UNIVERSIDAD PRIVADA FRANZ TAMAYO

CARRERA INGENIERIA DE SISTEMAS

HITO3 – TAREA FINAL

MANEJO DE SPRING FRAMEWORK

Alumno: Adriana Contreras Arancibia

Materia: Programación III

Carrera: Ing. De Sistemas

Paralelo: Progra(1)

Docente: Lic. William R. Barra Paredes

Fecha: 10/04/2020

Github:

Cochabamba-Bolivia

TAREA FINAL HITO 3

MANEJO DE SPRING FRAMEWORK (REST API)

1.TEORIA

1.1 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.

Ejemplo

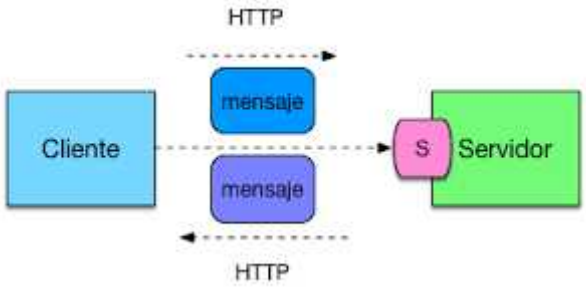


Figura I

ejemplo grafico de servicio rest(fuente arquitecturajava.com)

```
@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);
}
}
```

Figura II

ejemplo de parte de código de un servicio rest (Fuente arquitecturajava)

1.2 QUE ES JPA Y COMO CONFIGURAR EN UN ENTORNO SPRING.

Java Persistence API es un conjunto de clases y métodos que persistentemente almacenar la gran cantidad de datos a una base de datos que es proporcionada por Oracle Corporation, Para reducir la carga de escribir códigos relacionales para gestión de objetos, un programador sigue el "Proveedor" marco JPA, que permite la fácil interacción con instancia de la base de datos.

En un entorno Spring permite creación automática de beans de acceso a datos mediante la anotación @Repository. Asimismo, Spring reconoce las anotaciones del API estándar JPA, asimismo JPA permite el mapeo entre los campos del objeto y aquellos de la base de datos

Ejemplo

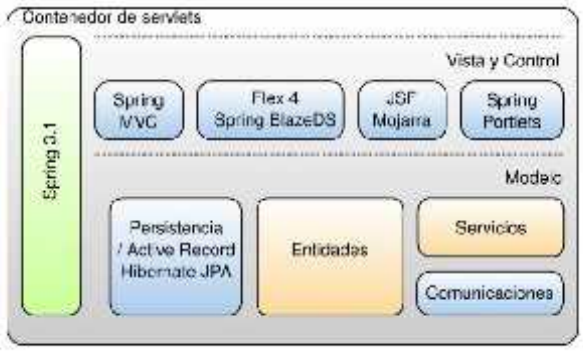


Figura III

Arquitectura entorno a la ejecución Modelo – Vista – controlador (Fuente slideshare)

1.3 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.

Figura IV

Maven aporta una semántica común al proceso de build y desarrollo del software(Fuente javiergarzas)

- POM

esponde 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

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.

```
    </plugins>
    <finalName>MavenEnterpriseApp-ear</finalName>
  </build>
    <dependencies>
        <dependency>
            <groupId>com.mycompany</groupId>
            <artifactId>MavenEnterpriseApp-ejb</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>ejb</type>
        </dependency>
        <dependency>
            <groupId>com.mycompany</groupId>
            <artifactId>MavenEnterpriseApp-web</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>war</type>
        </dependency>
    </dependencies>
</project>
```

Figura V

Imagen de las dependencias Maven POM.xml (Fuente dineshonjava)

1.4 QUÉ SON LOS SPRING ESTEREOTIPOS Y ANOTACIONES MUESTRE EJEMPLOS

Los estereotipos son un conjunto de anotaciones core que categorizan cada uno de los componentes asociándoles una responsabilidad concreta.

@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.

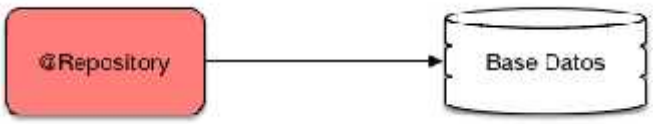


Figura VI

@Repository

@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.

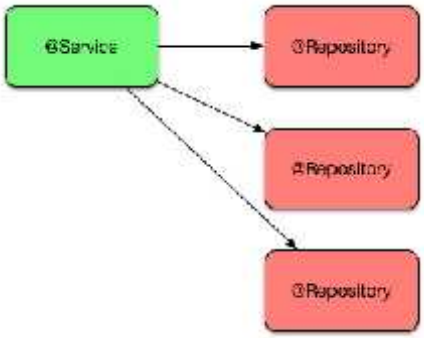


Figura VII

@Service

@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.

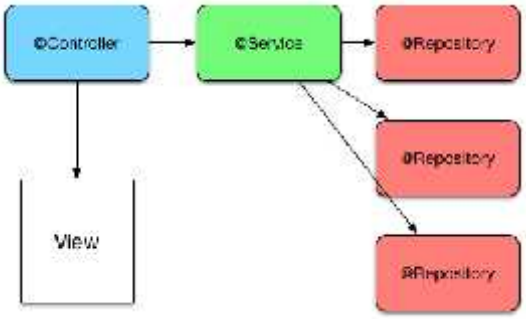


Figura VII

@Controller

@RestController que es una especialización de controller que contiene las anotaciones @Controller y @ResponseBody

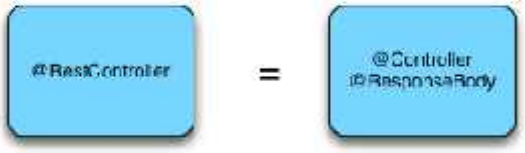


Figura IX

@RestController

1.5 DESCRIBA LAS CARACTERÍSTICAS PRINCIPALES DE REST.

Características 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.
- **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 (eliminar).
- **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
- **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:** 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.

2. PRACTICA

2.1 CREAR LOS PACKAGES NECESARIOS (DEBE REFLEJARSE EL MODELO MVC).

Figura X

Creación de los packages controller, modelo, repositorio y servicio, los cuales son necesarios para tener un modelo MVC(elaboración fuente propia, herramienta IntellJIdea)

2.2 BASE DE DATOS

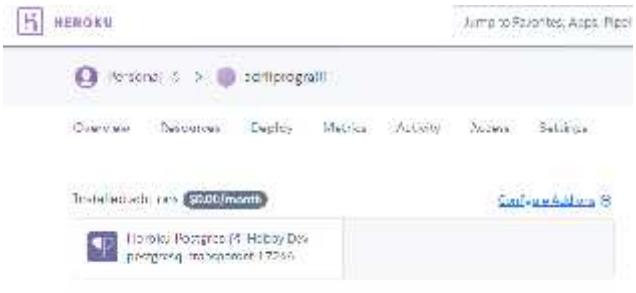


Figura XI

Creación de una base de datos la cual nos permitirá guardar los registros de coronavirus en un determinado departamento (elaboración propia, herramienta Heroku)

```
-- auto-generated definition
create table corona_virus
(
    id_corona_virus   integer     not null
        constraint corona_virus_pkey
            primary key,
    casos_contagiados integer,
    casos_recuperados integer,
    casos_sospechosos integer,
    nombre_dep        varchar(50) not null
);

alter table corona_virus
    owner to aecovanrdqbkfr;
```

DDL de la tabla generada en la base de datos

2.3 REST API

Generar los siguientes servicios

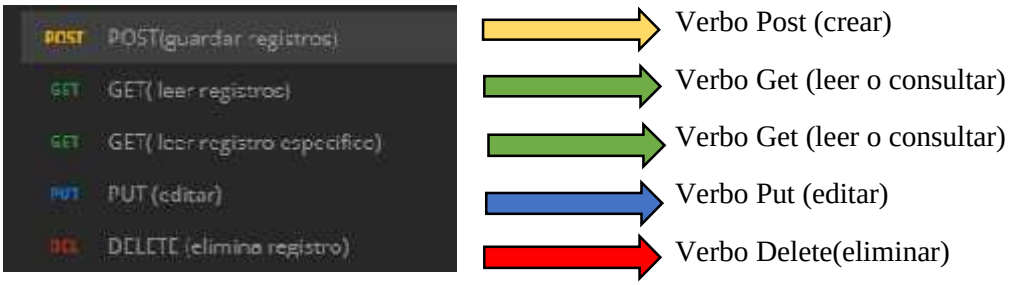


Figura XII

Teniendo conocimiento de los verbos http, podemos usarlos para crear, leer, editar y eliminar (elaboración propia, herramienta Postman)

2.3.1 POST

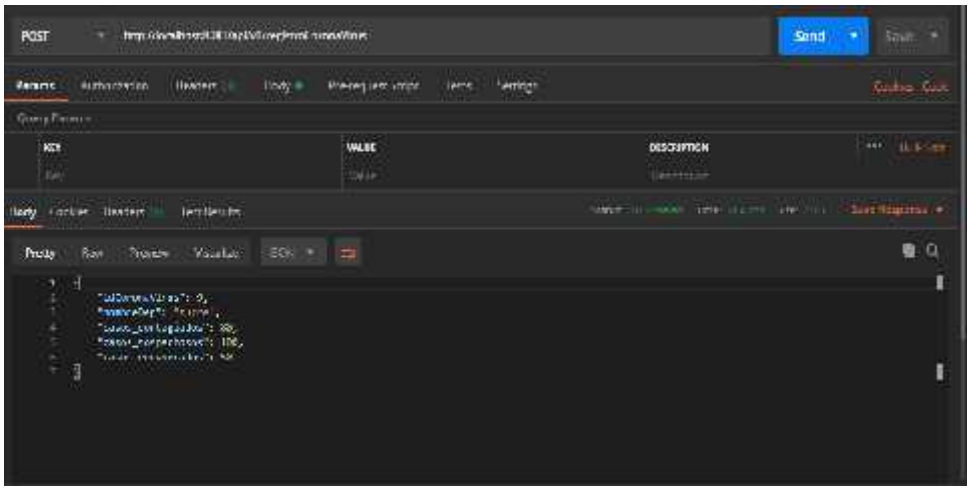
- Propuesto

Comportamiento esperado:

- Captura resolución

```
//crear registros de corona virus
@PostMapping("/registroCoronaVirus")
public ResponseEntity save(@RequestBody CoronaVirusModel persona) {
    try {
        return new ResponseEntity<>(coronaService.save(persona), HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>( "Error: null, HttpStatus.EXPECTATION_FAILED);
    }
}
```

- Captura Postman, verbo POST (crear)



- Código que resuelve el problema

```
//crear registros de corona virus
@PostMapping("/registroCoronaVirus")
public ResponseEntity save(@RequestBody CoronaVirusModel persona) {
    try {
        return new ResponseEntity<>(coronaservice.save(persona),
HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(null,
HttpStatus.EXPECTATION_FAILED);
    }
}
```

Código de la funcion que permite guardar registros, que se encuentra dentro de la clase UserControllerRest

```
@Override
public CoronaVirusModel save(CoronaVirusModel pModel) {
    return coronaRepo.save(pModel);
}
```

Código de la funcion save, este método se encuentra dentro de la clase CoronaService

- Logrado

Mediante el uso del verbo POST podemos crear un nuevo registro de Corona virus, llenando los campos específicos de la tabla que son el nombre del departamento, número de casos confirmados, sospechosos y recuperados, posteriormente estos registros se agregan a la base de datos en la tabla “Coronavirus” generando un id auto incrementable para cada registro.

2.3.2 PUT

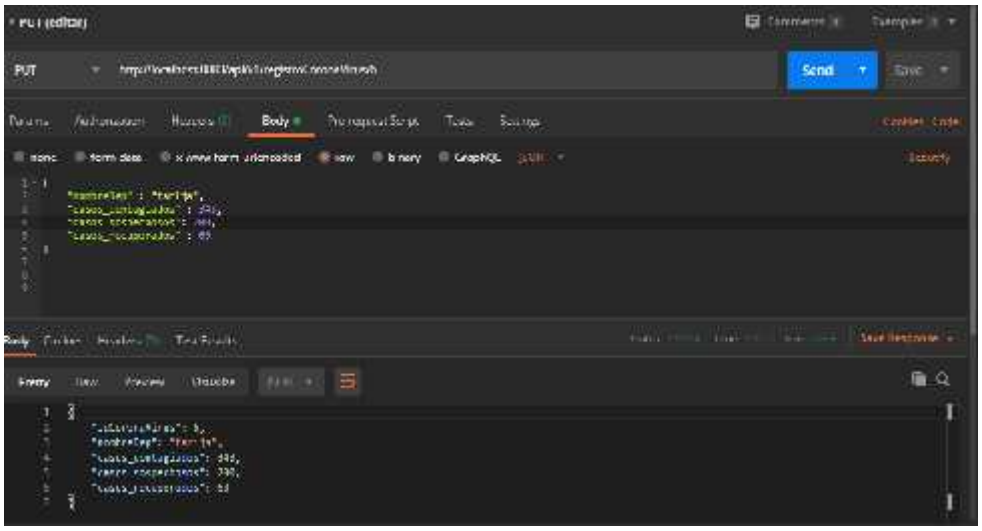
- Propuesto

  Comportamiento esperado:

  Solución:

- Captura resolución

- Captura Postman, verbo PUT (editar)



- Código que resuelve el problema

```
//editar registro
@PutMapping("/registroCoronaVirus/{idCorona}")
public ResponseEntity<CoronaVirusModel> updateMaterias(@PathVariable("idCorona")
Integer idPer, @RequestBody CoronaVirusModel pModel) {
    try {
       CoronaVirusModel pUpdate = coronaservice.update(pModel, idPer);///
        if (pUpdate != null) {
            return new ResponseEntity<>(pUpdate, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}
```

Código de la funcion que permite editar registros, que se encuentra dentro de la clase UserControllerRest

```
@Override
public CoronaVirusModel update(CoronaVirusModel pModel, Integer idPer) {
    Optional<CoronaVirusModel> registroCoronaVirus = coronaRepo.findById(idPer);
   CoronaVirusModel personaUpdate = null;

    if (registroCoronaVirus.isPresent()) {
        personaUpdate = registroCoronaVirus.get();
        personaUpdate.setNombreDep(pModel.getNombreDep());
        personaUpdate.setCasos_contagiados(pModel.getCasos_contagiados());
        personaUpdate.setCasos_sospechosos(pModel.getCasos_sospechosos());
        personaUpdate.setCasos_recuperados(pModel.getCasos_recuperados());
        coronaRepo.save(personaUpdate);
    }
    return personaUpdate;
}
```

Código de la funcion update, este método se encuentra dentro de la clase CoronaService

- Logrado

Mediante el uso del verbo PUT podemos editar un registro de Corona virus siempre que ingresemos el id del registro, luego debemos rellenar los campos específicos en el body o cuerpo que está en formato Json, posteriormente apretamos el botón “send” para llevar a cabo el editado del registro en nuestra base de datos en la tabla “Coronavirus.

2.3.3 GET

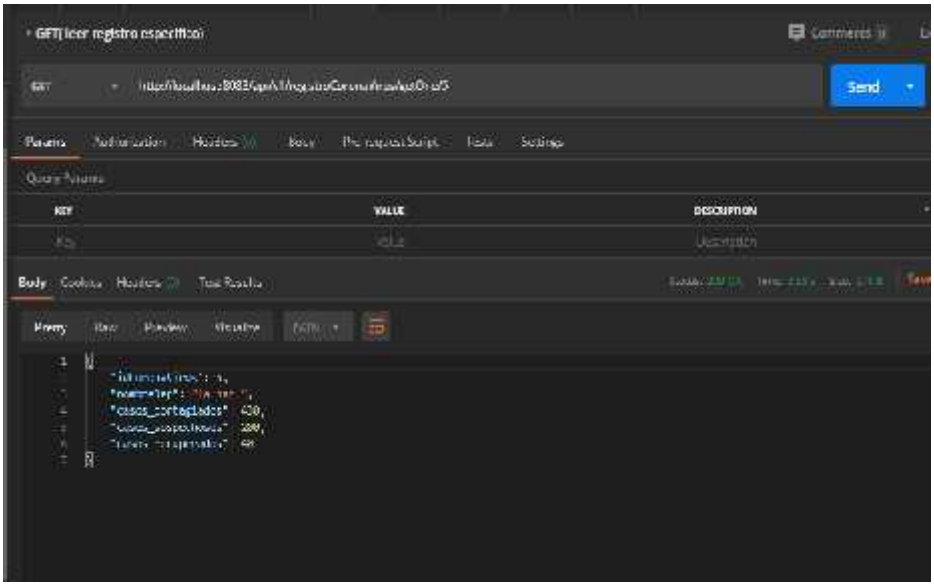
- Propuesto

  Comportamiento esperado:

  Solucion:

- Captura resolución IntellJidea

- Captura Postman, verbo GET (leer)



- Código que resuelve el problema

```
// leer un registro especifico

    @GetMapping("/registroCoronaVirus/getOne/{idCorona}")
    public ResponseEntity<CoronaVirusModel>
getPersonByIdPer(@PathVariable("idCorona") Integer idCorona) {
        try {
             CoronaVirusModel pModel = coronaservice.getPersonByIdPer(idCorona);

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


}
```

Código de la funcion que permite leer un registro especifico, que se encuentra dentro de la clase UserControllerRest

```
@Override
public CoronaVirusModel getCoronaByIdCorona(Integer idPer) {
    Optional<CoronaVirusModel> person = coronaRepo.findById(idPer);
    CoronaVirusModel pModel = null;

    if (person.isPresent()) {
        pModel = person.get();
    }
    return pModel;
}
```

Código de la funcion getCoronaByIdCorona este método se encuentra dentro de la clase CoronaService

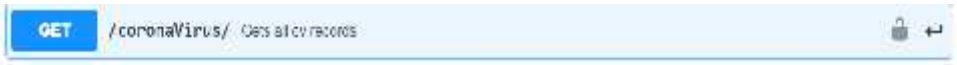
- Logrado

Mediante el uso del verbo GET podemos consultar un registro especifico de Corona virus y no así todos los registros a la vez siempre que ingresemos RegistroCoronaVirus, getOne y el id del registro dentro de la url, posteriormente de ser ejecutado nos mostrara solo el registro que hemos solicitado.

2.3.4 GET

- Propuesto

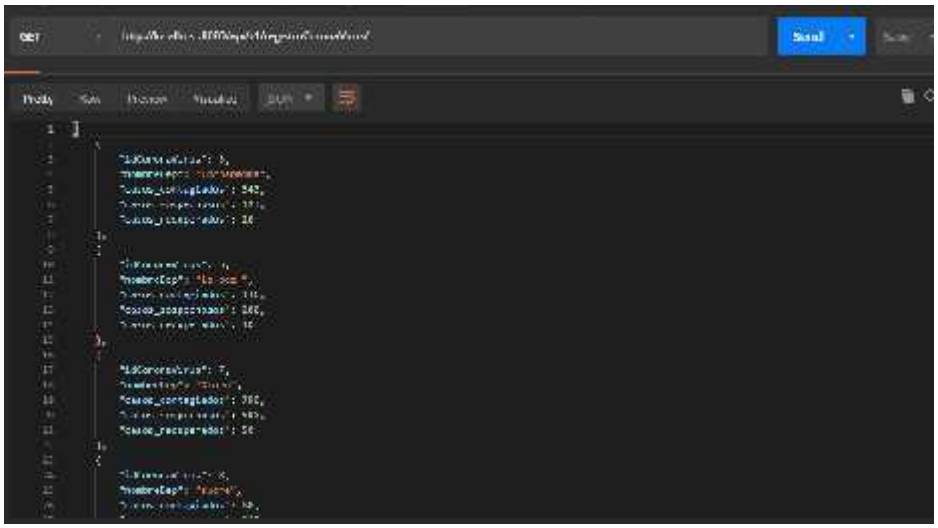
  Comportamiento esperado:



  Solucion:

- Captura resolución IntellJidea

- Captura Postman, verbo GET (leer)



- Código que resuelve el problema

```
//leer todoos  los registros o mostrar todos
@GetMapping("/registroCoronaVirus")
public ResponseEntity<List<CoronaVirusModel>> getAllPersons() {
    try {
        List<CoronaVirusModel> registroCoronaVirus =
coronaservice.getAllPersons();

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

Código de la funcion que permite leer todos los registros que están dentro de nuestra base de datos, se encuentra dentro de la clase UserControllerRest

```
//recorre los registros y añade a la lista
 @Override
 public List<CoronaVirusModel> getAllPersons() {
     List<CoronaVirusModel> persons = new ArrayList<CoronaVirusModel>();
     coronaRepo.findAll().forEach(persons::add);

     return persons;
 }
```

Código de la funcion getAll este método se encuentra dentro de la clase CoronaService

- Logrado

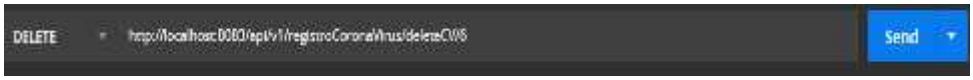
Mediante el uso del verbo GET podemos consultar un registro, en este caso consultaremos todos registros que han sido añadidos a nuestras tablas “coronavirus”

2.3.5 DELETE

- Propuesto

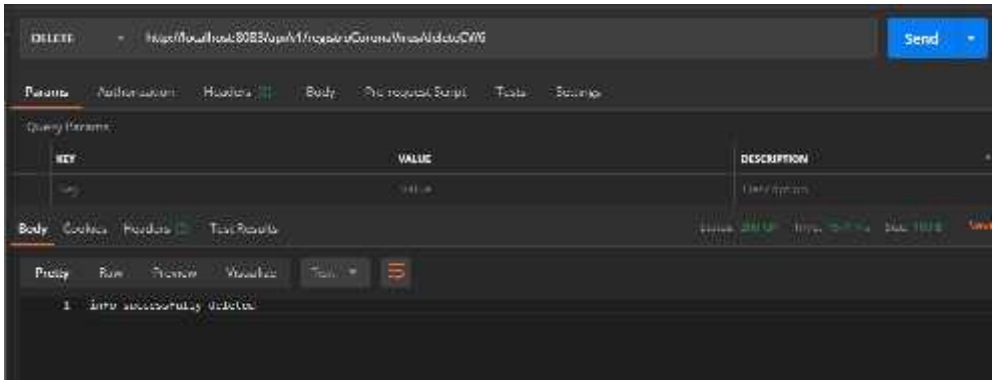
  Comportamiento esperado:

  Solucion:



- Captura resolución IntellJidea

- Captura Postman, verbo DELETE (eliminar)



- Código que resuelve el problema

```
//eliminar registro
@DeleteMapping("/registroCoronaVirus/deleteCV/{idCorona}")
public ResponseEntity<String> delete(@PathVariable("idCorona") Integer idPer) {
    try {
        coronaservice.delete(idPer);
        return new ResponseEntity<>("info successfully deleted", HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}
```

Código de la funcion que permite eliminar un registro especifico mediante el id, que se encuentra dentro de la clase UserControllerRest

```
//eliminar mediante id
@Override
public Integer delete(Integer idCorona) {
    coronaRepo.deleteById(idCorona);
    return 1;
}
```

Código de Delete este método se encuentra dentro de la clase CoronaService

- Logrado

Mediante el uso del verbo DELETE podemos eliminar un registro especifico de la tabla Corona virus y no así todos los registros a la vez siempre que ingresemos RegistroCoronaVirus, deleteCV y el id del registro que pretendemos eliminar, todo esto dentro de la url posteriormente de ser ejecutado nos mostrara el mensaje “info sucessfuly deleted” lo cual indica que efectivamente el registro ha sido borrado.