

UNIVERSIDAD PRIVADA FRANZ TAMAYO

HITO 3



Nombre Completo: Unifranz. Diego Emiliano Rivera Tapia

Asignatura: PROGRAMACIÓN III

Carrera: INGENIERÍA DE SISTEMAS

Paralelo: PROG (1)

Docente: Lic. William R. Barra Paredes

Fecha: 11/05/2020

github:

<https://github.com/DiegoRiveratapia/prograiii/tree/master/Hito%203/Procesual>

Parte Teórica.

1. Preguntas.

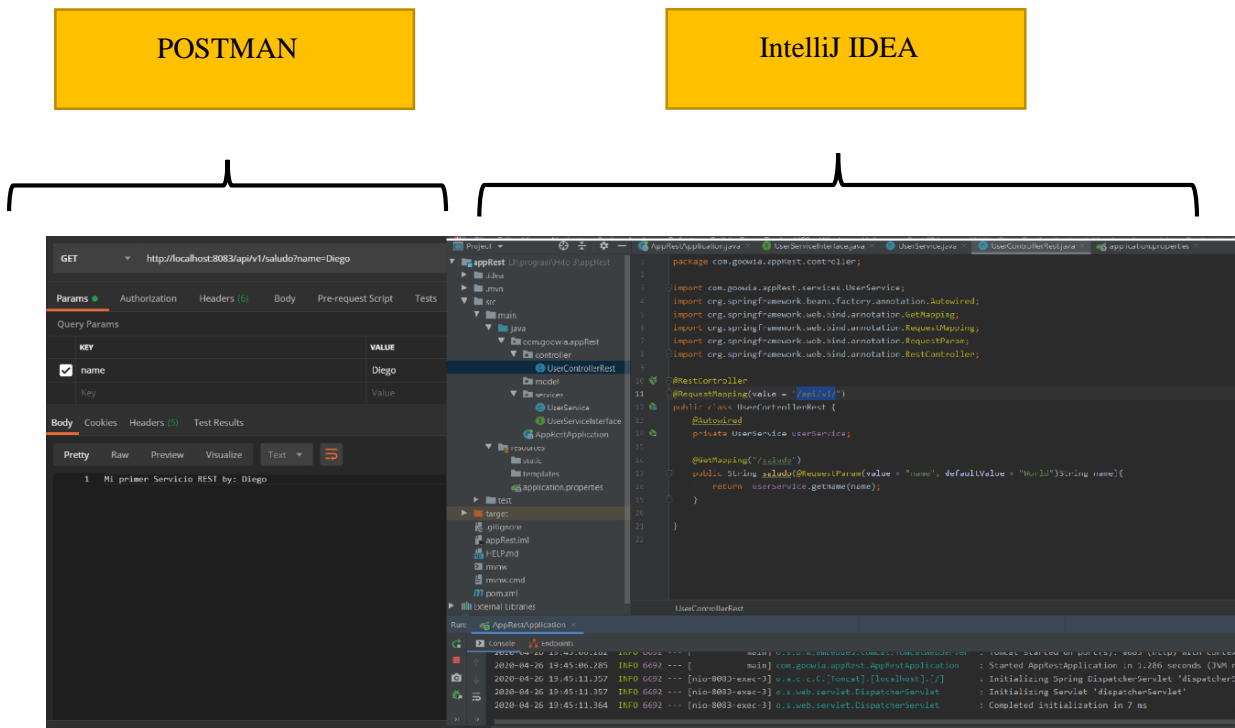
Responda de manera breve, clara y concisa posible.

- **Defina y muestre ejemplo de un servicio REST.**

REST es un estilo de arquitectura software para sistemas hipertexto distribuidos como la World Wide Web.

Es una interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

Ejemplo del uso de REST mediante Postman y IntelliJ Idea.



- **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).

Configuración de un entorno Spring:

Se ingresa a la página oficial de Spring y al apartado de Spring Initializr, Para comenzar a configurar.

Se seleccionan las dependencias.

Se seleccionan las características.

Se seleccionan los datos.

Se genera el entorno.

- **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.

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

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.

- **Qué son los Spring estereotipos y anotaciones muestre ejemplos.**

Un estereotipo de Spring se usa para definir Beans de Spring dentro de un contexto de Spring, existen varios estereotipos los más usados son 5:

@Controller, @Service, @Component, @RestController y @Repository.

@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**.

@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.

@Autowired: Esta anotación se aplica a campos, métodos de “setters” y constructores. La anotación @Autowired inyecta la dependencia del objeto implícitamente.

Ejemplos:

```
import java.util.Date;
import java.util.List;

@Service
public class PersonaService implements PersonaServiceInterface {

    @Autowired
    private PersonaRepo personaRepo;

    @Override
    public PersonaModel save(PersonaModel pModel) {
        return personaRepo.save(pModel);
    }

    @Override
    public PersonaModel update(PersonaModel pModel, Integer idPer) {
        Optional<PersonaModel> person = personaRepo.findById(idPer);
        PersonaModel personaUpdate = null;

        if (person.isPresent()) {
            personaUpdate = person.get();
            personaUpdate.setNombres(pModel.getNombres());
            personaUpdate.setApellidos(pModel.getApellidos());
            personaUpdate.setEmail(pModel.getEmail());
        }
    }
}
```

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

    @Autowired
    private UserService userService;

    @Autowired
    private PersonaService personaService;

    private final String NAME_APP = "GooWia Solutions";

    @GetMapping("/nameApp")
    public String nameApp() { return NAME_APP; }

    @GetMapping("/saludo")
    public String saludo(@RequestParam(value = "name", defaultValue = "World")String name){
        return userService.getName(name);
    }

    UserControllerRest > delete()
```

- **Describe las características principales de REST.**

Sus principales características se dividen en seis:

1. **Escalabilidad:** es un término usado en tecnología para referirse a la propiedad de aumentar la capacidad de trabajo o de tamaño de un sistema sin comprometer el funcionamiento y calidad normales del mismo.
La variedad de sistemas y de clientes crece continuamente. Gracias al protocolo HTTP, pueden interactuar con cualquier servidor HTTP sin ninguna configuración especial.
2. **Independencia:** Los clientes y servidores pueden tener puestas en funcionamiento complejas. HTTP permite una conexión mediante las URLs.
3. **Compatibilidad:** REST como utiliza HTTP sobre Transmission Control Protocol (TCP) en el puerto de red 80 evita el ser bloqueado.
Los servicios web se pueden utilizar sobre cualquier protocolo.
4. **Identificación de recursos:** HTTP es un protocolo centrado en URIs, donde los recursos son los objetos lógicos los cuales se comunican.
5. **Protocolo cliente/servidor sin estado:** Los mensajes al utilizar HTTP contienen toda la información para poder comprender la petición sin dificultad alguna.
HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.
6. **Operaciones bien definidas:** HTTP en sí define un conjunto de algunos verbos, los más importantes son POST, GET, PUT y DELETE.

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.
 - 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.

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

1. 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 trabajó en clases. Inclusive es posible utilizar la misma base de datos ya creada.

2. Spring Framework

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

3. 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
Considere las siguientes especificaciones para cada una de ellas.	

POST
/coronaVirus
Adds the new case CV in the store

Parameters
Try it out

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

Responses
Response content type

application/json

Code	Description
201	Created
471	Expectation Failed

PUT
/coronaVirus/{idCoronaVirus}
Updates a specific CV row

Parameters
Try it out

Name	Description
body * required object (body)	Parametros necesarios para la modificacion <div> Example Value Model </div> <pre>{ "idCoronaVirus": 0, "nombreDep": "Cochabamba", "casosContagiados": 56, "casosSospechosos": 120, "casosRecuperados": 7 }</pre> <div> Parameter content type <div>application/json</div> </div>
idCoronaVirus * required integer(\$int64) (path)	ID of Corona Virus <div> idCoronaVirus - ID of Corona Virus </div>

Responses
Response content type

application/json

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

GET

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

Returns a single CV

Parameters

Try it out

Name	Description
idCoronaVirus * required	ID of Corona Virus
integer(\$int64)	
(path)	

idCoronaVirus - ID of Corona Virus

Responses

Response content type application/json

Code	Description
200	successful operation
404	Pet not found
500	Internal server error

Example Value | Model

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

GET

/coronaVirus/ Gets all cv records

Returns all Corona Virus

Parameters

Try it out

No parameters

Responses

Response content type application/json

Code	Description
200	successful operation
404	Pet not found
500	Internal server error

Example Value | Model

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


DELETE

/coronaVirus/deleteCV/{idCoronaVirus} Deletes a CV

Parameters







Try it out

Name	Description
idCoronaVirus * required	<input type="text" value="idCoronaVirus"/>
integer(\$int64)	
(path)	

Responses

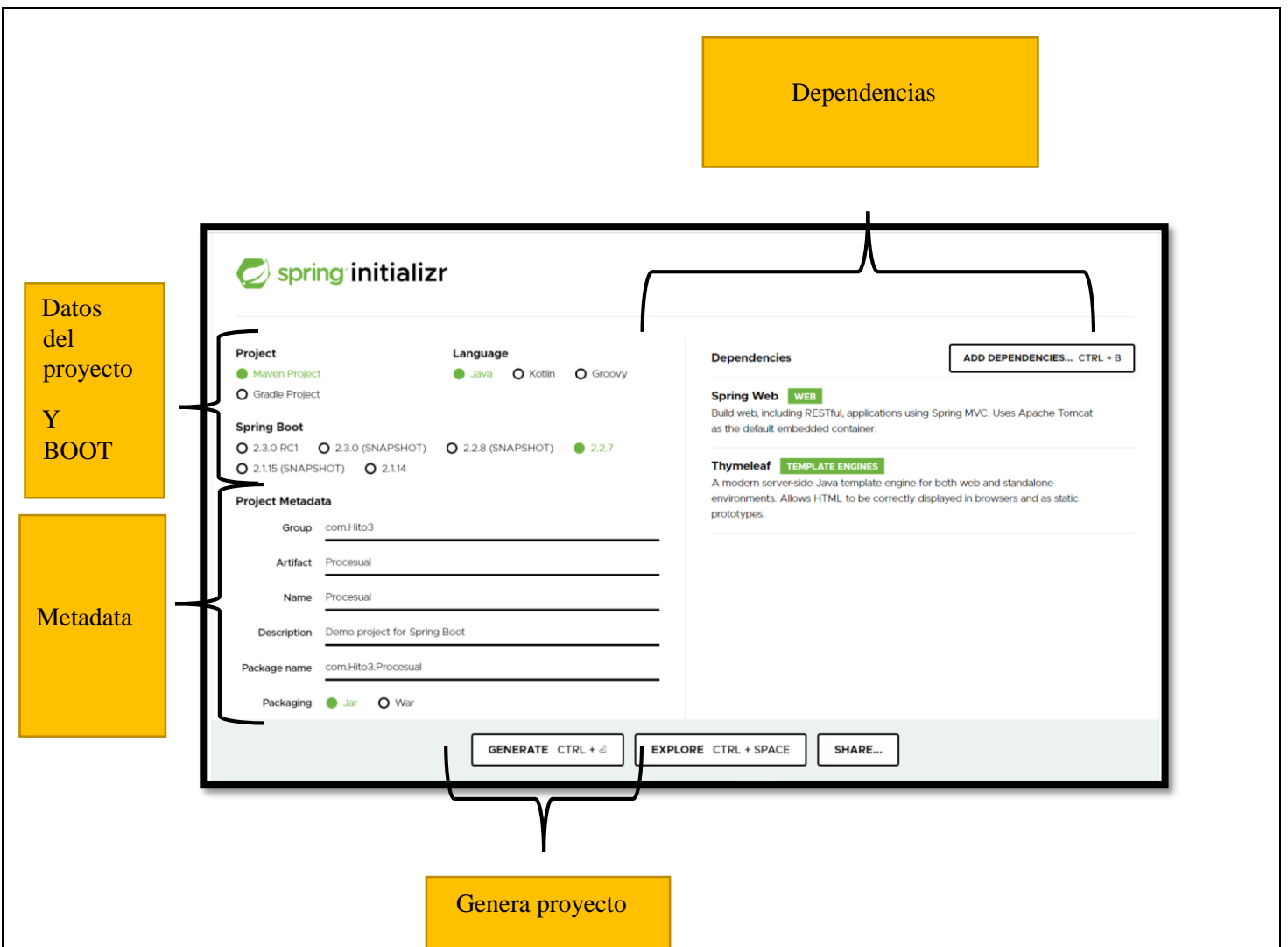
Response content type application/json

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

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

Empezamos creando un proyecto en el framework Spring desde la página principal:
<https://start.spring.io/>

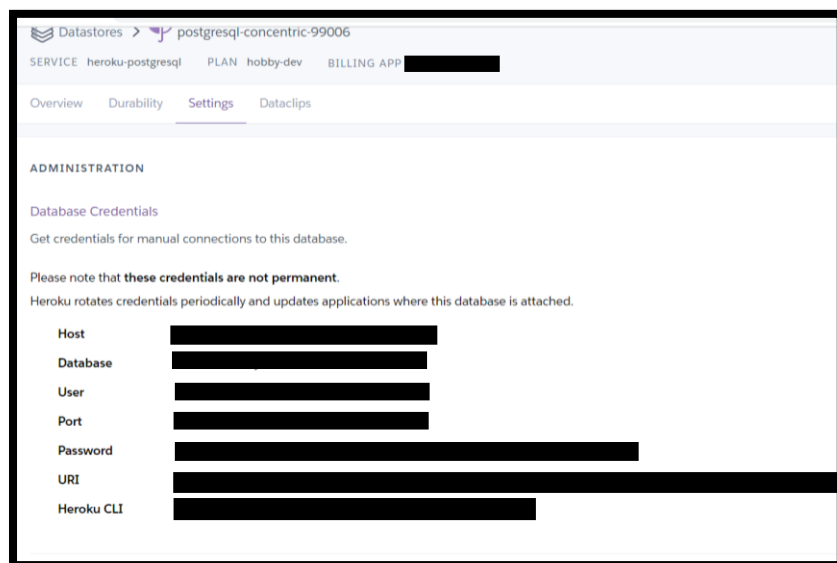
Creamos el proyecto con las dependencias de WEB y THYMELEAF. De definimos el lenguaje, SPRING BOOT, un nombre y grupo.



Destinamos la plataforma HEROKU como servicio de computación en la Nube para crear la conexión con nuestra base de datos. Dentro de HEROKU: <https://dashboard.heroku.com/>,

Procedemos a crear un pipeline y después una app, automáticamente nos creara un Datastore.

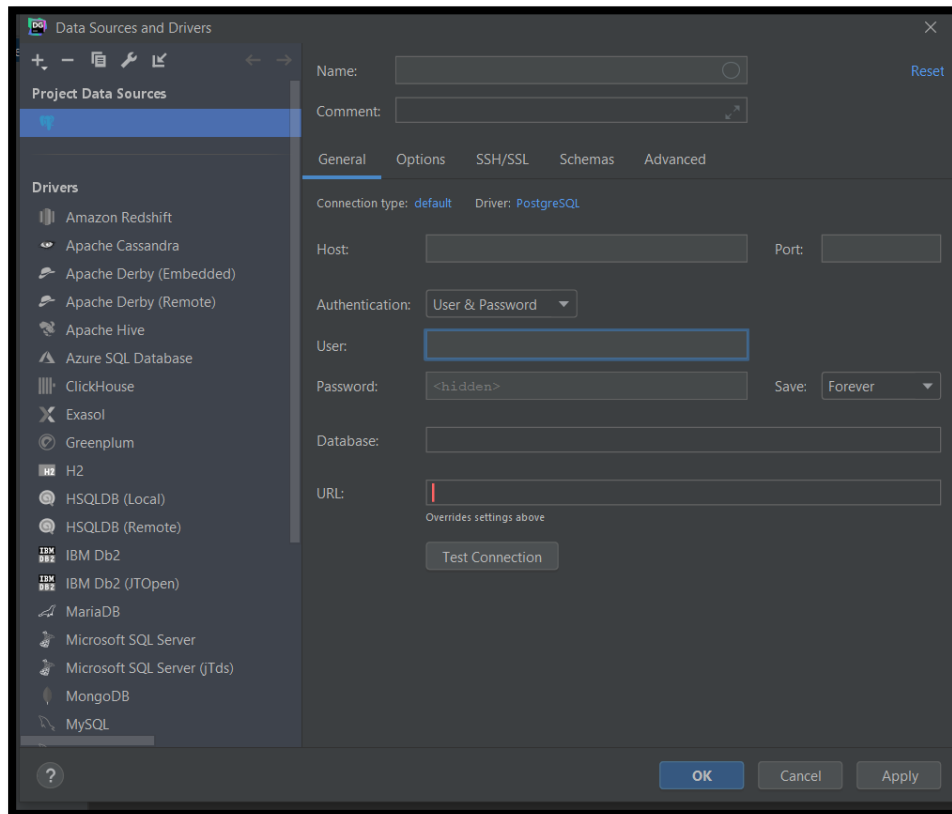
El cual tendrá los siguientes datos para poder crear la conexión.



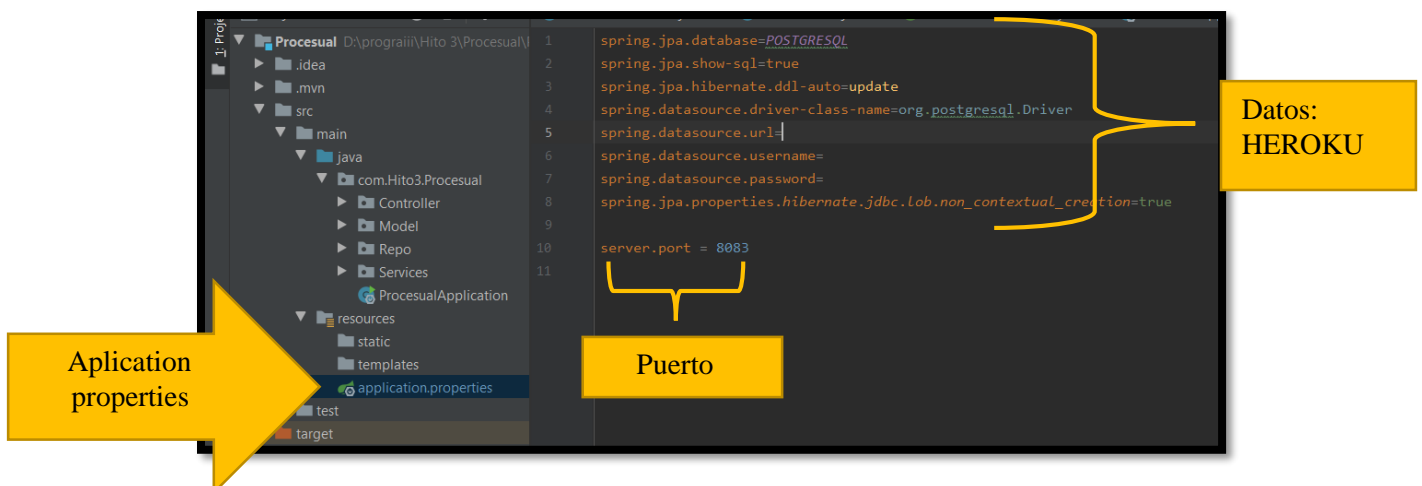
Con los datos obtenidos de HEROKU empezamos a configurar la base de datos.

Se usará jetbrains datagrip.

Una vez creado la data source procedemos a configurar la conexión ingresando los datos de nuestro proyecto en HEROKU en las propiedades de la base de datos.



Preparamos la conexión para el jetbrains intelliij idea de igual forma que la base de datos.



Añadimos las dependencias al proyecto creado con el framework Spring: springframework y postgresql.

```
16
17 <properties>
18   <java.version>1.8</java.version>
19 </properties>
20
21 <dependencies>
22   <dependency>
23     <groupId>org.springframework.boot</groupId>
24     <artifactId>spring-boot-starter-data-jpa</artifactId>
25   </dependency>
26   <dependency>
27     <groupId>org.postgresql</groupId>
28     <artifactId>postgresql</artifactId>
29     <scope>runtime</scope>
30   </dependency>
31   <dependency>
32     <groupId>org.springframework.boot</groupId>
33     <artifactId>spring-boot-starter-thymeleaf</artifactId>
34   </dependency>
35   <dependency>
36     <groupId>org.springframework.boot</groupId>
37     <artifactId>spring-boot-starter-web</artifactId>
38   </dependency>
39
40   <dependency>
41     <groupId>org.springframework.boot</groupId>
42     <artifactId>spring-boot-starter-test</artifactId>
43     <scope>test</scope>
44     <exclusions>
```

Dependencias
necesarias

Creamos un servicio REST:

MODELO

Creamos la tabla donde se almacenaran los datos: **Corona_Virus**

```
package com.Hito3.Procesual.Model;

import javax.persistence.*;
import java.sql.Date;

@Entity
@Table(name = "Corona_Virus")
public class VirusModel {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int idCoronaVirus;
    @Column(name = "nombre_Dep", length = 50, nullable = false)
    private String nombreDep;
    @Column(name = "Casos_Contagiados")
    private Integer casosContagiados;
    @Column(name = "Casos_Sospechosos")
    private Integer CasosSospechosos;
    @Column(name = "Casos_Recuperados")
    private Integer CasosRecuperados;

    public int getIdCoronaVirus() {
        return idCoronaVirus;
    }

    public void setIdCoronaVirus(int idCoronaVirus) {
        this.idCoronaVirus = idCoronaVirus;
    }
}
```

```

public String getNombreDep() {
    return nombreDep;
}

public void setNombreDep(String nombreDep) {
    this.nombreDep = nombreDep;
}

public Integer getCasosContagiados() {
    return casosContagiados;
}

public void setCasosContagiados(Integer casosContagiados) {
    this.casosContagiados = casosContagiados;
}

public Integer getCasosSospechosos() {
    return CasosSospechosos;
}

public void setCasosSospechosos(Integer casosSospechosos) {
    CasosSospechosos = casosSospechosos;
}

public Integer getCasosRecuperados() {
    return CasosRecuperados;
}

public void setCasosRecuperados(Integer casosRecuperados) {
    CasosRecuperados = casosRecuperados;
}
}

```

Creamos la interface REPO:

```

package com.Hito3.Procesual.Repo;

import com.Hito3.Procesual.Model.VirusModel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface VirusRepo extends JpaRepository <VirusModel,Integer> {
}

```

Creamos Services:

Clase:

```

package com.Hito3.Procesual.Services;

import com.Hito3.Procesual.Model.VirusModel;
import com.Hito3.Procesual.Repo.VirusRepo;
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 VirusServices implements VirusServicesInterface {
    @Autowired
    private VirusRepo virusRepo;

    @Override
    public VirusModel save(VirusModel vModel) {
        return virusRepo.save(vModel);
    }

    @Override
    public VirusModel update(VirusModel vModel, Integer idPer) {
        Optional<VirusModel> person = virusRepo.findById(idPer);
        VirusModel vUpdate = null;
        if (person.isPresent()) {
            vUpdate = person.get();
            vUpdate.setNombreDep(vModel.getNombreDep());
            vUpdate.setCasosContagiados(vModel.getCasosContagiados());
            vUpdate.setCasosSospechosos(vModel.getCasosContagiados());
            vUpdate.setCasosRecuperados(vModel.getCasosRecuperados());
            virusRepo.save(vUpdate);
        }
        return vUpdate;
    }

    @Override
    public Integer delete(Integer idPer) {
        virusRepo.deleteById(idPer);
        return 1;
    }

    @Override
    public List<VirusModel> getAllDep() {
        List<VirusModel> virus = new ArrayList<VirusModel>();
        virusRepo.findAll().forEach(virus::add);
        return virus;
    }

    @Override
    public VirusModel getVirusByIdPer(Integer idPer) {
        Optional<VirusModel> virus = virusRepo.findById(idPer);
        VirusModel vModel = null;
        if (virus.isPresent()) {
            vModel = virus.get();
        }
        return vModel;
    }
}

```

Interface

```
package com.Hito3.Procesual.Services;

import com.Hito3.Procesual.Model.VirusModel;

import java.util.List;

public interface VirusServicesInterface {
    public VirusModel save(VirusModel vModel);
    public VirusModel update(VirusModel vModel, Integer idCoronaVirus);
    public Integer delete(Integer idCoronaVirus);
    public List<VirusModel> getAllDep();
    public VirusModel getVirusByIdPer(Integer idCoronaVirus);
}
```

Creamos el Controller

```
package com.Hito3.Procesual.Controller;

import java.util.List;
import com.Hito3.Procesual.Model.VirusModel;
import com.Hito3.Procesual.Services.VirusServices;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/v2/")
public class UserControllerRest {
    @Autowired
    private VirusServices virusServices;

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

    @PutMapping("/virus/{idCoronaVirus}")
    public ResponseEntity<VirusModel> updateVirus(@PathVariable("idCoronaVirus")
Integer idCoronaVirus, @RequestBody VirusModel vModel) {
        try {
            VirusModel vUpdate = virusServices.update(vModel, idCoronaVirus);
            if (vUpdate != null) {
```

```

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

@GetMapping("/virus")
public ResponseEntity<List<VirusModel>> getAllDep() {
    try {
        List<VirusModel> persons = virusServices.getAllDep();

        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);
    }
}

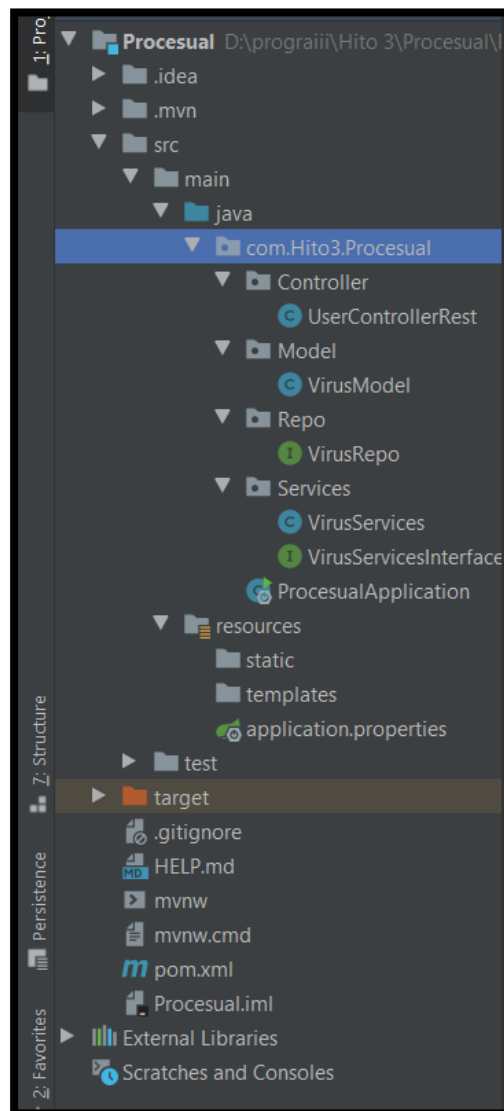
@GetMapping("/virus/{idPer}")
public ResponseEntity<VirusModel> getVirusByIdPer(@PathVariable("idPer") Integer
idPer) {
    try {
        VirusModel vModel = virusServices.getVirusByIdPer(idPer);

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

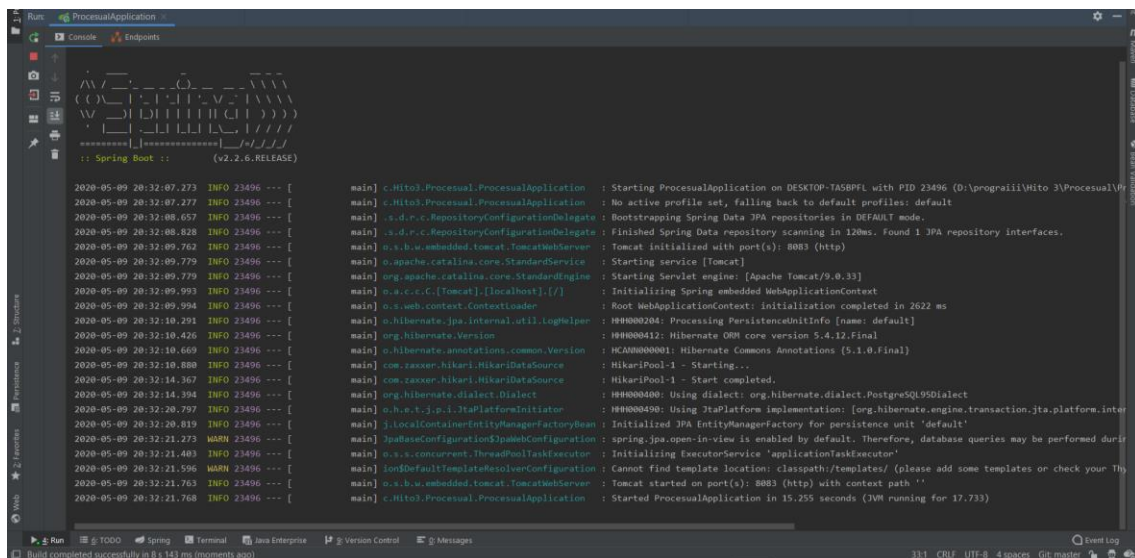
@DeleteMapping("/virus/{idPer}")
public ResponseEntity<String> delete(@PathVariable("idPer") Integer idPer) {
    try {
        virusServices.delete(idPer);
        return new ResponseEntity<>("Virus successfully deleted", HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.EXPECTATION_FAILED);
    }
}
}

```


Imagen de los packages:



Código ejecutado:



```
Run ProcessApplication
Console Endpoints
2020-05-09 20:32:07.273 INFO 23496 --- [main] c.Hito3.Processual.ProcessualApplication : Starting ProcessualApplication on DESKTOP-TASBPFL with PID 23496 (D:\prograiiii\Hito 3\ProcessualVirus\ProcessualApplication.jar)
2020-05-09 20:32:07.277 INFO 23496 --- [main] c.Hito3.Processual.ProcessualApplication : No active profile set, falling back to default profiles: default
2020-05-09 20:32:08.657 INFO 23496 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2020-05-09 20:32:08.828 INFO 23496 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 120ms. Found 1 JPA repository interfaces.
2020-05-09 20:32:09.762 INFO 23496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8083 (http)
2020-05-09 20:32:09.770 INFO 23496 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-05-09 20:32:09.770 INFO 23496 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]
2020-05-09 20:32:09.993 INFO 23496 --- [main] o.s.c.x.c.TomcatEmbeddedWebApplicationContext : Initializing Spring embedded WebApplicationContext
2020-05-09 20:32:10.291 INFO 23496 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2622 ms
2020-05-09 20:32:10.426 INFO 23496 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2020-05-09 20:32:10.669 INFO 23496 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.12.Final
2020-05-09 20:32:10.880 INFO 23496 --- [main] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5.1.0.Final)
2020-05-09 20:32:14.367 INFO 23496 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-05-09 20:32:14.394 INFO 23496 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-05-09 20:32:14.394 INFO 23496 --- [main] org.hibernate.dialect.Dialect : HH0000480: Using dialect: org.hibernate.dialect.PostgreSQL95Dialect
2020-05-09 20:32:20.797 INFO 23496 --- [main] o.h.e.s.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator]
2020-05-09 20:32:21.019 INFO 23496 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-05-09 20:32:21.273 WARN 23496 --- [main] jpase.config.jpa.web.JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2020-05-09 20:32:21.403 INFO 23496 --- [main] org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-05-09 20:32:21.596 WARN 23496 --- [main] org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter : Cannot find template location: classpath:/templates/ (please add some templates or check your Thymeleaf configuration)
2020-05-09 20:32:21.763 INFO 23496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8083 (http) with context path ''
2020-05-09 20:32:21.768 INFO 23496 --- [main] c.Hito3.Processual.ProcessualApplication : Started ProcessualApplication in 15.255 seconds (JVM running for 17.733)
```

Ejecutamos el programa ayudándonos de POSTMAN: Todo código esta ejecutado en JSON

Verbo POST:

Añadiendo datos a la tabla creada

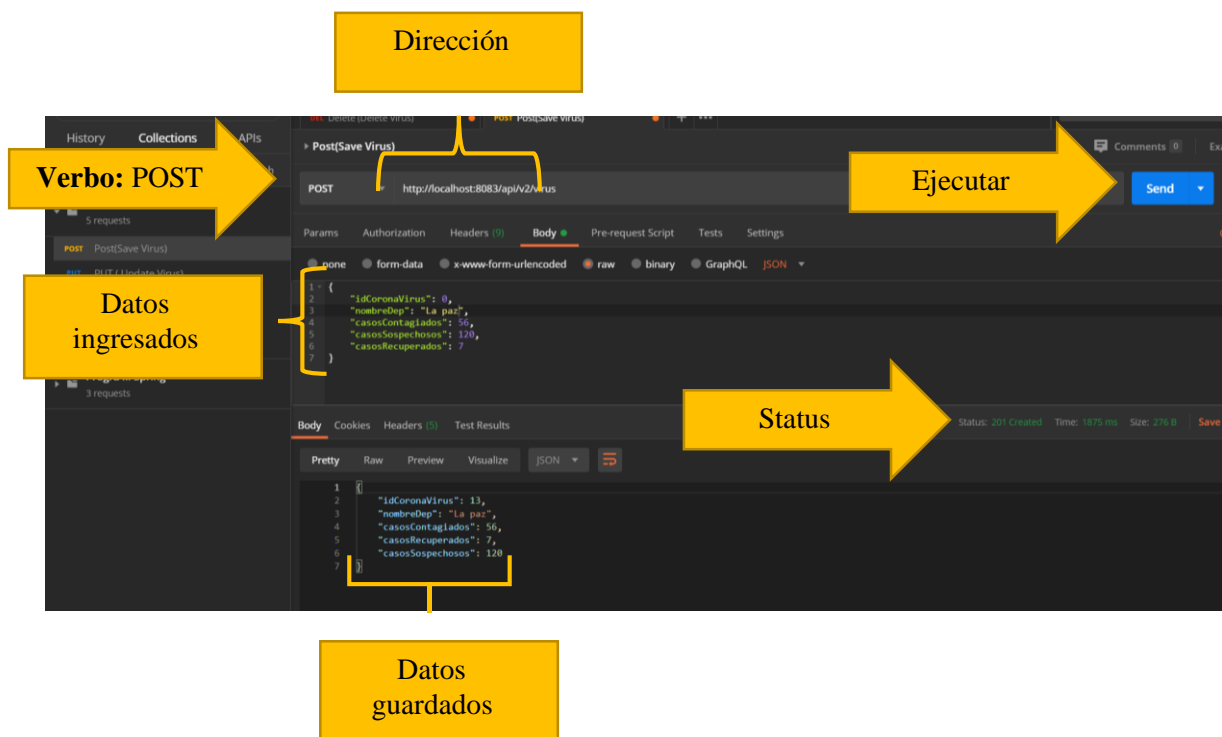


Diagram illustrating the execution of a POST request in Postman:

- Dirección:** The URL is `http://localhost:8083/api/v2/virus`.
- Verbo: POST:** The HTTP method is set to POST.
- Datos ingresados:** The request body is a JSON object:

```
{  "IdCoronaVirus": 0,  "nombreDep": "La paz",  "casosContagiados": 56,  "casosSospechosos": 120,  "casosRecuperados": 7}
```
- Ejecutar:** The request is sent.
- Status:** The response status is `201 Created`.
- Datos guardados:** The response body shows the saved data:

```
{  "IdCoronaVirus": 13,  "nombreDep": "La paz",  "casosContagiados": 56,  "casosRecuperados": 7,  "casosSospechosos": 120}
```

Verbo PUT:

Actualizamos o remplazamos los datos ingresados de la tabla.

Verbo: PUT

Id a ejecutar

Ejecutar

Datos a Actualizar

Status

Datos Actualizados

```
PUT http://localhost:8083/api/v2/virus/6

{
  "nombreDep": "La paz",
  "casosContagiados": 10,
  "casosSospechosos": 50,
  "casosRecuperados": 3
}
```

```
{
  "idCoronaVirus": 6,
  "nombreDep": "La paz",
  "casosContagiados": 10,
  "casosRecuperados": 3,
  "casosSospechosos": 10
}
```

Verbo GET: Por Id

Buscará y mostrará el id solicitado

Verbo: GET

Id a buscar

Ejecutar

Status

Datos encontrados

```
GET http://localhost:8083/api/v2/virus/6
```

```
{
  "idCoronaVirus": 6,
  "nombreDep": "La paz",
  "casosContagiados": 10,
  "casosRecuperados": 3,
  "casosSospechosos": 10
}
```

Verbo GET:

Buscará y mostrará todos los datos ingresados

Verbo: GET

Ejecutar

Status

Datos encontrados

```
1 {
2   {
3     "idCoronaVirus": 13,
4     "nombreDep": "La paz",
5     "casosContagiados": 56,
6     "casosRecuperados": 7,
7     "casosSospechosos": 120
8   },
9   {
10    "idCoronaVirus": 6,
11    "nombreDep": "La paz",
12    "casosContagiados": 10,
13    "casosRecuperados": 3,
14    "casosSospechosos": 10
15  }
16 }
```

Verbo DELETE:

Se encargará de eliminar permanentemente los datos del id buscado.

Verbo: DELETE

Id a eliminar

Ejecutar

Status

Mensaje de éxito

```
1 Virus successfully deleted
```

Tabla creada:






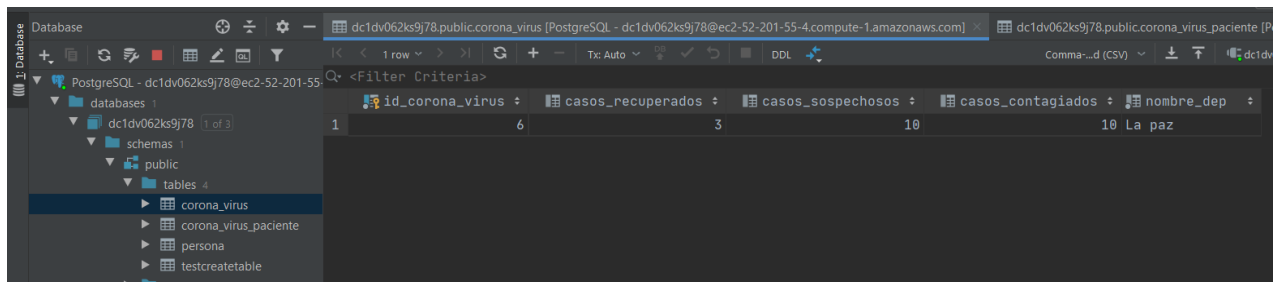
corona_virus	
 id_corona_virus	integer
 casos_recuperados	integer
 casos_sospechosos	integer
 casos_contagiados	integer
 nombre_dep	varchar(50)

Tabla en la base de datos:



	id_corona_virus	casos_recuperados	casos_sospechosos	casos_contagiados	nombre_dep
1	6	3	10	10	La paz

DDL

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

alter table corona_virus
owner to gudeharmoejqgo;
```