



UNIVERSIDAD PRIVADA FRANZ TAMAYO

DEFENSA HITO 3 - TAREA FINAL

Nombre Completo: **Univ. Juan Elían Álvarez Vallejos**

Asignatura: **PROGRAMACIÓN III**

Carrera: **INGENIERÍA DE SISTEMAS**

Paralelo: **PROG (1)**

Docente: **Lic. William R. Barra Paredes**

Fecha: **08/05/2020**

github:

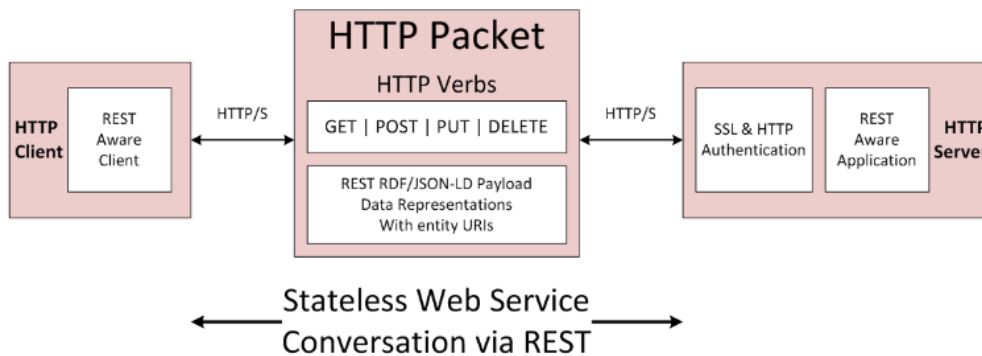
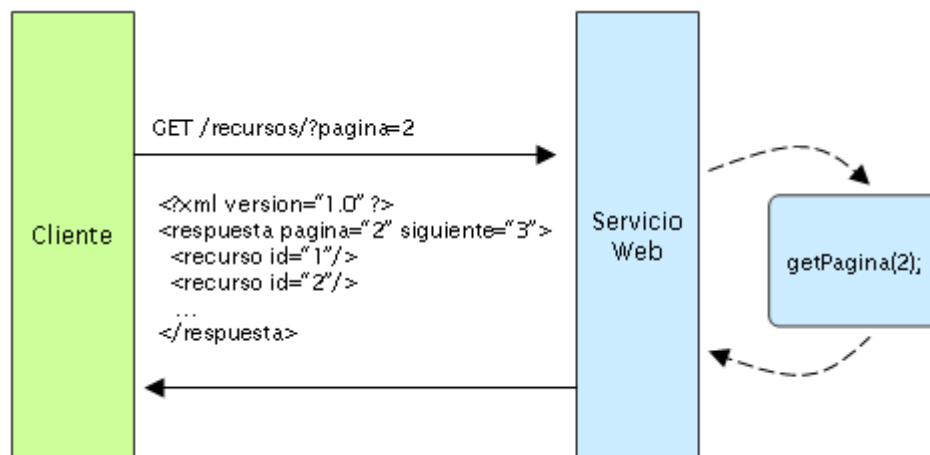
<https://github.com/JNELE/prograiii/tree/master/HITO%203>

Parte Teórica.

1. Preguntas.

Defina y muestre ejemplo de un servicio REST.

“Representational State Transfer” es un conjunto de principios de arquitectura para describir interfaces entre sistemas que use HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, no importa el formato (XML, JSON, etc.)



```
package com.chuidiang.examples.restful;

import java.util.LinkedList;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;

@Path("/service/")
public class RestfulService {

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public List<Data> getData(){
        List<Data> result = new LinkedList<>();
        result.add(new Data(1, "one"));
        result.add(new Data(2, "two"));
        return result;
    }

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    @Path("{id}")
    public Data getData(@PathParam("id") String id){
        if ("1".equals(id)){
            return new Data(1, "one");
        }
        if ("2".equals(id)){
            return new Data(2, "two");
        }
        throw new WebApplicationException(404);
    }
}
```

Que es JPA y como configurar en un entorno Spring.

Java Persistence API (**JPA**) es un framework que forma parte de Java, y ofrece un conjunto de interfaces y APIs para resolver el problema del almacenamiento de los objetos en una base de datos relacional.

Para configurar un proyecto utilizando JPA. Usaremos en un proyecto Maven creado a través de **Spring Boot** con todas sus dependencias, incluyendo las de Spring Data.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6   <groupId>com.arquitecturajava</groupId>
7   <artifactId>springData</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9   <packaging>jar</packaging>
10
11   <name>springData</name>
12   <description>Demo project for Spring Boot</description>
13
14   <parent>
15     <groupId>org.springframework.boot</groupId>
16     <artifactId>spring-boot-starter-parent</artifactId>
17     <version>1.3.6.RELEASE</version>
18     <relativePath /> <!-- lookup parent from repository -->
19   </parent>
20
21   <properties>
22     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23     <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24     <java.version>1.8</java.version>
25   </properties>
26
27   <dependencies>
28     <dependency>
29       <groupId>org.springframework.boot</groupId>
30       <artifactId>spring-boot-starter-data-jpa</artifactId>
31     </dependency>
32
33     <dependency>
34       <groupId>org.springframework.boot</groupId>
35       <artifactId>spring-boot-starter-test</artifactId>
36       <scope>test</scope>
37     </dependency>
38
39     <dependency>
40       <groupId>mysql</groupId>
41       <artifactId>mysql-connector-java</artifactId>
42       <version>5.1.39</version>
43     </dependency>
44
45   </dependencies>
46
47
48   <build>
49     <plugins>
50       <plugin>
51         <groupId>org.springframework.boot</groupId>
52         <artifactId>spring-boot-maven-plugin</artifactId>
53       </plugin>
54     </plugins>
55   </build>
56
57
```



Dependencias

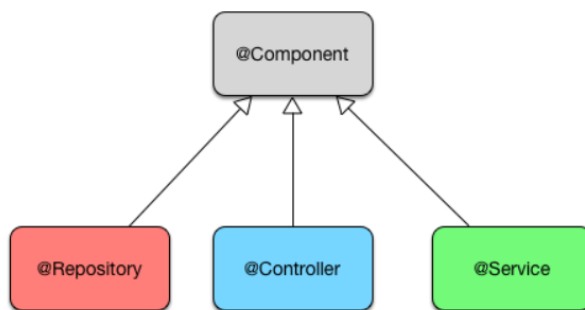
Que es MAVEN - POM.

Es una herramienta para la gestión y construcción de proyectos java, basado en el concepto POM (Proyect Object Model). También se puede generar arquetipos, gestionar librerías, compilar, empaquetar, generar documentación. Como se basa en patrones y estándares y trabaja con arquetipos, los cuales son configurables a través de su fichero protagonista, el pom.xml. Donde se puede configurar muchos aspectos de nuestro proyecto.

Qué son los Spring estereotipos y anotaciones muestre ejemplos.

Spring define un conjunto de anotaciones core que categorizan cada uno de los componentes asociandoles una responsabilidad concreta.

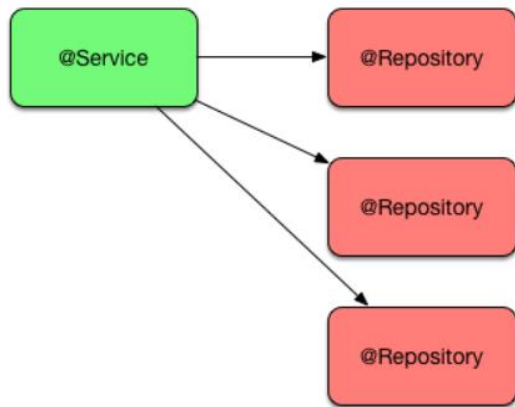
@Component: Es el estereotipo general y permite anotar un bean para que Spring lo considere uno de sus objetos.



@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.



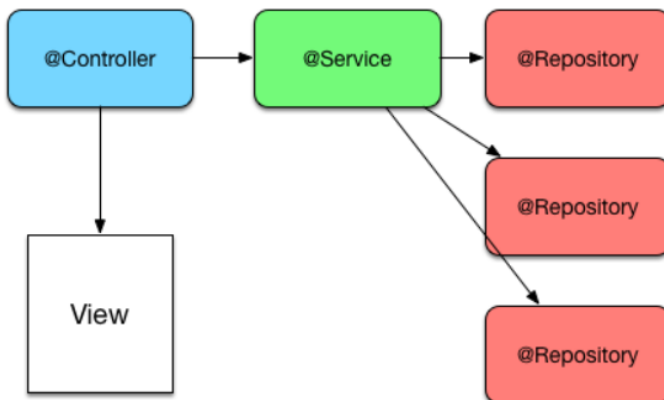
@Service: Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Su tarea fundamental es la de agregador.



```
@Service
public class VirusService implements VirusServiceInterfaz{
    @Autowired
    private CasosRepo casosRepo;

    @Override
    public CasosModel save(CasosModel cModel) {
        return casosRepo.save(cModel);
    }
}
```

@Controller: El último de los estereotipos que es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. Para ello se apoya habitualmente en algún motor de plantillas o librería de etiquetas que facilitan la creación de páginas.



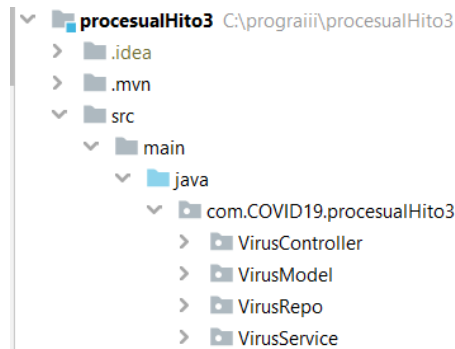
```
@RestController
@RequestMapping(value = "/api/v1")
public class UserControllerRest {
```

Describe las características principales de REST.

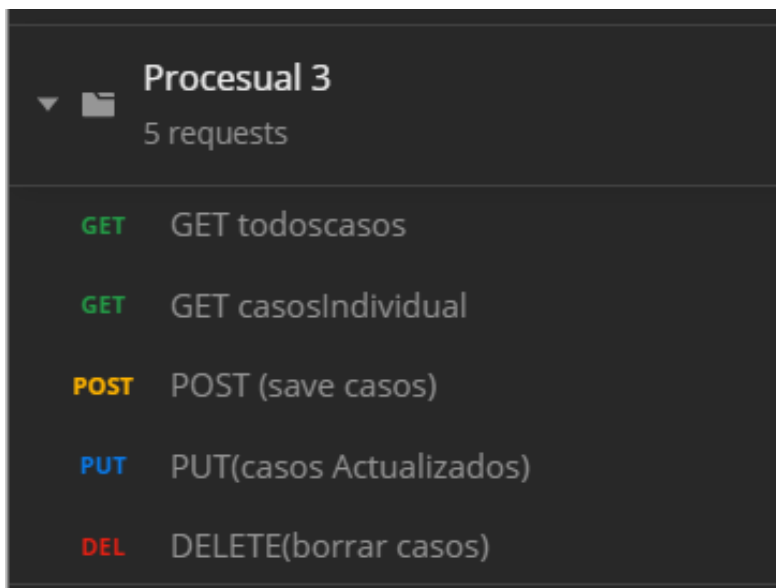
- **Protocolo cliente/servidor sin estado:** cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como **protocolo cliente-caché-servidor sin estado:** existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro **la misma respuesta para peticiones idénticas**. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro:
 - GET: recoge información de un recurso
 - PUT: modifica o actualiza el estado de un recurso
 - POST: crea un nuevo recurso en el servidor
 - DELETE: elimina un recurso del servidor.
- **Los objetos en REST siempre se manipulan a partir de la URI.** Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.
- **Interfaz uniforme:** para la 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 facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- **Sistema de capas:** arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
- **Uso de hipermedios:** hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Parte Práctica.

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.

- ii. Para este propósito la primera fase de desarrollo es la creación de un servicio rest que pueda crear, modificar y retornar estos datos.
- iii. Se tiene como base principal la siguiente tabla que nos servirá para poder generar toda esta información.

Bases de Datos

corona_virus	
 id_corona_virus	integer
 nombre_dep	varchar(50)
 casos_contagiados	integer
 casos_sospechosos	integer
 casos_recuperados	integer

El DDL de la base de datos


```
-- auto-generated definition
create table corona_virus
(
    idcoronavirus    integer    not null
        constraint corona_virus_pkey
            primary key,
    casoscontagiados integer,
    casosrecuperados integer,
    casossospechosos integer,
    nombredp         varchar(50) not null
);

alter table corona_virus
    owner to nnqpuwzgigklw;
```

La base de datos se crea desde el código que se generó en la clase.java MODEL donde se muestra más adelante; en esa clase se diseña la tabla y a su vez sus columnas y los valores que tendrán.

Codigo del intellij Idea

Controller – Model – Repo - Service

CONTROLLER – clase  Class

```
package com.COVID19.procesualHito3.VirusController;

import com.COVID19.procesualHito3.VirusModel.CasosModel;
import com.COVID19.procesualHito3.VirusService.VirusService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping(value = "/api/v1")
public class UserControllerRest {

    @Autowired
    private VirusService virusService;

    @GetMapping("/coronaVirus")
    public ResponseEntity<List<CasosModel>> getAllcasos() {
        try {
            List<CasosModel> persons = virusService.getAllcasos();

            if (persons.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            } else {
                return new ResponseEntity<>(persons, HttpStatus.OK);
            }
        } catch (Exception e) {
            return new ResponseEntity<>(null,
                HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/coronaVirus/getOne/{idcoronavirus}")
    public ResponseEntity<CasosModel>
    getcasosByIdcoronavirus(@PathVariable("idcoronavirus") Integer idcoronavirus)
    {
        try {
            CasosModel cModel =
```

```

virusService.getcasosByIdcoronavirus(idcoronavirus);

        if (cModel != null) {
            return new ResponseEntity<>(cModel, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

@PostMapping("/coronaVirus")
public ResponseEntity save(@RequestBody CasosModel virus){
    try{
        return new ResponseEntity<>(virusService.save(virus),
HttpStatus.CREATED);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}

@DeleteMapping("/coronaVirus/deleteCV/{idCoronaVirus}")
public ResponseEntity<String> delete(@PathVariable("idCoronaVirus")
Integer idcoronavirus) {
    try {
        virusService.delete(idcoronavirus);
        return new ResponseEntity<>("Departamento successfully deleted",
HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}

@PutMapping("/coronaVirus/{idCoronaVirus}")
public ResponseEntity<CasosModel>
updateCasos(@PathVariable("idCoronaVirus") Integer idcoronavirus, @RequestBody
CasosModel cModel) {
    try {
        CasosModel cUpdate = virusService.update(cModel, idcoronavirus);
        if (cUpdate != null) {
            return new ResponseEntity<>(cUpdate, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
}

```

MODEL – clase Class

```
package com.COVID19.procesualHito3.VirusModel;

import javax.persistence.*;

@Entity
@Table(name = "corona_virus")
public class CasosModel {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer idcoronavirus;

    @Column(name = "nombredep", length = 50, nullable = false)
    private String nombredep;

    @Column(name = "casoscontagiados")
    private int casoscontagiados;

    @Column(name = "casossospechosos")
    private int casossospechosos;

    @Column(name = "casosrecuperados")
    private int casosrecuperados;

    public Integer getIdcoronavirus() {
        return idcoronavirus;
    }

    public void setIdcoronavirus(Integer idcoronavirus) {
        this.idcoronavirus = idcoronavirus;
    }

    public String getNombredep() {
        return nombredep;
    }

    public void setNombresdep(String nombresdep) {
        this.nombredep = nombresdep;
    }

    public int getCasoscontagiados() {
        return casoscontagiados;
    }

    public void setCasoscontagiados(int casoscontagiados) {
        this.casoscontagiados = casoscontagiados;
    }

    public int getCasossospechosos() {
        return casossospechosos;
    }
}
```

```

    }

    public void setCasossospechosos(int casossospechosos) {
        this.casossospechosos = casossospechosos;
    }

    public int getCasosrecuperados() {
        return casosrecuperados;
    }

    public void setCasosrecuperados(int casosrecuperados) {
        this.casosrecuperados = casosrecuperados;
    }
}

```

REPO – interfaz Interface

```

package com.COVID19.procesualHito3.VirusRepo;

import com.COVID19.procesualHito3.VirusModel.CasosModel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CasosRepo extends JpaRepository<CasosModel, Integer> {

}

```

SERVICE – clase Class

```

package com.COVID19.procesualHito3.VirusService;

import com.COVID19.procesualHito3.VirusModel.CasosModel;
import com.COVID19.procesualHito3.VirusRepo.CasosRepo;
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 VirusService implements VirusServiceInterfaz{
    @Autowired
    private CasosRepo casosRepo;

    @Override
    public CasosModel save(CasosModel cModel) {
        return casosRepo.save(cModel);
    }
}

```

```

@Override
public CasosModel update(CasosModel cModel, Integer idcoronavirus) {
    Optional<CasosModel> casos = casosRepo.findById(idcoronavirus);
    CasosModel casosUpdate = null;

    if (casos.isPresent()) {
        casosUpdate = casos.get();
        casosUpdate.setNombresdep(cModel.getNombresdep());
        casosUpdate.setCasoscontagiados(cModel.getCasoscontagiados());
        casosUpdate.setCasossospechosos(cModel.getCasossospechosos());
        casosUpdate.setCasosrecuperados(cModel.getCasosrecuperados());
        casosRepo.save(casosUpdate);
    }
    return casosUpdate;
}

@Override
public Integer delete(Integer idcoronavirus) {
    casosRepo.deleteById(idcoronavirus);
    return 1;
}

@Override
public List<CasosModel> getAllcasos() {
    List<CasosModel> casos = new ArrayList<CasosModel>();
    casosRepo.findAll().forEach(casos::add);

    return casos;
}

@Override
public CasosModel getcasosByIdcoronavirus(Integer idcoronavirus) {
    Optional<CasosModel> casos = casosRepo.findById(idcoronavirus);
    CasosModel cModel = null;

    if (casos.isPresent()) {
        cModel = casos.get();
    }
    return cModel;
}
}

```

SERVICE – interfaz Interface

```
package com.COVID19.procesualHito3.VirusService;











import com.COVID19.procesualHito3.VirusModel.CasosModel;

import java.util.List;

public interface VirusServiceInterfaz {
    public CasosModel save(CasosModel cModel);
    public CasosModel update(CasosModel cModel, Integer idcoronavirus);
    public Integer delete(Integer idcoronavirus);
    public List<CasosModel> getAllcasos();
    public CasosModel getcasosByIdcoronavirus(Integer idcoronavirus);
}
```

REST – API

Generar los siguientes servicios.

POST	/coronaVirus	Adds the new case CV in the store		
PUT	/coronaVirus/{idCoronaVirus}	Updates a specific CV row		
GET	/coronaVirus/getOne/{idCoronaVirus}	Find CV row by ID		
GET	/coronaVirus/	Gets all cv records		
DELETE	/coronaVirus/deleteCV/{idCoronaVirus}	Deletes a CV		

- POST – para guardar los casos que se registren en los departamentos

POST /coronaVirus Adds the new case CV in the store

Parameters Try it out

Name	Description
body * required	Parametros necesarios para la creacion.
object (body)	<div>Example Value Model</div> <pre>{ "idCoronaVirus": 0, "nombreDep": "Cochabamba", "casosContagiados": 56, "casosSospechosos": 120, "casosRecuperados": 7 }</pre>
Parameter content type application/json	

Responses Response content type application/json

Code	Description
201	Created
471	Expectation Failed

Donde se manda los datos a guardar

POST POST (save casos) No Environment

POST (save casos) Comments 0

POST http://localhost:8083/api/v1/coronaVirus Send

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

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

```
1 {
2   "idcoronavirus": 0,
3   "nombredep": "SantaCruz",
4   "casoscontagiados": 1098,
5   "casosospechosos": 87,
6   "casosrecuperados": 20
7 }
8 }
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 1159 ms Size: 281 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "idcoronavirus": 15,
3   "nombredep": "SantaCruz",
4   "casoscontagiados": 1098,
5   "casosospechosos": 87,
6   "casosrecuperados": 20
7 }
```


Datos que se guardaron

Q* <Filter criteria>

	idcoronavirus	casoscontagiados	casosrecuperados	casosospechosos	nombredep
1	1	300	89	150	LaPaz
2	2	89	10	15	Cochabamba
3	15	1098	20	87	SantaCruz

Una vez que se manda los datos mediante el postman en formato Json se conecta con la base de datos y guarda la información que se mandó.

- PUT – sirve para modificación y actualización de los casos que se presentan

The screenshot shows a REST client interface for a PUT request. The URL is `/coronaVirus/{idCoronaVirus}` with the description "Updates a specific CV row".

Parameters: A "Try it out" button is present.

body * required: The description is "Parametros necesarios para la modificacion". The body is an object. An example JSON value is shown in a dark box:

```
{
  "idCoronaVirus": 0,
  "nombreDep": "Cochabamba",
  "casosContagiados": 56,
  "casosSospechosos": 120,
  "casosRecuperados": 7
}
```

The "Parameter content type" is set to `application/json`.

idCoronaVirus * required: The description is "ID of Corona Virus". The type is `integer($int64)`. The path is `idCoronaVirus - ID of Corona Virus`.

Responses: The "Response content type" is set to `application/json`.

Code	Description
200	Corona Virus updated
404	Corona Virus not found
417	Expectation Failed

Datos anteriores

	idcoronavirus	casoscontagiados	casosrecuperados	casosospechosos	nombredep
1	1	300	89	150	LaPaz
2	2	89	10	15	Cochabamba
3	15	1098	20	87	SantaCruz

Donde se ponen los datos que se cambiaran

PUT PUT(casos Actualizados) No Environment

PUT(casos Actualizados) Comments Ex

PUT http://localhost:8083/api/v1/coronaVirus/2 Send

Params Authorization Headers (3) Body Pre-request Script Tests Settings

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

```
1 {
2   "nombredep": "Cochabamba",
3   "casoscontagiados": 150,
4   "casosospechosos": 87,
5   "casosrecuperados": 15
6 }
7
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 747 ms Size: 275 B Save

Pretty Raw Preview Visualize JSON

```
1 {
2   "idcoronavirus": 2,
3   "nombredep": "Cochabamba",
4   "casoscontagiados": 150,
5   "casosospechosos": 87,
6   "casosrecuperados": 15
7 }
```

Datos modificados

	idcoronavirus	casoscontagiados	casosrecuperados	casosospechosos	nombredep
1	1	300	89	150	LaPaz
2	15	1098	20	87	SantaCruz
3	2	150	15	87	Cochabamba

En caso que este mal algún dato o se actualice los casos que presentan el departamento se utiliza el PUT; primero se manda el id del departamento para encontrar el departamento y seguido se cambian los datos que sea necesario y al final se actualiza.

- GET – para obtener los casos de un departamento en específico

GET `/coronaVirus/getOne/{idCoronaVirus}` Find CV row by ID

Returns a single CV

Parameters Try it out

Name	Description
idCoronaVirus <small>* required</small> integer(\$int64) (path)	ID of Corona Virus idCoronaVirus - ID of Corona Virus

Responses Response content type: application/json

Code	Description
200	successful operation Example Value Model <pre>{ "idCoronaVirus": 0, "nombreDep": "Cochabamba", "casosContagiados": 56, "casosSospechosos": 120, "casosRecuperados": 7 }</pre>
404	Pet not found
500	Internal server error

Donde se coloca la id del departamento

GET `casosIndividual` No Environment

GET `casosIndividual` Comments 0

GET `http://localhost:8083/api/v1/coronaVirus/getOne/15` Send

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

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

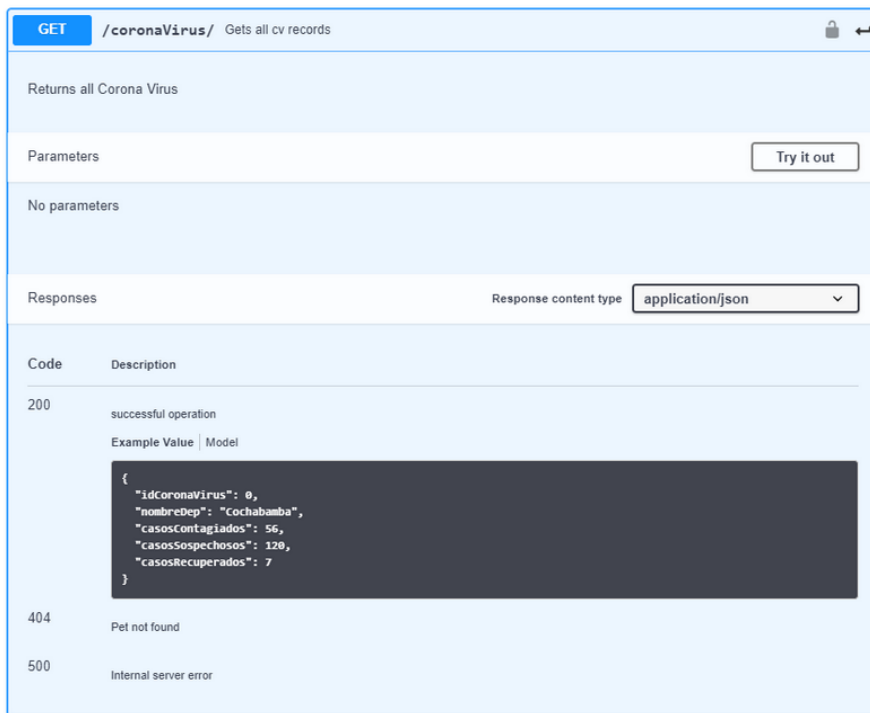
Body Cookies Headers (5) Test Results Status: 200 OK Time: 536 ms Size: 276 B

Pretty Raw Preview Visualize JSON 5

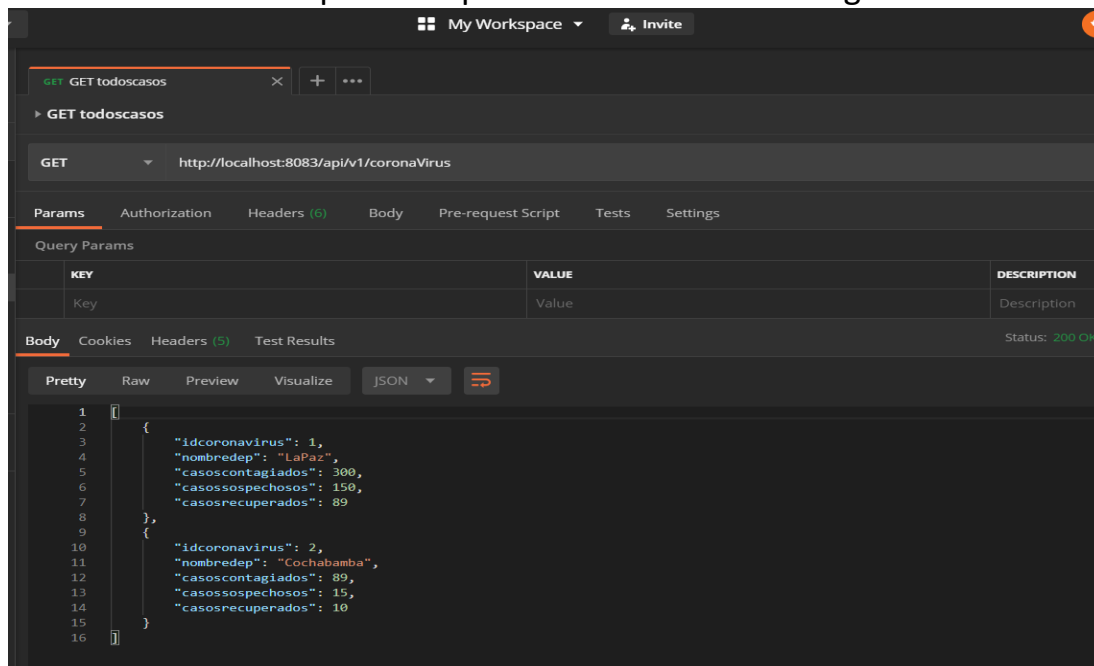
```
1 {
2   "idcoronavirus": 15,
3   "nombredep": "SantaCruz",
4   "casoscontagiados": 1098,
5   "casosospechosos": 87,
6   "casosrecuperados": 20
7 }
```

Se coloca la id del departamento para obtener toda su información de sus casos, y la información que muestra es la del departamento.

- GET – para obtener todo los casos registrados de todos los departamentos

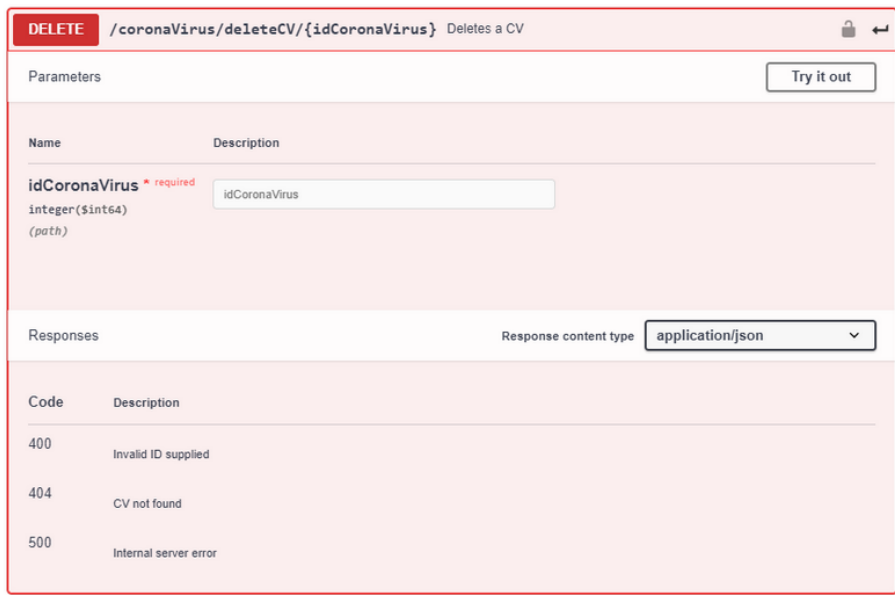


Donde se coloca la operación para mostrar todos los registros



Se manda el comando en la URL para ejecutarlo y así mostrar todos los registros.

DELETE – para eliminar algún registró



Swagger UI interface for the DELETE endpoint `/coronaVirus/deleteCV/{idCoronaVirus}`. The interface shows the method **DELETE** and the description "Deletes a CV".

Parameters:

Name	Description
idCoronaVirus * required	
integer(\$int64)	
(path)	

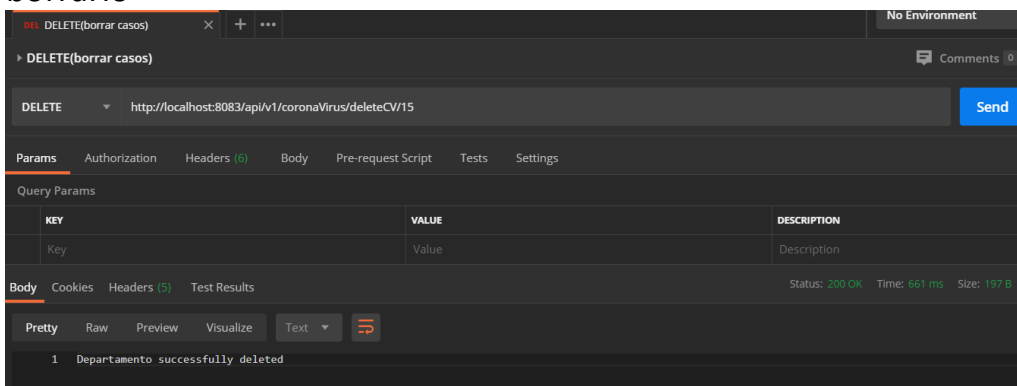
Responses:

Code	Description
400	Invalid ID supplied
404	CV not found
500	Internal server error

Registros antes de la eliminacion


	idcoronavir	casoscontagiados	casosrecuperados	casossospechosos	nombredep
1	1	300	89	150	LaPaz
2	15	1098	20	87	SantaCruz
3	2	150	15	87	Cochabamba

Donde se coloca el id del departamento para buscar su registro y poder borrarlo



Postman interface showing a DELETE request to `http://localhost:8083/api/v1/coronaVirus/deleteCV/15`. The status is **200 OK**, time is **661 ms**, and size is **197 B**. The response body shows: `1 Departamento successfully deleted`.

Después de que se eliminó al registro de SantaCruz

	 idcoronavirus ↕	 casoscontagiados ↕	 casosrecuperados ↕	 casossospechosos ↕	 nombredep ↕
1	1	300	89	150	LaPaz
2	2	150	15	87	Cochabamba