



# PROYECTO DE FIN DE HITO

NOMBRE : NICOLAS GONZALO AGUILAR ARIMOZA

SEMESTRE : 4 SEMESTRE

DOCENTE : DHEEYI WILLIAM

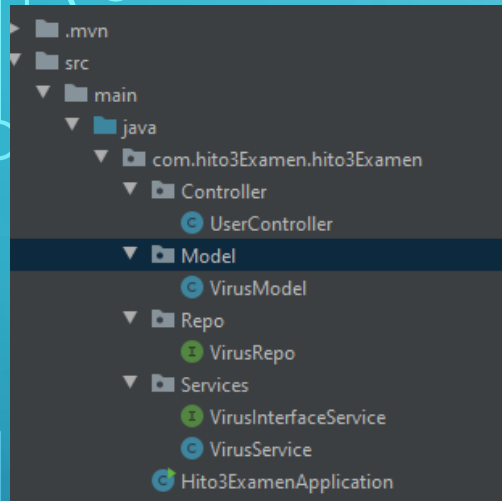
FECHA : 18/5/2020

# PRIMERA PREGUNTA

Crear el Entity para el modelo CoronavirusPaciente

corona_virus_pacie	
id_corona_virus	
nombre_dep	varchar
nombre_paciente	varchar
apellidos_paciente	varchar
edad_paciente	varchar
categoria	varchar
fullname	varchar
casos_contagiados	
casos_sospechosos	
casos_recuperados	

Primero creamos el PACKAGE MODEL y la clase VirusModel para crear las tablas columna por columna



Creacion de la  
clase

Creacion de las  
columnas 1 por 1

```
import javax.persistence.*;

@Entity
@Table(name = "corona_virus_paciente")
public class VirusModel {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int idCoronavirus;
    @Column(name = "nombresDepartamento", length = 50, nullable = false)
    private String nombresDepto;
    @Column(name = "nombresPaciente", length = 50, nullable = false)
    private String nombrePaciente;
    @Column(name = "apellidosPaciente", length = 50, nullable = false)
    private String apellidosPaciente;
    @Column(name = "edadPaciente")
    private int edadPaciente;
    @Column(name = "Categoria", length = 50, nullable = false)
    private String categoria;
    @Column(name = "fullname", length = 100, nullable = false)
    private String fullname;
    @Column(name = "CasosContagiados")
    private int casosContag;
    @Column(name = "CasosSostepochosos")
    private int casosSospechosos;
    @Column(name = "CasosRecuperados")
    private int casosRecuperados;
```

# POR ULTIMO EL DDL DEL DATAGRIP

```
-- auto-generated definition
create table corona_virus_paciente
(
    id_coronavirus      integer      not null
        constraint corona_virus_paciente_pkey
            primary key,
    apellidos_paciente  varchar(50)  not null,
    casos_contagiados   integer,
    casos_recuperados   integer,
    casos_sostepochosos integer,
    categoria            varchar(50)  not null,
    edad_paciente        integer,
    fullname             varchar(100) not null,
    nombres_paciente     varchar(50)  not null,
    nombres_departamento varchar(50)  not null
);

alter table corona_virus_paciente
    owner to tnfuzbvljgciin;
```

## PREGUNTA 2

- Generar el servicio
- Rest – Post para poder crear un nuevo caso

The screenshot shows a REST client interface for a POST request to the endpoint `/coronaVirusPaciente`. The description of the endpoint is "Adds the new case CV in the store".

**Parameters**

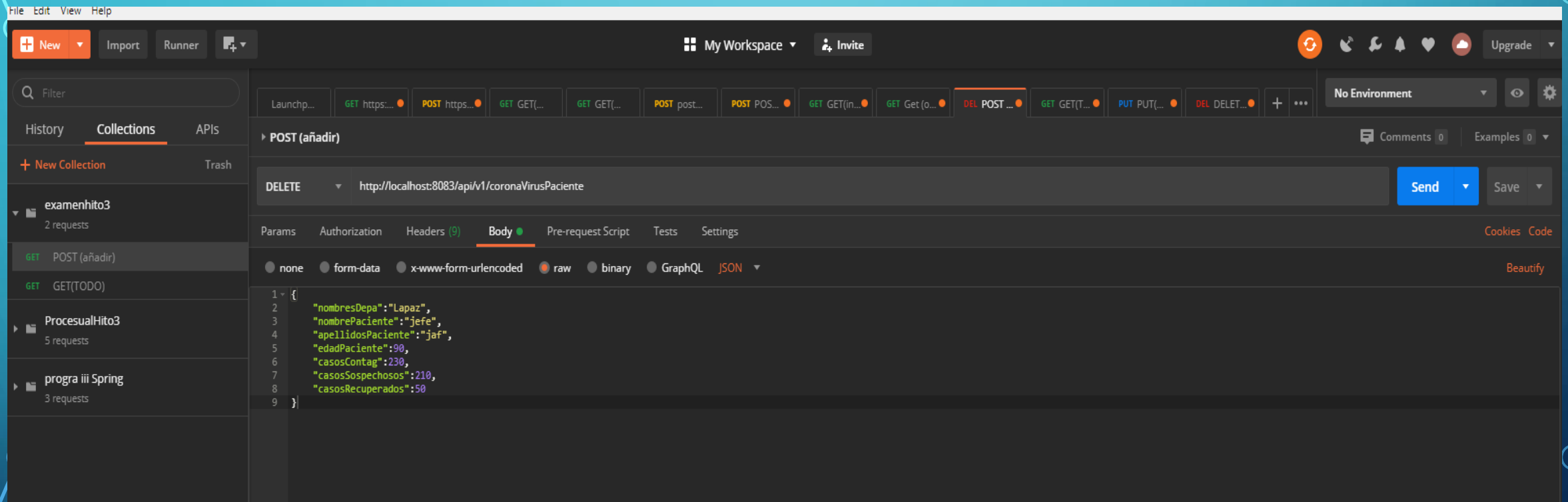
Name	Description
<b>body</b> <small>* required</small> object (body)	Parametros necesarios para la creacion. Example Value   Model <pre>{   "nombreDepartamento": "Cochabamba",   "nombrePaciente": "Marteja Tyy",   "apellidosPaciente": "Marteja Tyy",   "edad": 22,   "casosContagiados": 56,   "casosSospechosos": 120,   "casosRecuperados": 7 }</pre>

Parameter content type:

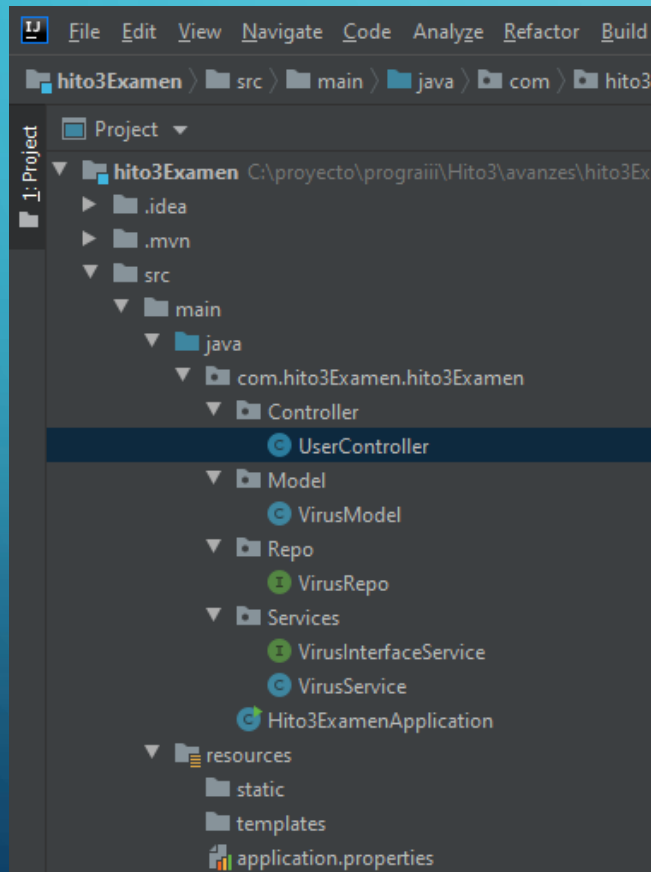
**Responses**

Response content type:

# Primero se crear el servicio en la aplicación postman



# CREAMOS LOS PACKAGE CONTROLLER Y LA CLASE USERCONTROLLER



Aquí creamos los servicios correspondientes



creamos el código en la clase virus Controller el Servicio Post para ingresar los datos y guardarlos

```
}  
@PostMapping("/coronaVirusPaciente")  
  
public ResponseEntity save(@RequestBody VirusModel VIRUS) {  
  
    try {  
        return new ResponseEntity<>(virusService.save(VIRUS), HttpStatus.EXPECTATION_FAILED);  
    } catch (Exception e) {  
        return new ResponseEntity<>( headers: null, HttpStatus.EXPECTATION_FAILED);  
    }  
}
```

Se utiliza el servicio save creado en la interface

Si todo esta correcto ingresa los datos en las tablas

Si no logra guardar saldrá un error



se crea el package Service como también la interface VirusInterfaceVirus y la clase Virus Service

- VirusInterfaceVirus

Se crean el save, update, delete, getall y el getbyideper para obtención y guardado de datos

```
package com.hito3Examen.hito3Examen.Services;  
  
import com.hito3Examen.hito3Examen.Model.VirusModel;  
  
import java.util.List;  
  
public interface VirusInterfaceService {  
    public VirusModel save(VirusModel virusModel);  
    public VirusModel update(VirusModel virusModel, Integer idDepar);  
    public Integer delete();  
    public List<VirusModel> getAllDepar();  
    public VirusModel getDeparByIdPer(Integer idDep);  
}
```

En el VirusService se maneja todo el guardado de los datos y insertacion de las tablas se aplica igualmente las condiciones de la pregunta de si es niño se le agrega una categoría y se junta sus nombres si es mayor a 20 es adolescente

```
@Service
public class VirusService implements VirusInterfaceService {
    @Autowired
    private VirusRepo virusRepo;
    @Override
    public VirusModel save(VirusModel virusModel){
        if(virusModel.getEdadPaciente()<10)
        {
            virusModel.setCategoria("nino");
        }
        else if (virusModel.getEdadPaciente()<20 && virusModel.getEdadPaciente()>10 )
        {
            virusModel.setCategoria("adolescente");
        }
        else if (virusModel.getEdadPaciente()>70)
        {
            return null;
        }
        virusModel.setFullname(virusModel.getNombrePaciente()+virusModel.getApellidosPaciente());
        return virusRepo.save(virusModel);
    }
}
```

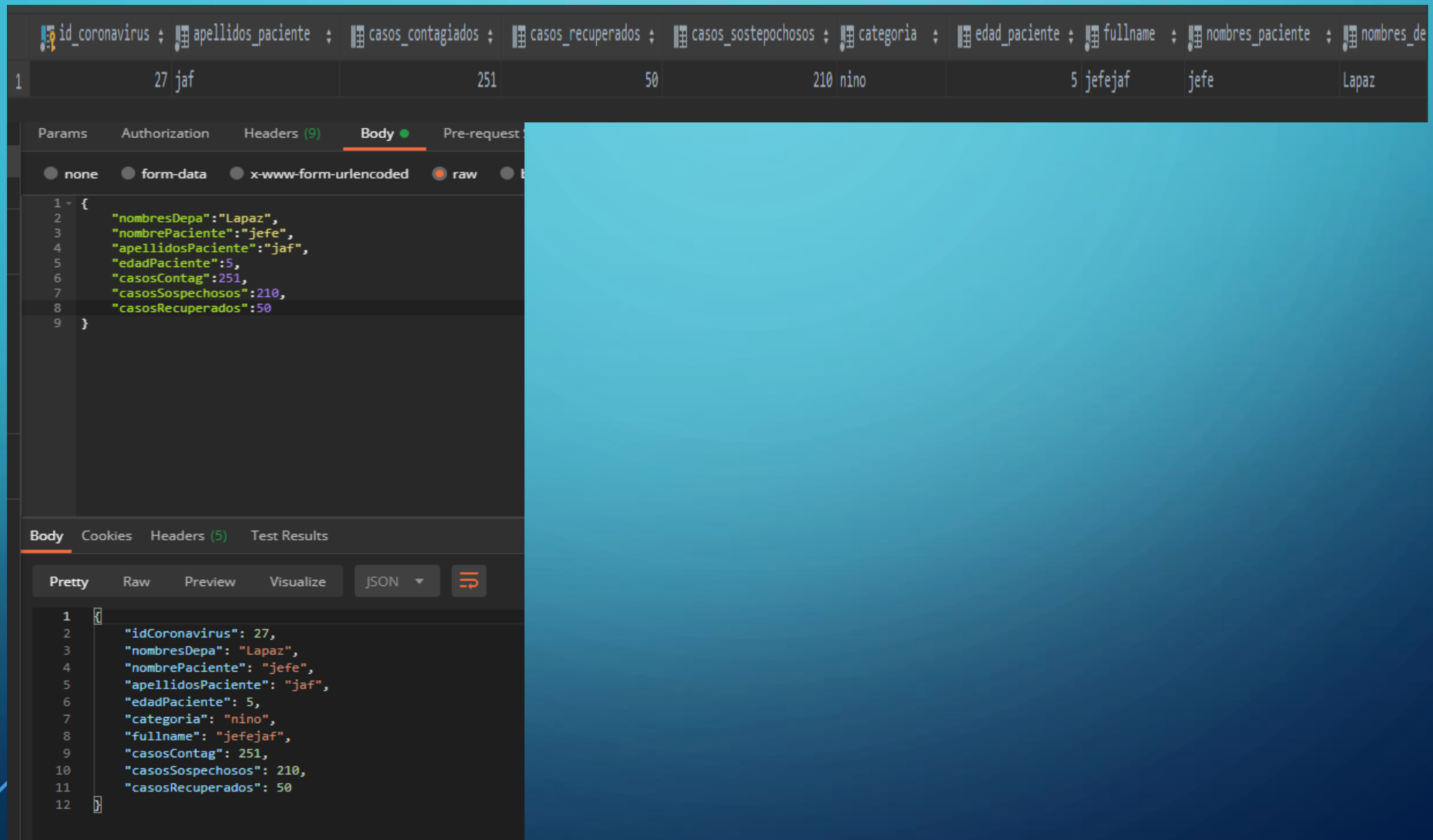
La restricción si es menor a 10 es niño

Restriccion de si es menor a 20 se lo marca en la categoría adolescente

Se crea el guardado en las tablas

Si es mayor a 20 es adulto

# Funcionamiento del post y el guardado del datagrip



The screenshot displays a REST client interface with a table of data at the top and a detailed view of a POST request body below.

	id_coronavirus	apellidos_paciente	casos_contagiados	casos_recuperados	casos_sostepochosos	categoria	edad_paciente	fullname	nombres_paciente	nombres_de
1	27	jaf	251	50	210	nino	5	jefejaf	jefe	Lapaz

The POST request body is shown in the 'Body' tab, set to 'raw' mode. It contains a JSON object with the following fields:

```
1 {  
2   "nombresDepa": "Lapaz",  
3   "nombrePaciente": "jefe",  
4   "apellidosPaciente": "jaf",  
5   "edadPaciente": 5,  
6   "casosContag": 251,  
7   "casosSospechosos": 210,  
8   "casosRecuperados": 50  
9 }
```

Below the raw JSON, the 'Pretty' view shows the same data formatted for readability:

```
1 {  
2   "idCoronavirus": 27,  
3   "nombresDepa": "Lapaz",  
4   "nombrePaciente": "jefe",  
5   "apellidosPaciente": "jaf",  
6   "edadPaciente": 5,  
7   "categoria": "nino",  
8   "fullname": "jefejaf",  
9   "casosContag": 251,  
10  "casosSospechosos": 210,  
11  "casosRecuperados": 50  
12 }
```

# PREGUNTA 3

- Crear los servicios para poder listar todos los pacientes u en su caso uno solo  
REST – GET



primero creamos el getall para obtener todos los datos ingresados y después el getid para obtener según su id del paciente

- GETALL

```
@GetMapping("/coronaVirusPaciente")  
  
public ResponseEntity<List<VirusModel>> getAllDepart() {  
    try {  
        List<VirusModel> depar = virusService.getAllDeepar();  
  
        if (depar.isEmpty()) {  
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);  
        } else {  
            return new ResponseEntity<>(depar, HttpStatus.OK);  
        }  
    } catch (Exception e) {  
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);  
    }  
}
```



Utilizamos el servicio  
GETALL para  
obtener todos los  
datos

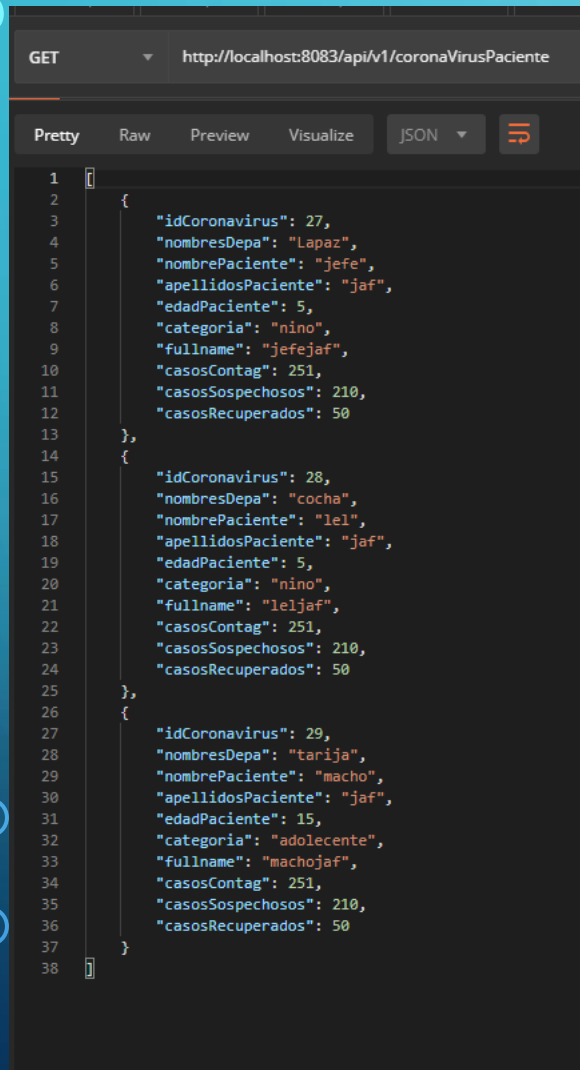
# creamos el GETALL y el GETID

```
}  
@Override  
public List<VirusModel> getAllDepar(){  
    List<VirusModel> depart = new ArrayList<>();  
    virusRepo.findAll().forEach(depart::add);  
    return depart;  
}  
@Override  
public VirusModel getDeparByIdPer(Integer idDEP){  
    Optional<VirusModel> depart = virusRepo.findById(idDEP);  
    VirusModel virusModel = null;  
    if(depart.isPresent()){  
        virusModel = depart.get();  
    }  
    return virusModel;  
}
```

Para obtener todos los datos  
ingresados

Para obtener los datos  
mediante su id

# FUNCIONAMIENTO DEL POSTMAN GETALL PARA OBTENER TODO LOS DATOS



The screenshot shows the Postman interface with a GET request to `http://localhost:8083/api/v1/coronaVirusPaciente`. The response is displayed in the 'Pretty' tab as a JSON array containing three patient records. The interface includes tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize', along with a 'JSON' dropdown and a 'Copy' icon.

```
1  [
2    {
3      "idCoronavirus": 27,
4      "nombresDepa": "Lapaz",
5      "nombrePaciente": "jefe",
6      "apellidosPaciente": "jaf",
7      "edadPaciente": 5,
8      "categoria": "nino",
9      "fullname": "jefejaf",
10     "casosContag": 251,
11     "casosSospechosos": 210,
12     "casosRecuperados": 50
13   },
14   {
15     "idCoronavirus": 28,
16     "nombresDepa": "cocha",
17     "nombrePaciente": "lel",
18     "apellidosPaciente": "jaf",
19     "edadPaciente": 5,
20     "categoria": "nino",
21     "fullname": "leljaf",
22     "casosContag": 251,
23     "casosSospechosos": 210,
24     "casosRecuperados": 50
25   },
26   {
27     "idCoronavirus": 29,
28     "nombresDepa": "tarija",
29     "nombrePaciente": "macho",
30     "apellidosPaciente": "jaf",
31     "edadPaciente": 15,
32     "categoria": "adolescente",
33     "fullname": "machojaf",
34     "casosContag": 251,
35     "casosSospechosos": 210,
36     "casosRecuperados": 50
37   }
38 ]
```



# FUNCIONAMIENTO DEL GETID PARA OBTENER SOLO 1

The screenshot displays the Postman interface with the 'Collections' tab selected. On the left sidebar, there are two collections: 'examenhito3' (containing 2 requests) and 'ProcesualHito3' (containing 5 requests). The 'examenhito3' collection is expanded, showing a 'GET POST (añadir)' request selected. The main panel shows the details of a 'GET' request to the URL 'http://localhost:8083/api/v1/coronaVirusPaciente/28'. The response is displayed in 'Pretty' JSON format, showing a single patient record.

```
GET http://localhost:8083/api/v1/coronaVirusPaciente/28

Pretty Raw Preview Visualize JSON

1 {
2   "idCoronavirus": 28,
3   "nombresDepa": "cocha",
4   "nombrePaciente": "lel",
5   "apellidosPaciente": "jaf",
6   "edadPaciente": 5,
7   "categoria": "nino",
8   "fullname": "leljaf",
9   "casosContag": 251,
10  "casosSospechosos": 210,
11  "casosRecuperados": 50
12 }
```

## Pregunta 4

- Crear un servicio
- REST –PUT que permita modificar un registro CVP

Crear un servicio REST - PUT que permita modificar un registro CVP.

**PUT** /coronaVirusPaciente /{idCoronaVirus} Updates a specific CV row

Parameters [Try it out](#)


Name	Description
<b>body</b> * required object (body)	Parametros necesarios para la modificacion  Example Value   Model <pre>{   "nombreDepartamento": "Cochabamba",   "nombrePaciente": "Martezk Tyy",   "apellidosPaciente": "Martezk Tyy",   "edad": 22,   "casosContagiados": 56,   "casosSospechosos": 120,   "casosRecuperados": 7 }</pre> Parameter content type application/json
<b>idCoronaVirus</b> * required Integer(\$int64) (path)	ID of Corona Virus  idCoronaVirus - ID of Corona Virus

Responses   
Response content type application/json

Code	Description
------	-------------

primero creamos el PUT para la actualización de datos mediante el idpaciente

```
}  
@PutMapping("/coronaVirusPaciente/{idCoronavirus}")  
public ResponseEntity<VirusModel> update(@PathVariable("idCoronavirus") Integer idDep, @RequestBody VirusModel DepModel) {  
    try {  
        VirusModel DUpdate = virusService.update(DepModel, idDep);  
        if (DUpdate != null) {  
            return new ResponseEntity<>(DUpdate, HttpStatus.OK);  
        } else {  
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);  
        }  
    } catch (Exception e) {  
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);  
    }  
}
```



Se utiliza el servicio update para poder actualizar los datos cambiados

# Creamos el servicio update

```
@Override
public VirusModel update(VirusModel virusModel,Integer idDep){
    Optional<VirusModel> virus = virusRepo.findById(idDep);
    VirusModel virusUpdate = null;
    if(virus.isPresent()){
        virusUpdate = virus.get();
        virusUpdate.setNombrePaciente(virusModel.getNombrePaciente());
        virusUpdate.setNombresDepa(virusModel.getNombresDepa());
        virusUpdate.setCasosContag(virusModel.getCasosContag());
        virusUpdate.setCasosRecuperados(virusModel.getCasosRecuperados());
        virusUpdate.setCasosSospechosos(virusModel.getCasosSospechosos());
        virusUpdate.setEdadPaciente(virusModel.getEdadPaciente());
        virusUpdate.setApellidosPaciente(virusModel.getApellidosPaciente());
        if(virusModel.getEdadPaciente()<10)
        {
            virusUpdate.setCategoria("nino");
        }
        else if (virusModel.getEdadPaciente()<20 && virusModel.getEdadPaciente()>10 )
        {
            virusUpdate.setCategoria("adolescente");
        }
        else{
            virusUpdate.setCategoria("adulto");
        }
        virusUpdate.setFullname(virusModel.getNombrePaciente()+virusModel.getApellidosPaciente());
    }
    return virusUpdate;
}
@Override
```

Pedimos los datos del Virus Model  
para la búsqueda por ID

Actualizamos con los nuevos datos  
ingresados

Utilizamos las condicionales de igual  
manera para el guardado si se cambia  
la edad del paciente

# FUNCIONAMIENTO DEL UPDATE

The screenshot displays a REST client interface with a PUT request to `http://localhost:8083/api/v1/coronaVirusPaciente/28`. The 'Body' tab is selected, showing a JSON payload. Below the request, the 'Test Results' section shows the response body in 'Pretty' format, which is a JSON object containing patient details and case statistics.

**Request:**

```
PUT http://localhost:8083/api/v1/coronaVirusPaciente/28
```

**Request Body (JSON):**

```
1 {
2   "nombresDepa": "tarija",
3   "nombrePaciente": "loco",
4   "apellidosPaciente": "lel",
5   "edadPaciente": 15,
6   "casosContag": 251,
7   "casosSospechosos": 210,
8   "casosRecuperados": 50
9 }
```

**Response Body (JSON):**

```
1 {
2   "idCoronavirus": 28,
3   "nombresDepa": "tarija",
4   "nombrePaciente": "loco",
5   "apellidosPaciente": "lel",
6   "edadPaciente": 15,
7   "categoria": "adolescente",
8   "fullname": "locolel",
9   "casosContag": 251,
10  "casosSospechosos": 210,
11  "casosRecuperados": 50
12 }
```

# PREGUNTA 5

- Evitar insertar en la base de datos nuevo casos CVP si la edad de el paciente es mayor a 70

```
public class VirusService implements VirusInterfaceService {  
    @Autowired  
    private VirusRepo virusRepo;  
    @Override  
    public VirusModel save(VirusModel virusModel){  
        if(virusModel.getEdadPaciente()<10)  
        {  
            virusModel.setCategoria("nino");  
        }  
        else if (virusModel.getEdadPaciente()<20 && virusModel.getEdadPaciente()>10 )  
        {  
            virusModel.setCategoria("adolescente");  
        }  
        else if (virusModel.getEdadPaciente()>70)  
        {  
            return null;  
        }  
        virusModel.setFullname(virusModel.getNombrePaciente()+virusModel.getApellidosPaciente());  
        return virusRepo.save(virusModel);  
    }  
}
```

En el save determinamos si la edad del paciente es mayor a 70 no guardara

Si se logra guardar el programa le pondrá la categoría correspondiente

# FUNCIONAMIENTO DE LAS CONDICIONALES

The screenshot displays a REST client interface with a POST request to `http://localhost:8083/api/v1/coronaVirusPaciente/28`. The request body is a JSON object with the following fields:

```
1 {  
2   "nombresDepa": "tarija",  
3   "nombrePaciente": "loco",  
4   "apellidosPaciente": "lel",  
5   "edadPaciente": 80,  
6   "casosContag": 251,  
7   "casosSospechosos": 210,  
8   "casosRecuperados": 50  
9 }
```

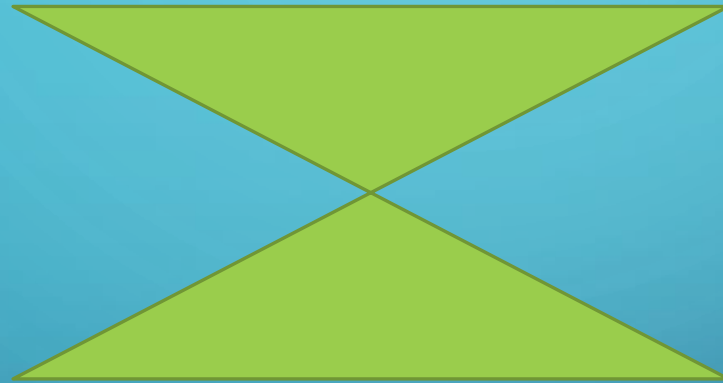
The response status is `405 Method Not Allowed`. The response body is a JSON object with the following fields:

```
1 {  
2   "timestamp": "2020-05-18T02:00:19.894+0000",  
3   "status": 405,  
4   "error": "Method Not Allowed",  
5   "message": "Request method 'POST' not supported",  
6   "path": "/api/v1/coronaVirusPaciente/28"  
7 }
```



## PREGUNTA 6

- Crear un servicio REST – DELETE que elimina todos los registros de la base de datos



# Primero creamos el delete para la conexcion del postman y utilizamos el delete de Services

```
@DeleteMapping("/coronaVirusPaciente")
public ResponseEntity<String> delete() {
    try {
        virusService.delete();
        return new ResponseEntity<>( body: "person successfully deleted", HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}
```

Utilizamos el servicio delete

# CREAMOS EL SERVICIO DELETE

```
@Override  
public Integer delete(){  
    virusRepo.deleteAll();  
    return 1;  
}  
@Override
```

Elimina todo el contenido de las tablas

# FUNCIONAMIENTO DELETE

The screenshot displays a REST client interface with a DELETE request to `http://localhost:8083/api/v1/coronaVirusPaciente`. The request body is a JSON object containing patient data. The response status is 200 OK, and the body contains the message "1 person successfully deleted".

**Request Details:**

- Method: DELETE
- URL: `http://localhost:8083/api/v1/coronaVirusPaciente`
- Body Type: raw (JSON)
- Body Content:

```
1 {  
2   "nombresDepa": "tarija",  
3   "nombrePaciente": "loco",  
4   "apellidosPaciente": "lel",  
5   "edadPaciente": 80,  
6   "casosContag": 251,  
7   "casosSospechosos": 210,  
8   "casosRecuperados": 50  
9 }
```

**Response Details:**

- Status: 200 OK
- Time: 850 ms
- Size: 191 B
- Body Content:

```
1 person successfully deleted
```