8-5-2022

Prueba Técnica

ALEXIS BAEZ

Contenido

Problema

Diseñar el Back-end de un sistema que nos permita registrar a los empleados y que los empleados puedan modificar su respectiva información, adicionalmente saber si el empleado tiene sus vacunas, dosis y tipo de vacuna.

Herramientas

Software usado:

- IDE IntelliJ (Java) versión 11.0.13
- Java versión 11.0.13 2021-10-19 LTS
- Base de datos PostgreSQL versión 6.1
- Power Designer versión 16.1
- Google Chrome

Solución

Base de datos

Se planteo un desarrollo estructurado y desagregado de la base de datos al más mínimo nivel, este se lo desarrollo en Power Designer.

El modelo lo podemos observar en la siguiente figura.

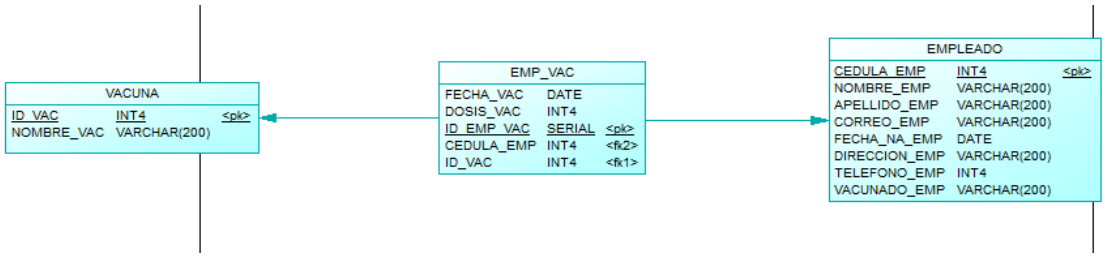

Figura 1.
Modelo base de datos

Una vez ya obtenido un modelo óptimo para la solución se procedió a crear nuestra base de datos en PostgreSQL.

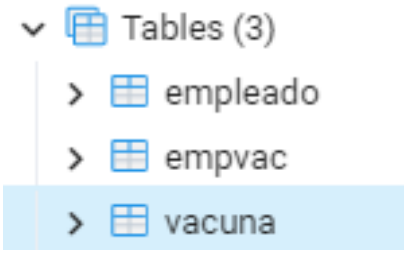

Figura 2.
Base de datos

Ya con nuestra base de datos generada y realizada las pruebas a papel para verificar su correcto funcionamiento se procedió a desarrollar la aplicación web.

Servidor

Para este desarrollo hemos generado servicios Rest en nuestro servidor mediante el framework **Quarkus** con anotaciones **CDI,** una implementación de la base de datos en **Hikari** y documentación por medio de OpenApi**.**

Todas las dependencias necesarias serán ubicadas en el archivo **build.gradle**.

```
implementation  enforcedPlatform("io.quarkus.platform:quarkus-bom:${project.quarkusVersion}")
implementation group:'io.quarkus', name:'quarkus-resteasy'
implementation group:'io.quarkus', name:'quarkus-arc'
implementation group:'io.quarkus', name:'quarkus-smallrye-health'

//json para quarkus
implementation("io.quarkus:quarkus-resteasy-jackson")

//driver postgresql
// https://mvnrepository.com/artifact/org.postgresql/postgresql
implementation group: 'org.postgresql', name: 'postgresql', version: '42.3.1'

//hikari
// https://mvnrepository.com/artifact/com.zaxxer/HikariCP
implementation group: 'com.zaxxer', name: 'HikariCP', version: '5.0.1'

//http cliente
implementation group: 'org.apache.httpcomponents', name: 'httpclient', version: '4.5.13'

//OpenApi
// https://mvnrepository.com/artifact/io.quarkus/quarkus-smallrye-openapi
implementation group: 'io.quarkus', name: 'quarkus-smallrye-openapi', version: '2.7.1.Final'
```

Figura 3.
Dependencias

La estructura de nuestro desarrollo esta de la siguiente manera.

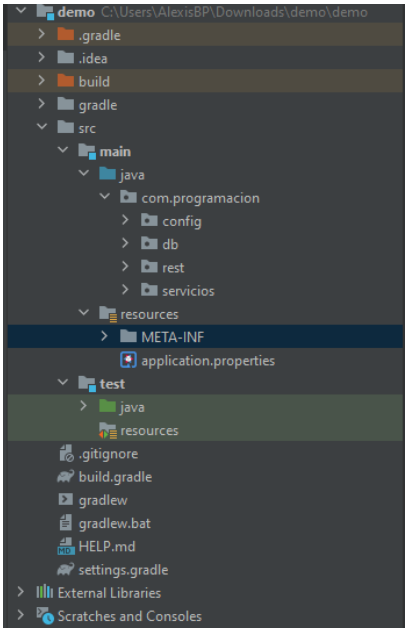

Figura 4.
Estructura.

En nuestra carpeta **config** encontraremos el pool de conexión a la base de datos por medio de **Hikari** y esta clase le designamos como un productor para que su inyección sea más rápida.

```java
@ApplicationScoped
public class DatabaseConfig {

    //pool de conexion
    @Produces
    @ApplicationScoped
    public DataSource dataSource(){

        HikariDataSource ds = new HikariDataSource();
        ds.setDriverClassName("org.postgresql.Driver");
        ds.setJdbcUrl("jdbc:postgresql://localhost:5432/krugerBD");
        ds.setUsername("postgres");
        ds.setPassword("admin");

        return ds;
    }
}
```

Figura 5.
DatabaseConfig

En nuestra carpeta db encontraremos las clases que hacen referencia a las tablas de las bases de datos y además una carpeta de los servicios que serán expuestos.

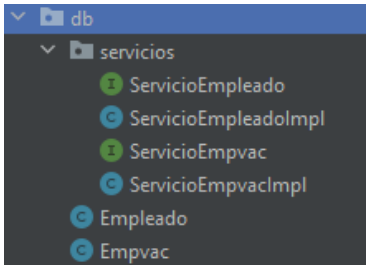

Figura 6.
Clases de las tablas.

Cada una de estas cuanta con los atributos que tenemos en la base de datos con sus respectivos **Getters** y **Setters** para la obtención de datos y la inserción de estos.

```java
public class Empleado {

    //atributos de la tabla empleado en la base de datos
    @Setter @Getter private Integer cedula;
    @Setter @Getter private String nombre;
    @Setter @Getter private String apellido;
    @Setter @Getter private String correo;
    @Setter @Getter private Date fechana;
    @Setter @Getter private String direccion;
    @Setter @Getter private Integer telefono;
    @Setter @Getter private String vacunado;

}
```

Figura 7.
Clase Empleado.

En nuestros servicios encontraremos todo lo referente a los **querys** y consultas que deseamos hacer en nuestra base de datos por medio del IDE en pocas palabras el (**CRUD**).

```java
@ApplicationScoped
public class ServicioEmpleadoImpl implements ServicioEmpleado{

    @Inject
    private DataSource dataSource;

    public Empleado findById  (Integer id){
        Connection con = null;
        Empleado p = null;
        try {
            con = dataSource.getConnection();
            PreparedStatement pstmt = con.prepareStatement( sql: "select * from EMPLEADO where CEDULA = ?");
            pstmt.setInt( parameterIndex: 1, id);
            ResultSet rs = pstmt.executeQuery();
            if(rs.next()){
                p = new  Empleado();
                p.setCedula(id);
                p.setNombre(rs.getString( columnLabel: "NOMBRE"));
                p.setApellido(rs.getString( columnLabel: "APELLIDO"));
                p.setCorreo(rs.getString( columnLabel: "CORREO"));
                p.setFechana(rs.getDate( columnLabel: "FECHANA"));
                p.setDireccion(rs.getString( columnLabel: "DIRECCION"));
                p.setTelefono(rs.getInt( columnLabel: "TELEFONO"));
                p.setVacunado(rs.getString( columnLabel: "VACUNADO"));
```

Figura 8.
Servicio de la clase Empleado FindById.

Y en nuestra carpeta **Rest** estarán ubicados los servicios **Rest** colocados en los diferentes **Path** y con su respectiva **función**, ya sea un **GET** para la obtención de datos o un **POST** para un **Insert**.

```java
@Path("/empleados")
@ApplicationScoped
public class RestEmpleado {
    @Inject
    ServicioEmpleadoImpl servicio;

    @GET
    @Path("/{id}") @Produces(MediaType.APPLICATION_JSON)
    @APIResponse(
            responseCode = "200",
            description = "empleado encontrada exitosamente",
            content = @Content(mediaType = MediaType.APPLICATION_JSON)
    )
    public Empleado findById (@PathParam("id") Integer id){
        Empleado p = servicio.findById(id);
        if (p==null){
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return p;
    }
```

Figura 9.
Rest de la tabla Empleado.

Como se observó a simple vista todo el desarrollo es con la misma lógica y lleva la misma secuencia para cada una de las tablas.

Con esto ya tenemos expuestos nuestros servicios Rest y nos queda el consumo de los mismo mediante un cliente web.

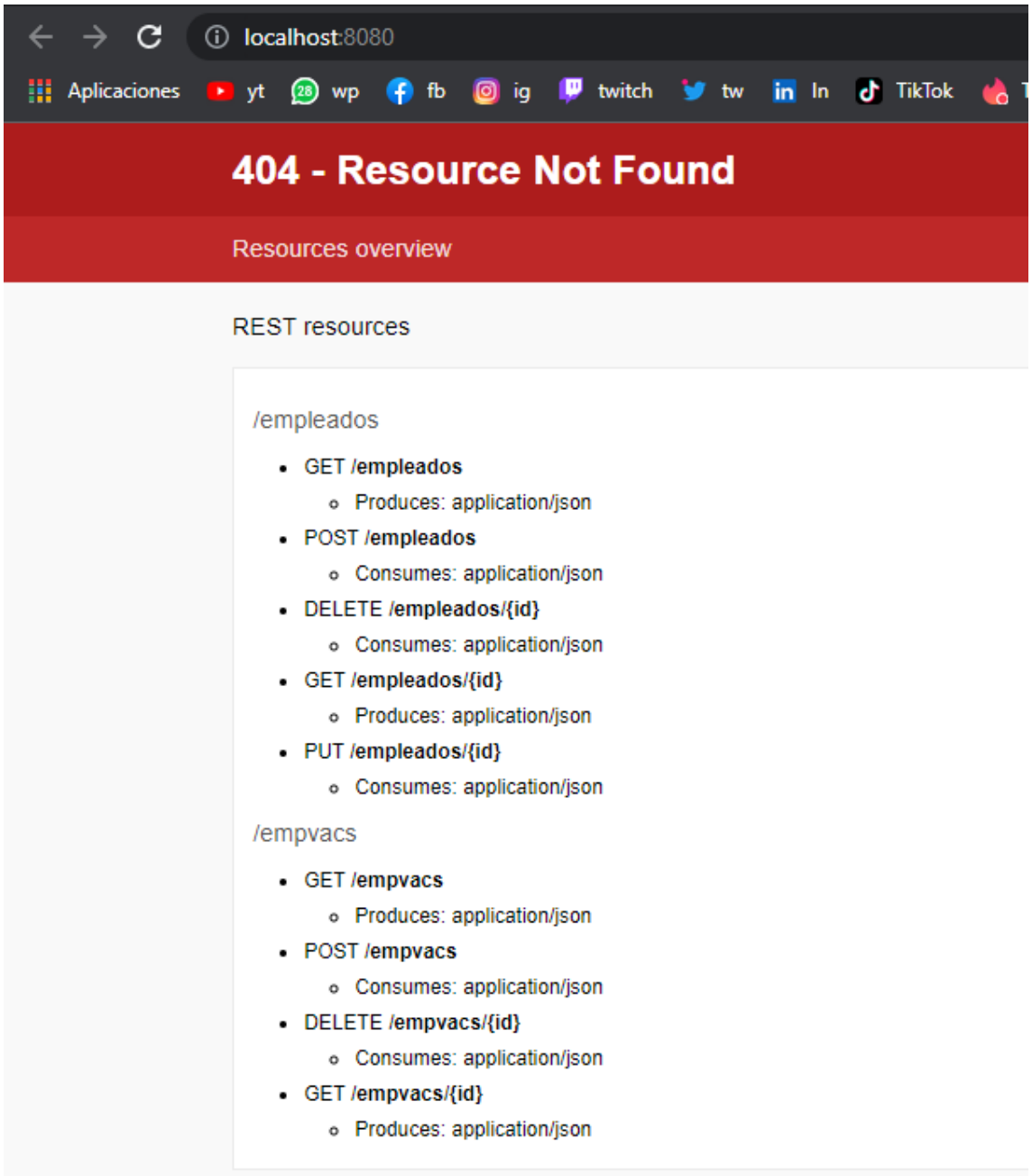

Figura 10.
Exposición de Servicios.

Ejecución.

Nos dirigimos a la siguiente ubicación para levantar las instancias del servidor.

\demo\build\quarkus-app

En esta ubicación abrimos un cmd y ejecutamos el siguiente código.

java -jar quarkus-run.jar

```
C:\Users\AlexisBP\OneDrive - Universidad Central del Ecuador\Escritorio\demo\demo\build\quarkus-app>java -jar quarkus-ru
n.jar
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2022-05-08 18:25:37,881 INFO  [io.quarkus] (main) demo 1.0-SNAPSHOT on JVM (powered by Quarkus 2.7.1.Final) started in 0
.901s. Listening on: http://0.0.0.0:9080
2022-05-08 18:25:37,886 INFO  [io.quarkus] (main) Profile prod activated.
2022-05-08 18:25:37,887 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy, resteasy-jackson, smallrye-context
-propagation, smallrye-health, smallrye-openapi, vertx]
2022-05-08 18:25:46,566 INFO  [com.zax.hik.HikariDataSource] (executor-thread-0) HikariPool-1 - Starting...
2022-05-08 18:25:46,709 INFO  [com.zax.hik.poo.HikariPool] (executor-thread-0) HikariPool-1 - Added connection org.postg
resql.jdbc.PgConnection@7c6a5e35
2022-05-08 18:25:46,710 INFO  [com.zax.hik.HikariDataSource] (executor-thread-0) HikariPool-1 - Start completed.
```

Figura 11.
Ejecución.

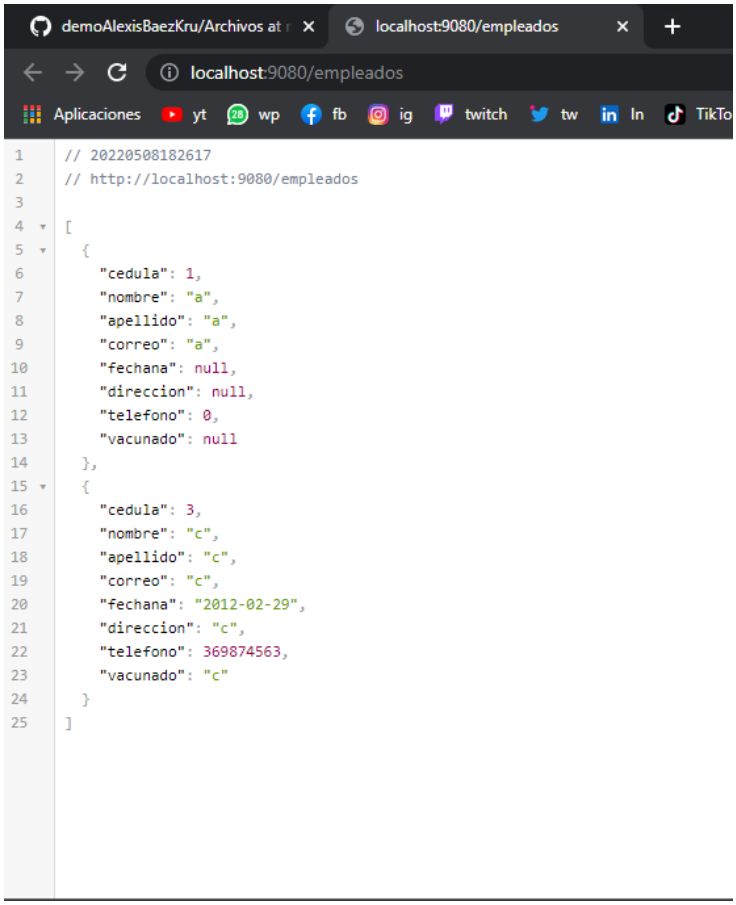

Figura 12.
Exposición de los datos.

Direcciones.

Servidor:

http://localhost:9080/empleados

http://localhost:9080/empvacs