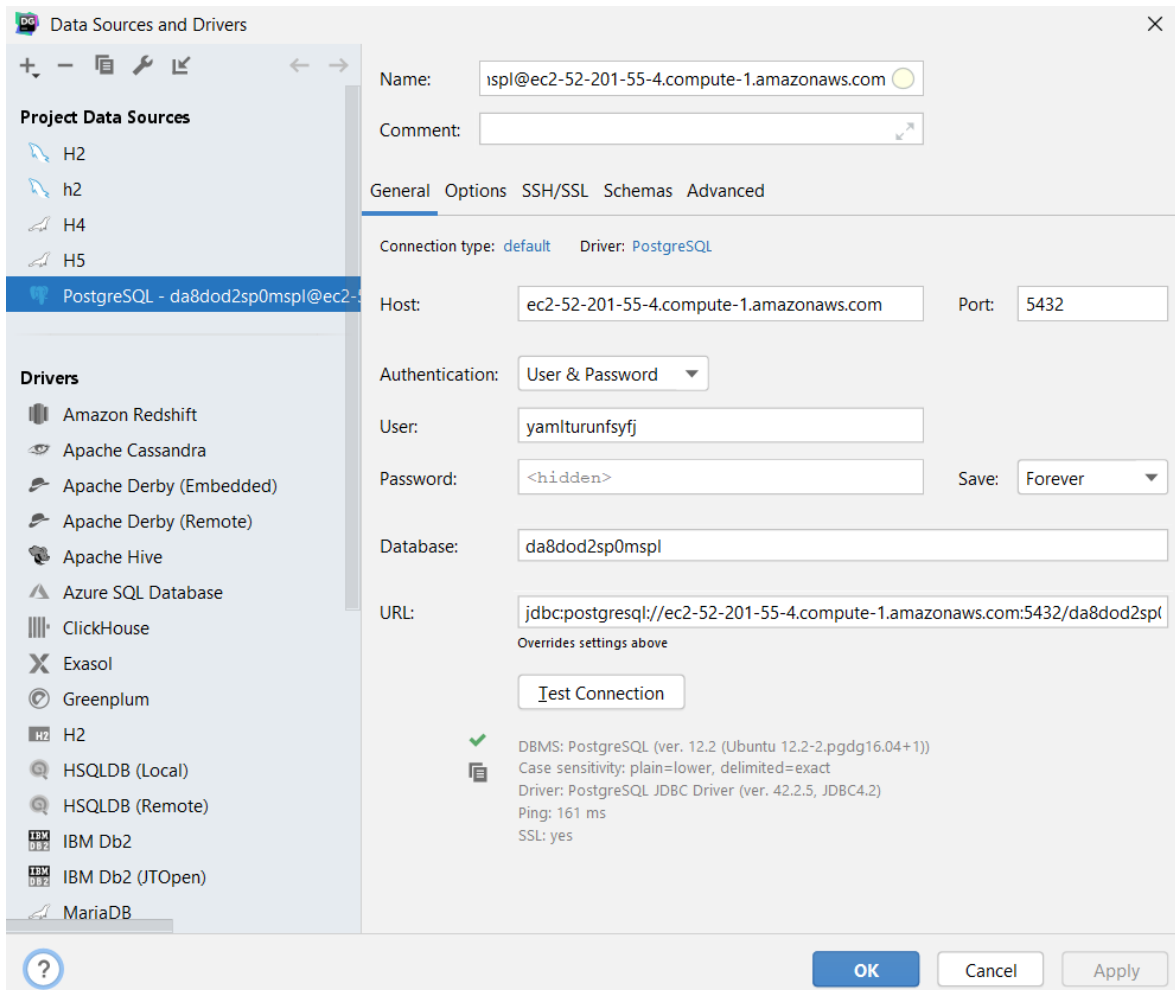
```

iii. Se tiene como base principal la siguiente tabla que nos servirá para poder generar toda esta información.

corona_virus	
 <b>id_corona_virus</b>	integer
 <b>nombre_dep</b>	varchar(50)
 <b>casos_contagiados</b>	integer
 <b>casos_sospechosos</b>	integer
 <b>casos_recuperados</b>	integer

## 2. Bases de Datos

Para poder gestionar esta tabla utilizar la plataforma HEROKU y una base de datos PostgreSQL de manera similar a la que se trabajo en clases. Inclusive es posible utilizar la misma base de datos ya creada.



### 3. Spring Framework

Es necesario crear un modelo MVC para poder solución a este problema, deberá de crear MODELS, SERVICES, REPOS y CONTROLLERS.

### 4. REST - API

Generar los siguientes servicios.

► POST Corona

POST http://localhost:8083/api/v1/corona

Params Authorization Headers (9) Body Pre-request Script Tests Settings

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

```
1 {  
2   "idDep" : 2,  
3   "nombreDepartamento" : "Oruro",  
4   "casosSospechosos" : 20,  
5   "casosConfirmados" : 5,  
6   "casosRecuperados" : 10  
7 }
```

► PUT corona

PUT http://localhost:8083/api/v1/updep/18

Params Authorization Headers (9) Body Pre-request Script Tests Settings

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

```
1 {  
2   "nombreDepartamento" : "new dep",  
3   "casosSospechosos" : 20,  
4   "casosConfirmados" : 5,  
5   "casosRecuperados" : 10  
6 }
```

▶ GET dep

GET

http://localhost:8083/api/v1/corona/18

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   "idDep": 18,  
3   "nombreDepartamento": "new dep",  
4   "casosSospechosos": 20,  
5   "casosConfirmados": 5,  
6   "casosRecuperados": 10  
7 }
```

▶ GET deps

GET

http://localhost:8083/api/v1/deps

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    {
3      "idDep": 17,
4      "nombreDepartamento": "Cochabamba",
5      "casosSospechosos": 70,
6      "casosConfirmados": 26,
7      "casosRecuperados": 10
8    },
9    {
10     "idDep": 18,
11     "nombreDepartamento": "new dep",
12     "casosSospechosos": 20,
13     "casosConfirmados": 5,
14     "casosRecuperados": 10
15   }
16 ]
```

▶ DELETE dep

DELETE ▼

http://localhost:8083/api/v1/corona/18

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

Text ▼



1

person successfully deleted